Examining the Efficacy of Virtual Reality for Learning Data Structures in Computer Science

Honors Project

By Matt Waterman

Spring 2023

ABSTRACT

As virtual reality (VR) continues to advance and become more user-friendly, it seems to have potential as a medium for education. Some research surrounding educational VR content exists, but none that delivers the same content across multiple modalities. The purpose of this study was to evaluate whether VR itself offers benefits to students. To do so, three learning modules were created using Unity, and then published both for the desktop and for the VR headset. These modules related to concepts in Data Structures and Algorithms, and had an embedded assessment. The participants were split into two groups, one doing the first two modules in VR, the other doing the first two on a computer. Both groups completed the final module in the other modality. Participants were not aware that only the results of the first two modules were being examined, nor that other participants got a different modality first. Participant results were evaluated from both the embedded assessment and the post survey. The post survey includes Likert scale evaluations of the usability, informativeness, and enjoyability of both modalities, as well as a section for open feedback. The results seem to show that virtual reality does not offer inherent benefits to learning outcomes, but does offer benefits to student focus and engagement. It is believed that with further study, and deeper student engagement, benefits may arise.

INTRODUCTION AND LITERATURE REVIEW

Virtual Reality seems to offer benefits to student interest, engagement, and information retention. To establish context for a possible study, and to understand what has already been done, it is necessary to review existing work in this area. Existing literature investigating Virtual Reality and its benefits for education is limited. There is a minimal number of studies that have a comparative structure between Virtual Reality and another medium.

One of the most relevant pieces of work in this area is on exploring search algorithms, and their visualizations in Virtual Reality as compared to desktop (Grivokostopoulou, 2016). By building a Virtual Reality environment to model and simulate the functionality of algorithms, the authors of this paper wanted to help students learn key algorithms to study computer science. Looking at the results, we see that not only were the test scores significantly higher than that of the control group, students saw value in the use of VR. Across the board, students reported that VR was useful in their enjoyment, engagement, and understanding. To the prompt: “The interaction with the visualizations of the algorithms assisted me in understanding the algorithm's way of function.” - 90.9% of students reported a 5/5 of the Likert scale - indicating they strongly agree with this sentiment. It seems like students consistently

self-report that their engagement, enjoyment, and understanding are all better with Virtual Reality.

It additionally seems that algorithm visualization is useful as an education tool, whether it's in virtual reality or otherwise. "Our data show that learning improves as the level of student engagement with Algorithm Visualization increases" (Grissom, 2003). Across different algorithms - including bubble sort, selection sort, and insertion sort, students had better results as they increased their engagement with visualization content. This conclusion is meaningful because the two groups had similar academic abilities, and were composed of all computer science majors. Assuming that the two groups were of equal abilities, this study points strongly to the idea that algorithm visualization increases student learning. If students learn better with Visualization even on a desktop, it's worth investigating whether VR makes a difference.

Over time, it seems that VR could make a big difference. At a Chinese university, they integrated Virtual Reality modules to an 'Introduction to Computing' course, which is for first-year undergraduate students (Li, 2015). Through this course, students interacted with a number of virtual reality simulations of common computer science concepts. The simulations included various computer science related topics - turing machines, CPU instruction decoding, and more. Across the board, students saw greater interest, academic success, and satisfaction with the VR simulation tools. Grades increased 12% on average, and in almost every category, over 80% positive results were reported.

To implement this software, it seems that Unity3D and Oculus headsets are the common established options. From a meta-analysis conducted over a number of virtual reality education applications, Unity and Oculus consistently produce results. (Akbulut, 2018). This analysis was not limited to just computer science, as the paper mentions a Chemical Engineering 3D practice simulation platform, a Surgical Education triage simulator, and a Gen-Ed simulation of a traditional classroom environment. All three of these used Unity3D and Oculus hardware to conduct their research. Additionally, it seems that Unity3D is the premier choice for Algorithm Visualization. In a master's thesis from 2019, one individual implements nearly a dozen different sorts, operations on graphs, pathfinding, and binary trees. The paper not only describes the abstract implementation, but provides exact code for each algorithm, and how the algorithms are visualized on screen (Buchbauer, 2019). Clearly, Unity3D, and the easy integration with Oculus provide the best platform for implementing educational content in VR.

In sum, the literature is somewhat limited. Overall, virtual reality seems to provide benefits to student engagement, interest, and learning. However, this might be because of VR, or it may be a general benefit of algorithm visualization. Additionally, little research has been done on what aspects of VR offer benefits to students. There is a lack of direct comparative work between the different mediums.

CONTEXT AND PURPOSE

With the literature providing sufficient context for this study, and the lack of targeted research in this area, we decided to move forward with this study.

In addition to its utility, personally, this study and process was meaningful for me.

Computer Science education is something I've been passionate about for a long time. For a good portion of high school and even into college, I worked as a "coding coach" for Code Wiz, at which I taught kids to code, primarily focused on making video games.

Through my time there, I grew into my role not only as a teacher, but helping them expand as well. This meant I had to make content and learning modules for them, to give to other coaches, so I also have expertise with iterating on educational content.

Though I've since transitioned to another job, I haven't yet lost my expertise in delivering effective teaching materials, nor my passion for teaching computer science.

Through that job, the potential for virtual reality also became clear. Using Unity and the Quest headset, I ran summer camps, teaching kids how to code. I saw the value it had as an engaging environment for the kids, how much they enjoyed it, and the efficacy with which it helped them learn.

As a result, I wanted to investigate whether VR was useful as purely an environment for learning. Could content, if delivered in VR, outperform its traditional counterparts, even without modification? Does VR, on its own, offer benefits for learning outcomes? That is the central question for this study, and one I sought to answer with my honors project.

STUDY DESIGN

To answer whether Virtual Reality as a medium offers benefits, participants were split into two groups. Each group would engage with one modality first in which they would complete two modules. Then, they would complete the final module in the opposite modality. (Figure 1) Each module contains an *embedded assessment.* An embedded assessment is defined as “integration of assessment within teaching and learning processes with emphasis on … establishing what, how much and how well students are learning in relation to the learning goals and expected outcomes.” (Gikandi, 2019) Only the results from the first two modules’ embedded assessments were recorded.

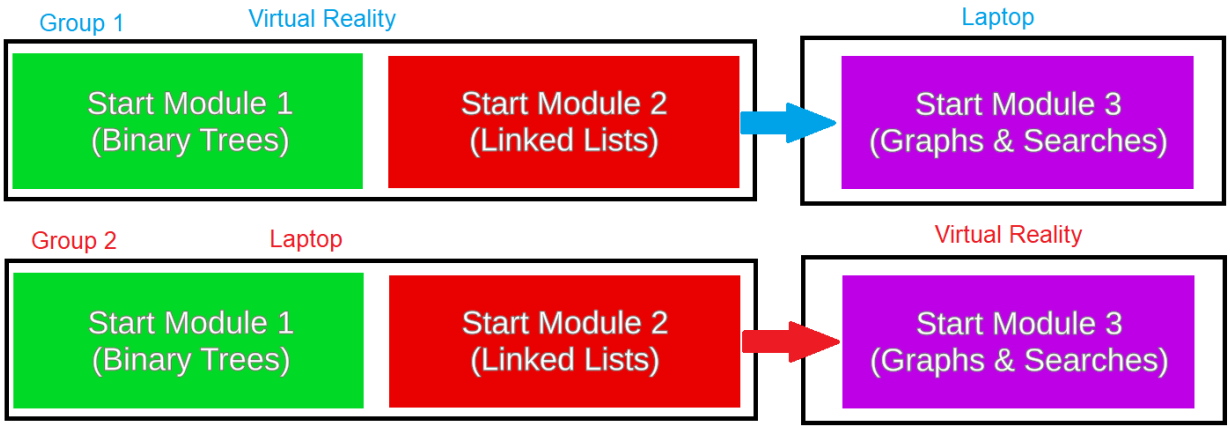
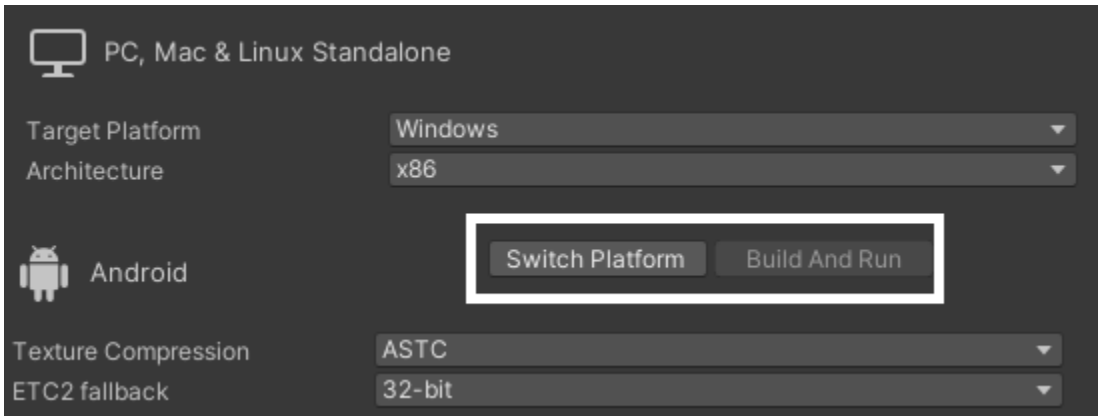


Figure 1 - Dual-modality model for participants

Participants were blind to the fact that some people were getting VR first and others got desktop first. Additionally, with a modality switch, every participant got some experience with VR. The idea of the study is that the content, other than the mode of interaction, is completely identical. Every line of text, every prompt, every quiz question is exactly the same. From an implementation perspective, the Unity engine allows for simple platform switching.



With simply switching the build target, and a simple if-statement on what platform is active, the input package would pick up and bind boiler plate code to the different devices.

```
case RuntimePlatform.WindowsPlayer:
    platform = Platforms.WindowStandalone;
    updateMethod = UpdateKBM;
    ActionsComponent.AssignPlayerReferences(false);
    Algorithms.vr = false;
    ActionsComponent.vr = false;
    break;
```

```
case RuntimePlatform.Android:
    platform = Platforms.Quest;
    updateMethod = UpdateQuest;
    ActionsComponent.AssignPlayerReferences(true);
    Algorithms.vr = true;
    ActionsComponent.vr = true;
    break;
```

The introductory content is also adaptive. (Figure 2)

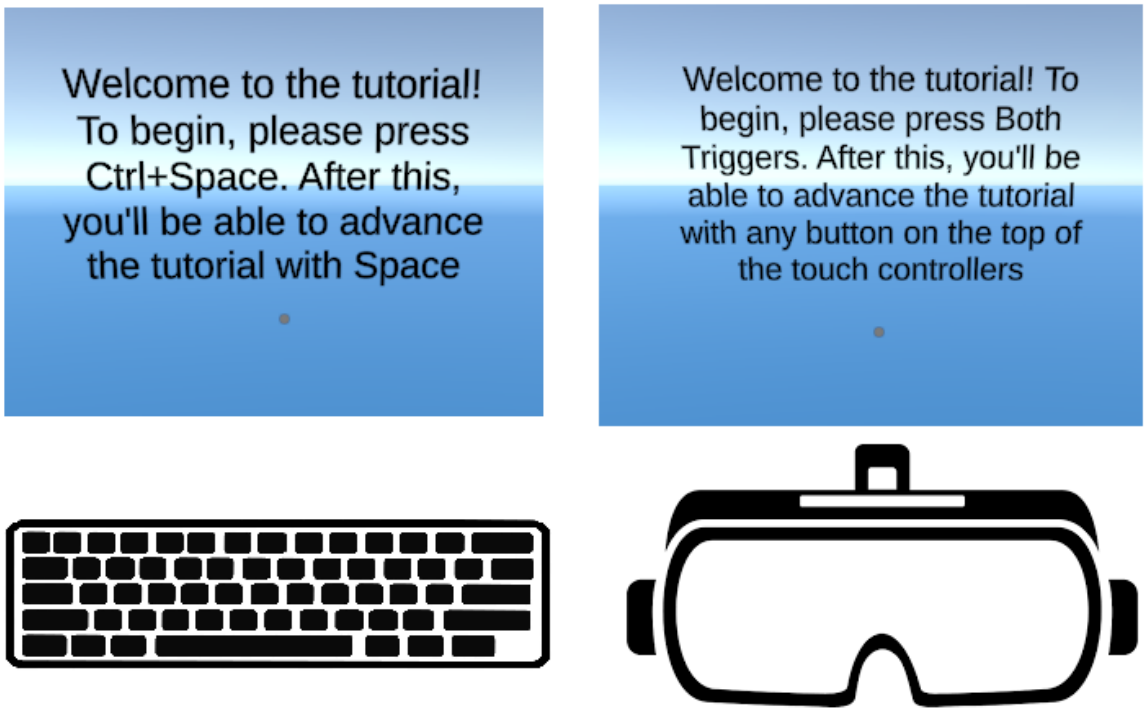


Figure 2 - Adaptive introductory content based on input device

Since the different modalities obviously mandate different input mechanisms, the tutorial must adapt to that as well.

Once a given participant has completed the introduction, and gotten a handle of the controls, they'll move onto the content. They'll complete the first two modules with the given modality, and swap for the other.

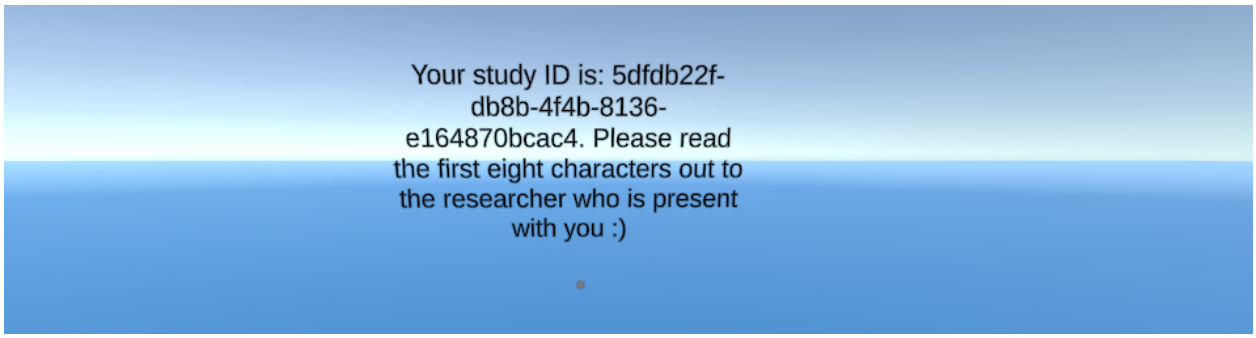
Finally, study participants take the post survey. The design of the post-survey was to collect some demographic info, and then to ask the user to compare their experiences with the two different versions of the software.

UMASS LOWELL

Consent for Research Participation
IRB #: 23-040-MAR-EXM
IRB Approval Date: 4/3/2023

Study title: Examining the Efficacy of Virtual Reality for Learning Data Structures in Computer Science

Before participants complete any of the modules, they're given a study ID, to stand in for collecting any identifying information.



This study ID was used as the identifier for the post-survey. Since no identifying information was collected, no signed consent forms were necessary, and the post-survey did not need any additional work for anonymization. This study was approved by the IRB, and the contents of the IRB submission and approval are appended to the end of this report.[1]

Recruitment materials for this study were extremely effective. In just the first day, there were over forty interested participants, from emailing the Computer Science Mailing List alone. Even without the posters and outreach to the ACM/UMLCS discord servers, we ended up with about sixty interested participants. Though there was only a sufficient budget for twenty, it's worthwhile to note the immediate and outsized interest in the participation for this study.

[1] Thanks again to Emily Sousa for the work she put in to helping amend my submission and expedite the process.The process to get this study approved was, though somewhat arduous, fairly painless, and I have her to thank for that.

UMASS LOWELL
Learning with Purpose

This survey will not collect any identifying information. You should have been given a randomized Globally Unique Identifier (GUID), but this is not related to your name, age, email, etc.

Please enter the first eight characters of your GUID provided by the software. If you don't remember, please ask for help.

In addition to the quantitative data gathered from the embedded assessment, ordinal data was gathered on the subjective experience with the software. (Figure 3) For both modalities, the following questions were asked:

I found the Virtual Reality experience enjoyable.

Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree

I found the desktop experience enjoyable.

Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree

Figure 3 - Post Survey Questions

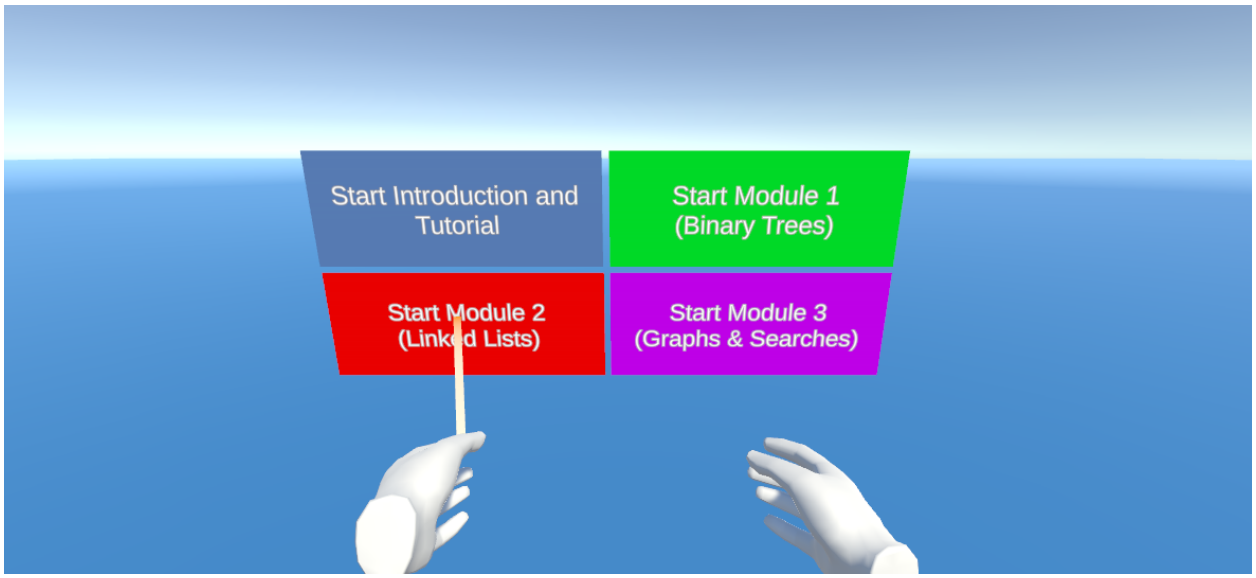
- Did you find it enjoyable?
- Did you find it informative

- Did you find it easy to use?

Finally, there was an open-response section, allowing participants to outline any other thoughts regarding the differences between modalities, ease of use, and their overall experience with the materials.

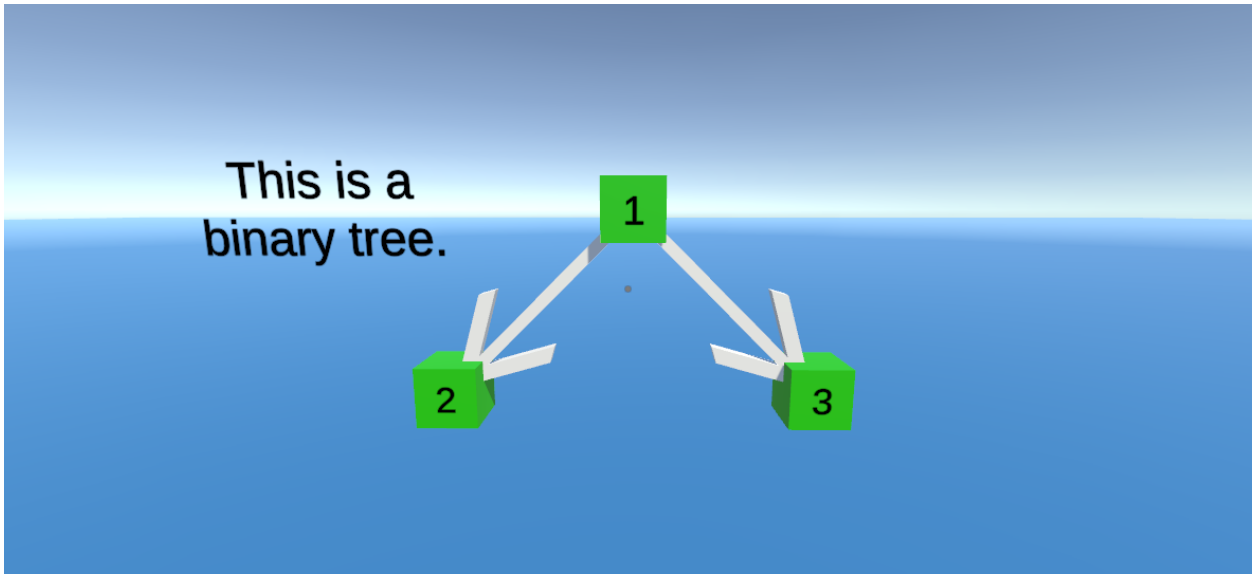
CONTENT DEVELOPMENT

To support the design of this study, the materials were developed in Unity. Unity is a 3D game and application making engine. It is approachable, allows for quick development, and supports building applications for multiple platforms. Due to these factors, in addition to personal experience, it was the clear choice for developing the learning materials.



Immediately, the software should allow users to be engaged. It includes a menu with dynamic animations, a clear title, and menu options that are easily accessible.

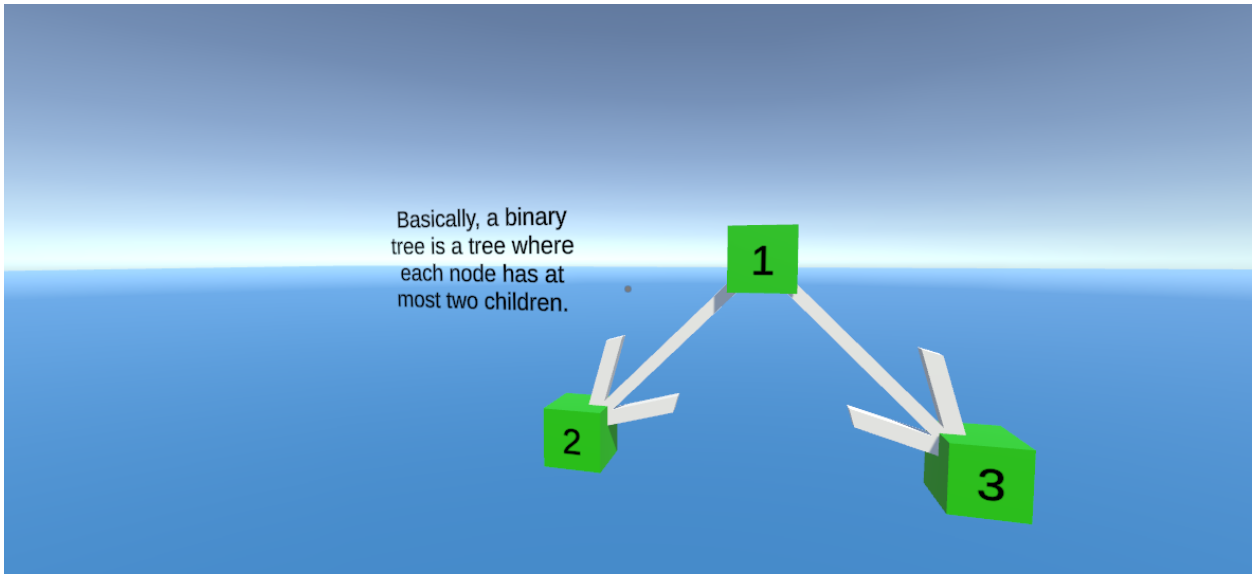
For the modules, the study design included topics that participants might have some familiarity with, but not something they were already likely to be experts in. For the content, the three main content focuses were selected to be Binary Trees, then Linked Lists, then finally Graphs and Searches.



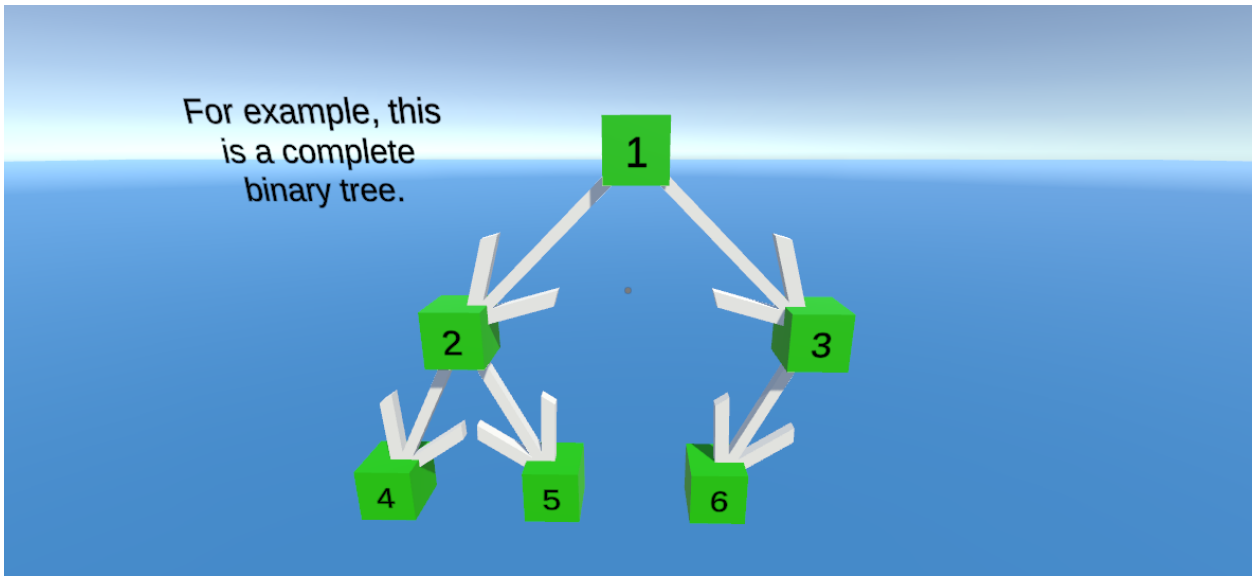
For binary trees, especially since this is the introductory module to the content, it was designed to be simple and intuitive. No complex algorithms, just an introduction to the data structure.

> **Definition 10.4.3. Binary Tree.**
>
> 1. A tree consisting of no vertices (the empty tree) is a binary tree
> 2. A vertex together with two subtrees that are both binary trees is a binary tree. The subtrees are called the left and right subtrees of the binary tree.

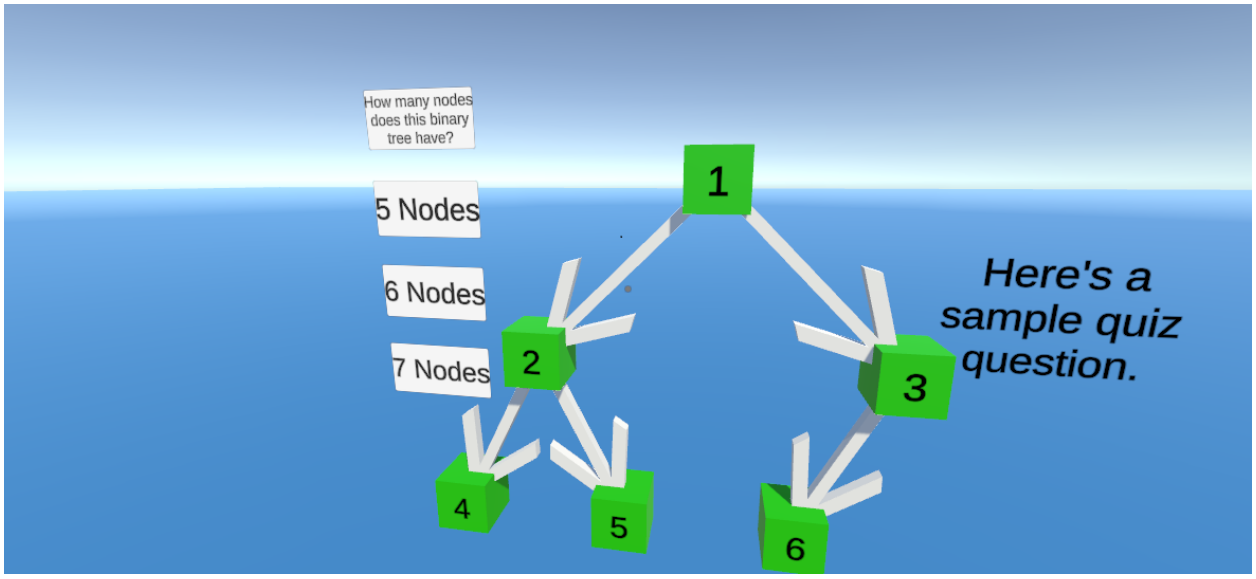
Initially, the content was based on existing class content, or that had been used to teach similar lessons. In the demo, the formal definition of a Binary Tree is provided,sourced from discrete structures. Though some contexts have a clear need for rigor in an academic context, quickly, the content was adapted to be much friendlier to the layman.



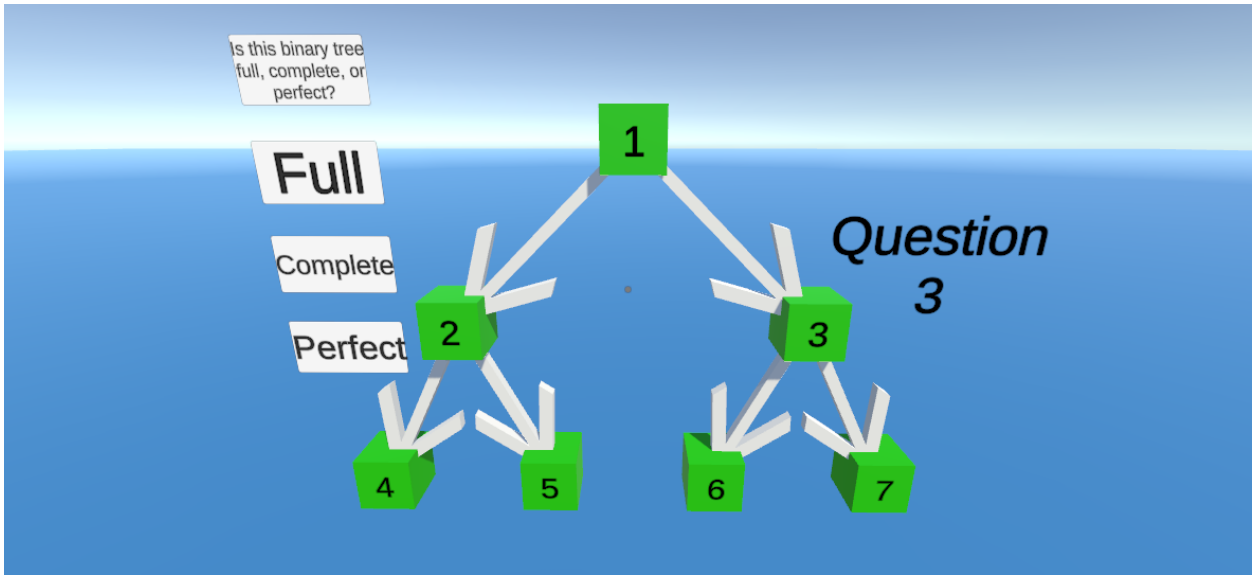
In the module, there is content to teach about the different properties of binary trees, and what it means to have a full, complete, and perfect binary tree.



After, the participants proceed with an embedded assessment. First, they get a sample quiz question to familiarize them with the format.

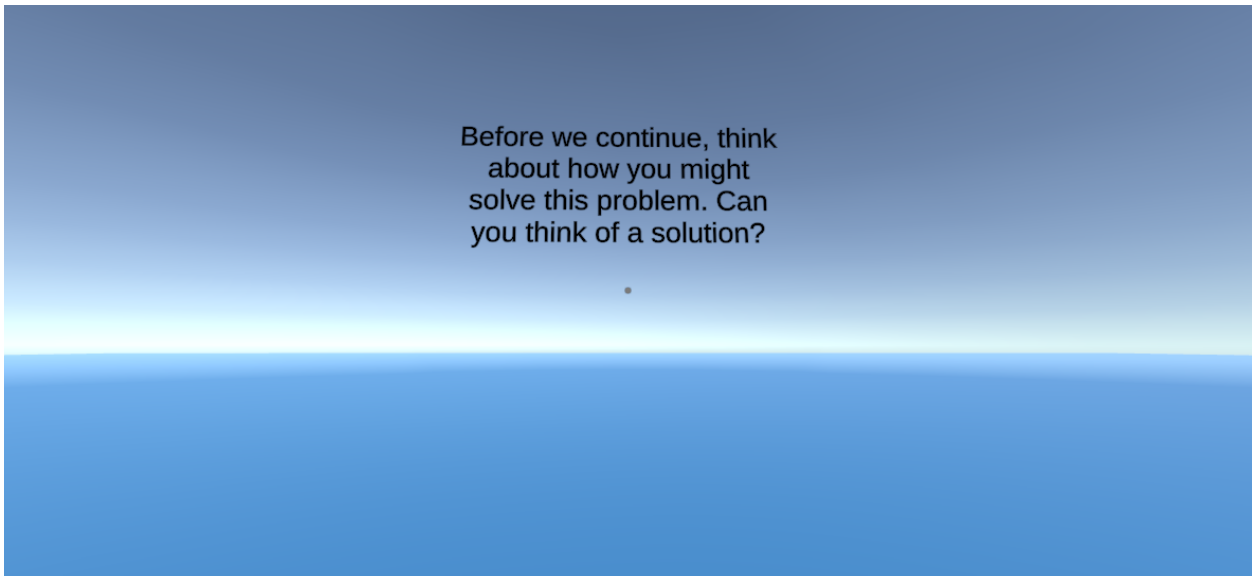


Then, students proceed through the actual content-related questions.

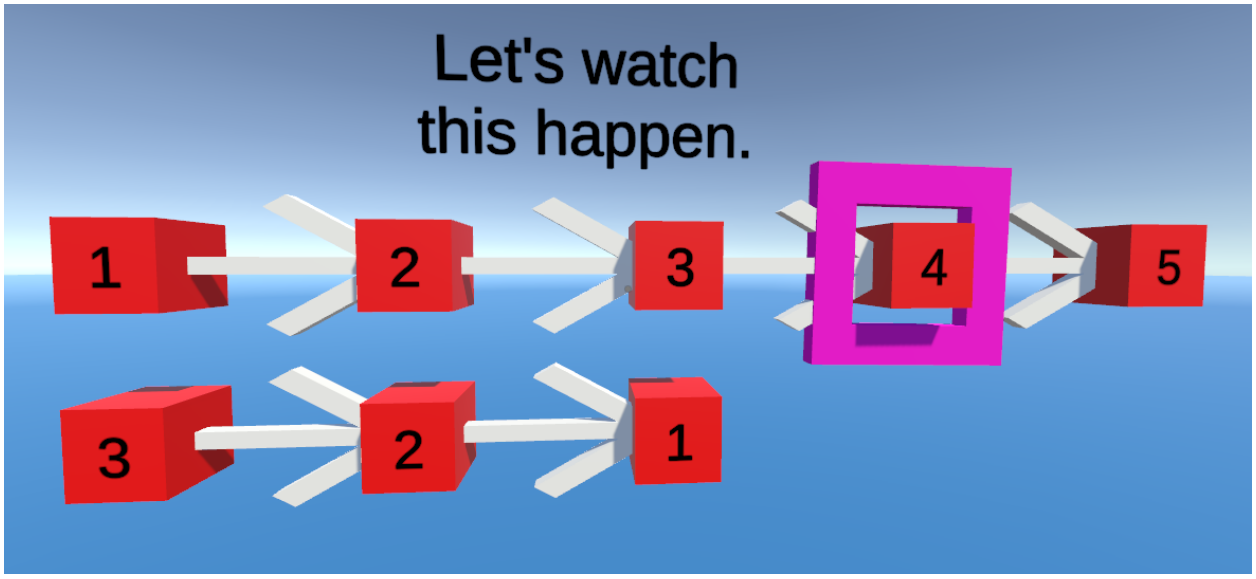


Once they've completed this, they'll move onto the module for Linked Lists.

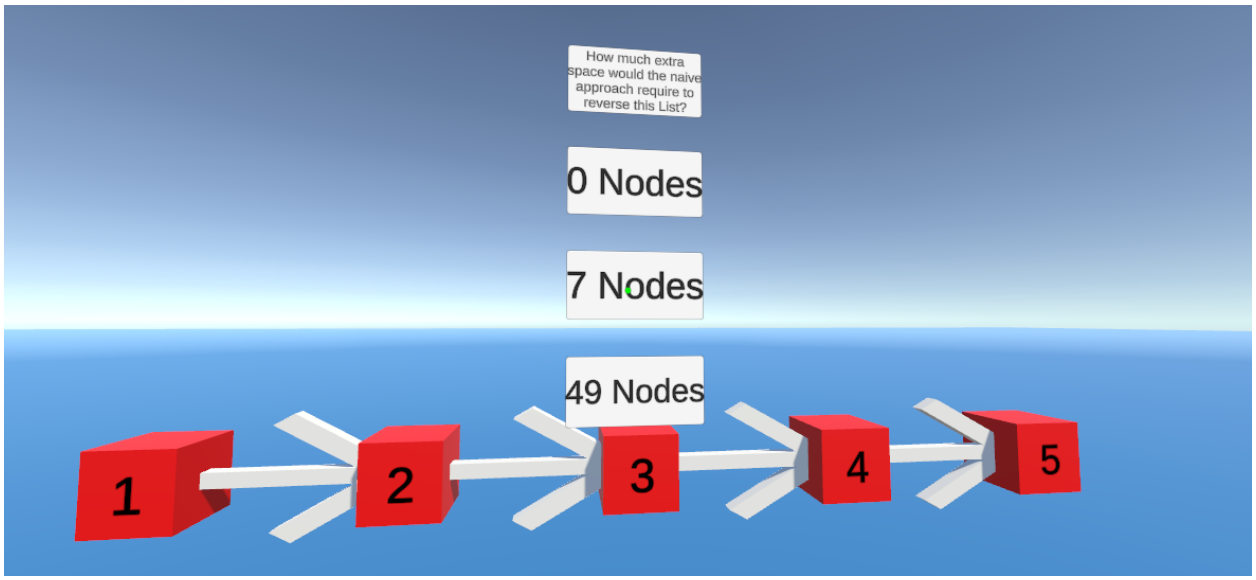
Now that they're familiar with the format, the content can get a little more complex. After a brief introduction to the concept of linked lists, they're presented with Linked List reversal.



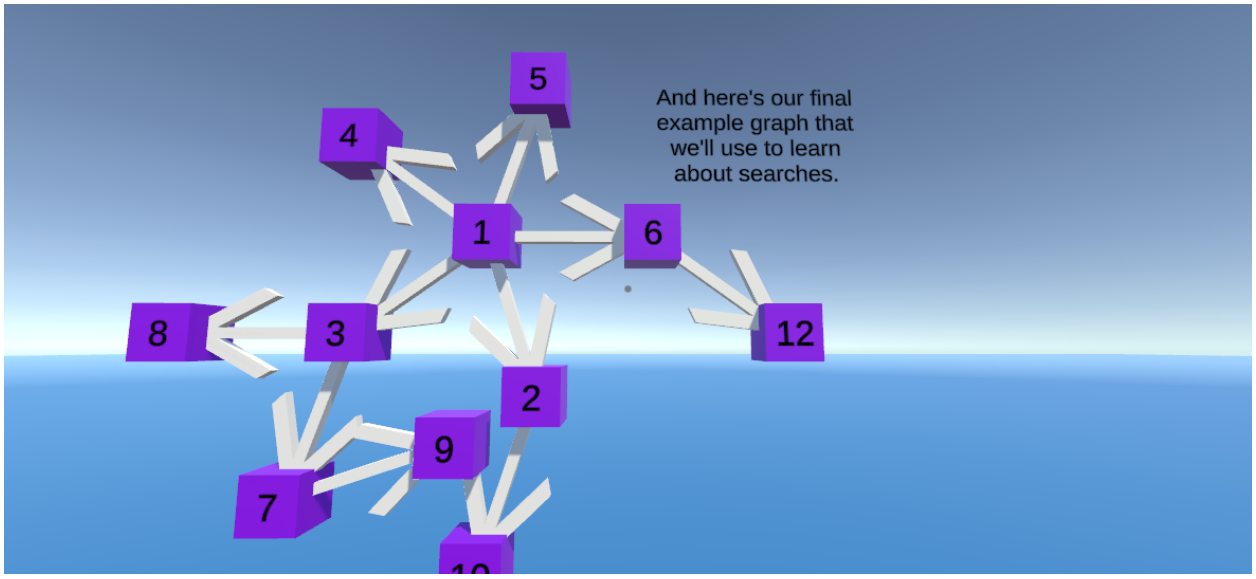
To keep students engaged, they are asked to try and think of a solution first. After this, they're presented with the "naive" and the "clever" solution for this problem.



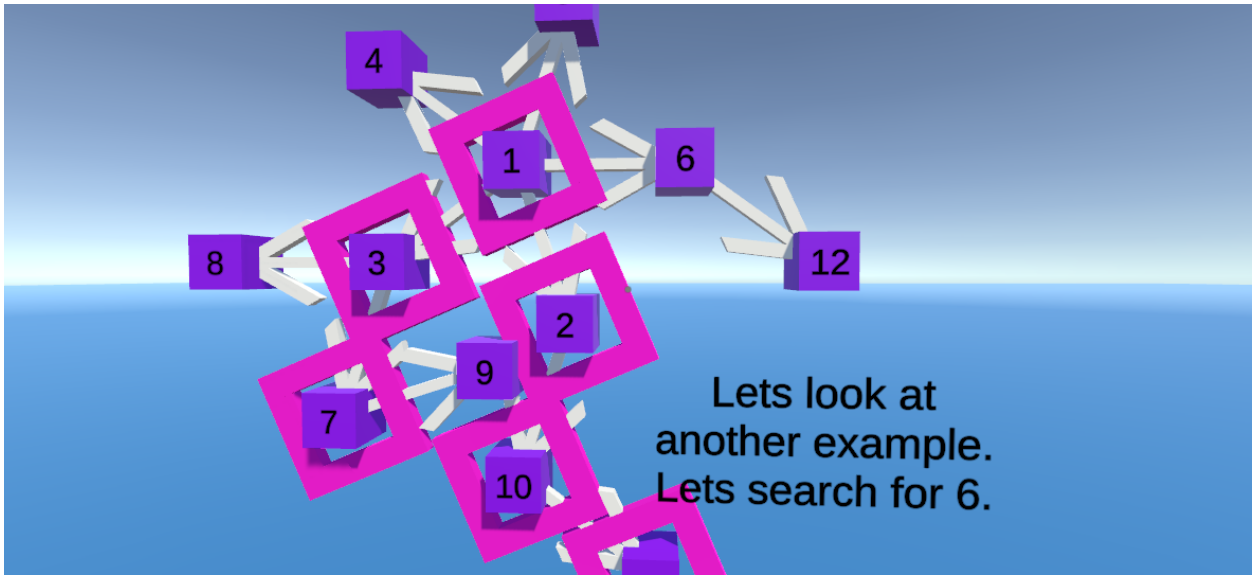
I also touch on complexity, as both are O(n) time, but O(n) and O(1) space complexity respectively.



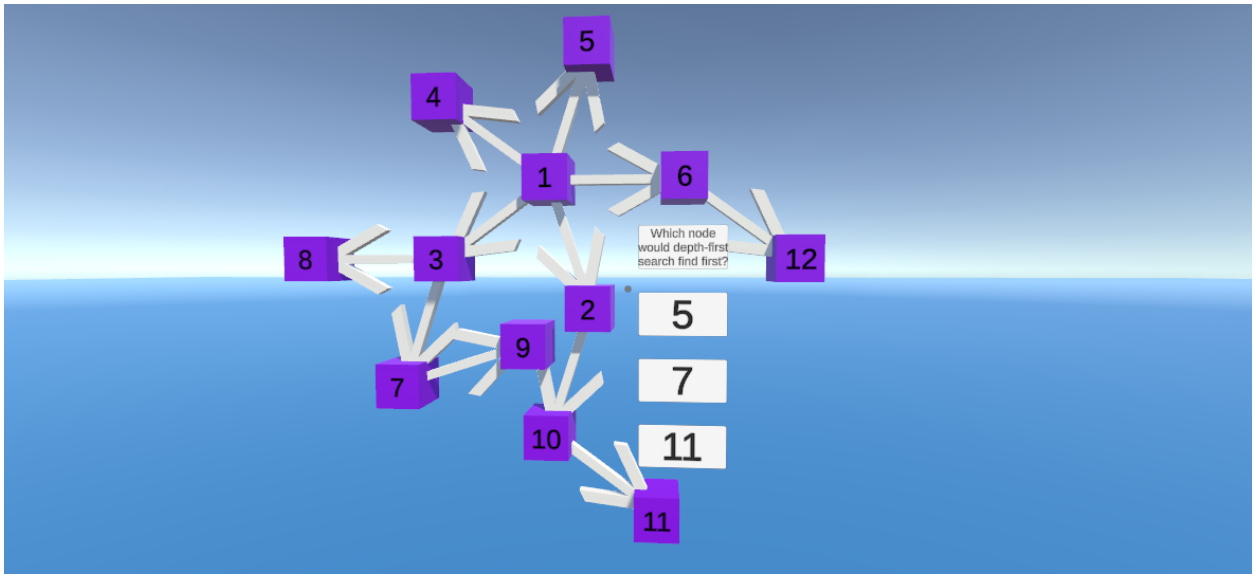
Again, students proceed through a sample quiz question, then the embedded assessment.



For the final module, there is both a dive into a data structure and relevant algorithms.



Participants learn about what a graph is, and then two different types of searches. They learn about the differences between depth and breadth first searches, and how different graphs and use cases can serve the different algorithms better.



Of course, they then proceed to complete a final embedded assessment, testing their understanding of both searches.

RESULTS

Perhaps unfortunately, the results lack sufficient evidence to reject the null hypothesis. The mean results from the embedded assessment were nearly identical, at about 75% correct. (Table 1) In fact, the mean score for the Linked List modules was exactly identical, with the exact same number of total correct answers between the two groups.

Table 1 - Data from embedded assessments

		Binary Trees	Linked Lists
PC	Mean	0.7428571429	0.775
VR	Mean	0.7571428571	0.775

(See appendix for raw data)

This, fairly conclusively, shows that Virtual Reality - in and of itself - does not offer benefits to learning outcomes. That said, there's still some useful information gained, both in the context of future work, and around student engagement with VR as a medium.

The only statistically significant differences observed were with the results on enjoyability. It's important to note that these results are ordinal: "strongly agree" only tells that someone agrees more than a "slightly agree" result, but not at all by how much. These results were tested by the Mann-Whitney U test, which is a test for ordinal data. According to that test, the results for “I would like to learn more content through Virtual Reality/Desktop” are statistically significant.

Otherwise, the data was not statistically significant with a p-value under 0.05. However, the open-feedback responses are still certainly informative. Participants reported that:

"I feel like it was more engaging to me to be in VR. helps me focus"

"I really enjoyed the VR and to treat it as a game in a sense."

"I think that learning about data structures in VR was very informative and far more enjoyable then learning about them on the desktop experience."

Clearly, some students find this to be a viable, more engaging, more fun medium compared to traditional content.

LIMITATIONS; CONTEXT FOR FUTURE WORK

These results also can serve as contextual work for future studies. This is for a couple reasons.

1. The modules were identical in content.

This study intended to show that VR might offer benefits in and of itself, that simply the fact of engaging with content in VR makes for better learning outcomes. Though that does not appear to be the case, there is certainly design space for learning materials that make use of the unique functionality that VR has to offer.

In some of the participant feedback, we heard:

"I think that the VR experience has more potential, but with the way these specific tutorials were set up, there wasn't as much benefit as there could have been to make the most of the medium. The VR experience felt like it was just the desktop version ported over to VR instead of something purpose-built to take advantage of VR."

This is true, as we wanted to investigate particularly whether VR had benefits in and of itself, but this also limited the possible scope. Another participant said:

"I believe that interacting with the material in VR has a lot of potential, but only if the material is broken down in a way that only VR could accomplish. Running learning software through a more complicated platform without using the unique capabilities of that platform would only serve to complicate the learning process."

For experienced Virtual Reality users, the modules didn't use the capacities of the medium enough. Originally, the software was intended to be a sandbox, in which you could move around, create new nodes, link them together, organize data structures, run algorithms, and so on. Though the functionality is there, it became clear throughout the testing that any tutorial that could introduce a non-experienced user to these tasks would rapidly exceed the target time for even the whole activity. Consequently, future work that discards the need for content to be exactly identical, and instead allows VR users to attain much deeper interaction with the learning materials may yet result in the better learning outcomes we sought in this study.

2. The target time was short.

One clear difference we noticed between some of our participants was their familiarity with VR. For users that often use VR, or even more so have a headset themselves, progressed through the tutorials and learning modules remarkably faster than those who didn't. Though time to complete was not a measured factor, it is sensible that this is the case.

With a much longer engagement per user, and full time for acclimatization to the environment, it is possible that differences could emerge.

3. This study tested things students already knew to some degree

One compounding factor with this study is students familiarity with the concepts. Due to the limited scope, we were unable to select students with only a limited range of familiarity with these concepts. Though this was inquired about on the post-survey, the sample size was insufficient to draw any conclusions by limiting to only students who had or hadn't seen the concepts before. In future work, a consideration for existing knowledge, maybe in the form of a pre-post comparison, could elucidate the impact that existing knowledge has on performance, whether it differs with VR, and remove this as a confounding factor.

4. Students said VR was more engaging.

As mentioned above, participant testimonials included a number of reports that VR was more immersive, helped people focus, and was overall more fun.

Though hard to quantify, consistently they reported that VR was more fun and engaging. Keeping students engaged is an unending challenge for educators, and virtual reality seems that it may offer benefits in this area. This may be partially because of the novelty of VR, which may wear off with longer intervention. Again, worth further investigation.

Consequently, the context for future studies is as follows. If we had access to infinite time, willing participants, and budget, the study design would basically be a significantly scaled version of this study.

Over the course of a semester or long time frame course, students could be separated into two groups. One would receive content in Virtual Reality, the other traditional materials. These materials could have some different activities, while covering the same conceptual content.

Retaining a modality switch could also still provide useful insight. With a long time frame, one could evaluate the differences in the pre-post swap. If students improved after switching to VR, and/or got worse after switching to traditional materials, that serves as further evidence for this phenomenon.

Regardless, the clear potential for extending the scope, the reported engagement by students, and the relative lack of research in this area all necessitates future study.

CONCLUSION

Though in this case, we failed to reject the null hypothesis, the process and data were still instructive. We accomplished about as much as is possible in a one-semester project, and received unprecedented interest in participating in the study.

Personally, I have deeply enjoyed the process of creating, refining, and testing this software, and I hope that the results that we found will be of use to further research!

Following are the supporting materials for this support. First, the contents of the approved IRB materials used to conduct the study. Then, a full printout of the source code used in the application.

REFERENCES

Akbulut, A., Catal, C., & Yıldız, B. (2018). On the effectiveness of virtual reality in the education of software engineering. *Computer Applications in Engineering Education*, *26*(4), 918–927. https://doi.org/10.1002/cae.21935

Buchbauer, B. (n.d.). *Educational Computer Science Visualizations in Virtual Reality*. Retrieved May 8, 2023, from https://diglib.tugraz.at/download.php?id=5df7944d0141a&location=browse.

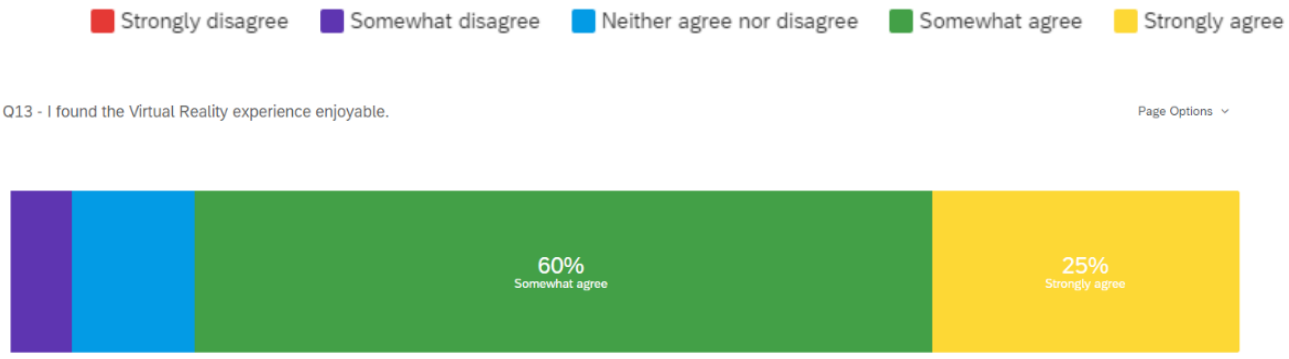
Gikandi, J. W. (2019). Meeting higher education expectations in the Digital age and reliability of assessment in e-learning settings. *Handbook of Research on Cross-Cultural Online Learning in Higher Education*, 79–100. https://doi.org/10.4018/978-1-5225-8286-1.ch005

Grivokostopoulou, F., Perikos, I., & Hatzilygeroudis, I. (2016). An innovative educational environment based on virtual reality and gamification for learning search algorithms. *2016 IEEE Eighth International Conference on Technology for Education (T4E)*. https://doi.org/10.1109/t4e.2016.029

Li, F., Li, D., Zheng, J., & Zhao, S. (2015). Virtual experiments for introduction of Computing: Using virtual reality technology. *2015 IEEE Frontiers in Education Conference (FIE)*. https://doi.org/10.1109/fie.2015.7344376

APPENDIX A - RAW DATA

GUID	Platform	BT %	LL %
5893c536	VR	0.8571428571	1
79582f06	VR	0.7142857143	0.75
0827691a	VR	0.7142857143	1
d8d8a8c3	VR	0.8571428571	1
74d0ce90	VR	0.7142857143	0.5
470e8c69	VR	0.8571428571	0.75
67115757	VR	0.4285714286	0.75
acee16b3	VR	0.8571428571	0.25
6f06591d	VR	0.7142857143	1
9dae8f49	VR	0.7142857143	0.75
	Mean	0.7428571429	0.775
	StDev	0.1312766548	0.2486072315
fc062d4d	PC	0.7142857143	1
31057984	PC	0.8571428571	0.5
5f106de76	PC	0.8571428571	0.5
0fb110a6	PC	0.7142857143	1
ed2f4543	PC	0.8571428571	0.75
5f1506ce	PC	1	1
174b5d0c	PC	0.7142857143	1
a0ac8a38	PC	0.8571428571	0.75
9bc3779a	PC	0.5714285714	1
071b554f	PC	0.4285714286	0.25
	Mean	0.7571428571	0.775
	StDev	0.1656431155	0.2751262337

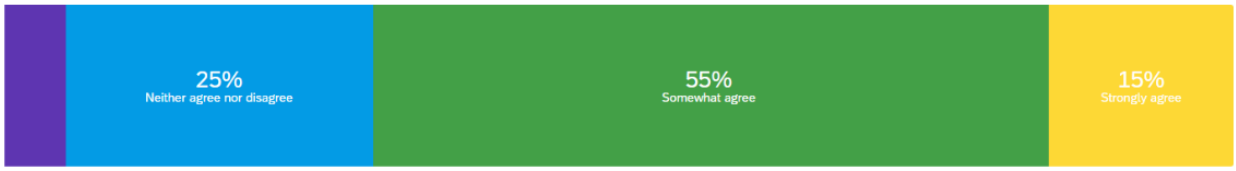


Q14 - I found the Virtual Reality experience informative.

Page Options

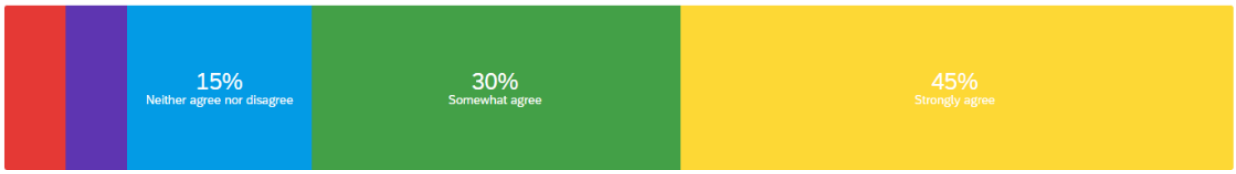
Q15 - I found the Virtual Reality experience easy to use.

Page Options



Q16 - I would like to learn more content through a Virtual Reality experience like this.

Page Options

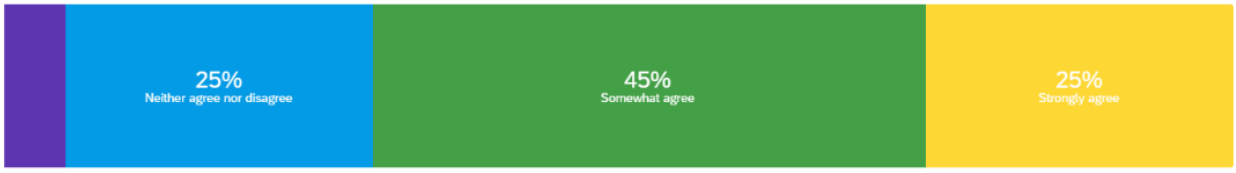


Q17 - I found the desktop experience enjoyable.

Page Options

Q18 - I found the desktop experience informative.

Page Options



Q19 - I found the desktop experience easy to use.

Page Options

Q20 - I would like to learn more content through a desktop experience like this.

Page Options

APPENDIX B - IRB SUBMISSION MATERIALS

Virtual Reality Study Interest Form

If you are a UML student who is over 18 years of age, and you're interested in helping us study Virtual Reality as a possible education tool, please fill out this form! You will receive a 25$ Amazon gift card as a result of participating in this study.

* Required

1. Are you over 18 years old? *

Mark only one oval.

- ◯ Yes
- ◯ No

2. Are you a UML Student? *

Mark only one oval.

- ◯ Yes
- ◯ No

3. Are you a Computer Science major? *

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ No, but I am going to change majors to become one.

Info

Please add your contact information so we can have you participate in the study :)

4. Student Email *

5. Phone Number (optional)

INVESTIGATOR STUDY PLAN - REQUIRED

1. TITLE
Examining the Efficacy of Virtual Reality for Learning Data Structures in Computer Science

2. EXTERNAL IRB REVIEW HISTORY*
N/A

3. PRIOR APPROVALS:
N/A

CONFLICT OF INTEREST (COI):
N/A

BIOHAZARDOUS AGENTS:
N/A

RADIATION:
N/A

4. BACKGROUND*
Virtual reality seems to have potential to be an extremely effective tool for aiding in learning. Some studies have attempted to examine how effective it is, but few examine precisely whether VR as a medium itself offers any substantial benefits.

This study will have a series of exercises and quizzes examining knowledge of data structures. These will be delivered in two formats: via an interactive VR interface and via a traditional desktop computer interface. We will split the study participants randomly into two groups of users. One group will be presented with a set of exercises in the VR format and then additional exercises in the desktop format. The other group will get the same set of exercises in the reverse order (i.e, desktop first and VR second).

The research will evaluate participant performance based on their interactions with whichever modality they experienced first. This intends to examine whether VR has any tangential benefit.

5. OBJECTIVES*
The hypothesis is that VR activities are superior to traditional interactive exercises.

The participants will be given three modules, one on Binary Trees, one on Linked Lists, and one on Graphs and Searches. These modules will include some instructive content, some interactive work with these data structures and algorithms, and finally, an embedded assessment.

6. STUDY OUTCOMES*

The primary outcome is a positive effect on students understanding of data structures and algorithms. Secondarily, students may also become more accustomed with virtual reality technology.

Hypothetically, students who receive a virtual reality experience should have better outcomes on the embedded assessments.

If participants report dizziness while using the VR headset, they will be asked to discontinue the study.

7. Inclusion and Exclusion Criteria*

The participants will be Computer Science students at UMass Lowell. Students must be of majority age to be qualified to participate in the study (i.e., they can self-consent).

No participants meeting any exclusion criterion will be included in the study.

8. Vulnerable Populations*

Yes, a professor will be involved in overseeing the study, but he will not be involved in recruitment nor in the consent process, so he will not be capable of knowing who participates in the study.

9. Setting

The study will be conducted in a lab on campus – Dandeneau 408.
Recruitment will occur via:

-A flyer posted on campus (attached)
-An email to the computer science mailing list (attached)
-A message to the UML-ACM discord (attached)

The researcher is a student at UML. Data analysis will take place on-campus and at home.

10. Resources Available

Primary Investigator – A faculty member who has the authority to make sure study procedures are followed. The PI should have CITI training, and prior experience supervising student researchers. The PI will devote up to one hour per week supervising this work.
Student Researcher – Will do the work as documented for carrying out the study, including the recruitment and consent process.The student researcher should have CITI training and strong digital literacy skills. The Student Researcher will devote on average eight hours per week to preparing and conducting the study.

The primary researcher and student researcher have been collaborating closely in the preparation in this document and both are fully aware of all procedures described here.

11. Study Timelines*

April 2023 – Study conducted; data obtained
May 2023 – Data analysis complete, research written up, presented to UML Honors College.

Each subject will be participating for about one hour.
There will probably be a few hours of recruitment effort to get participants for the study.

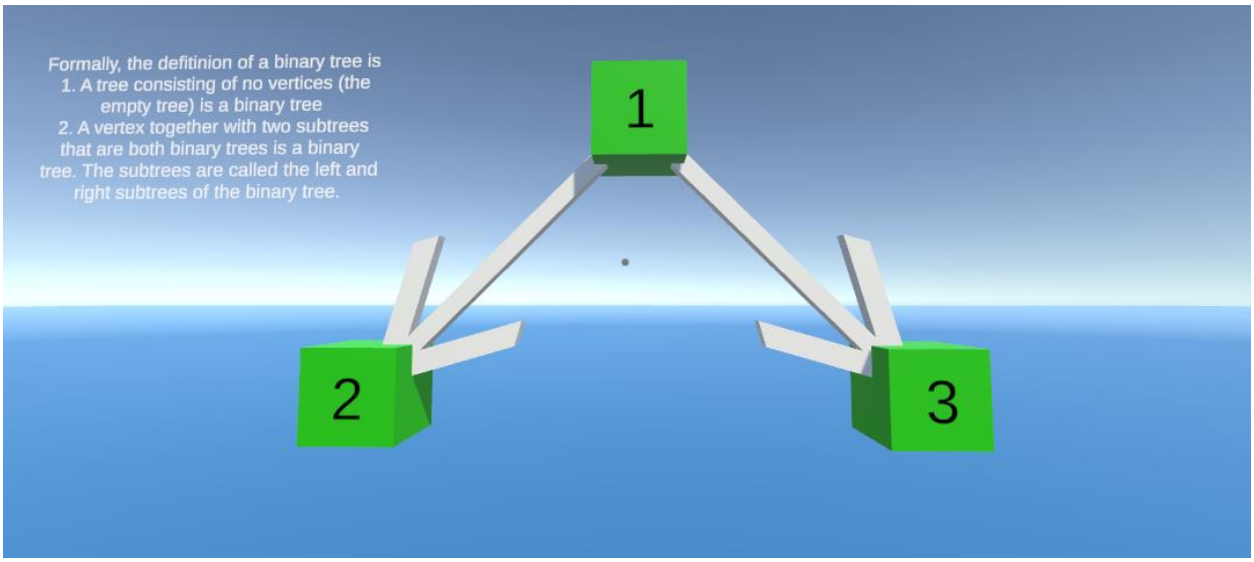
12. NUMBER OF SUBJECTS*

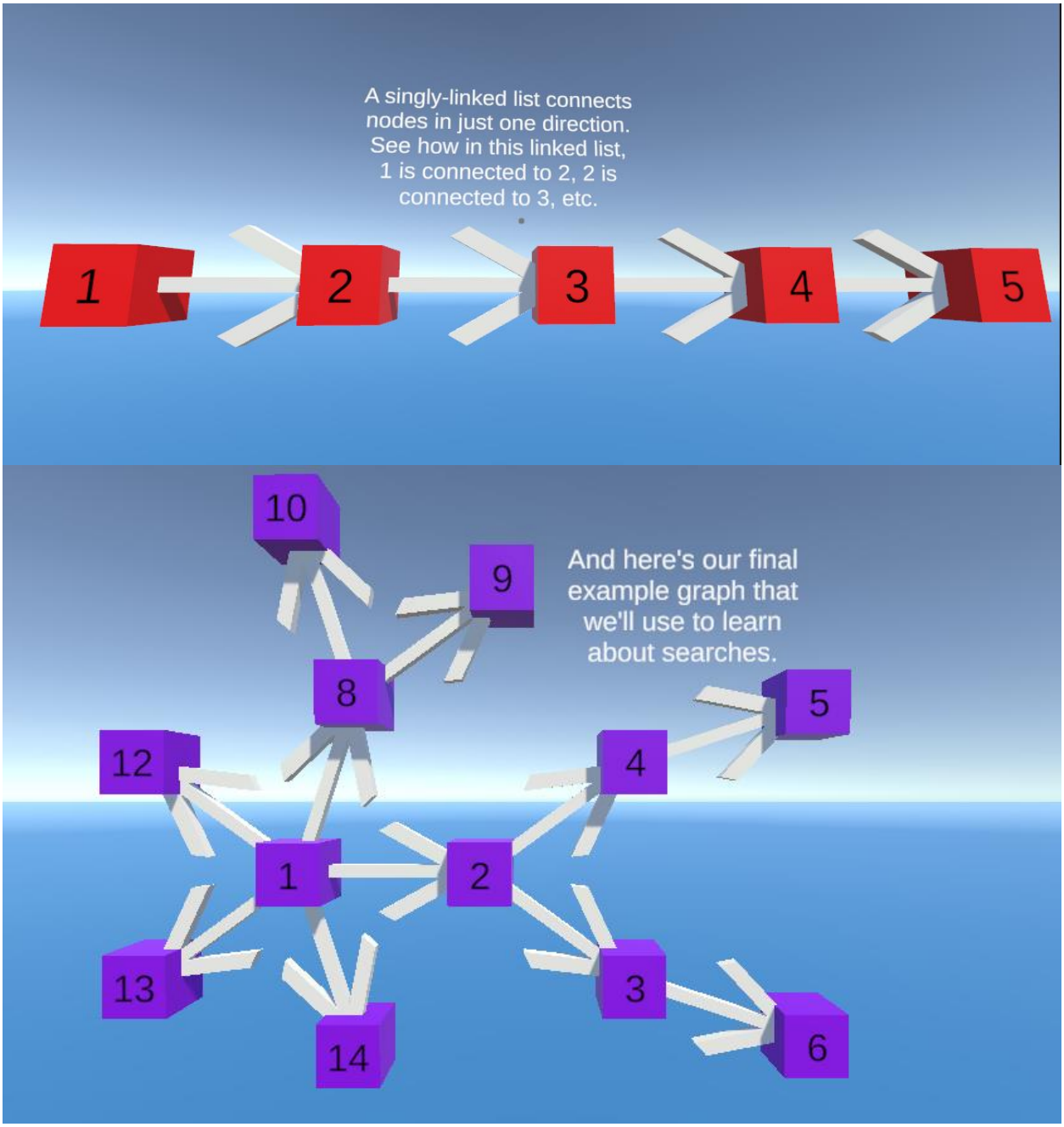
We anticipate recruiting 20 participants. All who qualify and do not meet any exclusion criterion will be accepted. All will be UML Computer Science students who consent to the study.

13. PROCEDURES INVOLVED*

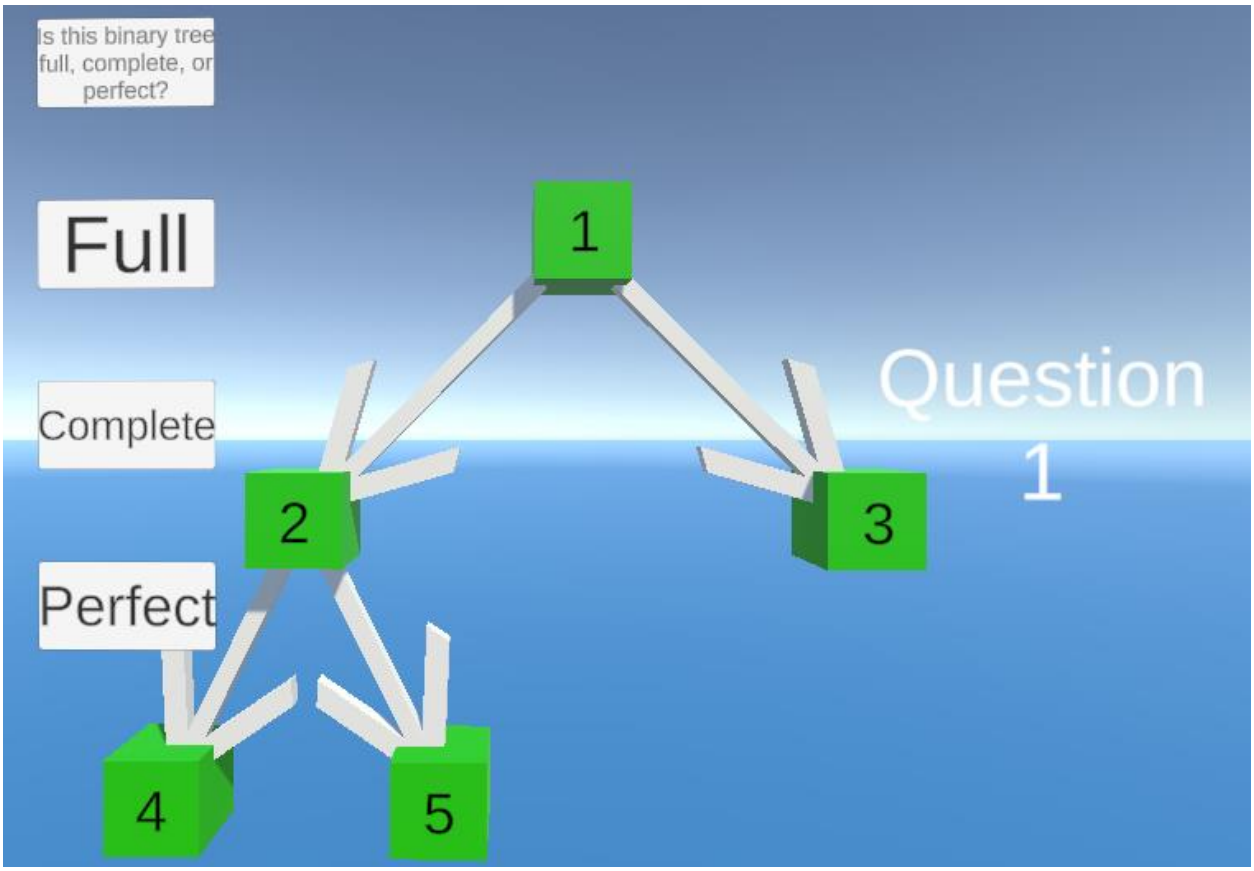
After the subject provides informed consent, they will be given one of the learning materials, either Virtual Reality (Quest 2) or on a laptop. They will complete two modules, including the embedded assessment, then switch modalities.
The learning materials each give an introduction to one concept in Data Structure and Algorithms. The three modules are on Binary Trees, Linked Lists, and Graphs/Searches.

Since the same content will be delivered in both VR and via a laptop, the content is shown in 3D space, so it can be independent of the medium through which it is delivered. The user will either control this through the keyboard and mouse, or via the VR controllers, and there will be a tutorial for both parts.





For measuring outcomes with each module, the experience has an embedded assessment. While students are learning, they will be asked “quiz questions” that test their knowledge of the subject material. They will be alerted to this fact.



Once they finish the first modality, the experience will generate a randomized GUID (Globally Unique Identifier) in the form of a 128 bit string (something like 91dca76c-7b8a-410f-9f61-7696d2c27407). This will be recorded for later use.

After completion of the second modality, the subject will begin the post survey. The user will enter the first part of this into their post-survey, and this will be their identifier. No names or other identifiers will be collected. Only the first part will be displayed to the user for them to copy over to the Qualtrics post-survey.

Student work in the experiences and in the post-survey will only be linked by this study ID (GUID).

Students who complete the procedure will receive a $25 Amazon giftcard.

The target time is 45 minutes. 5 minutes for the tutorial to the experience, and 10 minutes for each module, then another ten minutes for the post-survey. Each student will be scheduled for 1 hour to allow for technical difficulties or longer time in each module.

14. RECRUITMENT METHODS*

As aforementioned, recruitment will occur via:

-A flyer posted on campus (attached)
-An email to the computer science mailing list (attached)
-A message to the UML-ACM discord (attached)

It is believed that recruiting 20 computer science students is very reasonable. There are hundreds of computer science students, so this is a small percentage that is needed to recruit. This is outside a clinical setting.

Individuals who participate, due to the recruitment methods, must necessarily be UML students. Consequently, they are all adults. Participants may withdraw at any time.

Copies of recruitment materials are attached.

15. CONSENT PROCESS*

All study staff are familiar with and will follow HRP-802 INVESTIGATOR GUIDANCE: Informed Consent.

Once subjects arrive, they will be given a copy of the consent form. They will have time to read over it, and agree, but written documentation will not be obtained. They will be informed that participation is fully voluntary and optional, and they may choose to stop participation at any time. The student researcher will ask them for their verbal consent to participate.

There will be at least a few days delay between the end of recruitment and the start of participation, such that subjects can choose whether or not they wish to participate.

After each module, the investigator will check in with subjects to make sure they are comfortable and willing to continue the study. If any student wishes to revoke consent, they will be allowed to do so and will be told they are free to leave at any time.

16. PROCESS TO DOCUMENT CONSENT IN WRITING

Consent will not be documented in writing. Research presents no more than minimal risk of harm to subjects and involves no procedures for which written documentation of consent is normally required outside of the research context

17. WITHDRAWAL OF SUBJECTS WITHOUT THEIR CONSENT*

N/A

18. SHARING OF RESEARCH RESULTS WITH SUBJECTS*

Research results will be published to the UML Honors Project archive, but this will just include the general results.

Depending on study outcomes, we may subsequently prepare an article for publication in a computer science education conference (e.g. SIGCSE or CCSCNE).

19. RISKS TO SUBJECTS*
The risk is no more than minimal.

Students who engage with VR may become motion sick and may at any time remove the headset or decline to continue participating in the study to resolve this.

Students may become confused or frustrated by the subject material or the embedded assessment, but the risk of this is minor, as are the consequences.

There is the risk for disclosure.

20. POTENTIAL DIRECT BENEFITS TO SUBJECTS*
There are no direct benefits to participants.

21. DATA AND SPECIMEN ANALYSIS AND MANAGEMENT*
Data will be received on UMass Lowell's Qualtrics system. Students will be provided with a unique key which they will use as a confidential identifier when filling out the surveys.
No names or meaningful identifiers will be collected.

Data will be kept for four years after the study concludes for potential use in further research.

The experience will provide a GUID to match the results of the embedded assessment to the results of the pre-survey.

22. PROVISIONS TO MONITOR THE DATA TO ENSURE THE SAFETY OF SUBJECTS*
N/A

23. DATA AND SPECIMEN BANKING*
Per #21, we plan to keep the deidentified pre- and post- survey results for possible future use and interviews. After the study concludes, student researcher access to the data will be removed, and only the faculty advisor will have access to the data. The survey data will be deleted four years after the study concludes. There is no plan to provide access to the data to other parties.

24. CONFIDENTIALITY

-

Data will be a set of results from the embedded assessment (question #, success), and the results of the post survey. They will have the study ID as their identifier, but this is not linked to any subject information.

Only the PI and Student Researcher will have access to the data. Data will be stored on the laptop used to conduct the study, and on the Quest 2 (VR headset) used to conduct the study. This data may be transferred to the researcher's home computer during analysis. Access will only be available to study personnel, and will be password protected. Data from the post-survey will be saved on the Qualtrics servers. Data will be transferred to UML servers and kept for up to five years for future research, but then it will be completely destroyed. All data will be anonymous and will not contain identifiable information.

25. PROVISIONS TO PROTECT THE PRIVACY INTERESTS OF SUBJECTS
N/A

26. COMPENSATION FOR RESEARCH-RELATED INJURY
N/A

27. ECONOMIC BURDEN TO SUBJECTS
N/A

28. COMMUNITY-BASED PARTICIPATORY RESEARCH*
N/A

29. MULTI-SITE RESEARCH*
N/A

30. RESEARCH CONDUCTED IN A FOREIGN COUNTRY

N/A

31. DRUGS OR DEVICES
N/A

IRB Checklist

This checklist is provided for your convenience and is not a requirement for review.

	Investigator Study Plan (ISP)
	Consent form(s)
	Assent forms(s)
	Fact sheet(s)
	Surveys, measures, instruments, etc.
	Data collection sheets, case report forms, etc.
	Recruitment materials such as flyers, brochures, posters, scripts of radio ads, etc.
	Written approvals from ancillary reviews (COI, IBC, RSC,)
	Adverse event log
	Approval order for Humanitarian Use Device
	Certificates of translation or translator attestationsFact sheet(s)
	Compensation logAssent forms(s)
	Data safety monitoring plan
	Data use agreements, memoranda of understanding, Multi-site communication plan
	DMC or DSMB charter
	Documentation of data/specimen anonymity (i.e., provider will never break the code)Study staff training plan
	HIPAA authorization
	HIPAA waiver
	IND or IDE documentation
	Instructions for use or approved FDA labeling for devices
	Investigator brochure or package insert for drugs
	Measures to assess capacity to consent
	Multi-site communication plan
	Patient information sheet for Humanitarian Use Device
	Product labeling for Humanitarian Use Device
	Screening logConsent form(s)
	SOPs or Manuals of OperationsAuthorization to contact form
	Sponsor justification or FDA documentation for non-significant risk device study
	Study staff training planHIPAA authorization

Consent for Research Participation
IRB #: 23-040-MAR-EXM
IRB Approval Date: 4/3/2023

Study title: Examining the Efficacy of Virtual Reality for Learning Data Structures in Computer Science

Researcher[s]: Matt Waterman, BS Student; Dr. Fred Martin, PhD

We're inviting you to participate in a research study. Participation is completely voluntary. If you agree to participate, you can always change your mind and withdraw. There are no negative consequences, whatever you decide.

What is the purpose of this study?

The purpose of this study is to evaluate whether Virtual Reality is a better learning environment for teaching students Data Structures and Algorithms. To do this, we will have students interact with learning materials in both environments, and compare their results.

What will I do and how long will it take?

You will interact with both a Virtual Reality experience and an experience on a computer. You will be asked questions during the experience that are strictly about the content, related to Data Structures and Algorithms. Then, you will complete a post-survey that will ask questions about you as a UML Student, your experiences with Virtual Reality, your familiarity with various materials and your class history, and then about your experience with the various learning materials. This should take no longer than one hour.

Could being in this research hurt me?

- There is a risk that your online data could be intercepted: This is a risk you experience any time you provide information online. We're using a secure system to collect this data, Qualtrics, but we can't completely eliminate this risk.
- There is a chance your data could be seen by someone who shouldn't have access to it. We're minimizing this risk in the following ways:
    - All identifying information is removed and replaced with a study ID, so we are unable to link your data and post survey to you.
    - We'll store all electronic data on the servers for the online survey software (Qualtrics).
- You could experience motion sickness or discomfort as a result of using the Virtual Reality headset. You may take a break or choose to stop participating at any time.

What happens if I am injured because I took part in this research?

If you are injured while in the study, seek treatment and contact the your doctor as soon as you are able. The University of Massachusetts Lowell does not provide funds for the treatment of research-related injury. If you are injured as a result of your participation in this study, treatment will be provided. You or your insurance carrier will be expected to pay the costs of this treatment. No additional financial compensation for injury or lost wages is available. You do not give up any of your legal rights by signing this form.

What other choices do I have besides taking part in this research?

Your alternative is to not take part in the research.

Will being in this research help me in any way? No.

How many people will take part in this research? About 20 Students.

Will it cost me any money to take part in this research? None.

Will I receive any compensation or incentive for participating in this study? Yes, you will receive a $25 Amazon giftcard upon completion of the post-survey. You will not receive a gift card if you choose to withdraw from the study.

In order to receive a stipend for study participation, you will need to give us private information like your name, address and/or phone number. We will then share this information with the UML Student Affairs Financial Administration Department that require this information to process the payment. NOTE: You will need to provide your social security number and complete a W-9 (tax form) if you receive:

- $300 or more from a single study within a single calendar year at UML, or
- $600 or more in a calendar year across multiple research studies at UML.

UML may report the payment to the IRS and send you a 1099 form for tax purposes. The business offices and companies will keep the information as part of their financial records. The research team will destroy this information three years after study closure.

How will my information be stored and when will it be destroyed?

Your data does not include any personally identifying information, as it is only recorded with a study ID. It will be securely stored on Qualtrics servers, and will be kept for up to five years after the completion of the study.

- It is possible that we might use the research data [and specimens] in other future research. We may also share data [and specimens] with researchers and companies that are not part of UML. In these cases, we will not share your name or other information that identifies you directly, and we will not come back to you to ask you for your consent.

Who can see my data?

- We (the researchers) will have access to only your de-identified (no names, birthdate, address, etc. results from the embedded assessment and post-survey. This is so we can analyze the data and conduct the study.
- The Institutional Review Board (IRB) at UML may review all the study data. This is to ensure we're following laws and ethical guidelines.
- We may share our findings in publications or presentations. If we do, the results will be either the aggregated (grouped) data, without individual results, or the general trends from the study. This will not include any identifying information.

Contact information:

For questions about the research, complaints, or problems: Contact Matthew Waterman, matthew_waterman@student.uml.edu, 978-596-7425

For questions about your rights as a research participant, complaints, or problems: Contact the UMass Lowell IRB (Institutional Review Board) at 978-934-4134 or at IRB@uml.edu

Before we begin,

1) Are you 18 years of age or older? ___Yes or ___No If no, thank you for your time!
2) Do you have any questions? ___Yes or ___No
3) Do you agree to voluntarily participate in this research? ___Yes or ___No (STOP HERE)

Researcher Obtaining Verbal Consent:

Name of Researcher:_____

Date of Verbal Consent: _____ Time:_____

Obtained Verbal consent via: ☐ in-person

The following will be sent as an email to the CS-MSGS email mailing list, as well as the UML Association for Computing Machinery (ACM) discord server.

Hi All,

My name's Matt, and I'm conducting a research study about using Virtual Reality as an education tool for Data Structure and Algorithms under the supervision of Fred Martin, PhD. The goal of the research is to understand whether Virtual Reality offers benefits for student learning, and if so, what kind of benefits it offers.

If interested, participation will take about an hour, and you'll interact with both a virtual reality experience and a desktop experience, then take a post-survey. If you're interested in helping me study Virtual Reality, please reply to this or fill out this google form https://forms.gle/iQ8X2jd1utLRRXaN6.

As a thank you, you'll get a $25 amazon gift card at the end of the study! It'll be conducted on North campus, so there is no need for far travel.

If you have any questions contact me via email, at matthew_waterman@student.uml.edu.

I hope to hear from you!

Best,
Matt Waterman

OFFICE OF RESEARCH INTEGRITY
600 Suffolk St, Wannalancit Mills Suite 212
Lowell, MA 01854
T: 978-934-4134 **F**: 978-934-6012
https://www.uml.edu/research/integrity/

4/3/2023

Fred Martin
Computer Science
198 Riverside St
Olsen Hall 2nd & 3rd Floors
Lowell, MA 01854-
(978) 934-1964
fredm@cs.uml.edu

Dear Fred Martin, PhD:

The IRB reviewed the following submission:

Type of review:	**Initial**
Title:	**Examining the Efficacy of Virtual Reality for Learning Data Structures in Computer Science**
Principal investigator:	**Fred Martin, PhD**
IRB number:	**23-040**
Level of Review:	**Exempt 45 CFR 46.104(d)(3)(i)(A)**
IND or IDE, if any:	**N/A**
Funding Source, if any:	**N/A**
Documents reviewed:	**HRP-200, HRP-504, Consent, Email & Discord, Flyer, Post Survey, Google sign-up form**

This research is reviewed under the 2018 regulations. It is neither FDA nor DOJ regulated.

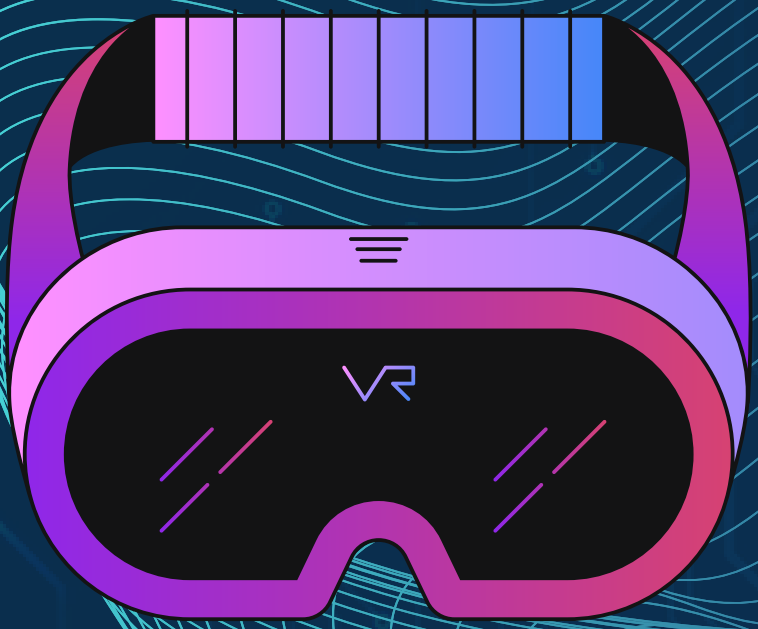
The IRB approved this study effective 4/3/2023.

Copies of any approved consent documents, consent scripts, or assent documents are attached.

In conducting this research, you are required to follow the requirements in "HRP-070 Policy: Investigator Obligations."

Sincerely,

Emily Sousa, MA, CIM, CIP
IRB Manager



VOLUNTEERS NEEDED FOR VIRTUAL REALITY STUDY!

Are you interested in helping us study VR as an education tool for Data Structures and Algorithms? It will take one hour of your time, and you'll get to interact with learning materials in Virtual Reality and on a laptop. Scan the QR code above to participate! Participants who complete the study will get a $25 Amazon gift card!

HTTPS://FORMS.GLE/IQ8X2JD1UTLRRXAN6
IRB# 23-040-MAR-EXM

Honors Project Post-Survey - Examining the Efficacy of Virtual Reality for Learning Data Structures

Start of Block: Block 1

Q1 This survey will not collect any identifying information. You should have been given a randomized Globally Unique Identifier (GUID), but this is not related to your name, age, email, etc.

Q2 Please enter the first eight characters of your GUID provided by the software. If you don't remember, please ask for help.

_____

End of Block: Block 1

Start of Block: Block 2

Q3 This section of questions will ask about your background as a UMass Lowell student. If you do not feel comfortable answering these questions, it is advised that you do not participate.

Q4 When did you become a computer science (CS) major?

- ○ Began as a freshman (1)
- ○ Transferred to UML (2)
- ○ Switched majors after first semester (3)
- ○ I intend to switch majors to become a CS major (4)
- ○ I am not a CS major and do not intend to be a CS major (5)
- ○ Other (6) _____

Q5 Which of the following classes have you completed?

	Completed at UML (1)	In Progress / Not Yet (2)	Completed Elsewhere (3)
COMP.1010 Computing I (1)	○	○	○
COMP.1020 Computing II (2)	○	○	○
COMP.2010 Computing III (3)	○	○	○
COMP.2040 Computing IV (4)	○	○	○
COMP.3040 Foundations of Computer Science (5)	○	○	○
COMP.4040 Analysis of Algorithms (6)	○	○	○

Q6 Did you take a computer science / programming course in high school?

- ○ Yes, as a standalone class (1)
- ○ Yes, as part of another course (2)
- ○ No (3)

Q7 Before this activity, were you familiar with these concepts? Check each that is a yes.

- ☐ Binary Trees (1)
- ☐ Linked Lists (2)
- ☐ Undirected Graphs (3)
- ☐ Directed Graphs (4)
- ☐ Binary Tree Properties (5)
- ☐ Linked List Reversal (6)
- ☐ Depth First Search (7)
- ☐ Breadth First Search (8)
- ☐ Algorithmic Complexity (9)

End of Block: Block 2

Start of Block: Block 3

Q8 These questions will ask you about your experiences and habits related to Gaming and VR.

Q9 Do you have experience with virtual reality?

- None (1)
- Not much (2)
- Some (3)
- Yes, a little (4)
- Yes, a lot (5)

Q10 Do you own a VR device? (Not including phone like Google Cardboard)

- Yes (1)
- No (2)

Q11 Do you play video games?

- No, never (1)
- Not Often (2)
- Sometimes (3)
- Yes, frequently (4)
- Yes, all the time (5)

End of Block: Block 3

Start of Block: Block 4

Q12 These questions will ask you about your experience with the various learning materials.

Q13 I found the Virtual Reality experience enjoyable.

- Strongly disagree (1)
- Somewhat disagree (2)
- Neither agree nor disagree (3)
- Somewhat agree (4)
- Strongly agree (5)

Q14 I found the Virtual Reality experience informative.

- Strongly disagree (1)
- Somewhat disagree (2)
- Neither agree nor disagree (3)
- Somewhat agree (4)
- Strongly agree (5)

Q15 I found the Virtual Reality experience easy to use.

- ○ Strongly disagree (1)
- ○ Somewhat disagree (2)
- ○ Neither agree nor disagree (3)
- ○ Somewhat agree (4)
- ○ Strongly agree (5)

Q16 I would like to learn more content through a Virtual Reality experience like this.

- ○ Strongly disagree (1)
- ○ Somewhat disagree (2)
- ○ Neither agree nor disagree (3)
- ○ Somewhat agree (4)
- ○ Strongly agree (5)

Q17 I found the desktop experience enjoyable.

- ○ Strongly disagree (1)
- ○ Somewhat disagree (2)
- ○ Neither agree nor disagree (3)
- ○ Somewhat agree (4)
- ○ Strongly agree (5)

Q18 I found the desktop experience informative.

- ○ Strongly disagree (1)
- ○ Somewhat disagree (2)
- ○ Neither agree nor disagree (3)
- ○ Somewhat agree (4)
- ○ Strongly agree (5)

Q19 I found the desktop experience easy to use.

- ○ Strongly disagree (1)
- ○ Somewhat disagree (2)
- ○ Neither agree nor disagree (3)
- ○ Somewhat agree (4)
- ○ Strongly agree (5)

Q20 I would like to learn more content through a desktop experience like this.

- ○ Strongly disagree (1)
- ○ Somewhat disagree (2)
- ○ Neither agree nor disagree (3)
- ○ Somewhat agree (4)
- ○ Strongly agree (5)

End of Block: Block 4

Start of Block: Block 5

Q21 The goal of the study is to explore whether interacting with Data Structure and Algorithms in VR is more or less effective than via a desktop interface. Do you have any thoughts on your experience you wish to share?

_____

Q22 What'd you like best about the experience?

_____

Q23 Was there anything that could be improved?

_____

End of Block: Block 5

APPENDIX C - FULL SOURCE CODE

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using System.Linq;
    4: using UnityEngine;
    5:
    6: /*
    7:  * Full: Check if the binary tree is full.
    8:  * Complete: Check if the binary tree is complete.
    9:  * Perfect: Check if the binary tree is perfect.
   10:  *
   11:  * Reverse: Reverse this linked list.
   12:  * Shift: Shift this linked list by the target value.
   13:  *
   14:  * DFS: Search for the target value by depth-first search.
   15:  * BFS: Search for the target value by breadth-first search.
   16:  */
   17:
   18:
   19: /// <summary>
   20: /// The Algorithms class will be split partially across five files. Algorithms.cs wi
ll contain general algorithm functionality, and each other file will be for one respective
type of node:
   21: /// <list type="bullet">
   22: /// <item>
   23: /// BinaryTreeAlgorithms.cs: Full, Complete, Perfect, Branch Sums
   24: /// </item>
   25: /// <item>
   26: /// LinkedListAlgorithms.cs: Reversal, Shift, ToBinaryTree
   27: /// </item>
   28: /// <item>
   29: /// DigraphAlgorithms.cs: DFS, BFS
   30: /// </item>
   31: /// <item>
   32: /// UndigraphAlgorithms.cs: DFS, BFS
   33: /// </item>
   34: /// </list>
   35: /// </summary>
   36: public partial class Algorithms : MonoBehaviour
   37: {
   38:     //Initialization
   39:     private static GameObject selectionRef;
   40:
   41:     public static void Init()
   42:     {
   43:         selectionRef = Resources.Load("Other/Selection") as GameObject;
   44:         waitTime = 1;
   45:     }
   46:
   47:     //Iteration + Iteration Modes
   48:     public enum IterationMode { Input = 0, Wait = 1, Instantaneous = 2 };
   49:     public static IterationMode iterationMode;
   50:     public static bool vr;
   51:     public static float waitTime;
   52:     public static bool inAlgo;
   53:
   54:     public static IEnumerator WaitForKeyDown(KeyCode keyCode)
   55:     {
   56:         while (!Input.GetKeyDown(keyCode))
   57:             yield return null;
   58:         yield return new WaitForEndOfFrame();
   59:         yield return new WaitForEndOfFrame();
   60:         yield return new WaitForEndOfFrame();
   61:     }
   62:
   63:     public static IEnumerator WaitForKeyUp(KeyCode keyCode)
   64:     {
   65:         while (!Input.GetKeyUp(keyCode))
   66:             yield return null;
```

```
67:         yield return new WaitForEndOfFrame();
68:         yield return new WaitForEndOfFrame();
69:         yield return new WaitForEndOfFrame();
70:     }
71:
72:     public static IEnumerator WaitForOVRInput(OVRInput.Button button)
73:     {
74:         while (!OVRInput.GetDown(button))
75:             yield return null;
76:         yield return new WaitForEndOfFrame();
77:         yield return new WaitForEndOfFrame();
78:         yield return new WaitForEndOfFrame();
79:     }
80:
81:     public static IEnumerator WaitSeconds(float s)
82:     {
83:         yield return new WaitForSeconds(s);
84:     }
85:
86:     public static IEnumerator WaitForIteration()
87:     {
88:         switch (iterationMode)
89:         {
90:             case IterationMode.Input:
91:                 if (vr)
92:                     yield return WaitForOVRInput(OVRInput.Button.One);
93:                 else
94:                     yield return WaitForKeyDown(KeyCode.Space);
95:                 break;
96:             case IterationMode.Wait:
97:                 yield return WaitSeconds(waitTime);
98:                 break;
99:             case IterationMode.Instantaneous:
100:                yield return null;
101:                break;
102:            default:
103:                yield return null;
104:                break;
105:        }
106:    }
107:
108:    public static void ChangeIterationMode(bool increment)
109:    {
110:        iterationMode += (increment) ? 1 : -1;
111:
112:        if ((int)iterationMode > 2)
113:            iterationMode = (IterationMode)0;
114:        if ((int)iterationMode < 0)
115:            iterationMode = (IterationMode)2;
116:
117:        //update on ui
118:    }
119:
120:    //Selection
121:    public static GameObject Select(GameObject g)
122:    {
123:        //Create new selection
124:        GameObject newSel = Instantiate(selectionRef);
125:
126:        //Parent & Position it at the gameobject
127:        newSel.transform.SetParent(g.transform);
128:        newSel.transform.localPosition = Vector3.zero;
129:
130:        //Return reference to it
131:        return newSel;
132:    }
133:
134:    public static GameObject Select(Node n)
```

```
  135:     {
  136:         return Select(n.gameObject);
  137:     }
  138:
  139:     public static void Deselect(GameObject g)
  140:     {
  141:         if(g.GetComponentInChildren<Select>())
  142:             Destroy(g.GetComponentInChildren<Select>().gameObject);
  143:     }
  144:
  145:     public static void Deselect(Node n)
  146:     {
  147:         Deselect(n.gameObject);
  148:     }
  149:
  150:     public static void DeselectAll(GameObject g)
  151:     {
  152:         foreach (Select s in g.GetComponentsInChildren<Select>())
  153:             Destroy(s.gameObject);
  154:     }
  155:
  156:     //DEBUG
  157:     public void TestSelectionRecursion(BinaryTree root)
  158:     {
  159:         StartCoroutine(SelectionRecursion(root));
  160:     }
  161:
  162:     private IEnumerator SelectionRecursion(BinaryTree root)
  163:     {
  164:         if (!root)
  165:             yield break;
  166:
  167:         Select(root.gameObject);
  168:
  169:         yield return StartCoroutine(WaitForIteration());
  170:
  171:         Deselect(root.gameObject);
  172:
  173:         StartCoroutine(SelectionRecursion(root.left));
  174:         StartCoroutine(SelectionRecursion(root.right));
  175:         yield return StartCoroutine(WaitForIteration());
  176:     }
  177:
  178:     //To Run from UIManager:
  179:     //
  180:     public IEnumerator RunAlgorithm(UIManager.NodeType type, int index, Structure st
r, int value)
  181:     {
  182:
  183:         //Debug.Log($"Run call in algo with {type.ToString()}, i {index}");
  184:         if (!str)
  185:             Debug.LogWarning($"struct null");
  186:
  187:         //check if already in algo
  188:
  189:         Debug.Log($"Algorithm iteration started");
  190:         inAlgo = true;
  191:
  192:         //nested switch for type, index
  193:         switch(type)
  194:         {
  195:             case UIManager.NodeType.BinaryTree:
  196:                 switch(index)
  197:                 {
  198:                     case 0:
  199:                         yield return StartCoroutine(Full((BinaryTreeStructure)str));
  200:                         break;
  201:                     case 1:
```

```
  202:                         yield return StartCoroutine(Complete((BinaryTreeStructure)st
r));
  203:                         break;
  204:                     case 2:
  205:                         yield return StartCoroutine(Perfect((BinaryTreeStructure)str
));
  206:                         break;
  207:
  208:                 }
  209:                 break;
  210:             case UIManager.NodeType.LinkedList:
  211:                 switch (index)
  212:                 {
  213:                     case 0:
  214:                         yield return StartCoroutine(Reverse((LinkedListStructure)str
));
  215:                         break;
  216:                     case 1:
  217:                         yield return StartCoroutine(Shift((LinkedListStructure)str,
value));
  218:                         break;
  219:                 }
  220:                 break;
  221:             case UIManager.NodeType.Digraph:
  222:                 switch (index)
  223:                 {
  224:                     case 0:
  225:                         yield return StartCoroutine(DFS((DigraphStructure)str, value
));
  226:                         break;
  227:                     case 1:
  228:                         yield return StartCoroutine(BFS((DigraphStructure)str, value
));
  229:                         break;
  230:                 }
  231:                 break;
  232:             case UIManager.NodeType.Undigraph:
  233:                 switch (index)
  234:                 {
  235:                     case 0:
  236:                         yield return StartCoroutine(DFS((UndigraphStructure)str, val
ue));
  237:                         break;
  238:                     case 1:
  239:                         yield return StartCoroutine(BFS((UndigraphStructure)str, val
ue));
  240:                         break;
  241:                 }
  242:                 break;
  243:
  244:         }
  245:         Debug.Log($"Algorithm iteration ended");
  246:         inAlgo = false;
  247:         yield break;
  248:     }
  249:
  250:
  251:
  252:     //Undigraph
  253:
  254:     //dfs
  255:
  256:     //bfs
  257:
  258: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using System.Linq;
    4: using UnityEngine;
    5:
    6: public partial class Algorithms
    7: {
    8:     //Binary Tree
    9:
   10:     //Full
   11:     public IEnumerator Full(BinaryTreeStructure bts)
   12:     {
   13:         //run the check if the tree is actually full
   14:         bool full = realFullCheck(bts.root);
   15:         Debug.Log($"Check start: {full}");
   16:
   17:         //recurse to select
   18:         yield return StartCoroutine(VisualFull(bts.root));
   19:
   20:         Debug.Log($"Check completion!");
   21:
   22:         //wait for last input
   23:         yield return StartCoroutine(WaitForIteration());
   24:
   25:         //deselect everything
   26:         foreach (Select s in bts.GetComponentsInChildren<Select>())
   27:             Destroy(s.gameObject);
   28:
   29:     }
   30:
   31:     //visually present whether this is full or not - mirroring the real check. inste
ad of returning or false, just yield break.
   32:
   33:     private IEnumerator VisualFull(BinaryTree root)
   34:     {
   35:
   36:         //if it's null, end here
   37:         if (!root)
   38:             yield break;
   39:
   40:         //select
   41:         Select(root.gameObject);
   42:
   43:         //wait for input
   44:         yield return StartCoroutine(WaitForIteration());
   45:
   46:         //deselect and mirror real check
   47:         Deselect(root.gameObject);
   48:
   49:         if (root.left ^ root.right)
   50:         {
   51:             Select(root.gameObject).GetComponent<Select>().SetRight($"As this node o
nly has one child, the tree is not full.");
   52:             yield break;
   53:         }
   54:
   55:         //each level needs to happen in parallel
   56:         StartCoroutine(VisualFull(root.left));
   57:         StartCoroutine(VisualFull(root.right));
   58:         yield return StartCoroutine(WaitForIteration());
   59:
   60:         yield break;
   61:     }
   62:
   63:     //a binary tree is full if all nodes have zero or two children, or if it's null,
 therefore: check if it's null, if it's children are both null, or if its left or right are
 null
   64:     public bool realFullCheck(BinaryTree root)
```

```
   65:     {
   66:         //any null tree is full
   67:         if (!root)
   68:             return true;
   69:         //otherwise, check that it has zero or two children: if it has one child, le
ft xor right will be one, so return false
   70:         if (root.left ^ root.right)
   71:             return false;
   72:         //if it has zero or two children, check them.
   73:         if (realFullCheck(root.left) && realFullCheck(root.right))
   74:             return true;
   75:         //if not, return false
   76:         return false;
   77:     }
   78:
   79:     //complete
   80:
   81:     public IEnumerator Complete(BinaryTreeStructure bts)
   82:     {
   83:         //run the real check
   84:         BinaryTree root = bts.root;
   85:         bool complete = realCompleteCheck(root);
   86:
   87:         //mirror the real check, placing a marker on the current node, and also the
place where the property is broken first
   88:         //q for level order traversal
   89:         Queue<BinaryTree> q = new Queue<BinaryTree>();
   90:         q.Enqueue(root);
   91:
   92:         //bool for first enc of invalid
   93:         bool enc = false;
   94:
   95:         //selection components for scope
   96:         Select sel = null;
   97:         Select invalidSel = null;
   98:
   99:         //while the queue is valid
  100:         while (q.Count > 0)
  101:         {
  102:             BinaryTree next = q.Dequeue();
  103:
  104:             Select(next.gameObject);
  105:             Debug.Log($"Selected {next.Value}");
  106:
  107:             yield return StartCoroutine(WaitForIteration());
  108:
  109:             //deselect before iteration
  110:             Deselect(next.gameObject);
  111:
  112:             //if incomplete
  113:             if (enc && (next.left || next.right))
  114:             {
  115:                 //display breakage
  116:                 sel = Select(next.gameObject).GetComponent<Select>();
  117:                 sel.SetRight("Since there has already been a non-full node reached a
nd this node is not full, this tree is not complete.");
  118:
  119:                 //last iter
  120:                 yield return StartCoroutine(WaitForIteration());
  121:
  122:                 Deselect(next.gameObject);
  123:
  124:                 if (invalidSel)
  125:                     Deselect(invalidSel.transform.parent.gameObject);
  126:
  127:                 //return false
  128:                 yield break;
  129:             }
```

```
  130:
  131:             //if any node has only a right child, the tree is invalid
  132:             if (!next.left && next.right)
  133:             {
  134:                 //display breakage
  135:                 sel = Select(next.gameObject).GetComponent<Select>();
  136:                 sel.SetRight("Since this node has a right child, but no left child,
this tree is not complete.");
  137:
  138:                 //last iter
  139:                 yield return StartCoroutine(WaitForIteration());
  140:
  141:                 Deselect(next.gameObject);
  142:
  143:                 //return false
  144:                 yield break;
  145:             }
  146:
  147:             //check left child
  148:             if (next.left)
  149:                 q.Enqueue(next.left);
  150:             else
  151:             {
  152:                 enc = true;
  153:                 if (!invalidSel)
  154:                 {
  155:                     invalidSel = Select(next.gameObject).GetComponent<Select>();
  156:                     invalidSel.SetRight("This node does not have two children, so an
y future non-full node will make this tree incomplete.");
  157:                 }
  158:             }
  159:             //check right child
  160:             if (next.right)
  161:                 q.Enqueue(next.right);
  162:             else
  163:             {
  164:                 enc = true;
  165:                 if (!invalidSel)
  166:                 {
  167:                     invalidSel = Select(next.gameObject).GetComponent<Select>();
  168:                     invalidSel.SetRight("This node does not have two children, so an
y future non-full node will make this tree incomplete.");
  169:                 }
  170:                 //add an additional selection so that it sticks around
  171:                 //Select(next.gameObject);
  172:             }
  173:
  174:         }
  175:
  176:         //if the completeness property is not invalidated by the end of the traversa
l, then the tree is complete
  177:
  178:         //display valid
  179:         sel = Select(root.gameObject).GetComponent<Select>();
  180:         if (enc)
  181:             sel.SetRight("Since no non-leaf node was found after finding a not-full
node, the tree is complete.");
  182:         else
  183:             sel.SetRight("Since a non-full or non-leaf node was never found, the tre
e is complete.");
  184:
  185:         //last iter
  186:         yield return StartCoroutine(WaitForIteration());
  187:
  188:         //cleanup; end
  189:         Deselect(root.gameObject);
  190:         yield break;
  191:     }
```

```
  192:
  193:     public bool realCompleteCheck(BinaryTree root)
  194:     {
  195:         //Iterative solution works well here by level-order traversal.
  196:         //Completeness means that the tree is full until the last level, where all n
odes are aligned to the left
  197:         //The second occurence of a non-full node invalidates the completeness prope
rty, as a complete tree will invalidate that at maximum once, where the last node is left a
ligned without a right child.
  198:
  199:         //q for level order traversal
  200:         Queue<BinaryTree> q = new Queue<BinaryTree>();
  201:         q.Enqueue(root);
  202:
  203:         //bool for first enc of a node violating the fullness property
  204:         bool enc = false;
  205:
  206:         //while the queue is valid
  207:         while (q.Count > 0)
  208:         {
  209:             BinaryTree next = q.Dequeue();
  210:
  211:             //if second occurence
  212:             if (enc && (next.left || next.right))
  213:                 return false;
  214:
  215:             //if any node has only a right child, the tree is invalid
  216:             if (!next.left && next.right)
  217:                 return false;
  218:
  219:             //check left child
  220:             if (next.left)
  221:                 q.Enqueue(next.left);
  222:             else
  223:                 enc = true;
  224:
  225:             //check right child
  226:             if (next.right)
  227:                 q.Enqueue(next.right);
  228:             else
  229:                 enc = true;
  230:         }
  231:
  232:         //if the completeness property is not invalidated by the end of the traversa
l, then the tree is complete
  233:         return true;
  234:     }
  235:
  236:     //perfect
  237:     //any perfect tree must have 2^(depth+1) - 1 nodes, so count the depth and the n
umber of nodes, and compare
  238:
  239:     private void depthAndCountRecurse(BinaryTree root, int curDepth, ref int maxDept
h, ref int count)
  240:     {
  241:         //update depth
  242:         maxDepth = Mathf.Max(curDepth, maxDepth);
  243:
  244:         //increment node count
  245:         count++;
  246:
  247:         //recurse on left/right
  248:         if (root.left)
  249:             depthAndCountRecurse(root.left, curDepth + 1, ref maxDepth, ref count);
  250:         if (root.right)
  251:             depthAndCountRecurse(root.right, curDepth + 1, ref maxDepth, ref count);
  252:
  253:     }
```

```
  254:
  255:     public bool realPerfectCheck(BinaryTree root)
  256:     {
  257:         int depth = -1;
  258:         int count = 0;
  259:
  260:         depthAndCountRecurse(root, 0, ref depth, ref count);
  261:
  262:         Debug.Log($"Tree with root {root.Value} has depth {depth} and count {count},
 a perfect tree would have {(int)(Mathf.Pow(2, depth + 1) - 1)} nodes");
  263:
  264:         return count == (int)(Mathf.Pow(2, depth + 1) - 1);
  265:     }
  266:
  267:     //overload with references
  268:     public bool realPerfectCheck(BinaryTree root, ref int depth, ref int count)
  269:     {
  270:         depthAndCountRecurse(root, 0, ref depth, ref count);
  271:
  272:         Debug.Log($"Tree with root {root.Value} has depth {depth} and count {count},
 a perfect tree would have {(int)(Mathf.Pow(2, depth + 1) - 1)} nodes");
  273:
  274:         return count == (int)(Mathf.Pow(2, depth + 1) - 1);
  275:     }
  276:
  277:     //to iterate across the tree leveled, should yield for iteration only one per le
vel
  278:     public IEnumerator VisualLeveledTraverse(BinaryTree root)
  279:     {
  280:         List<BinaryTree> thisLevel = new List<BinaryTree>();
  281:         thisLevel.Add(root);
  282:
  283:         //while there are nodes in the tree
  284:         while (thisLevel.Count > 0)
  285:         {
  286:             //select each node in the current list
  287:             foreach (BinaryTree bt in thisLevel)
  288:                 Select(bt.gameObject);
  289:
  290:             //wait for iteration
  291:             yield return StartCoroutine(WaitForIteration());
  292:
  293:             //deselect all nodes
  294:             foreach (BinaryTree bt in thisLevel)
  295:                 Deselect(bt.gameObject);
  296:
  297:             //make a temp list of the next level
  298:             List<BinaryTree> nextLevel = new List<BinaryTree>();
  299:             foreach (BinaryTree bt in thisLevel)
  300:             {
  301:                 if (bt.left)
  302:                     nextLevel.Add(bt.left);
  303:                 if (bt.right)
  304:                     nextLevel.Add(bt.right);
  305:             }
  306:             //overwrite level list
  307:             thisLevel = nextLevel;
  308:         }
  309:
  310:         yield return new WaitForEndOfFrame();
  311:     }
  312:
  313:     public IEnumerator Perfect(BinaryTreeStructure bst)
  314:     {
  315:         int depth = 0;
  316:         int count = 0;
  317:
  318:         //run real check
```

```
 319:         bool perfect = realPerfectCheck(bst.root, ref depth, ref count);
 320:
 321:         //visually recurse across the tree
 322:         //yield return StartCoroutine(VisualLeveledTraverse(bst.root));
 323:         yield return StartCoroutine(SelectionRecursion(bst.root));
 324:
 325:         //one last iteration before display
 326:         yield return StartCoroutine(WaitForIteration());
 327:
 328:         //display result
 329:         Select s = Select(bst.root.gameObject).GetComponent<Select>();
 330:         s.SetLeft($"Since this node has a depth of {depth}, it should have 2^(d+1)-1
nodes, or {(int)(Mathf.Pow(2, depth + 1) - 1)} nodes");
 331:         s.SetRight($"Since this tree has {count} nodes, it {((perfect) ? "is" : " is
not")} a perfect tree.");
 332:
 333:         //last input
 334:         yield return StartCoroutine(WaitForIteration());
 335:
 336:         Deselect(s.transform.parent.gameObject);
 337:
 338:         yield break;
 339:     }
 340:
 341:     //branch sums
 342: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using System.Linq;
    4: using UnityEngine;
    5:
    6: public partial class Algorithms
    7: {
    8:     //bfs
    9:     //Breadth First Search across the graph for a certain value
   10:     public IEnumerator BFS(DigraphStructure dgs, int target)
   11:     {
   12:         //maintain list of all accessible nodes, a queue of lists for depth sets, an
d a list for the current depth.
   13:         List<Digraph> allNodes = new List<Digraph>();
   14:         Queue<List<Digraph>> depthSets = new Queue<List<Digraph>>();
   15:         List<Digraph> currentDepth = new List<Digraph>();
   16:
   17:         //first set should be depth set 0, with only the center element
   18:         currentDepth.Add(dgs.center);
   19:
   20:         while (currentDepth.Count != 0)
   21:         {
   22:             //enqueue a deep copy of the current depth list
   23:             depthSets.Enqueue(new List<Digraph>(currentDepth));
   24:
   25:             //generate next depth class from current
   26:             List<Digraph> nextDepth = new List<Digraph>();
   27:             foreach (Digraph dg in currentDepth)
   28:             {
   29:                 //Check if this node has the target value
   30:                 if(dg.Value == target)
   31:                 {
   32:                     Debug.Log($"found value {target}");
   33:
   34:                     //end the iteration
   35:                     Select(dg);
   36:                     dg.GetComponentInChildren<Select>().SetLeft($"Found target value
 {target}!");
   37:                     //wait for iteration
   38:                     yield return StartCoroutine(WaitForIteration());
   39:                     //deselect all
   40:                     foreach (Digraph g in allNodes)
   41:                         Deselect(g);
   42:                     Deselect(dgs.center); // not included in all nodes by choice, to
 find shortest non-trivial path to self.
   43:                     yield break;
   44:
   45:                 }
   46:
   47:                 foreach (Link l in dg.connectionsTo)
   48:                 {
   49:                     Digraph dest = (Digraph)l.destination;
   50:
   51:                     //if this is already a visited node, don't do anything
   52:                     if (!allNodes.Contains(dest))
   53:                     {
   54:                         allNodes.Add(dest);
   55:                         nextDepth.Add(dest);
   56:                     }
   57:                 }
   58:
   59:                 Select(dg);
   60:                 yield return StartCoroutine(WaitForIteration());
   61:             }
   62:             currentDepth = nextDepth;
   63:         }
   64:
   65:         //if this is reached, the target value is not contained in the graph
```

```
   66:         dgs.center.GetComponentInChildren<Select>().SetLeft($"Target value {target}
not found in the graph...");
   67:
   68:         yield return StartCoroutine(WaitForIteration());
   69:
   70:         foreach (Digraph g in allNodes)
   71:             Deselect(g);
   72:         Deselect(dgs.center);
   73:
   74:         yield return null;
   75:     }
   76:
   77:     //dfs
   78:     public IEnumerator DFS(DigraphStructure dgs, int target)
   79:     {
   80:         //Maintain list of visited nodes, and stack of nodes
   81:         List<Digraph> allNodes = new List<Digraph>();
   82:         Stack<Digraph> stack = new Stack<Digraph>();
   83:         stack.Push(dgs.center);
   84:
   85:         //iterative traversal
   86:         while(stack.Count > 0)
   87:         {
   88:             Digraph d = stack.Pop();
   89:
   90:             //if a node is already visited, move to the next one
   91:             if (allNodes.Contains(d))
   92:                 continue;
   93:
   94:             Select(d);
   95:             allNodes.Add(d);
   96:             yield return StartCoroutine(WaitForIteration());
   97:
   98:             //check equality
   99:             if(d.Value == target)
  100:             {
  101:                 d.GetComponentInChildren<Select>().SetLeft($"Target value {target} f
ound!");
  102:                 yield return StartCoroutine(WaitForIteration());
  103:                 foreach (Digraph dg in allNodes)
  104:                     Deselect(dg);
  105:                 yield break;
  106:             }
  107:
  108:             //continue traversal
  109:             foreach (Link l in d.connectionsTo)
  110:                 if (!allNodes.Contains((Digraph)l.destination))
  111:                     stack.Push((Digraph)l.destination);
  112:         }
  113:
  114:         //if this is reached, the target value is not contained in the graph
  115:         dgs.center.GetComponentInChildren<Select>().SetLeft($"Target value {target}
not found in the graph...");
  116:
  117:         yield return StartCoroutine(WaitForIteration());
  118:
  119:         foreach (Digraph g in allNodes)
  120:             Deselect(g);
  121:         Deselect(dgs.center);
  122:
  123:         yield return null;
  124:     }
  125:
  126:
  127:
  128:     //scc
  129: }
```

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public partial class Algorithms
{
    //Linked List

    //reversal
    public IEnumerator Reverse(LinkedListStructure lls)
    {

        //Initial setup
        LinkedList prev, cur, next;

        prev = lls.head;
        cur = prev.next;
        next = null;

        //First iteration
        Select(prev);
        Select(cur);
        Link.Disconnect(prev.nextLink);

        yield return StartCoroutine(WaitForIteration());

        while (cur.next != null)
        {
            //1. store next
            next = cur.next;
            //2 Make current point to previous
            Link.Connect(cur, prev);
            //3 Deselect previous before overwriting it
            Deselect(prev);
            //4. Update References make previous = current;
            prev = cur;
            cur = next;
            //5 select next node;
            Select(cur);
            //6. Wait for iteration
            yield return StartCoroutine(WaitForIteration());
        }

        //Last Iteration
        Deselect(prev);
        Link.Connect(cur, prev);

        //display
        Deselect(cur);
        Select s = Select(cur.gameObject).GetComponent<Select>();
        s.SetRight($"This linked list is now reversed.");

        //update LLS referencess
        LinkedList temp = lls.head;
        lls.head = lls.tail;
        lls.tail = temp;

        yield return StartCoroutine(WaitForIteration());

        Deselect(cur);

    }

    //shift
    public IEnumerator Shift(LinkedListStructure lls, int shift)
    {
        //To shift a linked list, need keep track of the head, tail, and nodes numbe
```

```
r of shift away from both ends
   69:         Select s = Select(lls.head).GetComponent<Select>();
   70:         s.SetLeft($"To shift a linked list by {shift}, need keep track of the head,
tail, and nodes number of {shift} away from both ends");
   71:         yield return StartCoroutine(WaitForIteration());
   72:         Deselect(lls.head);
   73:
   74:         //Count length of the list
   75:         int length = 1;
   76:         LinkedList current = lls.head;
   77:         List<int> values = new List<int>(); //take in all values for graphic impleme
ntation
   78:         while (current != null)
   79:         {
   80:             Select(current);
   81:             //outputText.text = $"Current length = {length}";
   82:             values.Add(current.Value);
   83:             yield return StartCoroutine(WaitForIteration());
   84:             length++;
   85:             Deselect(current);
   86:             current = current.next;
   87:         }
   88:
   89:         //Now iterate across the list again, with the true number of shifts and the
length known
   90:         shift %= length;
   91:         if (shift == 0)
   92:         {
   93:             //outputText.text = $"Shifting by some multiple of the length of the lis
t, therefore the original list is preserved.";
   94:             s = Select(lls.head).GetComponent<Select>();
   95:             s.SetLeft($"Shifting by some multiple of the length of the list, therefo
re the original list is preserved.");
   96:             yield return StartCoroutine(WaitForIteration());
   97:             Deselect(lls.head);
   98:             yield break;
   99:         }
  100:
  101:         //outputText.text = $"Iterating, keeping track of the nodes {Mathf.Abs(shift
)} from head/tail.";
  102:         yield return StartCoroutine(WaitForIteration());
  103:
  104:         LinkedList kForward = null, kBack = null, end = null;
  105:         current = lls.head;
  106:         int index = 1;
  107:         while (current.next != null)
  108:         {
  109:             Select(current);
  110:             if (index == Mathf.Abs(shift))
  111:             {
  112:                 kBack = lls.head;
  113:                 kForward = current;
  114:             }
  115:             if (index > Mathf.Abs(shift))
  116:                 kBack = kBack.next;
  117:             index++;
  118:             yield return StartCoroutine(WaitForIteration());
  119:             Deselect(current);
  120:             current = current.next;
  121:         }
  122:         end = current;
  123:
  124:         if (shift > 0)
  125:         {
  126:             //outputText.text = $"The new head will be {kBack.next.Value}, and the n
ew tail will be {kBack.Value}";
  127:             Select(kBack);
  128:             Select(kBack.next).GetComponent<Select>().SetLeft($"The new head will be
```

```
 {kBack.next.Value}, and the new tail will be {kBack.Value}");
  129:             yield return StartCoroutine(WaitForIteration());
  130:
  131:             //though it is theoretically efficient to shift in place, since unity is
 being used, it's more efficient to change the node values in place, rather than to actuall
y move GameObjects
  132:             for (int i = 0; i < shift; i++)
  133:             {
  134:                 //get last element, bring it to the front
  135:                 print($"Moving with c: {values.Count}, first = {values[0]}, and last
  = {values[values.Count - 1]}, full = {values.ToString()}");
  136:                 int item = values[values.Count - 1];
  137:                 values.RemoveAt(values.Count - 1);
  138:                 values.Insert(0, item);
  139:             }
  140:             current = lls.head;
  141:             index = 0;
  142:             while (current != null)
  143:             {
  144:                 current.Value = values[index];
  145:                 current = (LinkedList)current.next;
  146:                 index++;
  147:             }
  148:             yield return StartCoroutine(WaitForIteration());
  149:             //outputText.text = $"Linked list shifted by {shift}";
  150:             //kBack.GetComponentInChildren<Select>().SetRight($"Linked list shifted
by {shift}");
  151:             Deselect(kBack.next);
  152:             Deselect(kBack);
  153:             Select(lls.head).GetComponent<Select>().SetRight($"Linked list shifted b
y {shift}");
  154:             yield return StartCoroutine(WaitForIteration());
  155:             Deselect(lls.head);
  156:
  157:         }
  158:         else
  159:         {
  160:
  161:             Select(kForward);
  162:             //this is pretty dank
  163:             Select(kForward.next).GetComponent<Select>().SetLeft($"The new head will
 be {kForward.next.Value}, and the new tail will be {kForward.Value}");
  164:             yield return StartCoroutine(WaitForIteration());
  165:             //though it is theoretically efficient to shift in place, since unity is
 being used, it's more efficient to change the node values in place, rather than to actuall
y move GameObjects
  166:             for (int i = 0; i < Mathf.Abs(shift); i++)
  167:             {
  168:                 //get first element, bring it to the end
  169:                 int item = values[0];
  170:                 values.RemoveAt(0);
  171:                 values.Add(item);
  172:             }
  173:             current = lls.head;
  174:             index = 0;
  175:             while (current != null)
  176:             {
  177:                 current.Value = values[index];
  178:                 current = (LinkedList)current.next;
  179:                 index++;
  180:             }
  181:             yield return StartCoroutine(WaitForIteration());
  182:             Deselect(kForward.next);
  183:             Deselect(kForward);
  184:             Select(lls.head).GetComponent<Select>().SetRight($"Linked list shifted b
y {shift}");
  185:             yield return StartCoroutine(WaitForIteration());
  186:             Deselect(lls.head);
```

```
187:         }
188:
189:         //clear input and selections
190:         yield return StartCoroutine(WaitForIteration());
191:
192:         yield return null;
193:     }
194:
195:     //
196:     public IEnumerator VisualReverse(LinkedListStructure lls)
197:     {
198:
199:         //Initial setup
200:         LinkedList prev, cur, next;
201:
202:         prev = lls.head;
203:         cur = prev.next;
204:         next = null;
205:
206:         //First iteration
207:         Select(prev);
208:         Select(cur);
209:         Link.Disconnect(prev.nextLink);
210:
211:         yield return StartCoroutine(WaitForIteration());
212:
213:         while (cur.next != null)
214:         {
215:             //1. store next
216:             next = cur.next;
217:             //2 Make current point to previous
218:             Link.Connect(cur, prev);
219:             //3 Deselect previous before overwriting it
220:             Deselect(prev);
221:             //4. Update References make previous = current;
222:             prev = cur;
223:             cur = next;
224:             //5 select next node;
225:             Select(cur);
226:             //6. Wait for iteration
227:             yield return StartCoroutine(WaitForIteration());
228:         }
229:
230:         //Last Iteration
231:         Deselect(prev);
232:         Link.Connect(cur, prev);
233:
234:         //display
235:         Deselect(cur);
236:
237:         //update LLS referencess
238:         LinkedList temp = lls.head;
239:         lls.head = lls.tail;
240:         lls.tail = temp;
241:
242:         yield return StartCoroutine(WaitForIteration());
243:
244:         Deselect(cur);
245:
246:     }
247:
248:     //cycle (?)
249:
250:     //to BT
251: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public partial class Algorithms
    6: {
    7:     public IEnumerator BFS(UndigraphStructure ugs, int target)
    8:     {
    9:         //maintain list of all accessible nodes, a queue of lists for depth sets, an
d a list for the current depth.
   10:         List<Undigraph> allNodes = new List<Undigraph>();
   11:         Queue<List<Undigraph>> depthSets = new Queue<List<Undigraph>>();
   12:         List<Undigraph> currentDepth = new List<Undigraph>();
   13:
   14:         //first set should be depth set 0, with only the center element
   15:         currentDepth.Add(ugs.center);
   16:         allNodes.Add(ugs.center);
   17:
   18:         while (currentDepth.Count != 0)
   19:         {
   20:             //enqueue a deep copy of the current depth list
   21:             depthSets.Enqueue(new List<Undigraph>(currentDepth));
   22:
   23:             //generate next depth class from current
   24:             List<Undigraph> nextDepth = new List<Undigraph>();
   25:             foreach (Undigraph ug in currentDepth)
   26:             {
   27:                 //Check if this node has the target value
   28:                 if (ug.Value == target)
   29:                 {
   30:                     Debug.Log($"found value {target}");
   31:
   32:                     //end the iteration
   33:                     Select(ug);
   34:                     ug.GetComponentInChildren<Select>().SetLeft($"Found target value
 {target}!");
   35:                     //wait for iteration
   36:                     yield return StartCoroutine(WaitForIteration());
   37:                     //deselect all
   38:                     foreach (Undigraph g in allNodes)
   39:                         Deselect(g);
   40:                     yield break;
   41:
   42:                 }
   43:
   44:                 //Look through all reachable nodes
   45:                 List<Link> aggregateConnections = new List<Link>();
   46:                 aggregateConnections.AddRange(ug.connectionsTo);
   47:                 aggregateConnections.AddRange(ug.connectionsFrom);
   48:                 foreach (Link l in aggregateConnections)
   49:                 {
   50:                     //if this is already a visited node, don't do anything
   51:                     Undigraph dest = (Undigraph)l.destination;
   52:                     if (!allNodes.Contains(dest))
   53:                     {
   54:                         allNodes.Add(dest);
   55:                         nextDepth.Add(dest);
   56:                     }
   57:                     dest = (Undigraph)l.source;
   58:                     if (!allNodes.Contains(dest))
   59:                     {
   60:                         allNodes.Add(dest);
   61:                         nextDepth.Add(dest);
   62:                     }
   63:                 }
   64:
   65:                 Select(ug);
   66:                 yield return StartCoroutine(WaitForIteration());
```

```
    67:             }
    68:             currentDepth = nextDepth;
    69:         }
    70:
    71:         //if this is reached, the target value is not contained in the graph
    72:         ugs.center.GetComponentInChildren<Select>().SetLeft($"Target value {target}
not found in the graph...");
    73:
    74:         yield return StartCoroutine(WaitForIteration());
    75:
    76:         foreach (Undigraph g in allNodes)
    77:             Deselect(g);
    78:         Deselect(ugs.center);
    79:
    80:         yield return null;
    81:     }
    82:
    83:     public IEnumerator DFS(UndigraphStructure ugs, int target)
    84:     {
    85:         //Maintain list of visited nodes, and stack of nodes
    86:         List<Undigraph> allNodes = new List<Undigraph>();
    87:         Stack<Undigraph> stack = new Stack<Undigraph>();
    88:         stack.Push(ugs.center);
    89:
    90:         //iterative traversal
    91:         while (stack.Count > 0)
    92:         {
    93:             Undigraph d = stack.Pop();
    94:
    95:             //if a node is already visited, move to the next one
    96:             if (allNodes.Contains(d))
    97:                 continue;
    98:
    99:             Select(d);
   100:             allNodes.Add(d);
   101:             yield return StartCoroutine(WaitForIteration());
   102:
   103:             //check equality
   104:             if (d.Value == target)
   105:             {
   106:                 d.GetComponentInChildren<Select>().SetLeft($"Target value {target} f
ound!");
   107:                 yield return StartCoroutine(WaitForIteration());
   108:                 foreach (Undigraph ug in allNodes)
   109:                     Deselect(ug);
   110:                 yield break;
   111:             }
   112:
   113:             //continue traversal
   114:             foreach (Link l in d.connectionsTo)
   115:                 if (!allNodes.Contains((Undigraph)l.destination))
   116:                     stack.Push((Undigraph)l.destination);
   117:             foreach (Link l in d.connectionsFrom)
   118:
   119:                 if (!allNodes.Contains((Undigraph)l.source))
   120:                     stack.Push((Undigraph)l.source);
   121:         }
   122:
   123:         //if this is reached, the target value is not contained in the graph
   124:         ugs.center.GetComponentInChildren<Select>().SetLeft($"Target value {target}
not found in the graph...");
   125:
   126:         yield return StartCoroutine(WaitForIteration());
   127:
   128:         foreach (Undigraph g in allNodes)
   129:             Deselect(g);
   130:         Deselect(ugs.center);
   131:
```

```
  132:         yield return null;
  133:     }
  134: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class DemoStructureMarker : MonoBehaviour
    6: {
    7:     // Start is called before the first frame update
    8:     void Start()
    9:     {
   10:
   11:     }
   12:
   13:     // Update is called once per frame
   14:     void Update()
   15:     {
   16:
   17:     }
   18: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class Highlightable : MonoBehaviour
    6: {
    7:     public Material normal;
    8:     public Material highlight;
    9:     public bool highlighted;
   10:     private Renderer rend;
   11:
   12:     private void Start()
   13:     {
   14:         rend = GetComponent<Renderer>();
   15:         highlighted = false;
   16:         if (!normal)
   17:             Debug.LogError($"Error: Missing normal material on highlightable object
{gameObject.name}");
   18:         if (!highlight)
   19:             Debug.LogError($"Error: Missing highlight material on highlightable obje
ct {gameObject.name}");
   20:     }
   21:
   22:     public void Highlight()
   23:     {
   24:         rend.material = highlight;
   25:         highlighted = true;
   26:     }
   27:
   28:     public void Unhighlight()
   29:     {
   30:         rend.material = normal;
   31:         highlighted = false;
   32:     }
   33:
   34:     public void Toggle()
   35:     {
   36:         highlighted = !highlighted;
   37:         rend.material = (highlighted) ? highlight : normal;
   38:     }
   39: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class Pickupable : MonoBehaviour
    6: {
    7:     // Start is called before the first frame update
    8:     void Start()
    9:     {
   10:
   11:     }
   12:
   13:     // Update is called once per frame
   14:     void Update()
   15:     {
   16:
   17:     }
   18: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class Raycaster : MonoBehaviour
    6: {
    7:     //Retain position of last cast
    8:     public Vector3 castPosition;
    9:
   10:     //Info of last cast
   11:     public RaycastHit info;
   12:
   13:     //Returns hit object
   14:     public GameObject cast()
   15:     {
   16:         castPosition = transform.position;
   17:
   18:         if(Physics.Raycast(transform.position, transform.forward, out info))
   19:         {
   20:             return info.collider.gameObject;
   21:         }
   22:         return null;
   23:     }
   24:
   25:         //Raycast with offset from forward direction
   26:     public GameObject cast(float f)
   27:     {
   28:         castPosition = transform.position;
   29:
   30:         if (Physics.Raycast(transform.position + transform.forward * f, transform.fo
rward, out info))
   31:         {
   32:             return info.collider.gameObject;
   33:         }
   34:         return null;
   35:     }
   36:
   37:     //Returns hit object, takes direction parameter
   38:     public GameObject cast(Vector3 dir)
   39:     {
   40:         castPosition = transform.position;
   41:
   42:         if (Physics.Raycast(transform.position, dir, out info))
   43:         {
   44:             return info.collider.gameObject;
   45:         }
   46:         return null;
   47:     }
   48: }
```

```
 1: using System.Collections;
 2: using System.Collections.Generic;
 3: using UnityEngine;
 4: using UnityEngine.UI;
 5: using TMPro;
 6:
 7: public class Select : MonoBehaviour
 8: {
 9:     //selection has two images and two text components, on it’s left and right
10:     public Image leftI, rightI;
11:     public TMP_Text left, right;
12:
13:     public void SetLeft(string text)
14:     {
15:         leftI.gameObject.SetActive(true);
16:         left.text = text;
17:     }
18:
19:     public void SetRight(string text)
20:     {
21:         rightI.gameObject.SetActive(true);
22:         right.text = text;
23:     }
24:     public void HideLeft()
25:     {
26:         leftI.gameObject.SetActive(false);
27:         left.text = string.Empty;
28:     }
29:
30:     public void HideRight()
31:     {
32:         rightI.gameObject.SetActive(false);
33:         right.text = string.Empty;
34:     }
35:
36:     public void Clear()
37:     {
38:         HideLeft();
39:         HideRight();
40:     }
41: }
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/// <summary>
/// I'm gorilla the spinna....
/// Organgatangabangin...
/// </summary>
public class Spinner : MonoBehaviour
{
    public static bool sync = true;

    public float speed;
    public bool x, y, z;

    private Vector3 rotation;
    private Vector3 dir;

    private void Start()
    {
        rotation = Vector3.zero;
        dir = new Vector3(x ? 1 : 0, y ? 1 : 0, z ? 1 : 0);
        rotation = dir * speed * Time.deltaTime;
    }

    void Update()
    {
        if (sync)
            transform.localEulerAngles = dir * Time.realtimeSinceStartup * speed;
        else
            transform.Rotate(rotation);
    }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using System;
using TMPro;

public class TempText : MonoBehaviour
{

    public TMP_Text tempText; // text to assign temporarily to

    private float timer; //time for fade in out;
    private float length; //length of current time
    public Color textColor; // text color

    public float fadeInDec, fadeOutDec;
    public int seqIndex;
    private List<(string, int)> seqSet = new List<(string, int)>();

    // Start is called before the first frame update
    void Awake()
    {
        //init
        fadeInDec = fadeOutDec = 0.2f;
        textColor = new Color(0, 0, 0, 1);
    }


    // Update is called once per frame
    void Update()
    {
        timer -= Time.deltaTime;

        if (timer < 0)
        {
            if (seqIndex == seqSet.Count)
            {
                tempText.text = string.Empty;
            }
            else
            {
                setTempText(seqSet[seqIndex].Item1, seqSet[seqIndex].Item2);
                seqIndex++;
            }
        }
    }

    void LateUpdate()
    {
        float percentage = timer / length;
        if (percentage > (1 - fadeInDec))
        {
            textColor.a = ((1 - percentage) / fadeInDec);
        }
        else if (percentage < (1 - fadeInDec) && percentage > fadeOutDec)
        {
            textColor.a = 1;
        }
        else if (percentage < fadeOutDec)
        {
            textColor.a = percentage / fadeOutDec;
        }
        if (timer == 0)
        {
            tempText.text = ""; // delete text;
        }
```

```
   69:         //Debug.Log($"color: {tempText.color}");
   70:         tempText.color = textColor;
   71:     }
   72:
   73:     public void setTempText(string text)
   74:     {
   75:         //Debug.Log($"temp text set: {text} for {time}");
   76:         length = 200000;
   77:         timer = 100000;
   78:         tempText.text = text;
   79:     }
   80:
   81:     public void setTempText(string text, float time)
   82:     {
   83:         //Debug.Log($"temp text set: {text} for {time}");
   84:         length = time;
   85:         timer = time;
   86:         tempText.text = text;
   87:     }
   88:
   89:     public void interruptEnd()
   90:     {
   91:         seqSet.Clear();
   92:         seqIndex = 0;
   93:         timer = -1;
   94:         tempText.text = string.Empty;
   95:     }
   96:
   97:     public void sequencer(string sequence, char split)
   98:     {
   99:         seqSet.Clear();
  100:
  101:         string[] parts = sequence.Split(split);
  102:         for (int i = 0; i < parts.Length; i += 2)
  103:         {
  104:             seqSet.Add((parts[i], int.Parse(parts[i + 1])));
  105:         }
  106:
  107:         seqIndex = 0;
  108:     }
  109:
  110:     public void assignTempText(TMP_Text t)
  111:     {
  112:         tempText = t;
  113:     }
  114:
  115:     public void assignTextColor(Color c)
  116:     {
  117:         textColor = c;
  118:     }
  119:
  120:     public void Init()
  121:     {
  122:         //init
  123:         fadeInDec = fadeOutDec = 0.2f;
  124:         textColor = new Color(1, 1, 1, 1);
  125:
  126:     }
  127:
  128: }
  129:
  130:
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4: using UnityEngine.Events;
    5:
    6: public class UIAlgo : UIOption
    7: {
    8:     public UnityEvent Run;
    9:     public UIManager.NodeType type;
   10:     public int index;
   11: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4: using UnityEngine.Events;
    5:
    6: public class UIDemo : UIOption
    7: {
    8:     public UnityEvent Create;
    9:     public UIManager.NodeType type;
   10:     public int index;
   11: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4: using UnityEngine.Events;
    5:
    6: public abstract class UIOption : MonoBehaviour
    7: {
    8:     public string description;
    9:
   10:     public string GetDescription()
   11:     {
   12:         return description;
   13:     }
   14: }
```

```
1: using System.Collections;
2: using System.Collections.Generic;
3: using UnityEngine;
4: using UnityEngine.Events;
5:
6: public class UISelect : UIOption
7: {
8:     public UnityEvent Select;
9: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4: using UnityEngine.Events;
    5:
    6: public class UIValue : UIOption
    7: {
    8:     public UnityEvent Increment;
    9:     public UnityEvent Decrement;
   10:     public bool Continuous;
   11: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class BinaryTree : Node
    6: {
    7:     //Members
    8:     public BinaryTree left;
    9:     public Link leftLink;
   10:     public BinaryTree right;
   11:     public Link rightLink;
   12:
   13:     public override void AssignRefSrc(Link l)
   14:     {
   15:         //assign left, then right. if both are full, take location preference, based
 on x coord
   16:
   17:         if(!left)
   18:         {
   19:             leftLink = l;
   20:             left = (BinaryTree)l.destination;
   21:
   22:         }
   23:         else if(!right)
   24:         {
   25:             rightLink = l;
   26:             right = (BinaryTree)l.destination;
   27:         }
   28:         else
   29:         {
   30:             //locatory
   31:             if(l.destination.transform.position.x <= transform.position.x)
   32:             {
   33:                 //Delete existing link
   34:                 Link.Disconnect(leftLink); //Destroy(leftLink.gameObject);
   35:                 leftLink = l;
   36:                 left = (BinaryTree)l.destination;
   37:             }
   38:             else
   39:             {
   40:                 //Delete existing link
   41:                 Link.Disconnect(rightLink);
   42:                 rightLink = l;
   43:                 right = (BinaryTree)l.destination;
   44:             }
   45:         }
   46:     }
   47:
   48:     public override void AssignRefDest(Link l)
   49:     {
   50:         if (prevLink)
   51:         {
   52:             Link.Disconnect(prevLink);
   53:         }
   54:
   55:         prevLink = l;
   56:         prev = l.source;
   57:     }
   58:     public override void ClearRefSrc(Link l)
   59:     {
   60:         if(l == leftLink)
   61:         {
   62:             leftLink = null;
   63:             left = null;
   64:         }
   65:         else if(l == rightLink)
   66:         {
   67:             rightLink = null;
```

```
   68:             right = null;
   69:         }
   70:         else
   71:         {
   72:             Debug.LogError($"Error: ClearRef called on binary tree, but passed link
is not equal to either the left nor the right link");
   73:         }
   74:     }
   75:
   76:     public override void ClearRefDest(Link l)
   77:     {
   78:         prevLink = null;
   79:         prev = null;
   80:     }
   81:
   82:     public void SetRefSrc(Link l, bool onLeft)
   83:     {
   84:         if (onLeft)
   85:         {
   86:             leftLink = l;
   87:             if (l)
   88:                 left = (BinaryTree)l.destination;
   89:             else
   90:                 left = null;
   91:         }
   92:         else
   93:         {
   94:             rightLink = l;
   95:             if (l)
   96:                 right = (BinaryTree)l.destination;
   97:             else
   98:                 right = null;
   99:         }
  100:     }
  101:
  102:     public override void OnMovement()
  103:     {
  104:         Link.Reconnect(prevLink);
  105:         Link.Reconnect(leftLink);
  106:         Link.Reconnect(rightLink);
  107:     }
  108:
  109:
  110:     public override void Delete()
  111:     {
  112:         Link.Disconnect(prevLink);
  113:         Link.Disconnect(leftLink);
  114:         Link.Disconnect(rightLink);
  115:         Destroy(this.gameObject);
  116:     }
  117:
  118: }
```

```
1: ï»¿using System.Collections;
2: using System.Collections.Generic;
3: using UnityEngine;
4:
5: public class Digraph : GraphNode
6: {
7:
8: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5:
    6: /// <summary>
    7: /// Unlike other Node types, the graph nodes may have any number of connections. Thi
s is an abstract class which Undirected and Directed graph nodes will inherit from.
    8: /// </summary>
    9: public abstract class GraphNode : Node
   10: {
   11:     /// <summary>
   12:     /// Connections from other nodes to this node (this is destination)
   13:     /// </summary>
   14:     public List<Link> connectionsFrom;
   15:
   16:     /// <summary>
   17:     /// Connections to other nodes from this node (this is source)
   18:     /// </summary>
   19:     public List<Link> connectionsTo;
   20:
   21:     public override void AssignRefSrc(Link l)
   22:     {
   23:         connectionsTo.Add(l);
   24:     }
   25:
   26:     public override void AssignRefDest(Link l)
   27:     {
   28:         connectionsFrom.Add(l);
   29:     }
   30:
   31:     public override void ClearRefSrc(Link l)
   32:     {
   33:         //Since the graph nodes use links as their only source of connections, no ot
her references need to be handled.
   34:         return;
   35:     }
   36:
   37:     public override void ClearRefDest(Link l)
   38:     {
   39:         //Since the graph nodes use links as their only source of connections, no ot
her references need to be handled.
   40:         return;
   41:     }
   42:
   43:     public override void OnMovement()
   44:     {
   45:         foreach (Link l in connectionsTo)
   46:             Link.Reconnect(l);
   47:         foreach (Link l in connectionsFrom)
   48:             Link.Reconnect(l);
   49:     }
   50:
   51:     public override void Delete()
   52:     {
   53:         foreach (Link l in connectionsTo)
   54:             Link.Disconnect(l);
   55:         foreach (Link l in connectionsFrom)
   56:             Link.Disconnect(l);
   57:
   58:         Destroy(this.gameObject);
   59:     }
   60: }
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LinkedList : Node
{
    public LinkedList next;
    public Link nextLink;

    public override void AssignRefSrc(Link l)
    {
        //Delete existing link
        if (nextLink)
            Link.Disconnect(nextLink);

        nextLink = l;
        next = (LinkedList)l.destination;
    }

    public override void AssignRefDest(Link l)
    {
        if(prevLink)
        {
            Link.Disconnect(prevLink);
        }

        prevLink = l;
        prev = l.source;
    }

    public override void ClearRefSrc(Link l)
    {
        nextLink = null;
        next = null;
    }

    public override void ClearRefDest(Link l)
    {
        prevLink = null;
        prev = null;
    }



    public override void OnMovement()
    {
        Link.Reconnect(prevLink);
        Link.Reconnect(nextLink);
    }

    public override void Delete()
    {
        Link.Disconnect(prevLink);
        Link.Disconnect(nextLink);
        Destroy(this.gameObject);
    }


}
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using System.Reflection;
    4: using System;
    5: using UnityEngine;
    6: using TMPro;
    7:
    8: ///<summary>
    9: ///The Node class is an abstract class for all nodes of the data structures to inher
it from.
   10: ///The abstract class GraphNode inherits this class, which then handles undirected a
nd directed graphs.
   11: /// </summary>
   12: public abstract class Node : MonoBehaviour
   13: {
   14:     //Creation
   15:     public enum CreationSelection : uint { BinaryTree = 0, LinkedList = 1, Digraph =
 2, Undigraph = 3 };
   16:     public static CreationSelection selection;
   17:     static List<Type> NodeTypes = new List<Type> { typeof(BinaryTree), typeof(Linked
List), typeof(Undigraph), typeof(Digraph) };
   18:     static List<GameObject> CreationReferences;
   19:
   20:     private void Awake()
   21:     {
   22:         //On creation initialize text reference
   23:         valueText = GetComponentInChildren<TMP_Text>();
   24:     }
   25:
   26:     //Reference initialization - called once at game startup
   27:     public static void Init()
   28:     {
   29:         //Initialize references from the resource folder for instantiation
   30:         CreationReferences = new List<GameObject>();
   31:         for (int i = 0; i < 4; i++)
   32:         {
   33:             CreationSelection sel = (CreationSelection)i;
   34:             GameObject reference = Resources.Load($"Nodes/{sel.ToString()}") as Game
Object;
   35:             CreationReferences.Add(reference);
   36:         }
   37:         //Debug.Log($"init done {CreationReferences[0].name}");
   38:     }
   39:
   40:     public static Node Create(int value, CreationSelection selection)
   41:     {
   42:         GameObject obj;
   43:         //Instantiate a node from the references nodes in the resources folder
   44:         obj = Instantiate(CreationReferences[(int)selection]);
   45:         //Set its value to the passed value
   46:         Node ret = obj.GetComponent<Node>();
   47:         ret.Value = value;
   48:         //Set name
   49:         obj.name = $"{selection.ToString()} {value}";
   50:         //Return the new node
   51:         return ret;
   52:     }
   53:
   54:     public static Node Create(int value, int selection)
   55:     {
   56:         return Create(value, (CreationSelection)selection);
   57:     }
   58:
   59:     //Non-static members/properties
   60:     private TMP_Text valueText;
   61:     private int val;
   62:     public int Value
   63:     {
```

```
    64:         get
    65:         {
    66:             return val;
    67:         }
    68:         set
    69:         {
    70:             val = value;
    71:             if (val < 0)
    72:                 val = 999;
    73:             else if (val > 999)
    74:                 val = 0;
    75:             valueText.text = val.ToString();
    76:         }
    77:     }
    78:
    79:
    80:     public Node prev;
    81:     public Link prevLink;
    82:     /// <summary>
    83:     /// Assign references for when this node is the destination of a link.
    84:     /// </summary>
    85:     /// <param name="l">Link that points to this node</param>
    86:     public abstract void AssignRefDest(Link l);
    87:     /// <summary>
    88:     /// Assign references for when this node is the source of a link.
    89:     /// </summary>
    90:     /// <param name="l">Link that this node is the source of</param>
    91:     public abstract void AssignRefSrc(Link l);
    92:
    93:     /// <summary>
    94:     /// Clear references for when this node is the source of a link. Used for discon
nection.
    95:     /// </summary>
    96:     /// <param name="l">Link that this node is the source of</param>
    97:     public abstract void ClearRefSrc(Link l);
    98:
    99:     /// <summary>
   100:     /// Clear references for when this node is the destination of a link. Used for d
isconnection.
   101:     /// </summary>
   102:     /// <param name="l">Link that points to this node</param>
   103:     public abstract void ClearRefDest(Link l);
   104:
   105:
   106:     /// <summary>
   107:     /// When a node is moved, it should re-position all links between it.
   108:     /// </summary>
   109:     public abstract void OnMovement();
   110:
   111:     /// <summary>
   112:     /// Delete this node, and handle all necessary disconnections.
   113:     /// </summary>
   114:     public abstract void Delete();
   115:
   116:
   117:
   118:
   119: }
```

```
1: ï»¿using System.Collections;
2: using System.Collections.Generic;
3: using UnityEngine;
4:
5: public class Undigraph : GraphNode
6: {
7:
8: }
```

```
    1: ï»¿using System;
    2: using System.Reflection;
    3: using System.Collections;
    4: using System.Collections.Generic;
    5: using UnityEngine;
    6: using UnityEngine.UI;
    7: using UnityEngine.SceneManagement;
    8: using TMPro;
    9:
   10: //Why are XML List tags broken?
   11: /// <summary>
   12: /// This class holds all of the actions that the player can do. This class does not
directly interface with input, but instead its methods are called from <see cref="InputBind
ings"/>.
   13: /// Currently, the player can:
   14: /// <list type="bullet">
   15: /// <item>Move (8 directions)</item>
   16: /// <item>Jump (Ascend; Descend?)</item>
   17: /// <item>Toggle Flight</item>
   18: /// <item>Create Node</item>
   19: /// <item>Edit Node</item>
   20: /// <item>Pick Up a Node</item>
   21: /// <item>Move Node In/Out</item>
   22: /// <item>Connect Selected Node to Hovered Node</item>
   23: /// <item>Delete Selected Node</item>
   24: /// <item>Organize all existing nodes into structures</item>
   25: /// <item>Start Algorithm</item>
   26: /// <item>Step Forward</item>
   27: /// <item>Step Back</item>
   28: /// <item>Change Iteration Mode</item>
   29: /// <item>Change Algorithm</item>
   30: /// </list>
   31: /// </summary>
   32: public class Actions : MonoBehaviour
   33: {
   34:     //Player References:
   35:     private GameObject PlayerObject;
   36:     private Collider PlayerCollider;
   37:     private Rigidbody PlayerRigidbody;
   38:     private Camera PlayerCamera;
   39:     private SimpleCapsuleWithStickMovement oculusMovementComponent;
   40:
   41:     private TempText handInfo, screenInfo;
   42:
   43:     [HideInInspector]
   44:     public bool vr;
   45:
   46:     //Raycaster References
   47:     public Raycaster leftHandRaycaster;
   48:     public Raycaster rightHandRaycaster;
   49:     public Raycaster mouseLookRaycaster;
   50:     //Since there are three Raycasters in the scene, they'll be assigned to their re
spective purposes - in KBM, the center camera raycaster will take the left slot, and in VR,
 the hands will take their respective slots.
   51:     private Raycaster pointRaycaster;
   52:     private Raycaster teleportRaycaster;
   53:
   54:     private LineRenderer leftLR;
   55:     private LineRenderer rightLR;
   56:     float tpBuffer;
   57:
   58:     //Assign in inspector:
   59:     public GameObject teleportIndicator;
   60:
   61:     //Creation Station
   62:     public GameObject CreationStation;
   63:     bool csEnabled = true;
   64:
```

```
   65:     //Left Hand Tracking
   66:     GameObject pointedTo;
   67:     Node pointedNode;
   68:     Image pointedImage;
   69:     Highlightable pointedHighlightable;
   70:     Link pointedLink;
   71:
   72:     //Old UI
   73:     //UIOption pointedUIO;
   74:     //DemoStructureOption pointedDSO;
   75:     //AlgorithmOption pointedAO;
   76:
   77:     //New UI
   78:     UIOption pointedUI;
   79:     UIAlgo pointedAlgo;
   80:     UIDemo pointedDemo;
   81:     UISelect pointedSelect;
   82:     UIValue pointedValue;
   83:
   84:     float changeBuffer = -1;
   85:
   86:     //Keyboard Mouse Tracking:
   87:     public Image crosshair;
   88:     public TMP_Text screenText;
   89:
   90:     //Specific UI option components for specific incrementation
   91:     public UIOption typeUIO, valueUIO, iterationUIO, createUIO, demoTypeUIO, aValueU
IO;
   92:     public UIManager uiManager;
   93:
   94:     //Selected Members
   95:     Node selectedNode;
   96:     GameObject held;
   97:     Node heldNode;
   98:     private bool movingHeld;
   99:     private bool holdingStructure = false;
  100:
  101:     //Connection
  102:     [HideInInspector]
  103:     public bool connecting;
  104:     Node source, destination;
  105:     [HideInInspector]
  106:     public float connectionBuffer = -1;
  107:
  108:     //
  109:     private float dist;
  110:     public float Distance
  111:     {
  112:         get { return dist; }
  113:         set
  114:         {
  115:             dist = Mathf.Clamp(value, 3, 20);
  116:             MoveHeld();
  117:         }
  118:     }
  119:
  120:     //Though the input to create a node is in InputBindings, the only things that co
me from inputbindings are incrementing the nodecreation selection, or to trigger node creat
ion itself
  121:     //Node Creation is by Delegate methods, assigned at node creation increment / de
crement
  122:     public delegate Node NodeCreationMethod(int value);
  123:     NodeCreationMethod DelCreateNode;
  124:     public enum CreationSelection : uint { BinaryTree = 0, LinkedList = 1, Undiraph
= 2, Digraph = 3};
  125:     public CreationSelection selection;
  126:     List<Type> NodeTypes = new List<Type> { typeof(BinaryTree), typeof(LinkedList),
typeof(Undigraph), typeof(Digraph) };
```

```
  127:
  128:     public void Init()
  129:     {
  130:         //Assign Player References
  131:         AssignPlayerReferences(true);
  132:
  133:         //Assign component references;
  134:         uiManager = GetComponentInChildren<UIManager>();
  135:         oculusMovementComponent = GetComponentInChildren<SimpleCapsuleWithStickMovem
ent>();
  136:         leftLR = leftHandRaycaster.GetComponent<LineRenderer>();
  137:         rightLR = rightHandRaycaster.GetComponent<LineRenderer>();
  138:
  139:         //This is easier to do precisely on startup than in editor
  140:         rightLR.startWidth = 0.01f;
  141:         rightLR.endWidth = 0.05f;
  142:
  143:         leftLR.startWidth = 0.01f;
  144:         leftLR.endWidth = 0.05f;
  145:
  146:         if (GetComponentInParent<InputBindings>().inMainMenu)
  147:             return;
  148:
  149:         CreationStation = FindObjectOfType<UIManager>().gameObject;
  150:
  151:         //lazy but working
  152:         //settingsPanel = uiManager.gameObject.transform.Find("Settings").gameObject
;
  153:         //CloseSettings();
  154:     }
  155:
  156:     public void AssignPlayerReferences(bool vr)
  157:     {
  158:         this.vr = vr;
  159:
  160:         //If VR enabled:
  161:         if(vr)
  162:         {
  163:             PlayerCamera = transform.FindChildRecursive("CenterEyeAnchor").GetCompon
ent<Camera>();
  164:             PlayerObject = transform.FindChildRecursive("PlayerOculus").gameObject;
  165:             PlayerCollider = PlayerObject.GetComponent<Collider>();
  166:             PlayerRigidbody = PlayerObject.GetComponent<Rigidbody>();
  167:             pointRaycaster = leftHandRaycaster;
  168:             teleportRaycaster = rightHandRaycaster;
  169:             teleportIndicator.SetActive(true);
  170:             teleportIndicator.transform.localScale = Vector3.zero;
  171:             handInfo = PlayerObject.GetComponent<TempText>();
  172:         }
  173:
  174:         //If on KBM
  175:         else
  176:         {
  177:             PlayerCamera = Camera.main;
  178:             PlayerObject = PlayerCamera.transform.parent.gameObject;
  179:             PlayerCollider = PlayerObject.GetComponent<Collider>();
  180:             PlayerRigidbody = PlayerObject.GetComponent<Rigidbody>();
  181:             pointRaycaster = mouseLookRaycaster;
  182:             teleportRaycaster = null;
  183:             teleportIndicator.SetActive(false);
  184:             teleportIndicator.transform.localScale = Vector3.zero;
  185:             screenInfo = PlayerObject.GetComponent<TempText>();
  186:         }
  187:     }
  188:
  189:     void Start()
  190:     {
  191:         if (!PlayerObject)
```

```
  192:             Debug.LogError($"ERROR: Player Object is NULL");
  193:
  194:         IncrementNodeSelection(); // to fill in delegate method
  195:         selection = (CreationSelection)0; //reset selection
  196:
  197:         csEnabled = true;
  198:         Distance = 5;
  199:
  200:         MoveCS();
  201:     }
  202:
  203:     private void Update()
  204:     {
  205:         //Buffer for continuous inputs
  206:         tpBuffer -= Time.deltaTime;
  207:         changeBuffer -= Time.deltaTime;
  208:         connectionBuffer -= Time.deltaTime;
  209:
  210:         //If holding a node, constantly align it's position
  211:         if (held)
  212:             MoveHeld();
  213:     }
  214:
  215:     private void LateUpdate()
  216:     {
  217:         //After everything else is done - should decide whether stick can be used fo
r motion or not:
  218:
  219:         //If pointing at a ui element, left stick shouldn't be used for movement.
  220:         int speed = (pointedValue || movingHeld || Tutorial.restrictMovement) ? 0 :
12;
  221:         //Debug.Log($"Speed {speed}");
  222:         SetSpeed(speed);
  223:     }
  224:
  225:     //Actions
  226:
  227:     public void LoadMain()
  228:     {
  229:         SceneManager.LoadScene("Main Menu");
  230:     }
  231:
  232:     //Called from UI Option invoke
  233:     public void Run(int index)
  234:     {
  235:         if(selectedNode)
  236:             uiManager.Run(index, selectedNode);
  237:     }
  238:
  239:     public void CreateDemo(int index)
  240:     {
  241:         uiManager.CreateDemo(index);
  242:     }
  243:
  244:     public Vector3 GetLook()
  245:     {
  246:         return PlayerCamera.transform.forward;
  247:     }
  248:
  249:
  250:     public void LeftHandTracking()
  251:     {
  252:         Vector3[] pos = new Vector3[2];
  253:
  254:         //If holding something, don't assign new references for tracking things, and
 dont interact with ui.
  255:         if (held)
  256:         {
```

```
  257:             pos[0] = pointRaycaster.transform.position;
  258:             if(holdingStructure)
  259:                 pos[1] = heldNode.transform.position;
  260:             else
  261:                 pos[1] = held.transform.position;
  262:             leftLR.SetPositions(pos);
  263:             pointedUI = null;
  264:             return;
  265:         }
  266:
  267:         GameObject hit;
  268:         if (hit = pointRaycaster.cast(3))
  269:         {
  270:             //TempText.setTempText($"{leftRaycaster.info.point.ToString()}", 0.1f);
  271:             //TempText.setTempText($"Hit {hit.name}, drawing line to {hit.transform.
position.ToString()}", 5);
  272:
  273:             //If looking at a new object, reassign references.
  274:             if(hit != pointedTo)
  275:             {
  276:                 pointedTo = hit;
  277:                 pointedImage = hit.GetComponent<Image>();
  278:                 pointedNode = hit.GetComponent<Node>();
  279:                 pointedHighlightable = hit.GetComponent<Highlightable>();
  280:                 //pointedUIO = hit.GetComponent<UIOption>();
  281:                 pointedLink = hit.GetComponentInParent<Link>();
  282:                 //pointedDSO = hit.GetComponent<DemoStructureOption>();
  283:
  284:                 pointedUI = hit.GetComponent<UIOption>();
  285:                 pointedValue = hit.GetComponent<UIValue>();
  286:                 pointedDemo = hit.GetComponent<UIDemo>();
  287:                 pointedSelect = hit.GetComponent<UISelect>();
  288:                 pointedAlgo = hit.GetComponent<UIAlgo>();
  289:             }
  290:
  291:             //If looking at an interactable, draw a line.
  292:             if(pointedUI || pointedHighlightable || pointedLink)
  293:             {
  294:                 //Draw Line
  295:                 pos[0] = pointRaycaster.castPosition;
  296:                 pos[1] = pointRaycaster.info.point;
  297:             }
  298:             else
  299:             {
  300:                 pos[0] = Vector3.zero;
  301:                 pos[1] = Vector3.zero;
  302:             }
  303:
  304:         }
  305:         else
  306:         {
  307:             //If not looking at UI, no need to halt character from movement
  308:
  309:             //null references
  310:             pointedTo = null;
  311:             pointedNode = null;
  312:             pointedUI = null;
  313:             pos[0] = Vector3.zero;
  314:             pos[1] = Vector3.zero;
  315:         }
  316:         leftLR.SetPositions(pos);
  317:     }
  318:
  319:     public void ScreenTracking()
  320:     {
  321:         //This method works like left-hand tracking, except the crosshair color will
 be changed, instead of a line being drawn.
  322:         if(held)
```

```
323:         {
324:             crosshair.color = Color.red;
325:             pointedUI = null;
326:             return;
327:         }
328:
329:         GameObject hit;
330:         if (hit = pointRaycaster.cast(3))
331:         {
332:             //Debug.Log($"hit {hit.name}");
333:
334:             //If looking at a new object, reassign references.
335:             if (hit != pointedTo)
336:             {
337:                 pointedTo = hit;
338:                 pointedImage = hit.GetComponent<Image>();
339:                 pointedNode = hit.GetComponent<Node>();
340:                 pointedHighlightable = hit.GetComponent<Highlightable>();
341:                 pointedLink = hit.GetComponentInParent<Link>();
342:
343:                 //pointedUIO = hit.GetComponent<UIOption>();
344:                 //pointedDSO = hit.GetComponent<DemoStructureOption>();
345:                 //pointedAO = hit.GetComponent<AlgorithmOption>();
346:
347:                 //Debug.Log($"Getting components");
348:                 pointedUI = hit.GetComponent<UIOption>();
349:                 pointedValue = hit.GetComponent<UIValue>();
350:                 pointedDemo = hit.GetComponent<UIDemo>();
351:                 pointedSelect = hit.GetComponent<UISelect>();
352:                 pointedAlgo = hit.GetComponent<UIAlgo>();
353:             }
354:         }
355:         else
356:         {
357:             //If not looking at UI, no need to halt character from movement
358:             //null references
359:             pointedTo = null;
360:             pointedNode = null;
361:             pointedUI = null;
362:             pointedLink = null;
363:             //pointedDSO = null;
364:             //pointedAO = null;
365:             crosshair.color = Color.gray;
366:             return;
367:         }
368:
369:         //update visual colors -- UPDATE FOR NEW UI
370:         if (pointedUI)
371:             crosshair.color = Color.green;
372:         else if (pointedNode)
373:             crosshair.color = Color.blue;
374:         else
375:             crosshair.color = Color.gray;
376:     }
377:
378:     //Menu here:
379:     public void MenuSelect()
380:     {
381:         if (pointedSelect)
382:             pointedSelect.Select.Invoke();
383:         /*
384:         if(pointedUIO == startUIO)
385:         {
386:             SceneManager.LoadScene("Main Scene");
387:         }
388:         if(pointedUIO == shortTutorialUIO)
389:         {
390:             PlayerPrefs.SetInt("Tutorial Length", 0);
```

```
  391:             SceneManager.LoadScene("Tutorial");
  392:         }
  393:         if(pointedUIO == longTutorialUIO)
  394:         {
  395:             PlayerPrefs.SetInt("Tutorial Length", 1);
  396:             SceneManager.LoadScene("Tutorial");
  397:         }
  398:         */
  399:     }
  400:
  401:     public void StartMain()
  402:     {
  403:         SceneManager.LoadScene("Tutorial");
  404:     }
  405:
  406:     public void Start1()
  407:     {
  408:         SceneManager.LoadScene("Binary Tree Tutorial");
  409:     }
  410:
  411:     public void Start2()
  412:     {
  413:         SceneManager.LoadScene("Linked List Tutorial");
  414:     }
  415:
  416:     public void Start3()
  417:     {
  418:         SceneManager.LoadScene("Graphs Tutorial");
  419:     }
  420:
  421:
  422:     public void StartTutorial()
  423:     {
  424:         PlayerPrefs.SetInt("Tutorial Length", 1);
  425:         SceneManager.LoadScene("Tutorial");
  426:
  427:     }
  428:
  429:     public void StartShortTutorial()
  430:     {
  431:         PlayerPrefs.SetInt("Tutorial Length", 0);
  432:         SceneManager.LoadScene("Tutorial");
  433:     }
  434:
  435:     //Info to put in front of the player, either via their left hand, or screenspace
.
  436:     public string GetInfo()
  437:     {
  438:         string info = string.Empty;
  439:
  440:         string changeMechanism = (vr) ? "left thumbstick" : "scroll wheel";
  441:         string choiceMechanism = (vr) ? "trigger" : "mouse button";
  442:
  443:         //Info cases:
  444:         //Selected Node
  445:         if (selectedNode && !holdingStructure)
  446:         {
  447:             info += $"{UIManager.FormattedNodeName(selectedNode)} selected with a va
lue of {selectedNode.Value}\nPicking up this node will move the whole structure if it exist
s.";
  448:         }
  449:         //Holding Node
  450:         if (held)
  451:         {
  452:             if(holdingStructure)
  453:             {
  454:                 info += $"{UIManager.FormattedNodeName(selectedNode)} selected with
a value of {selectedNode.Value}\n Holding the entire structure.";
```

```
  455:             }
  456:             else
  457:             {
  458:                 info += $"{UIManager.FormattedNodeName(heldNode)} held with a value
of {heldNode.Value}\n";
  459:             }
  460:         }
  461:
  462:         //Looking at a UI Option
  463:         if (pointedUI)
  464:         {
  465:             if(pointedUI == typeUIO)
  466:             {
  467:                 info += $"Any created node will be a {UIManager.FormattedNodeName(ui
Manager.type)}. You can go up or down on the {changeMechanism} to change this type.\n";
  468:             }
  469:             else if(pointedUI == valueUIO)
  470:             {
  471:                 info += $"Any created node will have a value of {uiManager.Value}. Y
ou can go up or down on the {changeMechanism} to increase or decrease this value.\n";
  472:             }
  473:             else if (pointedUI == aValueUIO)
  474:             {
  475:                 info += $"Some algorithms require an input value. Any algorithm that
 needs one will use {uiManager.aValue}. You can go up or down on the {changeMechanism} to i
ncrease or decrease this value.\n";
  476:             }
  477:             else if(pointedUI == createUIO)
  478:             {
  479:                 info += $"Press the left {choiceMechanism} to create a {UIManager.Fo
rmattedNodeName(uiManager.type)} with a value of {uiManager.Value}";
  480:             }
  481:             else if(pointedUI == iterationUIO)
  482:             {
  483:                 //add more to this
  484:                 info += $"Algorithms will iterate according to this option. The curr
ent iteration mode is {Algorithms.iterationMode.ToString()}";
  485:             }
  486:             else
  487:             {
  488:                 info += pointedUI.GetDescription();
  489:             }
  490:         }
  491:
  492:         if(pointedLink)
  493:         {
  494:             if(pointedLink.Directed)
  495:                 info += $"This is a connection from {UIManager.FormattedNodeName(poi
ntedLink.source)} [{pointedLink.source.Value}] to {UIManager.FormattedNodeName(pointedLink.
destination)} [{pointedLink.destination.Value}]";
  496:             else
  497:                 info += $"This is a connection between {UIManager.FormattedNodeName(
pointedLink.source)} [{pointedLink.source.Value}] and {UIManager.FormattedNodeName(pointedL
ink.destination)} [{pointedLink.destination.Value}]";
  498:         }
  499:
  500:         return info;
  501:     }
  502:
  503:     public void ConnectNodes()
  504:     {
  505:         //If not pointing at a node, connection should be disabled.
  506:         if (!pointedNode)
  507:             return;
  508:
  509:         //Can be in two states
  510:         //1. Not yet connecting anything
  511:         if(!source)
```

```
  512:         {
  513:             source = pointedNode;
  514:             connecting = true;
  515:             if(!Tutorial.restrictPickupInfo)
  516:                 SetInfo($"Connecting {UIManager.FormattedNodeName(source)} [{source.
Value}] to...");
  517:             return;
  518:         }
  519:
  520:         //2. Connecting to the currently pointed node:
  521:         destination = pointedNode;
  522:         //Check self connection
  523:         if (source == destination)
  524:         {
  525:             connecting = false;
  526:             source = null;
  527:             destination = null;
  528:             if (!Tutorial.restrictPickupInfo)
  529:                 SetInfo($"Cannot connect a node to itself!", 3);
  530:             connectionBuffer = 3;
  531:             return;
  532:         }
  533:
  534:         if(source.GetType() != destination.GetType())
  535:         {
  536:             connecting = false;
  537:             source = null;
  538:             destination = null;
  539:             if (!Tutorial.restrictPickupInfo)
  540:                 SetInfo($"Cannot connect two nodes of different types!", 3);
  541:             connectionBuffer = 3;
  542:             return;
  543:         }
  544:
  545:         Link.Connect(source, destination);
  546:         if (!Tutorial.restrictPickupInfo)
  547:             SetInfo($"Connected {UIManager.FormattedNodeName(source)} [{source.Value
}] to {UIManager.FormattedNodeName(destination)} [{destination.Value}]", 3);
  548:         connecting = false;
  549:         source = null;
  550:         destination = null;
  551:         connectionBuffer = 3;
  552:
  553:     }
  554:
  555:     public void SetInfo(string text)
  556:     {
  557:         if(handInfo)
  558:             handInfo.setTempText(text);
  559:         if(screenInfo)
  560:             screenInfo.setTempText(text);
  561:     }
  562:
  563:     public void SetInfo(string text, float time)
  564:     {
  565:         if (handInfo)
  566:             handInfo.setTempText(text, time);
  567:         if (screenInfo)
  568:             screenInfo.setTempText(text, time);
  569:     }
  570:
  571:     public void ClearInfo()
  572:     {
  573:         if (connecting || connectionBuffer > 0)
  574:             return;
  575:         //Debug.Log($"Clearing");
  576:         SetInfo(string.Empty);
  577:     }
```

```
578:
579:     public void SelectNode()
580:     {
581:         //if not looking a node, selection shouldn't do anything
582:         if (!pointedNode)
583:             return;
584:
585:         //if nothing selected
586:         if (!selectedNode)
587:         {
588:             selectedNode = pointedNode;
589:             pointedHighlightable.Highlight();
590:             uiManager.UpdateAlgoSelection(selectedNode);
591:         }
592:         else
593:         {
594:             //if pointing at the selected node, deselect it
595:             if (pointedNode == selectedNode)
596:             {
597:                 selectedNode = null;
598:                 pointedHighlightable.Unhighlight();
599:             }
600:             //if pointing at a new node, deselect the old, select the new
601:             else
602:             {
603:                 selectedNode.GetComponent<Highlightable>().Unhighlight();
604:                 selectedNode = pointedNode;
605:                 pointedHighlightable.Highlight();
606:                 uiManager.UpdateAlgoSelection(selectedNode);
607:             }
608:         }
609:     }
610:
611:     public void SelectUIO()
612:     {
613:         if (pointedSelect)
614:             pointedSelect.Select.Invoke();
615:
616:         else if (pointedAlgo)
617:             pointedAlgo.Run.Invoke();
618:
619:         else if (pointedDemo)
620:             pointedDemo.Create.Invoke();
621:
622:         //Handled by invoke
623:         /*
624:         if (pointedUIO == createUIO)
625:             uiManager.CreateNode();
626:
627:         if(pointedDSO)
628:         {
629:             uiManager.CreateDemo(pointedDSO);
630:         }
631:
632:         if(pointedAO)
633:         {
634:             //Debug.Log($"Run call in actions");
635:             uiManager.Run(pointedAO, selectedNode);
636:         }
637:         */
638:     }
639:
640:     public void Flip()
641:     {
642:         if (!pointedLink)
643:             return;
644:
645:         Link.Flip(pointedLink);
```

```
646:     }
647:
648:     public void Increment(float f)
649:     {
650:         if (changeBuffer > 0)
651:             return;
652:         if (!pointedValue)
653:             return;
654:
655:         pointedValue.Increment.Invoke();
656:
657:         if (pointedValue.Continuous)
658:             changeBuffer = 0;
659:         else
660:             changeBuffer = 0.5f;
661:         //Handle by invoke!
662:
663:         /*
664:         //If the user is pointing at an option that can be incremented, then do so.
665:         if (changeBuffer > 0)
666:             return;
667:         if (!pointedUIO)
668:             return;
669:
670:         if(pointedUIO == typeUIO)
671:         {
672:             //change preview type
673:             uiManager.nextType();
674:             changeBuffer = 0.75f;
675:
676:         }
677:         else if(pointedUIO == valueUIO)
678:         {
679:             uiManager.incrementValue();
680:         }
681:         else if (pointedUIO == aValueUIO)
682:         {
683:             uiManager.incrementAlgoValue();
684:         }
685:         else if(pointedUIO == iterationUIO)
686:         {
687:             uiManager.nextIterationMode();
688:             changeBuffer = 0.75f;
689:         }
690:         else if(pointedUIO == demoTypeUIO)
691:         {
692:             uiManager.changeDemoType(true);
693:             changeBuffer = 0.75f;
694:         }
695:         */
696:     }
697:
698:     public void Decrement(float f)
699:     {
700:         if (changeBuffer > 0)
701:             return;
702:         if (!pointedValue)
703:             return;
704:
705:         pointedValue.Decrement.Invoke();
706:
707:         if (pointedValue.Continuous)
708:             changeBuffer = 0;
709:         else
710:             changeBuffer = 0.5f;
711:         /*
712:         if (changeBuffer > 0)
713:             return;
```

```
714:         if (!pointedUIO)
715:             return;
716:
717:         if (pointedUIO == typeUIO)
718:         {
719:             //change preview type
720:             uiManager.prevType();
721:             changeBuffer = 0.75f;
722:         }
723:         else if (pointedUIO == valueUIO)
724:         {
725:             uiManager.decrementValue();
726:         }
727:         else if (pointedUIO == aValueUIO)
728:         {
729:             uiManager.decrementAlgoValue();
730:         }
731:         else if (pointedUIO == iterationUIO)
732:         {
733:             uiManager.prevIterationMode();
734:             changeBuffer = 0.75f;
735:         }
736:         else if (pointedUIO == demoTypeUIO)
737:         {
738:             uiManager.changeDemoType(false);
739:             changeBuffer = 0.75f;
740:         }
741:         */
742:     }
743:
744:
745:     public void NextPanel()
746:     {
747:         //if (changeBuffer > 0)
748:         //    return;
749:         //changeBuffer = 0.5f;
750:         uiManager.nextPanel();
751:     }
752:
753:     //Settings
754:     GameObject settingsPanel;
755:     bool settingsVisible = false;
756:
757:     public void OpenSettings()
758:     {
759:         return;
760:         settingsVisible = !settingsVisible;
761:         settingsPanel.SetActive(settingsVisible);
762:
763:         if(settingsVisible)
764:         {
765:             SetCSEnabled(true);
766:             MoveCS();
767:         }
768:     }
769:
770:     public void CloseSettings()
771:     {
772:         settingsVisible = false;
773:         settingsPanel.SetActive(settingsVisible);
774:     }
775:
776:     public void ExitToMain()
777:     {
778:         SceneManager.LoadScene("Main Menu");
779:     }
780:
781:     //Algorithms stuff
```

```
  782:
  783:     public void Organize()
  784:     {
  785:         if (!selectedNode)
  786:             return;
  787:
  788:         if (Algorithms.inAlgo)
  789:             return;
  790:
  791:         //Organize around selected node:
  792:
  793:         GameObject structureObject;
  794:
  795:         if (selectedNode.GetType() == typeof(BinaryTree))
  796:             structureObject = BinaryTreeStructure.Organize((BinaryTree)selectedNode)
.gameObject;
  797:         else if (selectedNode.GetType() == typeof(LinkedList))
  798:             structureObject = LinkedListStructure.Organize((LinkedList)selectedNode)
.gameObject;
  799:         else if (selectedNode.GetType() == typeof(Undigraph))
  800:             structureObject = UndigraphStructure.Organize((Undigraph)selectedNode).g
ameObject;
  801:         else if (selectedNode.GetType() == typeof(Digraph))
  802:             structureObject = DigraphStructure.Organize((Digraph)selectedNode).gameO
bject;
  803:     }
  804:
  805:     public void Pickup()
  806:     {
  807:         //If holding something, drop it
  808:         if (held)
  809:         {
  810:             Drop();
  811:             return;
  812:         }
  813:
  814:         //If not holding anything, but not looking at any node to pick up, do nothin
g
  815:         if (!pointedNode)
  816:             return;
  817:
  818:         //If you're picking up a selected node, pick up the whole structure instead
  819:         if(pointedHighlightable.highlighted)
  820:         {
  821:             Structure s;
  822:             if(s = pointedNode.GetComponentInParent<Structure>())
  823:             {
  824:                 held = s.gameObject;
  825:                 holdingStructure = true;
  826:                 heldNode = pointedNode;
  827:                 Distance = Vector3.Distance(pointRaycaster.transform.position, held.
transform.position);
  828:                 foreach (Collider c in s.GetComponentsInChildren<Collider>())
  829:                 {
  830:                     c.enabled = false;
  831:                 }
  832:                 return;
  833:             }
  834:         }
  835:
  836:         held = pointedTo;
  837:         heldNode = pointedNode;
  838:         held.GetComponent<Collider>().enabled = false;
  839:         holdingStructure = false;
  840:
  841:         Distance = Vector3.Distance(pointRaycaster.transform.position, held.transfor
m.position);
  842:         //TempText.setTempText($"Distance: {Distance}", 0.1f);
```

```
  843:     }
  844:
  845:     private void Drop()
  846:     {
  847:         if(holdingStructure)
  848:         {
  849:             foreach (Collider c in held.GetComponentsInChildren<Collider>())
  850:             {
  851:                 c.enabled = true;
  852:             }
  853:             held = null;
  854:             heldNode = null;
  855:             holdingStructure = false;
  856:             return;
  857:         }
  858:         else
  859:         {
  860:             held.GetComponent<Collider>().enabled = true;
  861:             held = null;
  862:             heldNode = null;
  863:             return;
  864:         }
  865:     }
  866:
  867:     public void PickupMotion(float movement)
  868:     {
  869:         //While moving the held object, the stick cannot be used for both motion and
 pickup distance
  870:         Distance += movement * Time.deltaTime * 3;
  871:         //TempText.setTempText($"movement w {Distance}", 1);
  872:         movingHeld = true;
  873:     }
  874:
  875:     public void PickupMotionRelease()
  876:     {
  877:         //While not trying to move the held object, return to allowing stick for mot
ion
  878:         movingHeld = false;
  879:     }
  880:
  881:     private void MoveHeld()
  882:     {
  883:         //TempText.setTempText($"on movement s {held.name}", 0.1f);
  884:         if (!held)
  885:             return;
  886:
  887:         Vector3 targetPos = pointRaycaster.transform.position;
  888:         targetPos += pointRaycaster.transform.forward * Distance;
  889:         targetPos.y = Mathf.Clamp(targetPos.y, 1, 100);
  890:         held.transform.position = targetPos;
  891:
  892:         //TempText.setTempText($"on movement m {targetPos.ToString()} and {heldNode.
name}", 0.1f);
  893:
  894:         //this might be sl0w
  895:         if (!holdingStructure)
  896:             heldNode.OnMovement();
  897:
  898:         //TempText.setTempText($"on movement e", 0.1f);
  899:     }
  900:
  901:
  902:     //Node Creation
  903:     public void CreateNode(Vector3 position, int value)
  904:     {
  905:         Node newNode = DelCreateNode(value);
  906:         //newNode.transform.position = position;
  907:     }
```

```
  908:
  909:     public void IncrementNodeSelection()
  910:     {
  911:         //Increment Selection Enum
  912:         selection++;
  913:         if (selection > CreationSelection.Digraph)
  914:             selection = (CreationSelection)0;
  915:
  916:         /*
  917:         //Get delegate from selection by type
  918:         //Get Index
  919:         int selectionIndex = (int)selection;
  920:         Debug.Log($"Incrementing selection index to {selectionIndex}, or {selection.
ToString()}");
  921:         //Get Method Info from the list of types, pulling the "CreateNode" method
  922:         MethodInfo selectedCreationMethodInfo = NodeTypes[selectionIndex].GetMethod(
"Create");
  923:         //Convert into the delegate type for node creation.
  924:         DelCreateNode = (NodeCreationMethod)selectedCreationMethodInfo.CreateDelegat
e(typeof(NodeCreationMethod));
  925:         */
  926:     }
  927:
  928:     public void Delete()
  929:     {
  930:         if (pointedLink)
  931:             Link.Disconnect(pointedLink);
  932:         if (pointedNode)
  933:             pointedNode.Delete();
  934:     }
  935:
  936:     //Creation Station
  937:     public void MoveCS()
  938:     {
  939:         //align to player
  940:         Vector3 csPos = PlayerObject.transform.position + new Vector3(0, -2, 8);
  941:         csPos.y = 1.0f;
  942:         CreationStation.transform.position = csPos;
  943:     }
  944:
  945:     public void ToggleCS()
  946:     {
  947:         if(Tutorial.restrictCSToggle)
  948:         {
  949:             CreationStation.SetActive(true);
  950:             MoveCS();
  951:             return;
  952:         }
  953:
  954:         csEnabled = !csEnabled;
  955:         if (csEnabled)
  956:             MoveCS();
  957:         CreationStation.SetActive(csEnabled);
  958:     }
  959:
  960:     public void SetCSEnabled(bool enabled)
  961:     {
  962:         csEnabled = enabled;
  963:         if (csEnabled)
  964:             MoveCS();
  965:         CreationStation.SetActive(csEnabled);
  966:     }
  967:
  968:     public void TeleportAim()
  969:     {
  970:         if (!teleportRaycaster.cast(3))
  971:         {
  972:             Vector3[] blank = { Vector3.zero, Vector3.zero };
```

```
  973:             rightLR.SetPositions(blank);
  974:             return;
  975:         }
  976:         teleportIndicator.transform.position = teleportRaycaster.info.point;
  977:         teleportIndicator.SetActive(true);//teleportIndicator.transform.localScale =
 Vector3.one * 0.5f;
  978:
  979:
  980:         Vector3[] tpPos = { teleportRaycaster.castPosition, teleportRaycaster.info.p
oint};
  981:         rightLR.SetPositions(tpPos);
  982:
  983:     }
  984:
  985:     //stop drawing indicator when the teleport aim button is released
  986:     public void TeleportAimRelease()
  987:     {
  988:         Vector3[] blank = { Vector3.zero, Vector3.zero };
  989:         rightLR.SetPositions(blank);
  990:         teleportIndicator.SetActive(false); //teleportIndicator.transform.localScale
 = Vector3.zero;
  991:         return;
  992:     }
  993:
  994:     public void Teleport()
  995:     {
  996:         if (tpBuffer > 0)
  997:             return;
  998:         tpBuffer = 1;
  999:
 1000:         if (!teleportRaycaster.cast(3)) //debug invalid loc?
 1001:             return;
 1002:         teleportIndicator.transform.position = teleportRaycaster.info.point;
 1003:         //teleportIndicator.transform.localScale = Vector3.one * 2;
 1004:         //move player, but ignore the y position change.
 1005:         PlayerObject.transform.position = new Vector3(teleportRaycaster.info.point.x
, PlayerObject.transform.position.y, teleportRaycaster.info.point.z); // + Vector3.up;
 1006:
 1007:         MoveCS();
 1008:     }
 1009:
 1010:     public void SetSpeed(float speed)
 1011:     {
 1012:         oculusMovementComponent.Speed = speed;
 1013:     }
 1014:
 1015:     public void SetRotationAngle(float ang)
 1016:     {
 1017:         oculusMovementComponent.RotationAngle = ang;
 1018:     }
 1019: }
```

```
    1: ï»¿using System;
    2: using System.Collections;
    3: using System.Collections.Generic;
    4: using UnityEngine;
    5:
    6: public class InputBindings : MonoBehaviour
    7: {
    8:     //Cross Platform Input
    9:     public enum Platforms:uint { Editor = 0, WindowStandalone = 1, Web = 2, Quest =
3 };
   10:     public Platforms platform;
   11:     public delegate void PlatformInput();
   12:     PlatformInput updateMethod;
   13:
   14:     //Players to enable / disable
   15:     public GameObject playerKBM, playerQuest;
   16:
   17:     //Physics and Movespeed parameters
   18:     public float moveSpeed;
   19:
   20:     //Movement Variables
   21:     private int x, y, z;
   22:     private Vector3 offset;
   23:     private Vector3 jump;
   24:
   25:     //Camera Movement Variables
   26:     public float lookSpeed;
   27:     public float minLook, maxLook;
   28:     private Vector2 rotation;
   29:
   30:     //If this is on the startup screen or in the tutorial
   31:     public bool inMainMenu;
   32:
   33:     //Needed References
   34:     Camera PlayerCamera;
   35:     Actions ActionsComponent;
   36:
   37:     //Execution Order is important: platform assignment needs to happen before anyth
ing else.
   38:     private void Awake()
   39:     {
   40:         //Performance Check:
   41:         Debug.Log($"Time: {Time.realtimeSinceStartup}");
   42:
   43:         //check if in main menu, or in tutorial
   44:
   45:         //Call general reference/global initialization
   46:         Node.Init();
   47:         Link.Init();
   48:         Algorithms.Init();
   49:         if (!inMainMenu)
   50:             Algorithms.iterationMode = Algorithms.IterationMode.Input;
   51:
   52:         ActionsComponent = GetComponent<Actions>();
   53:         ActionsComponent.Init();
   54:
   55:         //Update Settings: 72fps
   56:         //Time.fixedDeltaTime = (1.0f / 72.0f);
   57:         Application.targetFrameRate = 72;
   58:
   59:         Debug.Log($"Platform: {Application.platform}");
   60:         //Bind input method based on platform
   61:         switch (Application.platform)
   62:         {
   63:             case RuntimePlatform.WindowsEditor:
   64:                 platform = Platforms.Editor;
   65:                 updateMethod = UpdateBoth;
   66:                 playerKBM.SetActive(true);
```

```
    67:                 playerQuest.SetActive(false);
    68:                 ActionsComponent.AssignPlayerReferences(false);
    69:                 Algorithms.vr = false;
    70:                 /*
    71:                 if (OVRManager.hasVrFocus)
    72:                 {
    73:                     playerKBM.SetActive(false);
    74:                     playerQuest.SetActive(true);
    75:                     ActionsComponent.AssignPlayerReferences(true);
    76:                     Algorithms.vr = true;
    77:                 }
    78:                 else
    79:                 {
    80:                     playerKBM.SetActive(true);
    81:                     playerQuest.SetActive(false);
    82:                     ActionsComponent.AssignPlayerReferences(false);
    83:                     Algorithms.vr = false;
    84:                 }*/
    85:                 break;
    86:             case RuntimePlatform.WindowsPlayer:
    87:                 platform = Platforms.WindowStandalone;
    88:                 updateMethod = UpdateKBM;
    89:                 ActionsComponent.AssignPlayerReferences(false);
    90:                 Algorithms.vr = false;
    91:                 ActionsComponent.vr = false;
    92:                 break;
    93:             case RuntimePlatform.Android:
    94:                 platform = Platforms.Quest;
    95:                 updateMethod = UpdateQuest;
    96:                 ActionsComponent.AssignPlayerReferences(true);
    97:                 Algorithms.vr = true;
    98:                 ActionsComponent.vr = true;
    99:                 break;
   100:             case RuntimePlatform.WebGLPlayer:
   101:                 platform = Platforms.Web;
   102:                 updateMethod = UpdateKBM;
   103:                 ActionsComponent.AssignPlayerReferences(false);
   104:                 Algorithms.vr = false;
   105:                 ActionsComponent.vr = false;
   106:                 break;
   107:             default:
   108:                 Debug.LogError($"ERROR: Platform {Application.platform} is unsupport
ed! How did you get here?");
   109:                 //TODO temp_text debug:
   110:                 Time.timeScale = 0;
   111:                 return;
   112:         }
   113:
   114:         //By default, the VR camera will be enabled to avoid rendering issues on sta
rtup.
   115:         if (updateMethod == UpdateKBM)
   116:         {
   117:             playerKBM.SetActive(true);
   118:             playerQuest.SetActive(false);
   119:             Cursor.lockState = CursorLockMode.Locked;
   120:         }
   121:         //Quest Specific Settings
   122:         else if (updateMethod == UpdateQuest)
   123:         {
   124:             playerKBM.SetActive(false);
   125:             playerQuest.SetActive(true);
   126:         }
   127:         else if (updateMethod == UpdateBoth)
   128:         {
   129:             playerKBM.SetActive(true);
   130:             playerQuest.SetActive(true);
   131:             Cursor.lockState = CursorLockMode.Locked;
   132:         }
```

```
  133:         //If no input method, log error.
  134:         else
  135:         {
  136:             Debug.LogError($"ERROR: No Input Method Bound.");
  137:         }
  138:
  139:         PlayerCamera = playerKBM.GetComponentInChildren<Camera>(); //Camera.main;
  140:
  141:         rotation = Vector2.zero;
  142:
  143:         Debug.Log($"Time: {Time.realtimeSinceStartup}");
  144:     }
  145:
  146:     // Update is called once per frame
  147:     void Update()
  148:     {
  149:         //Call delegate
  150:         updateMethod();
  151:     }
  152:
  153:     //Update method for Touch controllers
  154:     void UpdateQuest()
  155:     {
  156:         //According to the documentation, this is a requirement.
  157:         //OVRInput.Update();
  158:         //OVRInput.FixedUpdate();
  159:
  160:         //Actually, it seems that this is only required if OVRManager is excluded, a
nd it seems to introduce inconsistencies with GetDown and GetUp.
  161:
  162:         //If the left hand is looking at something it can click or pickup, it should
 be highlighted, and a line should be drawn to it
  163:         //This also needs to be called before anything else, so that references are
properly assigned this frame
  164:         ActionsComponent.LeftHandTracking();
  165:
  166:         //If the player is in the starting screen, all other actions should be disab
led
  167:         if(inMainMenu)
  168:         {
  169:             //When in the main menu, hand tracking should be shown when pointing at
options, but movement, snap turning, and all other actions should be disabled.
  170:             //MainMenuActions(); // or something else here
  171:             if (OVRInput.GetDown(OVRInput.Button.PrimaryIndexTrigger))
  172:             {
  173:                 ActionsComponent.MenuSelect();
  174:             }
  175:             return;
  176:         }
  177:
  178:         //If tutorial affects this
  179:         if(Tutorial.restrictMovement)
  180:         {
  181:             ActionsComponent.SetRotationAngle(0);
  182:             ActionsComponent.SetSpeed(0);
  183:         }
  184:         else
  185:         {
  186:             ActionsComponent.SetRotationAngle(45);
  187:             ActionsComponent.SetSpeed(12);
  188:         }
  189:
  190:         if (OVRInput.GetDown(OVRInput.Button.PrimaryIndexTrigger))
  191:         {
  192:             ActionsComponent.SelectUIO();
  193:         }
  194:
  195:         if(OVRInput.Get(OVRInput.Button.PrimaryThumbstick) && OVRInput.Get(OVRInput.
```

```
Button.SecondaryThumbstick) && OVRInput.Get(OVRInput.Button.PrimaryHandTrigger) && OVRInput
.Get(OVRInput.Button.SecondaryHandTrigger))
  196:         {
  197:             ActionsComponent.LoadMain();
  198:         }
  199:                 /*
  200:         if(OVRInput.Get(OVRInput.Button.SecondaryHandTrigger))
  201:         {
  202:             //aim teleport
  203:             //TempText.setTempText("Aiming", 5);
  204:             ActionsComponent.TeleportAim();
  205:         }
  206:         else
  207:         {
  208:             ActionsComponent.TeleportAimRelease();
  209:         }
  210:
  211:         //TempText.setTempText($"{OVRInput.Get(OVRInput.Axis1D.SecondaryIndexTrigger
)} + {OVRInput.GetDown(OVRInput.Button.SecondaryIndexTrigger)}", 0.1f);
  212:         if (OVRInput.Get(OVRInput.Axis1D.SecondaryIndexTrigger) > 0.95f)
  213:         {
  214:             //teleport
  215:             //TempText.setTempText("Teleporting", 5);
  216:             ActionsComponent.Teleport();
  217:         }
  218:
  219:
  220:         showInfo = OVRInput.Get(OVRInput.Touch.PrimaryThumbstick);
  221:
  222:         if(!Tutorial.restrictInfo && !(ActionsComponent.connecting || ActionsCompone
nt.connectionBuffer > 0))
  223:         {
  224:             if (showInfo)
  225:                 ActionsComponent.SetInfo(ActionsComponent.GetInfo());
  226:             else
  227:                 ActionsComponent.ClearInfo();
  228:         }
  229:
  230:         if (OVRInput.GetDown(OVRInput.Button.PrimaryIndexTrigger))
  231:         {
  232:             ActionsComponent.SelectUIO();
  233:             ActionsComponent.SelectNode();
  234:             ActionsComponent.Flip();
  235:         }
  236:         if (OVRInput.GetDown(OVRInput.Button.PrimaryHandTrigger))
  237:         {
  238:             ActionsComponent.Pickup();
  239:         }
  240:
  241:         if (OVRInput.Get(OVRInput.Button.PrimaryThumbstick))
  242:         {
  243:             ActionsComponent.PickupMotion(OVRInput.Get(OVRInput.Axis2D.PrimaryThumbs
tick).y);
  244:         }
  245:
  246:         if(OVRInput.GetUp(OVRInput.Button.PrimaryThumbstick))
  247:         {
  248:             ActionsComponent.PickupMotionRelease();
  249:         }
  250:
  251:         if ((OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick).y) >= 0.125f)
  252:         {
  253:             ActionsComponent.Increment((OVRInput.Get(OVRInput.Axis2D.PrimaryThumbsti
ck).y));
  254:         }
  255:
  256:         if (OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick).y <= -0.125f)
  257:         {
```

```
  258:             ActionsComponent.Decrement((OVRInput.Get(OVRInput.Axis2D.PrimaryThumbsti
ck).y));
  259:         }
  260:
  261:         if (OVRInput.GetDown(OVRInput.Button.SecondaryThumbstick))
  262:         {
  263:             ActionsComponent.NextPanel();
  264:         }
  265:
  266:         if (OVRInput.GetDown(OVRInput.Button.Start))
  267:         {
  268:             ActionsComponent.OpenSettings();
  269:         }
  270:
  271:
  272:         //Button presses one to four
  273:         //A
  274:         if (OVRInput.GetDown(OVRInput.Button.One))
  275:         {
  276:             ActionsComponent.Organize();
  277:         }
  278:         //B
  279:         if (OVRInput.Get(OVRInput.Touch.Two))
  280:         {
  281:             ActionsComponent.MoveCS();
  282:         }
  283:         if (OVRInput.GetDown(OVRInput.Button.Two))
  284:         {
  285:             ActionsComponent.ToggleCS();
  286:         }
  287:         //X
  288:         if (OVRInput.GetDown(OVRInput.Button.Three))
  289:         {
  290:             ActionsComponent.Delete();
  291:         }
  292:         //Y
  293:         if (OVRInput.GetDown(OVRInput.Button.Four))
  294:         {
  295:             //TempText.setTempText("four down", 1);
  296:             ActionsComponent.ConnectNodes();
  297:         }
  298:
  299:         //TempText.setTempText("End of update reached.", 0.2f);
  300:
  301:         /*
  302:         string buttons = "";
  303:
  304:         foreach (OVRInput.Button button in (OVRInput.Button[])Enum.GetValues(typeof(
OVRInput.Button)))
  305:         {
  306:             if (OVRInput.Get(button))
  307:                 buttons += button.ToString();
  308:         }
  309:
  310:         TempText.setTempText(buttons, 0.5f);
  311:         */
  312:     }
  313:
  314:     //Update method for all keyboard and mouse controls, currently in Editor, Standa
lone, or Web Player.
  315:     private bool showInfo;
  316:     void UpdateKBM()
  317:     {
  318:         //Performance check:
  319:         //Debug.Log($"FPS: {1.0f / Time.deltaTime }");
  320:
  321:         if (Input.GetKey(KeyCode.Q) && Input.GetKey(KeyCode.W) && Input.GetKey(KeyCo
de.E) && Input.GetKey(KeyCode.R))
```

```
322:         {
323:             ActionsComponent.LoadMain();
324:         }
325:
326:         //Keyboard movement, mouse looking
327:         KBMove();
328:         MouseLook();
329:
330:         //Raycast to track objects
331:         ActionsComponent.ScreenTracking();
332:
333:         if (inMainMenu)
334:         {
335:             //When in the main menu, hand tracking should be shown when pointing at
options, but movement, snap turning, and all other actions should be disabled.
336:             //MainMenuActions(); // or something else here
337:             if (Input.GetMouseButtonDown(0))
338:             {
339:                 ActionsComponent.MenuSelect();
340:             }
341:             return;
342:         }
343:
344:         //Other Actions
345:
346:         //Implement GetInfo/ClearInfo
347:
348:         //Mouse Buttons
349:         if (Input.GetMouseButtonDown(0))
350:         {
351:             ActionsComponent.SelectUIO();
352:             ActionsComponent.SelectNode();
353:             ActionsComponent.Flip();
354:         }
355:         if (Input.GetMouseButtonDown(1))
356:         {
357:             ActionsComponent.ConnectNodes();
358:         }
359:         if (Input.GetMouseButtonDown(2))
360:         {
361:             ActionsComponent.Pickup();
362:         }
363:
364:         //Debug.Log($"Scroll {Input.mouseScrollDelta}");
365:         if(Input.mouseScrollDelta.y != 0)
366:         {
367:             ActionsComponent.PickupMotion(Input.mouseScrollDelta.y * 10);
368:             if (Input.mouseScrollDelta.y < 0)
369:                 ActionsComponent.Decrement(1);
370:             else
371:                 ActionsComponent.Increment(1);
372:         }
373:
374:         if (Input.GetKeyDown(KeyCode.Tab))
375:             ActionsComponent.NextPanel();
376:
377:         if (Input.GetKeyDown(KeyCode.Escape) || Input.GetKeyDown(KeyCode.E))
378:             ActionsComponent.OpenSettings();
379:
380:         if (Input.GetKeyDown(KeyCode.Q))
381:             ActionsComponent.Organize();
382:
383:         if (Input.GetKeyDown(KeyCode.C))
384:             ActionsComponent.ToggleCS();
385:
386:         if(Input.GetKeyDown(KeyCode.X))
387:             ActionsComponent.MoveCS();
388:
```

```
  389:         if (Input.GetKeyDown(KeyCode.F))
  390:             ActionsComponent.Delete();
  391:
  392:         if(Input.GetKeyDown(KeyCode.BackQuote))
  393:         {
  394:             showInfo = !showInfo;
  395:             Debug.Log($"Togggling information display to {showInfo}");
  396:
  397:             //display the swap to the player
  398:         }
  399:
  400:         if (!Tutorial.restrictInfo && !(ActionsComponent.connecting || ActionsCompon
ent.connectionBuffer > 0))
  401:         {
  402:             if (showInfo)
  403:                 ActionsComponent.SetInfo(ActionsComponent.GetInfo());
  404:             else
  405:                 ActionsComponent.ClearInfo();
  406:         }
  407:
  408:
  409:     }
  410:
  411:     void MouseLook()
  412:     {
  413:         //Camera
  414:         rotation.y += lookSpeed * Input.GetAxis("Mouse X");
  415:         rotation.x += lookSpeed * -Input.GetAxis("Mouse Y");
  416:
  417:         rotation.x = Mathf.Clamp(rotation.x, minLook, maxLook);
  418:         //Debug.Log($"Rotation: {rotation}");
  419:         PlayerCamera.transform.eulerAngles = rotation;
  420:     }
  421:
  422:     void KBMove()
  423:     {
  424:         if (Tutorial.restrictMovement)
  425:             return;
  426:
  427:         x = y = z = 0;
  428:         Vector3 offset = Vector3.zero;
  429:
  430:         z += Input.GetKey(KeyCode.W) ? 1 : 0;
  431:         z += Input.GetKey(KeyCode.S) ? -1 : 0;
  432:         x += Input.GetKey(KeyCode.A) ? -1 : 0;
  433:         x += Input.GetKey(KeyCode.D) ? 1 : 0;
  434:
  435:         offset += PlayerCamera.transform.forward * z * moveSpeed * Time.deltaTime;
  436:         offset += PlayerCamera.transform.right * x * moveSpeed * Time.deltaTime;
  437:         offset.y = 0;
  438:
  439:         playerKBM.transform.position += offset; //ActionsComponent.Move(offset);
  440:         playerKBM.transform.position = new Vector3(playerKBM.transform.position.x, 4
.5f, playerKBM.transform.position.z);
  441:         Camera.main.transform.position = new Vector3(playerKBM.transform.position.x,
 4.5f, playerKBM.transform.position.z);
  442:     }
  443:
  444:     //In the windows editor, KBM or tethered Quest controls may be neccesary.
  445:     //By default, the VR camera will be enabled to avoid rendering issues on startup
.
  446:     private bool last = true;
  447:     void UpdateBoth()
  448:     {
  449:         //Debug.Log($"{Time.deltaTime}");
  450:
  451:         //if switching vr focus
  452:         if (OVRManager.hasVrFocus != last)
```

```
453:         {
454:             if (OVRManager.hasVrFocus)
455:             {
456:                 playerKBM.SetActive(false);
457:                 playerQuest.SetActive(true);
458:                 ActionsComponent.AssignPlayerReferences(true);
459:                 Algorithms.vr = true;
460:                 if (GetComponent<Tutorial>())
461:                     GetComponent<Tutorial>().SetTutorialReferences();
462:             }
463:             else
464:             {
465:                 playerKBM.SetActive(true);
466:                 playerQuest.SetActive(false);
467:                 ActionsComponent.AssignPlayerReferences(false);
468:                 Algorithms.vr = false;
469:                 if (GetComponent<Tutorial>())
470:                     GetComponent<Tutorial>().SetTutorialReferences();
471:             }
472:         }
473:
474:         if(OVRManager.hasVrFocus)
475:         {
476:             UpdateQuest();
477:         }
478:         else
479:         {
480:             UpdateKBM();
481:         }
482:     }
483:
484:     //Satisfy VRC Functional.2
485:     private void OnApplicationPause(bool pause)
486:     {
487:         if (pause)
488:             Time.timeScale = 0;
489:         else
490:             Time.timeScale = 1;
491:     }
492: }
493:
494:
495:
496:
497: /*
498:  *
499:
500:         //Capacitive sensing:
501:         //If the player touches the primary touchpad, it should give them info about
 what they're pointing at.
502:         if (OVRInput.Get(OVRInput.Touch.PrimaryThumbstick))
503:         {
504:             ActionsComponent.SetInfo(ActionsComponent.GetInfo());
505:         }
506:         else
507:         {
508:             ActionsComponent.ClearInfo();
509:         }
510: */
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4: using UnityEngine.UI;
    5: using TMPro;
    6: using System;
    7:
    8: public class UIManager : MonoBehaviour
    9: {
   10:     //This class will manage the 'creation station'
   11:
   12:     //Component References
   13:     Actions ActionsComponent;
   14:     Algorithms AlgorithmsComponent;
   15:
   16:     //Panels
   17:     public enum Panel { Creation = 0, DemoStructure = 1, Algorithm = 2, Info = 3,  }
;
   18:     public Panel VisiblePanel;
   19:     public Canvas creationPanel, algorithmPanel, infoPanel, demoStructPanel;
   20:     public List<Canvas> panels;
   21:
   22:     //Creation Location
   23:     public Transform locator;
   24:
   25:     //Materials
   26:     public Material llMat, btMat, dgMat, ugMat;
   27:     public Material[] materials = new Material[4];
   28:
   29:     //Type
   30:     public enum NodeType { BinaryTree = 0, LinkedList = 1, Digraph = 2, Undigraph =
3 };
   31:     public NodeType type;
   32:     public Image typeImage;
   33:     public TMP_Text typeText;
   34:
   35:     //Value
   36:     public Image valueImage;
   37:     public TMP_Text valueText;
   38:     private int val;
   39:     public int Value
   40:     {
   41:         get
   42:         {
   43:             return val;
   44:         }
   45:         set
   46:         {
   47:             val = value;
   48:             if (val < 0)
   49:                 val = 99;
   50:             else if (val > 99)
   51:                 val = 0;
   52:             valueText.text = val.ToString();
   53:             UpdatePreview();
   54:         }
   55:     }
   56:
   57:     //Iteration Mode
   58:     //LATER
   59:
   60:     //Preview
   61:     public Image previewImage;
   62:     public TMP_Text previewText;
   63:
   64:     public void Start()
   65:     {
   66:         materials[0] = btMat;
```

```
   67:         materials[1] = llMat;
   68:         materials[2] = dgMat;
   69:         materials[3] = ugMat;
   70:         panels.Add(creationPanel);
   71:         panels.Add(demoStructPanel);
   72:         panels.Add(algorithmPanel);
   73:         panels.Add(infoPanel);
   74:         ActionsComponent = GetComponentInParent<Actions>();
   75:         AlgorithmsComponent = GetComponentInParent<Algorithms>();
   76:
   77:         //Update the visuals on start.
   78:         Value = 0;
   79:         VisiblePanel = 0;
   80:         typeImage.material = materials[(int)type];
   81:         typeText.text = type.ToString();
   82:         UpdatePreview();
   83:
   84:         demoType = 0;
   85:         //Fill in demo panel references
   86:         for(int i = 0; i < 4; i++)
   87:         {
   88:             DemoTypeHolders[i] = demoStructPanel.transform.Find(((NodeType)i).ToStri
ng()).gameObject;
   89:             DemoTypeHolders[i].SetActive(false);
   90:         }
   91:
   92:         for (int i = 0; i < 4; i++)
   93:         {
   94:             AlgorithmHolders[i] = algorithmPanel.transform.Find(((NodeType)i).ToStri
ng()).gameObject;
   95:             AlgorithmHolders[i].SetActive(false);
   96:         }
   97:
   98:         changeDemoType(true);
   99:         changeDemoType(false);
  100:     }
  101:
  102:     public void nextPanel()
  103:     {
  104:         if (Tutorial.restrictPanelChange)
  105:             return;
  106:
  107:         panels[(int)VisiblePanel].gameObject.SetActive(false);
  108:         VisiblePanel++;
  109:         if ((int)VisiblePanel > 2) //flip this to 2 when info panel is ready
  110:             VisiblePanel = 0;
  111:         panels[(int)VisiblePanel].gameObject.SetActive(true);
  112:         //Debug.Log($"Next panel to {VisiblePanel.ToString()}");
  113:     }
  114:
  115:     //Creation Panel
  116:     public void nextType()
  117:     {
  118:         type++;
  119:         if ((int)type > 3)
  120:             type = (NodeType)0;
  121:
  122:         typeImage.material = materials[(int)type];
  123:         typeText.text = type.ToString();
  124:         UpdatePreview();
  125:     }
  126:
  127:     public void prevType()
  128:     {
  129:         type--;
  130:         if ((int)type < 0)
  131:             type = (NodeType)3;
  132:
```

```
133:         typeImage.material = materials[(int)type];
134:         typeText.text = type.ToString();
135:         UpdatePreview();
136:     }
137:
138:     public void incrementValue()
139:     {
140:         Value = Value + 1;
141:     }
142:
143:     public void decrementValue()
144:     {
145:         Value = Value - 1;
146:     }
147:     private void UpdatePreview()
148:     {
149:         previewText.text = val.ToString();
150:         previewImage.material = materials[(int)type];
151:     }
152:
153:     public void CreateNode()
154:     {
155:         GameObject newNode;
156:         newNode = Node.Create(Value, (int)type).gameObject;
157:         newNode.transform.position = locator.position;
158:     }
159:
160:     //Algorithms Panel
161:     private GameObject[] AlgorithmHolders = new GameObject[4];
162:     private int[] AlgoCount = { 3, 2, 2, 2 };
163:
164:     public TMP_Text iterationText;
165:     public Image algoSelectionImage;
166:     public TMP_Text algoSelectionText;
167:     public GameObject noStructureText;
168:     private NodeType selectedNodeType;
169:     private Structure selectedStructure;
170:
171:     public Image algoValueImage;
172:     public TMP_Text algoValueText;
173:     private int aval;
174:     public int aValue
175:     {
176:         get
177:         {
178:             return aval;
179:         }
180:         set
181:         {
182:             aval = value;
183:             if (aval < 0)
184:                 aval = 99;
185:             else if (aval > 99)
186:                 aval = 0;
187:             algoValueText.text = aval.ToString();
188:         }
189:     }
190:     public void incrementAlgoValue()
191:     {
192:         aValue = aValue + 1;
193:     }
194:
195:     public void decrementAlgoValue()
196:     {
197:         aValue = aValue - 1;
198:     }
199:
200:     public void nextIterationMode()
```

```
  201:     {
  202:         Algorithms.ChangeIterationMode(true);
  203:         iterationText.text = Algorithms.iterationMode.ToString();
  204:     }
  205:
  206:     public void prevIterationMode()
  207:     {
  208:         Algorithms.ChangeIterationMode(false);
  209:         iterationText.text = Algorithms.iterationMode.ToString();
  210:     }
  211:
  212:     public void UpdateAlgoSelection(Node selectedNode)
  213:     {
  214:         //hide previous
  215:         AlgorithmHolders[(int)selectedNodeType].SetActive(false);
  216:
  217:         //If this node is not part of a structure, it cannot be used in an algorithm
.
  218:         if (!selectedNode.GetComponentInParent<Structure>())
  219:         {
  220:             noStructureText.SetActive(true);
  221:             return;
  222:         }
  223:         noStructureText.SetActive(false);
  224:
  225:
  226:
  227:         int selectInt = 0;
  228:         if (selectedNode.GetType() == typeof(BinaryTree))
  229:             selectInt = 0;
  230:         else if (selectedNode.GetType() == typeof(LinkedList))
  231:             selectInt = 1;
  232:         else if (selectedNode.GetType() == typeof(Digraph))
  233:             selectInt = 2;
  234:         else if (selectedNode.GetType() == typeof(Undigraph))
  235:             selectInt = 3;
  236:
  237:         //Show/hide panel:
  238:         AlgorithmHolders[selectInt].SetActive(true);
  239:         for(int i = 0; i < 6; i++)
  240:         {
  241:             bool active = (i < AlgoCount[selectInt]);
  242:             AlgorithmHolders[selectInt].transform.GetChild(i).gameObject.SetActive(a
ctive);
  243:         }
  244:
  245:         selectedNodeType = (NodeType)selectInt;
  246:         algoSelectionImage.material = materials[selectInt];
  247:         algoSelectionText.text = FormattedNodeName(selectedNodeType);
  248:     }
  249:
  250:     /*
  251:     public void Run(AlgorithmOption ao, Node selectedNode)
  252:     {
  253:         //Debug.Log($"Run call in uim");
  254:         StartCoroutine(AlgorithmsComponent.RunAlgorithm(selectedNodeType, ao.index,
selectedNode.GetComponentInParent<Structure>(), aValue));
  255:     }*/
  256:
  257:     public void Run(int index, Node selectedNode)
  258:     {
  259:         //Debug.Log($"Run call in uim");
  260:         StartCoroutine(AlgorithmsComponent.RunAlgorithm(selectedNodeType, index, sel
ectedNode.GetComponentInParent<Structure>(), aValue));
  261:     }
  262:
  263:
  264:     //Demo Panel
```

```
  265:     public NodeType demoType;
  266:     public Image demoTypeImage;
  267:     public TMP_Text demoTypeText;
  268:
  269:     //Demo Panel Options
  270:     private GameObject[] DemoTypeHolders = new GameObject[4];
  271:
  272:     public void changeDemoType(bool inc)
  273:     {
  274:         //hide previous set
  275:         DemoTypeHolders[(int)demoType].SetActive(false);
  276:
  277:         demoType += (inc) ? 1 : -1;
  278:
  279:         if ((int)demoType > 3)
  280:             demoType = (NodeType)0;
  281:         if ((int)demoType < 0)
  282:             demoType = (NodeType)3;
  283:
  284:         demoTypeImage.material = materials[(int)demoType];
  285:         demoTypeText.text = demoType.ToString();
  286:
  287:         //Update the visuals
  288:         int t = (int)demoType;
  289:
  290:         DemoTypeHolders[t].SetActive(true);
  291:         Transform demos = DemoTypeHolders[t].transform;
  292:
  293:         for (int i = 0; i < 6; i++)
  294:         {
  295:             //if index is within the number of existing demo structures, allow that
component to be accessed
  296:             bool active = (i < DemoStructures.StructCount[t]);
  297:             demos.GetChild(i).gameObject.SetActive(active);
  298:         }
  299:     }
  300:
  301:     /*
  302:     public void CreateDemo(DemoStructureOption demo)
  303:     {
  304:         int index = demo.index;
  305:         GameObject newStructure = DemoStructures.CreateDemo((DemoStructures.NodeType
)demoType, index);
  306:
  307:         //ensure position above ground
  308:         float minPos = -1;
  309:         foreach(Transform t in newStructure.GetComponentInChildren<Transform>())
  310:         {
  311:             minPos = Mathf.Min(minPos, t.position.y);
  312:         }
  313:         Debug.Log(minPos);
  314:
  315:         newStructure.transform.position -= (Vector3.up*minPos);
  316:         newStructure.transform.position += Vector3.up;
  317:
  318:     }*/
  319:
  320:     public void CreateDemo(int index)
  321:     {
  322:         GameObject newStructure = DemoStructures.CreateDemo((DemoStructures.NodeType
)demoType, index);
  323:
  324:         //ensure position above ground
  325:         float minPos = -1;
  326:         foreach (Transform t in newStructure.GetComponentInChildren<Transform>())
  327:         {
  328:             minPos = Mathf.Min(minPos, t.position.y);
  329:         }
```

```
  330:         Debug.Log(minPos);
  331:
  332:         newStructure.transform.position -= (Vector3.up * minPos);
  333:         newStructure.transform.position += Vector3.up;
  334:
  335:     }
  336:
  337:     //Info Panel
  338:
  339:     //General
  340:     public static string FormattedNodeName(Node n)
  341:     {
  342:         if (!n)
  343:             return "Null Node";
  344:
  345:         if (n.GetType() == typeof(BinaryTree))
  346:             return "Binary Tree";
  347:         else if (n.GetType() == typeof(LinkedList))
  348:             return "Linked List";
  349:         else if (n.GetType() == typeof(Undigraph))
  350:             return "Undirected Graph";
  351:         else if (n.GetType() == typeof(Digraph))
  352:             return "Directed Graph";
  353:         else
  354:             throw new Exception("No type found from typeof in formatting node name")
;
  355:     }
  356:
  357:     public static string FormattedNodeName(UIManager.NodeType n)
  358:     {
  359:         switch (n)
  360:         {
  361:             case UIManager.NodeType.BinaryTree:
  362:                 return "Binary Tree";
  363:             case UIManager.NodeType.LinkedList:
  364:                 return "Linked List";
  365:             case UIManager.NodeType.Undigraph:
  366:                 return "Undirected Graph";
  367:             case UIManager.NodeType.Digraph:
  368:                 return "Directed Graph";
  369:             default:
  370:                 throw new Exception("No type found from typeof in formatting node na
me");
  371:         }
  372:     }
  373:
  374: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class BinaryTreeStructure : Structure
    6: {
    7:     //Contains all information about a binary tree that might be useful for an algor
ithm.
    8:     //Members
    9:     public BinaryTree root;
   10:     public List<BinaryTree> leaves; // list of references to all the leaves; not gua
ranteed to be on all on the bottom level unless complete.
   11:     //These are null if they are yet unchecked
   12:     public bool? full;
   13:     public bool? complete;
   14:     public int? width;
   15:     public int? depth;
   16:
   17:     //Methods
   18:
   19:     //Static Members
   20:     public const float widthFactor = 3;
   21:     public const float depthFactor = 3;
   22:
   23:     //Static Methods
   24:     private static BinaryTree FromArrayRecurse(int[] arr, int index, Transform struc
tObject)
   25:     {
   26:         if (index >= arr.Length)
   27:             return null;
   28:         //Allow -1 in the array for variant structure arrays
   29:         if (arr[index] == -1)
   30:             return null;
   31:
   32:         BinaryTree root = (BinaryTree)Node.Create(arr[index], Node.CreationSelection
.BinaryTree);
   33:         root.transform.SetParent(structObject);
   34:
   35:         BinaryTree left = FromArrayRecurse(arr, index * 2 + 1, structObject);
   36:         BinaryTree right = FromArrayRecurse(arr, index * 2 + 2, structObject);
   37:
   38:         //Connect the nodes, and force the left/right connectedness.
   39:         root.SetRefSrc(Link.Connect(root, left), true);
   40:         root.SetRefSrc(Link.Connect(root, right), false);
   41:
   42:         return root;
   43:     }
   44:
   45:     private static void TreeProperties(BinaryTree root, ref int width, ref int maxDe
pth, int curDepth, List<BinaryTree> leaves)
   46:     {
   47:         //If the current depth is greater than maxDepth, update maxDepth
   48:         maxDepth = Mathf.Max(curDepth, maxDepth);
   49:
   50:         //Check if this is a leaf:
   51:         if(root.left == null && root.right == null)
   52:         {
   53:             leaves.Add(root);
   54:             width++;
   55:             return;
   56:         }
   57:
   58:         //Otherwise, recurse on the left and right
   59:         if(root.left != null)
   60:         {
   61:             TreeProperties(root.left, ref width, ref maxDepth, curDepth + 1, leaves)
;
   62:         }
```

```
   63:         if(root.right != null)
   64:         {
   65:             TreeProperties(root.right, ref width, ref maxDepth, curDepth + 1, leaves
);
   66:         }
   67:     }
   68:
   69:     private static void PositionRecurse(BinaryTree root, Vector3 localPosition, int
depth)
   70:     {
   71:         if (!root)
   72:             return;
   73:
   74:         root.transform.localPosition = localPosition;
   75:         Link.Reconnect(root); //reposition all the links as well
   76:         depth++;
   77:
   78:         Vector3 leftPos = localPosition + Vector3.down * depthFactor + Vector3.left
* (depthFactor / depth);
   79:         Vector3 rightPos = localPosition + Vector3.down * depthFactor + Vector3.righ
t * (depthFactor / depth);
   80:
   81:         PositionRecurse(root.left, leftPos, depth);
   82:         PositionRecurse(root.right, rightPos, depth);
   83:     }
   84:
   85:     public static BinaryTreeStructure FromArray(int[] arr)
   86:     {
   87:         return FromArray(arr, Vector3.zero);
   88:     }
   89:
   90:     public static BinaryTreeStructure FromArray(int[] arr, Vector3 basePosition)
   91:     {
   92:         if (arr.Length == 0)
   93:             return null;
   94:
   95:         GameObject structObject = new GameObject($"Binary Tree Structure {arr[0]}");
   96:         BinaryTreeStructure structure = structObject.AddComponent<BinaryTreeStructur
e>();
   97:
   98:         //Recursively create the binary tree from the array
   99:         BinaryTree root = FromArrayRecurse(arr, 0, structObject.transform);
  100:
  101:         //Assign References
  102:         structure.root = root;
  103:
  104:         //After the first recursive pass the nodes exist, but the structure's proper
ties are not recorded
  105:         int width = 0, depth = 0;
  106:         structure.leaves = new List<BinaryTree>();
  107:         TreeProperties(root, ref width, ref depth, 0, structure.leaves);
  108:
  109:         structure.width = width;
  110:         structure.depth = depth;
  111:
  112:         //After the second recursive pass, all the nodes are created, and the struct
ure component is properly configured, but the nodes are not positioned
  113:         PositionRecurse(root, basePosition, 0);
  114:
  115:         //To keep the structure above the floor, position the object in the air
  116:         structObject.transform.position += Vector3.up * (2 + depthFactor * depth);
  117:
  118:         Debug.Log($"Binary Tree created with width {structure.width}, depth {structu
re.depth}, and {structure.leaves.Count} leaves");
  119:
  120:         return structure;
  121:     }
  122:
```

```
  123:     private static void SetParentRecurse(BinaryTree root, Transform parent)
  124:     {
  125:         if (root == null)
  126:             return;
  127:
  128:         root.transform.SetParent(parent);
  129:
  130:         SetParentRecurse(root.left, parent);
  131:         SetParentRecurse(root.right, parent);
  132:     }
  133:
  134:     public static BinaryTreeStructure Organize(BinaryTree center)
  135:     {
  136:         //Find the head - and if there's a loop in the list, a structure can't be ma
de.
  137:         List<BinaryTree> visited = new List<BinaryTree>();
  138:         BinaryTree head = center;
  139:
  140:         while (head.prev)
  141:         {
  142:             head = (BinaryTree)head.prev;
  143:             if (visited.Contains(head))
  144:             {
  145:                 Debug.LogWarning($"Looping binary tree structure attempted.");
  146:                 return null;
  147:             }
  148:             visited.Add(head);
  149:         }
  150:
  151:         BinaryTree root = head;
  152:
  153:         //Once it is known to be a valid binary tree, can organize into a structure
  154:         GameObject structObject = new GameObject($"Binary Tree Structure {head.Value
}");
  155:         BinaryTreeStructure structure = structObject.AddComponent<BinaryTreeStructur
e>();
  156:         structObject.transform.position = root.transform.position;
  157:         root.transform.SetParent(structure.transform);
  158:         structure.root = root;
  159:
  160:         //Recursively set parents
  161:         SetParentRecurse(root, structure.transform);
  162:
  163:         //Recursively get tree properties
  164:         int width = 0, depth = 0;
  165:         structure.leaves = new List<BinaryTree>();
  166:         TreeProperties(root, ref width, ref depth, 0, structure.leaves);
  167:
  168:         structure.width = width;
  169:         structure.depth = depth;
  170:
  171:         //After the second recursive pass, all the nodes are created, and the struct
ure component is properly configured, but the nodes are not positioned
  172:         PositionRecurse(root, Vector3.zero, 0);
  173:
  174:         //To keep the structure above the floor, position the object in the air base
d on the min y position of the leaves.
  175:         float minPos = int.MaxValue;
  176:         foreach(BinaryTree leaf in structure.leaves)
  177:         {
  178:             minPos = Mathf.Min(minPos, leaf.transform.position.y);
  179:         }
  180:         float offset = (1 - minPos);
  181:         Debug.Log($"Adjusting y position of structure, based on a min position of {m
inPos}, to an offset of {offset}, to a final y of {structObject.transform.position.y + offs
et}");
  182:         structObject.transform.position += Vector3.up * offset;
  183:
```

```
  184:         Debug.Log($"Binary Tree created with width {structure.width}, depth {structu
re.depth}, and {structure.leaves.Count} leaves");
  185:
  186:         return structure;
  187:     }
  188: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: using System;
    6: using System.Linq;
    7:
    8: public class DigraphStructure : Structure
    9: {
   10:     //int[][] adjacencyMatrix; //maybe?
   11:     public List<Digraph> nodes;
   12:     public Queue<List<Digraph>> depthSets;
   13:     public Digraph center;
   14:
   15:     //Static
   16:     public enum DrawMode { Rings = 0, OneRing = 1, Branched = 2 };
   17:     public static DrawMode mode = DrawMode.Branched;
   18:
   19:     private static void LogDepthSets(Queue<List<Digraph>> depthSets)
   20:     {
   21:         //log all the depth sets
   22:         for (int i = 0; depthSets.Count > 0; i++)
   23:         {
   24:             List<Digraph> seti = depthSets.Dequeue();
   25:             Debug.Log($"Set {i} has {seti.Count} elements");
   26:         }
   27:     }
   28:
   29:     public static Vector3 radialDirection(float circlePercent, float radiansOffset,
int number, int index)
   30:     {
   31:         //Divide arc into number of portions
   32:         float baseAngle = (2 * Mathf.PI * circlePercent) / number;
   33:         //Return the corresponding angle
   34:         float thisAngle = baseAngle * index + radiansOffset;
   35:         //get vector x and y
   36:         float x = Mathf.Cos(thisAngle);
   37:         float y = Mathf.Sin(thisAngle);
   38:         Vector3 dir = new Vector3(x, y, 0);
   39:         //Debug.Log($"RD Dump: num {number} index {index} base {baseAngle * Mathf.Ra
d2Deg} this {thisAngle * Mathf.Rad2Deg} dir {x} , {y}");
   40:         return dir;
   41:     }
   42:
   43:     private static List<Digraph> drawVisited;
   44:     /// <summary>
   45:     /// Initially, this will be called on the center of the graph to organize, with
a arcLength of 360deg for the full circle, and an existing offset of zero degrees.
   46:     /// Depth set 0 is center
   47:     /// Depth set 1 gets branches, which occupy portions of the circle: eg: at 2 nod
es, each 'branch' gets 1/2 the circle, 3 = 1/3, etc.
   48:     /// Each depth sets first node, the root of the branch must be central to this n
ew sector of the circle
   49:     /// To generate the branch, the bounds for the sector need to be the base offset
 in radians, and the length of the sector, aka the remaining portion of the circle
   50:     /// Each branch is recursively partitioned into more branches by depth set.
   51:     /// </summary>
   52:     private static void RecursiveDraw(Digraph root, float arcLengthDeg, float thisOf
fsetDeg, bool offsetCount)
   53:     {
   54:         //Generate all nodes that can be reached from this one:
   55:         List<Digraph> next = new List<Digraph>();
   56:         foreach (Link l in root.connectionsTo)
   57:         {
   58:             Digraph dest = (Digraph)l.destination;
   59:             //if this is already a visited node, don't do anything
   60:             if (!drawVisited.Contains(dest))
   61:             {
```

```
   62:                 next.Add(dest);
   63:                 drawVisited.Add(dest);
   64:             }
   65:         }
   66:
   67:         //Then, position them and recurse:
   68:         int count = next.Count;
   69:
   70:         for (int i = 0; i < count; i++)
   71:         {
   72:             //If nodes get deleted, no need to position or recurse
   73:             if (!next[i])
   74:                 return;
   75:
   76:             //Generate direction to next
   77:             float circlePercent = arcLengthDeg / 360f;
   78:             float radiansOffset = thisOffsetDeg * Mathf.Deg2Rad;
   79:             //Debug.Log($"Generating direction from length {arcLengthDeg} and offset
 {thisOffsetDeg} to {circlePercent}, {radiansOffset}, {count}, {i}");
   80:             int countForCall = (offsetCount) ? count - 1 : count;
   81:             if (countForCall <= 0)
   82:                 countForCall = 1;
   83:             Vector3 direction = DigraphStructure.radialDirection(circlePercent, radi
ansOffset, countForCall, i);
   84:
   85:             //position this next node;
   86:             next[i].transform.position = root.transform.position + direction * 3;
   87:
   88:             //for the next node, call the same function, with 1/count of the arc len
gth
   89:             float nextArcLengthDeg = arcLengthDeg / count;
   90:             //but the offset should be a function of this arc length, the next arc l
ength, and the count but it should be off by half the arc
   91:             float nextOffsetDeg = (nextArcLengthDeg * i) - (nextArcLengthDeg * 0.5f)
;
   92:
   93:             //Debug.Log($"Next call will be from length {nextArcLengthDeg} and offse
t {nextOffsetDeg} at index {i}");
   94:             RecursiveDraw(next[i], nextArcLengthDeg, nextOffsetDeg, true);
   95:         }
   96:     }
   97:
   98:
   99:     public static DigraphStructure Organize(Digraph center)
  100:     {
  101:         //Performance Check
  102:         //Debug.Log($"Time: {Time.realtimeSinceStartup}");
  103:
  104:         List<Digraph> allNodes = new List<Digraph>();
  105:         Queue<List<Digraph>> depthSets = new Queue<List<Digraph>>();
  106:         List<Digraph> currentDepth = new List<Digraph>();
  107:
  108:         //first set should be depth set 0, with only the center element
  109:         currentDepth.Add(center);
  110:
  111:         while(currentDepth.Count != 0)
  112:         {
  113:             //enqueue a deep copy of the current depth list
  114:             depthSets.Enqueue(new List<Digraph>(currentDepth));
  115:
  116:             //generate next depth class from current
  117:             List<Digraph> nextDepth = new List<Digraph>();
  118:             foreach (Digraph dg in currentDepth)
  119:             {
  120:                 foreach (Link l in dg.connectionsTo)
  121:                 {
  122:                     Digraph dest = (Digraph)l.destination;
  123:                     //if this is already a visited node, don't do anything
```

```
  124:                     if (!allNodes.Contains(dest))
  125:                     {
  126:                         allNodes.Add(dest);
  127:                         nextDepth.Add(dest);
  128:                     }
  129:                 }
  130:             }
  131:             currentDepth = nextDepth;
  132:         }
  133:
  134:         if(mode == DrawMode.OneRing)
  135:         {
  136:             float baseRadius = 6f;
  137:             Vector3 basePosition = center.transform.position;
  138:             List<Digraph> nodesToDraw = new List<Digraph>(allNodes);
  139:             int c = nodesToDraw.Count;
  140:             for (int i = 0; i < nodesToDraw.Count; i++)
  141:             {
  142:                 Digraph dg = nodesToDraw[i];
  143:                 Vector3 offset = baseRadius * radialDirection(1, 0, c, i + 1);
  144:                 dg.transform.position = basePosition + offset;
  145:             }
  146:         }
  147:         else if(mode == DrawMode.Rings)
  148:         {
  149:             float baseRadius = 2f;
  150:             Vector3 basePosition = center.transform.position;
  151:             for (int setIndex = 0; depthSets.Count > 0; setIndex++)
  152:             {
  153:                 List<Digraph> set = depthSets.Dequeue();
  154:                 //Debug.Log($"Set: {string.Join(" ", set)}");
  155:                 int count = set.Count;
  156:                 for (int i = 0; i < count; i++)
  157:                 {
  158:                     Digraph dg = set[i];
  159:                     float radius = baseRadius * (setIndex + 1);
  160:                     Vector3 offset = radius * radialDirection(1, 0, count, i + 1);
  161:                     dg.transform.position = basePosition + offset;
  162:                 }
  163:             }
  164:             center.transform.position = basePosition;
  165:         }
  166:         else if(mode == DrawMode.Branched)
  167:         {
  168:             //clear list
  169:             drawVisited = new List<Digraph>();
  170:             //recursively draw
  171:             RecursiveDraw(center, 360, 0, false);
  172:         }
  173:
  174:         //center
  175:         allNodes.Insert(0, center);
  176:
  177:         //create structure object
  178:         GameObject structObject = new GameObject($"Digraph Structure {center.Value}"
);//ll.gameObject.AddComponent<LinkedListStructure>();
  179:         DigraphStructure dgs = structObject.AddComponent<DigraphStructure>();
  180:         dgs.transform.position = center.transform.position;
  181:         dgs.center = center;
  182:         dgs.nodes = allNodes;
  183:         dgs.depthSets = depthSets;
  184:
  185:         //tripurpose loop: parent, reconnect, and get min y position for structure o
ffset
  186:         float minPos = int.MaxValue;
  187:         foreach (Digraph d in allNodes)
  188:         {
  189:             Link.Reconnect(d);
```

```
  190:             d.transform.SetParent(dgs.transform);
  191:             minPos = Mathf.Min(minPos, d.transform.position.y);
  192:         }
  193:         float yOffset = (1 - minPos);
  194:         structObject.transform.position += Vector3.up * yOffset;
  195:
  196:         //Performance Check
  197:         //Debug.Log($"Time: {Time.realtimeSinceStartup}");
  198:
  199:         return dgs;
  200:
  201:
  202:     }
  203:
  204: }
  205:
  206: /*
  207:  * private static List<Digraph> drawVisited;
  208:     private static void RecursiveDraw(Digraph root, float sectorStart, float sectorL
ength)
  209:     {
  210:         //Generate all nodes that can be reached from this one:
  211:         List<Digraph> next = new List<Digraph>();
  212:         foreach (Link l in root.connectionsTo)
  213:         {
  214:             Digraph dest = (Digraph)l.destination;
  215:             //if this is already a visited node, don't do anything
  216:             if (!drawVisited.Contains(dest))
  217:             {
  218:                 next.Add(dest);
  219:                 drawVisited.Add(dest);
  220:             }
  221:         }
  222:
  223:         //Then, position them and recurse:
  224:         int count = next.Count;
  225:         for(int i = 0; i < count; i++)
  226:         {
  227:             Vector3 direction = radialDirection(sectorLength, sectorStart, count, i)
;
  228:             next[i].transform.position = root.transform.position + direction * 3.5f;
  229:
  230:             float nextLength = sectorLength / count;
  231:             //this is a little bit voodoo, but it's basically just recursively parti
tioning a circle into smaller segments and putting the different nodes aorund it
  232:             float nextStart = (sectorLength*i) + ((nextLength / -2.0f) + (nextLength
 * (i)));
  233:
  234:             //Debug.Log($"Placing {next[i].Value} with length {sectorLength}, offset
 {sectorStart} and index {i}");
  235:             //Debug.Log($"Drawing next {next[i].Value} with length {nextLength}, off
set {nextStart}");
  236:
  237:             RecursiveDraw(next[i], nextStart, nextLength);
  238:         }
  239:     }
  240:
  241:     */
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: /// <summary>
    6: /// This class is a container for the connections between nodes, which can be either
 directed or undirected. For nodes, they can be either on aesthetic preference. For graphs,
 the directed-ness of the link is functional.
    7: /// </summary>
    8: public class Link : MonoBehaviour
    9: {
   10:     //Non Static members
   11:     public Node source;
   12:     public Node destination;
   13:     private bool dir;
   14:     public bool Directed
   15:     {
   16:         get { return dir; }
   17:         set
   18:         {
   19:             dir = value;
   20:             setDirected(value);
   21:         }
   22:     }
   23:
   24:     private GameObject[] children = new GameObject[3];
   25:
   26:     //Non-static methods
   27:     public void InitChildren()
   28:     {
   29:         //Quick init
   30:         for (int i = 0; i < 3; i++)
   31:             children[i] = transform.GetChild(i).gameObject;
   32:     }
   33:
   34:     private void setDirected(bool b)
   35:     {
   36:         children[1].SetActive(b);
   37:         children[2].SetActive(b);
   38:     }
   39:
   40:     //Static members
   41:     public static GameObject linkReference;
   42:     private const float scaleFactor = 1f;
   43:     private static readonly Vector3 baseLineScale = new Vector3(0.4f, 0.4f, 6f);
   44:
   45:     //Static methods
   46:     public static void Init()
   47:     {
   48:         linkReference = Resources.Load("Other/Link") as GameObject;
   49:     }
   50:
   51:     private static void Align(Transform link, Transform src, Transform dest)
   52:     {
   53:         //Set Scale:
   54:         float length = Vector3.Distance(src.position, dest.position);
   55:         length *= scaleFactor;
   56:         Vector3 lineScale = new Vector3(0.4f, 0.4f, length);
   57:         float zPos = lineScale.z * 0.5f - 0.855f;
   58:         Link lc = link.GetComponent<Link>();
   59:         lc.children[0].transform.localScale = lineScale;
   60:         lc.children[1].transform.localPosition = new Vector3(0, 0.766f, zPos);
   61:         lc.children[2].transform.localPosition = new Vector3(0, -0.766f, zPos);
   62:         link.transform.localScale = new Vector3(0.5f, 0.5f, 0.9f);
   63:
   64:         //Position link
   65:         link.position = (src.position + dest.position) / 2; //average positions
   66:         link.LookAt(dest); //point to destination
```

```
    67:         link.SetParent(src); //parent to node
    68:     }
    69:
    70:     public static Link Connect(Node src, Node dest)
    71:     {
    72:         if (src == null || dest == null)
    73:             return null;
    74:
    75:         //Create new link
    76:         GameObject link = Instantiate(linkReference);
    77:         Link linkComponent = link.GetComponent<Link>();
    78:         linkComponent.InitChildren(); //assign child references for scaling/position
ing in future
    79:
    80:         //Link name is by UUID
    81:         string name = $"{src.GetInstanceID()} to {dest.GetInstanceID()}";
    82:         link.name = name;
    83:
    84:         //Check for edge cases (self-connection, existing connection)
    85:         if (src == dest)
    86:         {
    87:             //Left hand info - can't connect a node to itself!
    88:             Debug.LogWarning($"Warning: Self-connection attached with {src.name} id
{src.GetInstanceID()}");
    89:         }
    90:
    91:         if (src.transform.Find(name))
    92:         {
    93:             //Destroy existing link - this might want to be reconnect?
    94:             Destroy(src.transform.Find(name));
    95:         }
    96:
    97:         //Align (position + rotation) the link
    98:         Align(link.transform, src.transform, dest.transform);
    99:
   100:         //Assign references in link component
   101:         linkComponent.source = src;
   102:         linkComponent.destination = dest;
   103:         linkComponent.InitChildren();
   104:         linkComponent.Directed = true; //eventually needs to depend on type, setting
s
   105:
   106:         //Assign references in both nodes
   107:         src.AssignRefSrc(linkComponent);
   108:         dest.AssignRefDest(linkComponent);
   109:
   110:         //if this is between two undirected nodes, the connection should be undirect
ed, otherwise should be directed
   111:         linkComponent.setDirected(src.GetType() != typeof(Undigraph));
   112:
   113:         //Return reference
   114:         return linkComponent;
   115:     }
   116:
   117:     public static void Reconnect(Node src, Node dest)
   118:     {
   119:         //Find existing link by name
   120:         string name = $"{src.GetInstanceID()} to {dest.GetInstanceID()}";
   121:         GameObject link = src.transform.Find(name).gameObject;
   122:
   123:         //Align (position + rotation) the link
   124:         Align(link.transform, src.transform, dest.transform);
   125:     }
   126:
   127:     public static void Reconnect(Link link)
   128:     {
   129:         //Sometimes this will be passed a null link
   130:         if (!link)
```

```
  131:             return;
  132:
  133:         //Reconnect(link.source, link.destination);
  134:         //Align (position + rotation) the link
  135:         Align(link.transform, link.source.transform, link.destination.transform);
  136:     }
  137:
  138:     public static void Reconnect(Node node)
  139:     {
  140:         foreach (Link l in node.transform.GetComponentsInChildren<Link>())
  141:             Reconnect(l);
  142:
  143:         Reconnect(node.prevLink);
  144:     }
  145:
  146:     public static void Reconnect(GraphNode node)
  147:     {
  148:         foreach (Link l in node.connectionsTo)
  149:             Reconnect(l);
  150:         foreach (Link l in node.connectionsFrom)
  151:             Reconnect(l);
  152:     }
  153:
  154:     public static void Disconnect(Link link)
  155:     {
  156:         if (link == null)
  157:             return;
  158:
  159:         link.source.ClearRefSrc(link);
  160:         link.destination.ClearRefDest(link);
  161:
  162:         Destroy(link.gameObject);
  163:     }
  164:
  165:     //Flipping is different for binary trees, linked lists, and graphs
  166:     public static void Flip(Link link)
  167:     {
  168:         if (!link)
  169:         {
  170:             Debug.LogError($"Error: Link to flip is null.");
  171:             return;
  172:         }
  173:
  174:         //vary by type - yeah there's some clever way to do this probably but System
.Reflection has more overhead than it's worth here
  175:         if (link.source.GetType() == typeof(BinaryTree))
  176:             Flip(link, (BinaryTree)link.source, (BinaryTree)link.destination);
  177:         else if (link.source.GetType() == typeof(LinkedList))
  178:             Flip(link, (LinkedList)link.source, (LinkedList)link.destination);
  179:         else if (link.source.GetType() == typeof(Undigraph) || link.source.GetType()
 == typeof(Digraph))
  180:             Flip(link, (GraphNode)link.source, (GraphNode)link.destination);
  181:         else
  182:             Debug.LogError($"No flip method to call on {link.source.GetType().ToStri
ng()}");
  183:     }
  184:
  185:     private static void Swap(Link existing)
  186:     {
  187:         //swap link references:
  188:         var temp = existing.source;
  189:         existing.source = existing.destination;
  190:         existing.destination = temp;
  191:         string name = $"{existing.source.GetInstanceID()} to {existing.destination.G
etInstanceID()}"; //Make name function? Or make it rename on the getter?
  192:         existing.name = name;
  193:         //reparent
  194:         existing.transform.SetParent(existing.source.transform);
```

```
  195:         Debug.Log($"Node flipped to be {existing.source.Value} to {existing.destinat
ion.Value} with new name {name}");
  196:     }
  197:
  198:     /// <summary>
  199:     /// Flips the connection between two binary tree nodes.
  200:     /// </summary>
  201:     /// <param name="existing">existing link between the two</param>
  202:     /// <param name="child">current child (destination) BT node</param>
  203:     /// <param name="parent">current parent (source) BT node</param>
  204:     private static void Flip(Link existing, BinaryTree parent, BinaryTree child)
  205:     {
  206:         //theoretically, this could be a setting - which one it disconnects
  207:         //since a binary tree can only have one parent, if the parent (which becomes
 the child) already has a parent, just disconnect these two nodes:
  208:         if(parent.prevLink)
  209:         {
  210:             Debug.Log($"Disconnecting {existing.name} from {existing.source.name} to
 {existing.destination.name} because {parent.prevLink.name}");
  211:             //Log to LH info that this is invalid and that the nodes were disconnect
ed
  212:             Disconnect(existing);
  213:             return;
  214:         }
  215:
  216:         //Check which child node this is, and null it:
  217:         if(child == parent.left)
  218:         {
  219:             parent.left = null;
  220:             parent.leftLink = null;
  221:         }
  222:         else if(child == parent.right)
  223:         {
  224:             parent.right = null;
  225:             parent.rightLink = null;
  226:         }
  227:         else
  228:         {
  229:             Debug.LogError($"Somehow the child is not the left nor right of the pare
nt bt.");
  230:         }
  231:
  232:         //Otherwise, the binary tree has no existing parent, and this swap is valid
  233:         //swap link references:
  234:         Swap(existing);
  235:
  236:         //parent becomes the destination, child becomes the source
  237:         //assign references in parent
  238:         parent.prev = child;
  239:         parent.prevLink = existing;
  240:         //reassign in the child
  241:         child.AssignRefSrc(existing);
  242:         //null the old link
  243:         child.prev = null;
  244:         child.prevLink = null;
  245:         //handle the flipping visually
  246:         Reconnect(existing);
  247:     }
  248:
  249:     private static void Flip(Link existing, LinkedList src, LinkedList dest)
  250:     {
  251:         //Since a linked list can only have one inwards-facing connection, disconnec
t this if it exists
  252:         if (src.prevLink)
  253:         {
  254:             Debug.Log($"Disconnecting {existing.name} from {existing.source.name} to
 {existing.destination.name} because {src.prevLink.name}");
  255:             //Log to LH info that this is invalid and that the nodes were disconnect
```

```
ed
  256:             Disconnect(existing);
  257:             //Clear references
  258:             src.next = null;
  259:             src.nextLink = null;
  260:             dest.prev = null;
  261:             dest.prevLink = null;
  262:             return;
  263:         }
  264:
  265:         //Otherwise, the flip is valid
  266:         //Disconnect the existing link
  267:         if (dest.nextLink)
  268:         {
  269:             Disconnect(dest.nextLink);
  270:             dest.next = null;
  271:             dest.nextLink = null;
  272:         }
  273:
  274:         //Swap link references
  275:         Swap(existing);
  276:
  277:         //move values accordingly
  278:         src.next = null;
  279:         src.nextLink = null;
  280:         src.prev = dest;
  281:         src.prevLink = existing;
  282:
  283:         dest.next = src;
  284:         dest.nextLink = existing;
  285:         dest.prev = null;
  286:         dest.prevLink = null;
  287:
  288:         //Handle visually
  289:         Reconnect(existing);
  290:     }
  291:
  292:     private static void Flip(Link existing, GraphNode src, GraphNode dest)
  293:     {
  294:         //To flip, must remove this link from the source's connectionsTo, and from d
estination's connectionsFrom, then add vice versa, and realign
  295:         src.connectionsTo.Remove(existing);
  296:         dest.connectionsFrom.Remove(existing);
  297:
  298:         src.connectionsFrom.Add(existing);
  299:         dest.connectionsTo.Add(existing);
  300:
  301:         //Swap link references
  302:         Swap(existing);
  303:
  304:         //Handle visually
  305:         Reconnect(existing);
  306:
  307:     }
  308: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class LinkedListStructure : Structure
    6: {
    7:     public LinkedList head;
    8:     public LinkedList tail;
    9:     public int length;
   10:
   11:     //Default parameters aren't available with a non-constant compile time value - s
o having basePosition = (0,2,0) is not allowed :(
   12:     public static LinkedListStructure FromArray(int[] arr)
   13:     {
   14:         return FromArray(arr, Vector3.up * 2, 3);
   15:     }
   16:
   17:     public static LinkedListStructure FromArray(int[] arr, Vector3 basePosition)
   18:     {
   19:         return FromArray(arr, basePosition, 2);
   20:     }
   21:
   22:     public static LinkedListStructure FromArray(int[] arr, Vector3 basePosition, flo
at length)
   23:     {
   24:         GameObject structObject;
   25:
   26:         //Handle edge cases
   27:         if (arr.Length == 0)
   28:             return null;
   29:         if (arr.Length == 1)
   30:         {
   31:             LinkedList ll = (LinkedList)Node.Create(arr[0], 1);
   32:             structObject = new GameObject($"Linked List Structure {ll.Value}");//ll.
gameObject.AddComponent<LinkedListStructure>();
   33:             LinkedListStructure lls = structObject.AddComponent<LinkedListStructure>
();
   34:             lls.head = ll;
   35:             lls.tail = ll;
   36:             lls.length = 1;
   37:             ll.transform.SetParent(structObject.transform);
   38:             ll.transform.localPosition = Vector3.zero;
   39:             lls.transform.position = basePosition;
   40:             return lls;
   41:         }
   42:
   43:         //Create empty structure object for parenting; create head of the list
   44:         LinkedList head = (LinkedList)Node.Create(arr[0], 1);
   45:         structObject = new GameObject($"Linked List Structure {head.Value}");
   46:         LinkedListStructure structure = structObject.AddComponent<LinkedListStructur
e>();
   47:         structure.length = 1;
   48:
   49:         //First position
   50:         structObject.transform.position = basePosition;
   51:         head.transform.SetParent(structObject.transform);
   52:         head.transform.localPosition = Vector3.zero;
   53:
   54:
   55:         //Iterate through the array
   56:         LinkedList cur = head;
   57:         LinkedList next = null;
   58:
   59:
   60:         for (int i = 1; i < arr.Length; i++)
   61:         {
   62:             //Create & position next object
   63:             next = (LinkedList)Node.Create(arr[i], 1);
```

```
   64:             next.transform.SetParent(structObject.transform);
   65:             next.transform.localPosition = Vector3.right * i * length;
   66:             //connect
   67:             Link.Connect(cur, next);
   68:             structure.length++;
   69:             cur = next;
   70:         }
   71:
   72:         //Parent last object
   73:         //cur.transform.SetParent(structure.transform);
   74:
   75:         //Assign references
   76:         structure.head = head;
   77:         structure.tail = cur;
   78:
   79:         return structure;
   80:     }
   81:
   82:     public static LinkedListStructure Organize(LinkedList center)
   83:     {
   84:
   85:         //Find the head - and if there's a loop in the list, a structure can't be ma
de.
   86:         List<LinkedList> visited = new List<LinkedList>();
   87:         LinkedList head = center;
   88:
   89:         while(head.prev)
   90:         {
   91:             head = (LinkedList)head.prev;
   92:             if(visited.Contains(head))
   93:             {
   94:                 Debug.LogWarning($"Looping linked list structure attempted.");
   95:                 return null;
   96:             }
   97:             visited.Add(head);
   98:         }
   99:
  100:         //Once it is known that a loop does not exist, organize the actual structure
  101:         GameObject structObject = new GameObject($"Linked List Structure {head.Value
}");
  102:         LinkedListStructure structure = structObject.AddComponent<LinkedListStructur
e>();
  103:         structObject.transform.position = head.transform.position;//new Vector3(head
.transform.position.x, 1, head.transform.position.y);
  104:         structure.head = head;
  105:         head.transform.SetParent(structure.transform);
  106:         head.transform.localPosition = Vector3.zero;
  107:         structure.length = 0;
  108:
  109:
  110:         Vector3 pos = Vector3.zero;
  111:
  112:         LinkedList cur = head;
  113:
  114:         while(cur)
  115:         {
  116:             structure.tail = cur;
  117:             cur.transform.SetParent(structure.transform);
  118:             cur.transform.localPosition = pos;
  119:             Link.Reconnect(cur.prevLink);
  120:             Link.Reconnect(cur.nextLink);
  121:             cur = (LinkedList)cur.next;
  122:             pos += Vector3.right * 4;
  123:
  124:             structure.length++;
  125:         }
  126:
  127:         return structure;
```

```
128:     }
129: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: /// <summary>
    6: /// Structure is an abstract class which distinct structures will inherit from. It i
s intended as a container class for an entire 'structure' - a whole Binary Tree, a whole Li
nked List, etc.
    7: /// </summary>
    8: public abstract class Structure : MonoBehaviour
    9: {
   10:     //public abstract Node Create(Node start);
   11:
   12: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class UndigraphStructure : Structure
    6: {
    7:     //int[][] adjacencyMatrix; //maybe?
    8:     public List<Undigraph> nodes;
    9:     public Queue<List<Undigraph>> depthSets;
   10:     public Undigraph center;
   11:
   12:     //Static
   13:     public enum DrawMode { Rings = 0, OneRing = 1, Branched = 2 };
   14:     public static DrawMode mode = DrawMode.OneRing;
   15:
   16:     private static void LogDepthSets(Queue<List<Undigraph>> depthSets)
   17:     {
   18:         //log all the depth sets
   19:         for (int i = 0; depthSets.Count > 0; i++)
   20:         {
   21:             List<Undigraph> seti = depthSets.Dequeue();
   22:             Debug.Log($"Set {i} has {seti.Count} elements");
   23:         }
   24:     }
   25:
   26:     private static Vector3 radialDirection(float circlePercent, float radiansOffset,
 int number, int index)
   27:     {
   28:         //Divide arc into number of portions
   29:         float baseAngle = (2 * Mathf.PI * circlePercent) / number;
   30:         //Return the corresponding angle
   31:         float thisAngle = baseAngle * index + radiansOffset;
   32:         //get vector x and y
   33:         float x = Mathf.Cos(thisAngle);
   34:         float y = Mathf.Sin(thisAngle);
   35:         Vector3 dir = new Vector3(x, y);
   36:         //Debug.Log($"RD Dump: num {number} index {index} base {baseAngle} this {thi
sAngle} dir {dir.ToString()}");
   37:         return dir;
   38:     }
   39:
   40:     /* Fancy other drawmode:
   41:          * Depth set 0 is center
   42:          * Depth set 1 gets branches, which occupy portions of the circle: eg: at 2
nodes, each 'branch' gets 1/2 the circle, 3 = 1/3, etc.
   43:          * Each depth sets first node, the root of the branch must be central to thi
s new sector of the circle
   44:          * To generate the branch, the bounds for the sector need to be the base off
set in radians, and the length of the sector, aka the remaining portion of the circle
   45:          * Each branch is recursively partitioned into more branches by depth set.
   46:          */
   47:     private static List<Undigraph> drawVisited;
   48:     private static void RecursiveDraw(Undigraph root, float sectorStart, float secto
rLength)
   49:     {
   50:         //Generate all nodes that can be reached from this one:
   51:         List<Undigraph> next = new List<Undigraph>();
   52:         List<Link> aggregateConnections = new List<Link>();
   53:         aggregateConnections.AddRange(root.connectionsTo);
   54:         aggregateConnections.AddRange(root.connectionsFrom);
   55:
   56:         foreach (Link l in aggregateConnections)
   57:         {
   58:             Undigraph dest = (Undigraph)l.destination;
   59:             //if this is already a visited node, don't do anything
   60:             if (!drawVisited.Contains(dest))
   61:             {
   62:                 next.Add(dest);
```

```
   63:                 drawVisited.Add(dest);
   64:             }
   65:             dest = (Undigraph)l.source;
   66:             //if this is already a visited node, don't do anything
   67:             if (!drawVisited.Contains(dest))
   68:             {
   69:                 next.Add(dest);
   70:                 drawVisited.Add(dest);
   71:             }
   72:         }
   73:
   74:         //Then, position them and recurse:
   75:         int count = next.Count;
   76:         for (int i = 0; i < count; i++)
   77:         {
   78:             Vector3 direction = radialDirection(sectorLength, sectorStart, count, i)
;
   79:             next[i].transform.position = root.transform.position + direction * 3.5f;
   80:
   81:             float nextLength = sectorLength / count;
   82:             //this is a little bit voodoo, but it's basically just recursively parti
tioning a circle into smaller segments and putting the different nodes aorund it
   83:             float nextStart = (sectorLength * i) + ((nextLength / -2.0f) + (nextLeng
th * (i)));
   84:
   85:             //Debug.Log($"Placing {next[i].Value} with length {sectorLength}, offset
 {sectorStart} and index {i}");
   86:             //Debug.Log($"Drawing next {next[i].Value} with length {nextLength}, off
set {nextStart}");
   87:
   88:             RecursiveDraw(next[i], nextStart, nextLength);
   89:         }
   90:     }
   91:
   92:
   93:     public static UndigraphStructure Organize(Undigraph center)
   94:     {
   95:         //Performance Check
   96:         //Debug.Log($"Time: {Time.realtimeSinceStartup}");
   97:
   98:         List<Undigraph> allNodes = new List<Undigraph>();
   99:         Queue<List<Undigraph>> depthSets = new Queue<List<Undigraph>>();
  100:         List<Undigraph> currentDepth = new List<Undigraph>();
  101:
  102:         //first set should be depth set 0, with only the center element
  103:         currentDepth.Add(center);
  104:
  105:         while (currentDepth.Count != 0)
  106:         {
  107:             //enqueue a deep copy of the current depth list
  108:             depthSets.Enqueue(new List<Undigraph>(currentDepth));
  109:
  110:             //generate next depth class from current
  111:             List<Undigraph> nextDepth = new List<Undigraph>();
  112:             foreach (Undigraph ug in currentDepth)
  113:             {
  114:                 List<Link> aggregateConnections = new List<Link>();
  115:                 aggregateConnections.AddRange(ug.connectionsTo);
  116:                 aggregateConnections.AddRange(ug.connectionsFrom);
  117:
  118:                 foreach (Link l in aggregateConnections)
  119:                 {
  120:                     Undigraph dest = (Undigraph)l.destination;
  121:                     //if this is already a visited node, don't do anything
  122:                     if (!allNodes.Contains(dest))
  123:                     {
  124:                         allNodes.Add(dest);
  125:                         nextDepth.Add(dest);
```

```
126:                     }
127:                     dest = (Undigraph)l.source;
128:                     //if this is already a visited node, don't do anything
129:                     if (!allNodes.Contains(dest))
130:                     {
131:                         allNodes.Add(dest);
132:                         nextDepth.Add(dest);
133:                     }
134:                 }
135:             }
136:             currentDepth = nextDepth;
137:         }
138:
139:         if (mode == DrawMode.OneRing)
140:         {
141:             float baseRadius = 6f;
142:             Vector3 basePosition = center.transform.position;
143:             List<Undigraph> nodesToDraw = new List<Undigraph>(allNodes);
144:             int c = nodesToDraw.Count;
145:             for (int i = 0; i < nodesToDraw.Count; i++)
146:             {
147:                 Undigraph dg = nodesToDraw[i];
148:                 Vector3 offset = baseRadius * radialDirection(1, 0, c, i + 1);
149:                 dg.transform.position = basePosition + offset;
150:             }
151:         }
152:         else if (mode == DrawMode.Rings)
153:         {
154:             float baseRadius = 2f;
155:             Vector3 basePosition = center.transform.position;
156:             for (int setIndex = 0; depthSets.Count > 0; setIndex++)
157:             {
158:                 List<Undigraph> set = depthSets.Dequeue();
159:                 //Debug.Log($"Set: {string.Join(" ", set)}");
160:                 int count = set.Count;
161:                 for (int i = 0; i < count; i++)
162:                 {
163:                     Undigraph dg = set[i];
164:                     float radius = baseRadius * (setIndex + 1);
165:                     Vector3 offset = radius * radialDirection(1, 0, count, i + 1);
166:                     dg.transform.position = basePosition + offset;
167:                 }
168:             }
169:             center.transform.position = basePosition;
170:         }
171:         else if (mode == DrawMode.Branched)
172:         {
173:             //clear list
174:             drawVisited = new List<Undigraph>();
175:             //recursively draw
176:             RecursiveDraw(center, 0, 1);
177:         }
178:
179:         //center
180:         allNodes.Insert(0, center);
181:
182:         foreach (Undigraph d in allNodes)
183:             Link.Reconnect(d);
184:
185:         //create structure object
186:         GameObject structObject = new GameObject($"Undigraph Structure {center.Value
}");//ll.gameObject.AddComponent<LinkedListStructure>();
187:         UndigraphStructure dgs = structObject.AddComponent<UndigraphStructure>();
188:         dgs.transform.position = center.transform.position;
189:         dgs.center = center;
190:         dgs.nodes = allNodes;
191:         dgs.depthSets = depthSets;
192:
```

```
  193:         //Loop has two uses: set parent in transform, and check if any part of the s
tructure is below the ground
  194:         float minPos = int.MaxValue;
  195:         foreach (Node n in allNodes)
  196:         {
  197:             n.transform.SetParent(dgs.transform);
  198:             minPos = Mathf.Min(minPos, n.transform.position.y);
  199:         }
  200:         float yOffset = (1 - minPos);
  201:         structObject.transform.position += Vector3.up * yOffset;
  202:
  203:         //Performance Check
  204:         //Debug.Log($"Time: {Time.realtimeSinceStartup}");
  205:
  206:         return dgs;
  207:
  208:
  209:     }
  210: }
```

```
    1: ï»¿using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4: using UnityEngine.UI;
    5: using TMPro;
    6:
    7: public class DemoStructures : MonoBehaviour
    8: {
    9:     //Structure Creation:
   10:     public enum NodeType { BinaryTree = 0, LinkedList = 1, Digraph = 2, Undigraph =
3 };
   11:     public static int[] StructCount = { 6, 4, 3, 2 };
   12:
   13:     public static GameObject CreateDemo(NodeType type, int index)
   14:     {
   15:         if (index > 5 || index < 0)
   16:             return null;
   17:         if (index >= StructCount[(int)type])
   18:             return null;
   19:         Debug.Log($"Creating structure of type {type.ToString()} index {index} again
st {StructCount[(int)type]}");
   20:         switch (type)
   21:         {
   22:
   23:             case NodeType.BinaryTree:
   24:                 return CreateBT(index);
   25:             case NodeType.LinkedList:
   26:                 return CreateLL(index);
   27:             case NodeType.Digraph:
   28:                 return CreateDG(index);
   29:             case NodeType.Undigraph:
   30:                 return CreateUG(index);
   31:             default:
   32:                 Debug.LogError("Type missing in demo creation");
   33:                 return null;
   34:         }
   35:     }
   36:
   37:
   38:     private static GameObject CreateLL(int index)
   39:     {
   40:         int[] demo = LinkedListArrays[index];
   41:         GameObject newStruct = LinkedListStructure.FromArray(demo).gameObject;
   42:         newStruct.AddComponent<DemoStructureMarker>();
   43:         return newStruct;
   44:     }
   45:
   46:     private static GameObject CreateBT(int index)
   47:     {
   48:         int[] demo = BinaryTreeArrays[index];
   49:         GameObject newStruct = BinaryTreeStructure.FromArray(demo).gameObject;
   50:         newStruct.AddComponent<DemoStructureMarker>();
   51:         return newStruct;
   52:     }
   53:     private static GameObject CreateDG(int index)
   54:     {
   55:         CreateDigraph demo = DigraphDemos[index];
   56:         GameObject newStruct = demo().gameObject;
   57:         newStruct.AddComponent<DemoStructureMarker>();
   58:         return newStruct;
   59:     }
   60:     private static GameObject CreateUG(int index)
   61:     {
   62:         CreateUndigraph demo = UndigraphDemos[index];
   63:         GameObject newStruct = demo().gameObject;
   64:         newStruct.AddComponent<DemoStructureMarker>();
   65:         return newStruct;
   66:     }
```

```
   67:
   68:     //Demo Structures Below:
   69:
   70:     //FROM ARRAY:
   71:
   72:     //LinkedLists
   73:
   74:     private static int[] ldemo0 = { 1, 2, 3, 4, 5 }; // Creates a list of numbers 1
- 5.
   75:     private static int[] ldemo1 = { 5, 4, 3, 2, 1 }; // Creates a list of numbers 5
- 1.
   76:     private static int[] ldemo2 = { 2, 3, 5, 7, 11, 13, 17 }; // Creates a list of t
he first seven primes.
   77:     private static int[] ldemo3 = { 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3,
2, 3, 8, 4 }; // Creates a list of the first 20 digits of pi.
   78:     private static int[][] lldemos = { ldemo0, ldemo1, ldemo2, ldemo3 };
   79:     static List<int[]> LinkedListArrays = new List<int[]>(lldemos);
   80:
   81:     //Binary Trees
   82:     private static int[] bdemo0 = { 1, 2, 3, 4, 5 }; //Creates a full binary tree wi
th five nodes.
   83:     private static int[] bdemo1 = { 1, 2, 3, 4, 5, 6 }; //Creates a complete binary
tree with six nodes.
   84:     private static int[] bdemo2 = { 1, 2, 3, 4, 5, 6, 7 }; //Creates a perfect [full
 & complete] binary tree with seven nodes.
   85:     private static int[] bdemo3 = { 10, 7, 13, 5, 9, 11, 15 }; //Creates a model bin
ary search tree with seven nodes.
   86:     private static int[] bdemo4 = { 1, 2, -1, 3, -1, -1, -1, 4 }; //Creates a left p
athological tree with four nodes.
   87:     private static int[] bdemo5 = { 1, -1, 2, -1, -1, -1, 3, -1, -1, -1, -1, -1, -1,
 -1, 4}; //Creates a right pathological tree with four nodes.
   88:     private static int[][] btdemos = { bdemo0, bdemo1, bdemo2, bdemo3, bdemo4, bdemo
5 };
   89:     static List<int[]> BinaryTreeArrays = new List<int[]>(btdemos);
   90:
   91:     //Digraphs
   92:     private delegate Structure CreateDigraph();
   93:
   94:     //Creates a triangular strongly connected directed graph.
   95:     private static Structure ddemo0()
   96:     {
   97:         GameObject structObject = new GameObject($"Demo Digraph Structure 3 Nodes");
   98:         DigraphStructure dgs = structObject.AddComponent<DigraphStructure>();
   99:
  100:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
  101:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
  102:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
  103:         Link.Connect(a, b);
  104:         Link.Connect(b, c);
  105:         Link.Connect(c, a);
  106:         a.transform.SetParent(dgs.transform);
  107:         b.transform.SetParent(dgs.transform);
  108:         c.transform.SetParent(dgs.transform);
  109:         a.transform.localPosition = Vector3.zero + Vector3.up * 3 + Vector3.left * 3
;
  110:         b.transform.localPosition = Vector3.zero + Vector3.up * 3 + Vector3.right *
3;
  111:         c.transform.localPosition = Vector3.zero + Vector3.down * 3;
  112:         Link.Reconnect(a);
  113:         Link.Reconnect(b);
  114:         Link.Reconnect(c);
  115:         return dgs;
  116:     }
  117:
  118:     //Creates a pentagonal strongly connected directed graph.
  119:     private static Structure ddemo1()
  120:     {
  121:         GameObject structObject = new GameObject($"Demo Digraph Structure 5 Nodes");
```

```
  122:         DigraphStructure dgs = structObject.AddComponent<DigraphStructure>();
  123:
  124:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
  125:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
  126:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
  127:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
  128:         Digraph e = (Digraph)Node.Create(5, Node.CreationSelection.Digraph);
  129:         Link.Connect(a, b);
  130:         Link.Connect(b, c);
  131:         Link.Connect(c, d);
  132:         Link.Connect(d, e);
  133:         Link.Connect(e, a);
  134:         a.transform.SetParent(dgs.transform);
  135:         b.transform.SetParent(dgs.transform);
  136:         c.transform.SetParent(dgs.transform);
  137:         d.transform.SetParent(dgs.transform);
  138:         e.transform.SetParent(dgs.transform);
  139:         a.transform.localPosition = Vector3.zero + Vector3.up*3;
  140:         b.transform.localPosition = Vector3.zero + Vector3.right * 3;
  141:         c.transform.localPosition = Vector3.zero + Vector3.down * 3 + Vector3.right
* 3;
  142:         d.transform.localPosition = Vector3.zero + Vector3.down * 3 + Vector3.left *
 3;
  143:         e.transform.localPosition = Vector3.zero + Vector3.left * 3;
  144:         Link.Reconnect(a);
  145:         Link.Reconnect(b);
  146:         Link.Reconnect(c);
  147:         Link.Reconnect(d);
  148:         Link.Reconnect(e);
  149:         return dgs;
  150:     }
  151:
  152:     //Creates a directed graph that branches out circularly.
  153:     private static Structure ddemo2()
  154:     {
  155:         List<Digraph> digraphs = new List<Digraph>();
  156:         for (int i = 0; i <= 12; i++)
  157:             digraphs.Add((Digraph)Node.Create(i, Node.CreationSelection.Digraph));
  158:
  159:         for (int i = 1; i < 4; i++)
  160:         {
  161:             Link.Connect(digraphs[0], digraphs[i]);
  162:             for (int j = 0; j < 3; j++)
  163:                 Link.Connect(digraphs[i], digraphs[3 * i + j + 1]);
  164:         }
  165:
  166:         return DigraphStructure.Organize(digraphs[0]);
  167:     }
  168:
  169:     private static CreateDigraph[] ddemos = { ddemo0, ddemo1, ddemo2 };
  170:     private static List<CreateDigraph> DigraphDemos = new List<CreateDigraph>(ddemos
);
  171:
  172:     //Undigraphs
  173:     private delegate Structure CreateUndigraph();
  174:     private static CreateUndigraph[] udemos = {udemo0, udemo1};
  175:     private static List<CreateUndigraph> UndigraphDemos = new List<CreateUndigraph>(
udemos);
  176:
  177:     //Creates an undirected graph that represents the hasse diagram of D6.
  178:     private static Structure udemo0()
  179:     {
  180:         Undigraph a = (Undigraph)Node.Create(1, Node.CreationSelection.Undigraph);
  181:         Undigraph b = (Undigraph)Node.Create(2, Node.CreationSelection.Undigraph);
  182:         Undigraph c = (Undigraph)Node.Create(3, Node.CreationSelection.Undigraph);
  183:         Undigraph d = (Undigraph)Node.Create(6, Node.CreationSelection.Undigraph);
  184:
  185:         GameObject structObject = new GameObject($"Demo Undigraph Structure D6");
```

```
186:         UndigraphStructure ugs = structObject.AddComponent<UndigraphStructure>();
187:         ugs.transform.position = d.transform.position;
188:
189:         a.transform.SetParent(ugs.transform);
190:         b.transform.SetParent(ugs.transform);
191:         c.transform.SetParent(ugs.transform);
192:         d.transform.SetParent(ugs.transform);
193:
194:         a.transform.localPosition = Vector3.down * 4;
195:         b.transform.localPosition = Vector3.down * 2 + Vector3.left * 2;
196:         c.transform.localPosition = Vector3.down * 2 + Vector3.right * 2;
197:         d.transform.localPosition = Vector3.zero;
198:
199:         Link.Connect(a, b);
200:         Link.Connect(a, c);
201:         Link.Connect(b, d);
202:         Link.Connect(c, d);
203:
204:         return ugs;
205:     }
206:
207:     //Creates an undirected graph that represents the hasse diagram of D30.
208:     private static Structure udemo1()
209:     {
210:         Undigraph a = (Undigraph)Node.Create(1, Node.CreationSelection.Undigraph);
211:         Undigraph b = (Undigraph)Node.Create(2, Node.CreationSelection.Undigraph);
212:         Undigraph c = (Undigraph)Node.Create(3, Node.CreationSelection.Undigraph);
213:         Undigraph d = (Undigraph)Node.Create(5, Node.CreationSelection.Undigraph);
214:         Undigraph e = (Undigraph)Node.Create(6, Node.CreationSelection.Undigraph);
215:         Undigraph f = (Undigraph)Node.Create(10, Node.CreationSelection.Undigraph);
216:         Undigraph g = (Undigraph)Node.Create(15, Node.CreationSelection.Undigraph);
217:         Undigraph h = (Undigraph)Node.Create(30, Node.CreationSelection.Undigraph);
218:
219:         GameObject structObject = new GameObject($"Demo Undigraph Structure D6");
220:         UndigraphStructure ugs = structObject.AddComponent<UndigraphStructure>();
221:         ugs.transform.position = d.transform.position;
222:
223:         a.transform.SetParent(ugs.transform);
224:         b.transform.SetParent(ugs.transform);
225:         c.transform.SetParent(ugs.transform);
226:         d.transform.SetParent(ugs.transform);
227:         e.transform.SetParent(ugs.transform);
228:         f.transform.SetParent(ugs.transform);
229:         g.transform.SetParent(ugs.transform);
230:         h.transform.SetParent(ugs.transform);
231:
232:         //30
233:         h.transform.localPosition = Vector3.zero;
234:         //6,10,15
235:         e.transform.localPosition = Vector3.down * 2 + Vector3.left * 2;
236:         f.transform.localPosition = Vector3.down * 2;
237:         g.transform.localPosition = Vector3.down * 2 + Vector3.right * 2;
238:         //2,3,5
239:         b.transform.localPosition = Vector3.down * 4 + Vector3.left * 2;
240:         c.transform.localPosition = Vector3.down * 4;
241:         d.transform.localPosition = Vector3.down * 4 + Vector3.right * 2;
242:         //1
243:         a.transform.localPosition = Vector3.down * 6;
244:
245:         Link.Connect(a, b);
246:         Link.Connect(a, c);
247:         Link.Connect(a, d);
248:
249:         Link.Connect(b, e);
250:         Link.Connect(b, f);
251:         Link.Connect(c, e);
252:         Link.Connect(c, g);
253:         Link.Connect(d, f);
```

```
254:         Link.Connect(d, g);
255:
256:         Link.Connect(e, h);
257:         Link.Connect(f, h);
258:         Link.Connect(g, h);
259:
260:         return ugs;
261:     }
262: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class MainMenuVisuals : MonoBehaviour
    6: {
    7:     GameObject btObj, llObj;
    8:     BinaryTreeStructure bt;
    9:     LinkedListStructure ll;
   10:
   11:     private void Start()
   12:     {
   13:         //Create binary tree and linked list:
   14:         btObj = DemoStructures.CreateDemo(DemoStructures.NodeType.BinaryTree, 2);
   15:         llObj = DemoStructures.CreateDemo(DemoStructures.NodeType.LinkedList, 0);
   16:         bt = btObj.GetComponent<BinaryTreeStructure>();
   17:         ll = llObj.GetComponent<LinkedListStructure>();
   18:
   19:         //Move objects
   20:         btObj.transform.position = new Vector3(-15, 10.5f, 4.15f);
   21:         btObj.transform.eulerAngles = new Vector3(0, -45, 0);
   22:         llObj.transform.position = new Vector3(15.5f, 4.74f, 8.66f);
   23:         llObj.transform.eulerAngles = new Vector3(0, 45, 0);
   24:
   25:         //modified algorithm setup
   26:         Algorithms ac = GetComponent<Algorithms>();
   27:         Algorithms.iterationMode = Algorithms.IterationMode.Wait;
   28:
   29:         //Run visual routines in parallel
   30:         StartCoroutine(VisualBT(bt));
   31:         StartCoroutine(VisualLL(ll));
   32:
   33:         //Stop player movement
   34:         Tutorial.restrictMovement = true;
   35:
   36:     }
   37:
   38:     private IEnumerator VisualBT(BinaryTreeStructure bts)
   39:     {
   40:         Algorithms ac = GetComponent<Algorithms>();
   41:         while (true)
   42:         {
   43:             yield return StartCoroutine(ac.VisualLeveledTraverse(bts.root));
   44:             yield return new WaitForEndOfFrame();
   45:         }
   46:     }
   47:
   48:     private IEnumerator VisualLL(LinkedListStructure lls)
   49:     {
   50:         Algorithms ac = GetComponent<Algorithms>();
   51:         while (true)
   52:         {
   53:             yield return StartCoroutine(ac.VisualReverse(lls));
   54:             yield return new WaitForEndOfFrame();
   55:         }
   56:     }
   57:
   58:     private void Update()
   59:     {
   60:         //Debug.Log($"Setting rotation with counts {ll.GetComponentsInChildren<Spinn
er>().Length} and {bt.GetComponentsInChildren<Spinner>().Length}");
   61:         foreach (Select s in ll.GetComponentsInChildren<Select>())
   62:         {
   63:             s.transform.eulerAngles = Vector3.zero;
   64:             s.transform.localEulerAngles = Vector3.zero;
   65:             //Debug.Log($"select: {s.transform.rotation} {s.transform.eulerAngles} {
s.transform.eulerAngles}");
   66:         }
```

```
   67:         foreach (Select s in bt.GetComponentsInChildren<Select>())
   68:         {
   69:             s.transform.eulerAngles = Vector3.zero;
   70:             s.transform.localEulerAngles = Vector3.zero;
   71:             //Debug.Log($"select: {s.transform.rotation} {s.transform.eulerAngles} {
s.transform.eulerAngles}");
   72:         }
   73:     }
   74: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class ScreenshotHelper : MonoBehaviour
    6: {
    7:     public GameObject leftHand, rightHand;
    8:     public LineRenderer leftLR, rightLR;
    9:
   10:     public Vector3 basePosition, targetPosition;
   11:     public Transform baseAsTransform, targetAsTransform;
   12:     public bool left;
   13:     public string filename;
   14:
   15:
   16:     // Start is called before the first frame update
   17:     void Start()
   18:     {
   19:
   20:     }
   21:
   22:     // Update is called once per frame
   23:     void Update()
   24:     {
   25:         if (Input.GetKeyDown(KeyCode.K))
   26:             ScreenshotTest();
   27:     }
   28:
   29:     public void ScreenshotTest()
   30:     {
   31:         ScreenCapture.CaptureScreenshot("testScreenshot1.png");
   32:         //ScreenCapture.CaptureScreenshot("testScreenshot2", 2);
   33:     }
   34:
   35:     public void Capture(string fname)
   36:     {
   37:         ScreenCapture.CaptureScreenshot(fname);
   38:     }
   39:
   40:     public void MoveHand(GameObject hand, Vector3 position)
   41:     {
   42:         hand.transform.position = position;
   43:     }
   44:
   45:     public void PointHand(GameObject hand, Vector3 position)
   46:     {
   47:         hand.transform.LookAt(position);
   48:     }
   49:
   50:     public void DrawLine(LineRenderer lr, Vector3 start, Vector3 end)
   51:     {
   52:         Vector3[] arr = { start, end };
   53:         lr.SetPositions(arr);
   54:     }
   55: }
```

```
1: ï»¿using System.Collections;
2: using System.Collections.Generic;
3: using UnityEngine;
4:
5: public class StructureManager : MonoBehaviour
6: {
7:     //lots of stuff to do here also
8: }
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

/// <summary>
/// Class used for unit testing and creation of demo structures.
/// </summary>
public class UnitTesting : MonoBehaviour
{
    //Sample Structures

    //Unit Testing:
    private delegate void TestMethod();
    private List<TestMethod> unitTests = new List<TestMethod>();

    private List<IEnumerator> testCoroutines = new List<IEnumerator>();

    private bool tested;

    //For multiple unit tests
    private Vector3 basePosition;

    //components for testing
    Actions actionsComponent;
    UIManager uIManagerComponent;
    private void Start()
    {
        actionsComponent = GetComponent<Actions>();
        uIManagerComponent = GetComponentInChildren<UIManager>();

        basePosition = new Vector3(0, 3, 2);

        tested = false;

        //scene setup
        Tutorial.restrictInfo = false;
        Tutorial.restrictMovement = false;
        Tutorial.restrictPickupInfo = false;
        Tutorial.restrictCSToggle = false;

        //unitTests.Add(TestRadialPositioning2);
        //unitTests.Add(TestRadialPositioning3);
        //unitTests.Add(TestDGOrganizationBranched1);

        //Algorithm Testing
        //testCoroutines.Add(TestLLRCoR());
        //testCoroutines.Add(TestLLShift());
        //testCoroutines.Add(TestBFS1());
        //testCoroutines.Add(TestDFS1());

        //testCoroutines.Add(TestUBFS1());
        //testCoroutines.Add(TestUDFS1());
    }

    private void RunTests()
    {
        foreach (TestMethod test in unitTests)
            test();

        foreach (IEnumerator test in testCoroutines)
            StartCoroutine(test);
    }

    private void Update()
    {
        //release cursor for testing
        if (Input.GetKeyDown(KeyCode.L))
```

```
   69:             Cursor.lockState = CursorLockMode.Locked;
   70:         if (Input.GetKeyDown(KeyCode.Semicolon))
   71:             Cursor.lockState = CursorLockMode.None;
   72:
   73:
   74:         //In update so it's after all awake/start methods
   75:         if(!tested)
   76:         {
   77:             tested = true;
   78:             RunTests();
   79:             Debug.Log($"End of Tests Time: {Time.realtimeSinceStartup}");
   80:             return;
   81:         }
   82:     }
   83:
   84:     private void TestDGOrganizationBranched1()
   85:     {
   86:         List<Digraph> digraphs = new List<Digraph>();
   87:         for (int i = 0; i <= 12; i++)
   88:             digraphs.Add((Digraph)Node.Create(i, Node.CreationSelection.Digraph));
   89:
   90:         for (int i = 1; i < 4; i++)
   91:         {
   92:             Link.Connect(digraphs[0], digraphs[i]);
   93:             for (int j = 0; j < 3; j++)
   94:                 Link.Connect(digraphs[i], digraphs[3 * i + j + 1]);
   95:         }
   96:
   97:         DigraphStructure.Organize(digraphs[0]);
   98:     }
   99:
  100:     private void TestRadialPositioning3()
  101:     {
  102:         List<Digraph> digraphs = new List<Digraph>();
  103:         for (int i = 0; i <= 12; i++)
  104:             digraphs.Add((Digraph)Node.Create(i, Node.CreationSelection.Digraph));
  105:
  106:         for (int i = 1; i < 4; i++)
  107:         {
  108:             Link.Connect(digraphs[0], digraphs[i]);
  109:             for (int j = 0; j < 3; j++)
  110:                 Link.Connect(digraphs[i], digraphs[3 * i + j + 1]);
  111:         }
  112:
  113:         drawVisited = new List<Digraph>();
  114:         RecursiveDraw(digraphs[0], 360, 0, false);
  115:
  116:         foreach (Digraph d in digraphs)
  117:             Link.Reconnect(d);
  118:     }
  119:
  120:
  121:     private static List<Digraph> drawVisited;
  122:     /// <summary>
  123:     /// Initially, this will be called on the center of the graph to organize, with
a arcLength of 360deg for the full circle, and an existing offset of zero degrees.
  124:     /// </summary>
  125:     /// <param name="root"></param>
  126:     /// <param name="arcLengthDeg"></param>
  127:     /// <param name="thisOffsetDeg"></param>
  128:     private void RecursiveDraw(Digraph root, float arcLengthDeg, float thisOffsetDeg
, bool offsetCount)
  129:     {
  130:         //Generate all nodes that can be reached from this one:
  131:         List<Digraph> next = new List<Digraph>();
  132:         foreach (Link l in root.connectionsTo)
  133:         {
  134:             Digraph dest = (Digraph)l.destination;
```

```
  135:             //if this is already a visited node, don't do anything
  136:             if (!drawVisited.Contains(dest))
  137:             {
  138:                 next.Add(dest);
  139:                 drawVisited.Add(dest);
  140:             }
  141:         }
  142:
  143:         //Then, position them and recurse:
  144:         int count = next.Count;
  145:
  146:         for(int i = 0; i < count; i++)
  147:         {
  148:             //Generate direction to next
  149:             float circlePercent = arcLengthDeg / 360f;
  150:             float radiansOffset = thisOffsetDeg * Mathf.Deg2Rad;
  151:             Debug.Log($"Generating direction from length {arcLengthDeg} and offset {
thisOffsetDeg} to {circlePercent}, {radiansOffset}, {count}, {i}");
  152:             int countForCall = (offsetCount) ? count - 1 : count;
  153:             Vector3 direction = DigraphStructure.radialDirection(circlePercent, radi
ansOffset, countForCall, i);
  154:
  155:             //position this next node;
  156:             next[i].transform.position = root.transform.position + direction * 3;
  157:
  158:             //for the next node, call the same function, with 1/count of the arc len
gth
  159:             float nextArcLengthDeg = arcLengthDeg / count;
  160:             //but the offset should be a function of this arc length, the next arc l
ength, and the count but it should be off by half the arc
  161:             float nextOffsetDeg = (nextArcLengthDeg * i) - (nextArcLengthDeg*0.5f);
  162:
  163:             Debug.Log($"Next call will be from length {nextArcLengthDeg} and offset
{nextOffsetDeg} at index {i}");
  164:             RecursiveDraw(next[i], nextArcLengthDeg, nextOffsetDeg, true);
  165:         }
  166:     }
  167:
  168:     private void TestRadialPositioning2()
  169:     {
  170:         List<Digraph> digraphs = new List<Digraph>();
  171:         for (int i = 0; i <= 12; i++)
  172:             digraphs.Add((Digraph)Node.Create(i, Node.CreationSelection.Digraph));
  173:
  174:         digraphs[0].transform.position = Vector3.zero;
  175:         //position first ring;
  176:         for(int i = 0; i < 3; i++)
  177:         {
  178:             //calculate direction
  179:             Vector3 direction = DigraphStructure.radialDirection(1, 0, 3, i);
  180:             digraphs[i + 1].transform.position = direction * 3;
  181:
  182:             float nextArcDeg = 120f; // 360 deg in circle / 3
  183:
  184:             for (int j = 0; j < 3; j++)
  185:             {
  186:                 float baseOffset = nextArcDeg * i * Mathf.Deg2Rad;
  187:                 float thisArc = nextArcDeg; //* 0.75f;
  188:                 float arcOffset = nextArcDeg * Mathf.Deg2Rad * -1 / 2f;
  189:                 Vector3 secondDir = DigraphStructure.radialDirection(nextArcDeg / 36
0f, baseOffset + arcOffset, 2, j);
  190:
  191:                 digraphs[(3*(i+1)) + j + 1].transform.position = digraphs[i + 1].tra
nsform.position + secondDir * 3;
  192:             }
  193:
  194:         }
  195:
```

```
  196:         for (int i = 1; i < 4; i++)
  197:         {
  198:             Link.Connect(digraphs[0], digraphs[i]);
  199:             for (int j = 0; j < 3; j++)
  200:                 Link.Connect(digraphs[i], digraphs[3 * i + j+1]);
  201:         }
  202:
  203:         //then position second ring
  204:         //could call with: i * circlePercent as offset
  205:     }
  206:
  207:     private void TestRadialPositioning1()
  208:     {
  209:         //  for (int i = 0; i < 3; i++)
  210:         //      Debug.Log($"dir: {DigraphStructure.radialDirection(1, 0, 3, i)}");
  211:
  212:         //for (int i = 0; i < 4; i++)
  213:         //     Debug.Log($"dir: {DigraphStructure.radialDirection(1/3.0f, -45*Mathf.
Deg2Rad, 3, i)}");
  214:
  215:         float initialCirclePercent = 120f / 360f;
  216:         float nextCirclePercent = initialCirclePercent * 0.75f;
  217:         float baseOffset = (nextCirclePercent * Mathf.PI * -1);
  218:         int count = 3;
  219:         for (int i = 0; i < count; i++)
  220:             DigraphStructure.radialDirection(nextCirclePercent, baseOffset, count -
1, i);
  221:
  222:         //the others should call with a hard offset of 120deg and 240 added
  223:
  224:     }
  225:
  226:     private IEnumerator TestUBFS1()
  227:     {
  228:         Algorithms ac = GetComponent<Algorithms>();
  229:
  230:         Undigraph a = (Undigraph)Node.Create(1, Node.CreationSelection.Undigraph);
  231:         Undigraph b = (Undigraph)Node.Create(2, Node.CreationSelection.Undigraph);
  232:         Undigraph c = (Undigraph)Node.Create(3, Node.CreationSelection.Undigraph);
  233:         Undigraph d = (Undigraph)Node.Create(4, Node.CreationSelection.Undigraph);
  234:         Undigraph e = (Undigraph)Node.Create(5, Node.CreationSelection.Undigraph);
  235:         Undigraph f = (Undigraph)Node.Create(6, Node.CreationSelection.Undigraph);
  236:         Undigraph g = (Undigraph)Node.Create(7, Node.CreationSelection.Undigraph);
  237:         Undigraph h = (Undigraph)Node.Create(8, Node.CreationSelection.Undigraph);
  238:         Undigraph i = (Undigraph)Node.Create(9, Node.CreationSelection.Undigraph);
  239:         Undigraph j = (Undigraph)Node.Create(10, Node.CreationSelection.Undigraph);
  240:         Undigraph k = (Undigraph)Node.Create(11, Node.CreationSelection.Undigraph);
  241:         Undigraph l = (Undigraph)Node.Create(12, Node.CreationSelection.Undigraph);
  242:         Undigraph m = (Undigraph)Node.Create(13, Node.CreationSelection.Undigraph);
  243:         Undigraph n = (Undigraph)Node.Create(14, Node.CreationSelection.Undigraph);
  244:
  245:         //d1
  246:         Link.Connect(a, b);
  247:         Link.Connect(b, a);
  248:         Link.Connect(c, a);
  249:         //d2
  250:         Link.Connect(c, d);
  251:         Link.Connect(e, c);
  252:         //d3
  253:         Link.Connect(f, e);
  254:
  255:         UndigraphStructure ugs = UndigraphStructure.Organize(a);
  256:         yield return StartCoroutine(ac.BFS(ugs, -1));
  257:         yield return StartCoroutine(ac.BFS(ugs, 10));
  258:     }
  259:
  260:     private IEnumerator TestUDFS1()
  261:     {
```

```
262:         Algorithms ac = GetComponent<Algorithms>();
263:
264:         Undigraph a = (Undigraph)Node.Create(1, Node.CreationSelection.Undigraph);
265:         Undigraph b = (Undigraph)Node.Create(2, Node.CreationSelection.Undigraph);
266:         Undigraph c = (Undigraph)Node.Create(3, Node.CreationSelection.Undigraph);
267:         Undigraph d = (Undigraph)Node.Create(4, Node.CreationSelection.Undigraph);
268:         Undigraph e = (Undigraph)Node.Create(5, Node.CreationSelection.Undigraph);
269:         Undigraph f = (Undigraph)Node.Create(6, Node.CreationSelection.Undigraph);
270:         Undigraph g = (Undigraph)Node.Create(7, Node.CreationSelection.Undigraph);
271:         Undigraph h = (Undigraph)Node.Create(8, Node.CreationSelection.Undigraph);
272:         Undigraph i = (Undigraph)Node.Create(9, Node.CreationSelection.Undigraph);
273:         Undigraph j = (Undigraph)Node.Create(10, Node.CreationSelection.Undigraph);
274:         Undigraph k = (Undigraph)Node.Create(11, Node.CreationSelection.Undigraph);
275:         Undigraph l = (Undigraph)Node.Create(12, Node.CreationSelection.Undigraph);
276:         Undigraph m = (Undigraph)Node.Create(13, Node.CreationSelection.Undigraph);
277:         Undigraph n = (Undigraph)Node.Create(14, Node.CreationSelection.Undigraph);
278:
279:         //d1
280:         Link.Connect(a, b);
281:         Link.Connect(b, a);
282:         Link.Connect(c, a);
283:         //d2
284:         Link.Connect(c, d);
285:         Link.Connect(e, c);
286:         //d3
287:         Link.Connect(f, e);
288:
289:         UndigraphStructure ugs = UndigraphStructure.Organize(a);
290:         yield return StartCoroutine(ac.DFS(ugs, -1));
291:         yield return StartCoroutine(ac.DFS(ugs, 10));
292:     }
293:
294:     private IEnumerator TestDFS1()
295:     {
296:         Algorithms ac = GetComponent<Algorithms>();
297:
298:         //create digraph
299:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
300:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
301:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
302:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
303:         Digraph e = (Digraph)Node.Create(5, Node.CreationSelection.Digraph);
304:         Digraph f = (Digraph)Node.Create(6, Node.CreationSelection.Digraph);
305:         Digraph g = (Digraph)Node.Create(7, Node.CreationSelection.Digraph);
306:         Digraph h = (Digraph)Node.Create(8, Node.CreationSelection.Digraph);
307:         Digraph i = (Digraph)Node.Create(9, Node.CreationSelection.Digraph);
308:         Digraph j = (Digraph)Node.Create(10, Node.CreationSelection.Digraph);
309:         Digraph k = (Digraph)Node.Create(11, Node.CreationSelection.Digraph);
310:         Digraph l = (Digraph)Node.Create(12, Node.CreationSelection.Digraph);
311:         Digraph m = (Digraph)Node.Create(13, Node.CreationSelection.Digraph);
312:         Digraph n = (Digraph)Node.Create(14, Node.CreationSelection.Digraph);
313:
314:         //depth 1
315:         Link.Connect(a, b);
316:         Link.Connect(a, h);
317:         Link.Connect(a, l);
318:         Link.Connect(a, m);
319:         Link.Connect(a, n);
320:         //depth 2
321:         Link.Connect(b, c);
322:         Link.Connect(b, d);
323:         Link.Connect(h, i);
324:         Link.Connect(h, j);
325:         //d3
326:         Link.Connect(c, f);
327:         Link.Connect(c, g);
328:         Link.Connect(d, e);
329:         Link.Connect(l, k);
```

```
330:
331:         DigraphStructure dgs = DigraphStructure.Organize(a);
332:
333:         //start search
334:         //]yield return StartCoroutine(ac.BFS(dgs, 10));
335:         yield return StartCoroutine(ac.DFS(dgs, -1));
336:     }
337:
338:     private IEnumerator TestBFS1()
339:     {
340:         Algorithms ac = GetComponent<Algorithms>();
341:
342:         //create digraph
343:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
344:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
345:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
346:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
347:         Digraph e = (Digraph)Node.Create(5, Node.CreationSelection.Digraph);
348:         Digraph f = (Digraph)Node.Create(6, Node.CreationSelection.Digraph);
349:         Digraph g = (Digraph)Node.Create(7, Node.CreationSelection.Digraph);
350:         Digraph h = (Digraph)Node.Create(8, Node.CreationSelection.Digraph);
351:         Digraph i = (Digraph)Node.Create(9, Node.CreationSelection.Digraph);
352:         Digraph j = (Digraph)Node.Create(10, Node.CreationSelection.Digraph);
353:         Digraph k = (Digraph)Node.Create(11, Node.CreationSelection.Digraph);
354:         Digraph l = (Digraph)Node.Create(12, Node.CreationSelection.Digraph);
355:         Digraph m = (Digraph)Node.Create(13, Node.CreationSelection.Digraph);
356:         Digraph n = (Digraph)Node.Create(14, Node.CreationSelection.Digraph);
357:
358:         //depth 1
359:         Link.Connect(a, b);
360:         Link.Connect(a, h);
361:         Link.Connect(a, l);
362:         Link.Connect(a, m);
363:         Link.Connect(a, n);
364:         //depth 2
365:         Link.Connect(b, c);
366:         Link.Connect(b, d);
367:         Link.Connect(h, i);
368:         Link.Connect(h, j);
369:         //d3
370:         Link.Connect(c, f);
371:         Link.Connect(c, g);
372:         Link.Connect(d, e);
373:         Link.Connect(l, k);
374:
375:         DigraphStructure dgs = DigraphStructure.Organize(a);
376:
377:         //start search
378:         //]yield return StartCoroutine(ac.BFS(dgs, 10));
379:         yield return StartCoroutine(ac.BFS(dgs, -1));
380:     }
381:
382:     private IEnumerator TestLLShift()
383:     {
384:         Algorithms ac = GetComponent<Algorithms>();
385:         LinkedListStructure lls;
386:
387:         for (int i = 0; i < 1; i++)
388:         {
389:             lls = DemoStructures.CreateDemo(DemoStructures.NodeType.LinkedList, 0).G
etComponent<LinkedListStructure>();
390:             yield return StartCoroutine(ac.Shift(lls, i+1));
391:             Destroy(lls.gameObject);
392:         }
393:     }
394:
395:     private IEnumerator TestLLRCoR()
396:     {
```

```
  397:         Algorithms ac = GetComponent<Algorithms>();
  398:         LinkedListStructure lls;
  399:
  400:         for (int i = 0; i < 1; i++)
  401:         {
  402:             lls = DemoStructures.CreateDemo(DemoStructures.NodeType.LinkedList, i).G
etComponent<LinkedListStructure>();
  403:
  404:             Debug.Log($"Reversing on {i}");
  405:             yield return StartCoroutine(ac.Reverse(lls));
  406:             Destroy(lls.gameObject);
  407:
  408:         }
  409:         Debug.Log($"finished");
  410:     }
  411:
  412:     private IEnumerator TestPerfectionCoR()
  413:     {
  414:         Algorithms ac = GetComponent<Algorithms>();
  415:         BinaryTreeStructure bts;
  416:
  417:         for (int i = 0; i < 6; i++)
  418:         {
  419:             bts = DemoStructures.CreateDemo(DemoStructures.NodeType.BinaryTree, i).G
etComponent<BinaryTreeStructure>();
  420:
  421:             Debug.Log($"Checking perfect on {i}");
  422:             yield return StartCoroutine(ac.Perfect(bts));
  423:             Destroy(bts.gameObject);
  424:
  425:         }
  426:         Debug.Log($"finished");
  427:     }
  428:
  429:     private void TestPerfectCheck()
  430:     {
  431:         Algorithms ac = GetComponent<Algorithms>();
  432:         BinaryTree root;
  433:
  434:         for (int i = 0; i < 6; i++)
  435:         {
  436:             root = DemoStructures.CreateDemo(DemoStructures.NodeType.BinaryTree, i).
GetComponent<BinaryTreeStructure>().root;
  437:
  438:             Debug.Log($"Perfect on {i} with {root.transform.parent.gameObject.name}?
 {ac.realPerfectCheck(root)}");
  439:         }
  440:     }
  441:
  442:     private IEnumerator TestSequenceCoR()
  443:     {
  444:         for(int i = 0; i < 5; i++)
  445:         {
  446:             yield return StartCoroutine(Algorithms.WaitForIteration());
  447:             Debug.Log($"{i} reached");
  448:         }
  449:     }
  450:
  451:     private IEnumerator TestCompleteBinaryTreeCoR()
  452:     {
  453:         Algorithms ac = GetComponent<Algorithms>();
  454:         BinaryTreeStructure bts;
  455:
  456:         for (int i = 0; i < 6; i++)
  457:         {
  458:             bts = DemoStructures.CreateDemo(DemoStructures.NodeType.BinaryTree, i).G
etComponent<BinaryTreeStructure>();
  459:
```

```
  460:             Debug.Log($"Checking full on {i}");
  461:             yield return StartCoroutine(ac.Complete(bts));
  462:             Destroy(bts.gameObject);
  463:         }
  464:     }
  465:
  466:
  467:     private void TestCompleteBinaryTree()
  468:     {
  469:         Algorithms ac = GetComponent<Algorithms>();
  470:         BinaryTree root;
  471:
  472:         for (int i = 0; i < 6; i++)
  473:         {
  474:             root = DemoStructures.CreateDemo(DemoStructures.NodeType.BinaryTree, i).
GetComponent<BinaryTreeStructure>().root;
  475:
  476:             Debug.Log($"Complete on {i} with {root.transform.parent.gameObject.name}
? {ac.realCompleteCheck(root)}");
  477:         }
  478:     }
  479:
  480:     private IEnumerator TestFullBinaryTreeCoR()
  481:     {
  482:         Algorithms ac = GetComponent<Algorithms>();
  483:         BinaryTreeStructure bts;
  484:
  485:         for (int i = 0; i < 6; i++)
  486:         {
  487:             bts = DemoStructures.CreateDemo(DemoStructures.NodeType.BinaryTree, i).G
etComponent<BinaryTreeStructure>();
  488:
  489:             Debug.Log($"Checking full on {i}");
  490:             yield return StartCoroutine(ac.Full(bts));
  491:             Destroy(bts.gameObject);
  492:         }
  493:     }
  494:
  495:     private void TestFullBinaryTree()
  496:     {
  497:         Algorithms ac = GetComponent<Algorithms>();
  498:         BinaryTree root;
  499:
  500:         for(int i = 0; i < 6; i++)
  501:         {
  502:             root = DemoStructures.CreateDemo(DemoStructures.NodeType.BinaryTree, i).
GetComponent<BinaryTreeStructure>().root;
  503:
  504:             Debug.Log($"Full on {i} with {root.transform.parent.gameObject.name}? {a
c.realFullCheck(root)}");
  505:         }
  506:
  507:     }
  508:
  509:     private void TestRecursionSelection1()
  510:     {
  511:         GameObject g = DemoStructures.CreateDemo(DemoStructures.NodeType.BinaryTree,
 0);
  512:         BinaryTreeStructure bts = g.GetComponent<BinaryTreeStructure>();
  513:         BinaryTree r = bts.root;
  514:
  515:         Algorithms ac = GetComponent<Algorithms>();
  516:         ac.TestSelectionRecursion(r);
  517:     }
  518:
  519:     private IEnumerator TestSelection()
  520:     {
  521:         Algorithms.iterationMode = Algorithms.IterationMode.Input;
```

```
522:
523:         Node n = Node.Create(5, 0);
524:         GameObject g = n.gameObject;
525:
526:         Algorithms.Select(g);
527:
528:         yield return StartCoroutine(Algorithms.WaitForIteration());
529:
530:         Algorithms.Deselect(g);
531:
532:         yield return StartCoroutine(Algorithms.WaitForIteration());
533:
534:         Algorithms.Select(g);
535:
536:         yield return StartCoroutine(Algorithms.WaitForIteration());
537:
538:         Algorithms.Deselect(g);
539:     }
540:
541:     private IEnumerator TestIteration2()
542:     {
543:         Algorithms.iterationMode = Algorithms.IterationMode.Wait;
544:         Algorithms.waitTime = 1.5f;
545:         Algorithms.vr = false;
546:         for (int i = 0; i < 5; i++)
547:         {
548:             Debug.Log($"Iteration {i} at {Time.realtimeSinceStartup}");
549:             yield return StartCoroutine(Algorithms.WaitForIteration());
550:         }
551:     }
552:
553:     private IEnumerator TestIteration()
554:     {
555:         Algorithms.iterationMode = Algorithms.IterationMode.Input;
556:         Algorithms.vr = false;
557:         for(int i = 0; i < 5; i++)
558:         {
559:             Debug.Log($"Iteration {i}");
560:             yield return StartCoroutine(Algorithms.WaitForIteration());
561:         }
562:     }
563:
564:     private void TestDemo2()
565:     {
566:         uIManagerComponent.nextPanel();
567:         uIManagerComponent.nextPanel();
568:         uIManagerComponent.nextPanel();
569:         uIManagerComponent.nextPanel();
570:         uIManagerComponent.nextPanel();
571:         //uIManagerComponent.changeDemoType(true);
572:     }
573:
574:     private void TestDemoStructures()
575:     {
576:         for (int i = 0; i < 4; i++)
577:             for (int j = 0; j < 6; j++)
578:                 DemoStructures.CreateDemo((DemoStructures.NodeType)i, j);
579:     }
580:
581:     private void TestDemoStructuresPanel()
582:     {
583:         uIManagerComponent.changeDemoType(true);
584:         uIManagerComponent.changeDemoType(true);
585:         uIManagerComponent.changeDemoType(true);
586:     }
587:
588:     private void TestUndigraphOrganization()
589:     {
```

```
590:         Undigraph a = (Undigraph)Node.Create(1, Node.CreationSelection.Undigraph);
591:         Undigraph b = (Undigraph)Node.Create(2, Node.CreationSelection.Undigraph);
592:         Undigraph c = (Undigraph)Node.Create(3, Node.CreationSelection.Undigraph);
593:         Undigraph d = (Undigraph)Node.Create(4, Node.CreationSelection.Undigraph);
594:         Undigraph e = (Undigraph)Node.Create(5, Node.CreationSelection.Undigraph);
595:         Undigraph f = (Undigraph)Node.Create(6, Node.CreationSelection.Undigraph);
596:         Undigraph g = (Undigraph)Node.Create(7, Node.CreationSelection.Undigraph);
597:         Undigraph h = (Undigraph)Node.Create(8, Node.CreationSelection.Undigraph);
598:         Undigraph i = (Undigraph)Node.Create(9, Node.CreationSelection.Undigraph);
599:         Undigraph j = (Undigraph)Node.Create(10, Node.CreationSelection.Undigraph);
600:         Undigraph k = (Undigraph)Node.Create(11, Node.CreationSelection.Undigraph);
601:         Undigraph l = (Undigraph)Node.Create(12, Node.CreationSelection.Undigraph);
602:         Undigraph m = (Undigraph)Node.Create(13, Node.CreationSelection.Undigraph);
603:         Undigraph n = (Undigraph)Node.Create(14, Node.CreationSelection.Undigraph);
604:
605:         //d1
606:         Link.Connect(a, b);
607:         Link.Connect(b, a);
608:         Link.Connect(c, a);
609:         //d2
610:         Link.Connect(c, d);
611:         Link.Connect(e, c);
612:         //d3
613:         Link.Connect(f, e);
614:
615:         UndigraphStructure.Organize(a);
616:     }
617:
618:     private void TestDigraphOrganizationRandom()
619:     {
620:         List<Digraph> nodes = new List<Digraph>();
621:
622:         int n = 12;
623:         int l = 20;
624:
625:         for (int i = 0; i < n; i++)
626:             nodes.Add((Digraph)Node.Create(i, Node.CreationSelection.Digraph));
627:
628:         for (int i = 0; i < l; i++)
629:         {
630:             Link.Connect(nodes[Random.Range(0, n)], nodes[Random.Range(0, n)]);
631:         }
632:
633:         DigraphStructure.Organize(nodes[Random.Range(0, n)]);
634:     }
635:
636:     private void TestDigraphOrganization3()
637:     {
638:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
639:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
640:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
641:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
642:         Digraph e = (Digraph)Node.Create(5, Node.CreationSelection.Digraph);
643:         Digraph f = (Digraph)Node.Create(6, Node.CreationSelection.Digraph);
644:         Digraph g = (Digraph)Node.Create(7, Node.CreationSelection.Digraph);
645:         Digraph h = (Digraph)Node.Create(8, Node.CreationSelection.Digraph);
646:         Digraph i = (Digraph)Node.Create(9, Node.CreationSelection.Digraph);
647:         Digraph j = (Digraph)Node.Create(10, Node.CreationSelection.Digraph);
648:         Digraph k = (Digraph)Node.Create(11, Node.CreationSelection.Digraph);
649:         Digraph l = (Digraph)Node.Create(12, Node.CreationSelection.Digraph);
650:         Digraph m = (Digraph)Node.Create(13, Node.CreationSelection.Digraph);
651:         Digraph n = (Digraph)Node.Create(14, Node.CreationSelection.Digraph);
652:
653:         //depth 1
654:         Link.Connect(a, b);
655:         Link.Connect(a, h);
656:         Link.Connect(a, l);
657:         Link.Connect(a, m);
```

```
658:         Link.Connect(a, n);
659:         //depth 2
660:         Link.Connect(b, c);
661:         Link.Connect(b, d);
662:         Link.Connect(h, i);
663:         Link.Connect(h, j);
664:         //d3
665:         Link.Connect(c, f);
666:         Link.Connect(c, g);
667:         Link.Connect(d, e);
668:         Link.Connect(l, k);
669:
670:         DigraphStructure.Organize(a);
671:     }
672:
673:     private void TestDigraphOrganization2()
674:     {
675:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
676:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
677:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
678:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
679:         Digraph e = (Digraph)Node.Create(5, Node.CreationSelection.Digraph);
680:         Digraph f = (Digraph)Node.Create(6, Node.CreationSelection.Digraph);
681:         Digraph g = (Digraph)Node.Create(7, Node.CreationSelection.Digraph);
682:
683:         Link.Connect(a, b);
684:         Link.Connect(a, c);
685:         Link.Connect(a, d);
686:         Link.Connect(b, d);
687:         Link.Connect(b, e);
688:         Link.Connect(c, f);
689:         Link.Connect(d, f);
690:         Link.Connect(d, g);
691:         Link.Connect(e, f);
692:         Link.Connect(e, g);
693:         Link.Connect(f, g);
694:
695:         DigraphStructure.Organize(a);
696:     }
697:
698:     private void TestDigraphOrganization()
699:     {
700:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
701:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
702:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
703:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
704:         Digraph e = (Digraph)Node.Create(5, Node.CreationSelection.Digraph);
705:         Digraph f = (Digraph)Node.Create(6, Node.CreationSelection.Digraph);
706:
707:         Link.Connect(a, b);
708:         Link.Connect(b, c);
709:         Link.Connect(c, e);
710:         Link.Connect(e, d);
711:         Link.Connect(e, f);
712:         Link.Connect(d, b);
713:
714:         DigraphStructure.Organize(a);
715:     }
716:
717:     private void TestConnectionDirectionOnConnection()
718:     {
719:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
720:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
721:         a.transform.position = basePosition;
722:         b.transform.position = basePosition + Vector3.right*3;
723:
724:         Undigraph c = (Undigraph)Node.Create(3, Node.CreationSelection.Undigraph);
725:         Undigraph d = (Undigraph)Node.Create(4, Node.CreationSelection.Undigraph);
```

```
  726:         c.transform.position = basePosition + Vector3.up * 3;
  727:         d.transform.position = basePosition + Vector3.up * 3 + Vector3.right * 3;
  728:
  729:         Link.Connect(a, b);
  730:         Link.Connect(c, d);
  731:     }
  732:
  733:     private void TestLLStructure5()
  734:     {
  735:         LinkedList a = (LinkedList)Node.Create(1, 1);
  736:         a.transform.position = basePosition;
  737:         LinkedList b = (LinkedList)Node.Create(2, 1);
  738:         b.transform.position = basePosition + transform.right * 3;
  739:         LinkedList c = (LinkedList)Node.Create(3, 1);
  740:         c.transform.position = basePosition + transform.right * 6;
  741:         LinkedList d = (LinkedList)Node.Create(4, 1);
  742:         d.transform.position = basePosition + transform.right * 9;
  743:
  744:         Link aL = Link.Connect(a, b);
  745:         Link bL = Link.Connect(b, c);
  746:         Link cL = Link.Connect(c, d);
  747:
  748:         LinkedListStructure.Organize(a);
  749:
  750:         Link.Disconnect(bL);
  751:
  752:         LinkedListStructure.Organize(c);
  753:     }
  754:
  755:     private void TestBTStructure4()
  756:     {
  757:         BinaryTree a = (BinaryTree)Node.Create(1, Node.CreationSelection.BinaryTree)
;
  758:         BinaryTree b = (BinaryTree)Node.Create(2, Node.CreationSelection.BinaryTree)
;
  759:         BinaryTree c = (BinaryTree)Node.Create(3, Node.CreationSelection.BinaryTree)
;
  760:         BinaryTree d = (BinaryTree)Node.Create(4, Node.CreationSelection.BinaryTree)
;
  761:         BinaryTree e = (BinaryTree)Node.Create(5, Node.CreationSelection.BinaryTree)
;
  762:         BinaryTree f = (BinaryTree)Node.Create(6, Node.CreationSelection.BinaryTree)
;
  763:         BinaryTree g = (BinaryTree)Node.Create(7, Node.CreationSelection.BinaryTree)
;
  764:
  765:         Link ab = Link.Connect(a, b);
  766:         Link.Connect(a, c);
  767:
  768:         Link.Connect(b, d);
  769:         Link.Connect(b, e);
  770:
  771:         Link.Connect(c, f);
  772:         Link.Connect(c, g);
  773:
  774:         BinaryTreeStructure.Organize(f);
  775:
  776:         Link.Disconnect(ab);
  777:
  778:         BinaryTreeStructure.Organize(b);
  779:
  780:     }
  781:
  782:     private void TestBTStructure3()
  783:     {
  784:         BinaryTree a = (BinaryTree)Node.Create(1, Node.CreationSelection.BinaryTree)
;
  785:         BinaryTree b = (BinaryTree)Node.Create(2, Node.CreationSelection.BinaryTree)
```

```
;
  786:         BinaryTree c = (BinaryTree)Node.Create(3, Node.CreationSelection.BinaryTree)
;
  787:
  788:         Link.Connect(a, b);
  789:         Link.Connect(a, c);
  790:         Link.Connect(c, a);
  791:
  792:         BinaryTreeStructure.Organize(a);
  793:
  794:     }
  795:
  796:     private void TestBTStructure2()
  797:     {
  798:         BinaryTree a = (BinaryTree)Node.Create(1, Node.CreationSelection.BinaryTree)
;
  799:         BinaryTree b = (BinaryTree)Node.Create(2, Node.CreationSelection.BinaryTree)
;
  800:         BinaryTree c = (BinaryTree)Node.Create(3, Node.CreationSelection.BinaryTree)
;
  801:         BinaryTree d = (BinaryTree)Node.Create(4, Node.CreationSelection.BinaryTree)
;
  802:         BinaryTree e = (BinaryTree)Node.Create(5, Node.CreationSelection.BinaryTree)
;
  803:         BinaryTree f = (BinaryTree)Node.Create(6, Node.CreationSelection.BinaryTree)
;
  804:         BinaryTree g = (BinaryTree)Node.Create(7, Node.CreationSelection.BinaryTree)
;
  805:
  806:         Link.Connect(a, b);
  807:         Link.Connect(a, c);
  808:
  809:         Link.Connect(b, d);
  810:         Link.Connect(b, e);
  811:
  812:         Link.Connect(c, f);
  813:         Link.Connect(c, g);
  814:
  815:         BinaryTreeStructure.Organize(f);
  816:
  817:     }
  818:
  819:     private void TestBTStructure1()
  820:     {
  821:         BinaryTree a = (BinaryTree)Node.Create(1, Node.CreationSelection.BinaryTree)
;
  822:         BinaryTree b = (BinaryTree)Node.Create(2, Node.CreationSelection.BinaryTree)
;
  823:         BinaryTree c = (BinaryTree)Node.Create(3, Node.CreationSelection.BinaryTree)
;
  824:         BinaryTree d = (BinaryTree)Node.Create(4, Node.CreationSelection.BinaryTree)
;
  825:         BinaryTree e = (BinaryTree)Node.Create(5, Node.CreationSelection.BinaryTree)
;
  826:         BinaryTree f = (BinaryTree)Node.Create(6, Node.CreationSelection.BinaryTree)
;
  827:         BinaryTree g = (BinaryTree)Node.Create(7, Node.CreationSelection.BinaryTree)
;
  828:
  829:         Link.Connect(a, b);
  830:         Link.Connect(a, c);
  831:
  832:         Link.Connect(b, d);
  833:         Link.Connect(b, e);
  834:
  835:         Link.Connect(c, f);
  836:         Link.Connect(c, g);
  837:
```

```
  838:         BinaryTreeStructure.Organize(a);
  839:
  840:     }
  841:
  842:     private void TestLLStructure4()
  843:     {
  844:         LinkedList a = (LinkedList)Node.Create(1, 1);
  845:         a.transform.position = basePosition;
  846:         LinkedList b = (LinkedList)Node.Create(2, 1);
  847:         b.transform.position = basePosition + transform.right * 3;
  848:         LinkedList c = (LinkedList)Node.Create(3, 1);
  849:         c.transform.position = basePosition + transform.right * 6;
  850:         LinkedList d = (LinkedList)Node.Create(4, 1);
  851:         d.transform.position = basePosition + transform.right * 9;
  852:         LinkedList e = (LinkedList)Node.Create(5, 1);
  853:         e.transform.position = basePosition + transform.right * 9 + transform.up * -
3;
  854:         LinkedList f = (LinkedList)Node.Create(6, 1);
  855:         f.transform.position = basePosition + transform.right * 6 + transform.up * -
3;
  856:
  857:         Link aL = Link.Connect(a, b);
  858:         Link bL = Link.Connect(b, c);
  859:         Link cL = Link.Connect(c, d);
  860:         Link dl = Link.Connect(d, e);
  861:         Link el = Link.Connect(e, f);
  862:         //Link fl = Link.Connect(f, c);
  863:
  864:         LinkedListStructure.Organize(c);
  865:     }
  866:
  867:     private void TestLLStructure3()
  868:     {
  869:         LinkedList a = (LinkedList)Node.Create(1, 1);
  870:         a.transform.position = basePosition;
  871:         LinkedList b = (LinkedList)Node.Create(2, 1);
  872:         b.transform.position = basePosition + transform.right * 3;
  873:         LinkedList c = (LinkedList)Node.Create(3, 1);
  874:         c.transform.position = basePosition + transform.right * 6;
  875:         LinkedList d = (LinkedList)Node.Create(4, 1);
  876:         d.transform.position = basePosition + transform.right * 9;
  877:         LinkedList e = (LinkedList)Node.Create(5, 1);
  878:         e.transform.position = basePosition + transform.right * 9 + transform.up * -
3;
  879:         LinkedList f = (LinkedList)Node.Create(6, 1);
  880:         f.transform.position = basePosition + transform.right * 6 + transform.up * -
3;
  881:
  882:         Link aL = Link.Connect(a, b);
  883:         Link bL = Link.Connect(b, c);
  884:         Link cL = Link.Connect(c, d);
  885:         Link dl = Link.Connect(d, e);
  886:         Link el = Link.Connect(e, f);
  887:         Link fl = Link.Connect(f, c);
  888:
  889:         LinkedListStructure.Organize(c);
  890:     }
  891:
  892:     private void TestLLStructure2()
  893:     {
  894:         LinkedList a = (LinkedList)Node.Create(1, 1);
  895:         a.transform.position = basePosition;
  896:         LinkedList b = (LinkedList)Node.Create(2, 1);
  897:         b.transform.position = basePosition + transform.right * 3;
  898:         LinkedList c = (LinkedList)Node.Create(3, 1);
  899:         c.transform.position = basePosition + transform.right * 6;
  900:         LinkedList d = (LinkedList)Node.Create(4, 1);
  901:         d.transform.position = basePosition + transform.right * 9;
```

```
902:
903:         Link aL = Link.Connect(a, b);
904:         Link bL = Link.Connect(b, c);
905:         Link cL = Link.Connect(c, d);
906:         Link dl = Link.Connect(d, a);
907:
908:         LinkedListStructure.Organize(a);
909:     }
910:
911:     private void TestLLStructure1()
912:     {
913:         LinkedList a = (LinkedList)Node.Create(1, 1);
914:         a.transform.position = basePosition;
915:         LinkedList b = (LinkedList)Node.Create(2, 1);
916:         b.transform.position = basePosition + transform.right * 3;
917:         LinkedList c = (LinkedList)Node.Create(3, 1);
918:         c.transform.position = basePosition + transform.right * 6;
919:         LinkedList d = (LinkedList)Node.Create(4, 1);
920:         d.transform.position = basePosition + transform.right * 9;
921:
922:         Link aL = Link.Connect(a, b);
923:         Link bL = Link.Connect(b, c);
924:         Link cL = Link.Connect(c, d);
925:
926:         LinkedListStructure.Organize(a);
927:     }
928:
929:     private void TestGraphConnection()
930:     {
931:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
932:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
933:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
934:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
935:         a.transform.position = basePosition;
936:         b.transform.position = basePosition + transform.right * 3;
937:         c.transform.position = basePosition + transform.right * 6;
938:         d.transform.position = basePosition + transform.right * 9;
939:
940:         Link l1 = Link.Connect(a, b);
941:         Link l2 = Link.Connect(b, c);
942:         Link l3 = Link.Connect(c, d);
943:
944:         //basePosition += Vector3.up * 4;
945:     }
946:
947:     private void TestGraphConnectionFlip()
948:     {
949:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
950:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
951:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
952:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
953:         a.transform.position = basePosition;
954:         b.transform.position = basePosition + transform.right * 3;
955:         c.transform.position = basePosition + transform.right * 6;
956:         d.transform.position = basePosition + transform.right * 9;
957:
958:         Link l1 = Link.Connect(a, b);
959:         Link l2 = Link.Connect(b, c);
960:         Link l3 = Link.Connect(c, d);
961:
962:         Link.Flip(l1);
963:         Link.Flip(l2);
964:         Link.Flip(l3);
965:     }
966:
967:     private void TestBinaryTreeFromArray()
968:     {
969:         int[] arr = { 10, 5, 15, 2, 7, 13, 20 };
```

```
  970:         BinaryTreeStructure.FromArray(arr, Vector3.forward * 10);
  971:     }
  972:
  973:     private void TestBasicConnection()
  974:     {
  975:         BinaryTree b = (BinaryTree)Node.Create(1, 0);
  976:         b.transform.position = basePosition + new Vector3(0, 1, 0);// new Vector3(0,
4,2)
  977:         BinaryTree l = (BinaryTree)Node.Create(2, 0);
  978:         l.transform.position = basePosition + new Vector3(-1, -2, 0); //new Vector3(
1, 1, 2);
  979:         BinaryTree r = (BinaryTree)Node.Create(3, 0);
  980:         r.transform.position = basePosition + new Vector3(1, -2, 0); //new Vector3(-
1, 1, 2);
  981:         Link.Connect(b, l);
  982:         Link.Connect(b, r);
  983:         basePosition += Vector3.right * 5;
  984:     }
  985:
  986:     private void TestConnectionOverwriting1()
  987:     {
  988:         BinaryTree b = (BinaryTree)Node.Create(1, 0);
  989:         b.transform.position = basePosition + new Vector3(0, 1, 0);// new Vector3(0,
4,2)
  990:         BinaryTree l = (BinaryTree)Node.Create(2, 0);
  991:         l.transform.position = basePosition + new Vector3(-1, -2, 0); //new Vector3(
1, 1, 2);
  992:         BinaryTree r = (BinaryTree)Node.Create(3, 0);
  993:         r.transform.position = basePosition + new Vector3(1, -2, 0); //new Vector3(-
1, 1, 2);
  994:
  995:
  996:
  997:         //Create nodes for overwriting
  998:         BinaryTree l2 = (BinaryTree)Node.Create(2, 0);
  999:         l2.transform.position = basePosition + new Vector3(2, -2, 0);
 1000:         BinaryTree r2 = (BinaryTree)Node.Create(3, 0);
 1001:         r2.transform.position = basePosition + new Vector3(-2, -2, 0);
 1002:
 1003:         Link.Connect(b, l);
 1004:         Link.Connect(b, r);
 1005:
 1006:         //Connection overwriting
 1007:         Link.Connect(b, l2);
 1008:         Link.Connect(b, r2);
 1009:
 1010:         basePosition += Vector3.right * 5;
 1011:     }
 1012:
 1013:     private void TestConnectionDirectedness()
 1014:     {
 1015:         BinaryTree b = (BinaryTree)Node.Create(1, 0);
 1016:         b.transform.position = basePosition + new Vector3(0, 1, 0);// new Vector3(0,
4,2)
 1017:         BinaryTree l = (BinaryTree)Node.Create(2, 0);
 1018:         l.transform.position = basePosition + new Vector3(-1, -2, 0); //new Vector3(
1, 1, 2);
 1019:         BinaryTree r = (BinaryTree)Node.Create(3, 0);
 1020:         r.transform.position = basePosition + new Vector3(1, -2, 0); //new Vector3(-
1, 1, 2);
 1021:         Link ll = Link.Connect(b, l);
 1022:         Link rl = Link.Connect(b, r);
 1023:
 1024:         ll.Directed = true;
 1025:         rl.Directed = false;
 1026:
 1027:         basePosition += Vector3.right * 5;
 1028:     }
```

```
1029:
1030:     private void TestBasicReconnection()
1031:     {
1032:         BinaryTree b = (BinaryTree)Node.Create(1, 0);
1033:         b.transform.position = basePosition + new Vector3(0, 1, 0);// new Vector3(0,
4,2)
1034:         BinaryTree l = (BinaryTree)Node.Create(2, 0);
1035:         l.transform.position = basePosition + new Vector3(-1, -2, 0); //new Vector3(
1, 1, 2);
1036:         BinaryTree r = (BinaryTree)Node.Create(3, 0);
1037:         r.transform.position = basePosition + new Vector3(1, -2, 0); //new Vector3(-
1, 1, 2);
1038:         Link.Connect(b, l);
1039:         Link.Connect(b, r);
1040:
1041:         l.transform.position += Vector3.left;
1042:         r.transform.position += Vector3.right;
1043:
1044:         Link.Reconnect(b);
1045:
1046:         basePosition += Vector3.right * 5;
1047:     }
1048:
1049:     private void TestBasicFlipping()
1050:     {
1051:         BinaryTree b = (BinaryTree)Node.Create(1, 0);
1052:         b.transform.position = basePosition + new Vector3(0, 1, 0);// new Vector3(0,
4,2)
1053:         BinaryTree l = (BinaryTree)Node.Create(2, 0);
1054:         l.transform.position = basePosition + new Vector3(-1, -2, 0); //new Vector3(
1, 1, 2);
1055:         BinaryTree r = (BinaryTree)Node.Create(3, 0);
1056:         r.transform.position = basePosition + new Vector3(1, -2, 0); //new Vector3(-
1, 1, 2);
1057:         Link ll = Link.Connect(b, l).GetComponent<Link>();
1058:         Link rl = Link.Connect(b, r).GetComponent<Link>();
1059:
1060:         Link.Flip(ll);
1061:         Link.Flip(ll);
1062:
1063:         Link.Flip(rl);
1064:
1065:         basePosition += Vector3.right * 5;
1066:     }
1067:
1068:     private void TestFlippingLL()
1069:     {
1070:         LinkedList a = (LinkedList)Node.Create(1, 1);
1071:         a.transform.position = basePosition;
1072:         LinkedList b = (LinkedList)Node.Create(2, 1);
1073:         b.transform.position = basePosition + transform.right * 3;
1074:         LinkedList c = (LinkedList)Node.Create(3, 1);
1075:         c.transform.position = basePosition + transform.right * 6;
1076:         LinkedList d = (LinkedList)Node.Create(4, 1);
1077:         d.transform.position = basePosition + transform.right * 9;
1078:
1079:         Link aL = Link.Connect(a, b);
1080:         Link bL = Link.Connect(b, c);
1081:         Link cL = Link.Connect(c, d);
1082:
1083:         Link.Flip(bL);
1084:         Link.Flip(aL);
1085:         Link.Flip(cL);
1086:         Link.Flip(cL);
1087:
1088:     }
1089:
1090:     private void TestConnectionLL2()
```

```
1091:     {
1092:         LinkedList a = (LinkedList)Node.Create(1, 1);
1093:         a.transform.position = basePosition;
1094:         LinkedList b = (LinkedList)Node.Create(2, 1);
1095:         b.transform.position = basePosition + transform.right * 3;
1096:         LinkedList c = (LinkedList)Node.Create(3, 1);
1097:         c.transform.position = basePosition + transform.right * 6;
1098:         LinkedList d = (LinkedList)Node.Create(4, 1);
1099:         d.transform.position = basePosition + transform.right * 9;
1100:
1101:         Link.Connect(a, b);
1102:         Link.Connect(a, c);
1103:         Link.Connect(b, c);
1104:
1105:     }
1106:
1107:     private void TestLHI()
1108:     {
1109:         Actions ac = GetComponent<Actions>();
1110:         ac.GetInfo();
1111:         //LeftHandInfo lc = GetComponent<LeftHandInfo>();
1112:
1113:         ac.SetInfo("test1");
1114:         ac.SetInfo("test2", 5);
1115:         //ac.SetInfo("test3", 3, 0.1f, 0.1f);
1116:     }
1117:
1118:     private void TestLLFromArray()
1119:     {
1120:         int[] arr = { 5, 10, 15, 20, 25, 30 };
1121:         LinkedListStructure.FromArray(arr);
1122:         LinkedListStructure.FromArray(arr, Vector3.up * 4);
1123:         LinkedListStructure.FromArray(arr, Vector3.up * 6, 3);
1124:     }
1125:     private void TestLinkScale()
1126:     {
1127:         int[] arr = { 5, 10, 15, 20, 25, 30 };
1128:         LinkedListStructure.FromArray(arr, Vector3.up * 2, 5);
1129:         LinkedListStructure.FromArray(arr, Vector3.up * 4, 10);
1130:         LinkedListStructure.FromArray(arr, Vector3.up * 6, 15);
1131:     }
1132:
1133:     private void TestBacklog1()
1134:     {
1135:         //unitTests.Add(TestBasicConnection);
1136:         //unitTests.Add(TestConnectionOverwriting1);
1137:         //unitTests.Add(TestConnectionDirectedness);
1138:         //unitTests.Add(TestBasicReconnection);
1139:         //unitTests.Add(TestBasicFlipping);
1140:         //unitTests.Add(TestFlippingLL);
1141:         //unitTests.Add(TestLHI);
1142:         //unitTests.Add(TestConnectionLL2);
1143:         //unitTests.Add(TestLLFromArray);
1144:         //unitTests.Add(TestLinkScale);
1145:         //unitTests.Add(TestBinaryTreeStructure);
1146:         //unitTests.Add(TestGraphConnection);
1147:         //unitTests.Add(TestGraphConnectionFlip);
1148:         //unitTests.Add(TestLLStructure1);
1149:         //unitTests.Add(TestLLStructure2);
1150:         //unitTests.Add(TestLLStructure3);
1151:         //unitTests.Add(TestLLStructure4);
1152:         //unitTests.Add(TestBTStructure1);
1153:         //unitTests.Add(TestBTStructure2);
1154:         //unitTests.Add(TestBTStructure4);
1155:         //unitTests.Add(TestLLStructure5);
1156:         //unitTests.Add(TestConnectionDirectionOnConnection);
1157:         //unitTests.Add(TestDigraphOrganization);
1158:         //unitTests.Add(TestDigraphOrganization2);
```

```
1159:         //unitTests.Add(TestDigraphOrganization3);
1160:         //unitTests.Add(TestDigraphOrganizationRandom);
1161:         //unitTests.Add(TestUndigraphOrganization);
1162:         //unitTests.Add(TestDemoStructuresPanel);
1163:         //unitTests.Add(TestDemoStructures);
1164:         //unitTests.Add(TestDemo2);
1165:         //unitTests.Add(TestRecursionSelection1);
1166:         //unitTests.Add(TestFullBinaryTree);
1167:         //unitTests.Add(TestCompleteBinaryTree);
1168:         //unitTests.Add(TestPerfectCheck);
1169:
1170:
1171:         //Algorithm Testing:
1172:         //testCoroutines.Add(TestIteration());
1173:         //testCoroutines.Add(TestIteration2());
1174:         //testCoroutines.Add(TestSelection());
1175:         //testCoroutines.Add(TestFullBinaryTreeCoR());
1176:         //testCoroutines.Add(TestCompleteBinaryTreeCoR());
1177:         //testCoroutines.Add(TestSequenceCoR());
1178:         //testCoroutines.Add(TestPerfectionCoR());
1179:     }
1180: }
```

```
    1: using System.Collections;
    2: using System.Collections.Generic;
    3: using UnityEngine;
    4:
    5: public class BinaryTreeTutorial : Tutorial
    6: {
    7:     // Start is called before the first frame update
    8:     void Start()
    9:     {
   10:         base.Start();
   11:     }
   12:
   13:     public override void InitTutorialSteps()
   14:     {
   15:         //TutorialStep[] tutorialStepsArray = { StartStep, WhatISBT1Step, WhatISBT2S
tep, BTProperties1Step, BTFullStep };
   16:         TutorialStep[] tutorialStepsArray = { StartStep, WhatISBT1Step, WhatISBT2Ste
p, BTProperties1Step, BTFullStep, BTCompleteStep, BTPerfectStep, BTStartQuizStep };
   17:         tutorialSteps = new List<TutorialStep>(tutorialStepsArray);
   18:
   19:         TutorialStep[] quizSteps = { BTDemoQuizStep, BTQuizAlertStep, BTQuizEasy1, B
TQuizEasy2, BTQuizEasy3, BTQuiz4, BTQuiz5, BTQuiz6, BTQuiz7, FinishStep };
   20:         tutorialSteps.AddRange(quizSteps);
   21:     }
   22:
   23:     public override void SetTutorialReferences()
   24:     {
   25:         base.SetTutorialReferences();
   26:     }
   27:
   28:
   29:     public override void RunTutorial()
   30:     {
   31:         base.RunTutorial();
   32:     }
   33:
   34:     #region TutorialSteps
   35:     public IEnumerator StartStep()
   36:     {
   37:         actions.SetCSEnabled(false);
   38:         restrictCSToggle = true;
   39:
   40:         worldspaceTempText.transform.position = Camera.main.transform.position + bas
eOffset;
   41:         worldspaceTempText.setTempText($"Welcome to the lesson on Binary Trees. To b
egin, please press {(actions.vr ? "both triggers" : "Ctrl+Space")}. After this, you can jus
t advance the tutorial normally.");
   42:         yield return WaitForAttentive();
   43:         //tempText.setTempText("Test B");
   44:
   45:         tempText.setTempText("");
   46:     }
   47:
   48:     public IEnumerator WhatISBT1Step()
   49:     {
   50:         BinaryTreeStructure ex1 = BinaryTreeStructure.FromArray(new int[]{1,2,3});
   51:         ex1.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward;
   52:         worldspaceTempText.tempText.transform.position = ex1.transform.position + Ve
ctor3.left * 5;
   53:         tempText.setTempText("");
   54:         worldspaceTempText.setTempText("This is a binary tree.");
   55:         yield return WaitForNextStep();
   56:         worldspaceTempText.setTempText("Formally, the defitinion of a binary tree is
\n 1. A tree consisting of no vertices (the empty tree) is a binary tree\n 2. A vertex toge
ther with two subtrees that are both binary trees is a binary tree. The subtrees are called
 the left and right subtrees of the binary tree.");
   57:         yield return WaitForNextStep();
```

```
   58:         worldspaceTempText.setTempText("Basically, a binary tree is a tree where eac
h node has at most two children.");
   59:         yield return WaitForNextStep();
   60:         Destroy(ex1.gameObject);
   61:     }
   62:
   63:     public IEnumerator WhatISBT2Step()
   64:     {
   65:         BinaryTreeStructure ex2 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 0).GetComponent<BinaryTreeStructure>();
   66:         ex2.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward * 2;
   67:         worldspaceTempText.tempText.transform.position = ex2.transform.position + Ve
ctor3.left * 5;
   68:         worldspaceTempText.setTempText("Here's a binary tree");
   69:         yield return WaitForNextStep();
   70:
   71:         Destroy(ex2.gameObject);
   72:         BinaryTreeStructure ex3 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 1).GetComponent<BinaryTreeStructure>();
   73:         ex3.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward * 3;
   74:         worldspaceTempText.tempText.transform.position = ex3.transform.position + Ve
ctor3.left * 5;
   75:         worldspaceTempText.setTempText("Here's another binary tree");
   76:         yield return WaitForNextStep();
   77:
   78:         Destroy(ex3.gameObject);
   79:         BinaryTreeStructure ex4 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 2).GetComponent<BinaryTreeStructure>();
   80:         ex4.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward * 2;
   81:         worldspaceTempText.tempText.transform.position = ex4.transform.position + Ve
ctor3.left * 5;
   82:         worldspaceTempText.setTempText("And here's another binary tree.");
   83:         yield return WaitForNextStep();
   84:
   85:         Destroy(ex4.gameObject);
   86:     }
   87:
   88:     public IEnumerator BTProperties1Step()
   89:     {
   90:         worldspaceTempText.tempText.transform.position = Camera.main.transform.posit
ion + baseOffset + Vector3.forward;
   91:         worldspaceTempText.setTempText("There are a few important types of binary tr
ees.");
   92:         yield return WaitForNextStep();
   93:         worldspaceTempText.setTempText("There are full binary trees, complete binary
 trees, and perfect binary trees. Let's start with full.");
   94:         yield return WaitForNextStep();
   95:     }
   96:     public IEnumerator BTFullStep()
   97:     {
   98:         worldspaceTempText.tempText.transform.position = Camera.main.transform.posit
ion + baseOffset + Vector3.forward * 1;
   99:         worldspaceTempText.setTempText("A full binary tree is a binary tree in which
 no node has just one child. Every node must have two or zero children.");
  100:         yield return WaitForNextStep();
  101:         BinaryTreeStructure ex3 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 0).GetComponent<BinaryTreeStructure>();
  102:         ex3.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward;
  103:         worldspaceTempText.tempText.transform.position = ex3.transform.position + Ve
ctor3.left * 5;
  104:         worldspaceTempText.setTempText("For example, this is a full binary tree.");
  105:         yield return WaitForNextStep();
  106:         worldspaceTempText.setTempText("Note that this binary tree has no nodes with
 just one child.");
```

```
  107:         yield return WaitForNextStep();
  108:
  109:         Destroy(ex3.gameObject);
  110:     }
  111:
  112:     public IEnumerator BTCompleteStep()
  113:     {
  114:         worldspaceTempText.tempText.transform.position = Camera.main.transform.posit
ion + baseOffset + Vector3.forward;
  115:         worldspaceTempText.setTempText("Next, complete binary trees. A complete bina
ry tree is a binary tree in each level is full, except the last level, in which all nodes m
ust fill from the left.");
  116:         yield return WaitForNextStep();
  117:         BinaryTreeStructure ex3 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 1).GetComponent<BinaryTreeStructure>();
  118:         ex3.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward;
  119:         worldspaceTempText.tempText.transform.position = ex3.transform.position + Ve
ctor3.left * 5;
  120:         worldspaceTempText.setTempText("For example, this is a complete binary tree.
");
  121:         yield return WaitForNextStep();
  122:         worldspaceTempText.setTempText("Note that this binary tree has all full leve
ls, except the last, in which it is full starting from the left.");
  123:         yield return WaitForNextStep();
  124:
  125:         Destroy(ex3.gameObject);
  126:     }
  127:
  128:     public IEnumerator BTPerfectStep()
  129:     {
  130:         worldspaceTempText.tempText.transform.position = Camera.main.transform.posit
ion + baseOffset;
  131:         worldspaceTempText.setTempText("Finally, perfect binary trees. A perfect bin
ary tree is both perfect and complete. Consequently, they will have 2^N - 1 nodes where N i
s the height.");
  132:         yield return WaitForNextStep();
  133:         BinaryTreeStructure ex3 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 2).GetComponent<BinaryTreeStructure>();
  134:         ex3.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward;
  135:         worldspaceTempText.tempText.transform.position = ex3.transform.position + Ve
ctor3.left * 5;
  136:         worldspaceTempText.setTempText("For example, this is a perfect binary tree."
);
  137:         yield return WaitForNextStep();
  138:         worldspaceTempText.setTempText("Note that this binary tree has all full leve
ls, and each node has exactly two or zero children.");
  139:         yield return WaitForNextStep();
  140:         worldspaceTempText.setTempText("Additionally, you can see that this binary t
ree has 2^3-1 (7) nodes, because it has three levels.");
  141:         yield return WaitForNextStep();
  142:
  143:         Destroy(ex3.gameObject);
  144:     }
  145:
  146:     public IEnumerator BTStartQuizStep()
  147:     {
  148:         worldspaceTempText.setTempText("Now we're going to start the quiz.");
  149:         yield return WaitForNextStep();
  150:     }
  151:
  152:     public IEnumerator BTDemoQuizStep()
  153:     {
  154:         //Create bt
  155:         BinaryTreeStructure ex3 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 1).GetComponent<BinaryTreeStructure>();
  156:         ex3.transform.position = Camera.main.transform.position + baseOffset + Vecto
```

```
r3.forward;
  157:
  158:         //Position text
  159:         worldspaceTempText.tempText.transform.position = ex3.transform.position + Ve
ctor3.right * 5 + Vector3.down * 2;
  160:         worldspaceTempText.setTempText("Here's a sample quiz question.");
  161:
  162:         Quiz quiz = Quiz.CreateQuiz("How many nodes does this binary tree have?", ne
w string[] { "5 Nodes", "6 Nodes", "7 Nodes" }, 1);
  163:         quiz.transform.position = ex3.transform.position + Vector3.left * 5 + Vector
3.down * 2;
  164:         quiz.qIdentifier = "SAMPLE";
  165:         yield return quiz.WaitForQuiz();
  166:
  167:         //Cleanup
  168:         Destroy(quiz.gameObject);
  169:         Destroy(ex3.gameObject);
  170:     }
  171:
  172:     public IEnumerator BTQuizAlertStep()
  173:     {
  174:         worldspaceTempText.tempText.transform.position = Camera.main.transform.posit
ion + baseOffset;
  175:         worldspaceTempText.setTempText("Ok, now we'll start the actual quiz. Here we
 go!");
  176:         yield return WaitForNextStep();
  177:     }
  178:
  179:     public IEnumerator BTQuizEasy1()
  180:     {
  181:         BinaryTreeStructure ex3 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 0).GetComponent<BinaryTreeStructure>();
  182:         ex3.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward;
  183:
  184:         //Position text
  185:         worldspaceTempText.tempText.transform.position = ex3.transform.position + Ve
ctor3.right * 5 + Vector3.down * 2;
  186:         worldspaceTempText.setTempText("Question 1");
  187:
  188:         Quiz quiz = Quiz.CreateQuiz("Is this binary tree full, complete, or perfect?
", new string[] { "Full", "Complete", "Perfect" }, 0);
  189:         quiz.transform.position = ex3.transform.position + Vector3.left * 5 + Vector
3.down * 2;
  190:         quiz.qIdentifier = "BT 1";
  191:         yield return quiz.WaitForQuiz();
  192:
  193:         //Cleanup
  194:         Destroy(quiz.gameObject);
  195:         Destroy(ex3.gameObject);
  196:     }
  197:
  198:     public IEnumerator BTQuizEasy2()
  199:     {
  200:         BinaryTreeStructure ex3 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 1).GetComponent<BinaryTreeStructure>();
  201:         ex3.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward;
  202:
  203:         //Position text
  204:         worldspaceTempText.tempText.transform.position = ex3.transform.position + Ve
ctor3.right * 5 + Vector3.down * 2;
  205:         worldspaceTempText.setTempText("Question 2");
  206:
  207:         Quiz quiz = Quiz.CreateQuiz("Is this binary tree full, complete, or perfect?
", new string[] { "Full", "Complete", "Perfect" }, 1);
  208:         quiz.transform.position = ex3.transform.position + Vector3.left * 5 + Vector
3.down * 2;
```

```
  209:         quiz.qIdentifier = "BT 2";
  210:         yield return quiz.WaitForQuiz();
  211:
  212:         //Cleanup
  213:         Destroy(quiz.gameObject);
  214:         Destroy(ex3.gameObject);
  215:     }
  216:
  217:     public IEnumerator BTQuizEasy3()
  218:     {
  219:         BinaryTreeStructure ex3 = DemoStructures.CreateDemo(DemoStructures.NodeType.
BinaryTree, 2).GetComponent<BinaryTreeStructure>();
  220:         ex3.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.forward;
  221:
  222:         //Position text
  223:         worldspaceTempText.tempText.transform.position = ex3.transform.position + Ve
ctor3.right * 5 + Vector3.down * 2;
  224:         worldspaceTempText.setTempText("Question 3");
  225:
  226:         Quiz quiz = Quiz.CreateQuiz("Is this binary tree full, complete, or perfect?
", new string[] { "Full", "Complete", "Perfect" }, 2);
  227:         quiz.transform.position = ex3.transform.position + Vector3.left * 5 + Vector
3.down * 2;
  228:         quiz.qIdentifier = "BT 3";
  229:         yield return quiz.WaitForQuiz();
  230:
  231:         //Cleanup
  232:         Destroy(quiz.gameObject);
  233:         Destroy(ex3.gameObject);
  234:
  235:         //ForceOpenCreation();
  236:     }
  237:
  238:     public IEnumerator BTQuiz4()
  239:     {
  240:         BinaryTreeStructure bt = BinaryTreeStructure.FromArray(new int[]{ 1,2,3, -1,
 -1, 6 });
  241:         bt.transform.position = Camera.main.transform.position + baseOffset + Vector
3.forward;
  242:
  243:         //Position text
  244:         worldspaceTempText.tempText.transform.position = bt.transform.position + Vec
tor3.right * 5 + Vector3.down * 2;
  245:         worldspaceTempText.setTempText("Question 4");
  246:
  247:         Quiz quiz = Quiz.CreateQuiz("What would need to be added to make this tree F
ULL?", new string[] { "Nothing, it already is. ", "Add two nodes 4 and 5, under 3.", "Remov
e 6 " }, 2);
  248:         quiz.transform.position = bt.transform.position + Vector3.left * 5 + Vector3
.down * 2;
  249:         quiz.qIdentifier = "BT 4";
  250:         yield return quiz.WaitForQuiz();
  251:
  252:         //Cleanup
  253:         Destroy(quiz.gameObject);
  254:         Destroy(bt.gameObject);
  255:     }
  256:
  257:     public IEnumerator BTQuiz5()
  258:     {
  259:         BinaryTreeStructure bt = BinaryTreeStructure.FromArray(new int[] { 1, 2, 3,
-1, -1, 6, 7 });
  260:         bt.transform.position = Camera.main.transform.position + baseOffset + Vector
3.forward;
  261:
  262:         //Position text
  263:         worldspaceTempText.tempText.transform.position = bt.transform.position + Vec
```

```
tor3.right * 5 + Vector3.down * 2;
  264:         worldspaceTempText.setTempText("Question 5");
  265:
  266:         Quiz quiz = Quiz.CreateQuiz("What would need to be added to make this tree C
OMPLETE?", new string[] { "Nothing, it already is. ", "Add two nodes 4 and 5, under 3.", "R
emove 6 " }, 1);
  267:         quiz.transform.position = bt.transform.position + Vector3.left * 5 + Vector3
.down * 2;
  268:         quiz.qIdentifier = "BT 5";
  269:         yield return quiz.WaitForQuiz();
  270:
  271:         //Cleanup
  272:         Destroy(quiz.gameObject);
  273:         Destroy(bt.gameObject);
  274:     }
  275:
  276:     public IEnumerator BTQuiz6()
  277:     {
  278:         BinaryTreeStructure bt = BinaryTreeStructure.FromArray(new int[] { 1, 2, 3,
-1, -1, 6 });
  279:         bt.transform.position = Camera.main.transform.position + baseOffset + Vector
3.forward;
  280:
  281:         //Position text
  282:         worldspaceTempText.tempText.transform.position = bt.transform.position + Vec
tor3.right * 5 + Vector3.down * 2;
  283:         worldspaceTempText.setTempText("Question 6");
  284:
  285:         Quiz quiz = Quiz.CreateQuiz("What would need to be added to make this tree P
ERFECT?", new string[] { "Nothing, it already is. ", "Add two nodes 4 and 5, under 3.", "Re
move 6 " }, 2);
  286:         quiz.transform.position = bt.transform.position + Vector3.left * 5 + Vector3
.down * 2;
  287:         quiz.qIdentifier = "BT 6";
  288:         yield return quiz.WaitForQuiz();
  289:
  290:         //Cleanup
  291:         Destroy(quiz.gameObject);
  292:         Destroy(bt.gameObject);
  293:     }
  294:
  295:     public IEnumerator BTQuiz7()
  296:     {
  297:         Quiz quiz = Quiz.CreateQuiz("How many nodes does a perfect binary tree have?
", new string[] { "2^n-1 ", "2^n", "2^n+1" }, 0);
  298:         quiz.transform.position = Camera.main.transform.position + baseOffset + Vect
or3.forward + Vector3.down * 2;
  299:         quiz.qIdentifier = "BT 7";
  300:         yield return quiz.WaitForQuiz();
  301:
  302:         //Cleanup
  303:         Destroy(quiz.gameObject);
  304:     }
  305:
  306:     private void ForceOpenCreation()
  307:     {
  308:         Tutorial.restrictCSToggle = false;
  309:         Tutorial.restrictPanelChange = true;
  310:         actions.SetCSEnabled(true);
  311:         actions.uiManager.creationPanel.gameObject.SetActive(true);
  312:         actions.uiManager.transform.GetChild(0).gameObject.SetActive(true);
  313:     }
  314:
  315:
  316:
  317:     #endregion
  318: }
```

```
    1: ï»¿using System;
    2: using System.Collections;
    3: using System.Collections.Generic;
    4: using System.Linq;
    5: using System.Text;
    6: using System.Threading.Tasks;
    7: using UnityEngine;
    8:
    9: class GraphsTutorial : Tutorial
   10: {
   11:     DigraphStructure ug1;
   12:
   13:     // Start is called before the first frame update
   14:     void Start()
   15:     {
   16:         base.Start();
   17:     }
   18:
   19:     public override void InitTutorialSteps()
   20:     {
   21:         TutorialStep[] tutorialStepsArray = { StartStep, IntroStep1, IntroGraphStep1
, IntroGraphStep2, IntroGraphStep3, SearchesIntroStep, DepthSetStep, DepthFirstSearchIntro,
 DepthFirstSearchIntro2, BreadthFirstSearch1, BreadthFirstSearch2 };
   22:         tutorialSteps = new List<TutorialStep>(tutorialStepsArray);
   23:
   24:         TutorialStep[] quizSteps = { QuizStartStep, DemoQuizStep, QuizAlertStep, Qui
zQ1BStep, QuizQ2BStep, QuizQ3BStep, QuizQ1DStep, QuizQ2DStep, QuizQ3DStep, FinishStep };
   25:         //tutorialSteps = new List<TutorialStep>(quizSteps);
   26:         tutorialSteps.AddRange(quizSteps);
   27:     }
   28:
   29:     public override void SetTutorialReferences()
   30:     {
   31:         base.SetTutorialReferences();
   32:     }
   33:
   34:
   35:     public override void RunTutorial()
   36:     {
   37:         base.RunTutorial();
   38:     }
   39:
   40:     #region TutorialSteps
   41:     public IEnumerator StartStep()
   42:     {
   43:         actions.SetCSEnabled(false);
   44:         restrictCSToggle = true;
   45:         worldspaceTempText.transform.position = Camera.main.transform.position + bas
eOffset;
   46:         worldspaceTempText.setTempText($"Welcome to the lesson on Graphs and Searche
s. To begin, please press {(actions.vr ? "both triggers" : "Ctrl + Space")}. After this, yo
u'll just need to press space to advance the tutorial.");
   47:         yield return WaitForAttentive();
   48:
   49:         tempText.setTempText("");
   50:     }
   51:
   52:     public IEnumerator IntroStep1()
   53:     {
   54:         tempText.setTempText("");
   55:         worldspaceTempText.setTempText("What is a graph?");
   56:         yield return WaitForNextStep();
   57:         worldspaceTempText.setTempText("Formally, a graph is a pair of sets G(V,E),
one of Vertices (V), and the other of Edges (E) that connect them.");
   58:         yield return WaitForNextStep();
   59:         worldspaceTempText.setTempText("Basically, a graph is a set of nodes that ha
ve some value, with connections between them.");
   60:         yield return WaitForNextStep();
```

```
   61:         worldspaceTempText.setTempText("Graphs are useful for a huge number of appli
cations in computer science. Path-finding, resource allocation, protein structures, users a
nd their friends - etc.");
   62:         yield return WaitForNextStep();
   63:         worldspaceTempText.setTempText("One easy way to understand graphs is airport
s and flights. Each airport is a node (vertex), and each flight that connects them is a lin
k (edge).");
   64:         yield return WaitForNextStep();
   65:         worldspaceTempText.setTempText("Almost anything that involves a connection b
etween objects can be represented by a graph.");
   66:         yield return WaitForNextStep();
   67:         tempText.setTempText("");
   68:     }
   69:
   70:
   71:     public IEnumerator IntroGraphStep1()
   72:     {
   73:         worldspaceTempText.setTempText("Lets take a look at a few example graphs.");
   74:         yield return WaitForNextStep();
   75:         ug1 = DemoStructures.CreateDemo(DemoStructures.NodeType.Digraph, 0).GetCompo
nent<DigraphStructure>();
   76:         ug1.transform.position = Camera.main.transform.position + new Vector3(-5, 0,
 distDepth);
   77:         worldspaceTempText.tempText.transform.position = ug1.transform.position + Ve
ctor3.right * 5 + Vector3.up * 2;
   78:         worldspaceTempText.setTempText("Here's an example graph.");
   79:         yield return WaitForNextStep();
   80:         tempText.setTempText("");
   81:         Destroy(ug1.gameObject);
   82:         yield return new WaitForSeconds(1);
   83:     }
   84:
   85:     public IEnumerator IntroGraphStep2()
   86:     {
   87:         ug1 = DemoStructures.CreateDemo(DemoStructures.NodeType.Digraph, 1).GetCompo
nent<DigraphStructure>();
   88:         ug1.transform.position = Camera.main.transform.position + new Vector3(-5, 0,
 distDepth);
   89:         worldspaceTempText.tempText.transform.position = ug1.transform.position + Ve
ctor3.right * 5 + Vector3.up * 2;
   90:         worldspaceTempText.setTempText("Here's another, more complicated example gra
ph.");
   91:         yield return WaitForNextStep();
   92:         tempText.setTempText("");
   93:         Destroy(ug1.gameObject);
   94:         yield return new WaitForSeconds(1);
   95:     }
   96:
   97:     public IEnumerator IntroGraphStep3()
   98:     {
   99:         ug1 = helperGraph2();
  100:         ug1.transform.position = Camera.main.transform.position + new Vector3(-2.5f,
 2, distDepth + 3);
  101:         worldspaceTempText.tempText.transform.position = ug1.transform.position + Ve
ctor3.right * 5 + Vector3.up * 2;
  102:         worldspaceTempText.setTempText("And here's our final example graph that we'l
l use to learn about searches.");
  103:         yield return WaitForNextStep();
  104:         tempText.setTempText("");
  105:         //Destroy(ug1.gameObject);
  106:         //yield return new WaitForSeconds(1);
  107:     }
  108:
  109:     public IEnumerator SearchesIntroStep()
  110:     {
  111:         worldspaceTempText.tempText.transform.position = Camera.main.transform.posit
ion + Vector3.forward * distDepth + Vector3.down * 2 + Vector3.right * 2;
  112:         worldspaceTempText.setTempText("The most common operation on a graph is goin
```

```
g to be a search. Searching is fundamental to almost all other algorithms that operate on g
raphs, so it's important to understand.");
  113:         yield return WaitForNextStep();
  114:         worldspaceTempText.setTempText("Let's learn about the two basic graph search
es - depth-first search and breadth-first search.");
  115:         yield return WaitForNextStep();
  116:
  117:     }
  118:
  119:     public IEnumerator DepthSetStep()
  120:     {
  121:         worldspaceTempText.setTempText("Before we look at searches, it's important t
o have some understanding of depth sets.");
  122:         yield return WaitForNextStep();
  123:         worldspaceTempText.setTempText("You can imagine that from any starting point
 on a graph, you can organize nodes into sets based on how far away they are.");
  124:         yield return WaitForNextStep();
  125:         worldspaceTempText.setTempText("The 0th depth set is the starting node, and
then the first depth set is any node connected to that starting node.");
  126:         yield return WaitForNextStep();
  127:         worldspaceTempText.setTempText("Then, the second depth set is any nodes conn
ected to those - meaning they're two steps away from the original node.");
  128:         yield return WaitForNextStep();
  129:         worldspaceTempText.setTempText("Though this might not seem useful, keep it i
n mind, as it is an important caveat of the different searches.");
  130:         yield return WaitForNextStep();
  131:
  132:     }
  133:
  134:     public IEnumerator DepthFirstSearchIntro()
  135:     {
  136:         worldspaceTempText.setTempText("What is a depth-first search?");
  137:         yield return WaitForNextStep();
  138:         worldspaceTempText.setTempText("A depth first search involves looking as dee
p as possible before backtracking.");
  139:         yield return WaitForNextStep();
  140:         worldspaceTempText.setTempText("Lets watch it play out on our graph. Lets se
arch for 10.");
  141:         yield return WaitForNextStep();
  142:         Algorithms.waitTime = 0.5f;
  143:         Algorithms.iterationMode = Algorithms.IterationMode.Wait;
  144:         yield return algorithms.RunAlgorithm(UIManager.NodeType.Digraph, 0, ug1, 10)
;
  145:         worldspaceTempText.setTempText("As you can see, it sees one first, then cont
inues down that branch.");
  146:         yield return WaitForNextStep();
  147:     }
  148:
  149:     public IEnumerator DepthFirstSearchIntro2()
  150:     {
  151:         worldspaceTempText.setTempText("Lets look at another example. Lets search fo
r 6.");
  152:         yield return WaitForNextStep();
  153:         Algorithms.waitTime = 0.5f;
  154:         Algorithms.iterationMode = Algorithms.IterationMode.Wait;
  155:         yield return algorithms.RunAlgorithm(UIManager.NodeType.Digraph, 0, ug1, 6);
  156:         worldspaceTempText.setTempText("As you can see, it continues to the depth of
 each branch, before moving onto the next one.");
  157:         yield return WaitForNextStep();
  158:         worldspaceTempText.setTempText("This is the key to depth-first search - it e
xplores the full depth of each branch before moving on to a different one.");
  159:     }
  160:
  161:     public IEnumerator BreadthFirstSearch1()
  162:     {
  163:         worldspaceTempText.setTempText("Now, let's consider breadth-first search. Un
like depth-first search, breadth-first search explores the full breadth before moving on to
 any further depth.");
```

```
  164:         yield return WaitForNextStep();
  165:         worldspaceTempText.setTempText("To understand this, lets return to our conce
pt of depth sets. First, breadth-first search will explore the first depth set - all nodes
attached to the root node.");
  166:         yield return WaitForNextStep();
  167:         worldspaceTempText.setTempText("Lets watch it play out on our graph. Lets ag
ain search for 10.");
  168:         yield return WaitForNextStep();
  169:         Algorithms.waitTime = 0.5f;
  170:         Algorithms.iterationMode = Algorithms.IterationMode.Wait;
  171:         yield return algorithms.RunAlgorithm(UIManager.NodeType.Digraph, 1, ug1, 10)
;
  172:         worldspaceTempText.setTempText("As you can see, unlike with a depth-first se
arch, breadth-first search looks at all the nodes connected to the root node first, and con
tinues that way.");
  173:         yield return WaitForNextStep();
  174:     }
  175:
  176:     public IEnumerator BreadthFirstSearch2()
  177:     {
  178:         worldspaceTempText.setTempText("Now, let's consider our other example. Lets
search for 6.");
  179:         yield return WaitForNextStep();
  180:         Algorithms.waitTime = 0.5f;
  181:         Algorithms.iterationMode = Algorithms.IterationMode.Wait;
  182:         yield return algorithms.RunAlgorithm(UIManager.NodeType.Digraph, 1, ug1, 6);
  183:         worldspaceTempText.setTempText("Unlike depth-first search, which performed p
oorly in this case, breadth-first search finds it fairly quickly.");
  184:         yield return WaitForNextStep();
  185:         worldspaceTempText.setTempText("Of course, one isn't plainly better than the
 other, instead, they are just two options for searching.");
  186:         yield return WaitForNextStep();
  187:     }
  188:
  189:     public IEnumerator QuizStartStep()
  190:     {
  191:         worldspaceTempText.setTempText("Now we're going to start the quiz! Let's sta
rt with a demo quiz question.");
  192:         yield return WaitForNextStep();
  193:     }
  194:
  195:     public IEnumerator DemoQuizStep()
  196:     {
  197:         if (ug1)
  198:             Destroy(ug1.gameObject);
  199:
  200:         ug1 = helperGraph2();
  201:         ug1.transform.position = Camera.main.transform.position + new Vector3(-2.5f,
 2, distDepth + 3);
  202:         worldspaceTempText.tempText.transform.position = ug1.transform.position + Ve
ctor3.right * 5 + Vector3.up * 2;
  203:         Quiz quiz = Quiz.CreateQuiz("How many nodes does this graph have?", new stri
ng[] { "11 Nodes", "12 Nodes", "13 Nodes" }, 1);
  204:         quiz.transform.position = ug1.transform.position + Vector3.right * 3 + Vecto
r3.down * 4.5f;
  205:         quiz.qIdentifier = "SAMPLE";
  206:         yield return quiz.WaitForQuiz();
  207:         Destroy(quiz.gameObject);
  208:     }
  209:
  210:     public IEnumerator QuizAlertStep()
  211:     {
  212:         worldspaceTempText.tempText.transform.position = Camera.main.transform.posit
ion + new Vector3(0, 2, distDepth);
  213:         worldspaceTempText.setTempText("Ok, now we'll start the actual quiz. Here we
 go!");
  214:         yield return WaitForNextStep();
  215:         worldspaceTempText.setTempText("");
```

```
  216:     }
  217:
  218:     public IEnumerator QuizQ1BStep()
  219:     {
  220:         Quiz quiz = Quiz.CreateQuiz("Which node would breadth-first search find firs
t?", new string[] { "5", "7", "11" }, 0);
  221:         quiz.transform.position = ug1.transform.position + Vector3.right * 3 + Vecto
r3.down * 4.5f;
  222:         quiz.qIdentifier = "GT 1";
  223:         yield return quiz.WaitForQuiz();
  224:         ug1.gameObject.SetActive(false);
  225:         yield return new WaitForSeconds(1);
  226:         ug1.gameObject.SetActive(true);
  227:
  228:         Destroy(quiz.gameObject);
  229:     }
  230:
  231:     public IEnumerator QuizQ2BStep()
  232:     {
  233:         Quiz quiz = Quiz.CreateQuiz("Which node would breadth-first search find firs
t?", new string[] { "7", "3", "8" }, 1);
  234:         quiz.transform.position = ug1.transform.position + Vector3.right * 3 + Vecto
r3.down * 4.5f;
  235:         quiz.qIdentifier = "GT 2";
  236:         yield return quiz.WaitForQuiz();
  237:         ug1.gameObject.SetActive(false);
  238:         yield return new WaitForSeconds(1);
  239:         ug1.gameObject.SetActive(true);
  240:         Destroy(quiz.gameObject);
  241:     }
  242:
  243:     public IEnumerator QuizQ3BStep()
  244:     {
  245:         Quiz quiz = Quiz.CreateQuiz("Which node would breadth-first search find LAST
?", new string[] { "7", "8", "11" }, 2);
  246:         quiz.transform.position = ug1.transform.position + Vector3.right * 3 + Vecto
r3.down * 4.5f;
  247:         quiz.qIdentifier = "GT 3";
  248:         yield return quiz.WaitForQuiz();
  249:         ug1.gameObject.SetActive(false);
  250:         yield return new WaitForSeconds(1);
  251:         ug1.gameObject.SetActive(true);
  252:         Destroy(quiz.gameObject);
  253:     }
  254:
  255:     public IEnumerator QuizQ1DStep()
  256:     {
  257:         Quiz quiz = Quiz.CreateQuiz("Which node would depth-first search find first?
", new string[] { "5", "7", "11" }, 2);
  258:         quiz.transform.position = ug1.transform.position + Vector3.right * 3 + Vecto
r3.down * 4.5f;
  259:         quiz.qIdentifier = "GT 4";
  260:         yield return quiz.WaitForQuiz();
  261:         ug1.gameObject.SetActive(false);
  262:         yield return new WaitForSeconds(1);
  263:         ug1.gameObject.SetActive(true);
  264:         Destroy(quiz.gameObject);
  265:     }
  266:
  267:     public IEnumerator QuizQ2DStep()
  268:     {
  269:         Quiz quiz = Quiz.CreateQuiz("Which node would depth-first search find first?
", new string[] { "9", "4", "12" }, 0);
  270:         quiz.transform.position = ug1.transform.position + Vector3.right * 3 + Vecto
r3.down * 4.5f;
  271:         quiz.qIdentifier = "GT 5";
  272:         yield return quiz.WaitForQuiz();
  273:         ug1.gameObject.SetActive(false);
```

```
  274:         yield return new WaitForSeconds(1);
  275:         ug1.gameObject.SetActive(true);
  276:         Destroy(quiz.gameObject);
  277:     }
  278:
  279:     public IEnumerator QuizQ3DStep()
  280:     {
  281:         Quiz quiz = Quiz.CreateQuiz("Which node would depth-first search find LAST?"
, new string[] { "7", "8", "9" }, 1);
  282:         quiz.transform.position = ug1.transform.position + Vector3.right * 3 + Vecto
r3.down * 4.5f;
  283:         quiz.qIdentifier = "GT 6";
  284:         yield return quiz.WaitForQuiz();
  285:         ug1.gameObject.SetActive(false);
  286:         yield return new WaitForSeconds(1);
  287:         ug1.gameObject.SetActive(true);
  288:         Destroy(quiz.gameObject);
  289:     }
  290:
  291:
  292:     #endregion
  293:
  294:     public DigraphStructure helperGraph1()
  295:     {
  296:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
  297:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
  298:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
  299:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
  300:         Digraph e = (Digraph)Node.Create(5, Node.CreationSelection.Digraph);
  301:         Digraph f = (Digraph)Node.Create(6, Node.CreationSelection.Digraph);
  302:         Digraph g = (Digraph)Node.Create(7, Node.CreationSelection.Digraph);
  303:         Digraph h = (Digraph)Node.Create(8, Node.CreationSelection.Digraph);
  304:         Digraph i = (Digraph)Node.Create(9, Node.CreationSelection.Digraph);
  305:         Digraph j = (Digraph)Node.Create(10, Node.CreationSelection.Digraph);
  306:         Digraph k = (Digraph)Node.Create(11, Node.CreationSelection.Digraph);
  307:         Digraph l = (Digraph)Node.Create(12, Node.CreationSelection.Digraph);
  308:         Digraph m = (Digraph)Node.Create(13, Node.CreationSelection.Digraph);
  309:         Digraph n = (Digraph)Node.Create(14, Node.CreationSelection.Digraph);
  310:
  311:         //d1
  312:         Link.Connect(a, b);
  313:         Link.Connect(b, a);
  314:         Link.Connect(c, a);
  315:         //d2
  316:         Link.Connect(c, d);
  317:         Link.Connect(e, c);
  318:         //d3
  319:         Link.Connect(f, e);
  320:
  321:         return DigraphStructure.Organize(a);
  322:     }
  323:
  324:     public DigraphStructure helperGraph2()
  325:     {
  326:         //create digraph
  327:         Digraph a = (Digraph)Node.Create(1, Node.CreationSelection.Digraph);
  328:         Digraph b = (Digraph)Node.Create(2, Node.CreationSelection.Digraph);
  329:         Digraph c = (Digraph)Node.Create(3, Node.CreationSelection.Digraph);
  330:         Digraph d = (Digraph)Node.Create(4, Node.CreationSelection.Digraph);
  331:         Digraph e = (Digraph)Node.Create(5, Node.CreationSelection.Digraph);
  332:         Digraph f = (Digraph)Node.Create(6, Node.CreationSelection.Digraph);
  333:         Digraph g = (Digraph)Node.Create(8, Node.CreationSelection.Digraph);
  334:         Digraph h = (Digraph)Node.Create(7, Node.CreationSelection.Digraph);
  335:         Digraph i = (Digraph)Node.Create(9, Node.CreationSelection.Digraph);
  336:         Digraph j = (Digraph)Node.Create(10, Node.CreationSelection.Digraph);
  337:         Digraph k = (Digraph)Node.Create(11, Node.CreationSelection.Digraph);
  338:         Digraph l = (Digraph)Node.Create(12, Node.CreationSelection.Digraph);
  339:
```

```
340:         //depth 1
341:         Link.Connect(a, f);
342:         Link.Connect(a, e);
343:         Link.Connect(a, d);
344:         Link.Connect(a, c);
345:         Link.Connect(a, b);
346:         //depth 2
347:         Link.Connect(c, g);
348:         Link.Connect(c, h);
349:         Link.Connect(b, j);
350:         Link.Connect(f, l);
351:         //d3
352:         Link.Connect(h, i);
353:         Link.Connect(j, k);
354:
355:         DigraphStructure dgs = DigraphStructure.Organize(a);
356:
357:         return dgs;
358:     }
359: }
```

```
    1: ï»¿using System;
    2: using System.Collections;
    3: using System.Collections.Generic;
    4: using System.Linq;
    5: using System.Text;
    6: using System.Threading.Tasks;
    7: using UnityEngine;
    8:
    9: class LinkedListTutorial : Tutorial
   10: {
   11:     LinkedListStructure ll1;
   12:
   13:     public override void InitTutorialSteps()
   14:     {
   15:         TutorialStep[] tutorialStepsArray = { StartStep, LinkedListStep1, LinkedList
Step2, LLReversedExampleStep, LLReversalIntro, LLReversalNaive, LLReversalProper, LLReversa
lExplanation, QuizStartStep };//{ StartStep, LinkedListStep1 };
   16:         tutorialSteps = new List<TutorialStep>(tutorialStepsArray);
   17:
   18:         TutorialStep[] quizStepsArray = { DemoQuizStep, QuizAlertStep, QuizQ1, QuizQ
2, QuizQ3, QuizQ4, FinishStep };
   19:         tutorialSteps.AddRange(quizStepsArray);
   20:
   21:     }
   22:
   23:     public override void SetTutorialReferences()
   24:     {
   25:         base.SetTutorialReferences();
   26:     }
   27:
   28:
   29:     public override void RunTutorial()
   30:     {
   31:         base.RunTutorial();
   32:     }
   33:
   34:     public IEnumerator StartStep()
   35:     {
   36:         actions.SetCSEnabled(false);
   37:         restrictCSToggle = true;
   38:
   39:         worldspaceTempText.transform.position = Camera.main.transform.position + bas
eOffset;
   40:         worldspaceTempText.setTempText($"Welcome to the lesson on Linked Lists. To b
egin, please press {(actions.vr ? "both triggers" : "Ctrl + Space")}. After this, you'll ju
st need to press space to advance the tutorial.");
   41:         yield return WaitForAttentive();
   42:         //tempText.setTempText("Test B");
   43:
   44:         tempText.setTempText("");
   45:     }
   46:
   47:     public IEnumerator LinkedListStep1()
   48:     {
   49:         LinkedListStructure ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.
LinkedList, 0).GetComponent<LinkedListStructure>();
   50:         ll1.transform.position = Camera.main.transform.position + new Vector3(-5, 0,
 distDepth);
   51:         worldspaceTempText.tempText.transform.position = ll1.transform.position + Ve
ctor3.right * 5 + Vector3.up * 2;
   52:         tempText.setTempText("");
   53:         worldspaceTempText.setTempText("This is a linked list.");
   54:         yield return WaitForNextStep();
   55:         worldspaceTempText.setTempText("Basically, a linked list is a data structure
 that connects individual nodes to each other. For this tutorial, we'll only consider singl
y-linked lists.");
   56:         yield return WaitForNextStep();
   57:         worldspaceTempText.setTempText("A singly-linked list connects nodes in just
```

```
one direction. See how in this linked list, 1 is connected to 2, 2 is connected to 3, etc."
);
    58:         yield return WaitForNextStep();
    59:         worldspaceTempText.setTempText("The first node in the list is called the Hea
d, and the last is called the Tail.");
    60:         yield return WaitForNextStep();
    61:
    62:         //Cleanup
    63:         Destroy(ll1.gameObject);
    64:     }
    65:
    66:     public IEnumerator LinkedListStep2()
    67:     {
    68:         LinkedListStructure ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.
LinkedList, 0).GetComponent<LinkedListStructure>();
    69:         ll1.transform.position = Camera.main.transform.position + new Vector3(-5, 0,
 distDepth);
    70:         worldspaceTempText.tempText.transform.position = ll1.transform.position + Ve
ctor3.right * 5 + Vector3.up * 2;
    71:         tempText.setTempText("");
    72:         worldspaceTempText.setTempText("Linked lists have a variety of applications,
 and are a necessary part of other data structures like stacks and queues.");
    73:         yield return WaitForNextStep();
    74:         worldspaceTempText.setTempText("For this example, we'll look at a common alg
orithm, Linked List reversal.");
    75:         yield return WaitForNextStep();
    76:         worldspaceTempText.setTempText("Linked List reversal is a common algorithm,
and is exactly what it sounds like - reversing a linked list.");
    77:         yield return WaitForNextStep();
    78:         worldspaceTempText.setTempText("Let's see an example of what a list looks li
ke reversed.");
    79:         yield return WaitForNextStep();
    80:         Destroy(ll1.gameObject);
    81:     }
    82:
    83:     public IEnumerator LLReversedExampleStep()
    84:     {
    85:         LinkedListStructure ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.
LinkedList, 0).GetComponent<LinkedListStructure>();
    86:         ll1.transform.position = Camera.main.transform.position + new Vector3(-5, 0,
 distDepth) + Vector3.down*2;
    87:
    88:         LinkedListStructure ll2 = DemoStructures.CreateDemo(DemoStructures.NodeType.
LinkedList, 1).GetComponent<LinkedListStructure>();
    89:         ll2.transform.position = ll1.transform.position + Vector3.up * 2;
    90:
    91:         worldspaceTempText.tempText.transform.position = ll2.transform.position + Ve
ctor3.right * 5 + Vector3.up * 2;
    92:         worldspaceTempText.setTempText("For example, these two lists are each other'
s reversed versions.");
    93:         yield return WaitForNextStep();
    94:         worldspaceTempText.setTempText("Not too difficult!");
    95:         yield return WaitForNextStep();
    96:         worldspaceTempText.setTempText("Now, let's learn about how linked list rever
sal algorithms.");
    97:
    98:         //Cleanup
    99:         Destroy(ll1.gameObject);
   100:         Destroy(ll2.gameObject);
   101:
   102:     }
   103:
   104:     public IEnumerator LLReversalIntro()
   105:     {
   106:         worldspaceTempText.setTempText("Before we continue, think about how you migh
t solve this problem. Can you think of a solution?");
   107:         yield return WaitForNextStep();
   108:         worldspaceTempText.setTempText("Did you think of something?");
```

```
  109:         yield return WaitForNextStep();
  110:         worldspaceTempText.setTempText("Ok, now that you’ve got a solution in mind,
let’s look at the naive approach for this algorithm.");
  111:         yield return WaitForNextStep();
  112:     }
  113:
  114:     public IEnumerator LLReversalNaive()
  115:     {
  116:         worldspaceTempText.setTempText("To reverse a list, you could simply traverse
 through the list in order, while creating the reversed list in parallel.");
  117:         yield return WaitForNextStep();
  118:         worldspaceTempText.setTempText($"Let’s watch this happen.");
  119:         yield return WaitForNextStep();
  120:
  121:         LinkedListStructure ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.
LinkedList, 0).GetComponent<LinkedListStructure>();
  122:         ll1.transform.position = Camera.main.transform.position + new Vector3(-5, 0,
 distDepth);
  123:
  124:         //DIY algorithm
  125:         LinkedList h = ll1.head;
  126:         LinkedList n = null;
  127:         LinkedList newll;
  128:         LinkedList newll2;
  129:
  130:         Algorithms.Select(h);
  131:         newll = Node.Create(h.Value, Node.CreationSelection.LinkedList) as LinkedLis
t;
  132:         newll.transform.position = h.transform.position + Vector3.down * 2;
  133:
  134:         List<GameObject> all = new List<GameObject>(new GameObject[]{ newll.gameObje
ct });
  135:
  136:         for (int i = 0; i < 4; i++)
  137:         {
  138:             Algorithms.Deselect(h);
  139:             n = h.next;
  140:             Algorithms.Select(n);
  141:             yield return WaitForNextStep();
  142:             newll2 = Node.Create(n.Value, Node.CreationSelection.LinkedList) as Link
edList;
  143:             foreach(GameObject g in all)
  144:                 g.transform.position += Vector3.right * 3;
  145:             newll2.transform.position = ll1.head.transform.position + Vector3.down *
 2;
  146:             all.Add(newll2.gameObject);
  147:             Link.Connect(newll2, newll);
  148:             newll = newll2;
  149:             h = n;
  150:         }
  151:
  152:         Algorithms.Deselect(h);
  153:         worldspaceTempText.setTempText($"Now we’ve got a second copy of the list, bu
t in reverse order!");
  154:         yield return WaitForNextStep();
  155:
  156:
  157:         worldspaceTempText.setTempText($"This works, but can we do better?");
  158:         yield return WaitForNextStep();
  159:
  160:         foreach (GameObject g in all)
  161:             Destroy(g);
  162:         Destroy(ll1.gameObject);
  163:     }
  164:
  165:     public IEnumerator LLReversalProper()
  166:     {
  167:         worldspaceTempText.setTempText($"Yes! Instead of creating an auxillary list,
```

```
 we can instead reverse this list in place.");
  168:         yield return WaitForNextStep();
  169:         worldspaceTempText.setTempText("To do so, we'll need to keep track of three
nodes. Our current, previous, and next nodes.");
  170:         yield return WaitForNextStep();
  171:         worldspaceTempText.setTempText("The basic process is as follows:");
  172:         yield return WaitForNextStep();
  173:         worldspaceTempText.setTempText("1. Make the current node point to the previo
us node\n2. Update previous to current, current to next, and next to the node next is point
ing to.\n3. Repeat until the end of the list.");
  174:         //worldspaceTempText.transform.position = Camera.main.transform.position + n
ew Vector3(0, 0, distDepth);
  175:         yield return WaitForNextStep();
  176:         worldspaceTempText.setTempText($"Now, let's watch this linked list reversal
play out.");
  177:         yield return WaitForNextStep();
  178:         LinkedListStructure ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.
LinkedList, 0).GetComponent<LinkedListStructure>();
  179:         ll1.transform.position = Camera.main.transform.position + new Vector3(-5, 0,
 distDepth);
  180:         yield return algorithms.RunAlgorithm(UIManager.NodeType.LinkedList, 0, ll1,
-1);
  181:         Destroy(ll1.gameObject);
  182:     }
  183:
  184:     public IEnumerator LLReversalExplanation()
  185:     {
  186:         worldspaceTempText.setTempText("This offers us a useful opportunity to learn
 about complexity.");
  187:         yield return WaitForNextStep();
  188:         worldspaceTempText.setTempText("These examples have the same time complexity
 - they both need to traverse the list just once to complete the operation.");
  189:         yield return WaitForNextStep();
  190:         worldspaceTempText.setTempText("However, where these algorithms differ is th
eir space complexity.");
  191:         yield return WaitForNextStep();
  192:         worldspaceTempText.setTempText("As we learned, the naive approach requires a
 whole extra copy of the list, whereas the smart approach does not need an extra list.");
  193:         yield return WaitForNextStep();
  194:         worldspaceTempText.setTempText("Using Big-O notation, the spatial complexity
 of the naive approach is O(n), whereas the spatial complexity of the smart approach is O(1
). The time complexity of both is O(n).");
  195:         yield return WaitForNextStep();
  196:
  197:     }
  198:
  199:     public IEnumerator QuizStartStep()
  200:     {
  201:         worldspaceTempText.setTempText("Now we're going to start the quiz! Let's sta
rt with a demo quiz question.");
  202:         yield return WaitForNextStep();
  203:     }
  204:
  205:     public IEnumerator DemoQuizStep()
  206:     {
  207:         worldspaceTempText.setTempText("");
  208:
  209:         ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.LinkedList, 2).GetCo
mponent<LinkedListStructure>();
  210:         ll1.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.left * 5 + Vector3.down*2;
  211:
  212:         Quiz quiz = Quiz.CreateQuiz("How many nodes does this linked list have?", ne
w string[] { "6 Nodes", "7 Nodes", "8 Nodes" }, 1);
  213:         quiz.transform.position = ll1.transform.position + Vector3.right * 6 + Vecto
r3.up * 2.5f;
  214:         quiz.qIdentifier = "SAMPLE";
  215:         yield return quiz.WaitForQuiz();
```

```
  216:         Destroy(quiz.gameObject);
  217:         Destroy(ll1.gameObject);
  218:     }
  219:
  220:     public IEnumerator QuizAlertStep()
  221:     {
  222:         worldspaceTempText.tempText.transform.position = Camera.main.transform.posit
ion + baseOffset;
  223:         worldspaceTempText.setTempText("Ok, now we'll start the actual quiz. Here we
 go!");
  224:         yield return WaitForNextStep();
  225:         worldspaceTempText.setTempText("");
  226:     }
  227:
  228:     public IEnumerator QuizQ1()
  229:     {
  230:         ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.LinkedList, 0).GetCo
mponent<LinkedListStructure>();
  231:         ll1.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.left * 5 + Vector3.down*2;
  232:         Quiz quiz = Quiz.CreateQuiz("How many operations would the naive approach re
quire to reverse this List?", new string[] { "0 Operations", "7 Operations", "49 Operations
" }, 1);
  233:         quiz.transform.position = ll1.transform.position + Vector3.right * 6 + Vecto
r3.up * 2.5f;
  234:         quiz.qIdentifier = "LL 1";
  235:         yield return quiz.WaitForQuiz();
  236:         yield return new WaitForSeconds(1);
  237:         Destroy(quiz.gameObject);
  238:         Destroy(ll1.gameObject);
  239:     }
  240:
  241:     public IEnumerator QuizQ2()
  242:     {
  243:         ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.LinkedList, 0).GetCo
mponent<LinkedListStructure>();
  244:         ll1.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.left * 5 + Vector3.down*2;
  245:         Quiz quiz = Quiz.CreateQuiz("How many operations would the smart approach re
quire to reverse this List?", new string[] { "0 Operations", "7 Operations", "49 Operations
" }, 1);
  246:         quiz.transform.position = ll1.transform.position + Vector3.right * 6 + Vecto
r3.up * 2.5f;
  247:         quiz.qIdentifier = "LL 2";
  248:         yield return quiz.WaitForQuiz();
  249:         yield return new WaitForSeconds(1);
  250:         Destroy(quiz.gameObject);
  251:         Destroy(ll1.gameObject);
  252:     }
  253:
  254:     public IEnumerator QuizQ3()
  255:     {
  256:         ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.LinkedList, 0).GetCo
mponent<LinkedListStructure>();
  257:         ll1.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.left * 5 + Vector3.down*2;
  258:         Quiz quiz = Quiz.CreateQuiz("How much extra space would the naive approach r
equire to reverse this List?", new string[] { "0 Nodes", "7 Nodes", "49 Nodes" }, 1);
  259:         quiz.transform.position = ll1.transform.position + Vector3.right * 6 + Vecto
r3.up * 2.5f;
  260:         quiz.qIdentifier = "LL 3";
  261:         yield return quiz.WaitForQuiz();
  262:         yield return new WaitForSeconds(1);
  263:         Destroy(quiz.gameObject);
  264:         Destroy(ll1.gameObject);
  265:     }
  266:
  267:     public IEnumerator QuizQ4()
```

```
  268:     {
  269:         ll1 = DemoStructures.CreateDemo(DemoStructures.NodeType.LinkedList, 0).GetCo
mponent<LinkedListStructure>();
  270:         ll1.transform.position = Camera.main.transform.position + baseOffset + Vecto
r3.left * 5 + Vector3.down*2;
  271:         Quiz quiz = Quiz.CreateQuiz("How much extra space would the smart approach r
equire to reverse this List?", new string[] { "0 Nodes", "7 Nodes", "49 Nodes" }, 0);
  272:         quiz.transform.position = ll1.transform.position + Vector3.right * 6 + Vecto
r3.up * 2.5f;
  273:         quiz.qIdentifier = "LL 4";
  274:         yield return quiz.WaitForQuiz();
  275:         yield return new WaitForSeconds(1);
  276:         Destroy(quiz.gameObject);
  277:         Destroy(ll1.gameObject);
  278:     }
  279: }
```

```
    1: ï»¿using System;
    2: using System.Collections.Generic;
    3: using System.Linq;
    4: using System.Text;
    5: using System.Threading.Tasks;
    6: using Unity;
    7: using UnityEngine;
    8: using TMPro;
    9: using System.Collections;
   10: using System.IO;
   11:
   12: class Quiz : MonoBehaviour
   13: {
   14:     public const float OPTION_SPACING = 1.5f;
   15:
   16:     //Members
   17:     public string qIdentifier = String.Empty;
   18:     public bool resolved = false;
   19:     public bool iResolved = false;
   20:     public bool? correct = null;
   21:     public GameObject resolutionObj;
   22:     public TextMeshProUGUI resolutionText;
   23:
   24:     public IEnumerator Resolve()
   25:     {
   26:         iResolved = true;
   27:
   28:         yield return new WaitForEndOfFrame();
   29:         //yield return new WaitForEndOfFrame();
   30:         //yield return new WaitForEndOfFrame();
   31:
   32:         //Write out to resolution file
   33:         if (!qIdentifier.Contains("SAMPLE") && PlayerPrefs.HasKey("study_id"))
   34:         {
   35:             string id = PlayerPrefs.GetString("study_id");
   36:             string path = Path.Combine(Application.persistentDataPath, $"{id.ToStrin
g()}-results.txt");
   37:             string qRes;
   38:             if (correct == true)
   39:                 qRes = "correct";
   40:             else
   41:                 qRes = "incorrect";
   42:
   43:             File.AppendAllText(path, $"{qIdentifier}:{qRes},\n");
   44:         }
   45:
   46:
   47:         resolutionObj.SetActive(true);
   48:         if(correct == true)
   49:             resolutionObj.GetComponentInChildren<TextMeshProUGUI>().text = "Yes, tha
t's correct!";
   50:         else if(correct == false)
   51:             resolutionObj.GetComponentInChildren<TextMeshProUGUI>().text = "Sorry, t
hat's incorrect";
   52:         else
   53:             resolutionObj.GetComponentInChildren<TextMeshProUGUI>().text = "Quiz que
stion incorrectly resolved...";
   54:
   55:         yield return new WaitForSeconds(1);
   56:         resolved = true;
   57:     }
   58:
   59:     public IEnumerator WaitForQuiz()
   60:     {
   61:         while (resolved == false)
   62:             yield return null;
   63:         yield return new WaitForEndOfFrame();
   64:     }
```

```
    public IEnumerator WaitForQuizImmediate()
    {
        while (iResolved == false)
            yield return null;
        yield return new WaitForEndOfFrame();
    }

    public static Quiz CreateQuiz(string prompt, string[] options, int correctIndex)
    {
        GameObject quizHolder = new GameObject();
        quizHolder.name = "Quiz Manager";
        GameObject uiPrefab = Resources.Load("Other/Worldspace Canvas") as GameObject;
        Quiz quiz = quizHolder.AddComponent<Quiz>();


        //Create resolution text first
        GameObject resolutionObj = Instantiate(uiPrefab);
        resolutionObj.transform.SetParent(quiz.transform);
        resolutionObj.transform.position = quiz.transform.position;
        resolutionObj.SetActive(false);
        quiz.resolutionObj = resolutionObj;

        Vector3 pos = quizHolder.transform.position;
        //Center the quiz objects around how many questions there are
        pos += Vector3.up * OPTION_SPACING * ((options.Length - 1.0f) / 2.0f);

        if(!string.IsNullOrEmpty(prompt))
        {
            GameObject promptObj = Instantiate(uiPrefab);
            promptObj.transform.SetParent(quiz.transform);
            promptObj.transform.position = pos + Vector3.up * OPTION_SPACING;
            promptObj.GetComponentInChildren<TextMeshProUGUI>().text = prompt;
        }

        List<GameObject> quizOptionObjects = new List<GameObject>();

        for (int i = 0; i < options.Length; i++)
        {
            string option = options[i];

            //Create this option
            GameObject optionObj = Instantiate(uiPrefab);
            optionObj.transform.SetParent(quiz.transform);
            optionObj.transform.position = pos;
            pos -= Vector3.up * OPTION_SPACING;

            //Assign text
            optionObj.GetComponentInChildren<TextMeshProUGUI>().text = option;

            //Add selection option
            UISelect optionUIS = optionObj.transform.GetChild(0).gameObject.AddComponent<UISelect>();
            optionUIS.Select = new UnityEngine.Events.UnityEvent();

            if (i == correctIndex)
            {

                optionUIS.Select.AddListener(() =>
                {
                    quizOptionObjects.ForEach(x => Destroy(x));
                    quiz.correct = true;
                    quiz.StartCoroutine(quiz.Resolve());
                });
            }
            else
            {
```

```
  131:                 optionUIS.Select.AddListener(() =>
  132:                 {
  133:                     quizOptionObjects.ForEach(x => Destroy(x));
  134:                     quiz.correct = false;
  135:                     quiz.StartCoroutine(quiz.Resolve());
  136:                 });
  137:             }
  138:             //Debug.Log($"i {i}: {((i == correctIndex) ? "Yes, that's correct!" : "S
orry, that's incorrect")}");
  139:
  140:             quizOptionObjects.Add(optionObj);
  141:         }
  142:
  143:         return quiz;
  144:     }
  145:
  146:     public static Quiz CreateQuiz(string[] options, int correctIndex)
  147:     {
  148:         GameObject quizHolder = new GameObject();
  149:         quizHolder.name = "Quiz Manager";
  150:         GameObject uiPrefab = Resources.Load("Other/Worldspace Canvas") as GameObjec
t;
  151:         Quiz quiz = quizHolder.AddComponent<Quiz>();
  152:
  153:
  154:         //Create resolution text first
  155:         GameObject resolutionObj = Instantiate(uiPrefab);
  156:         resolutionObj.transform.SetParent(quiz.transform);
  157:         resolutionObj.transform.position = quiz.transform.position;
  158:         resolutionObj.SetActive(false);
  159:         quiz.resolutionObj = resolutionObj;
  160:
  161:         Vector3 pos = quizHolder.transform.position;
  162:         //Center the quiz objects around how many questions there are
  163:         pos += Vector3.up * OPTION_SPACING * ((options.Length-1.0f) / 2.0f);
  164:
  165:
  166:         List<GameObject> quizOptionObjects = new List<GameObject>();
  167:
  168:         for(int i = 0; i < options.Length; i++)
  169:         {
  170:             string option = options[i];
  171:
  172:             //Create this option
  173:             GameObject optionObj = Instantiate(uiPrefab);
  174:             optionObj.transform.SetParent(quiz.transform);
  175:             optionObj.transform.position = pos;
  176:             pos -= Vector3.up * OPTION_SPACING;
  177:
  178:             //Assign text
  179:             optionObj.GetComponentInChildren<TextMeshProUGUI>().text = option;
  180:
  181:             //Add selection option
  182:             UISelect optionUIS = optionObj.transform.GetChild(0).gameObject.AddCompo
nent<UISelect>();
  183:             optionUIS.Select = new UnityEngine.Events.UnityEvent();
  184:
  185:             if (i == correctIndex)
  186:             {
  187:
  188:                 optionUIS.Select.AddListener(() =>
  189:                 {
  190:                     quizOptionObjects.ForEach(x => Destroy(x));
  191:                     quiz.correct = true;
  192:                     quiz.StartCoroutine(quiz.Resolve());
  193:                 });
  194:             }
  195:             else
```

```
  196:             {
  197:                 optionUIS.Select.AddListener(() =>
  198:                 {
  199:                     quizOptionObjects.ForEach(x => Destroy(x));
  200:                     quiz.correct = false;
  201:                     quiz.StartCoroutine(quiz.Resolve());
  202:                 });
  203:             }
  204:             //Debug.Log($"i {i}: {((i == correctIndex) ? "Yes, that's correct!" : "S
orry, that's incorrect")}");
  205:
  206:             quizOptionObjects.Add(optionObj);
  207:         }
  208:
  209:         return quiz;
  210:     }
  211:
  212:     public delegate IEnumerator CheckMethod(Quiz q);
  213:
  214:     public static Quiz CreateQuiz(CheckMethod resolutionCheck)
  215:     {
  216:         GameObject quizHolder = new GameObject();
  217:         quizHolder.name = "Quiz Manager";
  218:         GameObject uiPrefab = Resources.Load("Other/Worldspace Canvas") as GameObjec
t;
  219:         Quiz quiz = quizHolder.AddComponent<Quiz>();
  220:
  221:         //Create resolution text first
  222:         GameObject resolutionObj = Instantiate(uiPrefab);
  223:         resolutionObj.transform.SetParent(quiz.transform);
  224:         resolutionObj.transform.position = quiz.transform.position;
  225:         resolutionObj.SetActive(false);
  226:         quiz.resolutionObj = resolutionObj;
  227:
  228:         GameObject optionObj = Instantiate(uiPrefab);
  229:
  230:         //Assign text
  231:         optionObj.GetComponentInChildren<TextMeshProUGUI>().text = "Submit";
  232:
  233:         IEnumerator ResolutionCheck()
  234:         {
  235:             yield return resolutionCheck(quiz);
  236:             quiz.StartCoroutine(quiz.Resolve());
  237:         }
  238:
  239:         //Add selection option
  240:         UISelect optionUIS = optionObj.transform.GetChild(0).gameObject.AddComponent
<UISelect>();
  241:         optionObj.transform.SetParent(quiz.transform);
  242:         optionObj.transform.position = quiz.transform.position;
  243:
  244:         optionUIS.Select = new UnityEngine.Events.UnityEvent();
  245:         optionUIS.Select.AddListener(() =>
  246:         {
  247:             quiz.StartCoroutine(ResolutionCheck());
  248:         });
  249:
  250:         return quiz;
  251:     }
  252: }
```

```
    1: using System;
    2: using System.Collections;
    3: using System.Collections.Generic;
    4: using System.Linq;
    5: using UnityEngine;
    6: using UnityEngine.SceneManagement;
    7: using TMPro;
    8: using System.IO;
    9:
   10: public class Tutorial : MonoBehaviour
   11: {
   12:     //Static references to inhibit actions/input
   13:     public static bool restrictInfo = false;
   14:     public static bool restrictPickupInfo = false;
   15:     public static bool restrictMovement = false;
   16:     public static bool restrictCSToggle = false;
   17:     public static bool restrictPanelChange = false;
   18:
   19:     //Depending on the platform, the tutorial will be different. Different text need
s to be swapped in and out
   20:     protected string settingsButton, pickupButton, howToMoveNodes, connectionButton,
 flipAction, deleteButton, selectionButton, changePanelButton, organizationButton, uiIntera
ctionDescription, nextStepButton;
   21:
   22:     //Component References
   23:     protected Actions actions;
   24:     protected TempText tempText;
   25:     protected Algorithms algorithms;
   26:     public TempText worldspaceTempText;
   27:
   28:     //Tutorial Steps
   29:     protected delegate IEnumerator TutorialStep();
   30:     protected List<TutorialStep> tutorialSteps;
   31:
   32:     //Public references
   33:     public TMP_Text vrTutorialText;
   34:
   35:     public Vector3 baseOffset = Vector3.zero;
   36:     public float distDepth = 6;
   37:
   38:     public void Start()
   39:     {
   40:         //If there is an instance of the Tutorial class, these will become true.
   41:         restrictInfo = true;
   42:         restrictPickupInfo = true;
   43:         restrictMovement = true;
   44:         restrictCSToggle = true;
   45:         restrictPanelChange = true;
   46:
   47:
   48:         actions = GetComponent<Actions>();
   49:         algorithms = GetComponent<Algorithms>();
   50:
   51:         SetTutorialReferences();
   52:
   53:         //Only one will be active, so the correct TempText will be assigned here.
   54:         tempText = GetComponentInChildren<TempText>();
   55:
   56:         InitTutorialSteps();
   57:         RunTutorial();
   58:     }
   59:
   60:     public virtual void RunTutorial()
   61:     {
   62:         //Start the tutorial
   63:         Debug.Log($"Starting tutorial.");
   64:
   65:         StartCoroutine(StartTutorial());
```

```
    }

    private void Update()
    {
        //Quick performance measure without the profiler.
        //Debug.Log($"t {Time.deltaTime}");
    }

    public virtual void SetTutorialReferences()
    {
        //Conditional initialization
        if (actions.vr)
        {
            baseOffset = new Vector3(0, 0, 9);
            distDepth = 9;

            settingsButton = "the Menu button on your left Touch controller";
            pickupButton = "left grip button";
            howToMoveNodes = "pressing in the left thumbstick, and moving it up or d
own";
            connectionButton = "Y";
            flipAction = "pressing the left trigger while pointing at a connection";
            deleteButton = "X";
            selectionButton = "the left trigger";
            changePanelButton = "the right thumbstick";
            organizationButton = "A";
            uiInteractionDescription = "press the left trigger while pointing at the
m. If you are pointing at an interactable object, a line will be drawn between it and your
hand to let you know.";
            nextStepButton = "the left trigger";
        }
        else
        {
            baseOffset = new Vector3(0, 0, 7);
            distDepth = 6;

            settingsButton = "E";
            pickupButton = "middle mouse button";
            howToMoveNodes = " scrolling with the mouse wheel";
            connectionButton = "the right mouse button";
            flipAction = "pressing the left mouse button while pointing at a connect
ion";
            deleteButton = "F";
            selectionButton = "the left mouse button";
            changePanelButton = "tab";
            organizationButton = "Q";
            uiInteractionDescription = "click the left mouse button while looking at
 them. If you are looking at an interactable UI element, your crosshair will turn green to
let you know.";
            nextStepButton = "the spacebar";
        }
    }

    private IEnumerator StartTutorial()
    {
        while(tutorialSteps.Count > 0)
        {
            Debug.Log($"There are {tutorialSteps.Count} steps remaining in the tutor
ial.");
            yield return StartCoroutine(tutorialSteps[0]());
            tutorialSteps.RemoveAt(0);
        }
        Debug.Log($"The tutorial has ended.");
    }

    public virtual void InitTutorialSteps()
    {
        TutorialStep[] steps = new TutorialStep[] { StartStep, IntroductionStep, Int
```

```
eractionStep, GUIDStep, FinishStep };
  127:         tutorialSteps = new List<TutorialStep>(steps);
  128:
  129:
  130:     }
  131:
  132:     #region OriginalTutorialSteps
  133:
  134:     private IEnumerator IntroductoryStep()
  135:     {
  136:         actions.SetCSEnabled(false);
  137:
  138:         yield return StartCoroutine(WaitForNextStep());
  139:
  140:         tempText.setTempText($"Welcome to the Data Structures Visualization tutorial
. Press {nextStepButton} to continue.");
  141:         yield return StartCoroutine(WaitForNextStep());
  142:
  143:         //if in vr
  144:         if(actions.vr)
  145:         {
  146:             tempText.setTempText($"For now, I'll be attached to your left hand, as t
hat is where info will be displayed while in-game. In a bit, I'll move to being in front of
 you. Press {nextStepButton} to continue.");
  147:             yield return StartCoroutine(WaitForNextStep());
  148:         }
  149:
  150:         tempText.setTempText($"This game functions more as a tool than a game, allow
ing you to visualize algorithms and data structures in the world in front of you. Press {ne
xtStepButton} to continue.");
  151:         yield return StartCoroutine(WaitForNextStep());
  152:
  153:         //tempText.setTempText($"During this tutorial, none of your actions will be
restricted. Feel free to discover the capabilities of the tool if you wish. " +
  154:        //     $"You may, at any time press {settingsButton} to access the settings a
nd leave the tutorial via the button. If you want a more guided introduction, continue with
 the tutorial.");
  155:         //yield return StartCoroutine(WaitForAny());
  156:     }
  157:
  158:
  159:     private IEnumerator InformationStep()
  160:     {
  161:         if(actions.vr)
  162:         {
  163:             //tempText.setTempText("After this step, I'll move to being in front of
you at something you'll learn about next.");
  164:             yield return StartCoroutine(WaitForNextStep());
  165:             tempText.setTempText("During the tutorial or during use at any time, you
 may touch the left thumbstick to enable information display. If this is on, information ab
out what you're pointing at will be displayed at your left hand, where I am now.");
  166:             yield return StartCoroutine(WaitForNextStep());
  167:             tempText.setTempText("This will provide general information about UI Opt
ions, Data structures, Algorithms, and the controls of the tool.");
  168:         }
  169:         else
  170:         {
  171:             tempText.setTempText("During the tutorial, the text I am currently occup
ying will be reserved for the contents of the tutorial. After that, you can press tilde to
toggle display of information.");
  172:             yield return StartCoroutine(WaitForNextStep());
  173:             tempText.setTempText("If this is on, information about what you're point
ing at will be displayed on-screen. This will provide general information about UI Options,
 Data structures, Algorithms, and the controls of the tool.");
  174:             yield return StartCoroutine(WaitForNextStep());
  175:         }
  176:     }
  177:
```

```
178:     private IEnumerator CSAndCreationStep()
179:     {
180:         actions.SetCSEnabled(true);
181:
182:         //assign temp text on CS for VR.
183:         if (actions.vr)
184:         {
185:             //clear left hand text, allow left-hand text to work independently
186:             TempText onCSText = gameObject.AddComponent<TempText>();
187:             onCSText.assignTempText(vrTutorialText);
188:             tempText.setTempText(string.Empty);
189:             tempText = onCSText;
190:             restrictInfo = false;
191:         }
192:
193:         tempText.setTempText($"What just appeared in front of you is called the Crea
tion Station. It is used for creating nodes & data structures, as well as running algorithm
s. To interact with these buttons, you can {uiInteractionDescription}");
194:         yield return StartCoroutine(WaitForNextStep());
195:         if(actions.vr)
196:         {
197:             tempText.setTempText("I'll be occupying this space from now on, so feel
free to touch the left thumbstick to show information about what you're pointing at.");
198:             yield return StartCoroutine(WaitForNextStep());
199:         }
200:
201:         //Add a note about toggling / moving the creation station.
202:
203:         tempText.setTempText($"The panel you're looking at currently is the Creation
 panel. On the left, it shows the type of node, the value of the node, and a preview of the
 node, respectively.");
204:         yield return StartCoroutine(WaitForNextStep());
205:         tempText.setTempText($"On the right is the create button! You can create a B
inary Tree node, a Linked List node, or a node for a Directed or Undirected graph.");
206:         yield return StartCoroutine(WaitForNextStep());
207:         tempText.setTempText($"Try creating one of each type of node.");
208:         yield return StartCoroutine(WaitForAllNodesToExist());
209:         tempText.setTempText($"Nice job! Next, we'll learn about what you can do wit
h these nodes. I'll get these out of your way for now.");
210:         foreach (Node node in FindObjectsOfType<Node>())
211:             Destroy(node.gameObject);
212:
213:     }
214:
215:     private IEnumerator PickupStep()
216:     {
217:         tempText.setTempText("Now that we've created some nodes, we won't want them
to all stack up in one place.");
218:         yield return StartCoroutine(WaitForNextStep());
219:         tempText.setTempText($"You can pickup nodes by pressing the {pickupButton}.
This will allow you to move them around. To drop them, you can press the button again.");
220:         yield return StartCoroutine(WaitForNextStep());
221:         tempText.setTempText($"Additionally, you can move nodes closer to or farther
 from you by {howToMoveNodes}.");
222:         yield return StartCoroutine(WaitForNextStep());
223:         tempText.setTempText($"Now, try creating and picking up some nodes. Move the
m apart a bit.");
224:         //During this part of the tutorial, the on-screen text should be allowed to
be overwritten.
225:         restrictPickupInfo = false;
226:         yield return StartCoroutine(WaitForTwoNodesToExist());
227:         Debug.Log($"Two nodes now exist.");
228:         yield return StartCoroutine(WaitForTwoNodesToBeFarApart());
229:         restrictPickupInfo = true;
230:         tempText.setTempText($"Nice job! Next, we'll learn how to connect these node
s.");
231:         yield return StartCoroutine(WaitForNextStep());
232:     }
```

```
  233:
  234:     private IEnumerator ConnectionStep()
  235:     {
  236:         tempText.setTempText("With our properly-spaced nodes, we can start connectin
g them. Conceptually, these connections represent the references that nodes hold to other n
odes. Like the \"next\" pointer in a Linked List, or the \"left\" and \"right\" pointers in
 a Binary Tree.");
  237:         yield return StartCoroutine(WaitForNextStep());
  238:         tempText.setTempText($"By pressing {connectionButton}, you can connect two n
odes. Nodes can only be connected if they are the same type! ");
  239:         yield return StartCoroutine(WaitForNextStep());
  240:         tempText.setTempText($"When you connect two Linked List or Binary Tree nodes
, existing connections may be overwritten, because they may only have a limited number of c
onnections facing in or out.");
  241:         yield return StartCoroutine(WaitForNextStep());
  242:         tempText.setTempText($"You may also flip the connection between two nodes, b
y {flipAction}, but this may also overwrite some connections.");
  243:         yield return StartCoroutine(WaitForNextStep());
  244:         tempText.setTempText($"Now, try connecting some nodes!");
  245:         //During this part of the tutorial, the on-screen text should be allowed to
be overwritten.
  246:         restrictPickupInfo = false;
  247:         yield return StartCoroutine(WaitForConnections());
  248:         restrictPickupInfo = true;
  249:         tempText.setTempText($"Nice job. Next, we'll learn about selecting nodes.");
  250:         yield return StartCoroutine(WaitForAny());
  251:     }
  252:
  253:     private IEnumerator SelectionAndDeletionStep()
  254:     {
  255:         tempText.setTempText($"When you press {selectionButton} while pointing at a
node, it will select it. Selecting a node has a few uses, some related to things we haven't
 yet learned.");
  256:         yield return StartCoroutine(WaitForNextStep());
  257:         tempText.setTempText($"When you pick up a selected node, it will move the en
tire structure it is a part of with it. You'll learn more about this when we cover structur
es.");
  258:         yield return StartCoroutine(WaitForNextStep());
  259:         tempText.setTempText($"Additionally, to run any algorithm, we must first sel
ect a node to run that algorithm on.");
  260:         yield return StartCoroutine(WaitForNextStep());
  261:         tempText.setTempText($"You may also want to remove nodes or links from your
structures. To do so, you can press {deleteButton} while pointing at a node or link, and it
 will delete it");
  262:         yield return StartCoroutine(WaitForNextStep());
  263:         tempText.setTempText($"Now, try selecting a node, and then deleting it.");
  264:         yield return StartCoroutine(WaitForSelection());
  265:         yield return StartCoroutine(WaitForDeletion());
  266:         tempText.setTempText($"Nice job.");
  267:         yield return StartCoroutine(WaitForNextStep());
  268:     }
  269:
  270:     private IEnumerator MovementStep()
  271:     {
  272:         restrictMovement = false;
  273:         tempText.setTempText("Now we will learn to move!");
  274:         yield return StartCoroutine(WaitForNextStep());
  275:         if (actions.vr)
  276:         {
  277:             tempText.setTempText("To avoid motion sickness, snap turning is enabled.
 Try using the right thumbstick to rotate. Once you've rotated to your heart's content, pre
ss the left trigger to advance the tutorial.");
  278:             yield return StartCoroutine(WaitForOVRInput(OVRInput.Button.PrimaryIndex
Trigger));
  279:             tempText.setTempText("Additionally, movement via teleportation is enable
d. If you want, you can use the right hand grip button to preview your teleporting location
, and the right trigger button to teleport. Try that now.");
  280:             yield return StartCoroutine(WaitForOVRInput(OVRInput.Button.SecondaryHan
```

```
dTrigger));
  281:             yield return StartCoroutine(WaitForOVRInput(OVRInput.Button.SecondaryInd
exTrigger));
  282:             tempText.setTempText("Excellent! Continuous motion is available via the
left thumbstick, but only use that if you are accustomed to motion in VR! Don't get motion
sick while you're learning.");
  283:             //yield return StartCoroutine(WaitSeconds(3));
  284:             yield return StartCoroutine(WaitForNextStep());
  285:             tempText.setTempText($"Once you are comfortable moving, press the left t
rigger to advance the tutorial.");
  286:             yield return StartCoroutine(WaitForOVRInput(OVRInput.Button.PrimaryIndex
Trigger));
  287:         }
  288:         else
  289:         {
  290:             tempText.setTempText("On keyboard-and-mouse, motion is standard to a 3D
First Person Character - WASD to move, and the mouse to look around.");
  291:             //yield return StartCoroutine(WaitSeconds(3));
  292:             yield return StartCoroutine(WaitForAny());
  293:             tempText.setTempText("Try that now.");
  294:             yield return StartCoroutine(WaitForKeyDown(KeyCode.W));
  295:             yield return StartCoroutine(WaitForKeyDown(KeyCode.A));
  296:             yield return StartCoroutine(WaitForKeyDown(KeyCode.S));
  297:             yield return StartCoroutine(WaitForKeyDown(KeyCode.D));
  298:             tempText.setTempText($"Once you are comfortable moving, press anything t
o advance the tutorial.");
  299:             yield return StartCoroutine(WaitForAny());
  300:         }
  301:     }
  302:
  303:     private IEnumerator OrganizationAndDemoStructuresStep()
  304:     {
  305:         tempText.setTempText($"Now that we've learned to create, move, and connect n
odes, we should learn how to organize them into Structures.");
  306:         yield return StartCoroutine(WaitForNextStep());
  307:         tempText.setTempText($"By pressing {organizationButton} with a node selected
, it will organize that node into a structure, placing its nodes evenly and creating a Stru
cture that algorithms can be run on.");
  308:         yield return StartCoroutine(WaitForNextStep());
  309:         //tempText.setTempText($"For Binary Trees and Linked Lists, they will be org
anized in the traditional visual fashion - vertical and left-to-right organization respecti
vely. Graphs will be organized in a way that the nodes and links should be appropriately pl
aced.");
  310:         //yield return StartCoroutine(WaitForAny());
  311:         tempText.setTempText($"Try organizing some nodes into a structure. Try creat
ing a binary tree and a linked list structure.");
  312:         //During this part of the tutorial, the on-screen text should be allowed to
be overwritten.
  313:         restrictPickupInfo = false;
  314:         yield return StartCoroutine(WaitForBinaryTreeAndLinkedList());
  315:         restrictPickupInfo = true;
  316:         tempText.setTempText($"Nice! In case the structure-creation-process seems to
o laborious, there are also some structures you can easily load into the workspace.");
  317:         yield return StartCoroutine(WaitForNextStep());
  318:         tempText.setTempText($"Now would be a good time to look at the other panels
of the Creation Station. By pressing the {changePanelButton}, you can change which panel yo
u're using. The Structure panel is next after the Creation panel, so try getting to that no
w.");
  319:         yield return StartCoroutine(WaitForNextStep());
  320:         tempText.setTempText($"Once you've done that, you can add structures of all
types by changing which type is selected, much like you would on the Creation panel.");
  321:         yield return StartCoroutine(WaitForNextStep());
  322:         tempText.setTempText($"Now, try adding some of these sample structures into
the workspace. Remember that if you select a node and then pick it up, it will move the ent
ire structure.");
  323:         yield return StartCoroutine(WaitForDemoStructures());
  324:         tempText.setTempText($"Excellent! Lastly, we'll cover running algorithms. Af
ter that, you'll be fully educated on the capabilities of the tool!");
```

```
  325:         yield return StartCoroutine(WaitForNextStep());
  326:     }
  327:
  328:     private IEnumerator AlgorithmStep()
  329:     {
  330:         tempText.setTempText($"Now we finally have everything we need to run some al
gorithms. Let’s start by selecting a node that is part of a structure - either one you crea
ted yourself, or one that is from the sample structures.");
  331:         yield return StartCoroutine(WaitForSelection());
  332:         tempText.setTempText($"Next, switch to the algorithms panel on the Creation
Station. It will display what algorithms are available for that type of node.");
  333:         yield return StartCoroutine(WaitForNextStep());
  334:         tempText.setTempText($"On the algorithms panel, the three options on the sid
e show the type of node you have selected, the Iteration mode, and the value parameter. " +
  335:             $"The iteration mode can be Input, where the algorithm will step forward
 on button press, Wait, where the algorithm will wait one second between steps, and Instant
aneous, where the algorithm will complete instantly." +
  336:             $"Some algorithms need a parameter (shifting linked list, searching), so
 the value parameter will decide what it uses.");
  337:         yield return StartCoroutine(WaitForNextStep());
  338:         tempText.setTempText($"Now, try running an algorithm, and watch how it runs!
 It will highlight the nodes that are being used, and show output text in the workspace.");
  339:         yield return StartCoroutine(WaitForAlgoRun());
  340:         tempText.setTempText($"Nice. Now wait for this algorithm to finish iterating
!");
  341:         yield return StartCoroutine(WaitForAlgoEnd());
  342:         tempText.setTempText($"Excellent! You are now fully aware of the capabilitie
s of the tool. I’ll load you into a fresh workspace now.");
  343:         yield return StartCoroutine(WaitForNextStep());
  344:         //let user know about scene change
  345:         yield return WaitSeconds(3);
  346:         SceneManager.LoadScene("Main Scene");
  347:     }
  348:
  349:     #endregion OriginalTutorialSteps
  350:
  351:     #region TutorialEnumerators
  352:     //Other coroutines for moving forward:
  353:     public IEnumerator WaitForAny()
  354:     {
  355:         //Debug.Log($"Waiting any with {actions.vr} vr");
  356:         if(actions.vr)
  357:         {
  358:             while (!OVRInput.GetDown(OVRInput.Button.Any))
  359:                 yield return new WaitForEndOfFrame();
  360:             //three-frame buffer for OVRInput
  361:             yield return new WaitForEndOfFrame();
  362:             yield return new WaitForEndOfFrame();
  363:             yield return new WaitForEndOfFrame();
  364:         }
  365:         else
  366:         {
  367:             //Any key down not working?
  368:             while (!Input.anyKeyDown)
  369:                 yield return null;
  370:             yield return new WaitForEndOfFrame();
  371:             yield return new WaitForEndOfFrame();
  372:             yield return new WaitForEndOfFrame();
  373:         }
  374:     }
  375:
  376:     public IEnumerator WaitForAttentive()
  377:     {
  378:         if (actions.vr)
  379:         {
  380:             while (!(OVRInput.Get(OVRInput.Axis1D.SecondaryIndexTrigger) > 0.95f &&
OVRInput.Get(OVRInput.Axis1D.PrimaryIndexTrigger) > 0.95f))
  381:                 yield return new WaitForEndOfFrame();
```

```
  382:             yield return new WaitForEndOfFrame();
  383:             yield return new WaitForEndOfFrame();
  384:             yield return new WaitForEndOfFrame();
  385:         }
  386:         else
  387:         {
  388:             while (!Input.GetKey(KeyCode.Space) || !Input.GetKey(KeyCode.LeftControl
))
  389:                 yield return null;
  390:             yield return new WaitForEndOfFrame();
  391:             yield return new WaitForEndOfFrame();
  392:             yield return new WaitForEndOfFrame();
  393:         }
  394:     }
  395:
  396:     public IEnumerator WaitForNextStep()
  397:     {
  398:         if (actions.vr)
  399:         {
  400:             while (!OVRInput.GetDown(OVRInput.Button.One) && !OVRInput.GetDown(OVRIn
put.Button.Two) && !OVRInput.GetDown(OVRInput.Button.Three) && !OVRInput.GetDown(OVRInput.B
utton.Four))
  401:                 yield return new WaitForEndOfFrame();
  402:             yield return new WaitForEndOfFrame();
  403:             yield return new WaitForEndOfFrame();
  404:             yield return new WaitForEndOfFrame();
  405:         }
  406:         else
  407:         {
  408:             while (!Input.GetKeyDown(KeyCode.Space))
  409:                 yield return null;
  410:             yield return new WaitForEndOfFrame();
  411:             yield return new WaitForEndOfFrame();
  412:             yield return new WaitForEndOfFrame();
  413:         }
  414:     }
  415:
  416:     public IEnumerator WaitForAlgoRun()
  417:     {
  418:         while (!Algorithms.inAlgo)
  419:             yield return new WaitForEndOfFrame();
  420:     }
  421:
  422:     public IEnumerator WaitForAlgoEnd()
  423:     {
  424:         while (Algorithms.inAlgo)
  425:             yield return new WaitForEndOfFrame();
  426:     }
  427:
  428:     public IEnumerator WaitForDemoStructures()
  429:     {
  430:         do
  431:         {
  432:             yield return new WaitForEndOfFrame();
  433:         } while (FindObjectsOfType<DemoStructureMarker>().Length < 3);
  434:     }
  435:
  436:     public IEnumerator WaitForBinaryTreeAndLinkedList()
  437:     {
  438:         do
  439:         {
  440:             yield return new WaitForEndOfFrame();
  441:         } while (!FindObjectOfType<BinaryTreeStructure>() || !FindObjectOfType<Linke
dListStructure>());
  442:     }
  443:
  444:     public IEnumerator WaitForSelection()
  445:     {
```

```
  446:         bool flag = false;
  447:         while(!flag)
  448:         {
  449:             foreach(Highlightable h in FindObjectsOfType<Highlightable>())
  450:             {
  451:                 if (h.highlighted)
  452:                     flag = true;
  453:             }
  454:             yield return new WaitForEndOfFrame();
  455:         }
  456:     }
  457:
  458:     public IEnumerator WaitForDeletion()
  459:     {
  460:         int maxCount = FindObjectsOfType<Node>().Length;
  461:         int prevMaxCount = maxCount;
  462:
  463:         //wait for count to decrease
  464:         while(prevMaxCount >= maxCount)
  465:         {
  466:             maxCount = FindObjectsOfType<Node>().Length;
  467:             if (maxCount < prevMaxCount)
  468:                 yield break;
  469:             yield return new WaitForEndOfFrame();
  470:             prevMaxCount = maxCount;
  471:         }
  472:     }
  473:
  474:     public IEnumerator WaitForConnections()
  475:     {
  476:         //wait for three connections to exist between nodes.
  477:         do
  478:         {
  479:             yield return new WaitForEndOfFrame();
  480:         } while (FindObjectsOfType<Link>().Length < 3);
  481:     }
  482:
  483:     public IEnumerator WaitForTwoNodesToBeFarApart()
  484:     {
  485:         float distance = -1;
  486:         do
  487:         {
  488:             //Iterate across each pair of nodes that exists, and check whether they
are some reasonable distance apart.
  489:             List<Node> nodes = new List<Node>(FindObjectsOfType<Node>());
  490:             List<Tuple<Node,Node>> nodePairs = nodes.SelectMany(x => nodes, (x, y) =
> Tuple.Create(x, y)).Where(tuple => tuple.Item1 != tuple.Item2).ToList();
  491:             foreach(Tuple<Node, Node> pair in nodePairs)
  492:             {
  493:                 distance = Math.Max(Vector3.Distance(pair.Item1.transform.position,
pair.Item2.transform.position), distance);
  494:                 //Only check one pair per frame, in order not to overload the game.
  495:                 //Just kidding, this doesn't start meaningfully affecting framerate
until there are over one hundred nodes on screen, so it's a non issue.
  496:                 //yield return new WaitForEndOfFrame();
  497:             }
  498:             Debug.Log($"Current max distance: {distance}");
  499:             yield return new WaitForEndOfFrame();
  500:         } while (distance < 7);
  501:     }
  502:
  503:     public IEnumerator WaitForTwoNodesToExist()
  504:     {
  505:         do
  506:         {
  507:             yield return new WaitForEndOfFrame();
  508:         } while (FindObjectsOfType<Node>().Length < 2);
  509:     }
```

```
  510:
  511:     public IEnumerator WaitForAllNodesToExist()
  512:     {
  513:         //While all four node types are not present in the scene, wait.
  514:         do
  515:         {
  516:             yield return new WaitForEndOfFrame();
  517:         } while (!(FindObjectOfType<BinaryTree>() && FindObjectOfType<LinkedList>()
&& FindObjectOfType<Digraph>() && FindObjectOfType<Undigraph>()));
  518:
  519:     }
  520:
  521:     //Coroutines coopted from Algorithms.cs
  522:     public static IEnumerator WaitForKeyDown(KeyCode keyCode)
  523:     {
  524:         while (!Input.GetKeyDown(keyCode))
  525:             yield return null;
  526:         yield return new WaitForEndOfFrame();
  527:     }
  528:
  529:     public static IEnumerator WaitForKeyUp(KeyCode keyCode)
  530:     {
  531:         while (!Input.GetKeyUp(keyCode))
  532:             yield return null;
  533:     }
  534:
  535:     public static IEnumerator WaitForOVRInput(OVRInput.Button button)
  536:     {
  537:         while (!OVRInput.GetDown(button))
  538:             yield return null;
  539:         yield return new WaitForEndOfFrame();
  540:     }
  541:
  542:     public static IEnumerator WaitSeconds(float s)
  543:     {
  544:         yield return new WaitForSeconds(s);
  545:     }
  546:
  547:     #endregion TutorialEnumerators
  548:
  549:     #region NewTutorialSteps
  550:
  551:     public IEnumerator StartStep()
  552:     {
  553:         actions.SetCSEnabled(false);
  554:         restrictCSToggle = true;
  555:
  556:         worldspaceTempText.transform.position = Camera.main.transform.position + bas
eOffset;
  557:         worldspaceTempText.setTempText($"Welcome to the tutorial! To begin, please p
ress {(actions.vr ? "Both Triggers" : "Ctrl+Space")}. After this, you'll be able to advance
 the tutorial with {(actions.vr ? "Any button on the top of the touch controllers" : "Space
")}");
  558:         yield return WaitForAttentive();
  559:
  560:     }
  561:
  562:     public IEnumerator IntroductionStep()
  563:     {
  564:         worldspaceTempText.setTempText($"Throughout the upcoming modules, you'll see
 a variety of learning materials in the 3D space in front of you.");
  565:         yield return WaitForNextStep();
  566:
  567:         worldspaceTempText.setTempText($"You can use {(actions.vr ? "your head to lo
ok around." : "your mouse to look around." )}");
  568:         yield return WaitForNextStep();
  569:     }
  570:
```

```
  571:     public IEnumerator GoBackStep()
  572:     {
  573:         worldspaceTempText.setTempText($"At any time, you can press {(actions.vr ? "
either trigger (as if it was a button)" : "the R key")} to go back one step.");
  574:         yield return WaitForNextStep();
  575:     }
  576:
  577:     public IEnumerator InteractionStep()
  578:     {
  579:         worldspaceTempText.setTempText($"Additionally, you'll have to interact with
many of the elements you see. Let's try that now.");
  580:         yield return WaitForNextStep();
  581:
  582:         if(actions.vr)
  583:         {
  584:             worldspaceTempText.setTempText($"You can point at them with your left ha
nd, and click the trigger on the controller to interact. A line will be drawn to it to let
you know it's interactable.");
  585:             yield return WaitForNextStep();
  586:         }
  587:         else
  588:         {
  589:             worldspaceTempText.setTempText($"You can use the mouse to look at the el
ement, and click on it. The crosshair in the center of your screen will turn green when you
 are looking at an interactable.");
  590:             yield return WaitForNextStep();
  591:         }
  592:
  593:         Quiz quiz = Quiz.CreateQuiz("", new string[] { "Press me!" }, 0);
  594:         quiz.transform.position = worldspaceTempText.transform.position + new Vector
3(0, -5, 0);
  595:         quiz.qIdentifier = "SAMPLE";
  596:         yield return quiz.WaitForQuizImmediate();
  597:         Destroy(quiz.gameObject);
  598:
  599:         quiz = Quiz.CreateQuiz("", new string[] { "Press me!" }, 0);
  600:         quiz.transform.position = worldspaceTempText.transform.position + new Vector
3(0, 5, 0);
  601:         quiz.qIdentifier = "SAMPLE";
  602:         yield return quiz.WaitForQuizImmediate();
  603:         Destroy(quiz.gameObject);
  604:
  605:         quiz = Quiz.CreateQuiz("", new string[] { "Press me!" }, 0);
  606:         quiz.transform.position = worldspaceTempText.transform.position + new Vector
3(-5, 0, 0);
  607:         quiz.qIdentifier = "SAMPLE";
  608:         yield return quiz.WaitForQuizImmediate();
  609:         Destroy(quiz.gameObject);
  610:
  611:         quiz = Quiz.CreateQuiz("", new string[] { "Press me!" }, 0);
  612:         quiz.transform.position = worldspaceTempText.transform.position + new Vector
3(5, 0, 0);
  613:         quiz.qIdentifier = "SAMPLE";
  614:         yield return quiz.WaitForQuizImmediate();
  615:         Destroy(quiz.gameObject);
  616:
  617:         worldspaceTempText.setTempText("Nice job! You've now got all you'll need to
complete the modules.");
  618:         yield return WaitForNextStep();
  619:
  620:     }
  621:
  622:     public IEnumerator GUIDStep()
  623:     {
  624:         worldspaceTempText.setTempText("You'll now recieve your study ID.");
  625:         yield return WaitForNextStep();
  626:
  627:         Guid id = Guid.NewGuid();
```

```
  628:
  629:         /*
  630:         if (PlayerPrefs.HasKey("study_id"))
  631:         {
  632:             worldspaceTempText.setTempText($"There is an existing cached study id -
{PlayerPrefs.GetString("study_id")}. Please let the researcher know about this.");
  633:             yield return WaitForNextStep();
  634:         }*/
  635:
  636:         PlayerPrefs.SetString("study_id", id.ToString());
  637:
  638:         string path = Path.Combine(Application.persistentDataPath, $"{id.ToString()}
-results.txt");
  639:         File.WriteAllText(path, $"{id.ToString()}|{Application.platform}|{actions.vr
}\n");
  640:
  641:         worldspaceTempText.setTempText($"Your study ID is: {id.ToString()}. Please r
ead the first eight characters out to the researcher who is present with you :)");
  642:         yield return WaitForNextStep();
  643:
  644:         worldspaceTempText.setTempText($"Your study ID is: {id.ToString()}. Are you
all set?");
  645:         yield return WaitForNextStep();
  646:
  647:
  648:     }
  649:
  650:     public IEnumerator FinishStep()
  651:     {
  652:         worldspaceTempText.setTempText($"Great job!");
  653:         yield return WaitForNextStep();
  654:
  655:         worldspaceTempText.setTempText($"You'll now be returned to the main menu. Pl
ease select the next module, and continue. Thank you for your participation in this study."
);
  656:         yield return WaitForNextStep();
  657:
  658:         SceneManager.LoadScene("Main Menu");
  659:     }
  660:
  661:     #endregion NewTutorialSteps
  662:
  663:
  664: }
```