NTNU
Norwegian University of
Science and Technology

Teaching Computer Science Algorithms Through Virtual Reality

Tom Xiang-Kun Kong
Anders Marstein Kruke

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Frank Lindseth, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Virtual reality has in recent years become exceedingly popular since devices of high enough quality have become available at an affordable price, starting with the Oculus Rift in 2016.

This thesis explores how virtual reality can be used to teach complex concepts for educational purposes. Specifically, it explores how virtual reality can be used in the course TDT4120 – Algorithms and Data Structures at NTNU to teach students the algorithms presented in there.

This was done by implementing a virtual reality application in the Unity game engine. The application tries to teach the students three different sorting algorithms — bubble sort, insertion sort and quicksort — by interacting with an array of boxes. Several helper functionalities were implemented as well, such as highlighting and color coding to assist the user, as well as instructional videos for each algorithm. Videos demonstrating the application can be found at `https://tinyurl.com/VirtSort`.

The application was evaluated based on user testing, where it was tested by NTNU students, both with and without experience from the TDT4120 course. The testing revealed that the application still has some way to go before being a viable learning tool. This can to some degree be attributed to users being somewhat inexperienced with using VR and still requiring too much prior knowledge of the algorithms to use the application efficiently.

Despite this, the results indicate that the problem is not with the concept itself, but rather the application missing some features. Overall, the conclusion leaves a positive impression of the usage of VR to teach algorithms, as well as other complex concepts.

Abstrakt

”Virtual reality”, eller virtuell virkelighet, har de siste årene blitt mer og mer populært etter at teknologi av høy nok kvalitet har blitt tilgjengelig til en rimelig pris, noe som startet med Oculus Rift i 2016.

Denne tesen utforsker hvordan virtual reality kan brukes for å lære bort komplekse konsepter for utdanningsformål. Mer spesifikt undersøker den hvordan virtual reality kan brukes innen emnet TDT4120 – Algoritmer og datastrukturer på NTNU for å lære studentene algoritmene som presenteres der.

Dette ble gjort ved å implementere en virtual reality-applikasjon i spillmotoren Unity. Applikasjonen prøver å lære studenter tre forskjellige algoritmer — bubble sort, insertion sort og quicksort — ved å interagere med en rekke (”*array*”), av bokser. Flere hjelpefunksjoner ble også implementert, blant annet utheving (*”highlighting”*) og fargekoding for å hjelpe brukeren, i tillegg til intruksjonsvideoer for hver algoritme. Videoer som demonstrerer applikasjonen finnes på `https://tinyurl.com/VirtSort`.

Applikasjonen ble evaluert basert på brukertesting, hvor den ble testet av studenter ved NTNU, både med og uten erfaring fra emnet TDT4120. Testingen avslørte at applikasjonen fortsatt har et stykke å gå før den kan vurderes som et fullverdig læringsverktøy. Dette kan til en viss grad tilskrives at brukerne for det meste er uarfarne i bruken av VR og at de er avhengig av for mye forhåndskunnskap om algoritmene for å bruke applikasjonen på en effektiv måte.

Til tross for dette indikerer resultatene på at problemet ikke er konseptet i seg selv, men i stedet manglende funskjonalitet i applikasjonen. Totalt sett etterlater konklusjonen et positivt inntrykk av bruken av VR for å lære bort algoritmer, i tillegg til andre komplekse konsepter.

Table of Contents

List of Tables

List of Figures

Listings

Abbreviations

API	=	Application Programming Interface
DFS	=	Depth-First Search
DoF	=	Degrees of Freedom
FoV	=	Field of View
HMD	=	Head-Mounted Display
IDE	=	Integrated Development Environment
OLED	=	Organic Light Emitting Diode
PSVR	=	Playstation VR
UI	=	User Interface
VR	=	Virtual Reality

Chapter 1

Introduction

2018 is an exciting time to live in as the fictional world is more real than ever before thanks to the advancements in virtual reality technology. As the technology matures and the mainstream adopts VR, there is little that indicates that VR is not here to stay. It is however, still a relatively young field, and while many actors focus on entertainment in VR, the potential to use VR for virtually anything means that it can have an impact in almost all fields of research, even education.

1.1 Motivation

An important aspect of computer science related courses at NTNU, such as Computer Science (MTDT) [5], Informatics (BIT) [1] and Communication Technology (MTKOM) [3] is to gain knowledge about computer algorithms and data structures. The course TDT4120 - Algorithms and Data Structures [27] is responsible for introducing these concepts to students, with the intention to teach basic theory about algorithms and data structures as well as provide examples of these. Many students find it difficult to understand the many concepts taught in this course. As these students advance to other courses that require TDT4120, they struggle to keep up with the new material, which results in an endless spiral of more gaps in their knowledge. Therefore it is important that as many students as possible fully understand the concepts, methods, algorithms and data structures taught in the course.

1.1.1 Using New Technology to Teach Difficult Subjects to Students

While the new students are becoming more and more technologically literate as the years go by, many aspects of the educational process is still done in a very traditional way, often still using pen and paper methods even for courses directly working with technology. While there are certain logistical reasons a programming exam might still be done using handwritten code, many courses still do not embrace technology, and tend to approach teaching in a more conservative manner.

In the course TDT4120 - Algorithms and Data Structures [27] at NTNU, algorithms have traditionally been taught using presentations by a lecturer and simple animations illustrating the algorithms, coupled with assignments were these algorithms have to be implemented by the student. These implementations are then run by an automated system, and the speed of the submitted algorithms are graded and published in a ranking as a gamification of the learning process. Many students find this motivating, and work harder to get a better score. While this is certainly motivating for the better students in the course, it may alienate those who struggle to grasp the fundamentals. This thesis wants to explore how new technology, specifically Virtual Reality, can be used in an alternative way to better facilitate learning.

1.1.2 Using Virtual Reality for Educational Purposes

Virtual Reality has been around for several years, but only recently has the cost come down to an affordable price where it is now feasible to try and use Virtual Reality for anything. The intuitiveness of VR is also a positive, as users can very quickly grasp how to use the technology and interact with a virtual world. With traditional technology, there is often a much bigger abstraction between the end-user and the software, such as the computer screen, mouse and cursor, which results in significant time spent on learning the tools themselves.

1.1.3 Limitations of Traditional Methods

Passive learning where students gather in a classroom, in which a lecturer gives information in a one-sided manner is a relic of the past, when books were costly and not affordable by most. As research regarding learning has continued, there has been a shift from the traditional behaviorism model of learning to a more constructivist [50] approach, as other forms of education can be much more effective than the traditional classroom model. They can give context to the material, or give students hands-on experience with the contents. This thesis will explore the benefits of going away from the more traditional teaching methods for established subjects and using new technology to enhance the learning experience.

1.2 Problem description

This thesis will explore the possibilities of the learning of algorithms, specifically targeting students of computer science, through virtual reality. To accomplish this, some educational software will be developed. With this software the thesis sets out to explore the possibilities of learning and teaching algorithms in a virtual 3D environment.

1.3 Research Questions

To explore the problem description, these research questions were written.

- **RQ1:** How can Virtual Reality be utilized to create an instructional environment to teach students the inner workings of algorithms?
- **RQ2:** What benefits and drawbacks can Virtual Reality provide compared to more traditional teaching methods?
- **RQ3:** Can such software be used for training and evaluation purposes?

1.3.1 Research Question 1

For subsequent research questions to be answerable, it must first and foremost be possible to create an instructional environment in VR. To this end this research question wants to look into how instructional environments should be constructed in VR to maximize learning, and how these environments may differ from traditional environments.

1.3.2 Research Question 2

This research question will look at the various benefits and drawbacks of Virtual Reality, and discuss whether or not it is worth pursuing compared to traditional teaching methods. Some of the general benefits and drawbacks were already listed in this chapter.

1.3.3 Research Question 3

In addition to the aspect of learning, this research question aims to answer if VR educational software can also be used for other purposes, such as training or for evaluation purposes, such as for examinations, and what functionality would be required to do so.

1.4 Contributions

Virtual reality for educational purposes is a relatively young field. Although studies exists concerning virtual reality as an educational tool, actual applications implementing these concepts are sparse, and are usually restricted to simple 360°videos and virtual tours.

What this thesis tries to do is explore how virtual reality can be used to teach complex problems at university levels. This is done by implementing ideas and suggestions from existing studies into a functioning application.

This is done by creating an environment where the user must interact directly with the environment. As far as other research seems to indicate, this has not previously been explored, as similar applications try to teach physical skills, such as medical procedures or operating machinery [17][8], not more abstract concepts such as algorithms.

1.5 Thesis Stucture

This thesis will in Chapter 2 present some background material to show and explain previous work, technology and theories used during the development of the application.

In Chapter 3 it will continue by explaining the implementation of the application, from equipment used, methods and overall structure, to more specific details of the application. This chapter will also explain how the final application was evaluated.

Chapter 4 will contain a descriptive walkthrough of the final application, as experienced by a user. It will also contain links to videos showcasing various functionalities of the application. Finally it will present the results of the evaluation.

Chapter 5 will mainly discuss the evaluation results, and how they relate to the requirements and research questions. It will also discuss certain aspects of the development and evaluation processes.

Finally, Chapter 6 will contain a conclusion, and presents suggestions for future work on the application.

Chapter 2

Background

This chapter will focus on the various background information gathered during the specialization project that is relevant for VirtSort. It will first describe the previous work done on the VirtSort project in section 2.1, before moving on to the theory behind it in section 2.2. Here, both theory regarding various algorithms and theories within learning and learning technologies will be discussed. In section 2.3 various options within the VR market are described and their pros and cons discussed, before finally moving on to discussing potential software in section 2.4.

2.1 Previous Work

VirtSort was first started as a Specialization Project at NTNU. During the Specialization Project, an initial prototype was created. This prototype supported only the bubblesort algorithm, and had the bare bones functionality for allowing the user to sort a list of numbers, represented as boxes on a table, in the correct order following the bubblesort algorithm. While this initial prototype also supported showing instructions to the user, no instructions were actually written to be displayed. Furthermore, it was not possible for the user to change the difficulty of the task.

Thus this project will focus on further development of this initial prototype. Development will focus on implementing various other algorithms, as well as improve customizability for the user to tailor the learning experience to their needs, and give proper instructions and feedback to the user.

2.2 Theory

2.2.1 Learning

Learning Algorithms

Bloom's Taxonomy [57] for learning is a well known model in learning theory created by Benjamin Bloom in 1956. The model is shown in figure 2.1. In it Bloom classifies learning objectives by complexity and specificity, and organizes them in a pyramid shape. The pyramid is divided in six slices from top to bottom, with the wide *Remember* field as the base. Building upwards are the fields *Understand*, *Apply*, *Analyze*, *Evaluate*, and finally *Create* at the top. What the model tries to show is that to do any one activity, one has to master everything that is below it.

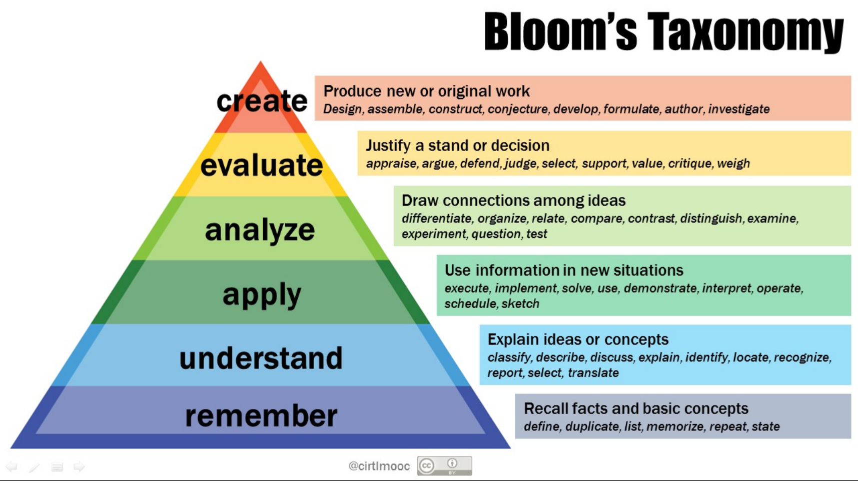

Bloom's Taxonomy
create
Produce new or original work
Design, assemble, construct, conjecture, develop, formulate, author, investigate
evaluate
Justify a stand or decision
appraise, argue, defend, judge, select, support, value, critique, weigh
analyze
Draw connections among ideas
differentiate, organize, relate, compare, contrast, distinguish, examine, experiment, question, test
apply
Use information in new situations
execute, implement, solve, use, demonstrate, interpret, operate, schedule, sketch
understand
Explain ideas or concepts
classify, describe, discuss, explain, identify, locate, recognize, report, select, translate
remember
Recall facts and basic concepts
define, duplicate, list, memorize, repeat, state
@cirtlmooc

Figure 2.1: Bloom's Taxonomy [4]

Looking at Bloom's Taxonomy in relation to how algorithms are currently being taught in TDT4120, one can see that it first introduces the sorting algorithms, then tries to explain the principles behind it. Afterwards, the student is required to submit an assignment where they are tasked with implementing said algorithm. This means that the course goes through all the steps of *remember*, *understand* and *apply*, and a student is tested on these during the assignment. This is important as later in the course the students are required to *analyze* and *evaluate* properties of various algorithms such as complexity, and explain why one algorithm may be better suited for a task than another. Excerpts from a lecture in TDT4120 explaining recursion and an animation of insertionsort has been included as appendix A.

While this is all well and good in theory, a problem arises if a student does not properly *understand* the core concepts of the algorithm by the time the assignment is due. Without a proper understanding of the algorithm, a student might find themselves struggling later in the course, and being the source of why the course has a reputation of being hard. Magnus Lie Hetland [16], the lecturer of TDT4120, stated that many students in his course seem to struggle with concepts such as recursion and red-black trees.

Learning Theory

To help students better understand concepts in TDT4120, it is important to look at the second layer of Bloom's Taxonomy, and how VirtSort can be used to help students in this regard. Several studies have reported that making the learning experience fun makes it more effective [58]. There is no definite way to make learning fun, but attempts have been made to specify some key points. Thomas W. Malone at the Xerox Palo Alto Research Center has suggested that the cornerstones of a fun learning experience are *challenge*, *fantasy*, and *curiosity* [56].

Challenge is self-explanatory: A game that is too easy is boring; a game that is too difficult can be disheartening. it is important to have a well balanced game that gives the user the proper amount of challenge for them to have fun. As different players will have different thresholds of what they find challenging, it is not easy to fine-tune this perfectly, and often even impossible. While in some cases a one-size fit-all approach to difficulty may be preferred, it is in most cases better for the user experience to allow the player to set their own difficulty level.

Fantasy in this context means that the problem or learning material should be presented as a fantasy scenario. That is, there needs to be a connection between what the user learns, and what the game is about. This connection can be both *extrinsic* and *intrinsic*. *Extrinsic* fantasy means that the fantasy is dependent on the user's skill, but not the other way around. An example of an extrinsic fantasy would be a baseball game where the player moves to the next base every time they answer a question correctly. In an *intrinsic* fantasy the skill depends on the fantasy as well. In other words the fantasy does not only depend on whether a skill is used correctly, but also how it is used compared to the proper way. If a game requires the player to throw darts to pop balloons, then the skill required, throwing darts, is an integral part of the fantasy. In most cases an intrinsic fantasy is a more fun experience compared to an extrinsic one, and therefore preferred.

Curiosity is the learner's desire to learn the material. This can be evoked by complexity in the game. As with *challenge*, striking a balance in complexity is important; the game has to be neither too simple nor too complex. Curiosity can be categorized in two categories: sensory and cognitive curiosity. Sensory curiosity is the attention and attraction to changing stimulations such as sounds and lights. An example of this is the spectacle that follows from a correct answer on a television quiz show. Cognitive curiosity is the user's desire to acquire additional knowledge. A commonly used technique in story writing is to end books, chapters, or TV shows in cliffhangers, presenting only partial knowledge. This makes the reader and viewer, or in the case of learning, the learner, desire for the complete picture, and is forced to seek out the answer for themselves.

Learning Technology

Learning technology describes technology used as a tool for learning, and its applicability for this purpose has been the subject of various studies. There are numerous ways technology can be applied to learning, and as new technology is developed, new possibilities emerge. It is important to note however, that simply using a learning technology will not necessarily enhance the learning experience; the right technology must be used the right time in the right way to be effective [55, Ch. 17].

In recent years *serious games* have seen a surge of popularity. These are learning environments that try to combine game and learning components together [55, pp. 218-219]. Although popular, few serious games are successful, and it seems to be hard to get them right. One example of a successful serious game is America's Army [2], a military simulator created for the US Army meant to distribute information and recruit new soldiers. After its release as a free-to-play game in 2002, the US Army could report a substantial increase in new recruits, and the recruits had already learned certain knowledge and skills that usually would have been learned after joining the army [65].

2.2.2 GameFlow

GameFlow is a model that tries to evaluate player enjoyment in a game [66]. As stated by Michael Zyda, it is important that the story takes precedence over the pedagogy in a serious game [9]. More generally, the act of learning should be in the background of a player's enjoyment of the game itself. Previous attempts of gamification, *edutainment*, developed purely instructional software with only some game elements [66, p. 29], which failed to be effective. Therefore, the focus of gamification have also gone into looking at what makes a game enjoyable for a user.

While many models that look at player enjoyment focuses on narrow sets of features, *GameFlow* tries to be a more extensive evaluation. This evaluation is based on eight different criteria listed in table 2.1.

Table 2.1: The criteria of GameFlow

Concentration	Games should require concentration and the player should be able to concentrate on the game
Challenge	Games should be sufficiently challenging and match the player's skill level
Player Skills	Games must support player skill development and mastery
Control	Players should feel a sense of control over their actions in the game
Goals	Games should provide the player with clear goals at appropriate times
Feedback	Players must receive appropriate feedback at appropriate times
Immersion	Players should experience deep but effortless involvement in the game
Social Interactions	Games should support and create opportunities for social interaction

2.2.3 Benefits of Virtual Reality

Virtual Reality provides several benefits to the end user in both learning theory and gameflow. The biggest strength of virtual reality is the strong sense of *presence* that it provides the user, as well as the *hands-on* nature of the technology. The technology is also very *intuitive* compared to traditional technology such as a mouse, keyboard and computer screen. All of these points make the user feel truly immersed in the virtual world, and greatly enhances both the *fantasy* and the *curiosity* cornerstones of Thomas W. Malone's theory for a fun learning experience. It also fullfills several of the criterias in GameFlow. In particular, curiosity can be invoked through only the fact that VR technology is used at all, as a user who has never experienced VR before will be curious to explore what the technology can do. This kind of curiosity would then indirectly lead them to try the various mechanics of the actual game, though there is no guarantee that it will remain over time, especially if VR technology becomes more commonplace. In the GameFlow model, *concentration* is enhanced through the transportation to a virtual world separate from the real world, keeping potential distractions out. *Control* is given to the user through the intuitive interactions provided by VR technology, and the *immersion* factor is self-explanatory.

2.2.4 Sorting Algorithms

Bubblesort

The bubblesort algorithm is a very simple and easy to understand sorting algorithm, and is thus a perfect algorithm for new students to learn. An implementation of bubblesort written in C# can be found in listing 2.1.

Listing 2.1: Implementation of bubblesort in C#

```csharp
static int[] bubbleSort(int[] array)
{
    int temp = 0;

    for (int i = 0; i < array.Length; i++) {
        for (int j = 0; j < array.Length - 1; j++) {
            if (array[sort] > array[sort + 1]) {
                temp = array[sort + 1];
                array[sort + 1] = array[sort];
                array[sort] = temp;
            }
        }
    }
    return array;
}
```

For anyone familiar with code, the functionality of the algorithm should be evident. The algorithm passes through the array of unsorted elements and for each element, compares that element with the rest of the array, *bubbling* the element towards the end of the array. For each iteration, one more element is sorted at the end of the array, and the algorithm continues until all elements have been compared once. This means that the algorithm has a complexity of $O(n^2)$. Since the bubblesort algorithm often performs worse than even other $O(n^2)$ algorithms such as insertionsort described in the next section, it is not a practical algorithm for general use. However its simplicity makes it effective for grasping some of the fundamental processes of algorithms. There are also some very easy optimizations that a student may stumble upon, such as not making comparisons for already sorted elements, and ending the algorithm when an iteration with no swaps have occured.

Insertionsort

Insertionsort is another algorithm with a complexity of $O(n^2)$, similar to bubblesort. The concept of insertionsort can be thought of in a recursive manner, where the element to be sorted is inserted in its correct position in a sorted list. While insertionsort is usually not implemented recursively (an implementation in C# can be found in listing 2.2) it is ideal for introducing the concept, as it is used during the introduction of recursion in TDT4120. Even though insertionsort shares the same complexity and worst case scenario as bubblesort, it is, on average, significantly faster than bubblesort.

Listing 2.2: Implementation of insertionsort in C#

```
static int[] insertionSort(int[] array)
{
 for (int i = 1; i < array.length; i++)
    {
        int j = i;
        while ((j > 0) && (array[j] < array[j - 1]))
        {
            int temp = array[j-1];
            array[k] = array[j];
            array[j] = temp;
            j--;
        }
    }
    return array;
}
```

The insertionsort algorithm iterates through every element from left to right, and each element is compared to its previous element and swapped until it reaches the beginning of the array, or the previous element is smaller than itself. As the elements to the left of the current elements are always sorted in the correct order, the new element will at this point be in its correct position.

Quicksort

Quicksort is a more complex algorithm to understand than the previous two, and while its worst-case performance is still $O(n^2)$, the average performance of quicksort is $O(n\log(n))$ comparisons. To sort an array, the quicksort algorithm selects a pivot at random, and sorts all elements smaller than the pivot to the left of the pivot, and all elements larger to the right of the pivot. At this point, the pivot will be in its correct position, and the quicksort algorithm is called recursively on the left side and right side. These are called the left partition, and the right partition. When a partition has zero or one element, it is considered sorted. An implementation of quicksort in C# has been provided in label 2.3.

Listing 2.3: Implementation of quicksort in C#

```
static void Quicksort(int[] array, int left, int right)
        {
            int i = left;
            int j = right;
            int pivot = 0;

            if (left < right)
            {
                pivot = Partition(array, left, right);
                Quicksort(array, left, pivot - 1);
                Quicksort(array, pivot + 1, right);
            }
        }

static int Partition(int[] array, int left, int right)
{
    int x = array[right];
    int i = left - 1;
    int temp;

    for(int j = left; j < right; j++)
    {
        if (array[j] < x)
        {
            ++i;
            temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    temp = array[i + 1];
    array[i + 1] = array[right];
    array[right] = temp;

    return i + 1;
}
```

2.3 Hardware

2.3.1 Virtual Reality

The term virtual reality (VR) was first coined by artist and computer scientist Jaron Lanier in 1987 [39], although the concept of Virtual Reality can be traced back as far as to the 1860's [34]. The first example of a virtual reality device was the Sensorama, developed in 1957 by Morton Heilig. It was a multimedia device capable of showing stereoscopic 3D, a fan to simulate wind and a device that emitted smells, among other things. VR also had a popularity spike in the 1990's, particularly thanks to the aforementioned Jaron Lanier, but soon people realized the technology could not live up to their expectations. Popularity waned, and VR disappeared from the public scene until the spark was once again reignited in the 2000s by the launch of Oculus Rift's Kickstarter campaign in 2012 [22].

Although the technology that lets a user experience a fully immersive, virtual world is collectively called Virtual Reality, many different technologies can be used to achieve the same effect to varying degrees. It is important to understand the advantages and disadvantages of the different options available today to pick the technology best suited for the project.

Oculus Rift

After Oculus Rift's Kickstarter launched in 2012 it quickly raised almost $2.5 million from backers [22], kicking off the current development of Virtual Reality for the mainstream. The first Development Kit (DK1) was launched in 2013, and Oculus was then acquired by Facebook in 2014 [60]. The Rift is currently in its third iteration, the Consumer Version 1 (CV1), which was released in 2016.

The Oculus Rift HMD uses two OLED panels with 1080 x 1200 pixels for each eye, putting the final resolution at 2160 x 1200 pixels. These panels run at 90Hz to ensure a smooth experience while minimizing risks of motion sickness. The headset offers 110° field of view [7] and has to be tethered to a powerful computer with cables.

The Rift uses an optical outside-in tracking system Oculus called *constellation* [37] where external infrared cameras point at the user wearing the HMD, and tracks the location of several infrared emitting diodes on the HMD. Using the unique arrangements of these diodes, the system can track the user in space in addition to pitch, yaw and roll available from the gyroscope and accelerometer in the headset, giving the Oculus Rift six DoF. The Rift comes with two tracking cameras, but supports adding additional cameras for improved tracking.

The CV1 originally launched with an Xbox controller, meaning that there was no hand-presence in the virtual world. In Fall 2016 however, Oculus launched the Oculus Touch controllers, giving the user full room-scale interaction.

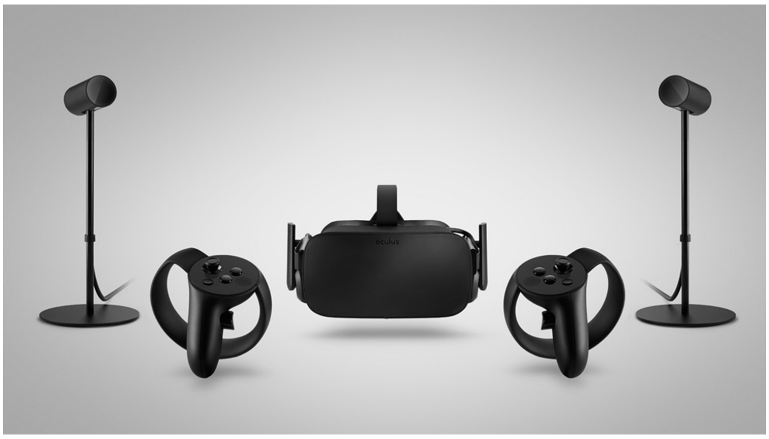


Figure 2.2: The Oculus Rift [6]

HTC Vive

The HTC Vive was released the same year as the Oculus Rift CV1, but unlike the Rift, the Vive released with controller support. These controllers are called the Vive Wands, and are the primary tool used for interacting with the virtual world while using a Vive. The Vive is a SteamVR headset, and is natively supported by Steam.

Similar to the Rift, the Vive also uses two OLED panels with a final resolution of 2160 x 1200 pixels, and a 110 degrees FoV [7]. Unlike the Rift however, the Vive uses two passive base stations mounted diagonally for tracking its spacial position. The cameras are directly on the HMD, and looking out for the base stations.

In 2018 HTC released the HTC Vive Pro, an upgraded version of the original Vive aimed at the professional space. With many improvements over its predecessor, such as an improved headstrap and improved resolution, among other things.

Figure 2.3: The HTC Vive [14]

Sony PlayStation VR

The PSVR is Sony's first VR device, and is driven by the PlayStation 4 [23]. It uses a similar tracking technology to the Rift, but instead of tracking light in the infrared space, it uses visible light to track both the HMD and the controllers. This sometimes causes tracking quality to degrade in very brightly lit rooms. Sony also repurposed the PlayStation Move controllers for hand-presence in the virtual world.

Figure 2.4: The PlayStation VR [53]

Microsoft Mixed Reality

In 2017 Microsoft also entered the space of Virtual Reality, although under their own umbrella term of Mixed Reality (MR), described as a spectrum of everything between Augmented Reality (AR) and VR [19]. These HMDs are manufactured by several of Microsoft's partners, such as Asus, Dell and Samsung, and are more affordable than both the Rift and the Vive at the time of writing [18]. They are also the first headsets to use inside-out tracking. This means that Microsoft Mixed Reality headsets and controllers require no additional equipment to be mounted in the room, as all tracking is done using the cameras attached to the HMD itself.

Figure 2.5: Various Microsoft Mixed Reality headsets [51]

Google Cardboard

The entry level VR HMD, a Google Cardboard [12] is what the name suggests; a cardboard box in which one can put a mobile phone inside to create an affordable HMD for the masses. While it certainly does not look like a high-end piece of tech, it is often someone's first introduction to VR. The cardboard construction also sets the expectations accordingly for the user; that the experience is, for better or for worse, only the lowest denominator for what VR really is. An increasingly big problem in the market today are the many mobile VR HMD's that look like high-end headsets similar to the Vive and Rift. As the average consumer try these good-looking HMDs, they might get the wrong impression of what high-end VR really can be.

The specifications of the Google Cardboard is completely dependent on the device used with it. As most phone screens only support up to 60Hz refresh rate or less, they are not ideal for extended sessions or experiences requiring lots of movement. Furthermore, they only offer three DoF, and do not track position in space. This makes them perfect for activities such as video consumption where rapid head movement is not necessary, but not very well suited for games, and certainly not suitable for room-scale experiences.

Figure 2.6: The Google Cardboard [67]

Google Daydream View

The Google Daydream View [13] is the more premium VR offering from Google, and is compatible with any Daydream-ready phone. The headset has three DoF, similar to the Google Cardboard, but does include a single controller. This controller also only have three DoF, so full hand presence is not achievable with it, but it is good enough for using as a simple pointing device in VR. Unlike the Google Cardboard, the headset itself also contains additional motion sensors, which gives the Daydream superior tracking fidelity compared to most, if not all, Cardboard experiences.

Figure 2.7: The Google Daydream View [46]

Samsung GearVR

Samsung GearVR is a VR headset developed by Samsung in collaboration with Oculus and first released in November 2015 [24]. The GearVR is a high-end option on the same tier as the Daydream, and requires the use of specific models from Samsung's Galaxy model series. The phone is placed inside the headset and connected with a micro-USB connector, and works as the device's display.

Figure 2.8: The Samsung GearVR [10]

Oculus Go

Oculus Go [20] is the newest VR offering from Oculus at the time of writing, and is a standalone HMD with similar capabilities to Google Daydream and Samsung GearVR. The difference though is that the Oculus Go is a completely standalone headset that does not need an additional mobile device to function. This allows Oculus to optimize the Go for the experiences it supports. In addition to the HMD itself, a single controller is also included. Both only have three DoF, however, and Oculus positions this as entry-level hardware for VR.

Figure 2.9: The Oculus Go [49]

2.3.2 Limitations and Challenges

As with any technology, working with VR comes with it its own set of limitations and challenges. Furthermore, these limitations and challenges may differ between all the choices that are available. Some HMDs might be better in some use cases, but worse in others, and choosing the correct HMD for a project depends on the requirements. These trade-offs include:

Complexity of Setup

Some HMDs, such as the Google Cardboard, are easy to set up; a user starts an app on their own mobile phone, and pops it into a cardboard enclosure, and they are ready to go. Other HMDs such as the Rift and the Vive require much more engagement from the user during the initial setup. Sensors have to be mounted or placed, tracking has to be calibrated, and HMDs have to be connected to a computer. Other HMDs still land somewhere in the middle. Microsoft's MR headsets still require an additional computer to run software, but setting up the HMD itself is as easy as plugging the headset to the computer. In the case of VirtSort, the initial set up is not a significant barrier, as it is intended as educational software to be used in universities. This means that the HMDs can be set up by professionals before they are used by the student.

Space Requirements

For HMDs such as the Google Cardboard and the GearVR, space is not a problem, as they only support tracking with three DoF, meaning that there is no movement in space. This is not the case for the Rift, Vive, PSVR or Microsoft's HMDs. Unless the experience does not require positional tracking, room-scale VR is seen as the most immersive experience, and as such HMDs that support room-scale should be used if possible to leverage the most benefits from VR technology.

Computing Power

VR requires considerable computing power. High-end systems such as Rift and Vive requires rendering of two separate high-resolution images 90 times a second. In an interview with VentureBeat in 2015, representatives from the famous GPU manufacturer Nvidia claimed graphic processors needed to be seven times more powerful to properly handle VR content [42]. In the case of the lower end devices, the experience will be limited by the processing power of the mobile device used, or the headset itself.

Cost

The cost of different VR devices range from the entry-level Google Cardboard at around \$8.00 to the expensive HTC Vive Pro at \$799. While the Google Cardboard itself is cheap, there is still the additional cost of buying a phone capable of running VR experiences. Meanwhile, for Rift and Vive a high-end computer is needed to properly drive them. Of course, if cost is not a primary concern, there is no reason to not use the high-end systems, as they have the same capabilities as their mobile counterparts, while offering additional benefits and a much better experience.

Health Risks

It is very common for users, especially those with no prior experience with VR technology, to experience nausea when using a VR HMD. To decrease the risk of nausea, the Rift and Vive uses panels that can display images at a rate of 90 frames per second. This is sufficient to create a smooth experience for the user. Additionally, there must be no discrepancy between the user's movements and what they experience in the headset. If the sensors of the Rift or Vive lose track of the user, the system falls back to three DoF tracking, which can be very disorienting if the user is physically moving in space.

Another common cause for nausea is moving from point A to point B in VR. Artificial movement, or locomotion, can often cause nausea for inexperienced users. VR applications should be confined within the room-scale environment that is tracked, so that all movement can be the natural movements of the user. To move for longer distances than the tracked playing area however, it is often preferred to use a teleportation mechanic instead of traditional stick movement found in various games. In playing areas where the tracking does not support 360° rotation, snap-turning is also preferred, where the user is turned in predetermined amounts.

Due to these concerns, using the high-end VR headsets is recommended rather than the cheaper options, as they greatly reduce the chance for nausea. For anyone who experiences even slight discomfort using VR headsets it is recommended to close their eyes and take a minute or two recomposing themselves, or outright take off the headset if it is too overwhelming.

2.4 Software

Using and developing VR applications requires the use of various software programs. To create applications, a game engine is normally used, and the most popular ones usually comes with integrated VR support, at least to some degree. A program to handle communication between the VR system and the computer is also required, and are often created by the VR manufacturers, but independent, often open source, programs also exist.

2.4.1 Game Engine

A game engine is a software tool used mainly to develop video games. Game engines comes in all shapes and sizes, from 2D engines with specific genres in mind, to powerful all-purpose 3D engines with an integrated physics engine, modeling and animation tools, artificial intelligence and networking. In recent years, with the rise of VR, more engines have added support for VR development to different degrees.

Unity

Figure 2.10: The Unity logo [48]

Unity [31] is currently the most popular multipurpose engine on the market. It supports both 2D and 3D graphics on 27 different platforms, including VR platforms such as SteamVR, PlayStation VR, GearVR and Windows Mixed Reality. Its popularity can be contributed to several factors. First of all, it uses a freemium model, where the base engine is free to use, but additional features such as source code access and premium support is available for a price. The development pipeline is also quite fast, with scripts written in C#. It also features an asset store where anyone can buy and sell various components, such as scripts and 3D models.

Unity support VR through an API called Unity VR and supports all the major VR systems. In addition, a plethora of VR assets are available on the asset store, like the SteamVR and Virtual Reality ToolKit assets.

Unreal Engine 4

Figure 2.11: The Unreal Engine 4 logo [47]

Unreal Engine 4 is the most recent version of game developer Epic Games' multipurpose game engine. Previous versions of the engine have been widely popular among professional developers, but after Epic made the Unreal Engine 4 free to use in 2015, it has also become a very popular engine among smaller and independent developers. To make this sustainable, although it is free to use, Epic claims a 5% royalty on gross revenue after the first $3'000 per product [41].

Chapter 3

Equipment, Methods and Implementation

This chapter will outline the choices made with regards to equipment, tools and frameworks, development and cooperation methods, and the implementation itself.

3.1 Equipment, Tools and Frameworks

Creating a virtual reality application requires usage and knowledge of a wide variety of equipment, tools and frameworks. During the course of the project, decisions had to be made regarding which of these would be beneficial for the project and the developers. In this section, the chosen equipment, tool and frameworks will be explained, as well as the process leading up to their inclusion.

3.1.1 Virtual Reality

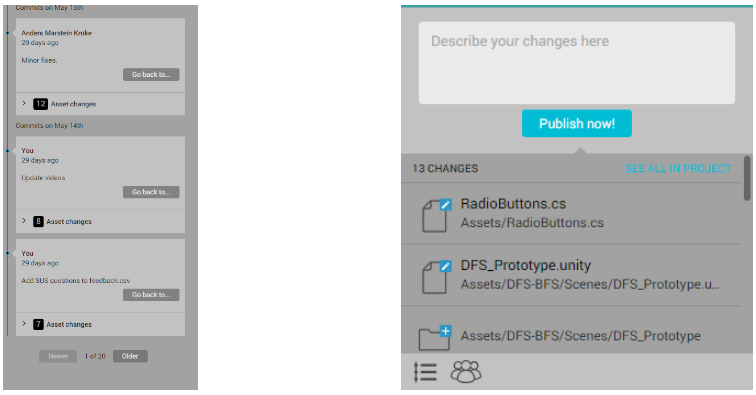
With all the different hardware available in today's market, the choice of which platform to support relied largely on the intent and purpose of the project, and availability of hardware. As discussed in section 2.2.3, there are many benefits that VR technology can provide, one of which is immersion. While cheaper headsets such as Oculus Go and Google Cardboard provides an immersive *passive* experience, due to the nature of VirtSort it was important that the *active* experience was immersive as well. The immersion of the user in this virtual world would be broken if there were no means of interaction outside of using a regular game controller, and thus lowering the value of using Virtual Reality over traditional technology in the first place.

Furthermore, while the Oculus Rift and the HTC Vive are certainly expensive for the average user, costing €450 [21] and $500 [36] respectively at the time of writing, the VirtSort project is intended for use at universities. Therefore cost in this case would be only a secondary concern. Coupled with the upwards trend of VR adoption today [29] and the recent price drops of both the Rift and Vive, the future of VR looks bright indeed.

The target platform for the project was thus considered to be virtual reality headsets for desktop computers, and the systems considered were Oculus Rift and HTC Vive. Due to experiences from the specialization project, it was decided to support both platforms. As discussed in section 3.1.3, this would not require much, if any, additional work. Furthermore, after the introduction of the Oculus Touch controllers, hand tracking became possible for the Rift as well, making the user experience for both systems virtually identical. Lastly, the university was able to provide both systems, as well as space to set up both, making testing for both simultaneously very easy.

3.1.2 Game Engine

In the specialization project prior to this thesis Unity became the engine of choice, as the original intention was to continue development of an already existing Unity project. At the start of this project, it was not considered necessary to switch engine. This was due to two main factors: first, some experience with the engine was already acquired during the specialization project. Second, the quick workflow was a good fit for a short-lived project using agile development, where fast, visible results were necessary. Furthermore, Unity has an integrated service called Collaborate [32] that allows for easy cooperation (figure 3.1). This was considered superior to Git + Git LFS, as the workflow for Git would require considerably more manpower comparatively, especially when dealing with merge conflicts on .yaml files of scenes [45]. The alternative engine, Unreal Engine, was soon discarded, as its complexity would have required a lot of time getting familiar with it, and it did not provide any additional features necessary for the project. To write code for Unity, a



Commits on May 15th

Anders Marstein Kruke
29 days ago
Minor fixes.
12 Asset changes

Commits on May 14th

You
29 days ago
Update videos
8 Asset changes

You
29 days ago
Add SUS questions to feedback.csv
7 Asset changes

1 of 20

Describe your changes here

13 CHANGES

RadioButtons.cs
Assets/RadioButtons.cs

DFS_Prototype.unity
Assets/DFS-BFS/Scenes/DFS_Prototype.u...

Assets/DFS-BFS/Scenes/DFS_Prototype

(a) Project history **(b)** Publishing changes

Figure 3.1: Unity Collaborate interface

code editor or IDE was needed. As development was performed on Windows computers, a natural choice was Microsoft's IDE Visual Studio, as it is powerful, free (the Community version) and integrates well with Unity. Alongside Visual Studio, it was decided to use

Visual Studio Code, a lightweight open source code editor first released by Microsoft in 2015 [35].

3.1.3 Frameworks

To make the development easier, especially with regards to virtual reality, several frameworks were used. Chief among these are *SteamVR*, which is Valve's main VR platform for Steam. Most importantly SteamVR implements the OpenVR interface, which lets VR applications communicate with all devices with an OpenVR supported driver. This allows an application to be built without restrictions to a single system, or necessitate writing custom code to support additional systems [44]. In this project, it allowed the application to support both the Vive and the Rift with no additional effort. SteamVR also provides a package for Unity that can be downloaded for free [25], and provides a lot of useful assets such as prefabs, models, materials, scenes and scripts.

Another useful package that was used is Virtual Reality Toolkit, or VRTK [38]. The package provides many useful functions for VR development, such as systems for teleporting, picking up objects and showing VR specific UI elements. It supports many different VR SDKs, including SteamVR, and also provides a VR simulator. This came in handy during development, as it greatly reduced the times the headset had to be worn for testing, which considerably sped up development times.

To properly render text, which plain Unity does not do very well, the package *TextMesh Pro* was used [28].

3.2 Methods

This section will explain the development methods used in the project, and how division of labour was handled in a two-man team.

3.2.1 Agile Development

To develop a prototype application in such a small amount of time required the use of agile development methods. This entails small and frequent iterations, at the cost of deeper structure and planning. This method works great with Unity, as it is based around small independent components, and allows for quickly implementing new components and applying changes to existing components. This also made development easier by requiring less oversight of the complete structure of both Unity and the custom code.

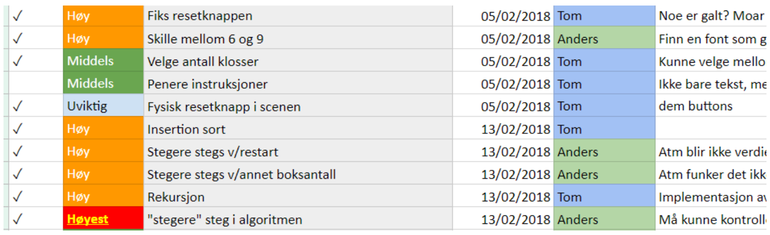
The method used was loosely based around SCRUM, which is one of the most popular agile development methods. It is based around sprints, which are small periods of time, usually weeks, where the next iteration of the product is developed. Before each sprint, it is planned out in detail by the entire team, such as deciding the scope, preparing a sprint backlog, and estimating development time. A *daily scrum* should also take place, where every team member reports their progress from last meeting, and what they plan to do until next time. This should typically take no longer than 15 minutes total.

In this project, each ”sprint” was based around the supervisor meetings usually held every two weeks. Here the newest iteration of the application was showcased to the supervisor, and decisions made were discussed. Then, a road ahead was planned through discussions with the supervisor - which features should be changed, dropped and introduced. After the meeting the developers would discuss how to best implement the changes and new features, and divide the work among them. This discussion also stayed active throughout the sprint period.

3.2.2 Division of Labor

In a team of two, where a lot of the work was done apart from each other, it was essential to divide the tasks properly. As mentioned above, most of the tasks, especially the most important ones, were agreed upon at the biweekly meetings, but new tasks were also added throughout the period as needed. These were then written down in a Google Sheets documents created to keep track of all tasks and issues. Google Sheet was chosen in favor of Trello, an online project management tool by Atlassian [30]. The decision was made to stick with a Google Sheet, as Google's overarching service Drive would also be used for uploading files, such as build files, and to create the testing questionnaire. In such a small project, it was decided that fewer, non-specialized tools was more beneficial than more and specialized tools as it would decrease overhead.

The sheet contained fields for completion state, priority, task description, deadline, person and comments. Furthermore, the priority and person fields were customized to color code the various options, and completed tasks could be filtered away, to make the sheet more clear as can be seen in Figure 3.2. The full Task List can be found in Appendix C.



✓	Høy	Fiks resetknappen	05/02/2018	Tom	Noe er galt? Moar
✓	Høy	Skille mellom 6 og 9	05/02/2018	Anders	Finn en font som g
✓	Middels	Velge antall klosser	05/02/2018	Tom	Kunne velge mello
	Middels	Penere instruksjoner	05/02/2018	Tom	Ikke bare tekst, me
✓	Uviktig	Fysisk resetknapp i scenen	05/02/2018	Tom	dem buttons
✓	Høy	Insertion sort	13/02/2018	Tom	
✓	Høy	Stegere stegs v/restart	13/02/2018	Anders	Atm blir ikke verdi
✓	Høy	Stegere stegs v/annet boksantall	13/02/2018	Anders	Atm funker det ikk
✓	Høy	Rekursjon	13/02/2018	Tom	Implementasjon a
✓	Høyest	"stegere" steg i algoritmen	13/02/2018	Anders	Må kunne kontroll

Figure 3.2: Task list - Google Sheets

To effectively communicate between the developers when apart, Facebook's integrated chat service Messenger was used. The main reason was that it was already a familiar tool, and its features were considered sufficient for its purpose. For communication involving the supervisor, e-mail was mostly used, interspersed with some mobile text messages.

3.3 Requirements

This section will present the requirements specification for the project. The case description will be described first, which details a macro overview of what the prototype is expected to deliver. Afterwards various requirements identified from the case description will be documented.

3.3.1 Case Description

The user is expected to go through two scenes while using VirtSort: Picking an algorithm, and sorting an array using that algorithm. A high level description of the scenes and their expected functionality have been provided. Using these scenarios, a set of functional and non-functional requirements were identified.

Scene 1 - Picking an Algorithm

When starting the game, the user should be able to spend some time to familiarize themselves in the virtual world. They may teleport around in a fixed area, and generally move around in the space. In their hands are model representations of the VR controllers they are holding, with clear indications for what each button does. When the user is ready, they may pick one of several algorithms to be taken to the next scene.

Scene 2 - The ”Algorithm Room”

In this scene, the user is presented with a table with boxes on top. To the left of the user are various panels that the user may use to change how difficult the problem is. This includes settings such as showing or hiding the numbers on the boxes, changing the amount of boxes, or changing the amount of feedback from performing the algorithm correctly. The user can also choose to watch a video explaining the algorithm they picked from Scene 1. Afterwards, the user may step forward to the table and grab a box with their hand. Using hand interactions made possible by VR, the user learns the inner workings of an algorithm by physically rearranging the boxes step by step according to the algorithm.

3.3.2 Functional Requirements

Table 3.1: Functional requirements

	Requirement	**Description**	**Priority**
FR1	Implement sorting algorithms	Some sorting algorithms, eg. Bubblesort, Insertionsort and Quicksort, must be implemented and playable by the user.	High
FR2	Step-by-step progression	The algorithm must have an interface that progresses it one step at a time, and to show the current state to the user.	High
FR3	Visualize the problem	Visualize the sorting problem, e.g. as a row of boxes with numbers representing the numbers in the array.	High
FR4	Ability to move around the scene through VR	Use the VR motion controllers to move around the scene. A common approach to VR movement is by teleportation.	Medium
FR5	Ability to interact with boxes through VR	To carry out the algorithm the user must be able to pick up the boxes and rearrange them.	High
FR6	Give user feedback while carrying out the algorithm	The user should get feedback on whether he has done something right or wrong, and where in the algorithm he is currently at, e.g. by showing which boxes have been placed in their final position.	High
FR7	Show instructions to the player	In-game instructions will allow users to play through the application without prior knowledge of the algorithm.	High
FR8	Random problem	The ability to create random problems will ensure a new challenge on each playthrough.	Medium
FR9	Problem size	Ability to change the size of the problem, i.e. the size of the array to be sorted.	Medium
FR10	Visualization of Recursion	Illustrate the concepts of visualization for the user, for example by separating recursive calls by table.	High
FR11	Show the user how to use the application	As many users will be first-time users of both VirtSort and VR, There needs to be instructions as to how the user interacts with the virtual world.	Medium
FR12	Adjust the problem according to the user's level of knowledge	Different users will have different levels of knowledge when using VirtSort. The program should facilitate for various difficulty settings to better suit individual needs.	High

3.3.3 Non-functional Requirements

Table 3.2: Non-functional requirements

	Requirement	**Description**	**Priority**
NFR1	Usability	As the application might be used by people who have little or no experience with VR, video games or computers in general, the application should be easy to use, with intuitive controls.	High
NFR2	Ability to teach	The main focus of the application is learning, so users need to be able to learn something after a reasonable amount of time.	High
NFR3	Extendability	The application should be able to be extended to include more sorting algorithms, and even other types of algorithms. Some of this is covered simply by using Unity for the implementation.	High

3.4 Structure

This section will explain the decisions made with regards to the structure of the application. Having a defined structure helps the development substantially, and was therefore an important part of the planning stage.

3.4.1 Overview

This project is a continuation of the specialization project from the previous semester, during which a basic bubblesort was implemented. The implementation goal for this thesis was first to create a supporting framework for sorting algorithms in general, and then to implement a couple of algorithms, including improving bubblesort and adding insertionsort and quicksort. It was also intended to develop several other prototypes for different kind of algorithm classes to explore the possibilites and solutions for teaching these through virtual reality.

As the project went on, it was decided to focus more on streamlining the sorting algorithms, until it became the sole focus of the project. This decision was made mainly to have a product thorough enough so that later testing would give a useful result. It was reasoned that simple prototypes might not be good enough to give a decent indication in regards to the effectiveness of learning algorithms through VR.

3.4.2 Architecture

To meet the non-functional requirements, the overall architecture of the application was decided early on, and had usability and modifiability in mind.

Usability

As the application is supposed to be a tool for learning, usability was naturally extremely important. Furthermore, as VR is still in its youth, most people will not be comfortable with the technology, making this aspect even more important. To support usability it was decided to make all interactions as simple as possible and intuitive as possible. This can be seen in the application as the player can only perform three basic actions, teleporting, pushing buttons and picking up objects, which requires a total of just two buttons.

The application also implements several methods to prevent the user from getting confused and of making game breaking mistakes, and help recover from any of these situations should they occur. An example of something potentially game breaking is losing one of the boxes, for example by throwing it away. If this box lands outside of the playable area within the game, the user would be unable to pick it back up. Thus, the moment it hits the floor, the box teleports back to its original position, or if that position is occupied, to a separate dedicated table to the side of the sort table. The player is simply not able to lose a box.

Another feature in VirtSort that help usability is the teleportation system. As discussed in section 2.3.2, teleportation is a common preventative measure for nausea. But teleportation in VR can be a confusing ordeal when first encountered, as it has no equivalent neither in real life nor in non-VR video games. To help with this, the player's movement is restricted to just an area immediately around the sorting table and options panels, as shown in figure 3.3. This hinders the player from getting lost by teleporting far off into the distance by accident, and makes it clear that there are not other areas to explore in the vicinity.

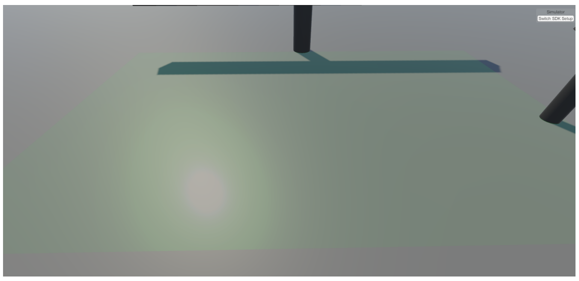


Figure 3.3: The green square is the playable area.

Modifiability

To make future development more streamlined, as well as the possibility of making VirtSort a platform for virtual learning of many different algorithms, modifiability was important during the implementation process. It should be easy to add additional algorithm types and to extend already implemented algorithms. This also entails using few hard coded values, but instead making values visible and editable in the Unity editor through public variables. A good example of this can be seen in the *R_GameManager.cs* script, which exposes most of its variables.

During the Specialization Project, some thought was put into support for several algorithms to be played out. In the first prototype, an interface was created for algorithms with methods to advance and go back in the algorithm. The main logic of advancing the algorithm and displaying changes to the user was placed in a separate Unity script called *Manager.cs*. While this worked fine for the first prototype, which only supported bubblesort, it was quickly discovered that there was a lack of flexibility for *Manager.cs* to support other sorting algorithms. Thus the code from the Specialization Project was refactored, such that the algorithm scripts could handle both the advancements in the algorithm itself, as well as the task of displaying changes to the user. Meanwhile the refactored *R_Manager.cs* was delegated to handling choice of algorithm, spawning the *Table* and *SortBox* instances, as well as handling user input unrelated to the algorithm itself, such as difficulty settings and restarting.

3.4.3 Conventions

A key to keeping code understandable, and thus maintainable, is to adhere to a set of conventions. This is especially important when several people work on the same code, as personal style preferences are bound to differ. During the project it was tried to follow certain practices to keep the code readable and consistent. The main guideline was first and foremost the Unity coding standards, but also Microsoft's C# Coding Conventions [43][40]. Although these were used as guidelines, it was not found necessary to keep a strict adherence to these.

3.5 Implementation

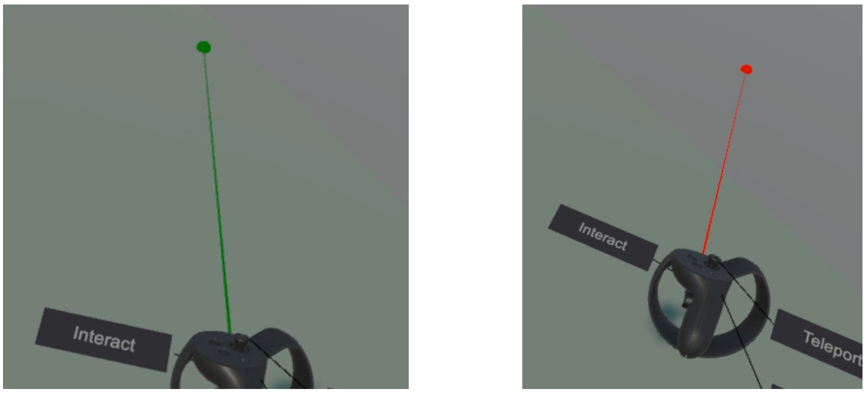
This section will describe how VirtSort has been implemented, as well as how various parts of the implementation function.

3.5.1 Functionality

To successfully use and navigate the application, three functionalities are implemented. Teleportation for movement, picking up and holding object to perform the algorithms, and pushing buttons to select various options.

Teleport

To move around the different areas of the application, the user can teleport, which has become the de facto method for transportation to minimize motion sickness. Teleportation is performed by holding down the thumbstick (Rift) or touchpad (Vive). This will display a guiding line that shows where the player will be teleported to, and its color will change depending if it is a valid area or not. Green is valid, red invalid, as shown in figure 3.4. To stop the player from moving away from the intended areas, valid teleportation is restricted to just a small area or specified points.



(a) Legal teleportation **(b)** Illegal teleportation

Figure 3.4: Teleportation guiding lines

Holding Objects

Picking up, holding and dropping objects is essential to perform the algorithms in the application. To pick up an object, the player must hold the controller inside the object to pick up (there is no collision between the controller and the objects) and push down the rear trigger (Rift and Vive). The object will be held as long as the trigger is held, unless the object is somehow hindered, for example by some piece of geometry. To drop the object, the user simply releases the trigger.

Pushing Buttons

To select various options throughout the application, pushable buttons were added. They can be pressed by touching the button with the controller and pressing the trigger button. The button changes color to bright yellow to indicate that the controller can interact with it, and will revert to its original color when the controller is moved away.

Controller Tooltip

The controller tooltip is a graphical component that explains the purpose of the buttons on the controller. The tooltip used in the application is a component included in the VRTK package, and consists of a text field over a colored square, with a line pointing from the text to the corresponding button. The implemented tooltip contains two fields, *Interact* and *Teleport*, which point to the trigger and thumbstick/touchpad respectively. Because long texts may be ignored, a decision was made to keep explanations short and concise. One problem with the tooltip component is that the lines may pass through the controller to reach its button, and thus may cause confusion, as seen in figure 3.5. As this was not a critical component of the application, and the underlying mechanics of the tooltip component are quite complex, not a lot of time was dedicated to resolving this issue as other issues took priority.

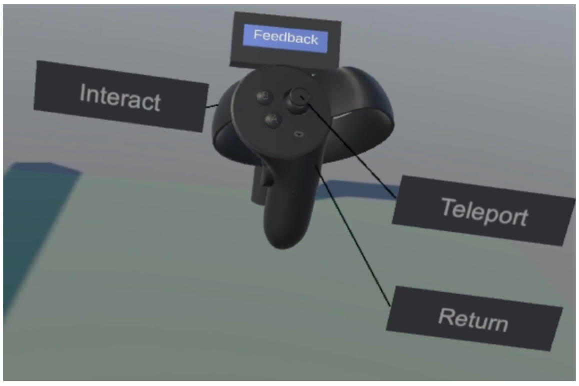


Figure 3.5: Controller tooltip

3.5.2 Prefabs, Objects and Materials

To make development faster and easier, several prefabs were made, along with various other objects. The most important of these were of course the Sort Boxes and Sort Box triggers, as well as the various types of buttons.

Sort Box

The Sort Box is the essence of the sorting algorithms and is represented by the *R_SortBox* prefab. The prefab consists of a *Cube* mesh, with six text meshes attached, one for each face of the cube, where the text shows the value assigned to the Sort Box.

To make the box grabbable it contains the *VRTK_Interactable Object* component from the VRTK package, which has an option *Is Grabbable*, which was set to true. Other relevant options include *Hold Grab Button To Grab*, which makes it necessary to hold down the trigger to keep holding the box, and *Stay Grabbed On Teleport*, which lets the player bring the box along when he teleports. All this not only allows the user to pick up the box, but will also highlight it with yellow when it is touched.

While the Sort Box is represented by a single cube, it actually contain two different cube meshes - one called Normal and one called Highlight. These cubes have different materials applied making them different colors, but are otherwise identical, as shown in figure 3.6. The purpose of these meshes is to either show the normal version or the highlighted version to for example indicate which boxes are currently grabbable and which are not. This highlighting must not be confused with the highlight applied by the *VRTK_Interactable Object* component mentioned above, which will be applied no matter what mesh is currently active.

The box also supports three different visibility modes for displaying the value. *Visible* makes the numbers visible at all times, and is intended for beginners. *Partial* makes the numbers visible only when the box is picked up, and *invisible* never shows the numbers, which are intended for more experienced users. To make sorting possible even when numbers are invisible, when two boxes are picked up, the one with the lowest value will be scaled down to 60% of its original size. This is handled by the *R_BoxResizer.cs* script,

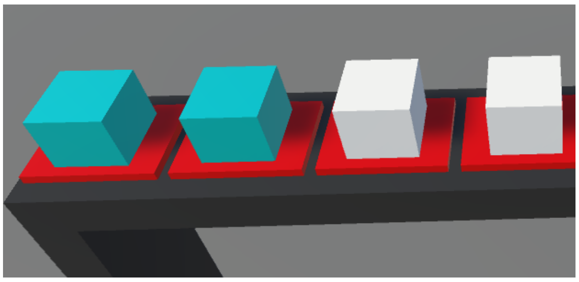
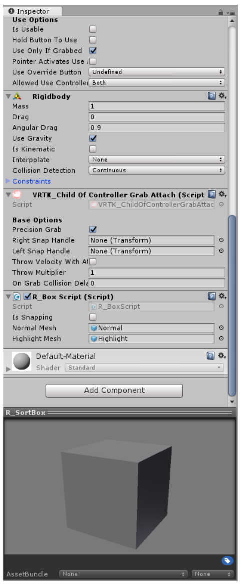


Figure 3.6: Boxes with (left) and without (right) highlighting, placed on SortBoxTriggers

which is part of the *Manager* prefab.

To implement the functionality of the Sort Box a custom script, *R_BoxScript.cs*, was written. Most important of all is the 'value' variable, which is an integer that holds the assigned value from the algorithm. The script also stores values concerned with the meshes, highlighting, scaling, number visibility, snapping and grabbing.



```
Inspector
Use Options
Is Usable                     [ ]
Hold Button To Use            [ ]
Use Only If Grabbed           [✓]
Pointer Activates Use         [ ]
Use Override Button           Undefined
Allowed Use Controller        Both

Rigidbody
Mass                          1
Drag                          0
Angular Drag                  0.9
Use Gravity                   [✓]
Is Kinematic                  [ ]
Interpolate                   None
Collision Detection           Continuous
Constraints

VRTK_Child Of Controller Grab Attach (Script
Script                        VRTK_ChildOfControllerGrabAttac

Base Options
Precision Grab                [✓]
Right Snap Handle             None (Transform)
Left Snap Handle              None (Transform)
Throw Velocity With At        [ ]
Throw Multiplier              1
On Grab Collision Dela        0

R_Box Script (Script)
Script                        R_BoxScript
Is Snapping                   [ ]
Normal Mesh                   Normal
Highlight Mesh                Highlight

Default-Material
Shader                        Standard

Add Component

R_SortBox

AssetBundle                   None        None
```

Figure 3.7: Components of *R_SortBox* prefab

Sort Box Trigger

The *R_SortBoxTrigger* prefab is a flat, colored square that represents a position in the array to be sorted, and shows its current state during solving of an algorithm. States are represented by different colors, which are implemented by using different materials. It also keeps track of which box, if any, is currently inside its box collider. The triggers can also be set to the state of Locked, where any box inside them will be ungrabbable. The Box Trigger itself does not handle its states however. As the trigger state is dependent upon the algorithm, the state is not decided internally in the Sort Box Trigger. Instead, a public API is available for the algorithm class itself to check the local variables and set its state. All Box Trigger functionality is implemented in the *R_TriggerScript.cs* script.

Listing 3.1: Example of *R_TriggerScript.cs*'s public API – The *isLocked* variable.

```
private bool isLocked = false;
public bool IsLocked
{
    set {
        isLocked = value;
        foreach (GameObject box in boxesInTrigger)
        {
            if (isLocked)
            {
                box.GetComponent<R_BoxScript>().IsGrabbable = false;
                box.GetComponent<Rigidbody>().isKinematic = true;
            }
            else
            {
                box.GetComponent<R_BoxScript>().IsGrabbable = true;
                box.GetComponent<Rigidbody>().isKinematic = false;
            }
        }
    }
    get { return isLocked; }
}
```

Sort Table

The Table prefab (figure 3.8) is where the algorithm is set up and performed. It consist of three pieces of geometry, the legs and table top, as well as an empty GameObject, BoxSpawn (figure 3.9), that help with positioning the boxes and triggers correctly. The table can be resized to fit any number of boxes by resizing the table top towards the right and moving the right leg accordingly. The table itself is not responsible for placing boxes or triggers, but the BoxSpawn provides the leftmost position on the table, and the script responsible for placement can use this to calculate the position of the other boxes and triggers as an offset from this position. How the table resizing and trigger/box positioning works can be seen in *ResizeTable.cs* on GitHub [11].

Buttons

Buttons are used to select various options. To support the different option types, different prefabs were made. Common for every button though, is a script that inherits from the *VRTK_InteractableObject* where the *Is Grabbable* option is toggled off, while *Is Usable* is on. The script will also have to extend the *StartUsing* method to implement its functionality, which usually entails broadcasting an event, which then can be handled in the appropriate object.

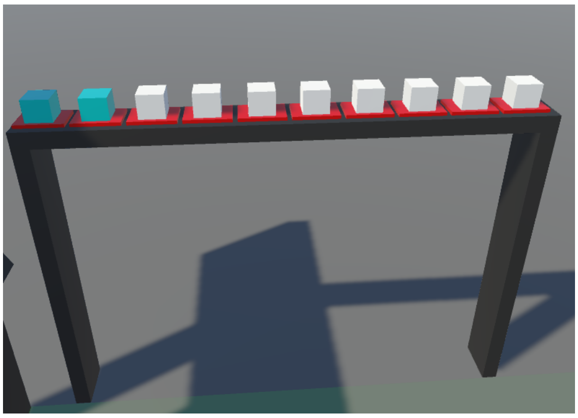


Figure 3.8: SortTable populated by boxes and triggers

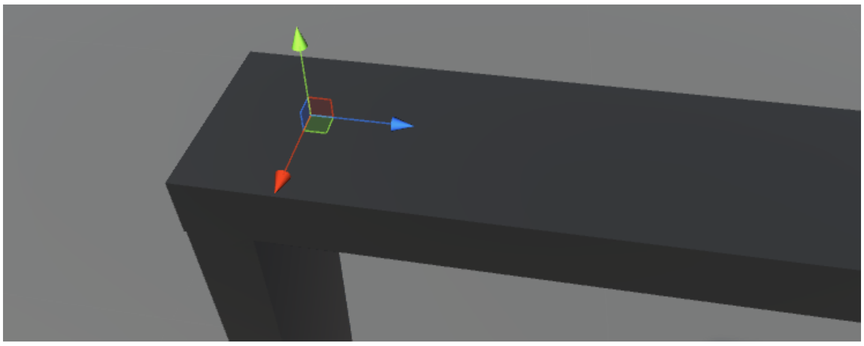


Figure 3.9: The BoxSpawn on an empty SortTable

The *Standard Button*, shown in figur 3.10, is the simplest of the button. It can be pressed to perform some action, like restarting an algorithm. These buttons are used, among other things, for choosing an algorithm in the start area. The prefab *R_Button* implements the Standard Button with the *ChangeAmountScript.cs*, so other buttons have to replace this script with their own.

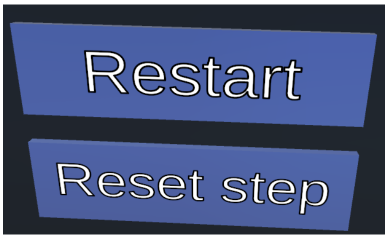


Figure 3.10: Standard buttons

The *Toggle Button*, shown in figure 3.11, is used to toggle a button on and off. To implement this, the *ToggleButton* prefab contains a parent meshless GameObject with two child objects - a green button and a red button with a *TextMeshPro* GameObject saying ”on” and ”off” respectivly. The parent object contains the *ToggleButton.cs* script and is standard for every instance of *ToggleButton*. This script handles the toggling, that is enabling and disabling, the correct child object based on the option state. To implement specific bahaviour for each instance, the child objects must each contain a component that inherits from the *ToggleButtonScript.cs* script and overrides the *Broadcast()* method. The method is then called from the parent object each time the option is toggled. There are two instances of the Toggle Button in VirtSort, for toggling box highlighting and trigger colors in the sorting area.

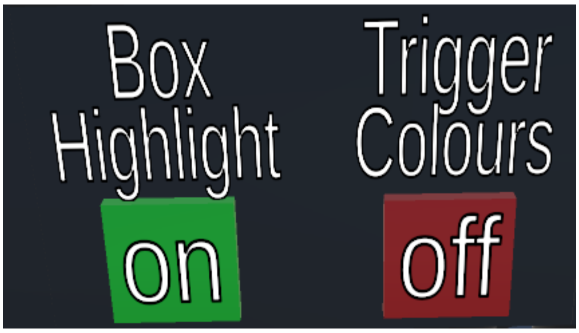


Figure 3.11: Toggle buttons

The *Radio Button*, shown in figure 3.12, is the last type of button, and allows the selection of one of several options. Like the Toggle Button it contains a parent object, the prefab *RadioButton_Manager* that consists of the *RadioButtons.cs* script. This script contains an array that holds an arbitrary number of *RedioButtonScript.cs* scripts, which includes the *RadioButton_Button* prefab. This prefab, like the Toggle Button, contains two different meshes. All buttons in the *RadioButtons.cs* script’s array has the blue *Idle* mesh activated, except the button corresponding to the currently selected option, which shows the green *Selected* mesh. The Radio Button has two instances in the application, one for selecting number of boxes to sort (5, 10, 15) and one to select visibility mode (Visible, Partial, Invisible) for box values.

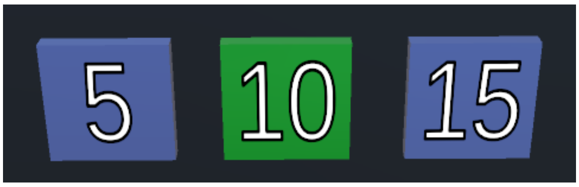


Figure 3.12: Radio buttons

Game Manager

The *Manager* prefab is responsible both for setting up the algorithm, handling events and otherwise make sure the application runs as intended. It is the Manager that initializes the Sort Table with appropriate size, as well as the Sort Boxes and Sort Box Triggers. References to these objects are then passed to the algorithm class, where their states can be read and modified directly. By handling the boxes and triggers, the Manager prefab is also able to handle events related to them. For example, to handle a Box Highlight event, the Manager can simply loop through all boxes and enable highlighting for each one. The Manager is encapsulated in another object called *Game Manager* that also handles other things like loading the instructional videos.

Listing 3.2: Event subscription and unsubscription in *R_Manager.cs*

```
private void RegisterEvents()
{
    NumberVisibilityButtonScript.NumberVisibilityButtonEvent +=
        HandleToggleVisibility;
    RestartButtonScript.RestartButtonEvent += HandleRestart;
    ResetStepButtonScript.ResetButtonEvent += HandleReset;
    ChangeAmountScript.AmountButtonEvent += HandleChangeAmount;
    TriggerColoursButtonScript.TriggerColourButtonEvent +=
        HandleTriggerColour;
    BoxHighlightButtonScript.BoxHighlightButtonEvent +=
        HandleHighlighting;
    ReturnButton.ReturnButtonEvent += HandleReturn;
}

private void DeregisterEvents()
{
    NumberVisibilityButtonScript.NumberVisibilityButtonEvent -=
        HandleToggleVisibility;
    RestartButtonScript.RestartButtonEvent -= HandleRestart;
    ResetStepButtonScript.ResetButtonEvent -= HandleReset;
    ChangeAmountScript.AmountButtonEvent -= HandleChangeAmount;
    TriggerColoursButtonScript.TriggerColourButtonEvent -=
        HandleTriggerColour;
    BoxHighlightButtonScript.BoxHighlightButtonEvent -=
        HandleHighlighting;
    ReturnButton.ReturnButtonEvent -= HandleReturn;
}

private void Awake()
{
    // Set up event handling
    RegisterEvents();

    /*

    ... code ...

    */
}

void OnDestroy()
{
    //Deregister events, otherwise causes MissingReferenceError due to
        calling non-existent subscriber.
    DeregisterEvents();
    SaveState();
}
```

Options Panels

There are several *Options Panels* throughout the application, as shown in figure 3.13, and are basically a panel with some buttons, and sometimes with some explanatory text. Outside of the panel to choose an algorithm in the start area, the different options panels are featured in the sorting algorithm area, with options such as: *Restart*, restarts the algorithm with new values; *Reset Step*, resets the step to the beginning of the current step; *Number of Boxes*, decides the number of boxes to sort: *five*, *ten* or *15*; *Number Visibility*, choose visibility for box values, *Visible*, *Partial* and *Invisible* as explained in section 3.5.2; *Return to Start*, returns the player to the start area; *Trigger colors*, toggle coloring on Box Triggers; *Box Highlight*, toggle highlighting on boxes; and *Play Video*, play a video explaining the current algorithm.

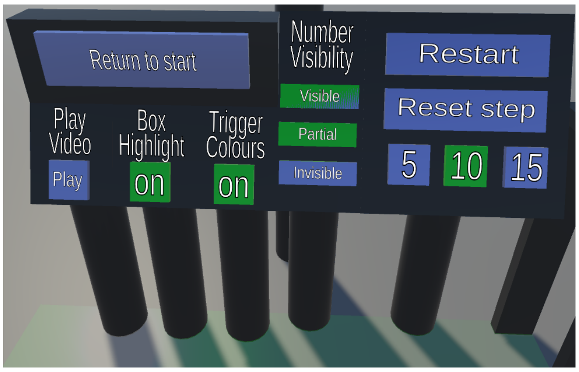


Figure 3.13: Options panels

Intructions Wall

The *Instructional wall*, shown in figure 3.14 was initially intended to show explanatory text for the user, like showing instructions for the algorithm and explaining the buttons of the controller. As the controller tooltips were introduced and it was decided to use instructional videos, displaying text instructions was deemed unnecessary, and thus the wall was repurposed to showing the videos. The videos are short instructional videos (figure 3.15) that explain an algorithm in three to five minutes, and were collected from Youtube. To avoid downloading another package to show videos directly from Youtube [62][63][64], and to support offline viewing, the videos were downloaded and imported into the project as assets. This made it possible to utilize the integrated media player, which made setup fast and simple.

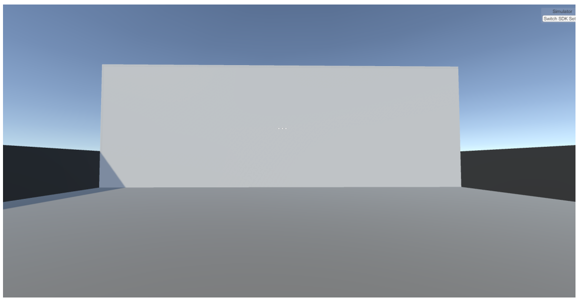


Figure 3.14: InstructionsWall

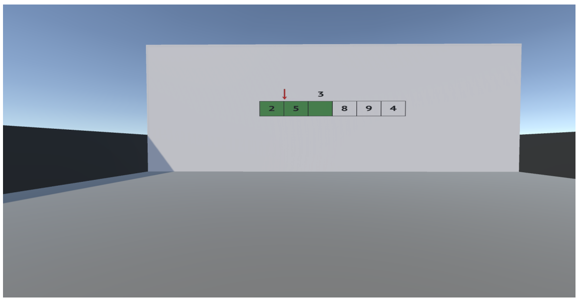


Figure 3.15: InstructionsWall showing a video

Explanation Panel

The *Explanation Panel*, shown in figure 3.16, is another instructional tool to help the player understand the application. It simply displays all variations of boxes and triggers with a short description below. It also states that these are only applicable given that box highlighting and trigger colors are enabled.

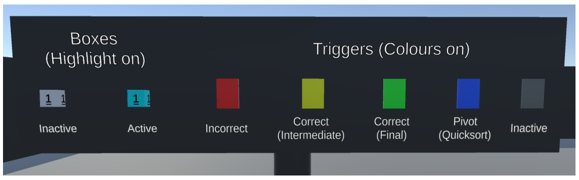


Figure 3.16: Explanation panel

Materials

Different materials were used to differentiate between different states. All materials were pretty basic, with only a difference in color. The different colors are green, representing selection or completion; yellow, representing some intermediate state; red, representing disabled or erroneous; and blue, which have various meanings depending on the context, such as the pivot in quicksort or the element to insert in insertionsort. Other materials were also created specifically to avoid using the standard light grey material and to liven up the surrounding geometry.

Other

There are also some less important objects and functionalities worth mentioning. To avoid losing boxes that are dropped, thrown or otherwise lost, they are teleported back to their last position on the main table. If that space has become occupied, the box is instead teleported to the *Box Respawn Table*. This is a small table besided the main sorting table with space for a single Sort Box, and can be seen in figure 3.17.

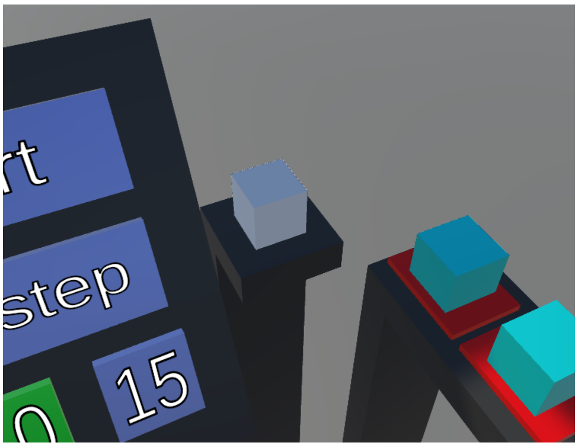


Figure 3.17: Box respawn table

3.5.3 Start Area

The *Start Area* scene, shown in figure 3.18, is the starting point of the application. The player is spawned close to a panel with several buttons, with each one representing an algorithm the player can perform. The player can teleport around the area around the table, and the buttons can be pressed to go to an algorithm.

As this may be the user's first experience with the application, and even with VR, the controller tooltip is displayed beside the controllers to explain the functions of the buttons. The tooltip is shown as long as the player stays in the start area, but disappears as soon as they travel to another. To further help the user getting to know the application, the player is placed some distance away from the algorithm selection panel, forcing them to get familiar with the teleportation function. After teleporting next to the panel, the user must learn how to interact with buttons. Thus, the user is forced to learn two of the three main functionalities before progressing to the sorting area.

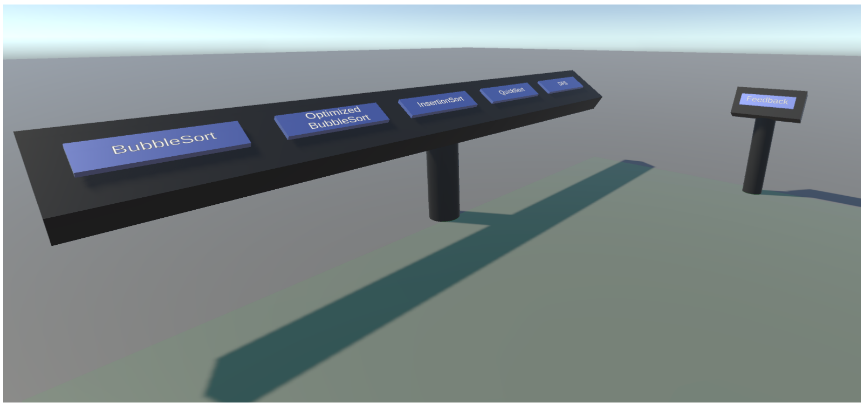


Figure 3.18: Start area

3.5.4 Sorting Area

The *Sorting Area* scene, shown in figure 3.19, is the main area of the application. This is where algorithms are initialized and solved by the user. All sorting algorithms use this room, and it consists of a walled in area with a Sort Table with Sort Boxes and Sort Box Triggers, an Instructional Wall, several Options Panels, an Explanation Panel, a Box Respawn Table, and a small area on the floor to teleport around on. All algorithms can be performed with either five, ten or 15 boxes.

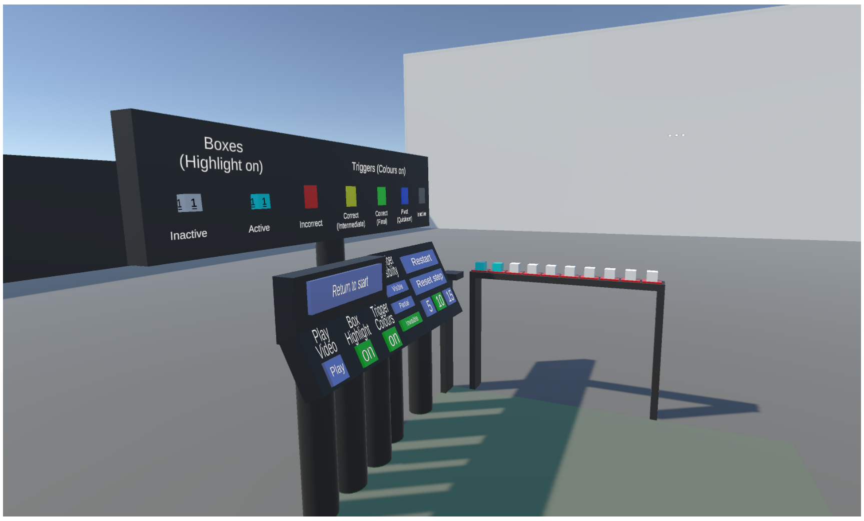


Figure 3.19: Sorting area

Bubblesort

The bubblesort is the simplest algorithm to perform. In the beginning, every trigger is colored red and the two leftmost boxes are highlighted. As you perform comparisons and swaps, the sorted boxes are marked with yellow triggers. This signifies that they have been compared and are correctly positioned for the current iteration of the algorithm. If the optimized bubblesort is chosen, after each iteration, the rightmost trigger is marked green, and will not be considered in subsequent iterations. This is because in each iteration, the largest element in the unsorted array (non-green triggers) is guaranteed to be moved to its correct position on the rightmost non-green trigger. The algorithm is completed when either an iteration is performed without any swaps or, if optimized, the second leftmost trigger is turned green, and will turn all triggers green. The blue trigger color is used for neither standard nor optimized bubblesort.

InsertionSort

Insertions sort is not much different in complexity from bubblesort. The currently sorted list is marked with yellow triggers, starting with only the leftmost trigger. The element to insert is marked with a blue trigger, and will at the beginning of an iteration be the box directly to the right of the sorted boxes. After the rightmost box is correctly inserted into the sorted list, the algorithm is completed, and the triggers all turn green.

Quicksort

Quicksort is by far the most complex algorithm to perform of the three, as it tries to illustrate recursion. For each recursion, the leftmost element is chosen as the pivot element, marked by a blue trigger. The player then have to sort the rest of the boxes into two lists of values smaller and larger than the pivot, with smaller boxes closest to the pivot and larger boxes furthest away. When this sorting is done, the rightmost of the smaller boxes is swapped with the pivot, placing the pivot between the two lists, with smaller values on the left and larger on the right. The pivot is now correctly positioned, and the trigger is turned green. The algorithm is then run recursively on the two partitions, first the list of smaller values, then larger values. All boxes not part of the current recursion is turned invisible to make it clear which boxes are currently being sorted. A base case is reached when either one or zero boxes are to be sorted.

3.5.5 DFS Prototype

The original intent of the project was to create scenes for a variety of algorithm types, including tree search algorithms. Thus a prototype was made for a depth-first search to explore how to effectively teach the algorithm, and to eventually create a functioning tree search algorithm scene with support for various algorithms such as depth-first and breadth-first search, iterative deepening depth-first search and beam search. As the project focus changed, as mentioned in section 3.4.1, only this prototype was created, which simply hard codes a single DFS case.

The prototype consists of several almost identical rooms. Each room represents a node in the search tree, which is physically traversed by the player by visiting each room. To traverse the tree, each room contains a door that leads back to its parent node. On the opposite wall are a set of doors leading to each of the current node's children. Unlike in the sorting algorithms, where the player can teleport around a certain area on the floor, here the player is bound to a certain spot in the center of the floor.

To move between rooms the player must teleport to the desired door by holding down the teleport button, aim the teleport pointer line at the door, and let go of the button. If the door is open, both the door and the line will turn green, and the player will be teleported to the corresponding room. Likewise, if the room is locked, the door and line will turn red. There are also lights on each door indicating the state of each room, such as green for open and unvisited, yellow for visited but not completed (all child nodes not visited), and red for visited and locked (when a room is visited and it and none of its children contain the target value).

To further help navigation, each room contains a map of the search tree above the child doors. Each node on the map is colored according to its state. Red and yellow matches the meaning of the door lights, while blue is the current node, grey are unvisited nodes, and green is the target node (when found). The current node also has a *"You are here –>"* text field next to it.

The player's goal is to find the room with the target value. The player can intendify a room's value by observing the large letters displayed on the two of room's walls, as shown in figure 3.20.

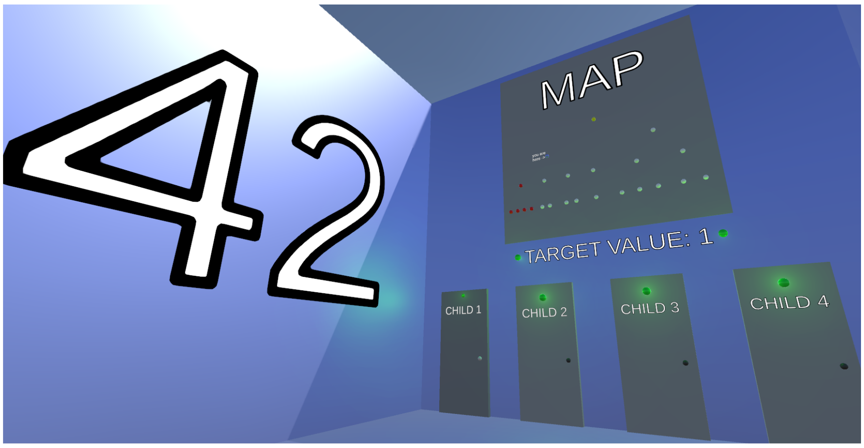


Figure 3.20: Depth-first search prototype room

3.6 Evaluation

To evaluate the usefulness of the application, a decision was made to perform a round of usability testing. Usability testing is a method that lets real users test the application and is meant to evaluate the usability of the application. This is important, as the developers, due to deep familiarity with the application, are often blind to faults in usability. What has become intuitive for the developer might not be intuitive for the end user. The testing was performed on in total twelve students, where some had taken to Algorithms and Data Structures course, and some had not. In this section, the testing will be explained in detail.

3.6.1 Testing Methodology

The testing session was performed on the finished application on 2018-05-16, on campus at Hackerspace in *Drivhuset*, in the south wing of *IT-bygget*. The students had signed up to allocated time slots beforehand, and were given ten minutes to explore the application. Due to the time constraints, and the focus of the project being the sorting algorithms, the DFS prototype room was removed from the selection panel in the start area to avoid confusion. The testing was performed on two different Oculus Rift systems, each with two Oculus Sensors connected. While one system had both sensors in the front, the other had one in front and one in the back. Minor differences could be noticed in the tracking accuracy, especially when the tester was turned backward, but not to a degree that interfered with the testing experience.

The test began by asking the tester whether they had used VR before. This was important, as it would determine how much details were given in the explanations to the tester. Although it was originally intended to let the tester ”fly blind”, that is, not receive any additional guidance beyond what is given in-game, the time constraints forced the policy to be changed to ”give help when asked”. This let the testers progress through the application without longer breaks due to not knowing what to do. This also made it easier to evaluate what the testers were struggling with, as they were forced to explain what they were struggling with or what they did not understand. The exception to this policy was for testers who tried quicksort. Here the specific quirks and differences of the implementation was explained in comparison to the in-game instructional video explaining the algorithm. This was also done for the other algorithms for the testers with no prior algorithm knowledge, as the videos were too long to watch in their entirety, given the time constraints.

When the testing was over, the participants were asked to fill out a questionnaire based on the System Usability Scale (SUS), which is a standard tool for measuring usability [26]. The SUS contains ten statements about the application, with the ability to give a score to each from *0 - strongly disagree* through *4 - strongly agree*. The questionnaire on the other hand contained values from one through five, but these were easily converted to SUS-equivalent values by subtracting by one. An additional question was added that asked whether the participant had completed the Algorithms and Data Structures course or not. As the course is held in the autumn semester, there was no need to distinguish students that were currently taking the course.

Chapter 4

Results

This chapter will present the working prototype of VirtSort that was developed as a description of functionalities that a user might experience while using the application. This is followed by a brief presentation of the test session.

4.1 The Application

The VirtSort application contains two complete scenes - the start area scene and sorting area scene, with an additional DFS prototype scene and a feedback scene used during user testing. A video was recorded exploring the functionality of the VirtSort prototype, and can be found on Youtube [33]. The full source code has also been made available on GitHub for the readers convenience [11].

4.1.1 Start Area

The first thing the player is presented with when starting the application is the start area. Here, the player starts facing a wide panel with five buttons on them - *BubbleSort*, *Optimized BubbleSort*, *InsertionSort*, *QuickSort* and *DFS* - but far enough away to not reach them. If the player looks to the right there another panel with a single button saying *Feedback*.

To reach the panel, the player needs to use the teleport function. To help the player find the button used for teleportation, which is pressing the touchpad or thumbstick for Vive and Rift respectively, a tooltip is visible around the controllers as long as the player stays in this area. The tooltip simply says *Teleport* and points to the buttons on the virtual controllers. To successfully teleport, the player must hold down the teleport button. This will procure a straight line originating from the tip of the controller pointing outwards. The line will change color depending on whether the target location is a valid teleportation spot or not. A valid area results in a green line, while an invalid area results in a red line. The only valid teleportation area in the starting area is a small square on the floor colored slightly greener than the rest of the floor.

When the player reaches the panel, it is time to choose an algorithm. The algorithms are sorted from the simple to complex from left to right, starting with bubblesort. To start an algorithm the player must push the desired button. This is done by holding the controller on the button, which will then turn yellow as long as the controller is in contact with it. As long as the button is yellow, the player can push the button. Then the button labeled as *Interact* by the tooltip must be pressed, and is the index finger trigger for both Vive and Rift.

The player can also move to the feedback panel, typically after trying one or more algorithms, and press the button to enter the Feedback Area.

4.1.2 Sorting Area

If either bubblesort, insertionsort or quicksort is chosen, the Sorting Area scene will be loaded. This area is identical for all algorithms, with the only difference being which algorithm the user is expected to perform. When entering the scene, the player can see three things, the sorting table with boxes on top, a set of option panels, and a panel with explanations of the different types of boxes and triggers.

The standard options are the same for all algorithms, and are as follows: Number of Boxes - *10*, Number Visibility - *Invisible*, Trigger colors - *On*, Box Highlight - *On*, but the player is free to change these options at any time. For a player unfamiliar with the chosen algorithm, the natural choice would be to watch the instructional video. This can be done by pressing the leftmost button saying *Play Video*. This will start a three to five minute video on the instructional wall which the player can see when turning to the right from the options panel. The video gives a short explanation of how the algorithm works, and should give the player enough information to try it out for themselves.

The algorithm is performed solely by picking up and switching the boxes on the table. If highlighting is enabled, the two boxes the player are supposed to pick up are highlighted by a teal color. When two boxes are picked up, the one with the smallest value will shrink in size. In this way, the player is able to determine their relative values, even if their exact values are not shown. If the boxes are the same value, they both stay their original size. Depending on the algorithm and the boxes' relative values, the player must decide whether to switch the two boxes, or if they should stay put.

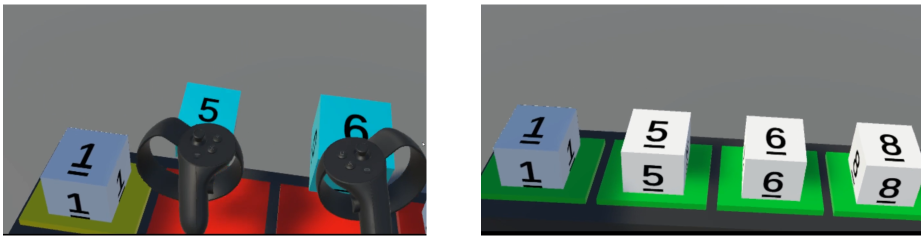
If the player for some reason gets confused in the middle of the algorithm, for example after making a mistake, the *Reset Step* button can be pressed. This makes the algorithm return to the beginning of the current iteration, placing all boxes back at their correct positions.

When the user successfully sort the list, every trigger will be colored green and all boxes will be immovable. The player now has two different choices: Either press the *Restart* button to retry the algorithm with new randomly generated values, or press the *Return to Start* button, which will transport the player back to the Starting Area. Before doing so however, this could be a good time to turn the number visibility to *Visible* to verify that the boxes actually were sorted correctly.

4.1.3 Bubblesort

Bubblesort will be the natural first choice for new players, as bubblesort is the simplest of the three algorithms. Initially, the two leftmost boxes will be highlighted, and the player must compare the two boxes (as seen in figure 4.1a. The larger of the two should be placed to the right, and the smaller one to the left. This operation is repeated for each pair of boxes starting from the left to the right, that is first indices ’0’ and ’1’, then ’1’ and ’2’, then ’2’ and ’3’, and so on. Once the rightmost pair have been compared, one iteration is complete. This is repeated until an iteration contains no swaps, that is for every pair of boxes that are picked up, every pair are put back down at their original position. When this happen, the algorithms is finished, and will be marked as completed by all the triggers turning green (figure 4.1b).

If the optimized version of bubblesort is chosen, after every iteration the rightmost unsorted box will be marked as sorted by turning the trigger underneath green. The sorted boxes will not be included in subsequent iterations, as their position will never change after being sorted. This is because the operations of an iteration is guaranteed to move the largest of the unsorted values to its correct position in a sorted list, which will be the rightmost position of the unsorted list.



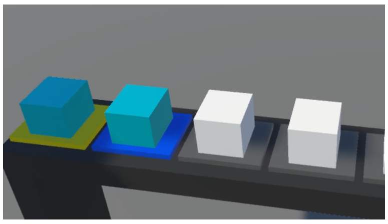
(a) Bubblesort in progress – comparing the second pair of boxes

(b) Bubblesort solved

Figure 4.1: The bubblesort algorithm

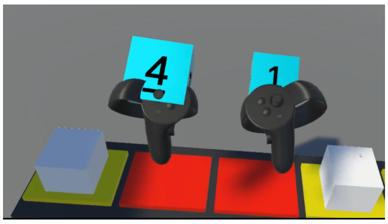
4.1.4 Insertionsort

The natural next step from bubblesort is insertionsort. Here the player is supposed to place, or insert, boxes one at a time into an already sorted list containing a subset of the boxes. The sorted list, marked by yellow triggers, will always be on the left of the table, starting with just the leftmost element. The element to be sorted into the list will always be the box immediately to the right of the sorted list, and is marked by a blue trigger (new from bubblesort). The player sorts the box by comparing it with the box to its left. If the unsorted box is smaller than the sorted box, they switch place. This operation is repeated until a sorted box is equal to or smaller than the unsorted box, at which point the unsorted box is placed in that position. All boxes to the right of the unsorted box will be larger, while all boxes to its left will be smaller, meaning the box is now correctly inserted. This process repeats until all boxes are inserted, and the array is sorted, turning the triggers green. These steps can be seen in figure 4.2.

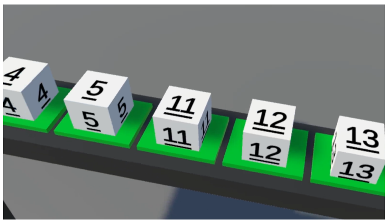


(a) Starting position – One element (yellow) is the sorted list, one element (blue) to be inserted.

(b) A box in the middle of insertion.



(c) The insertion box (right) compared with a box in the sorted list (left).



(d) The boxes have been successfully sorted.

Figure 4.2: Steps of the insertionsort.

4.1.5 Quicksort

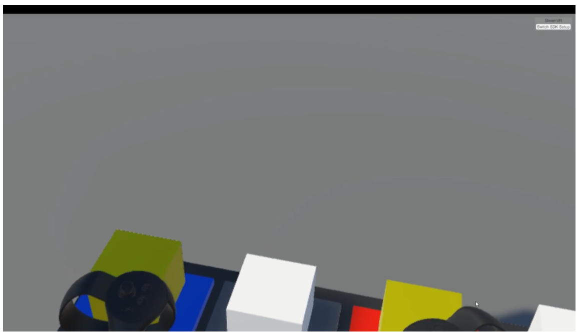


Figure 4.3: Quicksort – Comparing the pivot with the left index.

Quicksort is by far the most complicated of the algorithms, both conceptually and execution-wise, especially because of its use of recursion. The players goal for each recursive call, starting with the full list, is to split the list in two based on a pivot element. One list will contain all boxes with values smaller than or equal to the pivot, and the other boxes with larger values than the pivot. The pivot is then, when placed in the middle of these partial lists, or partitions, correctly placed.

In this implementation the pivot will always be chosen as the leftmost box in the current partition, and is marked by a blue trigger. The player then compares the pivot with every

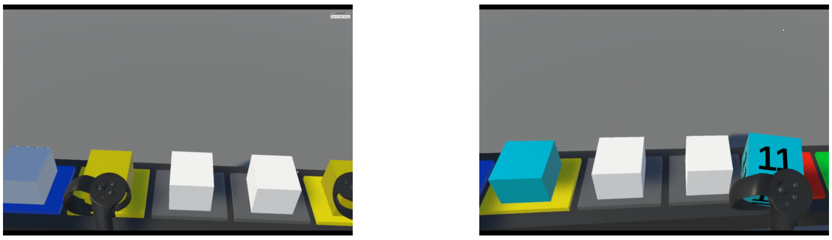

(a) Locating the left and right index (b) place the left box on the right trigger

Figure 4.4: Quicksort – Swapping the left and right index

box starting from left to right, marked by a yellow trigger (figure 4.3). If a box is larger than the pivot, the player puts the box back down and the triggers stays yellow. Then the pivot is compared with every box from right to left, also marked by a yellow trigger, until the player finds a box with smaller or equal value to the pivot. When this box is put back down, it too will be marked a yellow trigger. Now the two marked boxes are swapped, moving the larger box to the right side, and the smaller or equal to the left, as illustrated in figure 4.4. This operation of comparing the pivot with boxes from the left and the right is performed until a box that has already been swapped, or is at the end of the list, is reached. Now the list is sorted with the pivot furthest to the left, followed by every box with smaller or equal value, and finally every box with larger value. The last step is then to swap the pivot and the rightmost of the boxes with smaller or equal value, which is marked with a yellow trigger. The pivot is now sorted and marked with a green trigger, with all values smaller or equal to the left left, and all values larger to the right.

After this is done, the algorithm is applied recursively on each partition, first the one to the left of the pivot, then the one to the right. When a recursive algorithm is performed, all boxes that is not part of that partition will turn invisible, as shown in figure 4.5, making it clear which boxes the algorithm is currently applied to. The recursion is only stopped when the partition contains one or zero elements. When there are no more partitions to solve, the sorting is complete, and the all triggers are colored green.

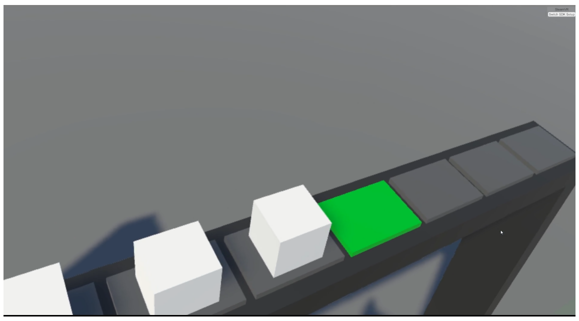


Figure 4.5: Quicksort – Boxes not part of the current recursive partition are made invisible

4.1.6 DFS Prototype

In the prototype the player is placed in the middle of the algorithm, in the first child of the root node. The player's goal is then to find the target value, in this case *1*. As the prototype uses hard-written values, there are no differences between subsequent executions. The search tree structure, the node values, target value, and the starting point will always be the same. The starting point of the prototype is in the middle of the algorithm, more specifically at the first child of the root node. The subtree of its first child has already been explored, and did not contain the desired value.

The player then continues by visiting the current node's second child. This is done simply by using the teleport ability by aiming at the door of the desired node. Each room may contain one door on the back wall leading back to the node's parent. The door is marked by a yellow light, and the text *Parent*. On the opposite wall, the front wall, there are doors to child nodes. Each door is marked by a green or red light, and the text *Child #*, where *#* is the child number. A green light indicates that the child node has yet to be explored, while a red light indicates that the child node and all its children have been explored, and did not contain the target value.

The first room the player moves to holds the value *13*. The player continues by visiting the first child of this room, a leaf node with value *55*. As the player moves around, a map is present on the front wall, above the child doors. Here small lights representing the nodes in the tree can be seen, and their colors represent the state of each node. The colors are as follows: *grey* nodes are unvisited nodes, *red* nodes are visited nodes without the target value, the *yellow* node is the root node, a *green* node a visited node containing the target value, and the *blue* node is the current node. The current node also has some text next to it saying *You are here* followed by an arrow pointing to the node.

After returning through the *Parent* door, the player visits the second child, which contains the target value. The room turns green, and the text *"SUCCESS!"* is displayed on the walls, as seen in figure 4.6. To return to the start area the player must press the *Return* button on one of the controllers, as there is no virtual button that can be pushed.

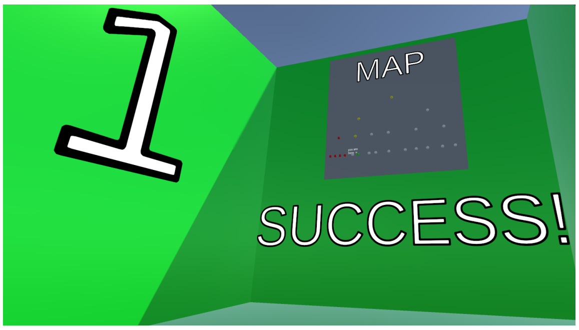


Figure 4.6: The final room of the DFS prototype

4.1.7 Feedback Area

After trying the different algorithms, players during the testing phase could leave feedback by entering the *Feedback Area*. This is reached by pressing the *Feedback* button (figure 4.7) in the *Start Area*, which is found on a separate panel to the right of the algorithm selection panel.

Figure 4.7: The feedback button

When entering the feedback area, seen in figure 4.8, the player will face a panel with a wall behind it. The wall displays each question of the questionnaire in turn, while the panel contains buttons representing the alternatives for each questions. For the first question, a *Yes* and *No* button will appear, while for every other question buttons representing the numbers one through five appear. To the left, above the *1* button, is some text saying *Strongly disagree*, while *Strongly agree* is written above the *5* button. When all questions have been answered a button appears that lets the player return to the start area.

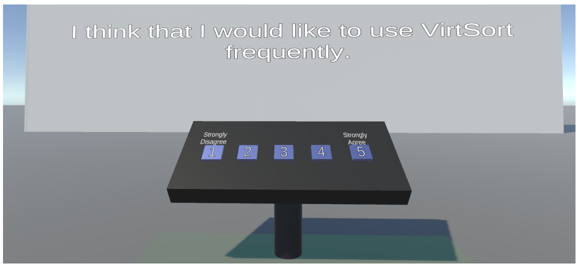


Figure 4.8: The feedback area

4.2 Evaluation

The user testing resulted in twelve responses to the questionnaire. Of the twelve participants, nine had completed the TDT4120 – Algorithms and Data Structures course, leaving three without prior knowledge to its contents. This does of course not exclude them from having any knowledge of algorithms in general.

The testing was completed successfully, with every participant getting to spend at least the ten minutes allocated. Furthermore, all participants also got to try at least one algorithm, sometimes two. Bubblesort and insertionsort were the most tested algorithm, but quicksort also got tested by a few participants. A case of motion sickness was also observed.

All participant answered the questionnaire, and the full results can be seen in appendix B. The results will be discussed in the chapter 5.

Chapter 5

Discussion

In this chapter the test results from section 4.2 will be discussed first. Afterwards, the prototype as a whole will be evaluated against the requirements detailed in section 3.3. Finally, the research questions described in section 1.3 will be brought up again, this time to see whether or not they can be answered.

5.1 Test Evaluation

5.1.1 Test Scores

To compute the SUS score, each answer given is summed, and then multiplied by *2.5*, which gives a final score between 0 and 100. As the questions given here were scored from one through five, they were first subtracted by one before being multiplied. This resulted in an average SUS score of 60.625, where 68 is considered average [26]. If the participants are divided into those who had and those who had not taken the algorithms course, those who did ended with an average score of 58.8, while those who did not had an average score of 65.8.

With answer values ranging from one through five, a score of four and above should be considered good. There were in total two questions that reached an average score above four. These were *I think that I could use VirtSort without the support of a technical person* with an average of exactly 4.0, and *I thought there was a lot of consistency in VirtSort* with an average of 4.08. Also worth mentioning is *I found the various functions in VirtSort were well integrated* with an average score of 3.9.

On the other hand, a value below three should be considered bad, but no questions received such a low average score. The lowest scoring questions though were *I found VirtSort to be simple* and *I think that I would like to use VirtSort frequently* with average scores of 3.25 and 3.17 respectively.

5.1.2 Looking at the Answers

Looking at the survey questions, some conclusions may be taken from the responses. The full results can be found in Appendix B

I think that I would like to use VirtSort frequently

As can be seen, the majority of testers neither agreed nor disagreed with the statement. This can indicate that they see the potential of the application, but not necessarily in its current form. It is also important to note that a majority of testers have already had the course ”Algorithms and Data Structures”. This means that they are expected to already know the concepts taught in VirtSort, and thus may not find the application as useful. Interestingly, the only tester who responded with a 5, or ”strongly agree”, did not have the course yet, although the other two responders without the course still responded with a 3.

I found VirtSort to be simple

Responses here seem to vary between slightly disagreeing and slightly agreeing. It is hard to discern whether this was mainly due to struggling with VR in general, or whether its due to shortcomings of the application, but both aspects certainly have at least some part to play.

I thought VirtSort was easy to use

Majority of responses here agreed that VirtSort was easy to use. This indicates that the various setting panels were well laid out and labeled, and the purpose of the application was well presented within the application itself. It can be speculated that due to the test environment having instructors guiding the test subjects through the application, they may have different experiences of how easy it is to navigate the virtual world. Though looking at the next question, this speculation might not be as likely as initially thought.

I think that I could use VirtSort without the support of a technical person

Responses here were also almost all leaning towards agreeing with the question. This supports that the application was indeed laid out in a logical manner for the user. It also indicates that the purpose of the application is evident from the contents within, or at least from only knowing the basic intent of the program.

I found the various functions in VirtSort were well integrated

This was again a question that scored highly. As the application is intended for learning the concepts of various sorting algorithms, it is important that the functionality is not only working, but also correct. In this case, the application worked as expected where the testers would play through various algorithms in a step by step manner.

I thought there was a lot of consistency in VirtSort

The responses yet again lean towards agreeing with the question. While users can choose algorithms on their own, the scene that they are moved to look identical. This means that if they understand how to interact through one algorithm, they also understand the interactions for all algorithms. One thing to note here, however, is that some testers expressed confusion around the coloring of the panels during quicksort, as it used the blue panels in a different way compared to the one used during insertionsort. This is an inconsistency that should be fixed in future iterations.

I would imagine that most people would learn to use VirtSort very quickly

The responses here indicate that VirtSort was relatively easy to understand for most test subjects. Since the application is secondary to the education, this is ideal as the user's focus can be spent on the algorithm itself. Part of this might also be the effect of the intuitive nature of VR. Even though many users expressed that they had difficulties performing the quicksort algorithm correctly, those concerns were more specific to the feedback given during that algorithm, rather than the mechanisms behind the application in general.

I found VirtSort very intuitive. I felt very confident using VirtSort

This was a question with somewhat mixed responses. While it was still overall positive, it is in no way as high as one would hope for an application intended for learning. As it already utilizes the inherent intuitiveness of VR controls, there is a possibility that the application itself is not nearly intuitive enough. However, from verbal feedback from testers, which will be discussed in more details in section 5.1.4, the main point of concern were with the confusing nature of quicksort. Even still, future development should also focus to improve the intuitiveness of the application, as many functions are crammed on various panels with loose functional grouping.

I could use VirtSort without having to learn anything new

This question can be interpreted in various ways. Therefore, having some mixed responses may have been inevitable. It can be read as using the application itself, in which case for some testers, they would have to learn to use VR technology. If they responded with the algorithms in mind, then they would also have to learn the algorithms to be able to perform the tasks correctly. This also means that any test subject with experience in both VR and TDT4120 would not necessarily need to learn anything new to use VirtSort. Considering the general profile of the test subjects - mostly computer science students - it should come as no surprise that while the responses are fairly spread out, they still tend towards agreeing.

5.1.3 Observations from the Test

It was made evident during the test that the application did not function as well without a proper room-scale environment. many testers tried to press buttons that were simply out of the active play area, which resulted in loss of tracking. While one way to fix this is to just require more space, it is not always feasible if the intent is to use the application on a larger amount of students. Rather, this observation could indicate that a dynamically scaling play area, similar to one found in games such as *Job Simulator* [15] could be beneficial. Doing this could also eliminate the need for the teleportation mechanic entirely, and rely solely on movement in the physical space environment, thus reducing additional steps that the user must learn. This could also drastically improve the immersion aspect of the application, as some users felt a disconnect when they had to teleport through the virtual world.

During the test, four algorithms were made available: bubblesort, optimized bubblesort, insertionsort and quicksort. After the first few test subjects had finished their session, it was clear that the regular bubblesort algorithm was simply too slow for the strict schedule on test day. While it is the simplest algorithm and is also a good way of illustrating how seemingly small changes can have vast improvements in run time, it was not ideal for the test. Thus, for later testers, they were told to skip the bubblesort algorithm entirely, which did not seem to have any noticeable effect on their ability to perform the other algorithms. On the contrary, some testers expected the regular bubblesort algorithm to be the optimized version, and were left confused when indicators did not light up as they expected.

5.1.4 Feedback from Test Subjects

Additionally, some testers gave verbal feedback after the test about their impressions of the application. These concerns will be addressed here.

Many testers stated that they first and foremost wanted to review their knowledge of various algorithms. This makes sense as a majority of the testers had previously completed the course TDT4120. During the tests, they reported that it was very simple to watch a video explaining the basics of bubblesort and insertionsort, before performing the algorithm themselves. This indicates that VirtSort can be effectively used as a reviewing tool for students who might have forgotten details over time. However, this response was not the same for the quicksort algorithm. Here, testers reported that it was much more confusing to properly follow the algorithm step by step. They also reported that without the use of the panel indicators or the instructors explaining how the application functioned, they may not have been able to perform the task correctly. In this case, written guidance could have been a possible solution. The current application relies solely on colors to indicate different states, which can be confusing when there are several states to keep track of. Furthermore, the video the user can watch is only an explanation of quicksort in general; it does not relate back to the objects within the game world. For bubblesort and insertionsort, the algorithms were simple enough that it was not a problem, but that was not the case for quicksort. Thus, creating tailor-made videos, or perhaps real-time animations showing off the algorithm in action could be more effective. In addition, all testers did the quicksort algorithm while not having numbers visible on the boxes. As it was confusing to keep track of what the algorithm was doing, further testing with numbers visible should be done.

One tester expressed that while the core gameplay loop of watching a video of an algorithm, then sorting an unsorted list using the algorithm was good, they felt a lack of feedback from clicking buttons, specifically the play button for the video. While other buttons in the application changed colors to indicate their state, the play button did not.

There was also no indication as to where the video would be playing, leaving the user confused as to whether or not anything happened after interacting with the button. This was an oversight during development, and should be addressed in future development of the application.

Another tester reported feeling nausea during testing. They had previously never used Virtual Reality before, making this their first time experiencing high-end VR. This was a concern discussed in section 2.3.2. To mitigate the effects of VR sickness, they were told to close their eyes and take it easy, and the testing period was cut short due to health concerns.

The cause of this can be several. As they had never used a virtual reality headset before, it could just be the result of inexperience with the technology. The application requires the user to move around in a 3D environment, and also teleport around, which can be disorienting. It is usually recommended that first time VR users start with seated experiences before moving on to more intense ones. It may also have been caused by the test session itself, as it was held in a room with less than ideal space. This meant that the user did not have full room-scale movement opportunities, and had to use the teleportation mechanic more often than one would expect. Additionally, the test was conducted with an Oculus Rift and only two tracking sensors in the front. This meant that there was only 180° tracking during the test. The sensor placement also meant that some spots could have tracking issues. With all these factors, the likelihood of a tracking error was fairly high, and could have been the cause of the initial nausea the user felt.

5.2 Evaluating the Feature Requirements

In this section the feature requirements detailed in tables 3.1 and 3.2 will be discussed.

5.2.1 Functional Requirements

- **FR1** - This requirement states that some sorting algorithms have to be implemented. In the prototype, the original bubblesort from the specialization project was improved upon, and both insertionsort and quicksort was implemented.
- **FR2** - Step-by-step progression was deemed important even during the specialization project, and was implemented then. However, the interface had to be rewritten during development to ensure step-by-step progression in general for all algorithms, as described in section 3.4.2.
- **FR3** - This was already implemented using the table and boxes during the specialization project.
- **FR4** - A teleportation mechanic was implemented during the specialization project, and was kept as-is. While it works as intended and gives users the freedom to move around the scenes, section 5.2.2 will discuss whether or not this was the correct approach.
- **FR5** - Further development went into the interactions the user has with the boxes. The user can now change how the boxes are displayed. Furthermore, quality of

life improvements such as snapping and respawning boxes to the table was also implemented.

- **FR6** - Feedback was given in the form of panels on top of the table in the specialization project. These panels were expanded upon further, and various new colors were added to indicate new functions such as intermediate states and selected pivot among others.

- **FR7** - The initial plan was to show written instructions using the instructions wall. As detailed in section 3.5.2, this was eventually scrapped for a video presentation of the algorithm. Whether or not this was the right choice was discussed in detail in section 5.1.3.

- **FR8** - A random problem is generated every time the user clicks the restart button.

- **FR9** - The problem size can now be changed between five, ten and 15 boxes by the user using a panel. This is one of many functions added for the **FR12** requirement as well.

- **FR10** - The implementation of the quicksort algorithm was the intended way of presenting the concept of recursion to the user. During each recursive call of the algorithm, the inactive partitions would be made invisible on the table where the user was sorting. This way, the user would see how the algorithm is doing the same actions, only on smaller and smaller problems until the whole list is sorted.

- **FR11** - During the first scene of the application, labels are shown to the user detailing what each button on the controllers do. While this should certainly be enough for individuals who have had previous VR experience, or even just gaming experience in general, this may not have been sufficient for every user. This is discussed in more detail in section 5.2.2.

- **FR12** - To facilitate for different level of users, the application lets the user set various different settings for their needs, such as the ability to highlight the currently active boxes, or turning off the panel highlighting entirely. This way, the user or instructor has full control over what is and is not shown to the user during the learning process.

Overall, many of the functional requirements listed in section 3.1 were met, or otherwise changed during development.

5.2.2 Non-Functional Requirements

The non-functional requirements were listed as three broad requirements, being Usability, Ability to teach and extendability.

Usability

The majority of testers answered that they indeed found VirtSort to be easy to use. One aspect of this is most certainly the use of VR technology, and the intuitiveness that it brings. Even testers who had no prior experience with VR could with some instructions begin to interact with the virtual world. This does show however, that there is an initial barrier to using the application with no prior VR experience. As there were many testers during the session, it was not possible to run a longer test to see if a user with no experience would be able to find their way through the application completely on their own. As VirtSort is seen as a learning tool for use at universities however, there is a high chance that an instructor will be available for guidance.

The controller tooltips were shown in the first scene, and most testers caught on to what the individual buttons did pretty quickly. Some however expressed confusion as to what the *interact* button did, which prompted the instructors to give them a short explanation. It was also unnatural for the same testers to press the *interact* button to interact with virtual buttons, rather than pressing them by physically moving their hands. This can indicate unfamiliarity with more traditional game mechanics, as it is an abstraction very common in games. Making buttons pushed using physical movement could make the experience even more intuitive for new VR users.

For the algorithms themselves, it was, as stated earlier in section 5.1.3, minimal problems for the majority of testers to watch the video and perform the algorithm for both bubblesort and insertionsort. Quicksort however proved to be a problem due to its complexity, and while indicators showed the user what they needed to do, it failed to properly convey why the algorithm did what it did. As such, there was a disconnect in the difficulty progression in terms of the *challenge* and *player skills* in the GameFlow model described in section 2.2.2. The jump from the simpler algorithms of bubblesort and insertionsort to the more complex quicksort was too big, and another intermediate algorithm in between would be ideal.

The video button giving no feedback when it was pressed is another usability concern, and was also an error in the feedback room created for the test session. Testers leaving feedback would not know if they had properly pressed a number or not, which resulted in them answering several questions without looking at them. As a result of this and the tight time schedule, the questionnaire was moved to a Google Forms instead.

It also has to be mentioned that the usage of virtual reality can cause nausea for some users, as experienced during this test session as well. Weighing the pros and cons however, very few actually experienced ”VR sickness”, and the majority of users were fine. There are also still more that can be done to mitigate nausea, so while it can be a problem if many users are unable to use the application, there are still many efforts left to explore before arriving at a conclusion.

Ability to Teach

To make a fun learning environment according to Thomas W. Malone, it had to have *Challenge*, *Fantasy* and *Fun*, as mentioned in section 2.2.1. The immersive nature of Virtual Reality already provides a base level of fantasy to the player. In the same section it was stated that the second layer of Bloom’s Taxonomy was of interest. In this regard, the application uses the second and third layers to assess if a user have sufficient knowledge. While watching the video of the algorithm can be considered part of the second layer, the user has to *apply* this knowledge to be able to solve the algorithm in the correct steps. This shows that if a user can successfully perform the task presented in VirtSort, they have understood the algorithm. This may be a premature conclusion, however. As shown in the test results, the majority of testers had already learned the algorithms, and the video was merely a refresher for them to perform the algorithm. As indicators can help show what to do in every step of the algorithm, sorting an array may be the result of simply following the indicators, rather than understanding the principles. To solve this, some sort of evaluation mode where any move is possible should be considered.

While some testers had no problems working through all algorithms, others were confused by how the quicksort algorithm worked. As discussed previously, the complexity gap between insertionsort and quicksort may have been too large. The quicksort algorithm was also intended as the algorithm to teach the concepts of recursion, but since even testers who have previously had the course TDT4120 struggled to keep up with the algorithm, part of the feedback was insufficient to properly convey recursion. Perhaps implementing another version of insertionsort in a recursive manner, as discussed in section 2.2.4 could have been the perfect intermediate step to more complexity as well as recursion as a concept, and should definitely be explored in future iterations.

Extendability

Extendability is important for a project like this, where various algorithms need to be implemented and tested. Even during the specialization project thought was put into the potential extendability for several sorting algorithms, and the initial project was designed accordingly. During further development however, it was evident that the first iteration of the architectural design of the application was flawed, as it did not offer the flexibility needed to expand with more complex algorithms. As the final prototype presents three different algorithms, it can be confidently stated that in its current state, VirtSort should be able to support most, if not all, in-line sorting algorithms using the same scene with the same boxes.

Unity as the editor of choice was also a conscious decision in terms of extendability, due to its highly modular way of piecing together a working project. As stated in section 4.1.6, a prototype of a tree traversing algorithm was created. This had no impact on the functionality of the sorting algorithms at all, as it was kept separate in its own scene. This means that new algorithms can be added easily by creating tailor-made environments using Unity scenes, without having to modify already existing scenes.

5.3 Research Questions

In this section the research questions written in section 1.3 will be discussed. It will look at how VirtSort manages to answer or give insight to the questions.

5.3.1 RQ1: How can Virtual Reality be utilized to create an instructional environment to teach students the inner workings of algorithms?

The instructional environment in this case is a very simple scene created in Unity. Because the scenes are virtual environments, they can be literally anything. As they say, the best way to learn is by doing [61]. The inherent strength of VR is that almost any conceivable instructional environment is constructable within VR, as long as sufficient time is spent. Thus it is possible to tailor make the experience to fit with what is going to be taught. In the physical world, there are many limitations to what can and can not be done, but these do not exist in VR. One can make the environment as lively or as bare bones as necessary, and might even create different environments depending on the needs of the user.

As for teaching the inner workings of algorithms, it was discussed earlier how some reported that they did indeed learn bubblesort and insertionsort during the short test period available. This shows that, at least for sorting algorithms, it is definitely possible to create a virtual environment for teaching. There is no reason to believe this can not be done for other algorithms, such as tree traversing or search algorithms. The question is not if it can be done, but rather how to create a compelling environment for the specific algorithm that is being taught. For sorting algorithms, a table and boxes was chosen as the analogy. For tree traversal, the proposed analogy is a rooms and doors model, as was described in section 4.1.6. These are certainly not the only possible analogies, and other environments should be considered in the future.

5.3.2 RQ2: What benefits and drawbacks can Virtual Reality provide for learning compared to more traditional teaching methods?

The instructional environment that VR can offer is very different from traditional methods. One of the main differences is how well the user can be isolated from other distractions. By completely immersing oneself in the virtual world, the user can fully concentrate on the task at hand. This can be a double edged sword however, especially for those who previously have not experienced VR before. They might focus more on the inherent wonder of using VR rather than the purpose of the application. In these cases, the application itself must be compelling enough for the user, or an instructor has to be nearby to keep them focused.

The intuitive controls means that anyone can perform the task, in this case sorting a table of boxes with various numbers, following an algorithm. This is also a hands-on experience, which one would otherwise only be able to do with physical blocks. This would be extremely inefficient to set up compared to the wonders of software which can generate a new problem at the click of a button. The drawback however, is the extra time it takes to set up the VR system properly, as well as the requirement for sufficient space. It is easy to place students in front of a computer screen to watch an animation of the algorithm in action, or even have them perform it themselves using the mouse. In exchange of a more immersive experience, there is instead cost in the setup.

5.3.3 RQ3: Can such software be used for training and evaluation purposes?

The short answer is that yes, it can be used for training purposes, as seen during the test session. Some test subjects had experience with VR, and signed up primarily to refresh their own knowledge around algorithms. In this regard VirtSort is very good, as it keeps the algorithm introduction short and concise with a video, before letting the user try their hands on sorting the boxes. The different settings allow the user to fine-tune the difficulty to their level of knowledge, and thus is effective for nudging the dormant memories.

It gets a bit trickier when talking about evaluation. While there is nothing inherently with the concept of VirtSort stopping any instructor to use it for evaluation, no direct functionality for it was implemented during development. One can imagine that a student can be put into this VR environment, completely separate from the real world, and tasked with sorting a pre-determined list with a set time limit, or simply using the correct algorithm indicated. Such functionality will be discussed further in section 6.2 as potential topics to explore in the future. It should be noted however that there would be significant logistical issues currently to conduct something like an exam within VR, as all students would have to have access to a VR system as well as enough space to freely move in the virtual environment. With the additional possibility of some students experiencing nausea, conducting whole exams in VR might not yet be viable.

5.4 Reflection

5.4.1 Virtual Reality as a Learning Technology

Many came to the test solely due to the fact that the application itself was in virtual reality. This can be seen as a positive effect, as it draws in people who may benefit from its educational capabilities using their *curiosity* for new technology. Pairing the ”boring” with novel new experiences is one way to make people willing to give the application a try.

Furthermore, VR has a very intuitive way of interacting with the application. This lowers the overhead of what the user has to learn to be able to use the learning technology effectively. In this case though, this might not be as important compared to traditional technology, as the users this application is aimed at would be at least somewhat familiar with learning new technology. One can even argue that introducing unfamiliar technology for technologically literate users might be detrimental compared to using traditional mouse, keyboard and screen applications. But in this case, the novelty of VR, ease of use in general, and the adaptability of such users could outweigh the potential learning curve of completely new technology. In fact, since many testers actively wanted to participate due to the use of VR, it might even be a non-issue as users will automatically want to try and understand how the technology works.

Of course, if VR technology keeps following its upward trend in the mainstream consciousness [29], the novelty of the technology will also wear off. This is inevitable with all new technology, and the question is if VR is here to stay, and if it does, will it still be good for learning without its novelty? With the strong immersive and intuitive nature of VR, there are plenty of opportunities for VR to shine in the education department. While this thesis is specifically targeting computer science students and algorithms, there are several other subjects were immersion or the one-to-one mapping of the user's physical body and the virtual world is of utmost importance. It is safe to say that Virtual Reality as a learning technology in general is here to stay, regardless of if it catches on as an entertainment device for the mainstream. Thus, even though part of the focus of this thesis was to explore how VR can be used in education, it should be noted that in the end, what is important is the underlying pedagogy, and not the technology itself going forward [55, Ch. 17].

5.4.2 The Importance of Testing

The unfortunate truth is that the testing phase for this project came very late in development. While VirtSort received overall positive feedback, there were also many points that sorely need improvement. During development, most test sessions were conducted bi-weekly by the supervisor, and additions, changes and the likes were discussed before development continued. While this worked well for having a good flow in development, it was inevitable that problems fell through the cracks that neither the supervisor nor the developers caught due to their inherent familiarity with the project. During future development, it is paramount that testing is done more frequently, and especially with test subjects with no familiarity of various topics relevant to the application.

Another aspect of the testing worth discussing is the questionnaire. It consisted of one self-made question, followed by the ten standard SUS questions. This decision was made to get the best possible overview of the application's strong and weak points. This might have been good enough for a general impression of its current state, which was important to be able answer the research questions for this thesis. On the other hand, it does not go much into detail of the specifics of the application, which would have been very useful for future development. In hindsight, these kinds of questions should have probably been included in the questionnaire to get better, more accurate insights, but ultimately the decision was made to keep it short and simple. It was also pointed out by one of the testers that a text field for ”additional feedback” would have been greatly beneficial. Luckily, this was somewhat relived by verbal feedback, but getting that information in writing would have been preferred.

The test itself had a very small sample size of twelve people, and all the testers only had ten minutes to try out the application. While the answers from the test is discussed extensively in section 5.1.2, they should be taken with a grain of salt due to these factors. Additionally, a question about the user's previous experience with VR, as well as other questions going into more details of the tester's previous experience would help immensely with correctly analyzing the answers.

5.4.3 Organizing Development

As mentioned in section 3.2, SCRUM-style development was used with the developers working mostly from separate locations. For most of the development time, this method worked out fine, but as time went by, some problems became apparent.

First of all, discussions around code was scarce. These were mostly done during after supervisor meetings every two weeks, and otherwise over Facebook Messenger. What became apparent was that text-based communication was not ideal to discuss larger pieces of codes, and thus made collaboration harder, especially working on code that was not your own. To alleviate this, it was eventually arranged to meet in person for pair programming sessions, which were carried out a few times towards the end of development. From these experiences, it is evident that more of these pair programming sessions should have been interspersed throughout the project. Sessions without pair programming could also have been arranged, as being together physically ensures questions are answered immediately, as opposed to over Messenger, where it can take hours to get a proper response.

As the project went on, general planning suffered as well. Although a task list was used to keep track of all tasks and issues, as mentioned in section 3.2.2, its use became less structured as time went by. In the beginning tasks where discussed after supervisor meetings, and the most important and urgent ones were distributed right away. Towards the end however, tasks were more often not documented, and assignment became more first come first serve between meetings, with reports of new functionalities being implemented passed through Messenger. This was not a big point of concern during development, as tasks were still being done and progress made, but in hindsight should probably been given more attention. Looking back there are many tasks that could have been prioritized better, and processes made more streamlined, had proper planning been performed. This was partially an effect from the short development time from the specialization project. Due to various unfortunate circumstances at the time, the time for streamlining the development pipeline was cut short. This should have been an indication that some more thorough planning could be beneficial, but considering the short development time for the project, a decision was made to start developing as soon as possible.

5.4.4 Focused Prototype vs. Broad Prototype

During the development of VirtSort, the focus changed from a rather broad prototype that implemented several types of algorithms, to a more focused prototype that concentrated solely on the sorting algorithms. Although both approaches have some merits on its own, with both pros and cons, it became a detriment to the final product. This was mostly due to spending time and effort, not only implementation-wise but also on research and planning, on features that never made it into the application. An example of this is the DFS prototype, which was meant only as a proof-of-concept and experimental playground for a fully implemented tree search algorithm scene. Development also began on a functional tree search scene, which ultimately got scrapped in the final product. These might be useful for future work, but ended up contributing minimally in regards to the product at the time of writing.

Considering the focus of the thesis, it was probably a good thing that the focus was changed. A broad prototype with many algorithms might not have given as good of an impression of VR's learning potential, as the cases would not be fully implemented, and it would be even harder to distinguish limitations of VR from limitations of the implementations. On the other hand, a broad prototype might have explored more ways to use VR as an educational tool, and thus uncovered more of its potential and pitfalls.

As a conclusion, there might not have been a ”right way” to approach the implementation. What is certain though, is that it would have been beneficial to have made that decision and stuck with it at the beginning of the project, in order to make the most of the available time. The main factor in deciding which approach to use, which was also the reason for the change midway in the project, was which approach would best answer the research questions.

Chapter 6

Conclusion and Future Work

This thesis has explored how VR can be used for teaching the intricacies of abstract and complex topics such as computer algorithms. In this chapter, the conclusions reached after the development of VirtSort, as well as suggestions for future development of the application can be found.

6.1 Conclusion

This thesis has explored an alternative way of teaching algorithms by developing a virtual reality based computer application, with focus on sorting algorithms. By utilizing virtual reality, the user is able to, ideally, learn algorithms in more active, intuitive and innovative ways. Although the prevalence of virtual reality today is limited, its growth certainly makes it an important tool to consider for the future in many fields of study, including learning environments. What this thesis has shown is that VR can be used as an effective tool for learning, especially by offering a substantially different experience than other learning methods.

The biggest challenge encountered in developing this application, was to make scenarios that were easily understandable and functionalities that were intuitive. This was made even more difficult by the fact that most people have little to no prior experience with a VR system, and must therefore not only learn the application, but also get familiar with the VR technology. As VR technology differs quite substantially from other interaction technologies, the need for instruction is non-trivial, even with its inherent intuitive nature. This thesis shows the viability of using a table and boxes analogy in VR for teaching sorting algorithms, but does not claim this to be the best analogy to use, and other possibilities should certainly be explored further.

Based on the experiences from the development of VirtSort, despite the challenges of translating algorithms to the virtual world, it is apparent that VR technology certainly could have a place in educational software, even for abstract concepts. Today's technology make VR a viable market both economically and practically, and powerful development software, such as Unity and Unreal Engine, are available for free. The problem is not the technology, but rather how to use it effectively. And although there are many papers discussing benefits of virtual reality education and training, there are not many application implementing these ideas yet. Thus, many avenues for future work can be found to further explore the possibilities of VR technology in education.

6.2 Future Work

This section will suggest several aspects that should be an important focus for any future work of the VirtSort application.

6.2.1 Teaching the User VR and the application

One of the main challenges for users trying out the application was getting familiar with VR and understanding every aspect of the application. Thus, a more structured learning method for the user should be considered, for example by offering a tutorial for first time users. This is a well known approach used widely in video games, where the player is gradually introduced to different aspects of the game. A similar mechanic should be considered for VirtSort, where the player is first familiarized with using VR, including movement and interaction. Next the player should learn how the various parts of the application works, such as pressing buttons, picking up boxes, followed by more advanced topics such as the sort table.

Furthermore, the help functionalities within the main game should be improved. First of all, the instructional videos should be replaced with custom videos made specifically for the application. This would allow the videos to use video clips taken from the application, and the explanation to be based around the in-application execution of the algorithm. This could make the transition from video instruction to execution more painless, and thus the overall experience more satisfying. Other additions could be showing a live demo of the algorithm, where the boxes are moved according to the algorithm by an invisible force, and could even contain explanatory audio to accompany it. There is also more work that could be done to improve features such as the trigger colors and box highlighting, to make the state and purpose of the boxes and triggers even clearer.

6.2.2 Gamification

Gamification was an aspect that was intended to be implemented, but was cut due to time constraints. Although there is need for further studies on gamification, current studies suggest has great potential to improve learning and the learning experience [52]. This was discussed as a natural addition to the application, as it already has many game-like features, by using both game-centric hardware and software. This would also mean the implementation of gamification features would most likely be relatively pain free.

The most considered gamification features were adding a timer timing the algorithm execution, combined with a leaderboard, where players can compare themselves to others. This was believed could motive the player to get to know the algorithms further, not by repetition for the sake of repetition, but to beat the competition. This is already proven to be effective as a leaderboard is used in TDT4120's assignment system. This mechanic could also be combined with an achievement system, which has proved a very effective motivator in the gaming scene, as first demonstrated by Microsoft for the Xbox 360 [54]. The player could for example be awarded achievements for completing an algorithm faster than a given time.

Another possibility for gamification could be to add suspense and tension, for example by introducing enemies. The player would then have to use their knowledge to defeat the enemies, preferably by combining several algorithms in the same ”level”. The player is then rewarded for being both fast and accurate, while at the same time being under pressure. After this, the player would hopefully have further solidified their knowledge of various algorithms without feeling like they are being taught. Depending on the implementation, this could be both an extrinsic and intrinsic approach. As intrinsic fantasies tend to perform better, as discussed in section 2.2.1, a possible implementation could be to make the array a representation of an army that the user must sort in the correct order to maximize damage to the enemy forces.

An excellent example of gamification can be seen in the game/learning tool Rocksmith for game consoles and PC, where the player uses an actual guitar. Here, there are several minigames that teaches and trains specific aspects of guitar playing, such as chords [59]. Very similar mechanics could easily be implemented for VirtSort as well, and could improve the players experience with the application.

6.2.3 Using VirtSort as an Evaluation Tool

To make VirtSort a tool for evaluation, some new functionality needs to be implemented. One of these would be the possibility for the instructor to specify the test conditions. Currently, all of the difficulty settings are available to the player, but for evaluation purposes these have to be set beforehand, and made inaccessible for the user. Making any move possible for the user should also be considered, so as to give minimal guidance to the user, letting them conduct the sorting in any conceivable way. With this, there needs to be a way to verify the moves to be correct, so each move must be recorded and saved for the instructor to evaluate at a later date.

For the user, making sure that they have full control of which moves end up as their submitted solution is important. In exams, only what the student believes is the correct answer is submitted, and VirtSort will have to function in a similar way for it to be viable for use in examinations. Of course, if the instructor wishes to use VirtSort to evaluate the current knowledge level of the student, various levels of logging should be made available.

A possible approach to an examination area being discussed around the start of the project was a scene with several areas containing different algorithms. For example, there could be various tables that the student is expected to sort with a specific algorithm, or a room the student can enter to solve the search tree problem. This way, all examination questions will be available in one area, making the exam more seamless than having to wait for loading times while changing scenes, or having to take on and off the HMD during the exam.

6.2.4 Implementing a Search Tree Scene

As some time was spent working on the DFS prototype and how to best represent a search tree in VR, some additional effort will be made to discuss how this could be successfully implemented with the functionality showcased by the prototype and support of various tree search algorithms. To accomplished this, there are a few major components that need to be implemented.

First of all, a room prefab, that could be customized with regards to number of parents, children and siblings that can be traveled to. The travel would be through a door prefab, as demonstrated in the prototype, with one wall for parent doors, one for children, one for the left sibling and one for the right sibling, where any wall could potentially be empty. The doors should be generated automatically based on properties of the node it represents, and thus should be initialized by the algorithm from a public interface provided by the room.

There should also be some way to view the overall structure of the search tree, although not necessarily as a physical map on the walls of the rooms. Other alternatives includes including it as an UI element, or as a pop-up from the controller sort of like the controller tooltips. regardless of the method, this overview should provide the player with a much better overview of how the traversal is executed.

Further additions and changes might even make it possible to implement graph traversal algorithms such as *Dijkstra's* and *A**. These algorithms typically jump more sporadically around the graph, and might be hard to represent in a comprehensible way through the "rooms and doors" analogy, however, so other analogies should be explored.

Bibliography

[1] 5-year msc in communication technology (commtech) - ntnu. https://www.ntnu.edu/studies/mtkom. Accessed: 2018-06-13.

[2] America's army. https://www.americasarmy.com/. Accessed: 2017-12-13.

[3] Bachelor in informatics 3 years - ntnu. https://www.ntnu.edu/studies/bit. Accessed: 2018-06-13.

[4] Bloom's taxonomy. https://cft.vanderbilt.edu/guides-sub-pages/blooms-taxonomy/. Accessed: 2018-06-12.

[5] Computer science (master, 5 years) - ntnu. https://www.ntnu.edu/studies/mtdt. Accessed: 2018-06-13.

[6] Developing for oculus rift. https://docs.unrealengine.com/en-us/Platforms/Oculus. Accessed: 2018-06-12.

[7] Digital trends: Oculus rift vs. htc vive. https://www.digitaltrends.com/virtual-reality/oculus-rift-vs-htc-vive/. Accessed: 2018-05-27.

[8] Forklift training in VR. https://unimersiv.com/forklift-training-vr/. Accessed: 2018-06-12.

[9] From visual simulation to virtual reality to games - IEEE journals & magazine. http://ieeexplore.ieee.org/abstract/document/1510565/?part=1. Accessed: 2017-12-13.

[10] Gear vr. http://www.samsung.com/au/wearables/gear-vr-r322/. Accessed: 2018-06-12.

[11] Github repository: Virtsort. https://github.com/txkong94/VirtSort. Accessed: 2018-06-12.

[12] Google cardboard. https://vr.google.com/cardboard/. Accessed: 2018-06-07.

[13] Google daydream. https://vr.google.com/daydream/. Accessed: 2018-06-07.

[14] Htc vive virtual reality headset with handheld controllers. https://www.xcite.com/htc-vibe-virtual-reality-headset-with-handheld-controllers.html. Accessed: 2018-06-12.

[15] Job simulator. https://jobsimulatorgame.com/. Accessed: 2018-06-09.

[16] Magnus lie hetland, associate professor. https://www.ntnu.edu/employees/mlh. Accessed: 2018-06-05.

[17] Medical realities. https://www.medicalrealities.com/. Accessed: 2018-06-12.

[18] Microsoft acer windows mixed reality headset with motion controllers. https://www.microsoft.com/en-us/p/acer-windows-mixed-reality-headset-with-motion-controllers/8ttsf1q97hkp/h79g/. Accessed: 2018-06-06.

[19] Mixed reality at microsoft. https://www.microsoft.com/en-us/mixed-reality. Accessed: 2018-06-11.

[20] Oculus go — oculus. https://www.oculus.com/go/. Accessed: 2018-06-11.

[21] Oculus rift. https://www.oculus.com/rift/. Accessed: 2018-06-06.

[22] Oculus rift: Step into the game. https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game. Accessed: 2017-12-13.

[23] Playstation vr - over 100 games and counting. feel them all. - playstation. https://www.playstation.com/en-us/explore/playstation-vr/. Accessed: 2018-06-11.

[24] Samsung gear vr with controller - the official samsung galaxy site. https://www.samsung.com/global/galaxy/gear-vr/. Accessed: 2018-06-11.

[25] Steamvr plugin - asset store. https://assetstore.unity.com/packages/templates/systems/steamvr-plugin-32647. Accessed: 2018-06-12.

[26] System usability scale (SUS). https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html. Accessed: 2018-06-07.

[27] TDT4120 - algorithms and data structures. https://www.ntnu.edu/studies/courses/TDT4120#tab=omEmnet. Accessed: 2018-05-22.

[28] TextMesh Pro - Asset store. https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126. Accessed: 2018-06-05.

[29] The vr hardware report: How stand-alone vr headsets will usher in mainstream adoption beginning in 2018 - business insider. http://www.businessinsider.com/the-vr-hardware-report-2018-3?r=US&IR=T&IR=T. Accessed: 2018-06-11.

[30] Trello. https://trello.com/. Accessed: 2018-06-12.

[31] Unity. https://unity3d.com/. Accessed: 2018-06-11.

[32] Unity - unity services - collaborate. https://unity3d.com/unity/features/collaborate. Accessed: 2017-12-13.

[33] Virtsort - youtube - youtube. https://www.youtube.com/playlist?list=PLuhO5YXkjLSy-qPoatYDg6GwSwIU32dmM. Accessed: 2018-06-13.

[34] Virtual Reality Society: who coined the term "virtual reality"? https://www.vrs.org.uk/virtual-reality/who-coined-the-term.html. Accessed: 2018-05-27.

[35] Visual studio code 1.0 release. https://code.visualstudio.com/blogs/2016/04/14/vscode-1.0#_the-history-of-vs-code. Accessed: 2018-06-12.

[36] VIVE VR system. https://www.vive.com/us/product/vive-virtual-reality-system/. Accessed: 2018-05-26.

[37] VRFocus Palmer Luckey explains Oculus Rift's constellation tracking and fabric. https://www.vrfocus.com/2015/06/palmer-luckey-explains-oculus-rifts-constellation-tracking-and-fabric/. Accessed: 2018-05-27.

[38] Vrtk - virtual reality toolkit. https://vrtoolkit.readme.io/. Accessed: 2018-06-12.

[39] Who coined the term "virtual reality"? - virtual reality society. https://www.vrs.org.uk/virtual-reality/who-coined-the-term.html. Accessed: 2017-12-13.

[40] C# coding conventions (C# programming guide). https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions, July 2015. Accessed: 2018-06-06.

[41] If you love something, set it free. https://www.unrealengine.com/en-US/blog/ue4-is-free, Mar 2015. Accessed: 2018-05-29.

[42] To handle VR graphics, gaming PCs have to be 7 times more powerful. https://venturebeat.com/2015/12/30/to-handle-vr-graphics-gaming-pcs-have-to-be-7-times-more-powerful/, Dec 2015. Accessed: 2018-05-28.

[43] Unity coding standards. `https://unity3d.college/2016/05/16/unity-coding-standards/`, May 2016. Accessed: 2018-06-06.

[44] Using unity at valve. `https://www.youtube.com/watch?v=4Gs5k2Fti1U`, Feb. 2016. Accessed: 2018-06-05.

[45] How to git with unity. `https://robots.thoughtbot.com/how-to-git-with-unity`, June 2017. Accessed: 2017-12-13.

[46] Review: Google daydream view. `http://www.gadgetguy.com.au/product/review-google-daydream-view/`, Dec. 2017. Accessed: 2018-06-12.

[47] Unreal engine 4.16 has released. `https://mmoexaminer.com/unreal-engine-4-16-released/`, May 2017. Accessed: 2018-06-12.

[48] Interview: Unity technologies on the future of unity. `https://www.n3rdabl3.com/2018/02/unity-technologies-interview-2018/`, Feb. 2018. Accessed: 2018-06-12.

[49] Oculus go pre-order page hits amazon, best buy stocks shelves. `https://www.roadtovr.com/oculus-go-pre-order-page-hits-amazon-best-buy-stocks-shelves/`, May 2018. Accessed: 2018-06-12.

[50] E. P. A. and N. T. J. Behaviorism, cognitivism, constructivism: Comparing critical features from an instructional design perspective. *Performance Improvement Quarterly*, 6(4):50–72.

[51] S. Clarke. Microsoft and partners prepare to democratize virtual reality this holiday season with new headsets, content experiences and more. `https://blogs.microsoft.com/firehose/2017/08/28/microsoft-and-partners-prepare-to-democratize-virtual-reality-this-holiday-season-with-new-headsets-content-experiences-and-more/`, Aug. 2017. Accessed: 2018-06-12.

[52] G. A. Darina Dicheva, Christo Dichev and G. Angelova. Gamification in education: A systematic mapping study. *Journal of Educational Technology & Society*, 18(3):75–88, July 2015.

[53] Dave. Why i'm not selling my playstation vr headset – reader's feature. `https://metro.co.uk/2017/07/15/why-im-not-selling-my-playstation-vr-headset-readers-feature-6781141/`, July 2017. Accessed: 2018-06-12.

[54] M. Jakobsson. The achievement machine: Understanding Xbox 360 achievements in gaming practices. *Game Studies*, 11(1):1–22, Feb. 2011.

[55] D. Jonassen, M. J. Spector, M. Driscoll, M. David Merrill, J. van Merrienboer, and M. P. Driscoll. *Handbook of Research on Educational Communications and Technology: A Project of the Association for Educational Communications and Technology*. Routledge, Sept. 2008.

[56] T. W. Malone. What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pages 162–169. ACM, Sept. 1980.

[57] R. Mcdaniel. Bloom's taxonomy. `https://wp0.vanderbilt.edu/cft/guides-sub-pages/blooms-taxonomy/`, June 2010. Accessed: 2017-12-13.

[58] A. McLaughlin and Growth. Fun in learning: Why entertainment is essential for adult education. `http://www.growthengineering.co.uk/why-fun-in-learning-is-important/`, Mar. 2017. Accessed: 2017-12-13.

[59] T. Ogilvie. Rocksmith 2014 review. `http://www.ign.com/articles/2013/10/22/rocksmith-2014-review`, Oct. 2013. Accessed: 2018-06-10.

[60] K. Orland. Facebook purchases VR headset maker oculus for $2 billion [updated]. `https://arstechnica.com/gaming/2014/03/facebook-purchases-vr-headset-maker-oculus-for-2-billion/`, Mar. 2014. Accessed: 2017-12-13.

[61] M. Prince. Does active learning work? A review of the research. *Journal of engineering education*, 93(3):223–231, June 2014.

[62] M. Sambol. Bubble sort in 2 minutes. `https://www.youtube.com/watch?v=xli_FI7CuzA`. Accessed: 2018-06-13.

[63] M. Sambol. Insertion sort in 2 minutes. `https://www.youtube.com/watch?v=JU767SDMDvA`. Accessed: 2018-06-13.

[64] M. Sambol. Quicksort in 2 minutes. `https://www.youtube.com/watch?v=Hoixgm4-P4M`. Accessed: 2018-06-13.

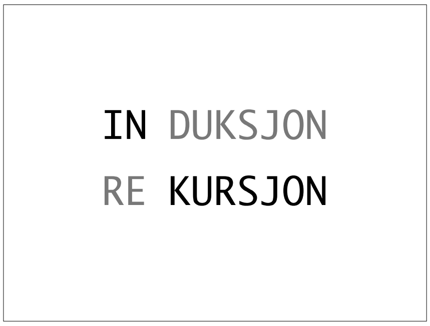
[65] T. Susi, M. Johannesson, and P. Backlund. Serious games : An overview. 2007.

[66] P. Sweetser and P. Wyeth. GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, July 2005.

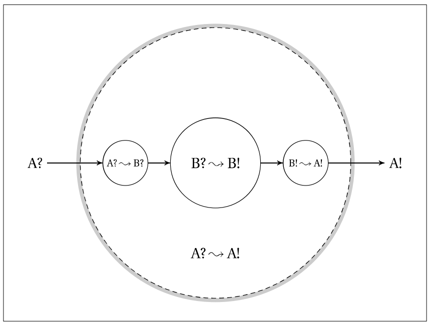
[67] Wikipedia contributors. Google cardboard. `https://en.wikipedia.org/wiki/Google_Cardboard`, 2018. Accessed: 2018-06-12.

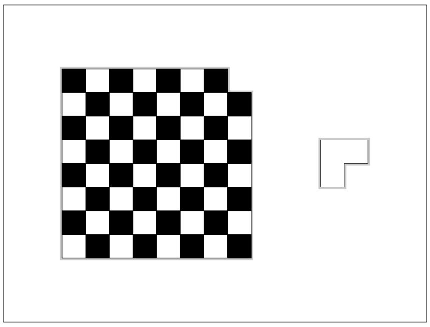
Appendices

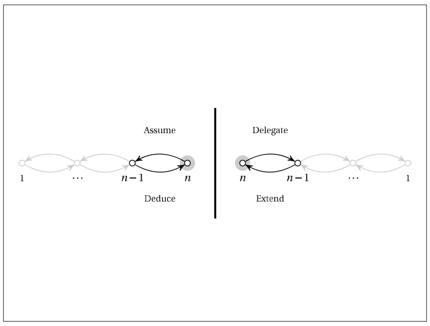
A TDT4120 - Excerpt from lecture

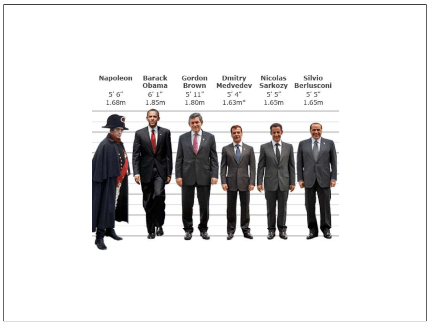

IN DUKSJON

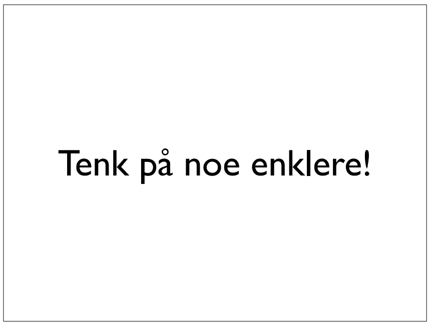
RE KURSJON



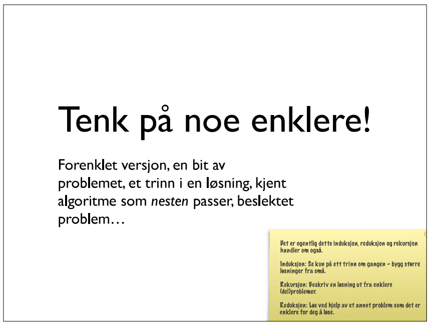





Napoleon 5′ 6″ 1.68m
Barack Obama 6′ 1″ 1.85m
Gordon Brown 5′ 11″ 1.80m
Dmitry Medvedev 5′ 4″ 1.63m*
Nicolas Sarkozy 5′ 5″ 1.65m
Silvio Berlusconi 5′ 5″ 1.65m



Tenk på noe enklere!



Tenk på noe enklere!

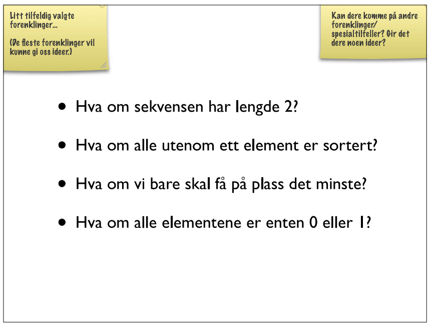
Forenklet versjon, en bit av problemet, et trinn i en løsning, kjent algoritme som *nesten* passer, beslektet problem…

Det er egentlig dette induksjon, reduksjon og rekursjon handler om også.

Induksjon: Se kun på ett trinn om gangen – bygg større løsninger fra små.

Rekursjon: Beskriv en løsning ut fra enklere (del)problemer.

Reduksjon: Løs ved hjelp av et annet problem som det er enklere for deg å løse.

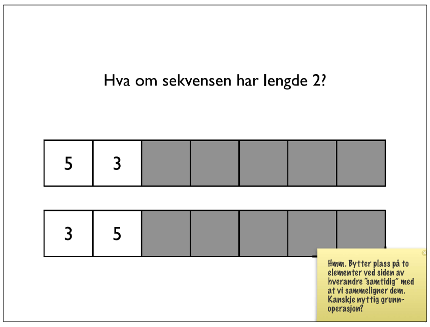


Litt tilfeldig valgte forenklinger…

(De fleste forenklinger vil kunne gi oss ideer.)

Kan dere komme på andre forenklinger/ spesialtilfeller? Gir det dere noen ideer?

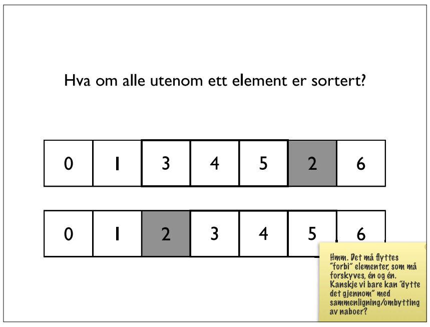
- Hva om sekvensen har lengde 2?
- Hva om alle utenom ett element er sortert?
- Hva om vi bare skal få på plass det minste?
- Hva om alle elementene er enten 0 eller 1?



Hva om sekvensen har lengde 2?

5	3	▨	▨	▨	▨	▨
3	5	▨	▨	▨	▨	▨

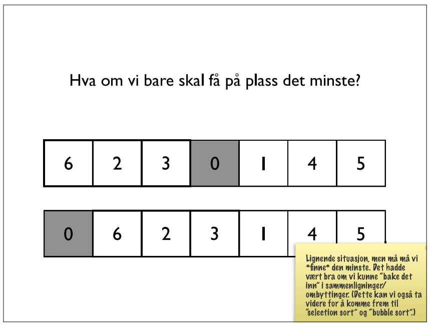
Hmm. Bytter plass på to elementer ved siden av hverandre "samtidig" med at vi sammeligner dem. Kanskje nyttig grunn-operasjon?



Hva om alle utenom ett element er sortert?

0	1	3	4	5	2	6
0	1	2	3	4	5	6

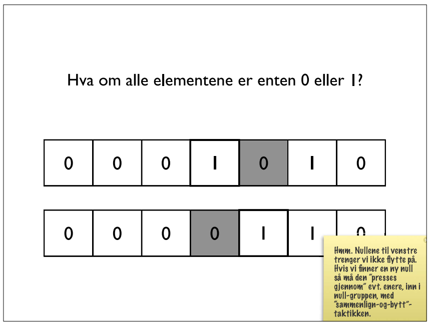
Hmm. Det må flyttes "forbi" elementer, som må forskyves, én og én. Kanskje vi bare kan "dytte det gjennom" med sammenligning/ombytting av naboer?



Hva om vi bare skal få på plass det minste?

6	2	3	0	1	4	5
0	6	2	3	1	4	5

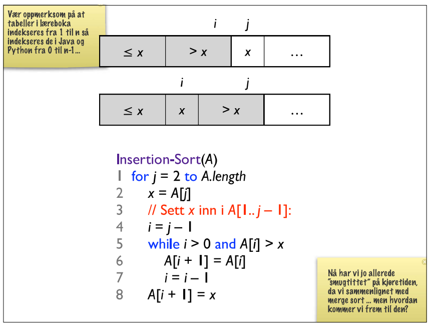
Lignende situasjon, men må må vi *finne* den minste. Det hadde vært bra om vi kunne "bake det inn" i sammenligninger/ ombyttinger. (Dette kan vi også ta videre for å komme frem til "selection sort" og "bubble sort".)



Hva om alle elementene er enten 0 eller 1?

0 0 0 1 0 1 0
0 0 0 0 1 1 0

Hmm. Nullene til venstre trenger vi ikke flytte på. Hvis vi finner en ny null så må den "presses gjennom" evt. enere, inn i null-gruppen, med "sammenlign-og-bytt"-taktikken.



Vær oppmerksom på at tabeller i læreboka indekseres fra 1 til n så indekseres de i Java og Python fra 0 til n-1…

i
j
≤ x
> x
x
…
i
j
≤ x
x
> x
…

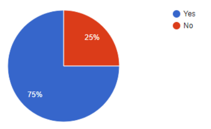
```
Insertion-Sort(A)
for j = 2 to A.length
    x = A[j]
    // Sett x inn i A[1..j – 1]:
    i = j – 1
    while i > 0 and A[i] > x
        A[i + 1] = A[i]
        i = i – 1
    A[i + 1] = x
```

Nå har vi jo allerede "smugtittet" på kjøretiden, da vi sammenlignet med merge sort … men hvordan kommer vi frem til den?

B Test Results

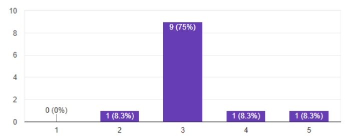
I have had the course 'Algoritmer og Datastrukturer' at NTNU

12 responses



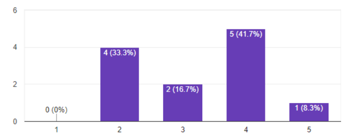
I think that I would like to use VirtSort frequently

12 responses



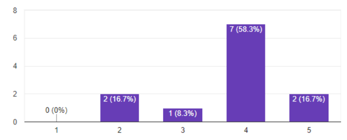
I found VirtSort to be simple

12 responses



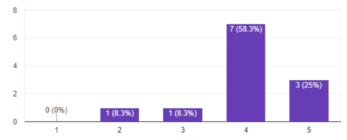
I thought VirtSort was easy to use

12 responses



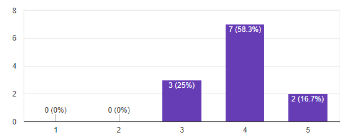
I think that I could use VirtSort without the support of a technical person

12 responses



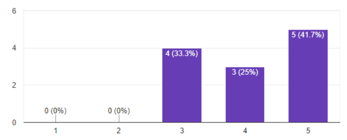
I found the various functions in VirtSort were well integrated

12 responses



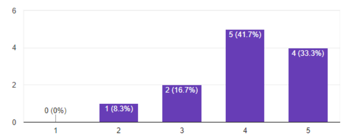
I thought there was a lot of consistency in VirtSort

12 responses



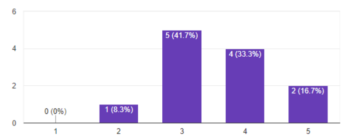
I would imagine that most people would learn to use VirtSort very quickly

12 responses



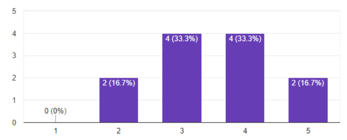
I found VirtSort very intuitive.,I felt very confidence using VirtSort

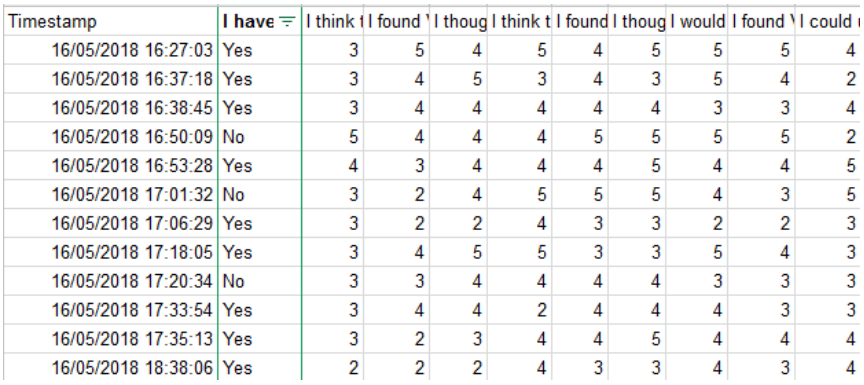
12 responses



I could use VirtSort without having to learn anything new

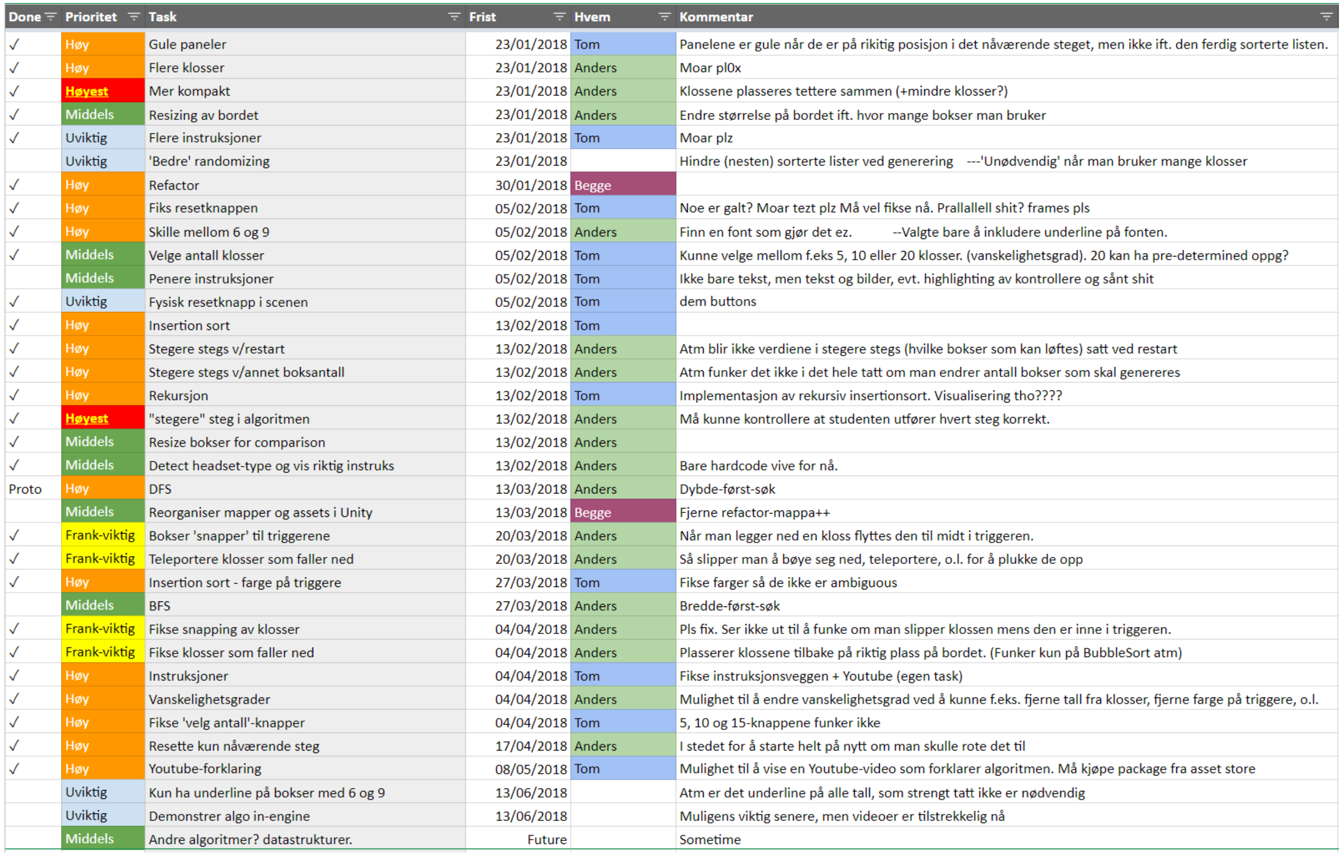
12 responses





Timestamp	**I have**	I think t	I found	I thoug	I think t	I found	I thoug	I would	I found	I could
16/05/2018 16:27:03	Yes	3	5	4	5	4	5	5	5	4
16/05/2018 16:37:18	Yes	3	4	5	3	4	3	5	4	2
16/05/2018 16:38:45	Yes	3	4	4	4	4	4	3	3	4
16/05/2018 16:50:09	No	5	4	4	4	5	5	5	5	2
16/05/2018 16:53:28	Yes	4	3	4	4	4	5	4	4	5
16/05/2018 17:01:32	No	3	2	4	5	5	5	4	3	5
16/05/2018 17:06:29	Yes	3	2	2	4	3	3	2	2	3
16/05/2018 17:18:05	Yes	3	4	5	5	3	3	5	4	3
16/05/2018 17:20:34	No	3	3	4	4	4	4	3	3	3
16/05/2018 17:33:54	Yes	3	4	4	2	4	4	4	3	3
16/05/2018 17:35:13	Yes	3	2	3	4	4	5	4	4	4
16/05/2018 18:38:06	Yes	2	2	2	4	3	3	4	3	4

C Task list



Done	Prioritet	Task	Frist	Hvem	Kommentar
✓	Høy	Gule paneler	23/01/2018	Tom	Panelene er gule når de er på rikitig posisjon i det nåværende steget, men ikke ift. den ferdig sorterte listen.
✓	Høy	Flere klosser	23/01/2018	Anders	Moar pl0x
✓	Høyest	Mer kompakt	23/01/2018	Anders	Klossene plasseres tettere sammen (+mindre klosser?)
✓	Middels	Resizing av bordet	23/01/2018	Anders	Endre størrelse på bordet ift. hvor mange bokser man bruker
✓	Uviktig	Flere instruksjoner	23/01/2018	Tom	Moar plz
	Uviktig	'Bedre' randomizing	23/01/2018		Hindre (nesten) sorterte lister ved generering ---'Unødvendig' når man bruker mange klosser
✓	Høy	Refactor	30/01/2018	Begge	
✓	Høy	Fiks resetknappen	05/02/2018	Tom	Noe er galt? Moar tezt plz Må vel fikse nå. Prallallell shit? frames pls
✓	Høy	Skille mellom 6 og 9	05/02/2018	Anders	Finn en font som gjør det ez. --Valgte bare å inkludere underline på fonten.
✓	Middels	Velge antall klosser	05/02/2018	Tom	Kunne velge mellom f.eks 5, 10 eller 20 klosser. (vanskelighetsgrad). 20 kan ha pre-determined oppg?
	Middels	Penere instruksjoner	05/02/2018	Tom	Ikke bare tekst, men tekst og bilder, evt. highlighting av kontrollere og sånt shit
✓	Uviktig	Fysisk resetknapp i scenen	05/02/2018	Tom	dem buttons
✓	Høy	Insertion sort	13/02/2018	Tom	
✓	Høy	Stegere stegs v/restart	13/02/2018	Anders	Atm blir ikke verdiene i stegere stegs (hvilke bokser som kan løftes) satt ved restart
✓	Høy	Stegere stegs v/annet boksantall	13/02/2018	Anders	Atm funker det ikke i det hele tatt om man endrer antall bokser som skal genereres
✓	Høy	Rekursjon	13/02/2018	Tom	Implementasjon av rekursiv insertionsort. Visualisering tho????
✓	Høyest	"stegere" steg i algoritmen	13/02/2018	Anders	Må kunne kontrollere at studenten utfører hvert steg korrekt.
✓	Middels	Resize bokser for comparison	13/02/2018	Anders	
✓	Middels	Detect headset-type og vis riktig instruks	13/02/2018	Anders	Bare hardcode vive for nå.
Proto	Høy	DFS	13/03/2018	Anders	Dybde-først-søk
	Middels	Reorganiser mapper og assets i Unity	13/03/2018	Begge	Fjerne refactor-mappa++
✓	Frank-viktig	Bokser 'snapper' til triggerene	20/03/2018	Anders	Når man legger ned en kloss flyttes den til midt i triggeren.
✓	Frank-viktig	Teleportere klosser som faller ned	20/03/2018	Anders	Så slipper man å bøye seg ned, teleportere, o.l. for å plukke de opp
✓	Høy	Insertion sort - farge på triggere	27/03/2018	Tom	Fikse farger så de ikke er ambiguous
	Middels	BFS	27/03/2018	Anders	Bredde-først-søk
✓	Frank-viktig	Fikse snapping av klosser	04/04/2018	Anders	Pls fix. Ser ikke ut til å funke om man slipper klossen mens den er inne i triggeren.
✓	Frank-viktig	Fikse klosser som faller ned	04/04/2018	Anders	Plasserer klossene tilbake på riktig plass på bordet. (Funker kun på BubbleSort atm)
✓	Høy	Instruksjoner	04/04/2018	Tom	Fikse instruksjonsveggen + Youtube (egen task)
✓	Høy	Vanskelighetsgrader	04/04/2018	Anders	Mulighet til å endre vanskelighetsgrad ved å kunne f.eks. fjerne tall fra klosser, fjerne farge på triggere, o.l.
✓	Høy	Fikse 'velg antall'-knapper	04/04/2018	Tom	5, 10 og 15-knappene funker ikke
✓	Høy	Resette kun nåværende steg	17/04/2018	Anders	I stedet for å starte helt på nytt om man skulle rote det til
✓	Høy	Youtube-forklaring	08/05/2018	Tom	Mulighet til å vise en Youtube-video som forklarer algoritmen. Må kjøpe package fra asset store
	Uviktig	Kun ha underline på bokser med 6 og 9	13/06/2018		Atm er det underline på alle tall, som strengt tatt ikke er nødvendig
	Uviktig	Demonstrer algo in-engine	13/06/2018		Muligens viktig senere, men videoer er tilstrekkelig nå
	Middels	Andre algoritmer? datastrukturer.	Future		Sometime