Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Sindre Aarstrand

UniVRsity - A Virtual University

Master's thesis in Computer Science
Supervisor: Frank Lindseth
June 2019

NTNU
Norwegian University of
Science and Technology

Sindre Aarstrand

UniVRsity - A Virtual University

Master's thesis in Computer Science
Supervisor: Frank Lindseth
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Virtual Reality is an “up and coming” technology. Hardware became available to consumers in 2016, and several big companies are investing a lot to improve these technologies, as well as making them more affordable. The main focus of VR from the start has been immersive games, but there is belief that there are other areas where VR can be utilized as well.

This thesis wanted to experiment with using VR in a learning setting. To do this, an application was developed with the aim of helping to teach topics from different university courses to students. The course “Algorithms and data structures” was the course chosen for the prototype, as this is a course many struggle with. This application was developed in the Unity 3D game engine, and it was evaluated through performing user tests followed by a questionnaire. The results revealed nothing in regards to learning, but rather how usable the developed prototype was, and whether the participants believed in using this technology for learning. The usability was not great, but it was not particularly bad either, as it could with minor additions become a lot better. And yes, the participants believed this technology should be used in learning. Unanimously so.

The prototype is hosted publicly on Github[4] and is available for use, with or without modifications, by anyone interested in doing so. A video tour of the final prototype can be watched by following this link[5].

Contents

1 Introduction

This project is a continuation of a project I did in the previous semester. The topic of that article, and therefore also the topic of this article, is using cross reality, or XR, for learning. This is a very broad topic and with help from the supervisor, it was narrowed down. It was agreed upon to focus on virtual reality, or VR, as the technology within the XR domain, and as for what to teach or for users to learn, it was decided to do courses within the computer science field. More specifically "Algorithms and data structures", as this is a course many computer science students struggle with. The mission for the previous project was to make a prototype of a VR application where the user can learn some of the syllabus from the course "Algorithms and data structures". It would do this by attempting to better visualize the abstract parts compared to illustrations in a text book and drawings on a whiteboard or blackboard. The mission for this project, is expanding and improving the prototype developed in the other project and perform tests on it to see if the improvements were adequate.

The previous project was based on two similar projects from the preceding year under the same supervisor. It used these for inspiration, and to learn from their successes and failures. However, the prototype in this previous project was not based on either of the prototypes from the earlier projects, it was made entirely from scratch.

1.1 Motivation

My motivation for this project was rooted in two things. The first was that I had not long before the previous project started tried playing a few games in VR at a friend's place, and I was really fascinated by how immersive the experience could be. When learning it helps, at least for me, when I am immersed and focused on the topic at hand, and I believe VR could be an excellent medium for aiding when it comes to learning and teaching. The second was that I was curious about how development for this technology worked. This was more of a motivation for the previous project, as I now know how this happens. The motivation then naturally shifted towards improving upon the prototype for this project. Seeing improvement and growth in one's creations is fun and motivating.

I imagine the University's motivation for having this topic available for a masters degree, lies in the fact that they want to explore different use cases for new technology, and especially to see if it can be used as aids for their courses in the future. Not that VR is especially new, but it is new in the sense that there has been improvements in the hardware and the availability of this hardware in recent times. This has put VR more in the spotlight, and there is a lot happening in this field currently. The University wants to be a part of this.

1.2 Research Questions

The research questions(RQs) this project will attempt to find an answer to are listed below.

- **RQ1**: Is the new prototype improved compared to the previous version?
- **RQ2**: Do people think VR is suited in a school or learning setting?
- **RQ3**: Will the developed prototype indicate that using VR might be suited for teaching/learning?

1.3 Contributions

The contributions of this project is mainly the prototype itself. The code of the prototype can be reused or extended by anyone who wishes to do so, as it is hosted publicly on Github[4]. A playable build of the current version is also available for download from a google drive link[6]. The evaluation revealed that all the participants believe that there is potential in using VR for learning. This might not necessarily be a contribution, but I mention it regardless because it could inspire more research into this field.

1.4 Thesis Outline

The structure of this article is as follows. The next chapter is the background. This chapter will present some theory about learning, learning combined with technology, and games. It will also present the technologies used in this project and the software considered and used. Methodology, the third chapter, will present the planning of the prototype, the tools for implementing the prototype, and the plan for how to perform the testing. The fourth chapter, Results, will describe how the prototype turned out, show some insight into how the prototype was made, and reveal the results of the testing. After that you can read about my thoughts around these results, and ideas for how things could have been done differently, in the fifth chapter: Discussion. In the final chapter, six, the conclusion as well as ideas for future work is presented.

2 Background

This section will provide background information about the technologies and software utilized in this project, as well as theory addressing learning, learning with multimedia, and enjoyment in games. It will build the foundation for the planning and design choices made in this project.

2.1 Virtual Reality

Virtual reality(VR) has many definitions, but in the context of this project VR refers to a virtual environment with spatial and orientation tracking of a head mounted display(HMD) and controllers. In VR the virtual environment's visuals are 100% computer graphics, while the interaction with these generated visuals are done through the movement of the physical body of the user. VR is also a part of the term XR, or Cross Reality, which includes VR, mixed reality(MR), augmented reality(AR), and a few more. Most of these use a mixture of the real world and graphics in the visuals, which contrasts to VR's 100% graphical visuals.

The HTC Vive(Figure 1) and the Oculus Rift(Figure 2) are the first generation of consumer VR-kits and are also currently the standard in desktop powered VR. This first generation of VR devices first hit the consumer market in 2016, but development kits and prototypes have been available a lot longer. Oculus just released two new VR sets, the Oculus Quest and the Oculus Rift S. The Oculus Quest is a standalone VR set(no need for a computer), like the Oculus Go, but with proper controllers and hardware capable of running some games. The Oculus Rift S has the sensors built it to the HMD and this makes setup a lot easier. It also has an LCD panel instead of the OLED in the original, and this means that the contrast and deep blacks are not as good in the Rift S but it has a higher resolution and less "screen door" effect. Valve is also about to launch its VR set, the Valve Index. The selling point for this VR set is its high refresh rate of the panels and the controllers, which does not need to be griped and has tracking for all the fingers individually. HTC has announced the Vive Cosmos, and a Vive Pro with eye tracking. Considering some reviews and hardware specifications, these VR sets will probably not be considered the second generation, but it is a step along the way.

Figure 1: HTC Vive Pro and controllers

Figure 2: Oculus Rift and Touch controllers

2.2 Software

Unity and Unreal Engine 4 were the only game engines considered for developing this application. They are the most common engines for independent developers to chose, especially for VR development. Both are also free to use for non-commercial use, and mostly free for commercial use as well, with a few limitations and or exceptions. This makes these engines ideal for beginners just starting out or wanting to test game development. One of the major differences in these engines is the lighting. In Unity the lighting is baked, while in Unreal it is not. This gives Unreal the edge when realistic graphics is wanted. Unreal also focuses more on 3D games giving them the advantage there. On the other hand, this gives Unity the edge in 2D games. They also use different programming languages.

2.2.1 Unity

Unity has support for several languages, but the two most used are C# and Javascript. To such an extent that Unity is dropping/has dropped support for other languages. Unity is based around game objects and scripts, or components, attached to these game objects. These components, if they inherit from the default unity class: MonoBehaviour, have a “Start” method which is run at the time the object is created, and an “Update” method which is run every frame. MonoBehavior also has access to many more methods, but these are the basics. Unity is a mixture of visually programming and coding. If a variable in Unity is marked as public, it can be accessed visually within the editor. Unity also has an Asset store within the editor where all sorts of assets, like models, textures, sounds, and visual effects, can be purchased or downloaded for free. These assets are made by the community, and the quality is therefore varying, but there is a rating system to help chose wisely.

Figure 3: Unity logo

2.2.2 Unreal Engine

Unreal Engine uses C++ for development. However, it also provides ”blueprints” which is an interface for scripting visually(drag n’ drop). Strictly speaking, this means that both Unreal Engine and Unity can create games without having to code. Unreal is based around actors, which are almost equivalent of game objects in unity. Actors have components just like game objects in unity has. In addition they can have blueprints or scripts attached to them.

Figure 4: Unreal Engine logo

2.2.3 SteamVR Plugin

Since there exist many different forms of controllers and HMDs, Valve has created a plugin for Unity and Unreal Engine, SteamVR plugin[7]. This plugin makes it possible to support a wide variety of devices using the same code. Using this will minimize the code needed to support a wide variety of devices. The difference between a VR game and a normal 3D game is that VR games need to somehow reflect the users movements and actions in real life within the game. I.e. move the hands to where the user moves their controllers, look in the direction the user is pointing their head, and all these things. The focus of this plugin is therefore to make the camera follow the users head, load models of the controllers in use, make these controller models follow the users hands, handle the inputs of these, and estimate how the users hands look while using these controllers.

2.3 Multimedia Learning Theories

The cognitive theory of multimedia[12], which is based upon the cognitive load theory[16], states that the visual input and the auditory input are processed on different channels, with a limited cognitive capacity for processing these inputs. This means if the visual channel is overloaded with input, the auditory channels processing will suffer, and vice versa. From this follows that irrelevant images, sounds, or words can harm the learning in the learner, and should therefor be avoided. In the book Multimedia learning[11], Mayers present twelve principles for better design when using multimedia for presenting material. These principles are backed by both the cognitive theory of multimedia, and empirical studies.

The principles are as follows:

- Coherence Principle - People learn better when extraneous words, pictures and sounds are excluded rather than included.
- Signaling Principle - People learn better when cues that highlight the organization of the essential material are added.
- Redundancy Principle - People learn better form graphics and narration than from graphics, narration and on-screen text.
- Spatial Contiguity Principle - People learn better when corresponding words and picture are presented near rather than far from each other on the page or screen.
- Temporal Contiguity Principle - People learn better when corresponding words and pictures are presented simultaneously rather than successively.
- Segmenting Principle - People learn better from a multimedia lesson is presented in user-paced segments rather than as a continuous unit.
- Pre-training Principle - People learn better from a multimedia lesson when they know the names and characteristics of the main concepts.
- Modality Principle - People learn better from graphics and narrations than from animation and on-screen text.
- Multimedia Principle - People learn better from words and pictures than from words alone.
- Personalization Principle - People learn better from multimedia lessons when words are in conversational style rather than formal style.
- Voice Principle - People learn better when the narration in multimedia lessons is spoken in a friendly human voice rather than a machine voice.
- Image Principle - People do not learn better from a multimedia lesson when the speaker's image is added to the screen.

The fact that VR gives the opportunity to do anything or be anywhere, has the potential to unintentionally be very distracting if the environment is very detailed and or dynamic, as this can draw the attention away from the presented material the user is supposed to interact with and learn from. However, always keeping the coherence principle in mind, and thinking twice when adding content of whether it is an aid in the learning process or not, can help in avoiding such a situation. An interesting environment will probably pique a users interest initially but might be distracting and decrease the efficiency and effectiveness of the learning material. There seems to be a trade-off there, unless the environment changes into something less distracting when the application requires the users attention elsewhere.

When it comes to deciding how to design the instructions, the redundancy principle suggests it has graphics and narration, but without text. The reasoning lies with the redundant text prompting extra visual processing, resulting in less learned. And while in the area, the modality principle suggests that instructions should consist of graphics and voice, rather than graphics and text. By having both text and graphics the visual channel might get overloaded, while with graphics and voice, the input is split among the auditory and visual channels.

Taking the pre-training principle into consideration, it would suggest that having this prototype as an addition to lectures or other classic teaching methods, rather than as a stand-alone product, is more effective. The reason for this is that knowing some of the terms and characteristics of a concept in advance means that the user has kind of allocated a space in his/her knowledge space where the new information can go. An analogy might be a cake. A cake can be eaten without knowing the ingredients or how it is made. However, presented with only the ingredients and it is very hard to imagine what they become when combined in a specific way.

One of the principals that the masters degree project of Kong and Kruke[9] followed well, is the segmentation principle. The prototype allows the players to control the pace of the learning to a high degree. The players can choose which sorting algorithm to learn, and they can do it step by step with as much time as they need per step. This is something to keep in mind for this project too.

Lastly from these principles it is worth noting that people learn better when words are in conversational style rather than formal style, because the learner will be prompted to try harder when they feel that someone is talking to them specifically. Also, if these words are in the form of voice rather than text, the learner will be more engaged if the voice is human and kind compared to robotic and cold. This is according to the personalization and voice principles.

2.4 Video Games and Fun

Even though this project is based around an educational game, it is still a game, and games are inherently made to be fun or entertaining. This is something which should, therefore, also be considered in this project. In 2005, Penelope Sweetster and Peta Wyeth, made an attempt at creating a model for evaluating and designing fun games. This model was based on Csikszentmihalyi's Flow theory[8], which is a theory or model on enjoyment in general, and the adapted version was named, creatively so, GameFlow[15].

GameFlow consists of eight elements; Concentration, Challenge, Player Skills, Control, Clear Goals, Feedback, Immersion, and Social Interaction. For someone to enjoy a game, all of these elements does not necessarily need to be fulfilled, but at least a few of them. As an example, a game can be fun even though it does not include social interaction. Anyways, let's review what lies within these elements.

- Concentration - Games should require concentration, but also help the player in doing so
- Challenge - Games should be sufficiently challenging
- Player skills - Games should support player skill development and mastery
- Control - Players should feel that they are in control of their actions in the game.
- Clear goals - Games should provide players with clear goals at appropriate times
- Feedback - Players should receive appropriate feedback at appropriate times
- Immersion - Players should experience deep but effortless involvement in the game
- Social Interaction - Games should support and create opportunities for social interaction

Using VR will surround the entire field of vision for the player and should in theory increase the users attention towards the VR application. However, it is still not guaranteed that the player will aim its attention within the virtual world to the place where the developer intended for the player to have their focus. It will only guarantee that the attentions is somewhere within the virtual world. Immersion is closely linked to concentration and is another point where using VR as a platform might be beneficial. The selling point for VR is the immersion that it gives and strives to get even better at. This suggests that there is less need to focus on designing the game in such a way that it promotes immersion, since the VR platform does a lot of work for this point by just being this platform. The other points will probably not gain too much just from the prototype being in VR. Nevertheless, all eight points should be taken kept in mind when designing the prototype.

2.5 Four Keys to Fun

Lazzaro found that there are four ways to make player have fun and enjoy games[10]. This is merely an identification of where the fun comes from, and does not directly provide a tool for better game design. However, knowing what can make a game fun might prove helpful anyways. It can also be used to analyze existing game design, even though it will not provide suggestions for improvements.

The four keys are as follows:

- Hard Fun (Fiero)
- Easy Fun
- Serious Fun
- People Fun

Hard fun is, as its name implies, when the fun comes from the difficulty within the game. If something is difficult and the player fails, they will feel frustration, and as they keep failing, their frustration will increase. However, if the player gets a tiny bit better every time, the player will eventually complete the challenge, and when that moment comes, all of the frustration goes away in an intense emotion referred to as "fiero", and after the excitement of finally having completed the challenge, the player feels relieved... until the next challenge is at hand, and the process repeats. It should be noted that there exists different kinds of frustration and not all of them will lead to the player having fun.

Easy fun comes from curiosity. If the game inspires curiosity the player will enjoy the game. Having freedom in the interaction can help in this regard, as the player might then be tempted to explore the boundaries of what is possible to do. The chain of emotion in this key of fun is: Curiosity - Surprise - Wonder or Awe - Relief - Back to curiosity again. Put into a scenario: A player is curious as to what happens when he does a certain kind of action, the result of the action performed was not expected in the player and is surprised. After the initial surprise the player is in wonder by this result and feels relieved as the itch from the curiosity is now gone. Then it repeats when the player pops up with a new question he or she wants answered. However, for this to be effective, the results of the exploration and fooling around need to be balanced between novelty and familiarity; too novel - and the player falls into disbelief, too predictable - and the player is bored or lose interest.

Serious fun comes when the player is playing to change how he or she thinks, behave, or to accomplish real work. This means that serious fun is very subjective. If a game is low on serious fun, it will feel like a waste of time. And the emotions linked to serious fun usually comes from the graphics, sound, and or atmosphere within the game. It can also be a more cognitive fun as it might give players something to think about outside of the game.

People fun has to do with everything which connects the game with other people. This can be by talking about the game with your friends face to face, by having a chat in-game, or what have you. Interacting with other people is an important source for feeling different emotions. It can also create identities and shape relationships.

The more a game applies these four key, the more entertaining it will be. And a fun game is a more effective one. Therefor these keys should be kept in mind both when designing the prototype, and during the testing.

2.6 System Usability Scale

The System Usability Scale[18], or SUS for short, is a way for measuring usability in a system, be it hardware or software. It consists of a questionnaire with 10 fixed statements where the responses to these should be on a scale from 0 to 4, where 0 equals "Strongly disagree" and 4 being "Strongly agree". The creator of this measuring tool is John Brooke, in the year 1986. SUS has become the industry standard due to its simplicity for the participants, and the fact that it does not need a large sample size to get reliable results. The following 10 items are the fixed statements of a SUS questionnaire:

- I think that I would like to use this system frequently.
- I found the system unnecessarily complex.
- I thought the system was easy to use.
- I think that I would need the support of a technical person to be able to use this system.
- I found the various functions in this system were well integrated.
- I thought there was too much inconsistency in this system.
- I would imagine that most people would learn to use this system very quickly.
- I found the system very cumbersome to use.
- I felt very confident using the system.
- I needed to learn a lot of things before I could get going with this system.

Scoring the response of the SUS questionnaire follows these steps[14]:

- Odd items: add the score to the cumulative score
- Even items: flip the score then add to the cumulative score(subtract given score from 4. In other words if a participant gives 1 as a response on a question then the flipped score will be 4 - 1 = 3)
- finally after summing all scores, multiply by 2.5 to increase the score range from 0-40 to 0-100

As the scores of a SUS is between 0 and 100 it can be tempting to look as these as a percentage. This is, however, the wrong way to interpret the scores. The average SUS score is 68 and is therefor in the percentile rank of 50%. The percentile rank increases quite rapidly from there and is in the percentile rank of 70% already at a score of 74, and percentile rank of 90%(top 10%) at a score of 80.4. Below the 68 score the percentile rank drops to 30% at about 60, and to 10% at about 47. In other words the percentile rank drops really fast close to the average score, and a bit less so further away from it. A score of 67 and 69 is therefore quite far apart from each other in terms of the percentile rank.

The System Usability Scale was used in the previous project as the next section will give some more information about. It should probably also be used in this project, as it is very easy to use for the participants, and it gives a good indication as to whether or not the application is user friendly.

2.7 Previous Work

As already stated, this project is a continuation of another project. The prototype developed in that project consisted of a single scene, which was for teaching sorting algorithms. This scene had implemented two sorting algorithms: bubble sort and insertion sort. Click here to see a video[3] of the functionality and contents of the first prototype. The interaction was purely through pointing with a laser in the right hand and pressing the trigger when pointing at objects which were interactive. This prototype was also user tested, though only by 5 people. The user test consisted mainly of a usability test, utilizing the System Usability Scale. What the results of this testing showed was that the usability slightly below average, but it also seemed to be related with whether the player had taken the course or not.

2.7.1 Design

As can be seen in video of the previous prototype, the design of the scene where the sorting happens is formed kind of like a semi circle. To the front, the representation of the array is shown, with the array elements represented as yellow cuboids with height visually representing their value, and the array itself represented as a cuboid laying on the ground with its elements hovering above it. To the left of the array, still to the front, lies a cube representing the storage area, where algorithms needing to save values outside of the array can place them. On the floor in front of where the player stands, there is a panel. This panel shows the state of the algorithm, the pseudo code for the algorithm, and shows output from the most recent comparison of elements. To the left are two walls with buttons aligned vertically. One for selecting algorithm, and one with buttons for starting a demonstration of the algorithm, starting the algorithm over again, and getting a new randomly generated array. The buttons are thin cuboids with big labels on them representing their respective actions. The other side, to the right, contains a wall with action buttons. The available actions are compare, swap, store, and copy to.

Reading a value as a number in text is easy, Seeing the value as the length of an object is easier. That is the reasoning behind the design of the array elements. The two projects from the year before mainly used cubes of equal size, though an option to show the difference in size as one shrinking and one growing was available in one of them. The reason for choosing equal sized cubes in their prototypes are most likely grounded in their choice of interaction. They represented arrays as cubes, and the player had to do the physical motion of picking up the cubes to sort them. Picking up big objects is more awkward than using hand-sized objects, and is probably their reason for their design choice. However, my design uses a laser pointer for interaction, and there is no more an issue of awkwardly picking up large objects. Using a laser pointer opens up a lot of new options compared to what the other prototypes had, and loses some opportunities in comparison also. One of the reasons behind choosing a laser pointer as interaction mechanism, was that it removes a lot of the distraction involved with having the ability to play around with stuff with virtual hands. Just having hands seem to distract people who has not tried VR before as they are amazed by the feeling of seeing their own hands move in the virtual world. And as the theory presented earlier suggests, distractions are bad when it comes to learning.

2.7.2 Development

The chosen game engine in the previous project was Unity, because it felt easier and quicker to use after some tinkering about in both Unity and Unreal. Research into which engine to choose for the project did not really lead anywhere, and suggested it should just be up to personal preference. Seeing as I was not particularly determined to make a realistic looking game, there was no need for the non-baked lighting which Unreal provides.

After the decision of which game engine to use, it had to be properly learned. This was done through a combination of watching youtube videos, reading the documentation, doing tutorials, and experimenting. The main channels used were FusedVR[2] for VR specifics and Brackeys[1] for general Unity things. Both of these channels used C# as their programming language of choice in their videos, and it ended up being the language I chose as well. Youtube videos are great because one can learn how professionals or people with a lot of experience with the engine uses it, and their workflows. They can also be helpful in terms of how to solve common problems or help in avoiding common mistakes beginners do. Before the development of the actual prototype went down, a couple of mini-VR-games were made. This was just to get used to the engine, and also how to interface with the VR gear. This experimentation proved to be very effective as it gives hands on experience and a sense of how to work with this technology.

To be able to use any VR-kit without writing hardware specific code, the SteamVR Plugin for Unity was used. To be able to work anywhere, the version control system Git was used, and it is hosted on Github, as a public repository. Hosting the project on GitHub in a public repository will enable other developers to further build upon this project, should it turn out to be a good one. There was no need to use Unity's collaboration tool/version control system, Unity Collaborate[17], as there was only one developer during the extent of this project.

2.7.3 Architecture

The architecture in the previous project, was focused around the game having to be modular, and easily expanded. To do this, sorting algorithms had to be generalized such that the common functionality would not need to be re-implemented, instead only specifics for the new algorithm. The same went for user actions. They were also generalized such that it would be easy to add new actions. This architecture makes it a lot easier to add new algorithms and actions, but it requires more thought into the initial development. The algorithms was generalized fairly well during the development, however, actions, not so much. Due to the way the algorithms were made, they were completely isolated from the arrays them selves, which meant that actions could not be generally designed. The result was actions just having a type, an enum describing what action it represents, and some integer values representing the position within the array the action should affect. This in turn meant that what the different types actions actually do, is up to whichever component is handling the action. The positive with this is that the action types can be reused in other scenes, as they have no actual actions linked to them. However, in this case it probably would have been smarter to not have the sorting algorithms be disconnected from the array representation, and therefor the actions separated from them.

2.7.4 Testing

The testing was quickly brushed over in the introduction of this prototype, saying it was mainly a SUS questionnaire and that 5 people participated. The results of the SUS was a score of 67.5 which is below average. However looking at the background of the testers the scores could indicate that having taken the course in advance was correlated to a higher ease of use. The average score of people having had the course was to 75.83, which is a good score. Just watching the tests also gave a lot of insight. People thought that the prototype was some some sort of playground, where one could also see how some algorithms worked. This was not the case, and it confused the testers. The laser pointer interaction seemed to be intuitive, but how to perform the different actions on the array was not very intuitive.

2.7.5 Unfinished Ideas

Some of the features or ideas for the project was not implemented due to time constraints and or other difficulties. The ones worth mentioning are these:

- Voice - Add voice which gives instructions instead of/in addition to the text.

- Overworld - Add an overworld where the player can travel to different courses.
- Tree traversing algorithms - Add a scene where the player could learn different tree traversing algorithms.
- Highlight line of code being executed
- Tutorial - Add a form of tutorial explicitly teaching the player what they can do.
- Remove the gun model - Should be a normal laser pointer instead

These ideas should be considered when planning the further development of this prototype in this project.

3 Methodology

This section will present how the prototype was planned, the different choices made in regards to how the development should be done, and also how the testing should be performed.

3.1 General Plan

The plan going into this project was to continue the development of the prototype developed in the previous project. Making improvements and adding new features to it, and then test it again to see if it was actually improved. As far as design goes there will not be any major changes, unless something which forces this occurs. There will however be additions to it, and these additions will try to achieve a similar style. This is to be consistent with how things work, not causing unnecessary confusion in the players. The planning of the expansion of content and making improvements will happen in iterations. Come up with a next step(a general idea, no specifics), figure out how to implement this idea, implement it, then from there figure out where to go next. The next step for each iteration would mostly come from the meetings with the supervisor.

Coming from the previous project the structure of the prototype was just a single scene. However, the plan for a finished product looked something like that which can be seen in Figure 5. The question marks represent any number of courses or topics, and the structure is hierarchical. The overworld is at the top of the hierarchy and form there the player can navigate to any course hub. The course hubs are the next level of the hierarchy where the player can either navigate back to the overworld, or to any topic within the course. The topics are the lowest level in the hierarchy, and the player can only navigate back to its course hub from there. This hierarchy will be the guide when adding new content in this project.

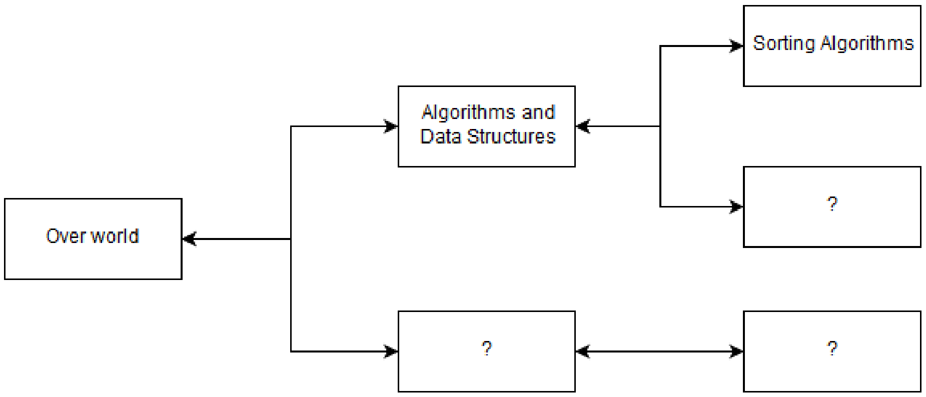


Figure 5: The idea of how the scenes in the final product should be arranged. The question marks can be any number of boxes.

3.1.1 Iteration 1

The goal for the first iteration was to add another topic from the course Algorithms and data structures. The topic chosen was data structures, despite the previous project having some ideas around tree traversing algorithms. The reason for this, was to better represent the content of the course. It is not only about algorithms. Another idea which were considered was to start adding another course like Machine learning. This would imply a scene structure like the one seen in Figure 6. However, this was discarded for the time being, and the structure would instead look like the one presented in Figure 7

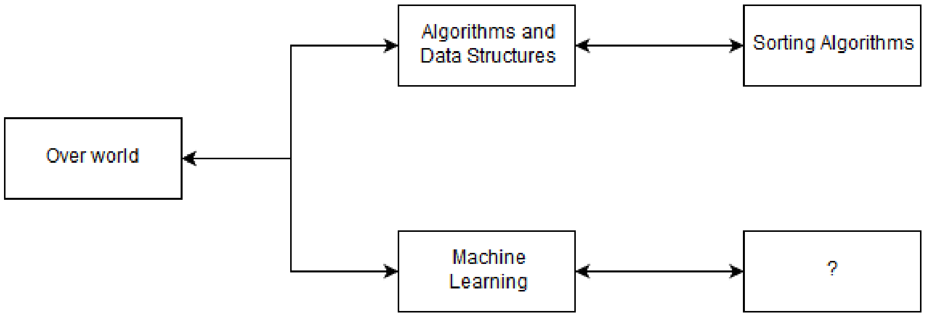


Figure 6: How the scenes in the game would be arranged, if the machine learning course was added instead of the data structure topic.

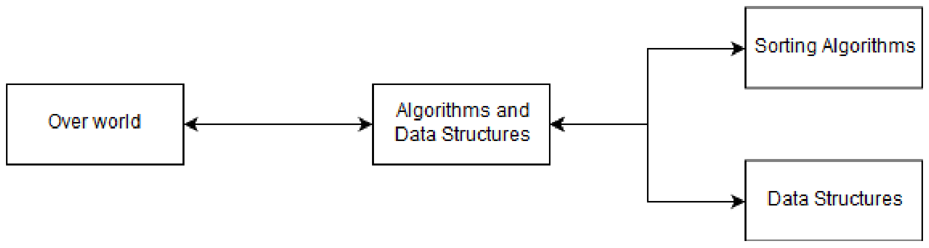


Figure 7: The plan for how the scenes in the game would be arranged, after the first iteration

3.1.2 Iteration 2

The next iteration after the initial data structure scene was added, was to go back to the sorting algorithm scene and add a better and more commonly used algorithm to the mix. The algorithm chosen was quick sort, as it fits these criteria.

3.1.3 Iteration 3

In the third iteration, the idea was to add some form of use case to each of the data structures. The idea behind that idea was to show the player how this data structure could be used in a real life scenario. As the data structures added in the previous iteration were a stack and a queue, and due to the supervisor teaching visual computations, the decision for use case landed on an image region growing algorithm. This algorithm needs to keep track of which pixels to explore, which works with both a stack and a queue, but will behave differently in how the pattern grows.

3.1.4 Iteration 4

Seeing as the overworld, course hubs, and the navigation between them was still not implemented, it was time to do this. This was planned already back in the previous project but not achieved in the duration of it, and it was long overdue to get started on this. It would still be arranged like the figure from iteration 1, Figure 7. My supervisor had an idea of making the overworld look similar to a school ground or university ground. However, due to time constraints, the looks of it was not the priority. The priority was getting the functionality down. Considering that I have no experience in creating 3D models, textures, or other forms of visuals, the results would probably be sub-par and it would cost me a lot of time.

3.1.5 Iteration 5

After the overworld and navigation was added, I wanted to add the merge sort algorithm. Though this required a lot of rework to the sorting scene in the state it was at, at the time. Merge sort is different to the other three sorting algorithms in that it creates new arrays in the process.

3.1.6 Iteration 6

This time I had realized that I had forgotten to improve things which were commented on or discovered during the testing in the previous project. I had been too focused on adding new features and improving stability and architecture, and forgotten probably the most important part; improving the usability. This iteration would therefore consist of making a lot of improvements in terms of usability and thoroughly testing the application to find and squash bugs before the testing started.

3.2 Development

The choice of game engine and programming language fell on the decision from the previous project, namely Unity and C#, as this project's intentions are to directly improve upon this same prototype. Recreating the previous prototype in another game engine would be a lot of unnecessary work, since the plan is to keep everything and just build on top of the existing prototype, and the other game engine would need to be learned, and probably the programming language the game engine uses as well. This does not fit in with the time schedule of this project. In addition, the gained experience with the game engine from the previous project would shorten further development times. In other words, I was already familiar with using Unity and C# at this point and did not really need to spend more time on learning the engine. This project would continue to work on the same public repository on Github as in the previous project.

This project does not intend to alter the architecture of the existing prototype. It should still focus on being modular, as this makes it easier for other developers to extend it and improve upon it. Having the prototype keep evolving and eventually encompassing a plethora of courses is kind of the idea or hope here. However, considering the time constraint, making things work is more important than having a good architecture. At least in the short term scenario.

3.2.1 Requirements

Following is a list of functional requirements for the project, sorted by the scenes. Non-functional requirements were not set, except for keeping the prototype performance efficient.

Common to all scenes

- Player - A camera and controller reflecting the physical orientation and position of the player hand his/her hands/controllers. Also has a laser pointer in the right hand for interacting with the world
- Show outline on interactive objects when hovering over with the laser pointer
- Interact with interactive objects by pressing the trigger button on the controller with the laser pointer while pointing at it
- Buttons - Interactive object
- Show which buttons are disabled by changing their color, and the color of the outline when hover over

Over world

- Show available courses as different buttons
- Course buttons - Loads the button's course
  - Algorithms and Data Structures

Algorithms and Data Structures

Course hub(entry point):

- Show available topics as buttons
- **Topic buttons** - Loads the button's topic
  - Data structures
  - Sorting algorithms
- Back navigation button - Go back to the over world

Data structures:

- **Play mode** - Learn how the selected structure works
  - Show available structures as buttons
  - **Structure buttons** - Go to play mode for selected structure
    * Stack
    * Queue
  - **Stack and Queue** - Same play mode
    * Show a representation of the data structure

        - Push/Enqueue button - Animate how the structure handles addition of data
        - Pop/Dequeue button - Animate how the structure handles removal/retrieval of data
        - Use case button - Go to use case mode for this structure
        - Back navigation button - Go back to course hub
        - Show explanation of each button
        - Show error message when doing an illegal action containing the error (Overflow/Underflow) and when it occurs

- **Use case mode** - See how the selected structure can be used in a real life scenario
    - **Stack and Queue** - Same use case: Image region growing
        - Back navigation button - Go back to play mode for this structure
        - Show a description of what this use case is when entering the use case mode. Press the button on this panel to hide it and start the algorithm.
        - Show a black and white image with random pattern. Each pixel is an interactive object.
        - Pixel(interactive object) - Select (show a border color to indicate that it is selected)
        - Show a representation of the data structure selected before entering use case mode
        - Show a representation of the the item from the data structure is currently being read/used ("Next" as it points to the next pixel to check)
        - Show a representation of the internal state of the algorithm (Pattern so far, visited pixels, which pixel is next, etc...)
        - Show an explanation of the color coding in the representation of the algorithm
        - **Action buttons**
            - Push/Enqueue - Animate addition of the selected Pixel to the data structure
            - Pop/Dequeue - Animate removal of next pixel from the data structure and into the "Next" position.
            - Check - Animate the checking(whether it is or is not part of the pattern) of the pixel.

        * **Algorithm interaction buttons**
            · Demo(toggle button) - Start/stop the demonstration of the algorithm. The demonstration animates how the algorithm works step by step.
            · Prev - Undo the last step in the algorithm(animate)
            · Next - Do the next step in the algorithm(animate)
        * An attempt at doing an action the algorithm does not expect, it will not perform the action and instead hint at the expected action by blinking the required elements and action button a few times.
        * Data items popped or dequeued into "Next" will fire a laser at the pixel it represents. To indicate which pixel should be checked next

Sorting Algorithms

- Back navigation button - Go back to course hub
- Show available sorting algorithms as buttons
- Show the array as a horizontal set of cuboids(array elements) on top of another cuboid(array stand or representation of the memory location). The elements has a height relative to their value: high value means a tall cuboid, and a low value means a short cuboid.
- Both the array and the the elements are interactive
- Show a Cube representing the storage (where array elements can be temporarily stored)
- Show a representation of the internal state of the algorithm
- Show pseudo code for the selected algorithm
- **Sorting algorithm buttons** - Generates a new random array and changes to this algorithm
    - Bubble sort
    - Insertion sort
    - Quick sort
    - Merge sort
- **Algorithm interaction buttons**
    - Demo(toggle button) - Start/stop demonstrating the algorithm. The demonstration shows the algorithm step by step in the correct order and explains what it is doing.

  - Prev - Undo the last step in the algorithm(animate)
  - Next - Do the next step in the algorithm(animate)
  - New array - Generate a new random array
  - Restart - Resets the algorithm, and array back to start

- **Action buttons**
  - Compare - Compare the value of two elements
  - Swap - Swap the values/position of one or two element(s)
  - Store - Clone one of the elements in the array to the storage
  - Copy to - Two step action: Copy selected element value into the next selection (limitations: can not copy to storage, use Store instead)
  - Pivot - Sets the selected element as a pivot
  - Split - Splits the selected array in two halves
  - Merge - Start the merge of two arrays
- An attempt at doing an action the algorithm does not expect, it will not perform the action and instead hint at the expected action by blinking the required elements and action button a few times.

3.2.2 Implementation

This part will talk a bit about how the major parts in the game was coded, how they work, and how they interact with each other. The different pieces will be classified as either one of these:

- A component - A C# class inheriting from MonoBehaviour.
- A prefab - A template for a game object which can be used to spawn predefined game objects at run-time or for speeding up the scene building process.
- A class - A C# class.
- An interface - A C# interface.

Pieces common on for all scenes:

- **LaserPointerPlayer**

  The "LaserPointerPlayer" prefab was a modified version of the "Player" prefab which is shipped out of the box with Steam VR. The "Player" prefab controls the tracking of the controllers and the HMD. The modified version removed the hand models, and their gesture simulation as this is not needed for the interaction within the game. Instead the hand models were replaced by spheres and attached to the right hand was a laser pointer which is used for interaction. Along with this laser pointer model, a component("LaserPointer") was added to control this model.

- **_FallbackGame**

  "_FallbackGame" is a prefab, and it came to be after a bug which was found late in the development. It was not found until very late in the process as the scenes were usually only tested individually, and not after transitioning from one to the other. As it turned out the player prefab was not destroyed on scene change, and every scene had one of these player prefabs, which meant there would be several player objects in the scene after the first scene was left. This was not obvious as most of the scenes had the player object in the exact same spot and they were just overlapping each other. However, when transitioning from a scene were this object was in another position than the previous scene, it became very obvious as one could see two laser pointers originating from different spots. The solution to this was to create this "_FallbackGame" prefab. The purpose of this "_FallbackGame" object was to check for a player(a "LaserPointer-Player" prefab) and if there was none, instantiate one, else, destroy itself. This made it possible to start the game from any scene and still visit other scenes without getting multiple players, and without only having a player in one of the scenes. This was the initial function of this prefab, but since there would only ever be one of these objects, the opportunity arose to move all other components which were required by all scenes into this same object. Kind of making it into the core of the game, and hence the name "_FallbackGame", where "Fallback" comes from the part where it checks whether or not it is needed, and "Game" coming from the contents being all the stuff required everywhere within the game. The contents are, excluding the player, the "EventManager" and the "ColorManager" components, as of the time of writing.

- **EventManager**

  The "EventManager" is a component which contains every event which could happen in the game, and allows other classes or components to subscribe to these events. The events are set up with delegate types for the different types of events, and matching static events using these delegate types. Having the events be static means that other components does not need a reference to the active "Event-Manager" in order to subscribe to events. Then, all a component need to do in order to subscribe to an event is to define a methos which takes the same parameters as the delegate type of the event, and append that method to the event. Like this:

```
// Subscribe to event
EventManager.Name_of_event += Name_of_function;

// Unsubscribe from event
EventManager.Name_of_event -= Name_of_function;
```

This component also has the responsibility to handle the laser pointer and the interaction with the controllers. It has a reference to the “LaserPointer”, a component attached to the “LaserPointerPlayer” object, which gives the “EventManager” information about the position and orientation of the pointer. Every frame it then uses this information to fire a ray cast from the tip of the laser pointer in the direction it faces. After that it uses the result of the ray cast in order to check whether it is pointing at an interactive object. If it is pointing at an interactive object, it will trigger the hover effect of that object, and if the player is also pressing the trigger button while pointing at this interactive object, it will fire the appropriate event according to the type of interactive object.

- **ColorManager**

  “ColorManager” is a simple component. All it does is provide a way to define colors in the Unity inspector and expose a static instance of this object. This public static object reference makes the defined colors available anywhere within the scene without having to instantiate an instance of this class or having a reference to one. It is also set up to have a Singleton architecture, meaning that there will only ever be a maximum of one instance of this class at any given time.

- **UIButton**

  “UIButton” is an abstract component. The “UIButton” component’s function is to provide an animation for pressing it, and also give the possibility to add methods to trigger when it is pressed. Methods can either be added through code, or through the inspector in Unity. To achieve this possibility an event class inheriting from the generic “UnityEvent<T>” class was created, internally in the “UIButton” class. The “T” can be replaced with whatever type you want to pass to this event. In this case:

```
[Serializable]
public class ButtonEvent: UnityEvent<UIButton> { }

public ButtonEvent onButtonPressed;
```

  As we can see, the “ButtonEvent” inherits from “UnityEvent<UIButton>”. This means that methods subscribed to this event will get a “UIButton” instance passed to them. This was to ensure that any property or functionality of any sub-classes of “UIButton” could be used if needed. The “UIButton” class itself, however, contains no useful information to be used by the subscribed method. The “[Serializable]” part along with the public “ButtonEvent” variable, enables the subscriptions to be done trough the Unity inspector. Calling all the subscribed methods is just a matter of calling the “Invoke” method and passing an instance of the required type along with it. like this:

```
// "this" is in this case the UIButton instance
onButtonPressed.Invoke(this);

```

Sub-classes of “UIButton” are “MenuButton”, “ActionButton”, and “ModeButton”. “MenuButton” just inherits from “UIButton” without altering or extending it in any way. The only reason it exists is to have a component name which indicates what kind of button this is.“ActionButton” is the most extended version. It adds functionality for blinking the buttons, as to hint which action to perform next, and it allows for action buttons to be multi-stepped. This means that when an action button which is set up as a multi-step action is pressed, it will enter an “in progress” state. And to complete this action, the next selection will be registered as the continuation of this action. “ModeButton” has only one addition, which is expose a field which can be used to specify what mode this button should trigger.

- **SceneChanger**

  “SceneChanger” is an abstract component. All it does is provide a method for transitioning to another scene by fading to black and then fading back in to the new scene. To do this it utilizes the “SteamVR_LoadLevel.Begin()” method, provided by the SteamVR plugin, passing the name of the scene to load and the duration of the fade as parameters.

  Current sub-classes are: “MainSceneChanger”, and “AlgDatSceneChanger”.

- **PlayerPositioner**

  “PlayerPositioner” is a component. Its function is to find the player and position it at the location of the game object this component is attached to, when the scene starts. This means that the “_FallbackPlayer” prefab can be placed anywhere in the scene as long as there a “PlayerPositioner” component at the position where the player should be.

The overworld and course hub for “Algorithms and data structures” does not contain any pieces which are exclusive to their scenes. They only use what they need from the common components listed above. The sorting algorithms scene and data structures scene on the other hand, does have exclusive parts. The data structures scene is split into two modes; play mode and use case mode, and these are the important parts from both modes:

- **Stage**

  The “Stage” component exposes methods for switching between the play- and use case modes, and makes sure both these modes are updated with the correct data structure to use(the chosen structure).

- **DStack**

  “DStack” is a component, and the naming of this is not very representative of what it does. It actually contains all the logic for both the stack and the queue. Originally it was split into two but realizing that they were very similar and that only a boolean value was needed to tell them apart, they were merged, and the “DStack” was arbitrarily chosen over the “DQueue”, and it was never renamed to fit its contents better. Its contents are definitions for animations for both the stack and the queue in regards to adding and removing data, exposes methods for triggering these animations, keeps references to a few positions it uses for animations, and it keeps track of how many data items it holds and can hold.

- **Play**

  “Play” is a component, which sets the “DStack” in the correct mode, based on the selected data structure. It also makes sure all of the game objects within the play mode is shown when in play mode, and hidden when in use case mode.

- **RegionGrowAlgorithm**

  The class “RegionGrowAlgorithm” generates a list of actions needed to be performed to extract the connected region/pattern in the image based on a seed point within the image. It exposes methods for retrieving these actions and this enables the game to perform the algorithm one step at a time.

- **ImageHandler**

  The “ImageHandler” component, is responsible for creating a two dimensional array of “Pixel” prefabs to represent an image. It also generates a random pattern in this image, and exposes methods for retrieving individual pixels and for selecting them. The “Pixel” prefab contains a cube mesh and the “Pixel” component, which has a boolean value which controls weather this pixel is black or white. The “Pixel” component is one of the components the “EventHandler” component looks for when pointing the laser pointer, making pixels selectable/interactive.

- **DataStructure**

  The “DataStructure” component does alot of the same as the “DStack” component, except it is modified to make the animations fit this part of the scene and with a few additions.

- **ImageRep**

  "ImageRep" is a component which generates a representation of the state within the algorithm. It does so by generating a miniature version of the image, but without the pattern. The colors of each pixel represents the state of it, and it can be one of the following: unvisited(default), visited, seed, next, added to stack/queue, or part of patter. It also shows a legend of all the colors and their meaning.

- **PatternRep**

  The "PatterRep" component is similar to the "ImageRep" but in contrast only has two states: part of patter, or not. It shows which pixels are confirmed as part of a region or pattern this far in the algorithm.

- **UseCase**

  "UseCase" is a component, and it does the opposite of the "Play" component, by hiding its belonging game objects when in play mode, and showing them when in use case mode. However, it also contains a lot more. It has references to all the parts in this part of the scene, all mentioned above, and it contains the logic for how and when the user can interact with them.

The sorting algorithms scene is the most complex scene and these are the most important pieces:

- **SortingManager**

  The SortingManager is a component which basically glues everything together, and keeps track of and handles most of the logic for the flow of the game. Basically the big boss commanding all its minions. This class is however dependent on a lot of other smaller components. This class does not know *how* to do things, only *what* to do. Therefore it must have references to the components which actually know *how* to perform these actions or handle specific events. It does not only know *what* to do, it also knows *when this is allowed* to do. In other words, it keeps track of a bunch of states and when it is notified of some interaction happening, it will decide whether or not this is legal and act accordingly. Some references it keeps are :

  ∗ SortingAlgorithm - An algorithm which the array should be sorted after

  ∗ Arrays - Keeps track of all the arrays, and the logic for interacting with them

  ∗ Comparison - Handles compare actions

    - TitleManager - Updates the title of the room to the current algorithm
    - StateManager - Shows the updated state of the algorithms internals
    - ActionManager - Handles the state of the Action buttons (active or not)
    - AlgoControlManager - Handles the state of the buttons controlling the algorithm(Demo, prev, and next)
    - MessagesManager - Shows messages to the player about things that are happening

- **Selectable**

  "Selectable" is an abstract component, which is the parent class for all selectable game objects in this scene except for the buttons. The selectable class contains everything which is common for all these objects, namely that they can be selected, and a few other things. Current sub-classes are the "ArrayManager", "SortingElement", "EmptyElement", and "PartialArray" components. These can all be selected and performed actions on.

- **SortingElement**

  "SortingElement" is both a prefab, and component, which is also attached to this prefab. The prefab contains a cube mesh to represent the array element, text to show which element this is, and a "SortingElement" component. This component contains some information about the element which is useful or needed by other components when performing actions on these. It is also as mentioned a sub-class of "Selectable".

- **ArrayManager**

  The "ArrayManager" component used to be the main component in terms of keeping track of all the "SortingElement" game objects, animating their positions and sizes, and expose methods for performing these actions. It is now reduced to only keeping track of its belonging elements, and exposes methods for getting these elements. The new main component for the arrays and elements now, is "Arrays".

- **Arrays**

  "Arrays" is a component. It handles the logic of how the different actions affect the array(s), and last but not least which array it should perform the action on. In the beginning this component contained all the animations for the arrays and array elements too, but it got really big and I wanted to separate the animations of the different actions with the logic behind them. And hence the "ElementAnimator" was born. However, it turned out some of the logic still had to happen within the animation and are therefore not completely separate after all... Nevertheless, it turned out to be helpful in terms of navigating the code, when all the animation logic was in one file, while most of the logic was in the other.

The "Arrays" component is also an evolution from "ArrayManager". As I added the merge sort algorithm there was need for more than one array, and therefore the ArrayManager which was doing everything for the original array was not sufficient any more. Hence the "Arrays" component was created. It stole the logic from "ArrayManager" and added some more to handle several arrays. The "ArrayManager" was cut down to only keep track of its elements. So the history went like this:

  - ArrayManager
  - ArrayManager and Arrays
  - ArrayManager, Arrays and ElementAnimator

- **PartialArray and CombinedArray**

  The "PartialArray" and "CombinedArray" components, were added alongside "Arrays" during the implementation of merge sort. These contain a reference to a prefab which lets them spawn a new array cuboid with its own elements. And they are used when splitting an array in two halves, and merging them back together. They contain similar logic to "ArrayManager".

- **SortingAlgorithm**

  "SortingAlgorithm" is an abstract class, and defines a set of abstract methods which all sub-classes needs to implement, a set of virtual methods which already has a definition but can be overridden by the sub-classes, and a set of defined methods which will always be the same for all sub-classes. Having this abstract class makes adding new algorithms a lot faster, as a lot of the code for each algorithm would be the same. It also makes the addition of new algorithms less error prone, as the core is already there for each algorithm and only what is specific for the algorithm needs to be added. The abstract methods are the "GenerateActions" method, and the "CheckForFocusChange" method. "GenerateActions" return Void and takes no parameters. It is called in the constructor and the intention of this method is to add all the "GameActions" needed to sort the array in the correct order and with the correct indexes to the actions list, as well as add an equal number of states to the states list to let the user peek inside the state of the algorithm. The "CheckForFocusChange" method should trigger an event in the "EventManager" when the algorithm reaches a state where there is a change in which elements are in focus.

A lot of the components explained above mention animations, and that they keep definitions of animations, and ways to trigger them. However, none explain how they are defined and how they work. The animations were coded, and the way the animations were coded was by using coroutines. Couroutines are a part of the “UnityEngine” library, and they work by defining a method with a return type of “IEnumerator”, and should contain the code for doing the animation, i.e move, rotate, and or scale objects. And to start the defined animation one needs to call the “StartCoroutine” method and provide the animation function as a parameter to that function. Like this:

```
IEnumerator SomeAnimation(){
    float duration = .6f;
    float elapsed = 0f;
    float prevTime = Time.time;
    //This while loop will run one iteration of the loop every
    frame for .6 seconds. This way of writing the coroutine makes
    the animation's duration independent of the frame-rate
    while(elapsed < duration){
        // Do something every frame here
        ...
        float time = Time.time;
        elapsed += time - prevTime;
        prevTime = time;
        yield return null;
    }
    //Do something at the end of the animation here
    ...
}

...
// Somewere inside a function which should trigger the
    SomeAnimation animation
StartCoroutine(SomeAnimation());
```

The special part about these coroutines is that is executes code until it reaches a “yield return ...” statement, then it will stop executing, and then continue from where it left off at the next frame. There is no limit on the amount of “yield return ...” statements a coroutine can have. the value after “yield return” can be either null or one of several different types for waiting specific times until resuming execution, like a set amount of time in seconds, or at the end of the frame instead of the beginning of the frame, and so on.

3.3 Evaluation

The evaluation in this project will be fairly similar to the form of evaluation performed in the previous project, whose form was briefly touched upon in the “Previous Work” section in the previous chapter. There will be a user test and a questionnaire for the participants to fill out afterwards and the evaluation will be based on those. The user tests will start with a short explanation about what the game is, and its intentions. It will give the participants some context in the form of what they can expect in terms of genre and setting. After the introduction of the game, they will play the game, or in other words perform the user test. No one will give guidance to them unless they explicitly ask for help. If a participant asks for help, it is a sign that some part of the game is not intuitive or something should be better explained within the game. The participants will be informed of this, to increase the threshold for asking for help. After the participants feel they have done enough, they will be asked to fill out the questionnaire. This questionnaire will contain a section regarding some background information of the participant, a part with the standard SUS questions for getting a usability score, and some general feedback regarding the game and their experience with it. The targets for this testing will be people who has taken or is taking the “Algorithms and data structures” course, as well as people who has not taken the course but has some experience and knowledge of programming. The level of VR experience a participant has is not of much importance, though a varied spectrum could be nice. This would open up the possibility to check for a connection between previous experience with VR and their understanding of the prototype. As already mentioned the previous project was only able to perform the testing with five people. This time around, the aim is to at least perform tests with double the amount of people.

4 Results

This section will present the results, in regards to development and evaluation. The part about development will briefly explain how the scenes turned out. The evaluation part will present the test results without any interpretation, just numbers. The interpretation of the test results will come in the next chapter: Discussion.

4.1 The Prototype

The development was halted in mid May, 2019, to make time for the user testing and writing the report. The final version of the prototype can be seen in this video[5]. It shows how the prototype works, how it looks, and a general look into all its content. For more details the code can be found on Github[4].

4.1.1 Overworld

The overworld is the starting point of the game. This scene is just a plain with a wall on top of it, and some hovering text, showing the name of the prototype; “UniVRsity”. The wall contains buttons for all the courses in the prototype, which is at this point in time only one; “Algorithms and data structures”. The overworld can be seen in the video and in Figure 8. Pressing a course button by pointing on it with the laser pointer and pressing the trigger button on the controller will make the game transition to the respective course’s course hub.

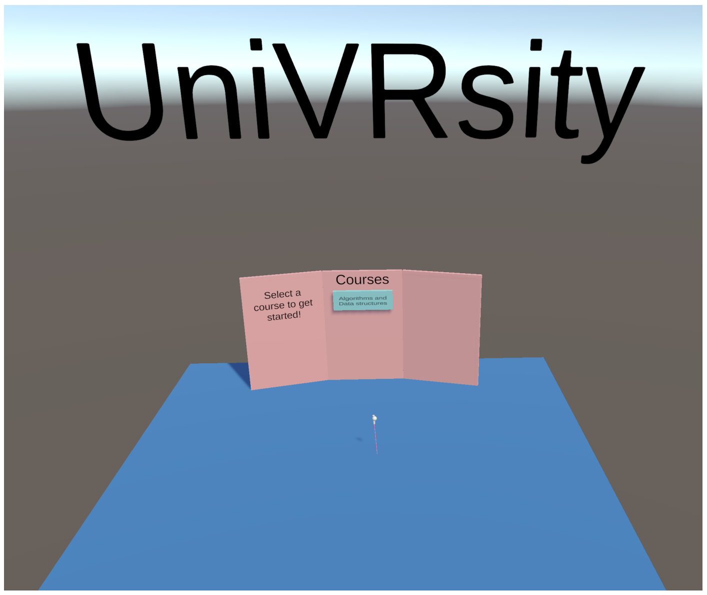

UniVRsity
Courses
Select a course to get started!
Algorithms and Data structures

Figure 8: The overworld.

4.1.2 Course Hub for Algorithms and Data Structures

The course hub for the algorithms and data structures course is very similar to the overworld scene. It has a wall with buttons for each topic, which will when pressed transition the game to the respective topic's scene, and a button for going back to the overworld scene. The topics available are "Data Structures" and "Sorting Algorithms". Figure 9 show this scene.

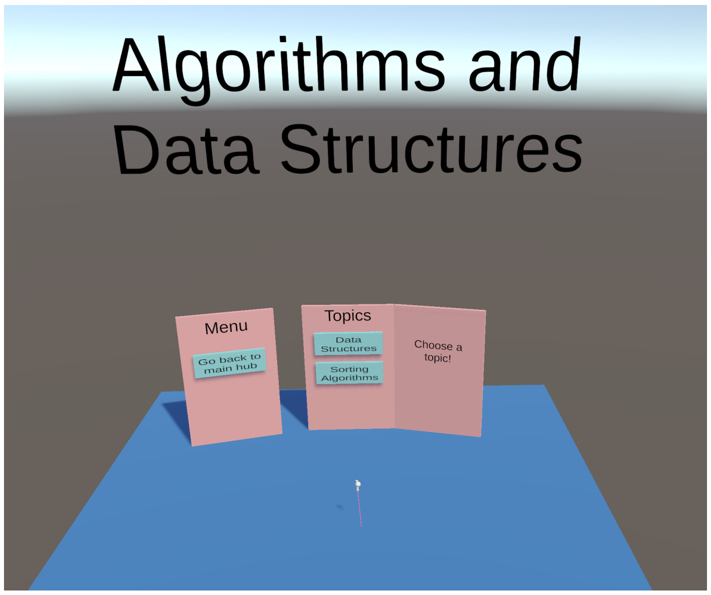


Figure 9: The algorithms and data structures course hub.

4.1.3 Data Structures

The data structures scene is split into two parts. These are the play mode and the use case mode, and can be seen in Figure 10, Figure 11, Figure 12, and Figure 13. This scene starts out in the play mode. In this mode the player can choose between the stack and the queue structures, and then play around with them to get a visual representation of what happens when data is added or removed to the selected structure. The use case mode is where the player is presented with a proper use case where the selected structure is used as a part of the solution to the problem. For both the stack and the queue the use case is an image region growing algorithm.

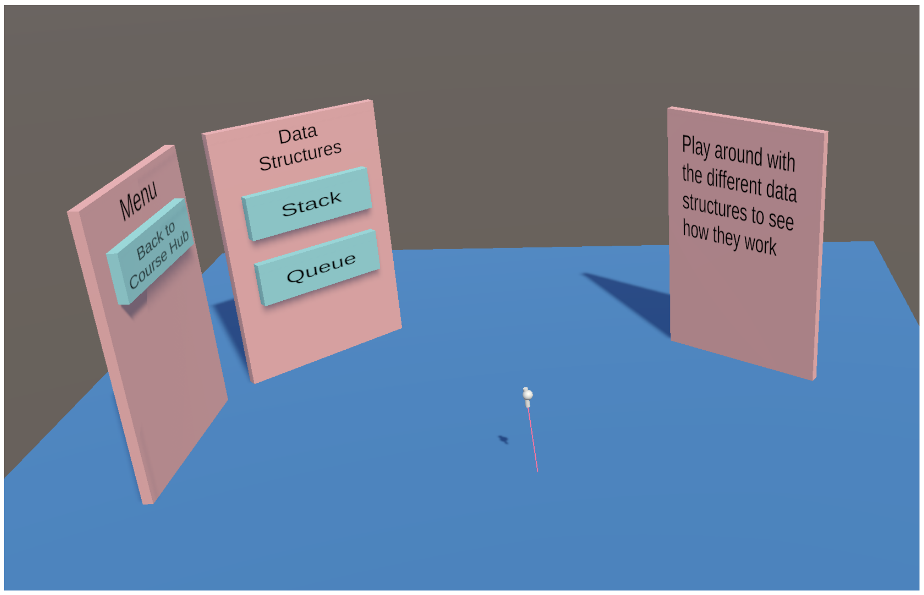

Menu
Back to Course Hub
Data Structures
Stack
Queue
Play around with the different data structures to see how they work

Figure 10: The play mode, before selecting a data structure.

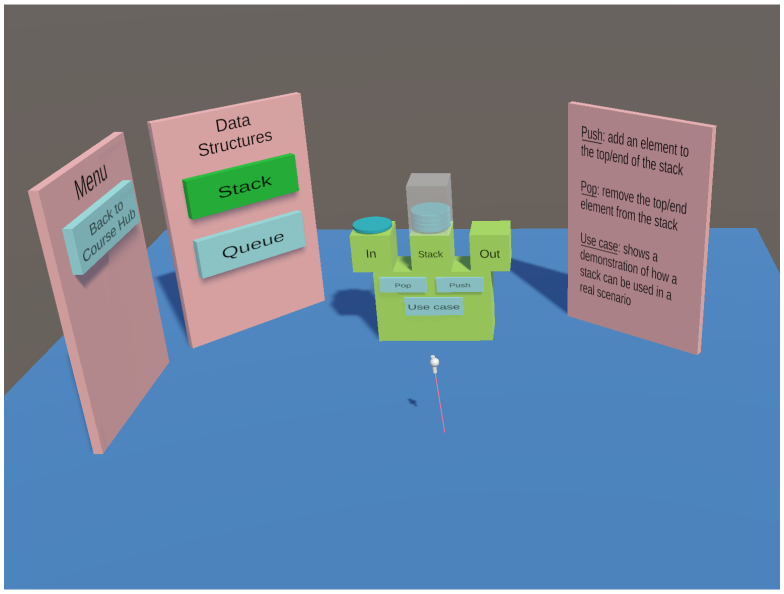
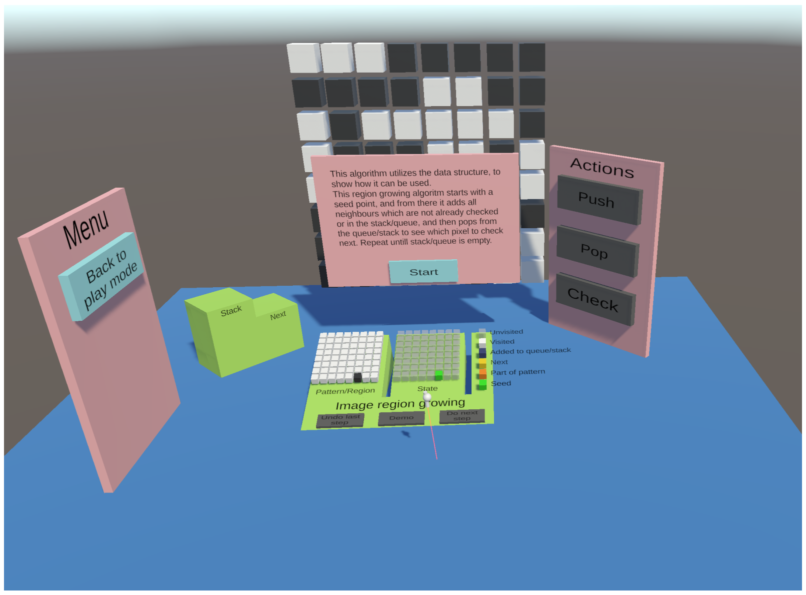

Menu
Back to Course Hub
Data Structures
Stack
Queue
In
Stack
Out
Pop
Push
Use case
Push: add an element to the top/end of the stack
Pop: remove the top/end element from the stack
Use case: shows a demonstration of how a stack can be used in a real scenario

Figure 11: The play mode, after selecting the stack structure and pushing a few elements to it.



This algorithm utilizes the data structure, to show how it can be used.
This region growing algoritm starts with a seed point, and from there it adds all neighbours which are not already checked or in the stack/queue, and then pops from the queue/stack to see which pixel to check next. Repeat untill stack/queue is empty.

Figure 12: The initial state of the use case mode.

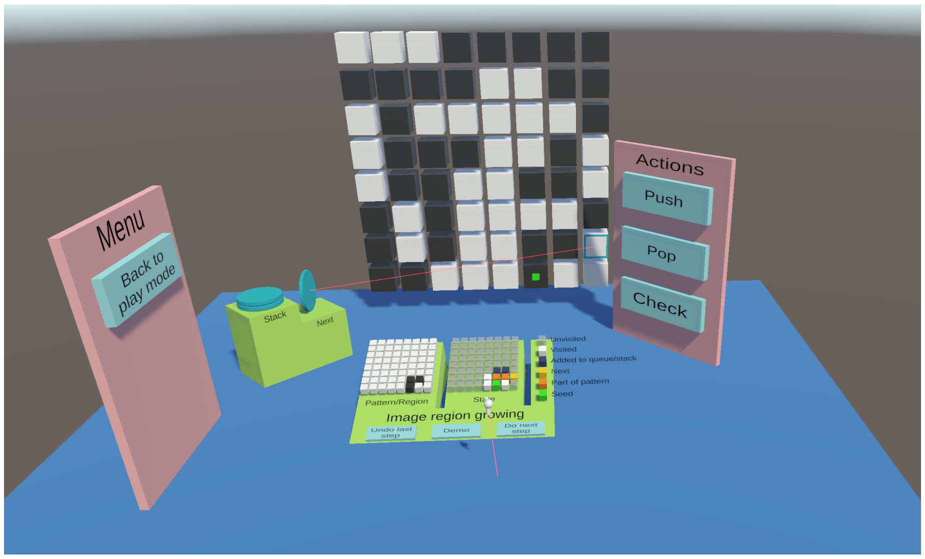
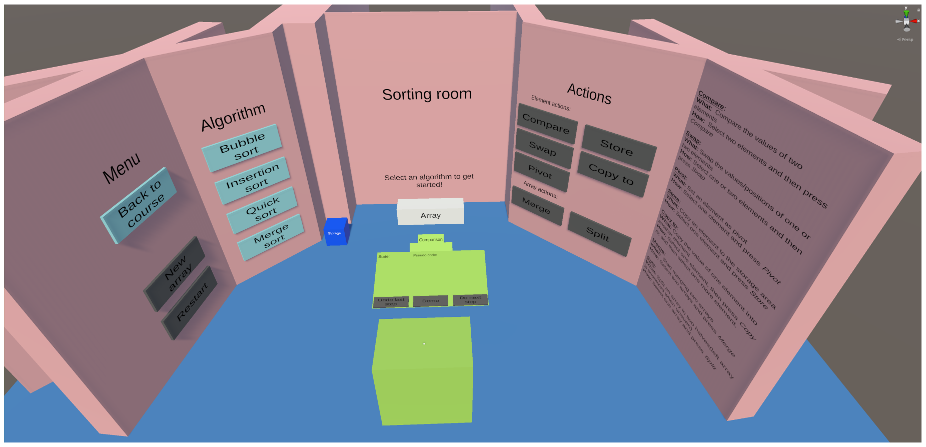


Figure 13: The use case mode after having done a few steps.

4.1.4 Sorting Algorithms

The sorting algorithms scene has kept the design of the previous prototype, with a few alterations. The player is now elevated by standing on a cube, to get a better view of everything in the room, and the demo button is moved from the menu wall to the algorithm panel in front and below the player, because connected things should be physically close to each other. Besides this, the room still has the same shape, and most parts are where they were previously. The content in this scene is expanded by adding more algorithms and actions, and text to explain the actions on the wall next to them . The scene can be seen in different states in Figure 14 and Figure 15.



Compare:
What: Compare the values of two elements
How: Select two elements and then press *Compare*
Swap:
What: Swap the values/positions of one or two elements
How: Select one or two elements and then press *Swap*
Pivot:
What: Set an element as pivot
How: Select one olement and press *Pivot*
Store:
What: Copy an element to the storage area
How: Select one element and press *Store*
Copy to:
What: Copy the value of one element into another element
How: Select one element, then press *Copy to*, and then select one more element.
Merge:
What: Start merging two arrays
How: Select two arrays and press *Merge*
Split:
What: Split an array in two halves(left array is prioritized when uneven)
How: Select one array and press *Split*

Figure 14: The initial state of the sorting algorithm scene.

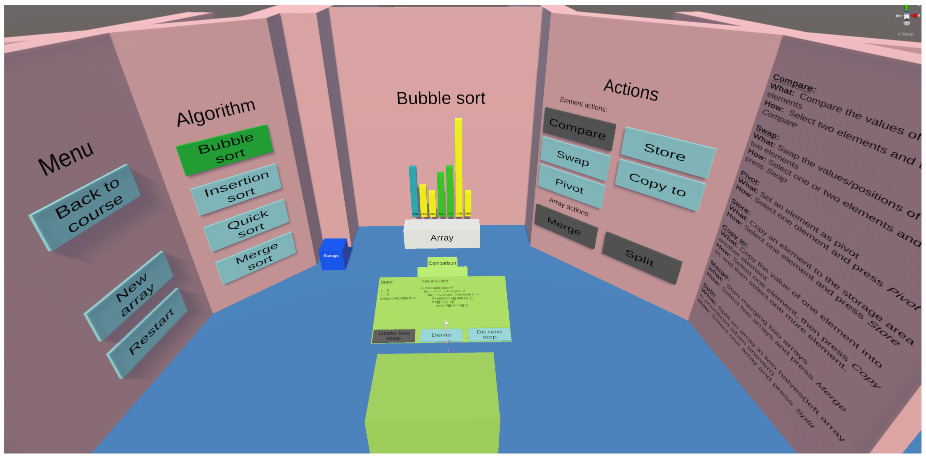


Figure 15: The sorting algorithm scene, having selected the bubble sort algorithm and one of the array elements. Blue/turquoise elements or arrays are selected, yellow are not selected/normal, and green means that the element in this position has the value/size it should have when the sorting is complete.

4.2 Evaluation

The user testing was performed in mid May, and 9 people participated. As planned, the participants were given a short description of the game and its intentions to give them some context. They were also informed that they may ask questions if they get stuck, but refrain from doing this needlessly. After the explanation they were asked to equip the VR headset hand the controller, and play the game(user test). Then they played until they felt that they had explored everything, or until they felt they had seen enough. Finally they were asked to fill out a google form. And during the last step some of the participants gave some oral feedback as well. This was all according to the plan presented in the method section.

4.2.1 Participants

Let us see who the participants were. According to their answers in the google form, all of them were students, and all but one had previously taken the course “Algorithms and data structures”. The majority of the participants had tried VR earlier, but were not very experienced with it. The details can be seen in Figure 16, Figure 17 and Figure 18.

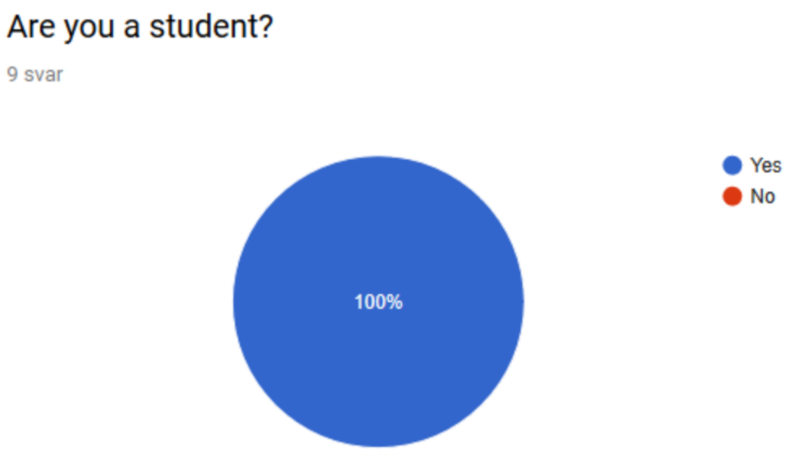


Figure 16: The share of the participants currently being students.

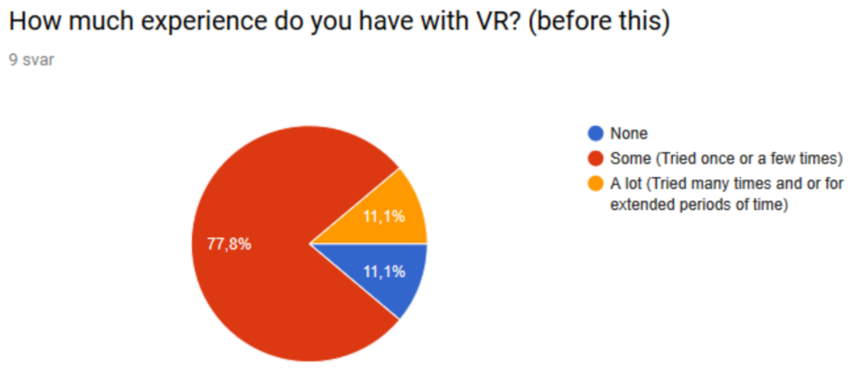


Figure 17: The amount of experience the participants had with using VR, prior to the user testing.

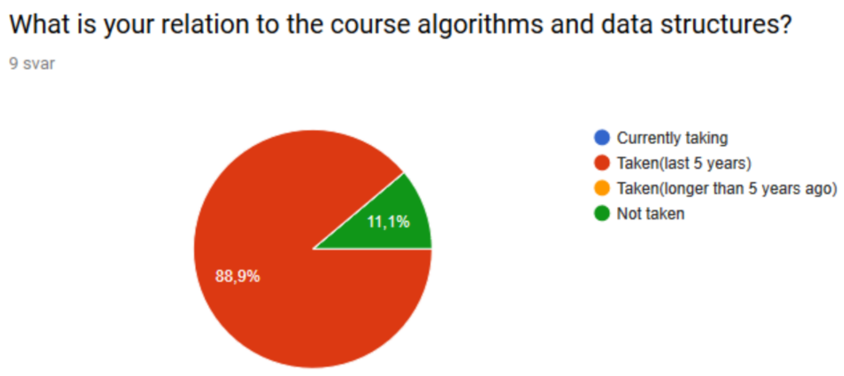


Figure 18: The relation between the participants and the course "Algorithms and data structures".

4.2.2 Usability

The results of the SUS gave an average score of 70.3. The average SUS score in general is 68, which means that the usability of this prototype scored above average. The participant which had not taken the course in the prototype gave a score of 85 which is the next highest score given. Only including the participants which had taken the course, made the average score drop to 68.4, which is only slightly above average. This is the almost the opposite result of the testing in the previous project. In that project the lowest score came from the participants not having taken the course, and the average rose when excluding these.

4.2.3 Discomfort

VR is infamous for its tendency to cause motion sickness in players, and some of the participants said they were experiencing this during the user testing. In fact three out of the nine participants said that they felt some degree of motion sickness, which is 33.3%. Some participants also felt other discomforts such as the game making them feel dumb, or due to other reasons. The details can be seen in Figure 19.

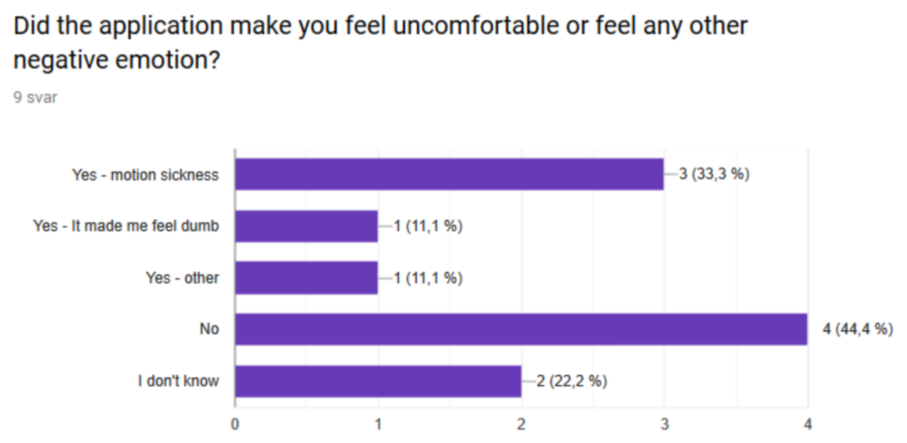


Figure 19: The response from the participants in regards to negative emotions felt during the user testing.

4.2.4 Thoughts and General Feedback

The thoughts of the participants in regards to using VR for teaching/learning were very positive, and their experience with the prototype was for the majority good. In fact everyone thought that VR in general could be useful for learning! Almost all thought that an application like the prototype also could be useful. The answers can be seen in Figure 20, Figure 21 and Figure 22.

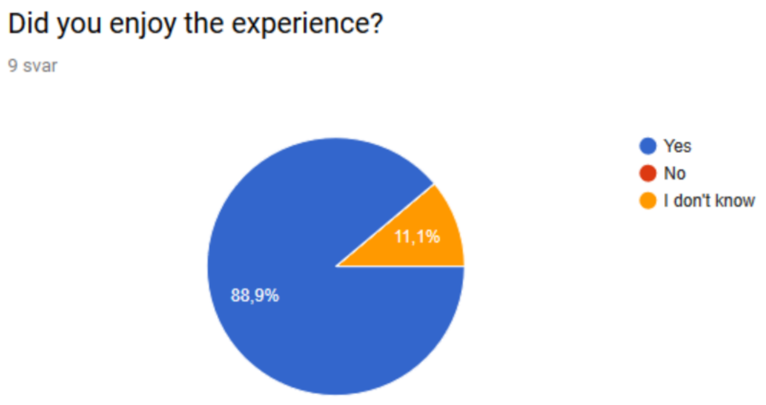


Figure 20: Whether or not the participants enjoyed the experience of playing “UniVRsity”.

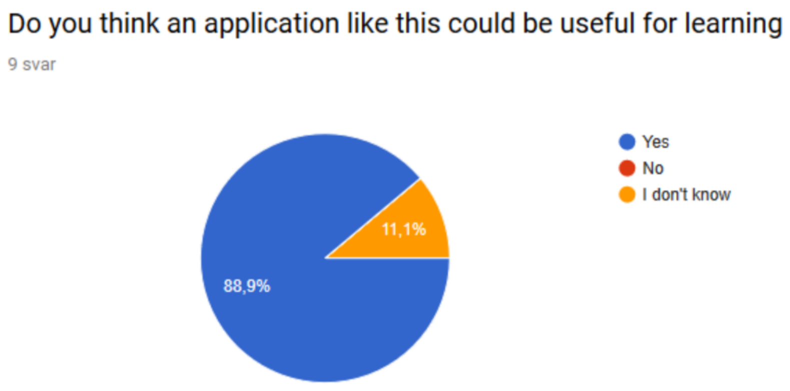


Figure 21: Whether or not the participants thought that “UniVRsity” could be useful for teaching/learning.

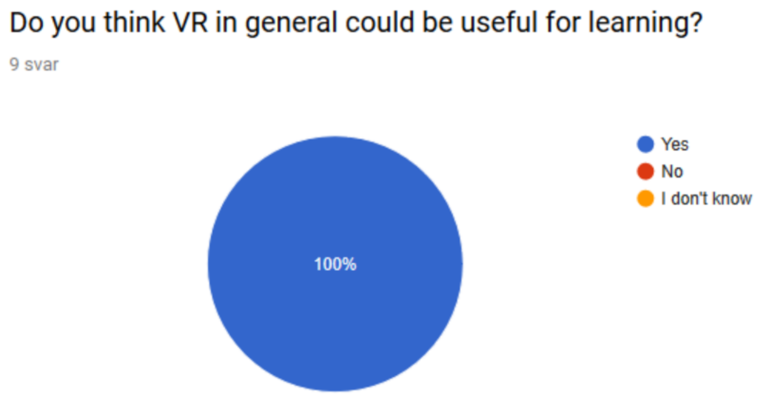


Figure 22: Whether or not the participants thought VR could be useful for teaching/learning in general.

The general feedback given was that it was not clear that the user could do the steps in the algorithms manually. Most only used the “Demo” or “Do next step” buttons when playing the game. Some also said that a lot of things happened simultaneously and it was hard to know where to focus, specifically for the use case in the data structures scene.

5 Discussion

This chapter will process the information and experience gathered and gained throughout the development and testing, and reflect around them. What was done well? What could have been done better? What have I learned? Does the results make sense? Is the data gathered sufficient for reaching a conclusion? Questions like these are what you will find reflections around in this chapter. Starting with the research questions, next the process and planning, then moving on to the development, the testing after that, and finally the results.

5.1 Summary

To sum up the results of this project, I would like to address the research questions proposed in the introduction.

5.1.1 RQ1

"Is the new prototype improved compared to the previous version?". Yes, the prototypes is improved somewhat. It has improved in terms of usability according to the SUS score, with a bigger sample group, though not by a lot. The lesson learned here is that there should *always* be a tutorial. There was also added quite a bit of new content to the prototype. The new content was of very similar form to the existing content and was therefor not a bottleneck in terms of usability. There is no empiric data to show whether the new prototype is better or worse at teaching players, or whether or not it is more fun to use.

5.1.2 RQ2

"Do people think VR is suited in a school or learning setting?". The short answer here is: yes. All the participants of the testing thinks VR is suited in a learning setting. This might be a bit biased as all the participants study technology, but their thoughts should not be ignored either. I think that VR will be more utilized for learning in the future.

5.1.3 RQ3

"Will the developed prototype indicate that using VR might be suited for teaching/learning?". Despite some usability issues, most of the participants of the testing though that an application similar to the prototype could aid in the learning process. However, not quite as many as those who thought VR in general could be useful for learning, suggesting this game should have done something differently.

5.2 Planning and Process

The planning and process was not done in a very structured manner. The upfront planning was minimal, and focused more on just making something and then see where to go from there. It was kind of like being on a journey through nature, and scouting a general direction to walk whenever at a peak or high point, then just stumbling along in this direction until stopping at a new point and repeating this process. The biggest obstacles along these paths were the design choices. “How should I represent this in VR?” or “What parts of this subject is important to emphasise?” and so on. The peaks and high points, were the meetings with my supervisor, where we discussed what had been done, and find the general direction to walk until the next meeting. These meetings with my supervisor were scheduled bi-weekly, with a few exceptions, and they gave me a sense of direction and also some sense of urgency to complete certain tasks, or reach certain milestones. Despite this, there were periods with low productivity due to lack of motivation. Especially in the beginning. I was struggling with a lot of design issues, and how I wanted things to work, and this killed my motivation for quite some time. I would get things done, but not anything beyond that. It was not until late March or early April, when I finally picked up the pace and found some motivation again. I think that this project would have benefited hugely by having a member dedicated to designing the game(concepts and functionality), and me doing the code, because coding is something I am a lot more passionate about. It would also benefit in general by having more people on board, as making a game requires many very different different skills. Games are a combination of graphics, game design, sound, music, code, and more. Doing all of this as one person is really hard.

5.3 Development

As an after thought, the way I handled animations was probably not the most efficient way. And by efficient, I mean efficient in terms of creating, not in terms of execution and performance. The animations were all done by code, which is kind of waste when there exists a tool for creating animations in the Unity editor. Using the tool in Unity allows for a visual generation of animations, which is a lot easier than sketching or imagining the different positions and rotations an object should have throughout the animation. In the beginning of the project some experimentation was done with the animator tool, but, as a coder, it felt a bit weird. The tool seemed fairly easy to use, at least for objects already in the scene. The problems began when I needed the animation to spawn different objects and then do animations on them. I will admit that not a lot of effort was put in to figuring out this, and instead decided to settle for writing the animations with code, because I already knew how to do it this way.

A part of the architecture or structure of the game which was not done optimally, was also linked to the animations. A lot of the animations contained logic in them, which made the scene reliant on the animations completing properly for the game to behave properly. This is bad design, and was the cause for a lot of the bugs having to be dealt with at the end of the development. Removing the logic from the animations would require some components to keep track of more states, but the payoff would be huge in terms of clean code, and less error prone code.

During the development, I always had performance in mind, as VR games need to run smoothly in order to decrease the strain on the user in terms of getting motion sickness. This made me very hesitant of using the inbuilt "Update" method of all the components. This method runs every frame for every component in the scene, and therefore means that doing a lot of work in these methods is a bad idea. However, The scenes in this prototype does not contain a lot of components compared to a lot of other games, and I overestimated the performance hit of using this method for small things. The work-arounds also caused a lot time loss due to putting logic which could have gone into the "Update" method in the "Start" or "Awake" methods instead. This caused a lot of null pointer exceptions due methods being called on instances not yet created, due to the nature of order of creation in scene elements. It also made debugging harder as the things happening is not just tied to the update method, but instead tied to many methods all over the place. This also makes it harder for new people to read and understand the flow of the code. I was even hesitant to use the "Update" method in combination with a boolean which it would check before doing any actual work, because I thought that the act of checking this boolean every frame could impact performance. The reality is that it will not affect the performance, unless there are a vast amount of components. The only component, or one of very few, I created which utilized this method was the "EventManager". This component used to update method to shoot a ray cast from the laser every frame to decide the length of the laser and also to interact with what it was pointed towards. The point here is that using the "Update" method would have simplified a lot of things, made the code cleaner and easier to read, and less buggy. Lesson learned.

5.4 Testing

The user testing was supposed to have started in late April or early May, but ended up happening mid May. The reason for this was a mixture of things. I was attending a confirmation of my cousin in Stavanger, and due to the SAS-strike I had to travel earlier than needed, and I got home later than planned. During this period I was not able to work too much on the project. This was late in April. Another reason was that I had a realization when I was showing VR to a friend and she wanted to see what I was making. I realized that I had not really addressed the usability issues found in the previous project and I had come up with a few other tweaks I could make to improve the usability as well.

And some of these tweaks took a lot longer to implement than I had thought. The start of May went into this part(subsubsection 3.1.6). And since quite a few bugs were found in the last project's user testing, I thoroughly tested the application in ways it was not meant to be used. This unveiled many bugs which I had not noticed earlier because I, the creator, know how the application is supposed to work and use it thereafter when testing new features. However, some of these bugs were surprisingly hard to squish and took quite a lot of time to fix, but As the user testing went smoothly around this time, it seems it was not in vain. No bugs were encountered during the user testing performed of the new prototype.

In regards to the participants of the user testing, they were all acquaintances or friends of mine. This means there might be some bias in the answers. The prototype was about learning a computer science course, and therefore people within this field of study is preferred for testing. Since I am also part of this field of study, it was always a risk of this happening. Nevertheless, this does not mean the results are useless, just that they need to be judged more critically. Having the answers collected be through an anonymous questionnaire, might mitigate some of the bias which could have been stronger if the answers were gathered through an interview. The quantity of participants was not very large, and could have gotten more accurate results if it had been bigger. However, the SUS does not require a large sample to be accurate, which is a saving grace for this project.

5.5 Results

This bit will discuss both the developed prototype and the results of the user tests. The prototype will be discussed in terms of what about it is backed by theory, what is done in spite of theory, and what could have been done differently. The testing will be discussed based on the different metrics gathered and what they could indicate.

5.5.1 Prototype

As suggested by the the multimedia learning theories, the use of voice is suggested to replace text(subsection 2.3). This was known in advance of the development, but despite this, there is no sound in the game. This means that the input load will only be handled by the visual "channel" instead of spread across the auditory and visual "channels". The addition of adding voice had long been a part of the plan, but it was never actually done. I guess the reason for putting it off for so long was that I did not have proper recording gear easily available to me, and that I did not really want my voice to be in the game. I considered using an artificial voice(text-to-speech software) instead, but this was advised against from the theory.

5.5.2 Usability

The SUS questionnaire yielded an average score of 70.3 as presented in the results chapter. This means that the usability of this prototype is slightly higher than that of the average application, assuming there is no bias present in the score. The visual observations made during the user testing seem to indicate that bias is present and that the score from the questionnaire got a higher score than it should have. Even just a single point lower score in one of the ten statements from all the participants would drag the score down to slightly below average(70.3 - 2.5 = 67.8). Only few of the participants were able to figure out that algorithms could be done manually and this is a pretty huge flaw. Only being able to watch the algorithms is not quite the same as actively participating, even though the application does not require the user to do things manually. Some might find it enough to just watch to learn the algorithm, but I believe there is merit to "learning by doing".

5.5.3 Usefulness

100% of all the participants, thinks VR could be helpful for teaching/learning. The sample is not huge, but it makes sense. The quality of the VR hardware today makes VR very immersive, and immersion is on some level very similar to focus, and focus is needed when learning. It also gives the ability to go anywhere or do anything, which is not possible in a normal classroom setting. The response from this question is therefor not a surprise at all. Most of the participants even thought that an application similar to the prototype could be helpful, and the prototype was not even that easy to use, showing just how much they believe in VR.

5.5.4 Motion Sickness

There are side effects present in the current hardware of VR, and some are more susceptible than others. The side effect I am referring to is of course the infamous motion sickness. The root of motion sickness comes from the brain receiving different signals of motion from different senses. Which is why one can easily get sick when reading a book or playing a game in the car; the eyes detect no motion, but the inner ear does detect motion[13]. It is said that the games where players are more prone to getting sick are games with locomotion and games with low frame rate. Avoiding these two will reduce the chance of players getting sick. Popular replacements for locomotion is teleportation, and in games where the player can use the joy stick on the controller to turn the camera, having it turn in chunks in stead of continuously is also preferred by most. The prototype does not contain locomotion nor camera control through the joystick, and the frame rate was not limited by the software, as far as I am aware. Despite this, about a third of the participants were somewhat affected by this. The triggers for this based on their feedback seemed to be from the sorting area, where the player is standing on a box to get some more height and a better view of everything, and from the sudden change in the environment when changing from play mode and use case mode in the data structure scene. The box in the sorting scene could not always be seen, depending on how far forward the player was standing. Not seeing the box one was standing on made it seem like the player was twice as tall or standing on a cliff. This was not optimal for avoiding sickness. The sudden change in the data structure scene were experienced as too intense by some of the participants.

5.6 Reflections

A possible solution for the problem of players not figuring out that algorithms can be performed manually, is to implement some form of optional tutorial, where the player will be presented with interactive explanations for everything possible to do within a scene. Having it be optional is important to avoid boring the players which already knows how things work.

Based on some of the feedback given after the user testing, it was explained that it was hard to understand what was happening in the use case scenario for the queue and stack. This was due to many things happening at once and there was no indication as to what action the user had just made. A suggestion from one of the testers for making this more clear was to highlight the most recent action in the state representation. I agree, this was a good suggestion.

Another thing several of the testers suggested was to show which line of code was being executed at the moment, making it easier to follow the sorting algorithms. This is a feature which I knew the other person doing the same task as me in parallel had implemented, and was one I had considered. In hindsight I wish I had implemented this, as it is kind of an obvious winner in terms of making things clear. However, for some reason I did not realize that this was needed, in time.

6 Conclusion and Future Work

We have reached the end. This section will conclude this article as well as give suggestions for further work.

6.1 Conclusion

If we zoom out a little bit, it becomes clear that the prototype developed in this project is not really useful yet, in its current form. The syllabus coverage even for the only course within is minimal. For it to become properly useful, it would need a lot more content, and the content would need to be good. And even before that point it is not certain that this prototype will help students learn more or more easily, before it is tested in terms of learning. Therefore, what has been done here in this project is only useful for someone which seeks to develop something similar or perform tests on or with something like this. For those who wishes to develop something similar, they can use the current prototype as a starting point, instead of starting from scratch. This could save them time, but it also kind of locks them in to a similar style, unless they do major modifications to the existing bits. For those wishing to further test VR in a learning setting, this prototype could also be used, saving time in terms of development. Though, I would recommend adding tutorials before doing further tests with this prototype. All in all, it has been a good experience for me personally even if it did not contribute anything to “the bigger picture”.

6.2 Future Work

There is a lot of things that can be done to improve this prototype and there are a lot of thing that can be done to further evaluate it. A good place to start for the prototype would be to add sound and voice to it, as well as a tutorial for all the topics. This would increase its usability as well as make it more entertaining to use. Possibly improving the effectiveness of the teaching as well, should the theory be correct. Other parts of the prototype which could be worked more on is adding more content. More topics to the current course, or new courses and topics for them.

For further evaluating the prototype, it could be interesting to see if it aids in the learning process. To do this one could team up with the one responsible for the course, and have some of the students taking the course use the prototype as a part of their course work. Then one could do a survey before the course started, or the specific topic evaluating for, to see how much knowledge the students had before starting. This would have to be done by both the students with the prototype included in their course work as well as those without it, such that the results can be measured against each other. There should also be two more surveys, one to check the knowledge and understanding right after the playing of the prototype or after the topic, to see what was learned in this period, and finally one at a later point to check for retained learning. Then compare the results of the group with the prototype in their course work against those with the normal course work, and see which group had higher learning. It would also be possible, and maybe even a good idea, to have three groups; normal course work, only prototype, and both.

References

[1] Brackeys. https://www.youtube.com/user/Brackeys.

[2] Fusedvr. https://www.youtube.com/channel/UCLO98KHpNx6JwsdnH04l9yQ.

[3] Sindre Aarstand. Univrsity(old). https://youtu.be/7m7B7zJ4KIQ. [Online; accessed 9-Dec-2018].

[4] Sindre Aarstrand. Univrsity. https://github.com/Zindre17/UniVRsity. [Online; accessed 15-Oct-2018].

[5] Sindre Aarstrand. Univrsity. https://youtu.be/VUHdUxoRH5Y. [Online; accessed 05-Jun-2019].

[6] Sindre Aarstrand. Univrsity - build. https://drive.google.com/open?id=1PkUTAYxPQBEKAE8IhJiY5-u_aZyCz_9P. [Online; accessed 31-May-2019].

[7] Valve Corporation. Steamvr unity plugin. https://valvesoftware.github.io/steamvr_unity_plugin/, https://github.com/ValveSoftware/steamvr_unity_plugin. [Online; accessed 20-May-2019].

[8] Mihaly Csikszentmihalyi. Flow : the psychology of optimal experience, 2008(1990).

[9] Tom Xiang-Kun Kong and Anders Marstein Kruke. Teaching computer science algorithms through virtual reality, 2018.

[10] Nicole Lazzaro. The four fun keys-chapter 20. In *Game Usability*, pages 317–343.

[11] Richard E. Mayer. *Multimedia Learning.*, volume 2nd ed. Cambridge University Press, 2009.

[12] Richard E. Mayer. *Cognitive Theory of Multimedia Learning*, page 43–71. Cambridge Handbooks in Psychology. Cambridge University Press, 2 edition, 2014.

[13] U.S. Department of Health and Human Services. Motion sickness. https://medlineplus.gov/motionsickness.html. [Online; accessed 30-May-2019].

[14] Jeff Sauro. Measuring usablitiy with the system usability scale(sus). https://measuringu.com/sus/, 2011. [Online; accessed 21-May-2019].

[15] Penelope Sweetser and Peta Wyeth. Gameflow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.

[16] John Sweller. Cognitive load theory, 2011.

[17] Unity Technologies. Collaborate. https://unity3d.com/unity/features/collaborate. [Online; accessed 21-May-2019].

[18] Unknown. System usability scale(sus). https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html. [Online; accessed 21-May-2019].

NTNU
Norwegian University of
Science and Technology