Jannik Hildebrandt, BSc

An Online Collaborative Learning Environment for Computer Science Education

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Johanna Pirker, BSc

Co-Supervisor

Dipl.-Ing. Michael Holly, BSc

Institute of Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, March 2021

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Over the past centuries, the industry's digital transformation has made computer science a relevant field of study. Companies worldwide are searching for new computer scientists and even other jobs nowadays often require fundamental computer science knowledge. Thus, computer science education has become increasingly important over the years. While the traditional approach of frontal instruction is still most common, modern learning approaches are rising. One of these methods is digital learning, utilizing computers or other technology in the learning process. It has the critical benefit that students can learn from home, often even without a dedicated instructor. However, this learning approach lacks a social component, as it leaves aside direct communication with others. That is where collaborative learning comes into play, another modern learning approach. It builds upon going through the learning process as part of a group, profiting from others' knowledge and experiences. The two methods can be combined to create computer-supported digital learning environments, emphasizing the advantages of both. However, there are only a few examples for this approach and even fewer, where it is integrated into a whole virtual learning world.

This thesis presents a multi-user network solution for the immersive learning environment Maroon, an interactive virtual laboratory and experiment environment that allows students to explore various experiments and phenomena first-hand. The multi-user feature adds collaboration to the environment, allowing users to work on the experiments together. To demonstrate the effectiveness of this collaborative environment in computer science education, this thesis introduces a new computer science experiment in Maroon. The experiment showcases essential aspects of algorithms by providing interactive visualizations. Furthermore, it involves a sorting challenge, where users can guess which of two competing algorithms will sort a field faster.

In multi-user mode, users compete who can guess best, providing a proof of concept for an interactive multi-user experience.

To compare the multi-user and the single-user version, an AB study with 35 participants was carried out. The goal was to compare the users' knowledge increase and their emotions in the learning process. The results showed that both groups significantly improved their knowledge. While the multi-users had a slightly higher increase, the single-users already had more prior knowledge, resulting in even knowledge level after the study. However, the multi-user participants were slightly happier throughout the study. Overall, the study showed that the created application is a valuable learning tool, both in single-user and multi-user modes.

Acknowledgements

First of all, I'd like to thank my supervisors Johanna Pirker and Michael Holly, who have guided me through this thesis's creation process with their experience and expertise. They always reliably supported me throughout the journey, even answering questions on weekends or in the middle of the night.

Also, I am grateful for all people who sacrificed some of their time to participate in my study. Their feedback really made me appreciate the created work.

Special thanks to my longtime good friend Kaleb Mertz and my sister Tanja for proofreading this thesis.

Furthermore, I thank my friends and family for always supporting me over the years of study. Especially my girlfriend Veronika has helped me get through the exciting but also stressful time of creating this thesis.

Finally, I thank my parents, who always supported me both financially and mentally. Without them, I would not have achieved this degree.

Contents

List of Figures

List of Figures

List of Tables

Listings

1. Introduction

Computer Science is a broad field of study with versatile focus areas. Many people associate it purely with programming. While it is true that computer scientists should know how to find their way around code, there is much more to it. Computer science is a theory-heavy field, and many basics and concepts have to be understood. However, now more than ever, there is an enormous request for computer scientists. Therefore, computer science education (CSE) is a topic with increasing importance. The traditional way to teach computer science at universities is a classic lecture format, where the lecturer presents the theory to the students in an in-person presentation. While this suits the auditory learning type, it is not the best solution for other learning types. Thus, it has proven valuable to also rely on other teaching methods (Aycock et al., 2019; Pirker et al., 2014). Two of these methods that have grown to become very popular are digital learning and collaborative learning. Digital learning depends on the use of computers to assist the students' learning process (Wheeler, 2012). Collaborative learning focuses on teamwork and acquiring knowledge as a group process. Both of these methods also teach valuable skills for business life. Digital learning familiarizes students with modern technology and prepares them to acquire knowledge autonomously. Collaboration, on the other hand, is crucial for almost every profession, as many companies tend to work in teams. Moreover, communication skills are developed in collaborative exercises (Laal & Laal, 2012). Although these two approaches are fundamentally different, it is possible to combine them into computer-supported collaborative learning. It enables students to collaborate in a digital learning environment, and it merges the advantages of the two methods. However, only a few examples have implemented this approach in large-scale interactive learning environments, and there are even fewer examples that have accomplished collaboration via the internet.

1.1. Objectives

The goal of this thesis is to create a computer-supported collaborative learning environment for CSE. This learning environment is implemented as part of the Maroon laboratory. The Maroon laboratory[1] is an immersive digital learning environment. It provides a set of interactive experiments that allow users to explore different phenomena in an immersive way. However, there is no support for collaborative learning. Therefore, this thesis's first objective is to add a multi-user network to Maroon that allows online collaboration. The system should be fully compatible with the current version of Maroon. Users should be able to host servers, and other users should join them to work on experiments together.

Maroon was developed as a modular learning platform, allowing developers to constantly be widening the spectrum. However, it does not have a section dedicated to CSE yet. Therefore, it is predestined to be extended for CSE. Sorting algorithm visualization is the perfect choice for the first experiment in this section. Sorting algorithms are a key concept in computer science, as they are a basic example to demonstrate the complexity of algorithms. Moreover, visualizing the algorithms can be engaging and educative. Therefore, this thesis's second objective is to create an experiment for Maroon dedicated to the visualization of sorting algorithms.

When teaching sorting algorithms, it is hard to show how efficient an algorithm is (Laxer, 2001). When running in real-time, the simulation is too fast to see a difference between different algorithms. In slower-paced visualizations on the other hand, there are rarely enough elements to display the complexity. Therefore, this thesis's third goal is to create an addition to the sorting visualization that demonstrates the different algorithms' efficiency. This addition will be implemented as a playful challenge. Having such a mini-game in the experiment can also lead to more active participation in the collaborative version (Azmi et al., 2015).

[1] https://maroon.tugraz.at/

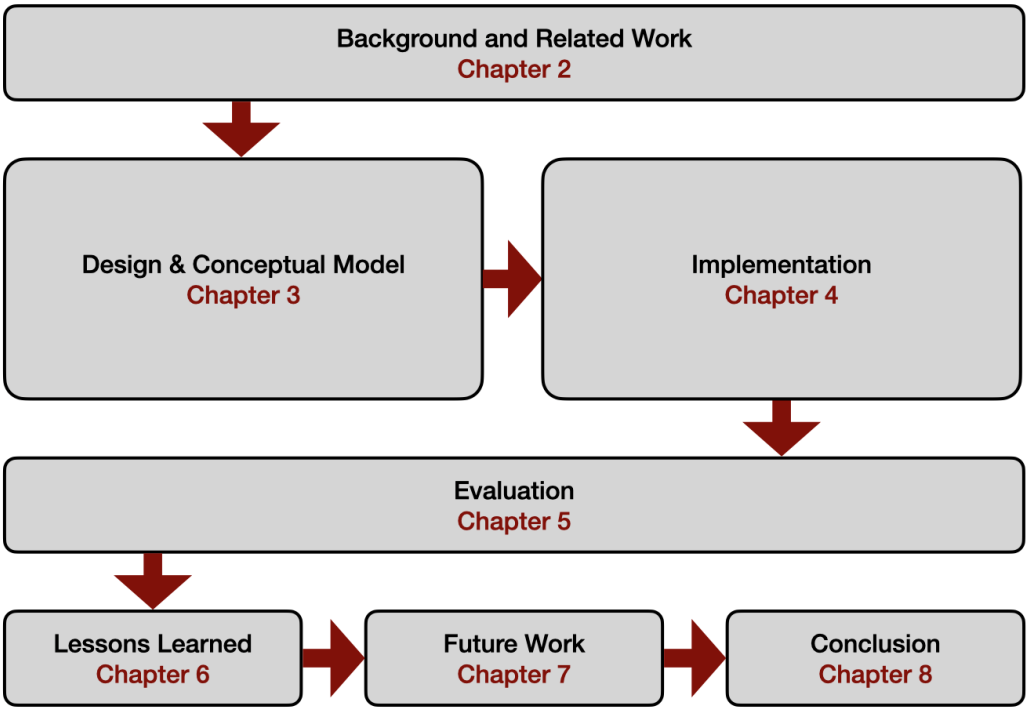


Figure 1.1.: The structure of this thesis.

1.2. Methodology and Structure

This thesis is split into four major parts. The first part focuses on the theory and the related work. The second part contains the design and implementation. The third part is an evaluation of the implementations, and the fourth and closing part summarizes the thesis and gives an outlook for the future. Figure 1.1 illustrates the structure in a diagram.

Chapter 2 gives a theoretical background of digital learning and collaborative learning in the context of computer science education. It introduces the learning environment Maroon, an immersive virtual laboratory. Moreover, it provides fundamental knowledge about sorting algorithms and algorithm visualization and presents related work.

Chapter 3 defines the objectives, the starting point, and the target group for this work. Moreover, it specifies functional and non-functional requirements. It provides background information about various approaches to implementing multi-user networks and justifies the decisions. Furthermore, it presents a design for the sorting experiment and the corresponding challenge.

Chapter 4 presents the implementation details. It defines how a network connection can be established and how the server handles user interactions. It presents a procedure to add multi-user support to new or existing experiments by providing a guideline for future developers. Furthermore, it presents the software architecture and the visualization details for the sorting experiment and presents the sorting challenge as a proof of concept for an interactive multi-user feature.

Chapter 5 analyzes the results of a study carried out to compare the newly created multi-user mode to single-user operation. It describes the methodology and procedure of the research and presents the standardized and environment-specific questionnaires used.

Chapter 6 outlines takeaways of the creation process of this thesis. It presents various challenges that had to be faced on the way and how they were addressed. Chapter 7 depicts multiple desirable features that were beyond the scope of this thesis but might be beneficial to be implemented in the future. Chapter 8 provides a final summary of the most critical aspects of this work.

2. Background and Related Work

2020 will go down in history as the year of the Coronavirus pandemic. The virus forced countries all around the world into a lockdown. Besides closed shops and companies sending their employees into home-office wherever possible, this also meant closed schools and universities. This caused the need for distance learning in a whole new dimension, as students' education, of course, had to be continued. Video conferences were held instead of regular lessons. Homework had to be submitted over the internet, and even exams were mainly carried out online. Overall, a significant shift towards digital learning has happened. Especially for the study of computer science, this is a promising development, as the field is naturally computer-focused. The following section discusses computer science education focusing on digital learning, identifies advantages and disadvantages of digital learning, and presents collaborative learning as a possible solution to address a critical issue.

2.1. Computer Science Education

It is safe to say that the computer has been the most dominant technology change in the last century. Nearly everyone uses a computer daily and most even own one. People use smartphones, tablets, and smartwatches - all miniature versions of computers. The devices are easy to use and become more intuitive every year. With the digital economy's growth comes the search for people that understand how things work in the deeper parts of a computer, behind the fancy windows and graphics. Thus, the industry is always on the lookout for computer scientists. In 2020 computer science jobs in the US grew twice as fast as the average job (University of California, 2020) and there were more than three open jobs per student. Computer

science is not the small, highly complex field that it used to be, where only a few nerds make up the entire field. As computers have become so common, everyone should at least learn the fundamentals of computer science. It has become a vital part of education, from schools to universities (University of California, 2020).

The traditional teaching approach for computer science education (CSE), as in many other degrees, is didactic teaching. The lecturer stands in front of the class and imparts knowledge to the students by presenting and explaining. However, studies show that using other teaching methods that are more engaging and interactive can enhance traditional teaching (Aycock et al., 2019; Pirker et al., 2014). If such methods rely on the use of technology, they belong to digital learning methods.

2.1.1. Digital Learning

Digital learning is also referred to as electronic learning (e-Learning), technology-enhanced learning, or distance learning (Lin et al., 2017; Wheeler, 2012). The primary goal of digital learning is to take advantage of modern technology to support the learning process of students (Wheeler, 2012). Digital learning methods are continually evolving and becoming more and more relevant for modern education. With the Coronavirus pandemic in 2020, they have become crucial, as suddenly students world-wide were forced to study from home. According to Peters (2000) we are at "*the beginning of a new era, in which distance education will develop into an extraordinarily open, flexible and variable form of teaching and learning which can be adapted and adjusted to the learning requirements of students, who will differ greatly from one another with regard to their age, social background and vocational orientation and position. A clear student-oriented form of studies will have been created.*" Especially in computer science, a subject specialized in computers and their software, it seems only logical to also utilize computers for the learning process. And indeed, digital learning plays a significant role in modern CSE. Below we discuss the advantages and disadvantages of digital learning to evaluate its value for CSE.

Benefits. It is hard to identify the benefits of digital learning in general, as it is a versatile field. They are strongly connected to the implementation and can differ for each concrete example. Nonetheless, several authors have identified the following advantages:

- Increased motivation of students and therefore increased time spent with the learning material (Lin et al., 2017)
- Saved time, as students can learn from anywhere at any time they want (Lynch, 2020)
- Reduction of overall costs, as no classrooms are required, and transportation costs can be omitted (Kruse, 2015; Lynch, 2020)
- Increased learning quality (Erhel & Jamet, 2013)
- Specific promotion of students by differentiating the learning experience (Haelermans et al., 2015) and personalizing the learning procedure (Lynch, 2020)
- Positive effects on the learning outcome (Lin et al., 2017)
- High number of people are trained at the same time (Welsh et al., 2003)
- Increased amount of study material that a student remembers (Kruse, 2015; Lynch, 2020)
- Continuous tracking of the gathered knowledge of users (Welsh et al., 2003)

Drawbacks. There are also some drawbacks to digital learning that have to be taken into account:

- Reduced amount of social interaction (Kruse, 2015; Lynch, 2020; Welsh et al., 2003)
- Technical knowledge and equipment needed both by the user and the provider (Kruse, 2015)
- High up-front costs, as money has to be invested for creating the learning application in the first place (Kruse, 2015; Welsh et al., 2003)
- Theory-heavy with few possibilities for interaction (Lynch, 2020)

One of the main identified disadvantages of digital learning is the lack of social interaction. This might also be why Warschauer (2007) observed that while out-of-school education is becoming more powerful and more

popular, studies have shown that paradoxically in-school-learning is becoming more decisive for the learning outcome. Thus, a social component has to be reintroduced to digital learning. This can be accomplished by adding the concept of collaborative learning to digital learning. The next part introduces computer-supported collaborative learning that describes this combination.

2.1.2. Computer-Supported Collaborative Learning

Collaborative learning is a teaching approach that requires a group of people to work together to solve a problem. It is a shift away from the commonly known teacher-centered learning activities used in schools and universities (Laal & Laal, 2012). The combination of collaborative learning and digital learning results in computer-supported collaborative learning (CSCL). "*CSCL is one of the most promising innovations to improve teaching and learning with the help of modern information and communication technology*" (Lehtinen et al., 1999). The created technology systems impart knowledge in a social and interactive way (Halavais, 2016). A particular form of CSCL is online collaborative learning, which utilizes the internet as a connection platform. Already in 2004, Roberts (2004) predicted that this would be the learning approach of the future.

We have to differentiate between asynchronous collaborative learning and synchronous collaborative learning. Asynchronous collaborative learning refers to platforms such as newsgroups, where students can post and reply to messages. However, the asynchronous nature of the communication reduces the flow of the conversation. Therefore, it does not feel natural to the users (Kreijns et al., 2003). In synchronous collaborative learning, users communicate over a live chat, or even better, over voice chat. This makes communication smoother and more natural. Below, we review the benefits and drawbacks of CSCL, especially in comparison to pure digital learning.

Benefits. There are many advantages to CSCL that various authors have identified:

- Group-work skills that are required both in social and work life are developed (Roberts, 2005)
- Groups learn faster and perform better than individuals (Johnson et al., 1990; O'Malley, 2012)
- Social benefits, such as building a diverse understanding and developing learning communities (Laal & Ghodsi, 2012; Roberts, 2005)
- Psychological benefits, such as increased self-esteem and reduced anxiety (Laal & Ghodsi, 2012; Roberts, 2005)
- Academic benefits, such as evolving critical thinking skills and improved classroom results (Johnson et al., 1990; Laal & Ghodsi, 2012; Roberts, 2005)
- Diverse backgrounds and multi-cultural groups (Roberts, 2005)
- Team members motivate each other (Laal & Laal, 2012)

Drawbacks. While CSCL addresses some digital learning problems, most of the drawbacks found in Section 2.1.1 still apply. Additionally, other disadvantages come with CSCL:

- Virtual social interaction cannot fully replace face-to-face communication (Gütl & Pirker, 2011)
- Presence of social interaction enhancing technology does not mean that users automatically make use of it (Kreijns et al., 2003)
- Know-how of the tools in use is required (Chang et al., 2009)

The following part gives some examples for existing learning environments in the setting of digital learning and CSCL.

2.1.3. Digital Learning Environments

A digital learning environment can come in various ways. It can be a simple questionnaire with feedback that a student can fill out to practice learning material up to an entire virtual learning world. Boticki et al. (2013) even presented a smartphone application, teaching users the concepts of sorting

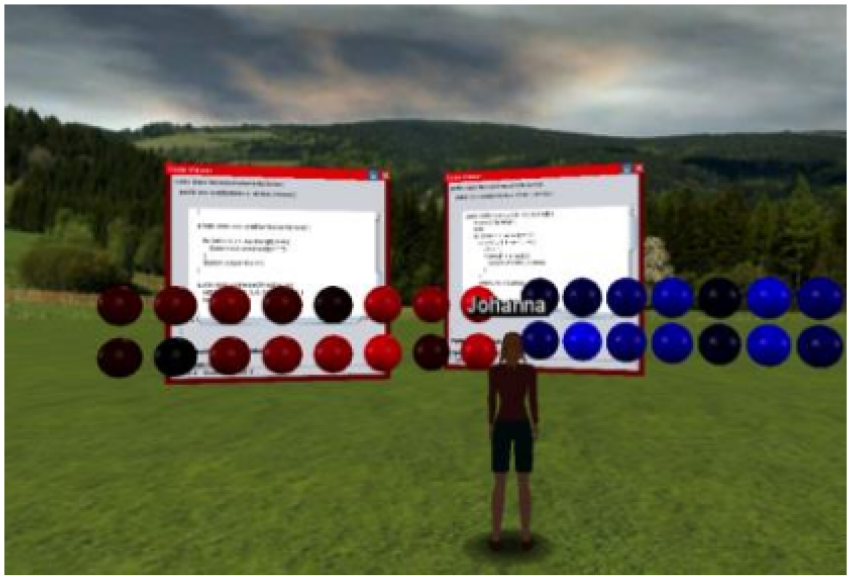


Figure 2.1.: Computer science experiment in the Virtual TEAL World by Pirker (2013). Used with permission.

algorithms. The usage of a smartphone allows even more of the ability for users to study when and wherever they want. The conducted study suggests that students, in general, like the idea of using smartphones for learning. However, it could not prove to have a positive impact on the knowledge gained. While the smartphone application is an example of digital learning, it does not include a collaborative aspect. An example of a learning environment focused mainly on CSCL is PeerSpace (Li et al., 2011). It is an online learning platform that supports both synchronous and asynchronous communication. It was designed as an assistive tool for computer science students in addition to their university education. They showed that students are more socially engaged with their peers when using this additional collaborative platform.

An example of a 3D learning world is the Virtual Teal world, created by Pirker (2013). It is an immersive and engaging digital learning environment designed for various interactive virtual experiments. The environment includes physics experiments as well as CSE-specific experiments, as shown in Figure 2.1. Moreover, it embraces collaborative learning, as groups of 3

to 9 perform activities together, but it does not allow collaboration via the internet. Such interactive learning platforms primarily address the disadvantage that digital learning is too theory-heavy. Based on this approach, Pirker (2017) developed a stand-alone virtual laboratory and experiment environment called Maroon. It is an interactive learning environment that uses different experiment rooms to impart knowledge of various topics. The following section presents the Maroon project in more detail.

2.2. Maroon

The Maroon laboratory[1] is a virtual laboratory that gives students at schools and universities an opportunity to experience science phenomena first-hand. It is a modular learning management tool, which allows content creators to extend the laboratory with various experiments from different fields. The current laboratory version contains seven physics experiments with a focus on electromagnetism. Pirker (2017) started the project at the University of Technology Graz. In 2017 it won the GOLC Award for the "Best Visualized Experiment", and in 2018 it was featured in Forbes Magazine. The Maroon project is open-source and available via GitHub[2]. It has gone through many stages of development, and today it comes in different versions. The most recognized part is the virtual reality (VR) version of the laboratory (Pirker et al., 2017). Users can explore it by the use of a VR headset, such as the HTC Vive. The second version is a desktop version that can run on any Windows or Mac computer. It allows users who do not have a VR headset to benefit from the learning platform. Additionally, there is a WebGL version obtainable via the browser. While the performance is worse than the downloaded version's, it can be more easily accessed.

The laboratory is implemented in Unity[3], as Unity already provides the necessary technology for 3D and 2D virtual worlds. It also comes with various assets that allow easy integration of VR. Unity also allows the managing of the distinct versions inside of one big project. The VR version

[1]https://maroon.tugraz.at/
[2]https://github.com/GameLabGraz/Maroon
[3]https://unity.com/

Figure 2.2.: The entering room of the Maroon laboratory. Screenshot taken from GameLab-Graz (2020).

and the desktop version differ fundamentally in that all controls have to be adapted for the corresponding platform. A 2D user interface that works well for the desktop is considered a poor VR choice in many cases. Therefore these two versions are kept separate inside the project. However, assets and code are shared across these versions. The WebGL version and the desktop version use the same controls, and therefore, these two versions are treated as one inside Unity. The *Unity Build Target* controls which version Unity builds from the project.

2.2.1. Maroon Laboratory

When entering the laboratory, users find themselves in a room with various experiments to their left and right side (Figure 2.2). This room acts as a 3D menu as users can walk up to these experiments and on-demand, enter a different scene that implements the functionality. Inside, users have the opportunity to explore the presented science simulation and interact with it. The experiments are implemented independently, each experiment in a separate scene. This modular system allows adjusting existing work without unintended side effects, making it easy to add new functionalities to Maroon.

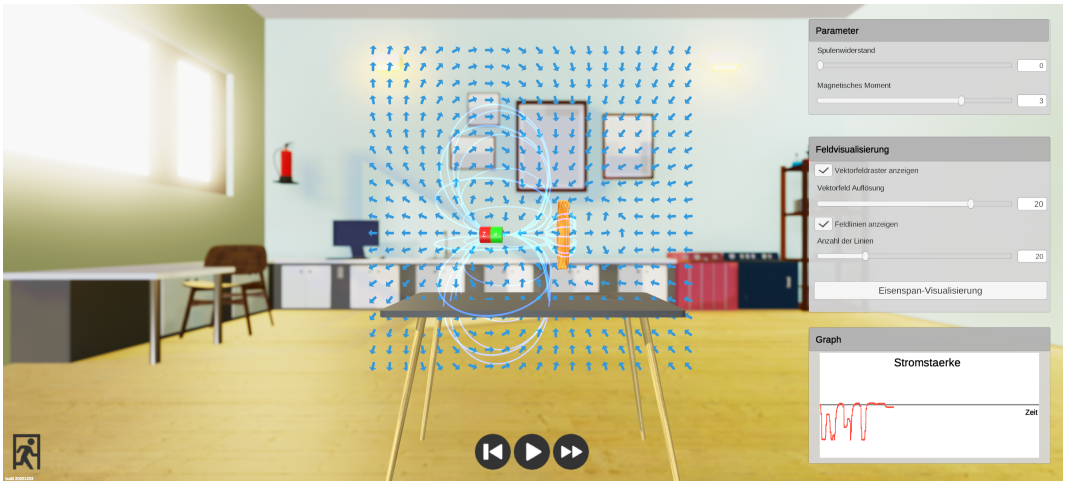


Figure 2.3.: The experiment view of the Faraday's Law experiment. Screenshot taken from GameLabGraz (2020).

2.2.2. Maroon Desktop & Browser

In the desktop and browser versions, users move around the laboratory with the keyboard and control the camera view-direction with the mouse. These controls are well-established in first-person computer games and provide an intuitive way of movement. The character controller[4] used in Maroon already comes with Unity out-of-the-box. Inside the experiment scenes, the camera is static, with the experiment setup placed in the center. On the right side is a user interface that lets users tweak the parameters or display valuable information. In Figure 2.3 the layout of the Faraday's Law scene is illustrated. As one of Maroon's current experiments, it showcases the described design. The control buttons are at the bottom of the screen. These control the flow of operation, as they allow the user to start and stop the simulation. Moreover, they can reset the simulation to a neutral state and, if paused, they can simulate frame by frame. The exit button, which brings users back to the main menu, is located in the lower-left corner. All newly added experiments should adopt this functionality. There is space for more user interfaces on the left side, which has for example been used to integrate an assessment system into Maroon.

[4]https://docs.unity3d.com/ScriptReference/CharacterController.html

2.2.3. Maroon Experiments

As discussed above, the experiment setup differs fundamentally between the two versions. However, it would be highly inefficient to develop two completely independent versions of an experiment. Therefore, the experiment design in Maroon follows specific rules. The core 3D visualization is platform-independent and is usable for both versions. Only the way how users interact with the experiment differs. A crucial part of the experiment implementation is the *Simulation Controller*. It is an integrated part of Maroon and controls whether the simulation is currently running or stopped. There is a template experiment scene available that already contains all standard components. It already connects the control buttons mentioned above to the corresponding features of the *Simulation Controller*. When creating a new experiment, the developer should use the template scene. Another standardized procedure is reset-handling. It should be possible to reset the simulation to a state that allows the user to start the experiment anew. To achieve this, Maroon has a script named *IResetObject*. It is an interface that each resettable object should implement. In practice, this means that a *ResetObject* method has to be added, which defines what to do with the object when users press the reset button.

So far, all experiments in Maroon are educating in the topic of Physics. However, a new series of experiments dedicated to CSE should be added. A must-have topic for this new category is sorting, as it is a classic example for comparing algorithms and their efficiency. The following section discusses the fundamentals of sorting and deals with algorithm visualization.

2.3. Sorting Algorithms

When starting education in computer science, it usually does not take long until students learn about sorting algorithms. This is not only the case because sorting is an important task that appears a lot in computer science. It is also an excellent way to learn about algorithms in general. It introduces

students to the concepts of algorithm complexity and teaches them programming paradigms. All in all, this explains why sorting algorithms are so prevalent in the early stages of CSE.

2.3.1. Sorting Fundamentals

There are various methods how to sort an ordered field of items. Putting the idea of how to sort it into concrete computer-interpretable rules transforms the idea into an algorithm. Comparing different algorithms is tricky, as the performance of an algorithm depends on the task. One algorithm might be especially good at sorting big fields in random orders, whereas another algorithm might outperform the first one on a smaller input that is nearly sorted (Estivill-Castro & Wood, 1992; Mishra & Garg, 2008). However, some measures allow comparing and classifying of sorting algorithms. These measures are mostly applicable to other kinds of algorithms as well. They are one reason students learn about sorting in the first place, as they are a fundamental CSE concept.

Complexity. The runtime (in seconds) of an algorithm depends on various factors, such as the field's size, the machine's computational power, and the unsorted elements' order. Thus some other way to measure the efficiency of an algorithm has to be found. Therefore the efficiency of a sorting algorithm is usually compared by its time complexity, denoted in the big O notation.

The big O notation gives an upper bound for the time complexity, dependent on the input field's size. The exact definition of the notation is the following: "$O(f(n))$ *denotes the set of all* $g(n)$ *such that there exist positive constants* C *and* n_0 *with* $|g(n)| \leq Cf(n)$ *for all* $n \geq n_0$" (Knuth, 1976). The notation has the advantage that all constants fall out of the equation, so there is no difference between $O(2n)$ and $O(n)$. This makes the time complexity of different algorithms more comparable, as these constants depend on the implementation, whereas the big O notation gives an implementation-independent measure.

Worst Case, Best Case, and Average Case. Apart from the size of the input field, the efficiency of a sorting algorithm can also vary with the order of the input field. An algorithm might perform well on a field that is almost sorted, but perform poorly on a reversed field. The same can happen the other way around. To take this into account when comparing different sorting algorithms, the time complexity of the algorithm is usually determined for three different cases. The **worst case** can be constructed manually and describes the case for which the algorithm has the worst time complexity. The same goes for the **best case**, which might even be that the field is already fully sorted. In the **average case** the field is equally distributed. In most scenarios the worst case and the average case are the interesting ones to look at. For many algorithms these two also have the same time complexity.

Stability. In a sorting field, a sorting element may appear multiple times. A sorting algorithm is defined as stable if it leaves the order of these elements the same as they were in the unsorted field (de Gouw et al., 2014). This is especially important if sorting by more than one value. Sorting algorithms can be classified into those that are stable and those that are not.

Adaptivity. A sorting algorithm is adaptive if it benefits from specific input orders. This means that the best-case time complexity is better than the average case's. Adaptive algorithms are exceptionally efficient in cases where they benefit from nearly sorted orders, as these orders frequently appear in practice (Estivill-Castro & Wood, 1992).

In-Place vs. Not-In-Place. Another criterion by which to compare sorting algorithms is memory usage. An algorithm works in-place if it sorts the field in the same memory location that it already occupies, and it does not need additional memory. This does not mean that it can not cache a single value, but there is never the need to store larger chunks of memory to a different location. Strictly speaking, the additional memory usage must not depend on the input field's size for the algorithm to be considered

in-place (Franceschini & Geffert, 2005). For some algorithms, it depends on the implementation if they are in-place or not.

Comparison Sorts vs. Non-Comparison Sorts. Closely related to in-place vs. not-in-place, algorithms can be classified by the way that they sort. Comparison-based algorithms are most common, and they operate by comparing the sorting elements to each other and arranging them accordingly. However, it is also possible to sort a field without comparisons. An example is the Radix Sort algorithm that Section 2.3.2 presents. For comparison-based algorithms, it has been proven that their average case time complexity can not be better than $O(n \log n)$, as there are at least $O(n \log n)$ comparisons necessary. If such an algorithm achieves this complexity for the worst case, it is called worst-case optimal. Non-Comparison sorts can even be linear ($O(n)$) under specific conditions.

Iterative vs. Recursive. It is hard to classify sorting algorithms into iterative and recursive ones, as for most of them, this depends on the implementation. Therefore this category is more of an implementation choice. While the recursive implementation is usually easier to understand, the iterative implementation needs slightly less memory, as it does not have to store return addresses. The complexity usually does not differ.

2.3.2. Selected Algorithms

This section presents nine algorithms that are of particular interest to CSE. These algorithms are either especially important or unique. An algorithm is regarded as important if widely used in practice. But important can also relate to simple algorithms commonly used as introductory examples. The end of this section compares the complexity and stability. Appendix A contains the implementation details and pseudo-code. The following explanations are based on Cormen et al. (2009).

Insertion Sort. Insertion sort starts with the second element, compares it to the first, and arranges them accordingly. Then it takes the third element and inserts it in the correct position. This is iteratively repeated for the whole field, so the front elements are always sorted. With an average time complexity of $O(n^2)$, insertion sort is usually not feasible in practice. However, it is one of the first algorithms taught to students when they learn about sorting. This is due to the intuitive way it can be implemented.

Merge Sort. Merge sort splits the elements into two halves. It recursively splits these halves further and sorts them individually. Once it can not split the halves any further, or the two parts have already been sorted, it merges them. Merging happens by continually comparing the leftmost elements of both halves and arranging them accordingly. Merge sort is a divide and conquer algorithm, which means it divides the problem into more minor ones until it can solve them. It is worst-case optimal, but it is not stable.

Heapsort. Heapsort uses the maximum heap data structure to sort the elements. This data structure always has the maximum element as its root element. The algorithm first builds a maximum heap of all elements. It then moves the root to the last unsorted position and again builds a heap of the remaining elements, using the fact that they are already pre-sorted. This is iteratively repeated until all elements are sorted, so the back always contains the sorted elements. Schaffer and Sedgewick (1993) proved that the time complexity is about $\frac{1}{2} n \log n$ in best case, about $n \log n$ in average case and $n \log n + O(n)$ in worst case. So using the big O notation heapsort sorts in-place in $O(n \log n)$ time for all inputs, which means it is worst-case optimal.

Quicksort. Quicksort sorts the elements using a pivot element. It moves all elements smaller than the pivot to the left of the field. All bigger ones move to the right. It then places the pivot in the middle and sorts the left and right parts recursively. Just as merge sort, quicksort uses the divide and conquer principle. It builds upon the premise that it is easy to sort smaller chunks of data if there is a divisional line that separates the data.

Therefore it manually introduces this line utilizing the pivot element (Hoare, 1962). The efficiency of quicksort depends on a smart choice for the pivot. A standard option is to use the last element of the field, but especially for (nearly) sorted or reversed fields, this leads to a complexity of $O(n^2)$. The same happens if the first element is chosen as a pivot. Ideally, the pivot is chosen randomly, but selecting a random element each time also requires computational effort.

Selection Sort. Selection sort scans through all the elements to find the smallest one. It moves the selected element to the first unsorted position and iteratively repeats the procedure until finished. The popularity of selection sort certainly does not come from its time complexity of $O(n^2)$, even in the best case. Instead, it is due to its ease of understanding and implementation, as always looking for the next fitting element is intuitive.

Bubble Sort. Bubble sort goes through the elements from left to right, constantly comparing it to its right neighbor. If an element is bigger than its neighbor, it swaps them, so the larger element moves to the right like a rising bubble. After each iteration, the biggest element reaches the rightmost position. The process is repeated until all elements are sorted. Bubble sort is a classic sorting algorithm that is commonly used in CSE. The importance has grown historically, as it has been described and analyzed in various researches (Astrachan, 2003). However, experts question the importance of bubble sort, as it is neither efficient nor as intuitive as, for example, insertion sort. "*In short, the bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems*" (Knuth, 1998).

Gnome Sort. Gnome Sort works by continually comparing two neighbors. One can imagine a little gnome starting at the most left element. If the right neighbor is bigger than the element, it swaps them and takes a step to the left. If it is smaller than the element, it moves to the right. This is repeated until it moves past the rightmost element, then all elements are sorted. Sarbazi-Azad (2000) initially presented gnome sort under the name

'Stupid Sort'. It was designed to be the most simple sorting algorithm in existence. However, the time complexity of $O(n^2)$ on average shows that it is not an efficient algorithm (Hammad, 2015).

Radix Sort. Radix sort moves the elements into different buckets to sort them. In the first pass, it puts each element into the bucket corresponding to the one digit. After that, it puts the elements back into the original field by emptying the buckets in ascending order. In the second pass, it sorts the elements by the tens digit, the third pass by the hundreds digit, and so on. This is repeated as often as the largest element's number of digits. Radix sort is the only selected algorithm that is not comparison-based. Therefore the lower bound for the time complexity of comparison-based algorithms does not hold for this one. Instead, the algorithm purely depends on the input size n and the largest element's size. For each of the d digits, one sorting iteration has to be performed. Therefore the time complexity is $O(dn)$ for all inputs. If the maximum size is known, it can be regarded as constant, and the time complexity becomes linear ($O(n)$).

Shellsort. Shellsort defines a gap between two element positions. It compares each element to the element that is positioned this gap away. It swaps the elements if necessary. After each iteration, the gap is halved. Once the gap reaches zero, sorting is finished. Shell (1959) originally introduced Shellsort, which is where the name originates. It is also known as 'Diminishing Increment Sort' because the increment (the gap) is reduced in every iteration (Knuth, 1998). While being easy to implement without any recursions, it is still effective regarding time complexity.

Comparison. As pointed out at the beginning of the section, it is hard to rank algorithms, as their performance is highly task-dependent. However, it is possible to compare the different metrics defined in Section 2.3.1. In Table 2.1 the time complexity and stability of the selected algorithms are compared.

	Worst Case	Best Case	Average Case	Stable
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$	✓
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	✓
Heapsort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	x
Quicksort	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	x
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	✓
Bubble Sort	$O(n^2)$	$O(n)$	$O(n^2)$	✓
Gnome Sort	$O(n^2)$	$O(n)$	$O(n^2)$	✓
Radix Sort	$O(dn)$	$O(dn)$	$O(dn)$	✓
Shellsort	$O(n^{3/2})$	$O(n \log n)$	$O(n^{4/3})$	x

Table 2.1.: Comparison of the selected sorting algorithms (Cormen et al., 2009; Hammad, 2015; Kazim, 2017; Mishra & Garg, 2008).

2.3.3. Sorting Algorithms in CSE

"*Algorithms and data structures . . . are at the heart of computer science, being the building blocks of any software*" (Baeza-Yates, 1995). Thus, it is an important task to start teaching the fundamental concepts of algorithms in the early stages of CSE. Introducing the concept of complexity is challenging, as computers have become so fast that students hardly realize if their code is efficient (Laxer, 2001). Traditionally teaching algorithms with chalk on the blackboard can be a challenging task. As algorithms are dynamic, it is hard to catch their essential aspects in a static drawing. Thus, experts agree that it is a good idea to use some form of dynamic algorithm visualization in the learning process (Baecker, 1998; Battistella et al., 2017; Naps et al., 2002).

Algorithm Visualization. Algorithm visualization refers to visualizing the steps of an algorithm on the computer. It is a common example of digital learning. However, studies have shown that visualization does not have to be of educational value. Instead, it depends on the implementation, if it improves the learning process (Hundhausen et al., 2002).

Naps et al. (2002) address this problem by defining eleven best practices for algorithm visualization:

1. Provide additional resources that put the visualization into a context.
2. Adapt to the user's knowledge level.
3. Provide additional views, such as an animation of the source code that displays the execution status.
4. Include information about the performance to allow conclusions about the efficiency.
5. Include an execution history, so learners can comprehend how they got to the current state.
6. Support forward and backward execution, so learners are always in full control.
7. Allow users to build custom visualizations.
8. Allow users to modify the input-data to engage them more actively.
9. Dynamically challenge users by asking questions, which force them to reflect on the tasks frequently.
10. Give the users feedback on their activities dynamically.
11. Add explanations to the visualization for more background information.

Following these practices ensures that a created algorithm visualization is not only a fancy animation but also a valuable contribution to CSE. A popular field to apply algorithm visualization to is sorting algorithms.

Sorting Visualizations. The field of sorting algorithms is very well-researched, and there are numerous ways to visualize the different sorting algorithms and how to best teach the differences between them. The most common approach to visualize sorting algorithms is to use bars of different lengths and have them sorted by the different algorithms. This form of visualization goes back to Baecker (1981), a film that laid the basics for many algorithm visualizations (Dershem & Brummund, 1998). The film ended with a race where all algorithms sort the same array side by side at the same time. This kind of comparison makes it easy to examine the run times of the different algorithms. It is still widely used nowadays. However, most of the time, it is hard to see how the algorithm works in the background. Examples

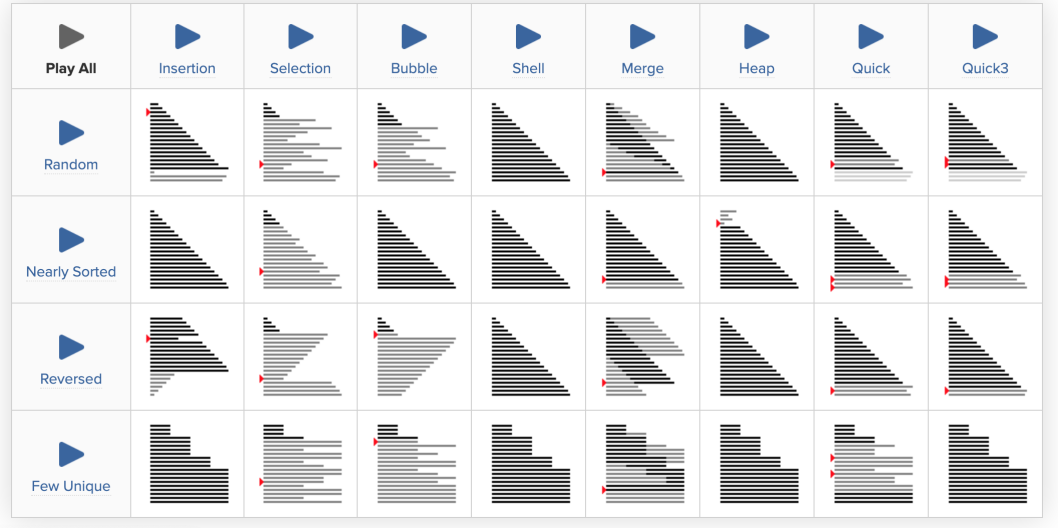


Figure 2.4.: Sorting algorithms visualised by bars of different lengths in the implementation of Toptal (2020). Screenshot by author.

for modern implementations of this kind are Toptal (2020), displayed in Figure 2.4, or Macrae (2016).

Zapponi (2014) takes a more artistic approach that displays the sorting algorithms in a very pleasing way. He permanently shows the swaps that have taken place as curved lines, as presented in Figure 2.5. This lets the user spot patterns that are hard to observe with the bar length method. The website also allows running several algorithms at the same time for easy comparison. However, when it comes to algorithms that usually do not sort in place, such as radix sort, the visualization is not intuitive.

Schnurr (2017) found another visually appealing solution. He displays the sorting algorithms as images, where the first line of pixels contains randomly distributed pixels of a color gradient. The pixels are then sorted by color using the different algorithms that the image should visualize. Each line of the image corresponds to one step of the sorting algorithm. His display method is very engaging and lets the user recognize patterns that show how the algorithm is working. Figure 2.6 shows two of the results.

Buchbauer (2019) presented a more immersive approach. He selected nine different sorting algorithms that should be taught to the users. One part

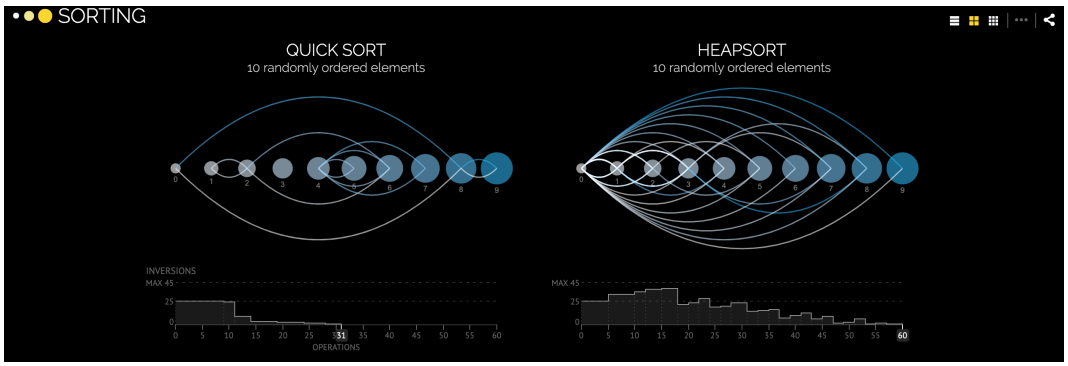
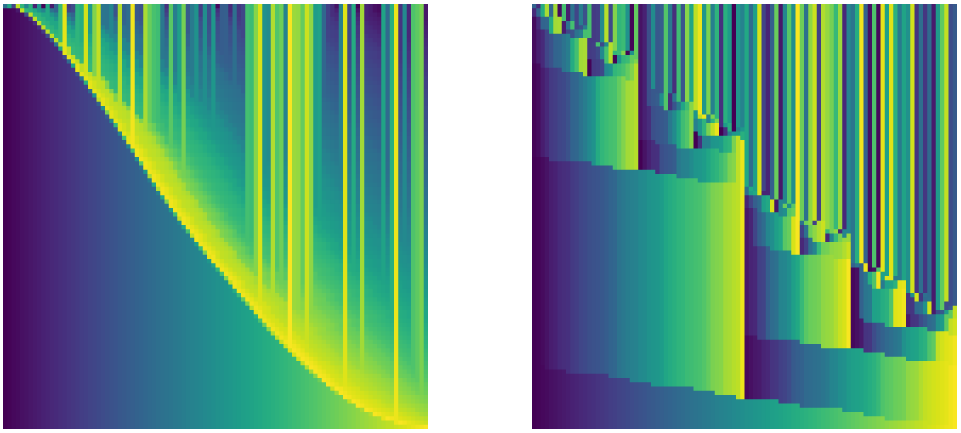


Figure 2.5.: Sorting algorithms visualised by Zapponi (2014). Screenshot by author.



(a) Bubble Sort

(b) Merge Sort

Figure 2.6.: Algorithms visualised by Schnurr (2017) under license ("GNU General Public License, version 3," 2007)

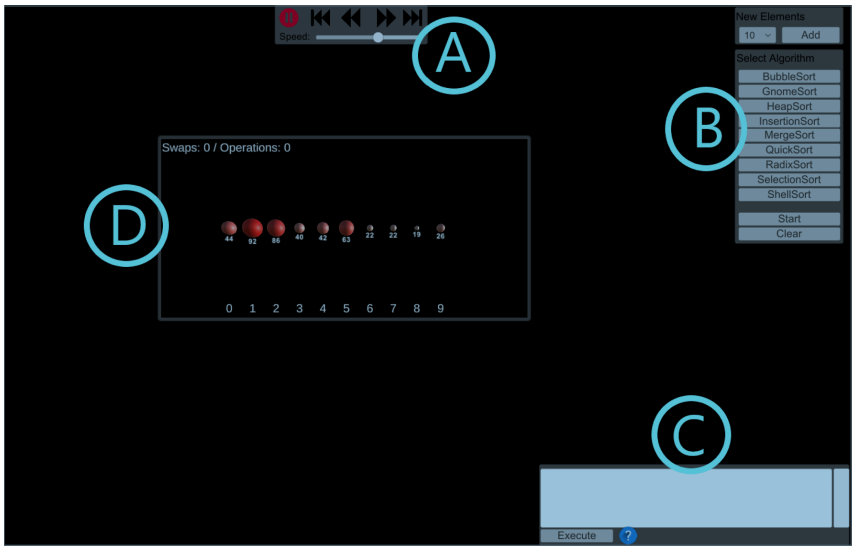


Figure 2.7.: Web application to visualize sorting algorithms by Buchbauer (2019).

of his work is a web application that allows users to go through the algorithms, step by step. Figure 2.7 shows the design of this web application. Moreover, he created a VR version that showcases the algorithms in a virtual world. He conducted a study to compare the two versions and gather user feedback for the visualization. Overall the feedback was very positive and encouraging. However, a point of criticism was that the two versions were too fundamentally different to compare them.

2.4. Summary

With the growth of the digital economy, CSE is an increasingly important task. Modern techniques, such as digital learning and collaborative learning, can enhance conventional teaching methods. They build on the premise of making the learning process more engaging to increase the learning motivation. Different authors prove that this can be achieved (Boticki et al., 2013; Li et al., 2011; Pirker, 2013). A notably immersive digital learning environment is the Maroon laboratory (Pirker, 2017). However, it does not allow for collaborative learning. There are only a few digital learning

platforms that support online collaboration. Another part that is missing in the Maroon laboratory is a section dedicated to CSE. A prevalent digital learning concept in CSE is algorithm visualization, as it allows to impart knowledge in a way that is not possible with traditional teaching methods. Sorting algorithms are a popular choice to learn the basics of algorithms. Therefore, this chapter presented fundamental knowledge about sorting algorithms. It outlined how to create a successful algorithm visualization and introduced existing ones. The following chapter defines the requirements for the Maroon extensions that we implement as part of this thesis. It outlines the theoretical background of various options for implementing multi-user networks in Unity and presents the design decisions.

3. Design & Conceptual Model

This chapter defines the objective, the target group, the functional and non-functional requirements, and the implementation's starting point. It discusses different design choices in the network part and the sorting experiment and justifies our decisions.

3.1. Starting Point & Objective

The Maroon laboratory presented in Section 2.2 is an immersive virtual learning environment, offering users the opportunity to explore interactive experiments. However, it does not offer the option to collaborate with other students. As Section 2.1.2 describes, a digital learning environment can significantly benefit from the addition of collaborative learning. Therefore, this thesis's primary goal is to create a multi-user network for Maroon, allowing online collaboration. This means that multiple users, from multiple computers, should enter the laboratory together and collaboratively experience the experiments. Part of this work is to equip existing experiments with multi-user support.

Additionally, this thesis aims to design an experiment dedicated to computer science education to expand Maroon's repertoire. As Section 2.3 points out, sorting algorithms are a good starting point to include computer science experiments, as they are fundamental to CSE and have the opportunity for exciting and educational visualizations. The experiment should serve as a proof of concept for the network part. In addition to the multi-user functionalities that all other experiments get, this work aims to add a more interactive multi-user feature. This feature should be implemented as a small game inside the sorting experiment to achieve more active engagement, an

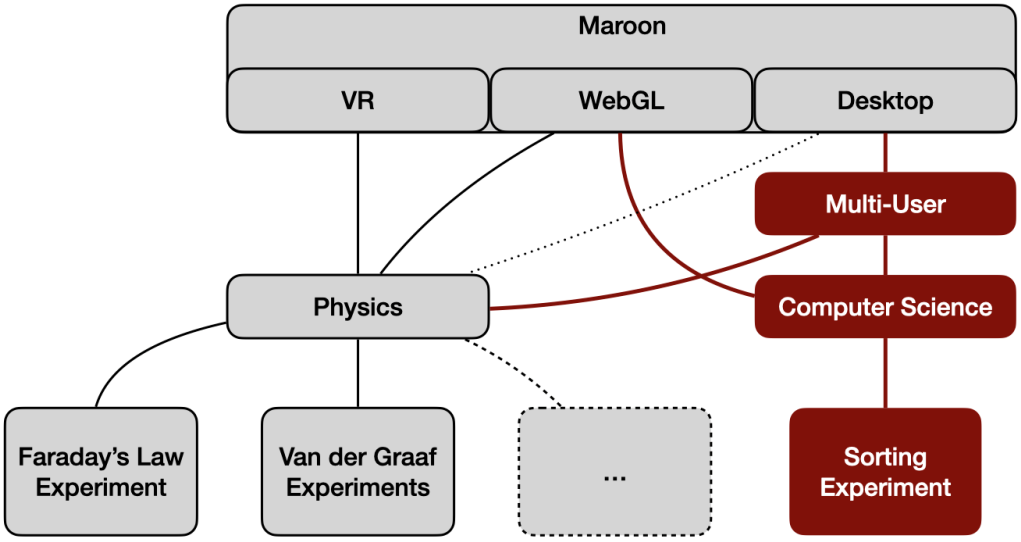


Figure 3.1.: A conceptual model of Maroon. All elements marked in red should be added as part of this thesis.

essential topic for algorithm visualization (Azmi et al., 2015; Naps et al., 2002). The mini-game should consist of a challenge, where all connected users can compare their assessment of different sorting situations.

As described in Section 2.2, Maroon comes in three versions, namely VR, WebGL, and desktop. The multi-user network and the sorting experiment should be implemented in the desktop version. The multi-user concept might be adapted for the other versions in the future (Chapter 7). Figure 3.1 displays a conceptual model of Maroon's setup, where all parts that should be implemented as part of this thesis are marked in red.

3.2. Target Group

As described by Holly (2019), the Maroon laboratory was designed for students of schools and universities as an addition to the standard lessons. With the online multi-user functionality, they have the additional option

to work on the experiments from home collaboratively. This allows group work exercises, even if it is impossible for the groups to meet in person.

The sorting experiment mainly focuses on computer science students. It can also be used for fundamental computer science education in schools, but it is most beneficial for students who just started their computer science degree. They might profit from the new computer science section of Maroon by either having it integrated into their lectures or by using it as an additional learning resource.

3.3. Requirement Analysis

Before starting the implementation of a project, it is necessary first to define the software's requirements. It is common practice to split these requirements into functional and non-functional ones. As the name suggests, functional requirements describe the functionalities that the created software should implement. Non-functional requirements, on the other hand, do not describe specific behavior but focus on more general concerns (Sommerville, 2007).

3.3.1. Functional Requirements

The identified functional requirements are based on the research performed in Chapter 2. For the sorting experiment and sorting challenge, the best practices for algorithm visualization by Naps et al. (2002) were taken into account.

1. Multi-User Network
    a) Users should be able to create a server, that other users can join
        i. over the local network.
        ii. over the internet.
    b) The created server should be published so that other users can see it.

   c) Users should be presented with a list of all available servers to join any of these.
   d) Existing experiments should be equipped with multi-user support if sensible.

2. Sorting Experiment
   a) Users should be able to
      i. control the flow of the sorting procedure. This includes step-wise execution both forward and backward.
      ii. control the size of the sorted field.
      iii. choose from various sorting algorithms.
   b) A highlighted pseudo-code should be displayed to the user that links the code to the visualization.
   c) The executed swaps and comparisons should be displayed to give users a notion of the execution history.
   d) A short description of the selected algorithm should support users' understanding.

3. Sorting Challenge
   a) It should be possible to compare algorithms by running them side by side on the same input data.
   b) Users' engagement should be increased by adding a competitive element.

3.3.2. Non-Functional Requirements

The non-functional requirements have been defined separately for the multi-user part and the sorting experiment as these parts differ fundamentally. The sorting challenge's non-functional requirements are already covered by the sorting experiment, as the challenge should be implemented as a part of the experiment.

Multi-User

1. **Availability.** The multi-user service should be available via the internet at all times.
2. **Scalability.** The system should support multiple users per server and multiple servers. One school class, about 30 people, should be able to collaborate on a server at once.
3. **Compatibility.** The network system should be fully compatible with the current version of Maroon. None of the current functionalities should be influenced negatively while the network is disabled.
4. **Usability.** The whole multi-user system should be intuitive to use. This includes joining and creating a server, interacting with other users and elements on the server, and leaving the server. User interfaces should be appealing to the user while matching the current style of Maroon.
5. **Maintainability.** The developed system should be easy to maintain, even by people who were not involved in the development process.
6. **Reliability.** The down-time of the network service should be minimal.
7. **Performance.** Users should not perceive performance issues. Lags and latency should be minimized.
8. **Security.** It should be possible to protect a server from uninvited joins.
9. **Integrability.** It should be easy to integrate the network component into future experiments.
10. **Reusability / Configurability.** Most of the system's parts should be reusable or at least configurable to implement virtual reality multi-user in the future.
11. **Extensibility.** It should be possible to extend the network with additional functionalities, such as a voice chat component if needed in the future.

Sorting Experiment & Sorting Challenge

1. **Usability.** The experiment controls should be intuitive to use. All experiment parts, including the user interfaces, should be visually pleasing while fitting the style of Maroon and the other experiments.
2. **Scalability.** The experiment should be usable by multiple users at once by integrating the multi-user network that is also part of this thesis.

3. **Performance.** Users should not perceive lags or other performance issues when using the experiments. The execution speed of the experiment should be frame-rate independent to allow fluent multi-user integration.
4. **Reliability.** The implementation should manage resources carefully to prevent an overload shutdown of Maroon.
5. **Compatibility.** The experiment should be compatible with the current Maroon version, and it should be integrated into the current laboratory.
6. **Maintainability.** The implementation should be easily understandable to allow maintenance.
7. **Reusability / Configurability.** It should be possible to reuse the algorithms and visualizations to create a virtual reality version of the experiment in the future.
8. **Extensibility.** It should be possible to add additional sorting algorithms to the visualization without too much effort.

Overall, the goal is to seamlessly integrate all new functionalities into Maroon while achieving good usability and performance. As one aim of this work is to implement a multi-user network in Unity, the following sections will discuss the available options before making the design decisions.

3.4. Network Architectures

Before tackling the different ways how to integrate a network component into Unity, it is good to know the differences, advantages, and disadvantages of the available architectures. This section gives an overview of the findings, compares them, and presents the selected choice. The following explanations and figures are based on Bharambe et al. (2008), Douglas et al. (2005), Schollmeier (2001), Stagner (2013), Unity (2018).

3.4.1. Local Network

The simplest solution for multi-user programs is to have the users physically close to each other. Having all users use the same machine is an example of

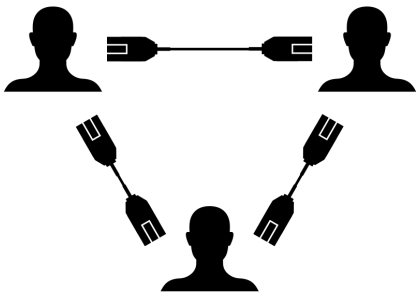


Figure 3.2.: Users connected via LAN.

this architecture. It is used in various multiplayer games, where each user has their own controller but plays on the same screen. This only allows a minimal number of players, as the screen space is restricted, but everyone can play simultaneously with no latency.

The more scalable local option is to have each player on a separate computer and connect them via a local area network (LAN), as illustrated in Figure 3.2. This can be done by physically connecting the players via LAN cables or over a router, supporting Wireless LAN. With this method, many players can simultaneously play together, and the network setup complexity is minimal. The advantage over online architectures is that there is hardly any latency, as everyone is close together. However, this is also the most significant disadvantage, as the players have to come together, which makes the reach very small. Depending on the project, a local multi-user setup can be just fine, but as soon as users are not at the same location, utilizing the internet is the way to go.

3.4.2. Dedicated Server

As the name suggests, this architecture relies on a server (can also be several servers) responsible for all appearing network traffic. Each party connects to this server only, so all the server logic is in one place. Figure 3.3 illustrates the architecture. Having all the logic in the same place makes it

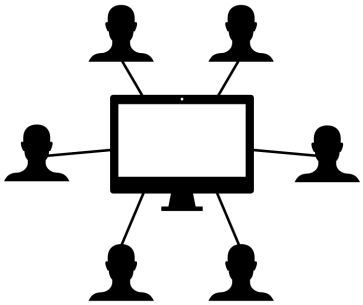


Figure 3.3.: The architecture of a dedicated server architecture, where the server is physically independent of the users.

easy to prevent people from misbehavior like cheating. Also, it is by far the best scaling option, and the latency is fairly low. Thus, it is nowadays the only reasonable choice for large-scale competitive games. The two major drawbacks are cost and complexity. In order to set up a dedicated server architecture, a full network stack has to be implemented. Moreover, a server is needed, which usually leads to rent and maintenance costs.

3.4.3. Peer to Peer (P2P)

"*A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P, ...) network, if the participants share a part of their own hardware resources (...). These shared resources are necessary to provide the Service and content offered by the network (...): They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (...).*" (Schollmeier, 2001). The essential benefit of P2P is that no additional server is needed to connect the players. Therefore there are no renting costs and no maintenance costs. The problem, especially when looking at games, is that there is no security whatsoever. As each player has the same rights and there is no central authority, it is hard to detect and prevent cheating users. There are various ways how to implement a P2P network.

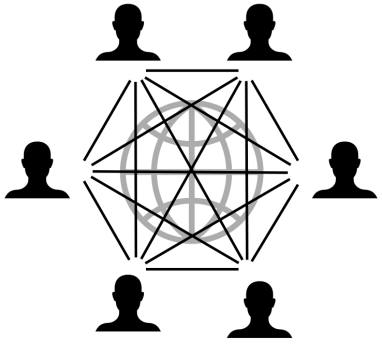


Figure 3.4.: The architecture of a direct peer to peer network.

Direct P2P. Figure 3.4 shows the architecture of a direct P2P network, where each player connects to all other players. This means that each player acts both as the server and the client at the same time. It is robust against players dropping out, but it is tough to keep all the peers synchronous. With each additional user, the system becomes more complex, so it does not provide much scalability. Typically the number of users is limited to around 10 to 20 concurrent users. The system's advantage lies in the peers' equality, as each peer has the same rights. The latency, however, is one of the drawbacks of this system. It is very high, as there are so many connections, and it may vary from player to player.

Client-Server P2P. In the client-server P2P network, one user acts as the server, called the host, and all other users connect to this peer as clients (Figure 3.5). This makes the network much less complex, as each peer only has to connect to the host. However, this introduces some new problems. The host has an advantage over the other users, as it has no latency and total control over everything. Especially in competitive games, this is a massive problem. Moreover, the whole program has to be remigrated to a new host or shut down when the host leaves. The remigration process is complex, and the program has to pause until completed. Therefore this architecture can only be used when there is a reliable host and when the latency is not a significant issue.

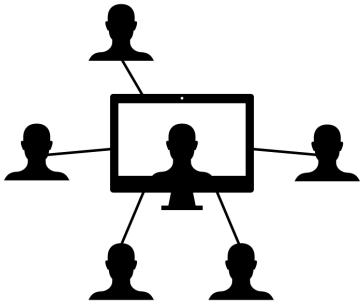


Figure 3.5.: The architecture of a client-server peer to peer network, where one user acts as the host.

A special kind of client-server P2P network is the Client-Server Relay P2P Network. It still utilizes one player acting as the server, but this time the other players connect to the host over a relay server (Figure 3.6). This reduces the host-advantage, but it is a worse solution regarding latency, as it adds another network node to cross. Moreover, the added relay makes the architecture substantially more complex. The advantage over the pure host system and the reason this architecture was established is the increased reach, as it presents a solution to overcoming the network address translation. The following section describes this problem in detail.

3.5. Network Address Translation Traversal

Network address translation (NAT) is the solution to the limited internet protocol version 4 (IPv4) addresses. Instead of each machine on a network having an independent IP address, only the router has an IP address that is reachable by the whole internet. This is called the public IP address. The router assigns all the other devices in the network a local IP address, which is only valid in the local network. The router then uses a NAT table to store these local IP addresses and redirect the traffic from outside the network to its corresponding device. This might not seem like a problem, but one peer has to host the server, as explained earlier. The other peers need the IP

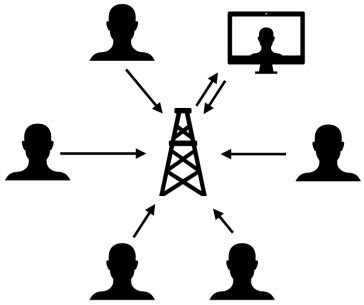


Figure 3.6.: The users connect to the host over a relay server.

address of this server to connect to it, but as the server is running on a device hidden behind NAT, it only has a local IP address and cannot be reached by the clients. Instead, the other peers will have to use the router's public IP address to connect to the server. However, by default, the router will not know what to do with the requests it receives, as it does not know which of the local devices the request targets. Outgoing connections work fine, so it is possible to connect to a server as a client, but incoming connections are declined when acting as a server. The procedure of overcoming this problem is called NAT traversal, and there are various ways to overcome this problem (Hu, 2005).

3.5.1. Internet Protocol Version 6

As mentioned above, the NAT was introduced because more devices are on the internet than available IPv4 addresses. With the latest internet protocol version 6 standard defined in Deering and Hinden (1998), which increased the size of an address, there are more than enough addresses available for each device, so there will be no need for NAT and therefore no need to implement NAT traversal. Nevertheless, it is uncertain when IPv4 will be deprecated, so we still need a solution for this problem until then.

3.5.2. Automatic Port Forwarding

The usual procedure to host a server hidden behind the NAT is to connect to the router through the browser and manually establish port forwarding. We tell the router that it should forward all accesses on a specific port to a specified device. If the firewall is set up correctly, other devices can then connect to the hosted server. However, users would have to manually set all things up, which, even with good documentation, can be challenging, especially if the users are not computer specialists. So a solution that works out of the box when starting the program is required. The program should set up the NAT traversal, and the users should have to do as little as possible for this. That is where automatic port forwarding comes into play. There are possibilities to automatically forward ports to a running program on a device.

Universal Plug and Play (UPnP). UPnP is a technology designed to make things as easy as possible for users. Microsoft initially introduced it, and it is now widely used. The key feature is that a device immediately works when plugged in, without any setup necessary. It is a convenient way to locate other devices on the same network and modify router settings remotely at the router-level. UPnP can also be used to establish automatic port forwarding. Users can request a temporal port forward from the router that stays active as long as in use. The only prerequisites are that both the router and the operating system need to have UPnP support, and both devices must have it enabled (Esnaashari et al., 2013; Hu, 2005). The big drawback of UPnP is that it is known to have security issues. While making it easy for devices to work out of the box, it also makes it easy for malware to exploit its weaknesses. Malicious programs can effortlessly request a port forward from the router, either on specific ports or all ports at once. Having all the computer's ports open to the whole internet gives an attacker various opportunities to do evil. While some experts advise disabling UPnP due to these vulnerabilities, others advise to keep the service active and use it with caution, as many devices depend on it (Garcia, 2011; Hemel, 2006).

NAT Port Mapping Protocol (NAT-PMP) & Port Control Protocol (PCP). NAT-PMP, defined by Chesire and Krochmal (2013), is Apple's answer to Microsoft's UPnP. It is simpler than UPnP, but it is used much less. In 2013 it was replaced by Apple's PCP. The protocol is fully compatible with NAT-PMP. While still much less in use than UPnP, Apple uses it in its newer NAT devices.

3.5.3. NAT Punch-through

A second option to achieve NAT traversal is NAT punch-through. It tricks the NAT by using a public server, the so-called facilitator. One problem of NAT is that the router only allows incoming messages from a specific IP address and port if this exact address-port combination has been messaged to before. This is why the router blocks incoming messages to the hosted server, and it is why NAT traversal is needed in the first place. In the NAT punching process, also referred to as hole punching, both peers message the facilitator first. The facilitator stores the IP addresses and port information and forwards them to the other respective peer. Both peers now have the port-address combination of the other peer. When sending messages to the other peer, the first message might get rejected by the router, but after that, the messages are no longer blocked. This works because the other peer has also sent a message and therefore allowed the communication (Hu, 2005; Thu et al., 2014). The session traversal utilities for NAT (STUN) protocol defined by Rosenberg et al. (2008) provides a standardized communication protocol. It should not be confused with its successor, the simple traversal of user datagram protocol through network address translators (also STUN), defined by Rosenberg et al. (2003). The old version was not only a standardized protocol but provided a complete solution for NAT traversal. Because of this old version, the phrases NAT punch-through and STUN are often used as synonyms.

3.5.4. Relay Server

As already stated, when hiding behind NAT, it still works fine to connect to a server outside of the NAT. Only hosting a server behind the NAT

causes problems. The idea behind a relay server is to run a separate server on a public IP address that allows connection for all peers. Section 3.4 already described the architecture. As the name suggests, the server relays all received messages. It forwards the messages coming from a client to the server and vice versa. The relay server is a straightforward solution for NAT traversal. The disadvantage is that a server with a public IP address is needed. All network traffic will be routed over this server. This leads to high latency, and the scalability is very limited. Another problem with the relay server is that the server does not know that it communicates with more than one client. For the server, it seems as if there is only one client, the relay server. The traversal using relays around NAT (TURN) protocol defined by Mahy et al. (2010) addresses this problem. It makes it possible for the server to communicate with different peers over the same relay. Often, TURN is used when NAT punch-through by STUN was not successful. The following section discusses the different network solutions available for Unity, some of which already come with integrated NAT traversal.

3.6. Multi-User Options for Unity

Unity's standard multi-user technology used to be UNet[1], which Unity provided by default. However, UNet was deprecated in 2018 to start all over with a new solution: Connected Games[2]. While this is a necessary step towards a well-functioning network technology in Unity, it leaves developers pending what to do in the transition time. Apart from the Unity out-of-the-box solutions, there are several other ways to tackle the problem of getting multi-user support into the project. This section compares the advantages and disadvantages of the various solutions to select one for this thesis.

3.6.1. UNet

As already pointed out, UNet was the old Unity standard for multi-user projects. UNet consists of different parts, most notably the High-Level

[1]https://docs.unity3d.com/Manual/UNet.html
[2]https://unity3d.com/partners/google/connectedgames

Scripting API (HLAPI), the Low-Level Scripting API (LLAPI), and the Relay Server and Matchmaker services. The LLAPI provides a basic Transport Layer setup that sets up a network and connects users. The HLAPI is built on the LLAPI and gives the user an easy starting point for building a P2P architecture. Additionally, a relay server can be added, as described in Section 3.5. UNet mainly focuses on P2P architectures, but Unity wants to shift to dedicated server architectures. Thus, Unity decided to deprecate UNet in 2018 and started over with Connected Games. According to House (2018a), Unity Technologies (2021), Unity will no longer ship with the HLAPI after 2018 but will support it for two more years. The same is applicable for the LLAPI, but from 2019 on.

3.6.2. Connected Games

In 2018 Unity announced Connected Games as the new standard for multi-user projects. Unity teamed up with Google Cloud Platform to create this new network technology. It should include all kinds of architectures in the distant future, but for now, it concentrates on dedicated server architectures. House (2018b) states this is due to this architecture's increased reach and scalability. Unity wants to first focus on the big flagship projects that should establish this new technology in well-known games. Thus, they moved away from the P2P architecture used by UNet. Just like UNet, the new system consists of two distinct parts:

Unity Transport Package. The Unity Transport Package[3] is the replacement for the UNet LLAPI. It establishes connections, sends messages between clients and servers, serializes the data, and takes care of all other underlying network operations.

Unity NetCode Package. The new network solution Unity NetCode[4] is not directly a replacement for the UNet HLAPI. While it still takes care of synchronizing the game, it does this in a completely different way. This is

[3]https://docs.unity3d.com/Packages/com.unity.transport@0.3/manual/index.html
[4]https://docs.unity3d.com/Packages/com.unity.netcode@0.0/manual/index.html

because the NetCode package is part of Unity's new data-oriented technology stack (DOTS). The DOTS project is part of a big refurbish plan for Unity. It focuses on bringing the advantages of multi-threading to Unity. The basis for DOTS is the Entity Component System (ECS), a new way of thinking in Unity. Instead of each game object having its scripts executed, the game objects (entities) only store the data (components). What to do with the data is determined by the systems (Unity Technologies, 2020). According to House (2019) the new DOTS NetCode will be released in early 2021.

Several disadvantages come with the new system. Rent has to be paid for a server on the Google Cloud Platform, and the documentation is not extensive by now. Unity's sample projects allow having a first insight, but there is no official documentation yet.

3.6.3. Mirror

Mirror[5] is the open-source successor of UNet, that ships under MIT license. It provides most of the functionalities known from UNet but addresses significant issues that UNet still included. In contrast to UNet, Mirror is still fully supported, and on their website, they assure that it also will be in the future. The fundamental components are already known from UNet, making this system easy to learn for developers who have worked with UNet before. It is easy to set up a network with a host and connecting clients. The included '*NetworkTransform*' component automatically synchronizes game objects over the connected parties.

3.6.4. Photon

Unlike Mirror, Photon[6] does not purely focus on Unity projects. It is a pay-to-use API that supports all kinds of network architectures. It offers various products for different network architectures. The Bolt and Quantum products are focused on dedicated server architectures. Photon PUN, on the

[5]https://mirror-networking.com/
[6]https://www.photonengine.com/

other hand, provides a P2P solution. The main advantage over Mirror is that it comes with integrated NAT traversal. Also, it is cross-platform out of the box. The Photon public cloud is used to run the projects, and a subscription plan has to be purchased to use it. It is free for up to 20 concurrent users (CCU), but for more users, the price ranges from 95$ one-time (100 CCU) up to 370$ per month (2000 CCU).

3.6.5. MLAPI

The Mid Level API[7] (MLAPI) is another open-source game networking stack that ships with an MIT license. It runs on top of UNet, and, as the name suggests, it is a network solution located somewhere between UNets LLAPI and HLAPI. It provides many simplifications over the LLAPI, such as keeping game objects in sync and handling underlying network setups. It provides much more control than the HLAPI. According to their website, it also provides more functionalities than Mirror does. However, MLAPI relies heavily on UNet, so it also suffers from the deprecation of UNet.

With the theoretical foundations laid, the following section presents the concrete design decisions.

3.7. Multi-User Design

This section compares the different options that were presented in the previous sections. We used the gathered information to make reasonable design decisions for Maroon.

3.7.1. Network Architecture Selection

Table 3.1 compares the different architecture properties that were presented in Section 3.4. As defined in the requirements in Section 3.3, the primary use case for multi-user in Maroon is to connect classes and their teachers.

[7]https://mlapi.network/

	Local	Direct P2P	Client-Server P2P	Dedicated Server
Latency	✓✓	x	x	✓
Scale	x	xx	x	✓✓
Cost	✓✓	✓✓	✓	xx
Complexity	✓✓	x	✓	xx
Reach	xx	x	✓	✓✓

Table 3.1.: Overview over the properties of the different multi-user architectures (Unity, 2018). The ranking in each category goes from xx - vital disadvantage, to ✓✓ - vital advantage.

To be usable by more than one class at once, different instances of Maroon have to run at the same time. In other words, it is not sufficient to have only one dedicated server for all users. It would be an option to have a dedicated server setup, where users launch a new instance on the server for each class. However, this would cause the need for a lot of server capacity, which limits the scalability. This means that more servers would be needed once the project and its online usage grow.

It is good to have a local network solution in addition to an online one. However, this thesis's primary requirement is to achieve online collaborative learning, so a purely local solution does not fit this objective.

For this thesis, we chose a P2P network, a client-server P2P architecture to be exact. The teacher, or any of the students, can host a server for the class, and all other class members can connect to this server. The low scalability of a P2P network is not a decisive factor, as it still works well for the number of people in one class. As all the instances of Maroon run separately, there is no limitation to the scalability overall. The only additional requirement is a list server that keeps track of all the available hosted servers (Section 4.1.2). However, a P2P architecture choice means that we have to overcome the network address translation.

3.7.2. Network Address Translation Traversal Comparison

Table 3.2 compares the advantages and disadvantages of the various NAT traversal options that were presented in Section 3.5. However, the choice

	UPnP	PCP	STUN	TURN
Success rate	x	xx	✓	✓✓
Latency	✓	✓	✓	x
Complexity	✓	✓	xx	x

Table 3.2.: Overview over various options of NAT Traversal. The ranking in each category goes from xx - vital disadvantage, to ✓✓ - vital advantage.

is not exclusively. All the presented methods can be combined to achieve better performance. A common approach is to try the methods sequentially. First, a direct connection is attempted to check if the server has a public IP address. If this fails, the server tries to set up automatic port forwarding, both by UPnP and PCP, as these are the easiest options. If this fails as well, the peers try NAT punch-through by STUN. This has a higher success rate, but dependent on the NAT, there is still a chance that it fails. If all other approaches failed, the system falls back to using a relay server applying TURN. While this is the worst option regarding latency and requires an external server, it is the only option with guaranteed success. This NAT traversal setup would also be desirable for Maroon, but it is not in this thesis's scope to implement all of these options by hand. Some network options for Unity already provide a solution for NAT traversal, but others do not. Therefore a reasoned choice for NAT traversal can only be made once a Unity network option is selected.

3.7.3. Selected Multi-User Option for Unity

This section presents which option we chose to implement a network solution for Maroon in Unity. The available options were presented in Section 3.6. Usually, this selection would be an easy one to make. With the project being open-source and having as few dependencies as possible, the logical choice would be to take an out-of-the-box solution that ships with Unity. With the particular state that Unity's networking code is currently in, this selection is a much harder one to make. UNet was the go-to solution for Unity so far, but now that it is deprecated, it will only be supported for a limited time. With a continually changing project such as Maroon, which is under constant development, it is not compatible with this thesis's objective to

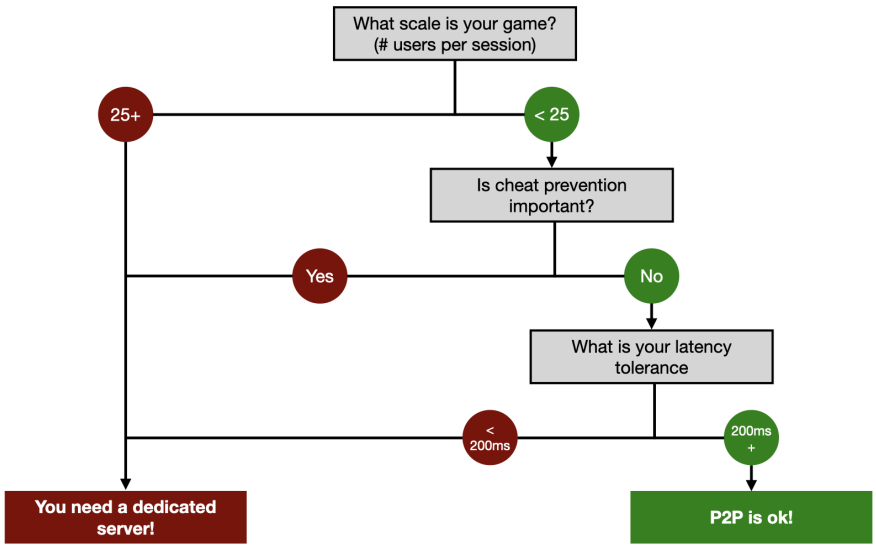


Figure 3.7.: The decision flow chart, redrawn from House (2019).

add a multi-user feature that has to be removed soon because it is not supported anymore. On the other hand, the new DOTS NetCode is not officially released yet, and therefore the documentation is hardly existent. This makes it very hard to work with, and as it is still under development, there is no guarantee that what works in the pre-released version will also work using the released version in 2021.

Unity is well-aware of the unpleasant situation that developers are in these days. Figure 3.7 shows in a flow chart how to handle the current situation as a developer from Unity's perspective. Following the path applicable to Maroon (marked in green) concludes that a P2P network is most feasible. Unity recommends using UNet, as long as it is still supported. However, due to the reasons stated above, this is not possible. Therefore other networking stacks, not provided by Unity, have to be taken into consideration.

Table 3.3 gives an overview of the features of the available technologies. Photon would be a suitable solution, as it is easy to use and already handles NAT traversal. However, it has to be purchased or subscribed, which violates the open-source approach of Maroon. MLAPI is also not an option, as it

	UNet	NetCode	Mirror	Photon	MLAPI
Future Support	x	✓	✓	✓	x
Documentation	✓	x	✓	✓	✓
Suitable License	✓	✓	✓	x	✓
Cost	✓	x	✓	x	✓
NAT Traversal	x	✓	x	✓	✓

Table 3.3.: Overview over network technologies for Unity.

relies on UNet. Therefore, the chosen networking API is Mirror. It offers the same functionalities as UNet, which according to Figure 3.7 is the option recommended by Unity. "*Mirror is the most compatible direct replacement for the deprecated Unity Networking API*" (Mirror Networking, 2020). In contrast to UNet, it will not be deprecated shortly. Moreover, Mirror ships with an MIT license, which allows us to use it in the Maroon project without any trouble. The 25 concurrent user limit in Figure 3.7 is calculated for UNet and does not apply to Mirror. The networking technology is so much more efficient than UNet that it can handle up to 500 concurrent users, which is more than enough for Maroon (Mirror Networking, 2020). However, the choice of Mirror means that there is no integrated solution for NAT traversal, so we have to find a custom solution.

3.7.4. Network Address Translation Traversal Selection

Mirror's website[8] recommends forwarding the ports manually as a solution for NAT Traversal. This is not a satisfying solution for Maroon, as we do not want users to have this effort. There are integrations for Mirror that provide NAT traversal, but none have a suitable license for Maroon. We considered using a relay server, but the limited scalability and the cost for a server made us reject the idea. With the relay server off the board, automatic port forwarding is the chosen solution. Research brought attention to Open.NAT[9], a C# library that supports port forwarding by both UPnP and PMP. It allows the user to discover a UPnP or PMP device in the current network, and if

[8]https://mirror-networking.com/list-server/
[9]https://github.com/lontivero/Open.NAT

it finds an enabled device, it sets up a custom port mapping. As the more recent PCP protocol is also compatible with PMP, PCP routers should work as well. The mapping is temporal and deleted after a specified time or when the application is closed. We tested the solution with a UPnP-enabled router, and it worked perfectly. As already discussed earlier, the disadvantage of automatic port forwarding is the low success rate. This is because not every router supports UPnP or PMP, and on many devices, it is disabled. However, according to a study carried out by Yang (2019), 76% of all routers on the internet have UPnP enabled. With multiple people trying to collaborate, this leaves only a minimal chance that none of the users can create a UPnP port mapping. Therefore it is feasible to rely on this method of NAT traversal for Maroon. The addition of a local network connection option allows the use in school or organization networks, where UPnP is usually disabled. It even works when the peers only connect over a smartphone that has an active WiFi hotspot. NAT punch-through would be a valuable addition for those cases where automatic port forwarding fails, so we leave it open as an extension for the future (Chapter 7). With all relevant multi-user design decisions made, the following section focuses on the sorting algorithm experiment's design.

3.8. Sorting Experiment Design

The sorting experiment should allow users to get to know different sorting algorithms. The visualization should be engaging and pleasing while still educational. As Naps et al. (2002) define in their eleven best practices for algorithm visualization, there are several properties that the visualization should fulfill. Users should control the visualization flow, so they can step through the visualization at their pace. Also, it should be possible to step backward for better traceability. A pseudo-code should be displayed to the user as technical background information. The current execution line should be highlighted in the pseudo-code to make it easier for the user to understand the execution order. Additionally, a short textual description of the algorithm should be displayed to explain the basic concept. As already described in Section 2.2, the setup of a Maroon experiment is predefined. There exists a template experiment that we can use for the creation of the

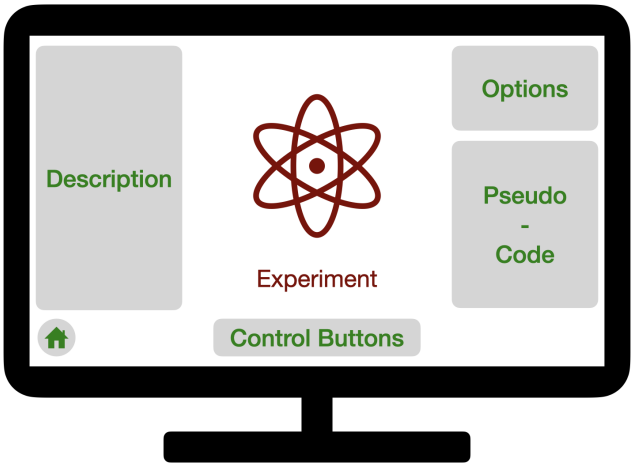


Figure 3.8.: A mock-up for the detail-view design.

new sorting experiment. Following the style of Maroon, we developed a detail-view mock-up, shown in Figure 3.8.

In addition to the detail-view covering all the aspects defined above, the experiment should contain a battle-view. In this view, users should let algorithms compete on different data sets. If these data sets are large enough, this allows a judgment of the algorithms' time complexity, which is usually hard to achieve with algorithm visualizations (Laxer, 2001). Moreover, the competition is entertaining to watch, as users can cheer along for their favorite. To amplify this, we introduce the sorting challenge. In this challenge, users can predict which algorithm will sort the field faster. They gain a point for each correct prediction. Especially in the multi-user version of the experiment, this can lead to positive competition. Moreover, having such a mini-game in a collaborative environment can lead to more active participation (Azmi et al., 2015). Figure 3.9 shows a mock-up of the battle-view.

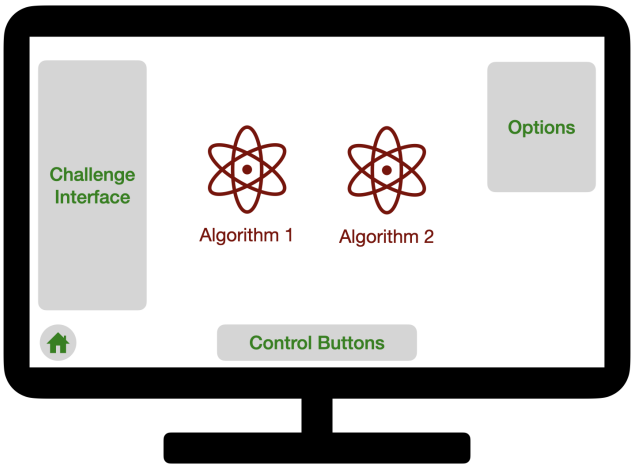


Figure 3.9.: A mock-up for the battle-view design.

3.9. Summary

This chapter defined the objectives and requirements for this thesis, based on the research in Chapter 2. It presented a client-server P2P network as the chosen architecture, as this architecture best suited the defined requirements. With the choice of this architecture, network address translation is a problem that we have to overcome. This chapter gave different solutions to the problem and introduced automatic port forwarding as the choice for this thesis. Moreover, this chapter deals with the current unfortunate situation of network development in Unity. As Unity is currently redesigning its network code, we chose Mirror as a substitute. Mirror provides all functionality that was provided by Unity before and more. The end of this chapter presents the design of the sorting experiment. It consists of a detail-view that teaches the various algorithms and a battle-view in which algorithms can compete. Additionally, the battle-view includes a challenge in which users can predict which algorithm will be faster. The next chapter deals with the implementation of all the defined functionalities.

4. Implementation

This chapter describes the implementation details based on the requirements and design defined in the previous chapter. It shows in detail how the multi-user network is established, which parts are required and how they interact. Moreover, it introduces the concept of control handling as a regulation tool to prevent chaos when having many users. Synchronizing experiments is introduced as an important topic, as it is a feature that future developers will have to implement for their experiments. Furthermore, this chapter presents the design of the sorting experiment and the associated sorting challenge.

4.1. Network Establishment

In Section 3.7.3, we chose Mirror to implement the network part. The crucial component in a network implementation with Mirror is the *Network Manager*. Mirror provides a base class that handles hosting a server, joining a server at a specified IP address, and all other kinds of basic functionalities. We derived from this base class to create a custom *Network Manager* for Maroon. We implemented this new class as a singleton that stays active across all of Maroon's scenes. It spawns a player object for all connected clients and is responsible for all messages sent between clients and the server. It handles the network setup process by calling the corresponding methods in the classes for automatic port forwarding, the list server, and the local network discovery. The following sections describe these classes and how the various parts act together to form the setup process.

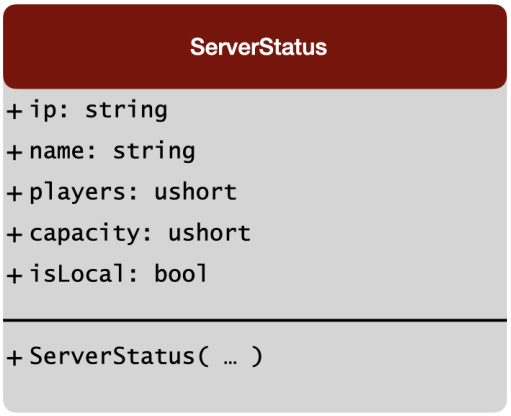


Figure 4.1.: Representation of the *ServerStatus* class.

4.1.1. Local Network Discovery

As defined in the requirements, Maroon should support online as well as local connections. Mirror already comes with an in-built network discovery base class. This class provides two essential options: (1) Advertising a server over the local network and (2) finding advertised servers. Deriving a class from this base class also allows the publication of custom server information. This allows passing information, such as the server name and the current players, over the local network. The *MaroonNetworkDiscovery* class extents the Mirror base class to advertise running servers on the network or search for active servers. To store the server information, we introduced the *ServerStatus* class, displayed in Figure 4.1. This class contains all information on servers published over the network, and one class instance is created for each server found.

4.1.2. List Server

With the client-server P2P architecture chosen, the need for a list server arises. Without it, the users would have to manually type in the host's IP address to connect to it. This requires additional effort to figure out the IP address and to distribute it to the other peers. Therefore, a list server is

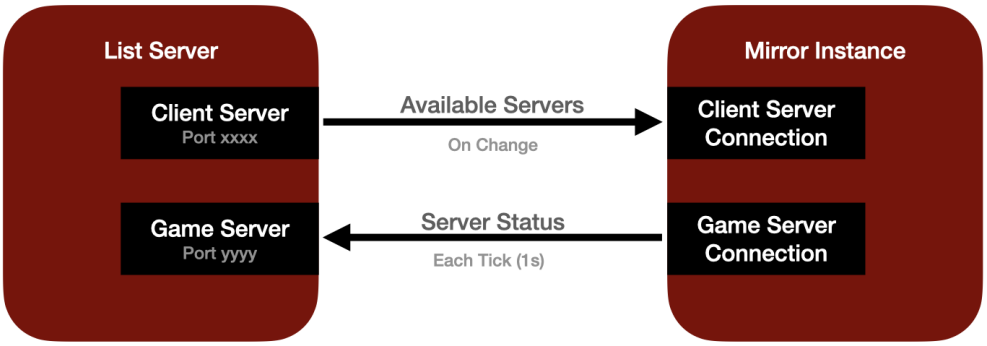


Figure 4.2.: Incoming and outgoing communication with the list server.

needed to keep track of which peer has hosted a server on which IP address. The list server then distributes this information to all clients connected to it, allowing users to join the requested server with one button click.

Since the Maroon team decided to use an independent server solution, we had to set up a custom list server. This meant that we had to write the server code ourselves. The custom implementation, written in NodeJS, allows running the server on all major hosting platforms. The client-side implementation of the list server ships under MIT license and provides a basis for the server-side implementation.

Figure 4.2 shows all the interactions of the existing implementation in Mirror and the list server. The list server consists of two separate parts, the client-server and the game server. Once the multi-user feature is enabled, each Mirror instance connects to the client-server. Over this server, the list server distributes all available matches to all connected clients. Once users decide to host a server, they disconnect from the client-server and connect to the game-server instead, which runs on a different port of the list server. Users running an active server provide their server information to the game server and update it continually. This allows the list server to have up-to-date information about all hosted servers continually, and it also keeps track of which servers are still running and which have quit service. We used all this gathered information to implement a custom version of the list server in

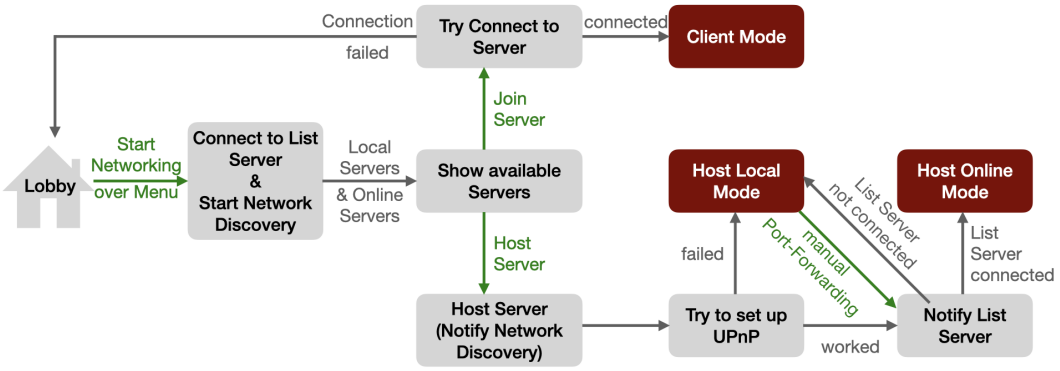


Figure 4.3.: Setting up the network. Green arrows are user interactions, grey arrows are executed automatically.

NodeJS. The server is fully compatible with Mirror's client-side code, which we customized for Maroon. All the servers received from the list server are stored in the *ServerStatus* class displayed in Figure 4.1. The only difference to above is that the *isLocal* attribute is set to false, so it is possible to distinguish between servers found over the local network and those received from the list server.

4.1.3. Network Setup

The network system should automatically be able to handle local and online servers at the same time. To make it as easy for the user as possible, we handle both options equally. It is even possible to have online users and local users connect to the same server. This subsection explains how the different parts act together to set up the network. Figure 4.3 shows the setup process. All states marked in red are the networked states in which the network is enabled.

The network part remains completely inactive before the main menu's network button is clicked for the first time. This ensures that the multi-user part of Maroon does not affect the offline version at all. Once the user has clicked on the network button, a connection to the list server is attempted, and the local network discovery is started. All the found servers are stored in

the same list displayed to the users over the menu, described in Section 4.2.2. If users select one of the available servers by clicking on it, a connection attempt is made. On success, they join the server as a client, which leaves no further actions to be made. Failing to connect brings users back to the offline state they started in, with the difference that the search for servers both over the list server and the local network remains active. An error message informs the user that a connection was not possible.

The more complex part comes into play when users decide to host a server. Once the host is started, it notifies the local network discovery of the new server. At the same time, it starts an attempt to automatically forward the necessary ports, as described in Section 3.7.4. If the attempt fails, the user proceeds to the local-only mode, in which clients can only join over the local network. In this case, the server will not send data to the list server and will not be displayed to online users. There is still the option to manually forward the ports by entering the router settings, which the network administrator can do. Over the menu, the user can confirm that they forwarded the ports to be recognized by the list server. If port forwarding is successful, however, the host sends the server information to the list server. Now, everything is set up for online users to join. If the host cannot establish a connection to the list server, it is once again in local-only mode, where clients can only join over the local network. Note that there are cases that the diagram in Figure 4.3 does not cover. It is, for example, possible to host a server without first listing the available servers. Moreover, there is no real difference between the *Host Local Mode* and the *Host Online Mode* in practice. The only difference is that the server is not reachable by online users. Therefore, it is possible to directly go from *Host Local Mode* to *Host Online Mode* if the ports are forwarded successfully. However, the connection to the list server is established only at a later point. The diagram shows the network setup concept, but it should not be regarded as an accurate flow chart of the implemented solution.

4.1.4. Connection and Disconnection Handling

While the previous section covered the network setup process itself, this section focuses on a client's behavior when connecting to a server. Moreover, it shows how to handle client disconnects, both planned and unplanned.

Password Protection. One desired feature, defined in Section 3.3, is to have optional password protection for a server. Students might be assigned homework in Maroon or want to work together on a private project in the future. In these cases, there must be a way to keep other users from joining the server uninvitedly. However, working with IT security is a critical topic. If something has to be secure for real, it must be implemented by relying on secure libraries. Passwords should never be accessible to anyone who did not enter them, and the communication channels should be encrypted. Overall, actual security is a complex goal to achieve. Apart from the password, Maroon does not have to exchange any sensitive data over the network. There is also no threat that any hacker might attack the Maroon network. Our solution only has to keep a user who entered the wrong password from joining the server. So it is sufficient in Maroon's case to knowingly waive IT security and send the password unencrypted to then directly compare it to the defined server password.

Once users want to host a server, they first have the option to set a server password using the password interface shown in Figure 4.4. Once they hit the OK button or the enter key, the server is started. When users select a server to connect to, they must also enter the password using the same interface. The password is then compared in the connection procedure.

Connection Procedure. Some parameters have to be checked when a client connects to a server. As explained above, a password has to be exchanged and confirmed, and due to reasons explained in Section 4.3, a client cannot join a server that is currently in an experiment. Therefore we developed the connection procedure presented in Figure 4.5. Once a client has connected to the server, it sends a connect message over the network that contains the password that the user entered via the interface earlier. When the server

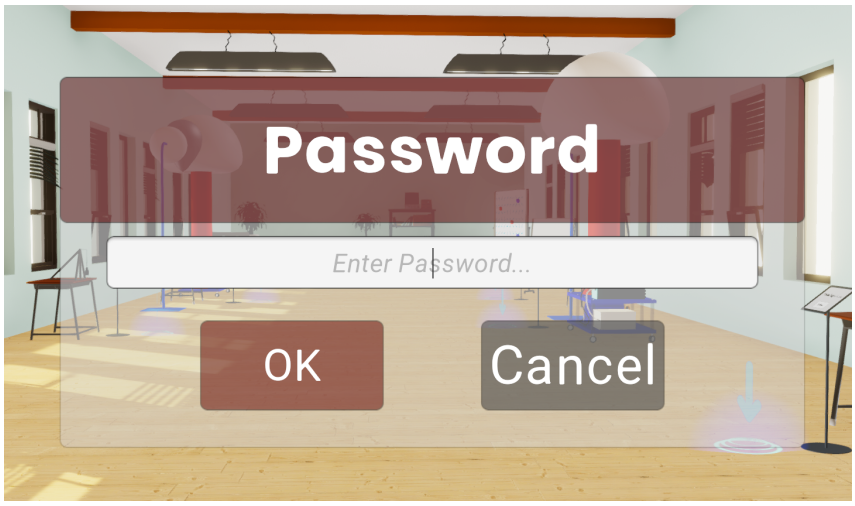


Figure 4.4.: When starting a host or connecting to a server a password can be entered.

receives this message, it checks if it is currently in an experiment and if the password that is received matches the password set on the server. In case it is in an experiment or the password was wrong, it sends a leave message back to the client that asks the client to leave the server. If both the password is correct and the server is currently in the lobby, the server adds the client to a list of authenticated clients. This allows spawning a player object later. Moreover a *CharacterSpawnMessage* is sent to the client. This message gives the client feedback that everything is fine and that it should send its character spawn information. When receiving this message, the client responds with a *CharacterSpawnMessage* itself, containing information about the position and rotation of the character in the laboratory. This allows the server to spawn the networked player object in the same position that the user was standing in when offline. This makes joining and leaving servers feel smoother, as the user does not randomly teleport through the laboratory. When the server receives this spawn information, it checks if the client is authenticated. If not, it immediately removes the client from the server. Otherwise, the server creates a player object for the new client and spawns it over the network. From this point onward, the client can participate in the collaborative learning environment.

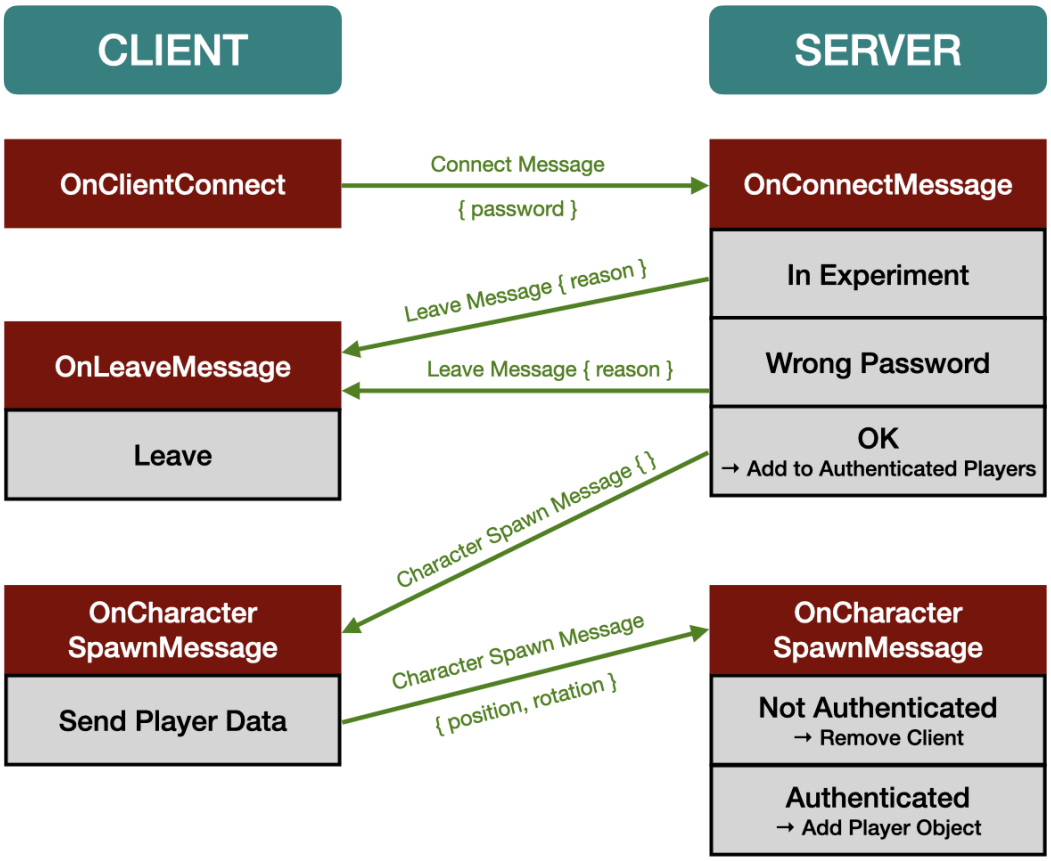


Figure 4.5.: The communication of client and server on connection. The green arrows are messages sent over the network.

State	Description
NoConnection	The server could not be reached when trying to connect.
Disconnect	The connection to the server was unexpectedly interrupted.
InExperiment	Could not connect to the server because it is in an experiment.
WrongPW	Could not connect to the server because the provided password was wrong.
Kicked	The user was kicked from the server by the host.
Usual	The client was actively stopped by the user over the menu.

Table 4.1.: The different disconnection states.

Disconnection Handling. When a client disconnects from the server, it is necessary to evaluate why this happened. To do this, we introduced the *DisconnectionState*. This state is stored on the client and can take the values shown in Table 4.1. When the client stops, the disconnection state can evaluate the reason, and a message can be displayed to the user. The disconnection state has to be kept up to date on the client. For this sake, we introduced the leave message mentioned above. The message contains a *DisconnectionState*. Once the client receives the message, it leaves the server and displays the associated message. If the client ignores the message, the server forces a disconnection to prevent malicious clients from exploiting the architecture. If a client disconnects from the server without receiving a leave message, we know that it was not a planned disconnection. Again, the corresponding message is shown to the users. Figure 4.6 shows the connection procedure diagram from above, extended by the disconnection states, displayed in orange.

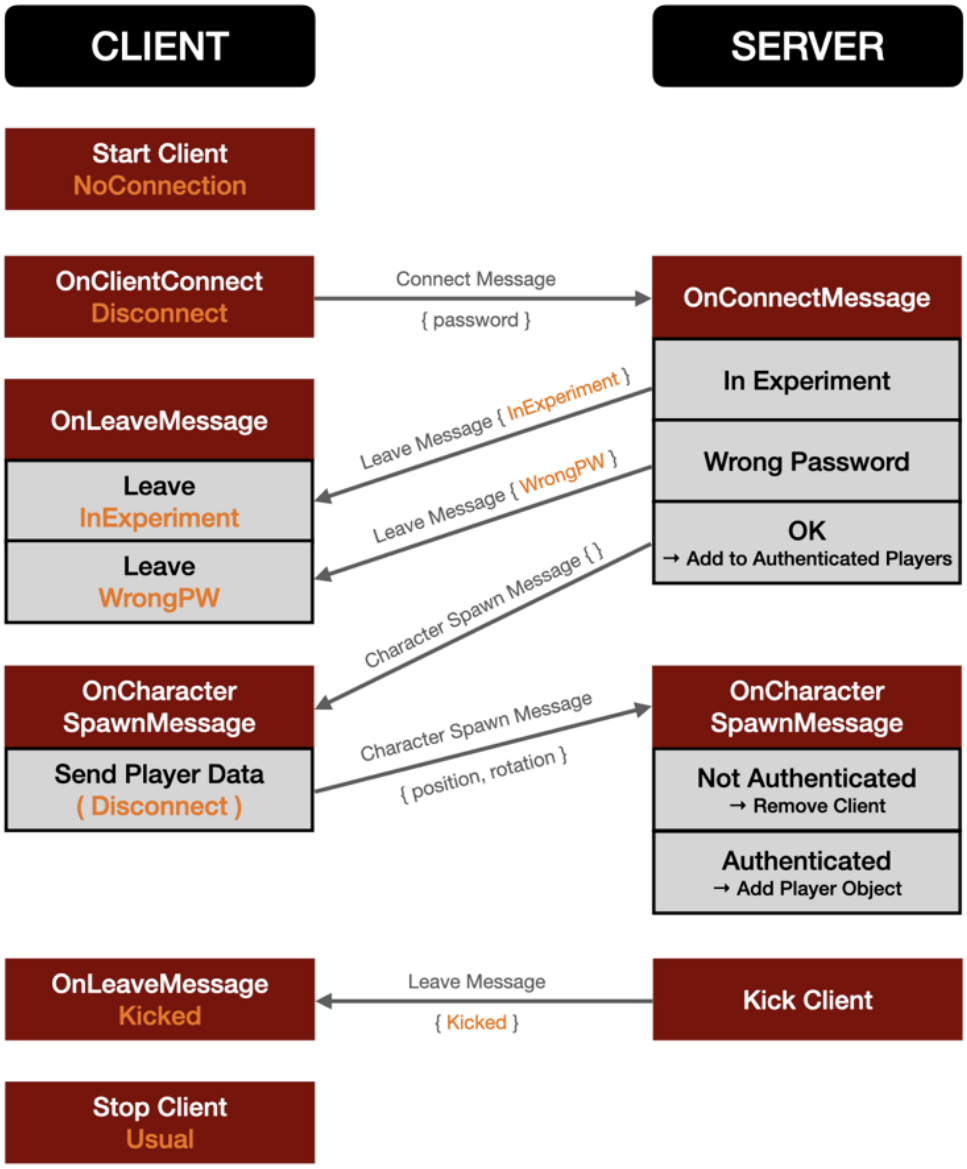


Figure 4.6.: The connection procedure diagram was extended by the disconnection states displayed in orange.

4.1.5. Naming

There are two components in our network that require choosing a name: users and servers. Allowing users to choose names freely allows them to choose offensive names. Therefore, we decided to assign all names automatically. As Maroon comes with support for different languages, the names have to be language-independent. For the user names, we chose some of the most famous scientists to match the theme of the laboratory (Stewart, 2020). As the selection is very male-heavy, we extended it by some of the most famous women in science (Curry, 2017). The name is assigned to a user by the server once connected. For the server names, we settled on the top-ranked universities' names for natural sciences (Quacquarelli Symonds, 2020). The name is chosen at random when starting the server, disregarding all names already taken by other servers on the list server or in the local network.

4.1.6. Ports

Several network parts in Maroon require a port. The most crucial one is the host port used for the connection of all the clients. The list server needs two ports, one for the client-server and one for the game-server. Moreover, the local network discovery needs a port to broadcast and receive server information over the local network. So in total, we need four ports, which have to be unique to avoid problems.

Ports can not just be selected randomly. A wide range of ports is already reserved for certified usage. The Internet Assigned Numbers Authority[1] (IANA) provides a register of all the taken ports. Cotton et al. (2011) describes how the port ranges are defined and which ports are open for public use. The ports 0-1,023 are the standardized ports or well-known ports. They are heavily occupied and should not be used for custom programs. The ports 1,024-49,151 are the user ports or registered ports. Together with the standardized ports, they are assigned by IANA, so the register gives information if these ports are open for use. The ports 49,152-65,535 are the

[1] https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

Service	Port Number
Host	13,118
List Server - Game Server	13,119
List Server - Client Server	13,120
Network Discovery	13,121

Table 4.2.: The port numbers chosen for Maroon.

Dynamic Ports or Private Ports. They are not assigned by IANA and can be assigned freely for private use.

For Maroon, a free port in the high (>10,000), mostly unoccupied, ports is the best choice. This leaves the option to register the port at IANA if wanted but still has a low risk of interfering with other programs. To choose a random port, we took the alphabetical position of the first three letters in Maroon: M - 13, A - 1, R - 18, resulting in port 13,118. Checking the IANA registry confirms that ports 12,866-13,159 are unassigned and free for use. Table 4.2 shows our final port assignments.

This section explained the implementation details of the network, its setup process, and the disconnection process. The following section deals with user interactions while users are on the server.

4.2. User Interaction

User interaction can be interpreted in different ways. On the one hand, it can mean the interaction of users that are on the same server. On the other hand, it can also mean how users interact with the program while on the server. Users' interaction with other users is a topic that we do not address in detail with our implementation, as it is not a pushing matter. Of course, interaction is a crucial topic for a collaborative learning environment, and as explained in Section 2.1.2, we favor synchronous communication between users. However, users nowadays are usually already connected over some voice chat. It is therefore not feasible to implement a custom communication solution. Instead, we leave it open for future work to add a voice chat component if desired (Chapter 7). This section mainly deals

with the interaction of users with the program. Moreover, it introduces the concept of control handling, which brings order to an otherwise chaotic system.

4.2.1. Control Handling

As defined in Section 3.3, it should be possible to work on a server for only a small group of students and for the whole class at once. If anyone can control anything, this introduces many problems. Two users could change something simultaneously, but we cannot tell which information arrives on the server faster due to network latency. Moreover, it can be unpleasant to users if another user immediately undoes something they just did. Especially in big groups, if many people interact, things change continuously without leaving users the chance to explore. Thus, we decided that only one user should control all actions synchronized over the network. It is still possible to have local user interactions that users can perform independently of other users. However, only one user should control essential operations like entering an experiment or changing the experiment parameters.

This section deals with the handling of who is in control. Section 4.3 deals with how the information of the user in control is synchronized over the network. For control handling, we implemented and compared two different approaches.

Approach 1: Granting Control. In this approach, the host starts as the user in control when starting a server. Once clients have joined, the host can grant them control over the menu. In the network column of the menu, a list of all connected clients can be seen, and clicking on this client hands over the control, as shown in Figure 4.7. The host always has the option to take back control over the menu actively. This prevents that an unresponsive user blocks other users from using the multi-user feature. It is also a security feature to take control of someone who purposefully misbehaves. If the user in control disconnects, the host automatically regains control.

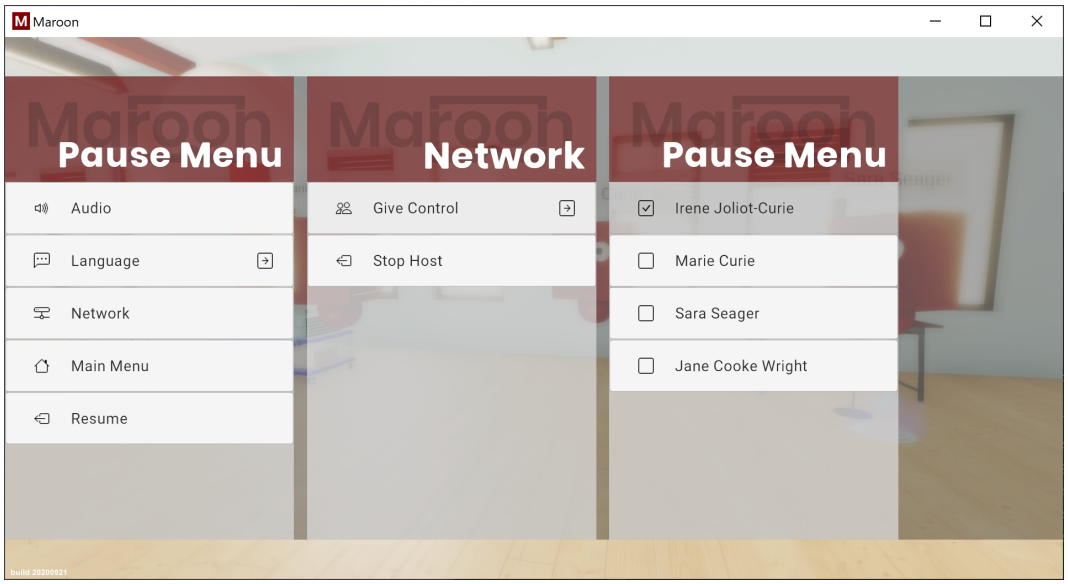
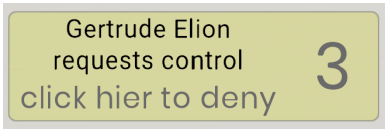


Figure 4.7.: Grant control to a client over the menu.

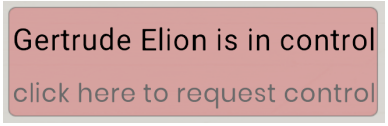
Approach 2: Taking Control. Once again, the host starts as the client in control. A box in the lower-left corner displays the current control information. When hovering over the box with the cursor, the box expands and displays the user's name and the server's name, as shown in Figure 4.8. For users that join, the box displays the name of the user in control. Clicking on the box sends a request to the user in control. The control request is displayed in the same box, together with a five-second countdown. If the control request is not declined in this period, control is granted to the user who requested it. Users can also cancel their requests by clicking on the box again. Canceling a request, either by users themselves or by the user in control, triggers a 15-second countdown in which they cannot request control. This keeps users from repeatedly spamming the request. Only one request can be made at a time, so all the connected clients have to be notified. The control user interface is both displayed in the lobby of the laboratory and inside each experiment. This means that it is always possible to change the user in control. Figure 4.9 displays all the states of the control user interface, and Figure 4.10 shows a flow chart diagram of how the control user interface states are reached. The colors in the diagram match the colors of the control user interface in the corresponding state.

Figure 4.8.: The control handling user interface is displayed in the lower left corner, here inside the Huygens' Principle experiment (Holly, 2019). The player name and server name are only displayed while the mouse hovers over the interface.

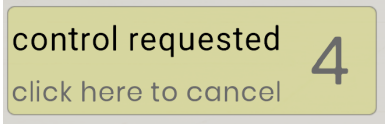
(a) The user is in control.



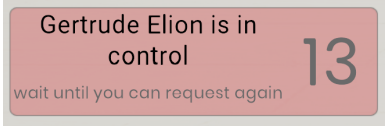
(b) Another user has requested control from you.



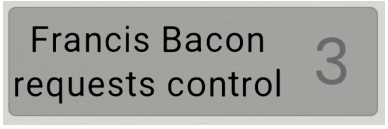
(c) Another user is in control.



(d) You have requested control



(e) Your last request has been canceled.



(f) Another user has requested control.

Figure 4.9.: All the states of the control handling user interface.

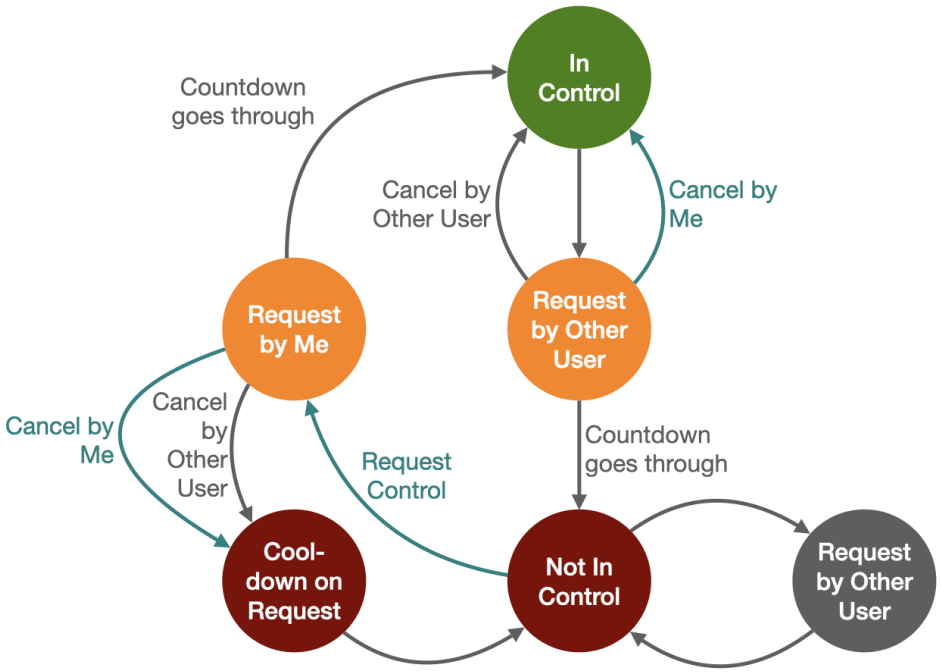


Figure 4.10.: Flow chart diagram for the control UI states. A colored arrow indicates an active input from the user.

	Approach 1 Grant Control	Approach 2 Take Control
Robustness	x	✓
Usability	x	✓
Inactive Users	x	✓
Troll Protection	✓	x

Table 4.3.: The two control handling approaches in direct comparison.

Comparison. The approach with granting control has the advantage that once users are in control, they can only actively lose this control and not bother with any requests. When looking at the use case where a teacher presents an experiment to a class, this is ideal. On the other hand, with the control-request approach, users in control might have to deal with numerous requests that they have to deny, which might be bothersome. However, this method's advantage is handling players who experience network problems or are inactive. In the control-granting approach, only the host can take control from them, but if the host itself is not actively participating, this leaves no user with the option to take control. Especially if the host might only run the server in the background for others to participate, this is a problem. With the control-taking approach, on the other hand, this is no problem at all. Users can leave their computers, even if they are in control, because other users can take control from them after a short delay. This option is also much more error-prone, which is crucial for network architecture.

Table 4.3 compares the two approaches. Overall, the second approach's advantages are more decisive, so we chose it for the final version. With the chosen solution, control visualization is already handled. A glance in the lower-left corner immediately tells the user who is currently in control. However, in the lobby, where the users can move around, there might still be confusion. For clarification, all user characters are displayed in red, apart from the user in control, who has a green character. This allows the other users to see what this user is doing or which experiment is about to start.

However, there is still a problem we have to address: a client that disturbs the experiments on purpose. First of all, this person should not be allowed to join the server in the first place. As described in Section 4.1.4, this can be

achieved using password protection. However, if unwanted users join the server, there has to be a way to deal with them. This would not have been a problem in the first approach, as the control can be taken from them by the host, and then they do not have an option to retake control. In the second approach, however, we have to find a solution to deal with the troublemaker. We solved this problem by adding an option for the host in the menu's network section. We introduced a *Kick User* button, which shows a list of all currently connected users. Clicking on a name removes the corresponding user from the server. The following section introduces the network menu, including this new button.

4.2.2. Menu

Maroon already provides a pause menu that lets users control language and sound settings. We decided to utilize this menu for the network part as well. Activities such as hosting a server or joining a server can be performed over this menu. The menu relies on a column design, so clicking a category button in the first column opens a second column with the corresponding category. Figure 4.11 shows the new network category of the menu.

The displayed buttons in the network column depend on the state of the network. When the user is offline, the menu contains the two buttons shown in Figure 4.11. The *Host Server* button, as the name suggests, starts a server. The *Join Server* button opens a third column, which lists all available servers. The local servers are listed first in alphabetical order, with the house icon marking them as local. The online servers are displayed in alphabetical order below the local servers, but instead of the house icon, the button shows the ping to the server. For all servers, the button contains the server's name, the number of currently active users, and the maximum number of users.

Once users have started a host or connected to a server, there are new options available. If they are connected to a server, the only button displayed is the *Leave Server* button, leaving the server and returning to offline mode. When starting a host, the user can instead stop the host with the *Stop Host* button.

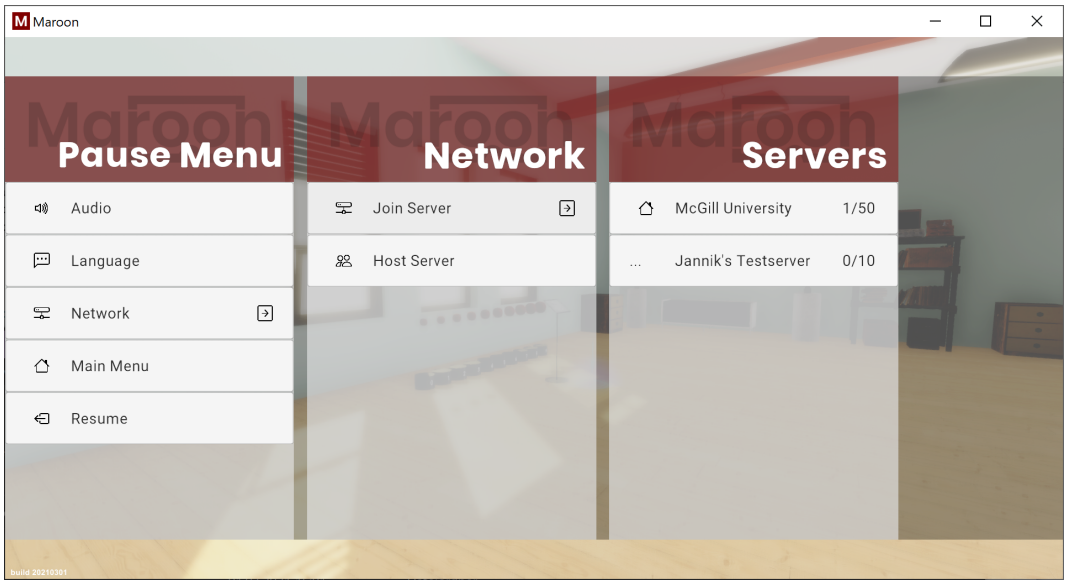


Figure 4.11.: The newly added network menu.

Additionally, the host can remove any user but itself from the server, using the *Kick Player* button introduced in the previous section.

The last button displayed is the *Ports Mapped Manually* button. As explained in Section 4.1, this is only displayed if the automatic port forwarding on a host failed. It allows declaring that the ports have been forwarded manually so online users can join the server. In addition to the appearance of the button, the user also receives an error message. The following section deals with this message and other ones and how they are displayed to the user.

4.2.3. Messages

A network solution needs to keep the user up to date with what is happening. Therefore some way is needed to display information to the user. Maroon already has a help character that presents messages to the user. According to Baylor (2009), such a virtual agent increases the motivation, so we decided to use the character to transmit our messages. Different kinds of messages can occur while running the network.

Connection State	Client Active	Server Active	List Server Connection	Ports Forwarded
Offline	x	x	✓	-
Offline No Connection To List Server	x	x	x	-
Client Online	✓	x	-	-
Host Online	x	✓	✓	✓
Host Online No Connection To List Server	x	✓	x	✓
Host Online Ports Not Mapped	x	✓	-	x

Table 4.4.: All possible connection states for Maroon. The - indicates that the state does not depend on that parameter.

Network Status Messages. As already covered in Section 4.1, the setup for the network is a complex process, so the need for user information is high. The user should be updated whenever the network status changes. This status mainly depends on the following parameters:

- Is the network client active (client only mode)?
- Is the network server active (host mode)?
- Is there a connection to the list server?
- Was the automatic port forwarding successful?

We defined different connection states depending on the combination of these parameters. Table 4.4 shows the defined states. Each second, the state is evaluated and stored in the *NetworkManager* class. Each time the connection status changes, a message is displayed to the user by the help character. When a user fails to start a host because a different server runs on that socket, a message is displayed. This is not a connection state message, but definitely, the user has to be informed about it.

Besides the messages displayed by the virtual agent, we introduced the network status station. It is a screen that the user can walk to in the virtual laboratory (Figure 4.12). It displays messages about the current network status and, in some cases, information on how to fix an issue. There is also

Figure 4.12.: The Network Status Station: The screen displays the network status, the red light indicates that there is a problem.

a light bulb attached to the station. The light indicates if there is a problem of interest to the user. If everything is fine, the light is green. If there is no connection to the list server or the port mapping failed, the light is red or yellow. If the user is offline, the light is switched off, so it is a good indicator for users in the scene to know if they are currently online.

Connection Messages. The second main category of messages consists of the connection messages. Users should know when they have successfully connected to a server and, even more critically, when disconnected from a server. The host is additionally informed each time a client joins or leaves the server. Once clients connect to a server, they are greeted by a welcome message. This message includes the assigned name of the user and the server's name. This message is also displayed to a host when starting the server initially. The disconnection messages depend on the disconnection state, defined in Section 4.1.4, and they display why the connection to the server stopped. All the messages are once again displayed to the user by the help character.

4.2.4. Scene Handling

As defined in Section 3.7.3, we use Mirror to implement multi-user. The name of Mirror reflects the strategy pursued, as the server mirrors scenes to the clients. This means that the server and the client always have to be in the same scene. Thus, users must enter experiments simultaneously, and they cannot work on distinct experiments on the same server. We ensure this by assigning the task of scene management to the network manager. As Maroon was not originally designed for multi-user scenarios, scene management was handled by different classes at the same time. This is the most fundamental change that had to be made to the existing framework, as now the network manager has to take over these functionalities.

Therefore, we created a new method in the *NetworkManager* that takes care of the scene management and is a direct replacement for the *LoadScene* method to be called before. Listing 4.1 shows the code of this new *EnterScene* method. It takes the scene name as a parameter, just as the scene management's *LoadScene* method does. If the user is offline, the method forwards the request to the scene management, so the offline behavior does not change. If the user is online, we perform a couple of checks. Firstly, we check if the user tries to enter the main menu scene. As already mentioned, all connected clients have to be in the same scene, so it is impossible to switch to the main menu scene while on a server. Therefore, we deny this with an error message. Secondly, we check if the request was made by the client in control. As described in Section 4.2.1, only one player at a time can be in control, which includes the right of changing scenes. If the client is not in control, the scene change is denied with an error message. The third check that we perform verifies if the requested scene is valid. This validity check was introduced to ensure that only those experiments that support multi-user can be entered in multi-user mode. To implement this, we added a list of scenes to the network manager class that can be accessed over Unity's editor. Once an experiment has been set up for multi-user, the developer should add the experiment scene to this list. Only if the scene is in the list, the validity check succeeds. If the check fails, an error message is displayed once again, and the scene change is denied. If none of the three checks fails, a network message is sent to the server containing the requested scene's name.

When the server receives the message, it first checks that the client that made the request is the client in control. Then it double-checks that the requested scene is valid. This prevents that a client requests a scene that is not valid on the server, which can happen if the client and the server are not running the same version. The server spawns a countdown screen over the network that is visible to all clients. This five-second countdown is a warning to every client that the scene is about to change. It allows everyone to finish what they are doing before entering the next scene. Moreover, the client in control has the opportunity to cancel the countdown in case any client is not ready yet. Once the countdown reaches zero, the server initiates a scene change that is automatically mirrored to all other clients.

As mentioned above, developers have to customize an experiment to add multi-user functionality manually. The following section covers how the developer can add this functionality and how we made it as straightforward as possible.

4.3. Synchronizing Experiments

Since Maroon's existing experiments should be equipped with multi-user support, we had to find an effective way to enable collaboration without redesigning the experiments. The problem is that these experiments were designed for single-user interactions. We found the solution to mirror all the actions taken by the controlling client to all other clients. This limits the interaction possibilities of those clients that are not in control. Nevertheless, it allows collaboration in experiments that were not designed for this purpose. When implementing the solution, we must be careful not to interfere with the functionalities of the experiment. Therefore we tried to change the existing implementation as little as possible while still reaching full functionality.

From an implementation perspective, the critical question is which data we have to send over the network. For simply mirroring the controlling client's experiment, it would be possible to share the whole screen. However, this has no advantage over sharing the screen over an external platform. It takes away the opportunity to specifically design an experiment for multi-user,

```
public void EnterScene(string sceneName)
{
    if (offline)
    {
        SceneManager.LoadScene(sceneName);
    }
    else
    {
        if (sceneName.Contains("Menu"))
        {
            DisplayError("Main Menu Denial");
            return;
        }
        if (IsInControl)
        {
            if (CheckSceneValid(sceneName))
            {
                SendMessageToServer(sceneName);
            }
            else
            {
                DisplayError("Experiment Not Enabled");
            }
        }
        else
        {
            DisplayError("Control Denial");
        }
    }
}
```

Listing 4.1: The *EnterScene* method of the *NetworkManager*, that is called by clients.

where each user can influence an experiment individually. Moreover, there would be immense amounts of data sent over the network. So we decided to implement a more elegant way that utilizes the fact that each instance of Maroon has all the necessary information to compute an experiment. The program already repeatedly evaluates the user input and recalculates the experiment state. So in order to synchronize the experiment, it is enough to synchronize the user input.

4.3.1. Synchronizing User Input

User input strongly varies from experiment to experiment, and our solution should not synchronize the user input of a specific experiment but for all of them. It should also be easy for future developers to synchronize user inputs to enable multi-user for their experiments. Therefore, we implemented a base class that provides basic functionalities. Developers only have to derive from this class and adapt it for their experiment. Figure 4.13 shows a class diagram for the new *ExperimentNetworkSync* class. It synchronizes the starting and stopping of experiments by adding a listener to the simulation controller's associated events. The virtual functions *OnGetControl* and *OnLoseControl* are called when a client gains control or loses it. Developers can use these functions to enable and disable user interactions. The class also provides the functions *AddListeners* and *RemoveListeners*, and automatically calls them. Listeners are a key part of synchronizing the user input. Each user interface element has some event attached to it, such as an *onClick*[2] event for buttons or an *onValueChanged*[3] event for sliders. We can add listeners to these events that do not change the rest of the experiment's behavior. They listen if there is any user input and allow sending this user input to the other clients.

The way that Mirror is designed, clients can only communicate with the server and not directly with other clients. Therefore all the information has to be sent to the host first. As each client can be the client in control, a method was needed to distribute the user input from any client to all other

[2]https://docs.unity3d.com/530/Documentation/ScriptReference/UI.Button-onClick.html

[3]https://docs.unity3d.com/540/Documentation/ScriptReference/UI.Slider-onValueChanged.html

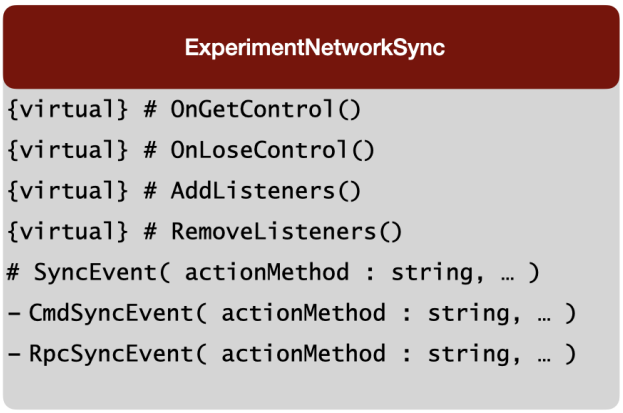


Figure 4.13.: Representation of the *ExperimentNetworkSync* class.

clients. Using the functionalities that Mirror provides, we achieved this by designing the architecture shown in Figure 4.14. Clients receive the user input, check if they are in control, and send the user input to the server using a command message. The server then distributes this message to all connected clients via a client remote procedure call (client RPC), including the client that initially sent the message. This client can then ignore the message as the action as it already acted. All other clients who receive the message perform the action connected to the event that has been triggered. This action has to be defined by the developer. It usually includes invoking the event that was triggered on the client in control.

It generates much code to write a listener, a command, a client RPC, and an action method for each possible user interaction. Therefore the base class provides the method *SyncEvent* that already implements most of this functionality. It takes the name of the action method as a parameter. Developers have to create a listener on the event that they want to observe, and inside this listener, they call the *SyncEvent* method, passing the name of the action method. For each event, they have to implement this action method, which has to have the return type *IEnumerator*[4] so it can be started as a subroutine. The *SyncEvent* method implements all logic needed for a client to be in sync with the client in control. Developers should add listeners to the overwritten *AddListeners* method and remove them in the

[4]https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html

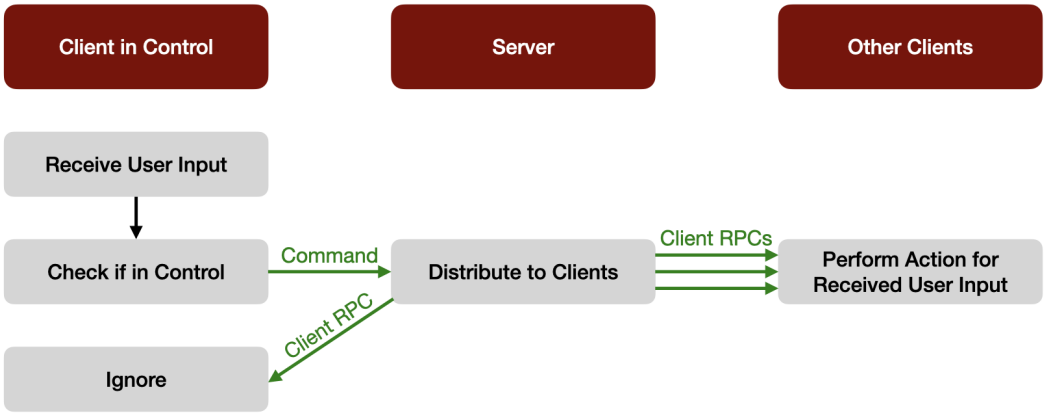


Figure 4.14.: Synchronizing a user input over the network using Mirror. All green arrows represent network messages.

RemoveListeners method to prevent problems when leaving the server while still being inside an experiment.

The *SyncEvent* method, as described above, works fine for all events that do not carry any additional information. However, most of the events have some information attached to them. The *onValueChanged* event of a slider, for example, carries the updated value as a float data type, and the *onEndEdit* event of an input field carries the input text as a string. To allow easy synchronization of these methods, we implemented various versions of the *SyncEvent* method, taking various input parameters. This method is currently available for the data types float, integer, string, and bool, but it is straightforward to add more variations. The corresponding action method also has to take the required data type as a parameter.

Unfortunately, not all user inputs trigger an event that we can capture using a listener. In these cases, developers must find a custom solution and implement synchronization themselves, using Mirror's network tools. When implementing the synchronization for the existing experiments, described in the following section, we came up with several custom solutions to be used as a guideline for other developers. These methods mainly rely on the *SyncVar* attribute provided by Mirror that automatically synchronizes a variable on all clients.

4.3.2. Adding Multi-User to Existing Experiments

The current framework consists of eight experiments. Table 4.5 shows a list of these experiments and indicates if we added multi-user support. We did not add support to the Coulomb's Law experiment, as it is still under constant development. We implemented a solution for the other seven experiments and added them to the list of multi-user enabled experiments, introduced in Section 4.2.4.

We created a guideline for future developers that they can follow to add multi-user support to their experiments. This guideline is published on the wiki of Maroon[5]. It presents the steps that developers have to take:

1. Derive from the *ExperimentNetworkSync* base class and implement
   a) all user input fields as serialized fields.
   b) the corresponding listeners using the provided *SyncEvent* methods.
   c) the action methods triggered by the listeners.
   d) all custom synchronizations.
   e) the control handling methods, overwriting the provided methods.
2. Place the script on an empty game object in the scene and link all the input fields.
3. Add the *network.xml* language file to the experiment's language manager, so all the network-specific texts can be displayed.
4. Add the scene to the enabled experiments in the *NetworkManager*. This allows users to enter the scene while in multi-user mode.

We followed this guideline when adding multi-user support to the existing experiments so that these experiments can serve as an example for future developers. Figure 4.15 shows four instances of Maroon, connected over the network, in the Faraday's Law experiment. Only one user is in control and therefore has the user interface enabled. The following section describes how we designed the sorting algorithm experiment and how we used this section's guideline to add multi-user support.

[5]https://github.com/GameLabGraz/Maroon/wiki

Experiment	Multi-User Support	Description
Falling Coil & Faraday's Law	✓	Two experiments focused on the current that is induced in a coil moving inside a magnetic field or the other way around (Holly, 2019).
Huygen's Principle	✓	This experiment focuses on waves and how these behave when hitting different numbers of slits in a wall (Holly, 2019).
Van de Graaf	✓	Two experiments focused on van de Graaf generators that create charges. In one experiment these are discharged to a balloon, in the other one they are discharged to a grounder (Pirker et al., 2017).
Pendulum	✓	Users can swing a pendulum of adaptable weight and rope length. Using an integrated stopwatch they can make observations of the frequency (Wolf, 2019).
Coulomb's Law	x	In this experiment the interaction of charged objects is visualized both in a two-dimensional and a three-dimensional setting (Brettschuh, 2019).
Whiteboard	✓	This scene is not an experiment but instead provides background information on the other experiments (Pirker et al., 2017).

Table 4.5.: A list of all the experiments in Maroon, showing if we implemented multi-user support.

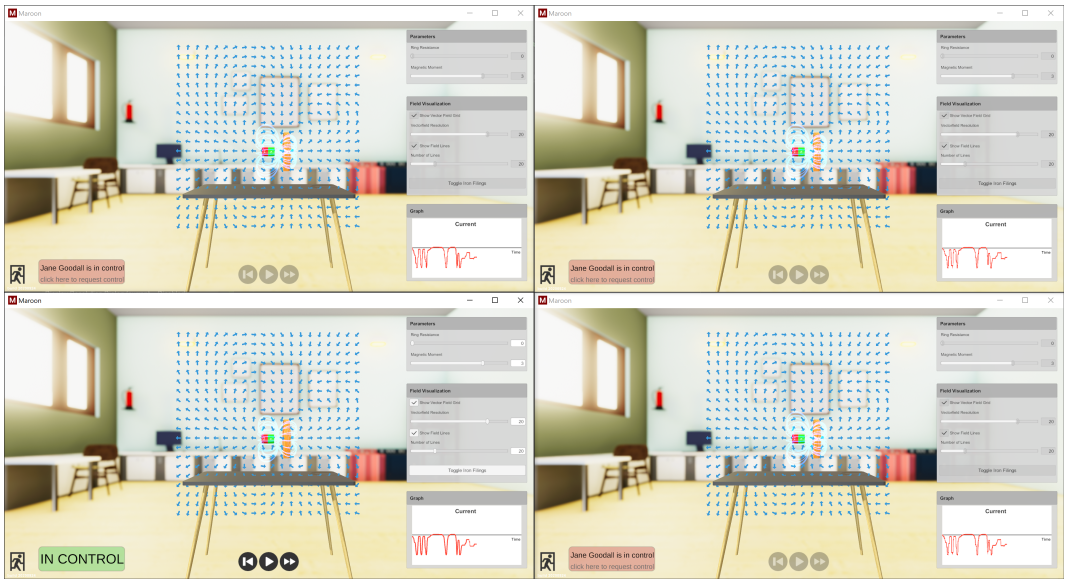


Figure 4.15.: Four instances of Maroon running the Faraday's Law experiment, but only one user is in control. The experiment was originally designed by Holly (2019).

4.4. Sorting Experiment Implementation

In Section 3.8 we designed the sorting experiment to consist of two views. The detail-view lets users step through algorithms forward and backward and provides additional information, such as a description and pseudo-code. The battle-view allows comparing two algorithms on a larger input field directly. It contains the sorting challenge, which should engage users. Both views support all nine algorithms that we described in Section 2.3.2. We implemented a *SortingController* class responsible for changing the view by enabling and disabling the corresponding components. This section describes the implementation of the different views in more detail.

4.4.1. Detail-View

The detail-view allows users to examine algorithms in detail. Figure 4.16 shows a screenshot of the user interface that we created based on the design

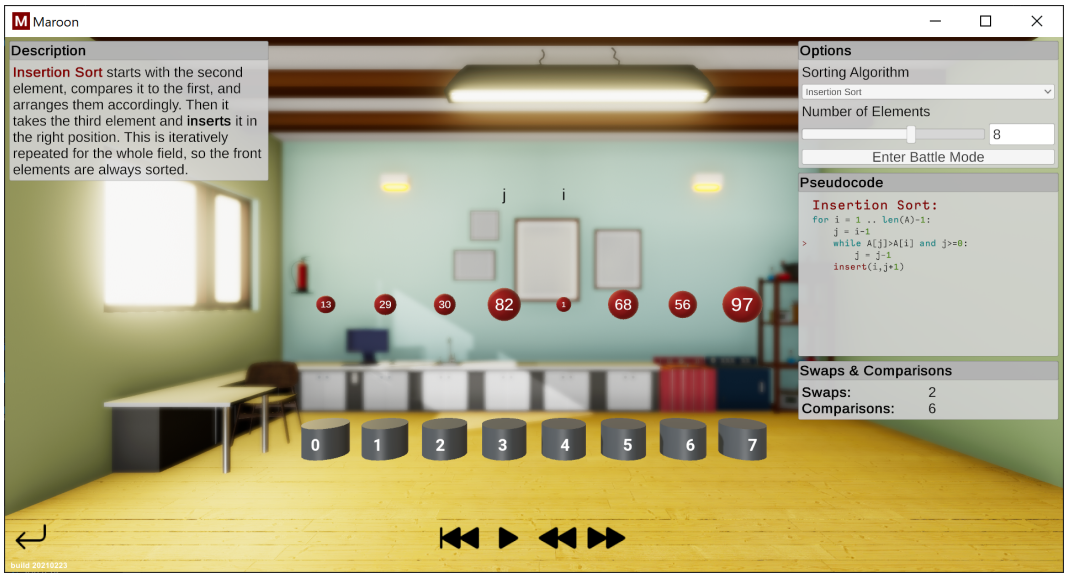


Figure 4.16.: The user interface of the detail-view.

defined in Section 3.8. The description on the left side gives a short overview of the idea behind the algorithm. In the operations panel on the right, users can select the algorithm that they want to examine. They can set the number of elements in the sorting field up to a maximum of ten elements. The *Enter Battle Mode* button switches to the battle-view. Below the options, the highlighted pseudo-code is placed. An arrow indicates the current execution line. Below that, the program counts the swaps and comparisons that the algorithm needs to sort the field. This allows users to keep track of what already happened and to compare different algorithms. The buttons at the bottom of the screen are the control buttons. From left to right, these buttons allow to reset the simulation, play and pause the execution and step through the algorithms backward and forward.

Software Architecture. The implementation of the sorting algorithms themselves is commonly known and straightforward. Appendix A contains the pseudo-codes that we used as an implementation basic. What increases the implementation complexity is that we want to have interactive visualizations of algorithms that display all the current state's vital

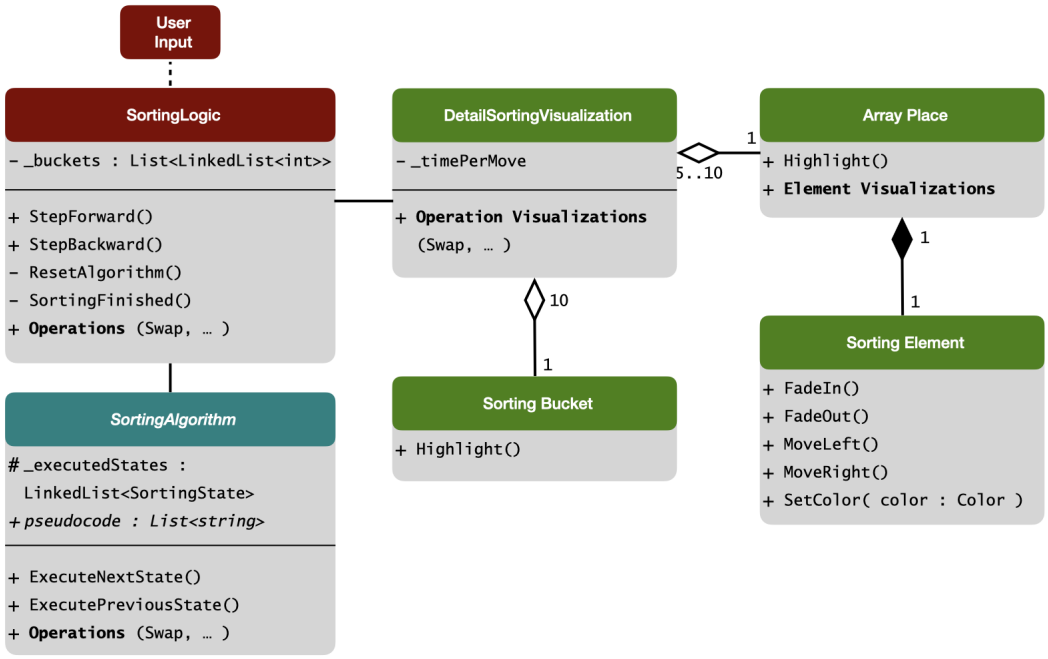


Figure 4.17.: A UML diagram of the detail-view's architecture.

information. Figure 4.17 shows the architecture we came up with. It is based on the model-view-controller concept (Krasner, Pope, et al., 1988). The *SortingLogic*, marked in red, acts as the controller. It receives the user input, such as taking steps forward and backward and directs the other classes. The *SortingAlgorithm* class in blue is an abstract class. It acts as the model by providing all the information about specific algorithms. It is also responsible for storing the execution history. The *DetailSortingVisualization*, shown in green, acts as the view and visualizes the current sorting state. The following paragraph discusses the various visualizations.

Visualizations. We designed the sorting elements as spheres of different sizes. The size allows seeing if the elements are arranged correctly at one glance. While executing a sorting algorithm, operations are performed on the elements. To notify users about these operations, we visualized the following operations on the field:

Figure 4.18.: The radix sort visualization has additional buckets. In this picture the element of value 26 is sorted to the bucket number 6.

- **Comparison.** To indicate that we compare two elements, we set a spot highlight to them.
- **Swapping.** We animated the swapping of two elements by shrinking both elements' size until they disappear and let them reappear at the other position. In the process, we highlight the elements with the same highlight used for comparison.
- **Inserting.** Just as in the swapping animation, the element disappears and reappears in the new position. Simultaneously, all elements between the new and the old position shift by one place in a smooth movement to make room.
- **Moving from/to bucket.** This operation is only needed for radix sort. To allow the visualization, we added ten buckets to the scene when switching to radix sort. When an element is moved to a bucket, it disappears. While disappearing, the bucket that the element is moved to is highlighted. When moving from a bucket, the element reappears in the new position, and again the bucket is highlighted. Figure 4.18 shows the moving to bucket animation in action.

Apart from these operations, we visualized other features that help the user to understand the current state. In recursive algorithms, we highlight the subset of elements that is currently under investigation. This makes it easier for users to follow the recursion. Moreover, we display the state of all local variables that are used in the pseudo-code. Variables that store an index are displayed above the corresponding element to link the index to the element. All other variables are displayed below the pseudo-code.

As shown in Figure 4.17, the *DetailSortingVisualization* orchestrates the visualizations. It stores a list of all *ArrayPlace* instances, which contain a *SortingElement* and the highlight for this element. The *SortingElement* class implements all the visualizations performed on the sphere that represents the element, such as moving to the left and right, changing the color, disappearing, and appearing.

Sorting Algorithm Implementation. As described above, the sorting algorithm acts as the model and is implemented as an abstract class. Each algorithm derives from this class to implement the logic specific to that algorithm. The complexity of the algorithm class comes from the requirement that users should take backward steps. This means that we have to store an execution history to be able to recover previous states. Figure 4.19 shows the implemented architecture, using insertion sort as an example. The *SortingAlgorithm* class stores a linked list of *SortingState*. These states are the basis of each sorting algorithm, as they store all the execution information, such as the local variable values. They are linked to a line of the pseudo-code via the *SortingStateLine* enumeration. When the *Execute* method is called, it executes all needed operations, such as swap, in the SortingAlgorithm class, which forwards them to the SortingLogic. Moreover, it sets the local variables' values and defines which line of the pseudo-code comes next. The *Next* method then creates a new sorting state for the next line, allowing to store the old state in the execution history. What was especially difficult to handle were subroutine calls, as they often appear in recursive algorithms. We addressed this, by adding the methods *EnterSubroutineWithExitLine* and *LeaveSubroutine* of the *SortingElement* class that automatically store all vital information and later recover it.

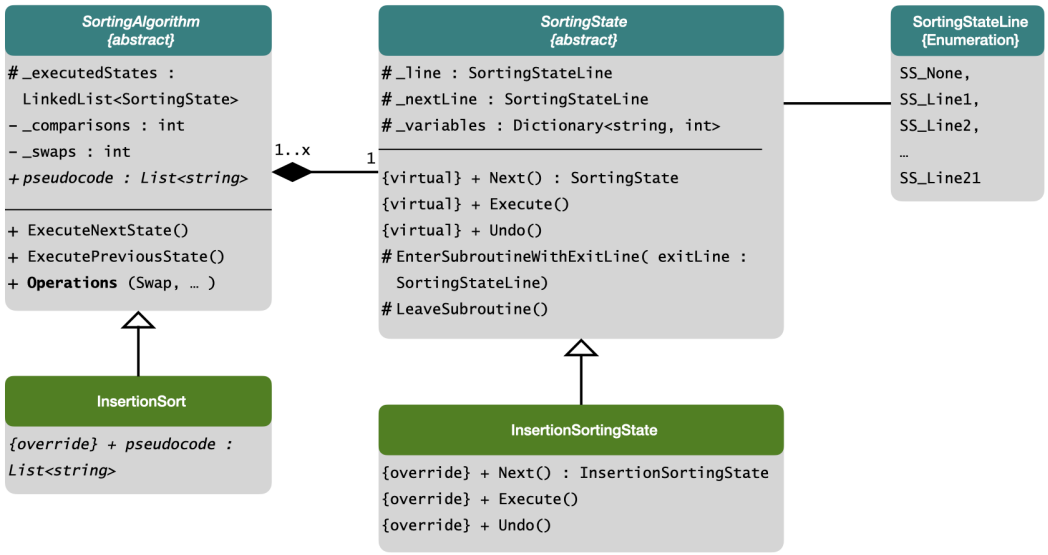


Figure 4.19.: A UML diagram of the *SortingAlgorithm* class and its dependencies.

The *ExecuteNextState* and *ExecutePreviousState* methods of the *SortingAlgorithm* class are called by the corresponding functions of the sorting logic. Listing 4.2 shows the implementation of the *ExecuteNextState* method. First, we create a new state, using the *Next* method of the previous state. After updating the pseudo-code visualization, we check if sorting is finished. If there is still an executable state, we execute it. We store the executed state in a linked list to recover it if users want to go backward. Afterward, we update the visualizations of the subset, the indices, and the execution information. The *ExecutePreviousState* method recovers the last element from the linked list, containing all the visualization information, and updates the visualization. We call the *Undo* method of the state, which reverses any operations that the state's execution performed, such as swaps or inserts.

To implement a specific sorting algorithm, we have to derive classes both from the *SortingAlgorithm* class and the *SortingState* class. All the algorithm-specifics are implemented in the '*Execute*' method of the *SortingState*.

```
public void ExecuteNextState()
{
    newState = _executedStates.Last.Value.Next();
    sortingLogic.SetPseudocode();
    if (newState.GetLine() != SortingStateLine.SS_None)
    {
        newState.Execute();
        _executedStates.AddLast(newState);
        sortingLogic.MarkCurrentSubset();
        sortingLogic.DisplayIndices();
    }
    else
    {
        sortingLogic.SortingFinished();
    }
    sortingLogic.SetSwapsComparisons(_swaps, _comparisons);
}
```

Listing 4.2: Implementation of the *ExecuteNextState* method in the *SortingAlgorithm* class.

4.4.2. Battle-View

The battle-view allows users to execute two sorting algorithms on a large field side by side. We implemented it using the design defined in Section 3.8. Figure 4.20 shows the implemented battle-view. In the center, two algorithms operate on a sorting field. On the right, users can select different options. Besides changing the selected algorithms, they can adapt the execution speed and change the arrangement mode. The arrangement mode can be random, sorted, or reversed. These states cover the best-case and worst-case scenarios for many algorithms, making them a valuable selection choice. On the left side is the sorting challenge, which we describe at the end of this chapter.

Software Architecture. Using a model-view-controller architecture has the advantage that parts of the architecture can be exchanged. We tried to implement the battle-view by reusing the model and the controller of the detail-view, in our case, the *SortingAlgorithm* and the *SortingLogic* class. We replaced the view component with a new visualization for the battle-view.

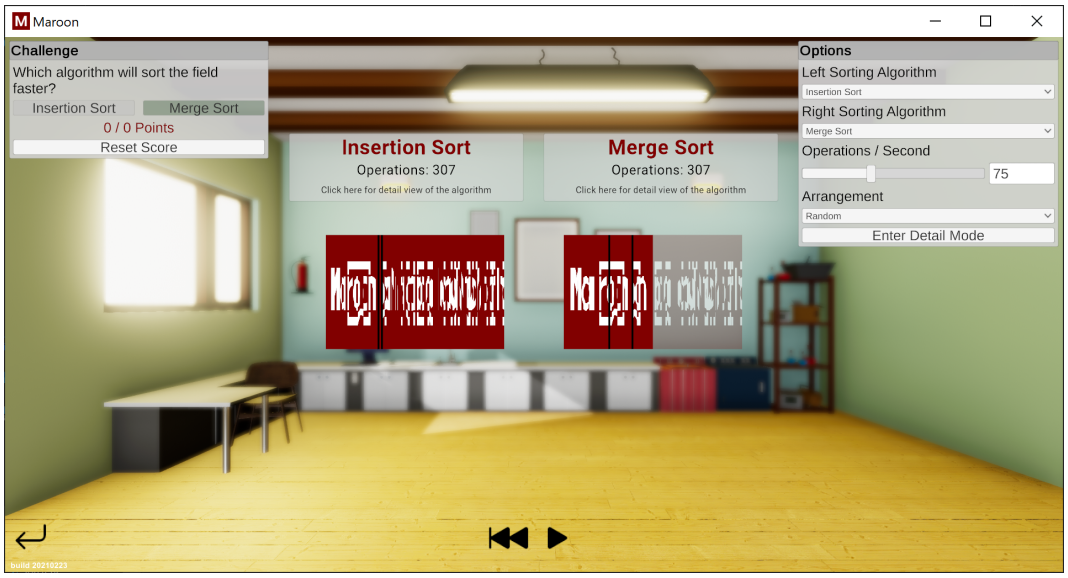


Figure 4.20.: The battle-view of the sorting experiment.

However, sorting fields in the battle-view are larger, and the execution speed is faster than in the detail-view. Testing the approach, we realized that we used too many resources and could not reach the desired speed. Therefore we came up with an independent solution for the battle-view.

There is no need to move backward in the battle-view, so we do not need to store an execution history. We moved away from the model-view-controller approach due to the performance issues mentioned above. Instead, we created one class responsible for all the battle-view executions: the *BattleSorting* class. Figure 4.21 shows its class diagram. Each algorithm is implemented as a co-routine that calls the various visualization functions. The *SortingStarter* method starts the co-routine and the *operationsPerSecond* define how fast the co-routine is executed. All operations are executed directly on the sorting elements implemented in the *SortingColumn* class, representing one column of the sorting field.

Visualizations. We want the sorting algorithms to compete on a field of 100 elements, but sorting 100 spheres of different sizes does not work for this amount. Especially as we want to sort two fields side by side. Therefore,

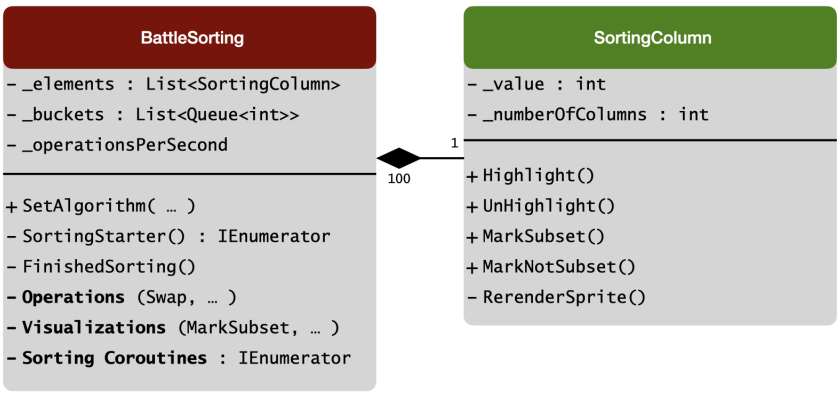


Figure 4.21.: A UML diagram of the battle-view's architecture.

we came up with the idea of splitting an image into 100 stripes and using them as sorting elements. We decided to use Maroon's logo for this purpose. To be consistent with the detail-view, we visualized the same actions:

- **Comparison.** To indicate that we compare two columns, they are both colored in black.
- **Swapping & Inserting.** At the speed that the simulation runs in, it is sufficient to adapt the elements' positions and pause the animation for a moment.
- **Moving from/to bucket.** If an element has been moved to a bucket, its transparency is lowered. Once it is moved from the bucket, the transparency is set back to normal.
- **Subsets.** The same visualization is used to mark the current subset of a recursive algorithm. All elements that are not part of the subset have lower transparency.

Local variables' values do not have to be visualized in the battle-view, as there is no pseudo-code in this view.

4.4.3. Sorting Challenge

The sorting challenge is a mini-game integrated into the battle-view. Users can guess which algorithm will sort the field faster. In single-user mode, it

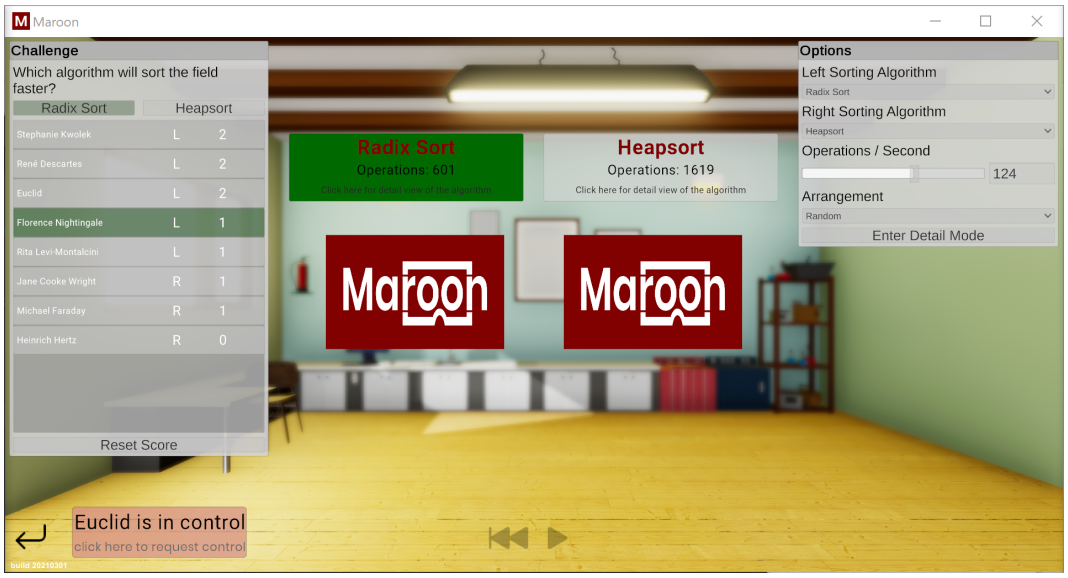


Figure 4.22.: Eight users compete in the sorting challenge, shown on the left side of the screen.

counts how many points of possible points the user achieved. However, the challenge is most interesting in multi-user mode.

We added multi-user support to the sorting experiment using the guideline presented in Section 4.3. As in the other experiments, the controlling user's actions are mirrored to the other users. However, we used the experiment as a proof of concept that it is also possible to implement an interactive solution for all connected users. Each user can individually guess which algorithm will sort the fields faster, and they compete who guesses best. Once again, users receive a point if they guess correctly. The user's choice is hidden to other users until the sorting starts. When started, the choices are revealed, and the guessing interface is disabled until selecting new algorithms or resetting. The user in control additionally has the option to reset the scores. Figure 4.22 shows an example where eight users compete in the challenge.

4.5. Summary

This chapter described the implementation details for both the multi-user network and the sorting experiment. It presented the procedure of establishing a network connection where both local and online users can join the same server. Simple password protection was added to keep uninvited users from joining. To prevent chaos on a server with numerous users, we developed a control handling approach, where only one user has control over the adjustable parameters. Any user can request the control at any time. Moreover, this chapter presented a solution to how the multi-user network can be added to existing experiments. A base class with helpful methods was created for use by future developers. Furthermore, we provided a guideline with all the necessary steps.

The end of the chapter described the implementation of the sorting experiment. A sophisticated software architecture had to be developed for the detail-view, allowing the execution of algorithms stepwise, both forward and backward. The designed architecture worked well for the slow-paced detail-view. However, it was not suitable for the battle-view, where the algorithms compete on larger data sets and run faster. Therefore, the battle-view needed an independent implementation. Moreover, it presented the sorting challenge as a proof of concept for an interactive multi-user implementation. The following chapter presents a study conducted on the sorting experiment, where we compared collaborative groups, using our multi-user network, to autonomous users.

5. Evaluation

To evaluate the effectiveness of the sorting visualization experiment for computer science education in combination with a multi-user network, we performed an AB study with 35 participants. The following research focuses were set for the study:

- Comparing single-user to multi-user operation of Maroon.
- Exploring how collaboration affects learners' emotions and learning outcomes.
- Identifying student expectations for conceptual learning in a collaborative environment.
- Evaluating the usability of the created program.

5.1. Methodology & Procedure

We constructed our study according to the framework provided by Naps et al. (2002), designed for measuring the effectiveness of algorithm visualizations. The framework suggests, to query the participants with the same theory questions before and after using the application to measure the knowledge increase. We adapted the framework by splitting the participants into two groups: (A) single-user and (B) multi-user. The single-user participants received the instructions online and performed the tasks autonomously. The multi-user participants were divided into pairs, where each pair received a timeslot. The multi-user pairs executed similar instructions but were connected via voice chat and joined a server for performing the tasks. They could work on the experiments collaboratively using the new multi-user part. However, they also only received their instructions online and did not get any tutorial or instructor support. The participants had

to perform the following activities: (1) pre-questionnaire, (2) predefined tasks, (3) post-questionnaire. Appendix B presents all the questions of the two questionnaires. In the study they were provided both in English and German to prevent misconceptions.

5.1.1. Pre-Questionnaire

The pre-questionnaire was conducted to evaluate the users' experiences and prior knowledge. In the first part, the users had to rate their confidence in using computers, their programming skills, and their experience with sorting algorithms by reacting to statements on a scale from *strongly disagree* (1) to *strongly agree* (5). Moreover, they indicated whether they have worked with e-learning tools before. In the second part, we queried their knowledge on sorting algorithms with ten questions. We created these questions based on Bloom's taxonomy (Bloom, 1956), splitting the learner's depth of understanding into six levels: (1) knowledge, (2) comprehension, (3) application, (4) analysis, (5) synthesis, and (6) evaluation. We will not reach the two deepest levels with our learning application, as this is not possible in such a short time. Therefore, we focused our questions on the remaining four levels. We created one question for the knowledge level, two questions for the comprehension level, three questions for the application level, and four questions for the analysis level.

5.1.2. Tasks

After finishing the pre-questionnaire, the users were asked to perform a series of tasks in Maroon's newly added sorting experiment. As mentioned above, the multi-user group had to connect to a server to perform the tasks. Both groups executed the following tasks:

1. Familiarize yourself with merge sort, insertion sort, and radix sort without using the battle-view mode yet. There will be a small challenge at the end, where you can apply your knowledge. Take as much time as you need.

2. Once you feel confident that you understand the sorting algorithms switch to the battle-view. Before you start sorting, always choose one of the two algorithms in the challenge.
   a) First, let merge sort compete against insertion sort.
   b) Next, let merge sort compete against radix sort.
   c) Last, change the arrangement to sorted. Then let insertion sort compete against radix sort.

While the users performed these tasks, we automatically measured their time in the detail-view and their performance in the challenge.

5.1.3. Post-Questionnaire

After these tasks, the users were asked to fill out a post-questionnaire. The questionnaire gathered information about users' perception of the experiment and evaluated if their knowledge increased throughout the experience. It was made up of the following sections.

Overall Impressions. In this section, users rated their overall impression of the sorting experiment, the battle-view, and the sorting challenge. They provided textual feedback and suggestions for improvement. The multi-user participants answered additional questions on the multi-user implementation.

Theory Questions. As suggested by Naps et al. (2002), we asked the same ten questions that the users already answered as part of the pre-questionnaire. This allowed judging if the users' knowledge increased accurately.

Computer Emotion Scale (CES). We assessed the users' emotions while working with the program using the computer emotion scale defined by Kay and Loverock (2008). They introduced the scale to measure feelings while learning new software. It measures twelve feelings, where users can rate how often they felt it on a scale from *none of the time* (0) to *All of the time* (3). The feelings are then evaluated in four groups: happiness, anger, anxiety, and sadness.

System Usability Scale (SUS). Brooke (1996) developed the SUS as an easy evaluation tool for usability. It has become so commonly used that it now also provides a benchmark to compare a system's usability to other programs. Users rate the usability with ten different questions that they answer on a scale from *strongly disagree* to *strongly agree*. The results are interpreted as values between 0 and 4. The values are summed up and multiplied by 2.5, resulting in a scale from 0 to 100 points. Over the years, many ways to interpret the scale have emerged. However, for this study we rely on the interpretation by Bangor et al. (2009), as suggested by Brooke (2013).

Learning Experience. To measure the learning experience, we adapted the questionnaire from Pirker et al. (2020). The adapted version contains 16 questions that users answer on a Likert scale from *fully disagree* (1) to *fully agree* (7). The questions determine if users enjoyed the learning process and think that the learning environment could be used in practice.

Collaboration. To evaluate the collaboration with the partner in the multi-user group, we adapted the classroom community scale by Rovai (2002). While the scale is focused on classroom collaboration, many of the questions are also suitable to measure collaboration in small groups. The questions are answered on a Likert scale from *fully disagree* (0) to *fully agree* (4).

Online Student Engagement. The online student engagement scale by Dixson (2015) provides a scale to measure students' engagement in online learning scenarios. Students describe themselves concerning certain behaviors, thoughts, and feelings. The scale goes from *not at all characteristic of me* (1) to *very characteristic of me* (5).

5.2. Participants

As the study required some prior knowledge on sorting algorithms, we targeted participants who have a computer science background. Thirty-five people participated in the study, aged between 21 and 34 (M=26.91; SD=2.98). There were six female and 28 male participants. We assigned 15 participants (3 female, 12 male) to the single-user group and 20 participants

(3 female, 17 male) to the multi-user group. The multi-user group pairs were chosen randomly, depending on the timeslots in which the users were available. Both groups rated themselves as expert computer users (single-user: M=3.93; SD=1.22, multi-user: M=4.10; SD=1.12). Regarding the experience as a programmer, the single-user participants (M=3.87; SD=1.46) rated themselves slightly higher than the multi-user participants (M=3.20; SD=1,24). Also, with sorting algorithms, the single-user group (M=2.73; SD=0.80) has slightly more experience than the multi-user group (AVG=2.35; SD=0.88). None of the users ranked themselves as an expert in the topic of sorting algorithms. Out of the 35 participants, 28 have worked with e-learning tools before. The following section presents the results that we obtained in our study.

5.3. Results

The study results were evaluated for each group individually to compare the two groups. This section presents the outcome by category.

Gathered Knowledge. Figure 5.1 compares the results of the two groups on the theory questions in the pre-questionnaire and post-questionnaire. Confirming the single-user participants' self-assessment, they performed better on the pre-questionnaire theory questions than the multi-user participants (single-user: M=5.83; SD=1.67, multi-user: M=4.50; SD=7.74). After the study, both groups' knowledge improved, with the single-user group still slightly ahead (single-user: M=7.93; SD=0.98, multi-user: M=7.74; SD=7.93). However, the multi-user group had a higher increase (M=3.24; SD=2.27) than the single-user group (M=2.10; SD=2.00). For both groups, the increase was highly significant ($\alpha < 0.001$). As for the following sections, the significance was determined with a two-sample t-test.

On-Task Measurements. The time that users spent in detail-view before entering the battle-view challenge varied enormously across the groups. The single-users spent 9.6 minutes on average in detail-view, with a mentionable

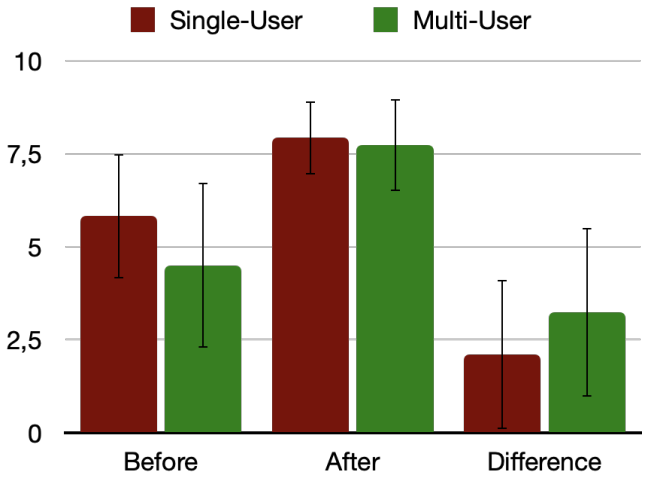


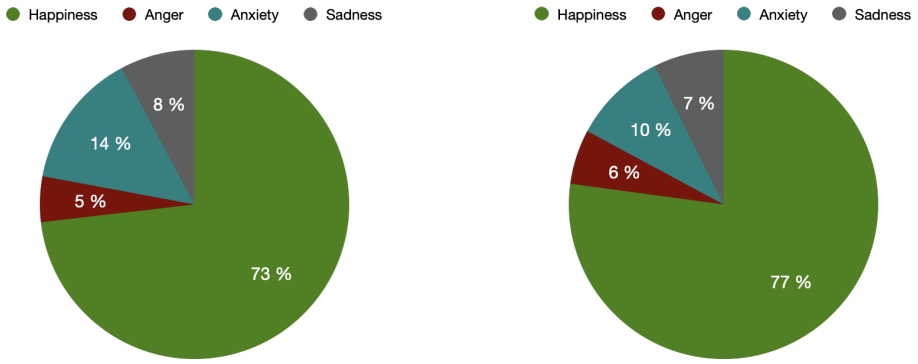
Figure 5.1.: Results on the theory questions before and after learning with Maroon.

standard difference of 10.7 minutes, while the multi-users spent 14 minutes (SD=7.4min) on average. Both groups performed equally well on the three tasks in the sorting challenge (single-user: M=2.40; SD=0.63, multi-user: M=2.56; SD=0.51).

Overall Impression. All users agreed that they liked the sorting experiment (M=4.23, SD=0.74). There was a difference in the perception of the battle-view. The single-user group (M=4.73; SD=0.46) liked it significantly (two-sample t-test: $\alpha < 0.02$) better than the multi-user group (M=3.95; SD=1.10). However, both groups agreed that the battle-view "*gave a clearer understanding of the complexity of the algorithms*" (M=4.14; SD=1.09).

The multi-user group overall liked the multi-user feature (M=3.60; SD=1.23). However, three of the 15 users disliked it, with one user stating to like it *not at all*. They disliked the concept that only one user is in control and would rather have more interactions for all clients.

Emotions. The CES results were evaluated for the four categories happiness, anger, anxiety, and sadness. While Figure 5.2 shows the composition of the groups' feelings, Table 5.1 shows the observed results as numbers. The



(a) Single-User (b) Multi-User

Figure 5.2.: Comparison of CES results for single-users and multi-users.

Feeling	Single-User		Multi-User	
	AVG	SD	AVG	SD
Happiness	1.69	0.68	1.89	0.74
Anger	0.11	0.31	0.14	0.37
Anxiety	0.33	0.35	0.24	0.38
Sadness	0.18	0.36	0.18	0.40

Table 5.1.: The obtained values of the CES.

obtained emotions are predominantly positive, with the multi-user group being slightly happier. The only significant (two-sample t-test: $\alpha < 0.05$) difference was that the single-user participants (M=0.87; SD=0.34) felt more dispirited than the multi-user participants (M=0.53; SD=0.50).

Usability. The average system usability score of the single-user group was 81.67 (SD=9.71), which according to Bangor et al. (2009) indicates good usability. On the other hand, the average SUS of the multi-user group was 72.24, indicating OK to good usability. However, the answers of the multi-user group were diverse (SD=17.08). Nevertheless, none of the users scored less than 35.7 points, indicating poor usability. The two groups differed most significantly (two-sample t-test: $\alpha < 0.02$) on whether the system was easy to use (single-user: M=4.27; SD=0.70, multi-user: M=3.53; SD=0.96). Figure 5.3 shows the achieved SUS of both groups, extended by the adjective

Figure 5.3.: Results of the SUS, extended by the adjective rating scale of Bangor et al. (2009).

rating scale of Bangor et al. (2009).

Learning Experience. Overall, the feedback in this section was positive. Both groups agreed that they learned something through the experience (M=6.09; SD=1.46) and that the application is a good supplement for learning (M=6.06; SD=1.19). They even stated that the experience inspired them to learn more about sorting algorithms (M=5.00; SD=1.71). Both groups would prefer learning with the application at home (M=5.06; SD=1.84) to learning with it in the classroom (M=4.68; SD=1.98).

Collaboration. The observed collaboration in the multi-user group was very high. 10 of the 20 users stated to fully agree that they can rely on their team partner, while none disagreed. The users did not feel uneasy about exposing their knowledge gaps (M=0.68; SD=0.92) or reluctant to speak openly (M=0.47; SD=0.82). Also, they disagreed with the statement that their team partner did not help them learn (M=0.42; SD=0.82).

Engagement. The groups differed significantly (two-sample t-test: $\alpha < 0.02$) on the characteristic, if they participate actively in small group discussion forums (single-user: M=2.53; SD=3.63, multi-user: M=3.58; SD=1.17). Overall, the multi-user group rated their engagement higher in 17 of the 19 characteristics.

5.4. Discussion

The study results were overwhelmingly positive. Participants enjoyed the chance to refresh their knowledge on sorting algorithms. We observed that many participants explored the sorting experiment beyond the defined tasks, investigating those algorithms that were not part of the study. Multiple pairs of the multi-user group stayed in the experiment after all tasks were finished to compare more algorithms in the battle-view. Several users also mentioned the battle-view as the standout positive feature. One user of the single-user group suggested to "*implement a leaderboard for the Battle mode and let players compete with each other*" and another user suggested that "*Everything could be online without the need to download a file*". Both of these features are already implemented but were not part of the single-user study. However, it is positive that users request supported features.

The comparison showed that the multi-user group's knowledge level increased more considerably than the single user group's. However, this is relativized by the fact that the single-users performed better overall. Both groups increased their knowledge significantly (two-sample t-test: $\alpha < 0.001$). The multi-user participants were slightly happier throughout the study. Moreover, the multi-users tend to spend more time on the application. Their feedback and the great assessments of the sorting challenge indicate that they like the addition of a competitive element to the collaborative environment.

The usability was good for both groups, especially when keeping in mind that they did not have a tutorial. However, the SUS Score of the single-user group was significantly (two-sample t-test: $\alpha < 0.1$) better than the multi-user groups'. This result is not surprising, as the experiment controls were the same for both groups, but the multi-user group had the additional

effort of joining a server and dealing with the server's control handling. Nevertheless, we take it as a hint that the network part's usability can be improved further.

The study was also an excellent opportunity to test the multi-user part's performance and stability and the sorting experiment. None of the users experienced any crashes or other notable issues. However, one user reported experiencing lags. Several users pointed out that the program was resource-heavy. We could trace this problem back to the fact that it does not limit the framerate. Predominantly positive was that the test server for the multi-user group ran for several hours without any problems, with constantly new users connecting. We can therefore conclude that our network implementation is stable.

6. Lessons Learned

This chapter discusses the experiences we had in the creation process of this thesis. It is chronologically split into the three project phases: (1) theoretical background and literature research, (2) design and development process, and (3) the evaluation of the created application.

6.1. Theory

The literature research presented digital learning and collaborative learning in the context of computer science education. While both are effective methods on their own, they can be combined into collaborative e-learning. If done right, this can create a productive learning platform with social components, allowing users to benefit from exchanging with others. However, the literature research showed that not every implementation of an e-learning platform is an exemplary implementation. Many things have to be kept in mind to successfully create a practical learning application, such as engaging and motivating users. Moreover, we had to refresh our knowledge on sorting algorithms before starting the development. Many concepts of algorithms can be effectively showcased by using sorting algorithms. However, the implementation has to be well-planned.

6.2. Development

For development, we used Unity, as already predetermined by the framework. It proved to be a valuable environment for the implementation of our 3D experiment. However, we had to find a way to create a multi-user

network. With Mirror, we found a tool that provides all the functionalities we need, such as hosting and joining servers and synchronizing data over the network. Nevertheless, it does not provide a solution for NAT traversal, so we utilized Open-NAT to establish automatic port forwarding.

A big challenge for the development process was to work on such a continually evolving project as Maroon, where multiple programmers work on the project simultaneously. The workflow is well defined, and by using the version control platform Github, there were never any problems of getting in each other's way. However, the multi-user part spans across vast parts of the project, so it had to be adapted whenever new features were added. The coordination was achieved in regular meetings with Maroon's project lead.

6.3. Evaluation

The evaluation was based on a framework for algorithm visualization studies provided by Naps et al. (2002). Following their instructions, we queried the participant's knowledge on sorting algorithms before and after learning with our application, using the same questions. Moreover, we extended the post-questionnaire by well-researched standardized questionnaires and environment-specific questions. The study was executed in two groups, one where users worked independently and one where users worked in pairs, using the new multi-user feature. The study showed that the participants liked the application and deepened their knowledge. It also revealed various improvements for the future, which are presented in the following section.

7. Future Work

Our implementation fulfills all the requirements that we defined in Chapter 3. However, some desirable functionalities exceed these requirements, so we left them open for implementation in the future. This chapter presents these functionalities. The first section focuses on the feedback we received as part of our study. The second section presents ideas that we came up with during the design and implementation phases.

7.1. Evaluation Results

In the study, users had the opportunity to suggest improvements, both for the sorting experiment and the multi-user part. For the sorting experiment, they mostly requested more control over the parameters. They would like to adapt the execution speed in the detail-view and the input field's size in the battle-view. Moreover, several users reported that they did not identify the Maroon logo as the sorting field first. It would be desirable to exchange the Maroon logo with other representations, such as bars of different lengths. These parameters are already accessible via code, so only the corresponding user interface elements have to be created and linked for implementing these features.

Some users noted that while the experiment visualizes the algorithms' time complexity effectively, it never states the concrete time complexity in big O notation (Section 2.3.1). This was mainly done because the big O notation is never explained inside the experiment, and not all users are familiar with the concept. The suggestion could be implemented by adding a set of slides to the whiteboard experiment that explain the big O notation. Inside the

experiment, the time complexity of the algorithms could then be added to the existing description.

One issue that both the single-users and the multi-users identified is that it is easy to overlook the challenge. This can be adjusted by changing the challenge's appearance, so it stands out. Moreover, the challenge could be required before being able to start the sorting.

For the multi-user part, the main criticism was usability. It should be able to join servers over the main menu instead of first entering the laboratory. Moreover, it should be possible to enter the main menu when on a server. This feature is strongly connected to the new modular setup of Maroon, where users can switch from a computer science laboratory to a physics or a chemistry laboratory over the main menu. The main menu has to be synchronized over the network to enable this feature.

Another improvement can be made concerning interactivity. While the sorting challenge was a pleasant exception, many parts of the sorting experiment were only accessible for the client in control. At the same time, the other user did not always know what is happening. One user suggested displaying the controlling client's mouse position to keep the other clients involved. Overall, it would be desirable to design future experiments in ways where all clients are more actively engaged simultaneously.

7.2. Technical Improvements for Multi-User

This thesis's work laid the basis for multi-user functionalities in Maroon by providing the option to host and join servers. However, many other things can contribute to the network part, and there are sections of Maroon that do not have multi-user support yet. This section describes some ideas for future implementations.

7.2.1. WebGL & VR Support

As described in Section 2.2, Maroon consists of three individual versions. While we equipped the desktop version with multi-user support, both the WebGL and the VR version currently do not have it. To add network support to WebGL, we would have to change to WebSocket transport, as WebGL does not support the current TCP transport. Moreover, WebGL does not support local network discovery, so that feature will have to be disabled.

Adding multi-user support to the VR version is more complex because it is more distant from the desktop version. However, the desktop version's implementation can be used as a guideline to implement the new functionality. The core modules could be reused, reducing the efforts for future development.

7.2.2. Steam Networking

Steam Networking is a network service provided by Steamworks[1]. In the design phase, we considered using it as a network technology layer on top of Mirror. It provides network messages over the Steam servers, and it automatically takes care of NAT traversal. It even falls back on Steam's relay servers if no direct connection is possible, so a connection from the client to the server can be guaranteed. With the FizzySteamworks[2] add-on, it is drag-and-drop supported for Mirror, so the setup complexity is minimal. Another advantage would be that we could also use Steamworks' other functionalities, such as a lobby system or voice chat. The main disadvantage is that each user would need a Steam account, and Steam would have to be running in the background when starting Maroon. Also, an app ID would be needed to correctly use the network, for which we would need to register our program in the Steam library. Moreover, Steam does not support WebGL. It would be possible to create a hybrid solution, where users can decide if they want to host a regular server using automatic port mapping or a Steam server. This would combine all the advantages of the two methods, but it increases the network code's complexity and makes it harder to maintain.

[1] https://partner.steamgames.com/doc/features/multiplayer/networking
[2] https://github.com/Chykary/FizzySteamworks

	Current	Steam Networking	Hybrid
NAT Traversal	✓	✓✓	✓✓
Independence	✓✓	xx	x
Server costs	x	✓	x
WebGL Support	✓	x	✓
Complexity	x	✓	xx

Table 7.1.: Comparison of the current solution using port forwarding, Steam networking and a hybrid approach. he ranking in each category goes from xx - vital disadvantage, to ✓✓ - vital advantage.

Table 7.1 compares the three different options. Based on the findings, we decided to stick to the independent solution, relying on automatic port forwarding. This decision was made in consultation with the project lead of Maroon. The main reason was that the program is currently shipped only over the website, and the freedom to choose a distributing platform should not yet be sacrificed. The hybrid solution would be too complicated, both for the user and for maintenance. However, if Maroon is added to the Steam library in the future, this decision might be reconsidered. Note that Epic Games[3] also provides a network service that handles NAT traversal. Everything stated in this section about Steam also applies to Epic Games.

7.2.3. NAT Punch-through

As already described in Section 3.7.4, NAT punch-through would be a valuable addition to the network setup. In case the automatic port forwarding fails, clients could try to connect to the server using hole-punching. It might be possible to use the server we acquired for the list server (Section 4.1.2) as the facilitator for the hole-punching process.

[3]https://dev.epicgames.com/en-US/services

7.2.4. User Interactions

Currently, we assume that users are already connected via voice chat over a platform independently of Maroon. While this is a reasonable assumption, it might be desirable in the future to add a voice chat component directly to Maroon. This would allow users who are not already connected or do not know each other to connect to servers and collaborate with other users on the server.

Moreover, we advise replacing the multi-user avatar that is spawned for each user in the laboratory room. Currently, we reused the model of the help character. We leave it open to more artistically talented developers on the Maroon team to create a new character. This character could be equipped with animations that allow users to communicate language-independently. We already provide the functionalities to synchronize these animations over the network in our current network solution.

7.2.5. Experiment Status Saving

Maroon's development team currently plans to add status saving to experiments. This would allow saving the current status of an experiment to return to it later. However, this would also mean that we could send this current status to other clients over our network, which means that we can lift the restriction that users can not join running experiments. Moreover, it would be possible to create a database on our server to upload and download interesting experiment setups.

8. Conclusion

Computer science is a future-oriented field, but it is also already heavily requested in the present. The demand for computer scientists has constantly grown over the past years, and there are no signs that this trend will change anytime soon. Therefore, computer science education has become an increasingly important topic. As in many other courses, the traditional way to teach computer science has been in-person presentation by a lecturer. However, new and innovative methods of learning are becoming more popular. Two of these methods are collaborative learning and digital learning. Especially the latter has gained importance through the recent coronavirus pandemic. The two methods can be combined, resulting in computer-supported collaborative learning. While still being accessible from everywhere, this adds a social component to digital learning allowing students to benefit from each other.

This work presented the design and implementation of a computer-supported collaborative learning environment, building upon the immersive learning and experiment environment Maroon. A multi-user network was added to the laboratory, allowing users to experience the experiments together. Moreover, computer science was added to Maroon as a brand new educational area. As the first contribution to this field, a sorting experience was implemented explaining and visualizing different sorting algorithms. Besides investigating the algorithms' details, users can let the algorithms compete on a larger data set to develop a feeling for time complexity. Moreover, there is a challenge where users can guess which algorithm will be faster, increasing the students' engagement through involvement. This challenge was used as a proof of concept for an interactive multi-user experience, where all connected users compete against each other.

An evaluation was carried out to compare the sorting experiment when used alone or in a group. It showed that both groups of people significantly

increased their knowledge of sorting algorithms. The multi-users were slightly happier throughout the experience, and they tended to spend more time with the application. However, the usability evaluation showed that the multi-user controls could be improved further. Overall, the users agreed that adding a multi-user function to the framework makes it a more attractive application and that the sorting experiment is a valuable tool for learning.

Appendices

Appendix A.

Sorting Algorithms

We used the following pseudo-codes to implement the various sorting algorithms. They are also displayed to the users in the detail-view of the sorting experiment. The implementations are adapted from the *GeeksForGeeks* website[1]. We define the following variables and functions:

- *A* is the sorting field, where *A[i]* is the *i*th element of the field. *len(A)* defines the length of the field.
- *insert(i,j)* moves the element at index *i* to index *j*. All elements between *i* and *j* are increase or decreased by one to make room.
- *swap(i,j)* swaps the elements at indices *i* and *j*.
- *moveToBucket(i,b)* stores the element at index *i* into the *b*th bucket.
- *moveFromBucket(i,b)* places the first element in the *b*th bucket at index *i*.

A.1. Insertion Sort

```
for i = 1 .. len(A)-1:
    j = i-1
    while A[j]>A[i] and j>=0:
        j = j-1
    insert(i,j+1)
```

Listing A.1: The pseudo-code of insertion sort.

[1] https://www.geeksforgeeks.org/

A.2. Merge Sort

```
mergeSort(i, j):
    if i<j:
        k = (i+j)/2
        mergeSort(i,k)
        mergeSort(k+1,j)
        merge(i,k,j)

merge(i, k, j):
    r = k+1
    l = i
    while l<r and r<=j:
        if A[r]<A[l]:
            insert(r,l)
            r = r+1
        l = l+1
```

Listing A.2: The pseudo-code of merge sort.

A.3. Heapsort

```
n = len(A)
for i = len(A)/2+1 .. 0: #Build heap
    heapify(i,n)
for i = len(A)-1 .. 1: #Iteratively take maximum
    swap(0,i)
    n = n-1
    heapify(0,n)

heapify(j, n):
    l = 2*j+1
    r = 2*j+2
    m = j
    if l<n and A[l]>A[j]:
        m = l
    if r<n and A[r]>A[m]:
        m = r
    if m!=j:
        swap(j,m)
        heapify(m,n)
```

Listing A.3: The pseudo-code of heapsort.

A.4. Quicksort

```
quickSort(l, r):
    if l<r:
        k = partition(l,r)
        quickSort(l,k-1)
        quickSort(k+1,r)

partition(l, r):
    p = A[r] #pivot element
    k = l
    for j = l .. r-1:
        if A[j]<=p:
            swap(j,k)
            k = k+1
    swap(k,r)
    return k
```

Listing A.4: The pseudo-code of quicksort.

A.5. Selection Sort

```
for i = 0 .. len(A)-1:
    m = i
    for j = i+1 .. len(A)-1:
        if A[j]<A[m]:
            m = j
    if i != m:
        swap(i,m)
```

Listing A.5: The pseudo-code of selection sort.

A.6. Bubble Sort

```
for i = 0 .. len(A)-1:
    for j = 0 .. len(A)-1-i:
        if A[j]>A[j+1]:
            swap(j,j+1)
```

Listing A.6: The pseudo-code of bubble sort.

A.7. Gnome Sort

```
while i<len(A):
    if i==0:
        i = i+1
    if A[i]>=A[i-1]:
        i = i+1
    else:
        swap(i,i-1)
        i = i-1
```

Listing A.7: The pseudo-code of gnome sort.

A.8. Radix Sort

```
key = max(A)
exp = 1
while key / exp > 1:
    countingSort(exp)
    exp = exp * 10

countingSort(exp):
    for i = 0 .. len(A)-1:
        b = (A[i]//exp)%10
        moveToBucket(i,b)
    i = 0
    for b = 0 .. 10:
        if bucket[b] not empty:
            moveFromBucket(i,b)
            i = i + 1
```

Listing A.8: The pseudo-code of radix sort.

A.9. Shellsort

```
gap = len(A)/2
while gap>0:
    for i = gap .. len(A)-1:
        j = i
        while A[j-gap]>A[j] and j>=gap:
            swap(j,j-gap)
            j = j-gap
    gap = gap/2
```

Listing A.9: The pseudo-code of shellsort.

Appendix B.

Questionnaires

B.1. Pre-Questionnaire

General Questions.

#	Question	Answer Type
1	UserID (for linking the responses)	Text
2	I am an expert in computer usage	1 fully disagree - 5 fully agree
3	I am an expert in Usage of video games	1 fully disagree - 5 fully agree
4	I am an experienced programmer	1 fully disagree - 5 fully agree
5	I am an expert on the topic of sorting algorithms	1 fully disagree - 5 fully agree
6	Have you ever used e-learining tools?	Yes or No
7	What experience have you had with e-learning tools?	Text

Theory Questions.

#	Question	Answer Type
1	Name 3 sorting algorithms, except Merge Sort, Insertion Sort or Radix Sort.	Text
2	Explain merge sort in your own words	Text
3	You have a field of 100 integer values between 1 and 1,000,000. Which of the following algorithms would you suggest for efficient sorting? Explain why.	Merge Sort, Insertion Sort or Radix Sort + Text
4	What is a drawback of Radix Sort compared to Insertion Sort?	Text
5	Describe in words how a sorting field has to be arranged to lead to a worst case scenario for insertion sort	Text
6	Which parameters does the efficiency of Radix sort depend on?	Text
7	Which algorithm are you seeing in action? [Gif of Merge Sort]	Merge Sort, Insertion Sort or Radix Sort
8	Argue if Merge Sort or Insertion Sort performs better on large fields of randomized data. Explain why.	Text
9	How many insert operations does Merge Sort need to put the following field in ascending order? [2 3 5 1 6 4]	Number
10	How many insert operations does Insertion Sort need to put the following field in ascending order? [2 3 5 1 6 4]	Number

B.2. Post-Questionnaire

Overall Impressions: Sorting.

#	Question	Answer Type
1	UserID (for linking the responses)	Text
2	Did you participate in this study as part of a group?	Yes or No
3	How did you like the sorting experiment?	1 not at all - 5 very much
4	What did you like?	Text
5	What did you not like?	Text
6	The challenge made the Battle Mode more interesting.	1 fully disagree - 5 fully agree
7	Knowing that there is a challenge at the end motivated me to be more active.	1 fully disagree - 5 fully agree
8	How did you like the Battle Mode?	1 not at all - 5 very much
9	Comments	Text
10	The battle mode gave me a clearer understanding of the complexity of the algorithms.	1 fully disagree - 5 fully agree
11	Overall Impression: What could be improved? Where do you see possibilities?	Text
12	I would like to do the challenge in a bigger group, to compete with other people.	1 fully disagree - 5 fully agree

Overall Impressions: Multi-User.

#	Question	Answer Type
1	How did you like the multi-user feature of Maroon?	1 not at all - 5 very much
2	What did you like most about it?	Text
3	How did you like the interactions (Joining a server, taking control, ...)?	Text
4	What could be done better in terms of the multi-user part?	Text
5	The multi-user part was well-integrated into Maroon.	1 fully disagree - 5 fully agree
6	Adding multi-user functionality to Maroon makes it more attractive for use in practice.	1 fully disagree - 5 fully agree

Theory Questions. The same ten questions as in the pre-questionnaire.

Computer Emotion Scale (Kay & Loverock, 2008). While using the sorting algorithm experiment I felt... (0 none of the time - 3 all of the time)

#	STATEMENT
1	Satisfied
2	Disheartened
3	Anxious
4	Irritable
5	Excited
6	Dispirited
7	Insecure
8	Frustrated
9	Curious
10	Helpless
11	Nervous
12	Angry

System Usability Scale (Brooke, 1996). (1 strongly disagree - 5 strongly agree)

#	STATEMENT
1	I think that I would use this system frequently
2	I found the system unnecessarily complex
3	I thought the system was easy to use
4	I think that I would need the support of a technical person to be able to use this system
5	I found the various functions in this system were well integrated
6	I thought there was too much inconsistency in this system
7	I would imagine that most people would learn to use this system very quickly
8	I found the system very cumbersome to use
9	I felt very confident using the system
10	I needed to learn a lot of things before I could get going with this system

Learning Experience (Pirker et al., 2020). Answer the question with regard to the sorting algorithm experiment. (1 fully disagree - 7 fully agree)

#	STATEMENT
1	I would like to learn with it
2	It is a good idea to use it for learning
3	It is a good supplement to regular learning
4	I learned something with it
5	It makes the content more interesting
6	It makes the content easier to understand
7	It makes learning more engaging
8	It makes learning more fun
9	It makes learning more interesting
10	I would like to learn with it at home
11	I would like to learn with it in the classroom
12	The experience inspired me to learn more about sorting algorithms
13	Learning was more motivating than ordinary exercises
14	I would rather like to learn sorting algorithms with the sorting algorithm experiment than with traditional methods
15	I find regular computer science classes boring
16	Seeing the sorting algorithm visualizations on the computer was engaging

Collaboration (Rovai, 2002). How did you feel about the collaboration in your group? (1 strongly disagree - 5 strongly agree)

#	STATEMENT
1	I feel that I am encouraged to ask questions
2	I feel that it is hard to get help when I have a question
3	I do not feel a spirit of community
4	I feel uneasy exposing gaps in my understanding
5	I feel reluctant to speak openly
6	I feel that I can rely on my team partner
7	I feel that my team partner does not help me learn
8	I feel that I am given ample opportunities to learn
9	I feel uncertain about my team partner

Online Student Engagement (Dixson, 2015). How well do the following behaviors, thoughts, and feelings describe you, when in an online lecture? (1 not at all characteristic of me - 5 very characteristic of me)

#	STATEMENT
1	Making sure to study on a regular basis
2	Putting forth effort
3	Staying up on the readings
4	Looking over class notes between getting online to make sure I understand the material
5	Being organized
6	Taking good notes over readings, PowerPoints, or video lectures
7	Listening/reading carefully
8	Finding ways to make the course material relevant to my life
9	Applying course material to my life
10	Finding ways to make the course interesting to me
11	Really desiring to learn the material
12	Having fun in online chats, discussions or via email with the instructor or other students
13	Participating actively in small-group discussion forums
14	Helping fellow students
15	Getting a good grade
16	Doing well on the tests/quizzes
17	Engaging in conversations online (chat, discussions, email)
18	Posting in the discussion forum regularly
19	Getting to know other students in the class

Personal Information.

#	Question	Answer Type
1	Age	Number
2	Gender	Female, Male, or Other
3	How did you like the sorting experiment?	1 not at all - 5 very much
4	Highest level of education	Text
5	Profession	Student, Employed, Self-Employed, Unemployed, or other
6	Field of study / Job title	Text

Appendix C.

Installation Guide

C.1. Installation

- Desktop:
  - Download the Maroon project from https://github.com/GameLabGraz/Maroon.
  - Build the project.
  - Run the created executable.
- WebGL: Visit https://maroon.tugraz.at/

C.2. System Requirements

- Desktop:
  - For the required Unity version, please refer to https://github.com/GameLabGraz/Maroon.
  - For the requirements for running Unity or a Unity-created executable, refer to https://docs.unity3d.com/2019.4/Documentation/Manual/system-requirements.html.
- WebGL: Any recent version of Chrome, Firefox, or Edge.

Bibliography

Astrachan, O. (2003). Bubble sort: An archaeological algorithmic analysis. *ACM Sigcse Bulletin*, *35*(1), 1–5.

Aycock, J., Wright, H., Hildebrandt, J., Kenny, D., Lefebvre, N., Lin, M., Mamaclay, M., Sayson, S., Stewart, A., & Yuen, A. (2019). Adapting the"unessay"for use in computer science, In *Proceedings of the western canadian conference on computing education*.

Azmi, S., Iahad, N. A., & Ahmad, N. (2015). Gamification in online collaborative learning for programming courses: A literature review. *ARPN Journal of Engineering and Applied Sciences*, *10*(23), 18087–18094.

Baecker, R. (1981). Sorting out sorting. University of Toronto.

Baecker, R. (1998). Sorting out sorting: A case study of software visualization for teaching computer science. *Software visualization: Programming as a multimedia experience*, *1*, 369–381.

Baeza-Yates, R. A. (1995). Teaching algorithms. *SIGACT News*, *26*(4), 51–59. https://doi.org/10.1145/219817.219828

Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, *4*(3), 114–123.

Battistella, P. E., Von Wangenheim, C. G., Von Wangenheim, A., & Martina, J. E. (2017). Design and large-scale evaluation of educational games for teaching sorting algorithms. *Informatics in Education*, *16*(2), 141–164.

Baylor, A. L. (2009). Promoting motivation with virtual agents and avatars: Role of visual presence and appearance. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *364*(1535), 3559–3565.

Bharambe, A., Douceur, J. R., Lorch, J. R., Moscibroda, T., Pang, J., Seshan, S., & Zhuang, X. (2008). Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. *ACM SIGCOMM Computer Communication Review*, *38*(4), 389–400.

Bloom, B. S. (1956). Taxonomy of educational objectives: The classification of educational goals. *Cognitive domain*.

Boticki, I., Barisic, A., Martin, S., & Drljevic, N. (2013). Teaching and learning computer science sorting algorithms with mobile devices: A case study. *Computer Applications in Engineering Education*, *21*(S1), E41–E50.

Brettschuh, S. (2019). *Exploring the visualization of coulomb's law in an educational virtual reality environment*. Graz University of Technology.

Brooke, J. (1996). Sus: A "quick and dirty'usability. *Usability evaluation in industry*, *189*.

Brooke, J. (2013). Sus: A retrospective. *Journal of usability studies*, *8*(2), 29–40.

Buchbauer, B. (2019). *Educational computer science visualizations in virtual reality* (Master's thesis). Graz University of Technology. https://online.tugraz.at/tug_online/wbabs.showThesis?pThesisNr=68248&pOrgNr=37

Chang, V., GÃ1/4tl, C., Kopeinik, S., & Williams, R. (2009). Evaluation of collaborative learning settings in 3d virtual worlds. *International Journal of Emerging Technologies in Learning (iJET)*, *4*(2009).

Chesire, S., & Krochmal, M. (2013). *Nat port mapping protocol (nat-pmp)* (RFC No. 6886). RFC Editor. RFC Editor. https://www.rfc-editor.org/rfc/rfc6886.txt

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to algorithms*. Amazon Digital Services LLC. https://books.google.at/books?id=aefUBQAAQBAJ

Cotton, M., Eggert, L., Touch, J., & Westerlund, M. (2011). *Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry* (RFC No. 6335). RFC Editor. RFC Editor. https://www.rfc-editor.org/rfc/rfc6335.txt

Curry, C. (2017). *17 famous female scientists who helped change the world*. Retrieved May 11, 2020, from https://www.famousscientists.org/popular/

de Gouw, S., de Boer, F., & Rot, J. (2014). Proof pearl: The key to correct and stable sorting. *Journal of automated reasoning*, *53*(2), 129–139.

Deering, S., & Hinden, R. (1998). *Internet protocol, version 6 (ipv6) specification* (RFC No. 2460). RFC Editor. RFC Editor. https://www.rfc-editor.org/rfc/rfc2460.txt

Dershem, H. L., & Brummund, P. (1998). Tools for web-based sorting animation, Atlanta, Georgia, USA, Association for Computing Machinery. https://doi.org/10.1145/273133.274301

Dixson, M. D. (2015). Measuring student engagement in the online course: The online student engagement scale (ose). *Online Learning*, *19*(4), n4.

Douglas, S., Tanin, E., Harwood, A., & Karunasekera, S. (2005). Enabling massively multi-player online gaming applications on a p2p architecture, In *Proceedings of the ieee international conference on information and automation*.

Erhel, S., & Jamet, E. (2013). Digital game-based learning: Impact of instructions and feedback on motivation and learning effectiveness. *Computers & education*, *67*, 156–167.

Esnaashari, S., Welch, I., & Komisarczuk, P. (2013). Determining home users' vulnerability to universal plug and play (upnp) attacks, In *2013 27th international conference on advanced information networking and applications workshops*. IEEE.

Estivill-Castro, V., & Wood, D. (1992). A survey of adaptive sorting algorithms. *ACM Computing Surveys (CSUR)*, *24*(4), 441–476.

Franceschini, G., & Geffert, V. (2005). An in-place sorting with o (n log n) comparisons and o (n) moves. *Journal of the ACM (JACM)*, *52*(4), 515–537.

GameLabGraz. (2020). *Maroon*. Retrieved February 3, 2021, from https://maroon.tugraz.at/

Garcia, D. (2011). Universal plug and play (upnp) mapping attacks. *DEFCON-19*.

Gnu general public license, version 3 [Last retrieved 2021-01-20]. (2007).

Gütl, C., & Pirker, J. (2011). Implementation and evaluation of a collaborative learning, training and networking environment for start-up entrepreneurs in virtual 3d worlds, In *2011 14th international conference on interactive collaborative learning*. IEEE.

Haelermans, C., Ghysels, J., & Prince, F. (2015). Increasing performance by differentiated teaching? experimental evidence of the student benefits of digital differentiation. *British Journal of Educational Technology*, *46*(6), 1161–1174.

Halavais, A. (2016). Computer-supported collaborative learning. *The International Encyclopedia of Communication Theory and Philosophy*, 1–5.

Hammad, J. (2015). A comparative study between various sorting algorithms. *International Journal of Computer Science and Network Security (IJCSNS)*, *15*(3), 11.

Hemel, A. (2006). Universal plug and play: Dead simple or simply deadly? In *5th system administration and network engineering conference, delft, the netherlands*.

Hoare, C. A. (1962). Quicksort. *The Computer Journal*, *5*(1), 10–16.

Holly, M. S. (2019). *Interactive physics simulations in a room-scale virtual laboratory* (Master's thesis). Graz University of Technology. https://online.tugraz.at/tug_online/wbAbs.showThesis?pThesisNr=68264&pOrgNr=37

House, B. (2018a). *Evolving multiplayer games beyond unet*. Retrieved April 23, 2020, from https://blogs.unity3d.com/2018/08/02/evolving-multiplayer-games-beyond-unet/

House, B. (2018b). *Multiplayer connected games: First steps forward*. Retrieved April 23, 2020, from https://blogs.unity3d.com/2018/09/12/multiplayer-connected-games-first-steps-forward/

House, B. (2019). *Navigating unity's multiplayer netcode transition*. Retrieved August 7, 2020, from https://blogs.unity3d.com/2019/06/13/navigating-unitys-multiplayer-netcode-transition/

Hu, Z. (2005). Nat traversal techniques and peer-to-peer applications, In *Hut t-110.551 seminar on internetworking*. Citeseer.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, *13*(3), 259–290. https://doi.org/https://doi.org/10.1006/jvlc.2002.0237

Johnson, D. W., Johnson, R. T., Stanne, M. B., & Garibaldi, A. (1990). Impact of group processing on achievement in cooperative groups. *The Journal of Social Psychology*, *130*(4), 507–516.

Kay, R. H., & Loverock, S. (2008). Assessing emotions related to learning new software: The computer emotion scale. *Computers in Human Behavior*, *24*(4), 1605–1623.

Kazim, A. (2017). A comparative study of well known sorting algorithms. *International Journal of Advanced Research in Computer Science*, *8*(1).

Knuth, D. E. (1976). Big omicron and big omega and big theta. *SIGACT News*, *8*(2), 18–24. https://doi.org/10.1145/1008328.1008329

Knuth, D. E. (1998). *The art of computer programming: Volume 3: Sorting and searching*. Addison-Wesley Professional.

Krasner, G. E., Pope, S. T. Et al. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, *1*(3), 26–49.

Kreijns, K., Kirschner, P. A., & Jochems, W. (2003). Identifying the pitfalls for social interaction in computer-supported collaborative learning environments: A review of the research. *Computers in human behavior*, *19*(3), 335–353.

Kruse, K. (2015). *Benefits and drawbacks of e-learning*. Retrieved January 15, 2021, from https://brucedwatson.wordpress.com/2015/05/19/benefits-and-drawbacks-of-e-learning

Laal, M., & Ghodsi, S. M. (2012). Benefits of collaborative learning. *Procedia-social and behavioral sciences*, *31*, 486–490.

Laal, M., & Laal, M. (2012). Collaborative learning: What is it? *Procedia-Social and Behavioral Sciences*, *31*, 491–495.

Laxer, C. (2001). Treating computer science as science as: An experiment with sorting (poster session), In *Proceedings of the 6th annual conference on innovation and technology in computer science education*, Canterbury, United Kingdom, Association for Computing Machinery. https://doi.org/10.1145/377435.377710

Lehtinen, E., Hakkarainen, K., Lipponen, L., Rahikainen, M., & Muukkonen, H. (1999). Computer supported collaborative learning: A review. *The JHGI Giesbers reports on education*, *10*, 1999.

Li, C., Dong, Z., Untch, R., Chasteen, M., & Reale, N. (2011). Peerspace-an online collaborative learning environment for computer science students, In *2011 ieee 11th international conference on advanced learning technologies*. IEEE.

Lin, M.-H., Chen, H.-G., & Liu, K.-S. (2017). A study of the effects of digital learning on learning motivation and learning outcome. *Eurasia Journal of Mathematics, Science and Technology Education*, *13*(7), 3553–3564. https://doi.org/10.12973/eurasia.2017.00744a

Lynch, M. (2020). *5 advantages and 5 disadvantages of e-learning*. Retrieved January 15, 2021, from https://www.thetechedvocate.org/5-advantages-and-5-disadvantages-of-e-learning

Macrae, C. (2016). *Sorting algorithms visualised*. Retrieved May 5, 2020, from https://macr.ae/article/sorting-algorithms

Mahy, R., Matthews, P., & Rosenberg, J. (2010). *Traversal using relays around nat (turn)* (RFC No. 5766). RFC Editor. RFC Editor. https://www.rfc-editor.org/rfc/rfc5766.txt

Mirror Networking. (2020). *Mirror networking - open source networking for unity*. Retrieved April 24, 2020, from https://mirror-networking.com/

Mishra, A. D., & Garg, D. (2008). Selection of best sorting algorithm. *International Journal of intelligent information Processing*, *2*(2), 363–368.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J. Á. (2002). Exploring the role of visualization and engagement in computer science education, Aarhus, Denmark, Association for Computing Machinery. https://doi.org/10.1145/960568.782998

O'Malley, C. (2012). *Computer supported collaborative learning* (Vol. 128). Springer Science & Business Media.

Peters, O. (2000). Digital learning environments: New possibilities and opportunities. *The International Review of Research in Open and Distributed Learning*, *1*(1). https://doi.org/10.19173/irrodl.v1i1.3

Pirker, J., Lesjak, I., & Guetl, C. (2017). Maroon vr: A room-scale physics laboratory experience, In *2017 ieee 17th international conference on advanced learning technologies (icalt)*. https://doi.org/10.1109/ICALT.2017.92

Pirker, J. (2013). The virtual teal world-an interactive and collaborative virtual world environment for physics education. *Dr. Diss. Master's thesis, Graz Univ. Technol*.

Pirker, J. (2017). *Immersive and engaging forms of virtual learning* (Doctoral dissertation). Graz University of Technology.

Pirker, J., Holly, M., & Gütl, C. (2020). Room scale virtual reality physics education: Use cases for the classroom, In *2020 6th international conference of the immersive learning research network (ilrn)*. IEEE.

Pirker, J., Riffnaller-Schiefer, M., & Gütl, C. (2014). Motivational active learning: Engaging university students in computer science education, In *Proceedings of the 2014 conference on innovation & technology in computer science education*.

Quacquarelli Symonds. (2020). *Natural sciences*. Retrieved May 11, 2020, from https://www.topuniversities.com/university-rankings/university-subject-rankings/2020/natural-sciences

Roberts, T. S. (2004). *Online collaborative learning: Theory and practice*. IGI Global.

Roberts, T. S. (2005). Computer-supported collaborative learning in higher education.

Rosenberg, J., Mahy, R., Matthews, P., & Wing, D. (2008). *Session traversal utilities for nat (stun)* (RFC No. 5389). RFC Editor. RFC Editor. https://www.rfc-editor.org/rfc/rfc5389.txt

Rosenberg, J., Weinberger, J., Huitema, C., & Mahy, R. (2003). *Stun - simple traversal of user datagram protocol (udp) through network address translators (nats)* (RFC No. 3489). RFC Editor. RFC Editor. https://www.rfc-editor.org/rfc/rfc3489.txt

Rovai, A. P. (2002). Development of an instrument to measure classroom community. *The Internet and higher education*, *5*(3), 197–211.

Sarbazi-Azad, H. (2000). Stupid sort: A new sorting algorithm. *Newsletter (Computing Science Department, Univ. of Glasgow)(599)*, *4*.

Schaffer, R., & Sedgewick, R. (1993). The analysis of heapsort. *Journal of Algorithms*, *15*(1), 76–100.

Schnurr, D. (2017). *Visualizing sorting algorithms in 2d space*. Retrieved May 5, 2020, from https://medium.com/@dschnr/visualizing-sorting-algorithms-in-2d-space-c85dcda72f5c

Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, In *Proceedings first international conference on peer-to-peer computing*. IEEE.

Shell, D. L. (1959). A high-speed sorting procedure. *Communications of the ACM*, *2*(7), 30–32.

Sommerville, I. (2007). *Software engineering*. Addison-Wesley. https://books.google.at/books?id=B7idKfLoH64C

Stagner, A. (2013). *Unity multiplayer games*. Packt Publishing. https://books.google.at/books?id=BWVpAgAAQBAJ

Stewart, D. (2020). *Our most popular scientists – top 100*. Retrieved May 11, 2020, from https://www.famousscientists.org/popular/

Thu, H. T. T., Park, J., Won, Y., & Kim, J. (2014). Combining stun protocol and udp hole punching technique for peer-to-peer communication across network address translation, In *2014 international conference on it convergence and security (icitcs)*. IEEE.

Toptal. (2020). *Sorting algorithms animations*. Retrieved May 5, 2020, from https://www.toptal.com/developers/sorting-algorithms

Unity. (2018). *Connected games: Building real-time multiplayer games with unity and google - unite la*. Retrieved April 21, 2020, from https://www.youtube.com/watch?v=CuQF7hXlVyk

Unity Technologies. (2020). *Performance by default*. Retrieved August 7, 2020, from https://unity.com/dots

Unity Technologies. (2021). *Multiplayer and networking*. Retrieved March 2, 2021, from https://docs.unity3d.com/Manual/UNet.html

University of California. (2020). *The importance of computer science education*. Retrieved February 11, 2021, from https://sites.uci.edu/cs1c/importance-of-computer-science-education/

Warschauer, M. (2007). The paradoxical future of digital learning. *Learning Inquiry*, *1*(1), 41–49.

Welsh, E. T., Wanberg, C. R., Brown, K. G., & Simmering, M. J. (2003). E-learning: Emerging uses, empirical results and future directions. *international Journal of Training and Development*, *7*(4), 245–258.

Wheeler, S. (2012). E-learning and digital learning. In N. M. Seel (Ed.), *Encyclopedia of the sciences of learning* (pp. 1109–1111). Boston, MA, Springer US. https://doi.org/10.1007/978-1-4419-1428-6_431

Wolf, J. P. (2019). *User assessment in 3d environments* (Master's thesis). Graz University of Technology. https://online.tugraz.at/tug_online/wbAbs.showThesis?pThesisNr=66360&pOrgNr=37

Yang, T. (2019). *Upnp-enabled connected devices in the home and unpatched known vulnerabilities*. Retrieved August 11, 2020, from https://blog.trendmicro.com/trendlabs-security-intelligence/upnp-enabled-connected-devices-in-home-unpatched-known-vulnerabilities/

Zapponi, C. (2014). *Sorting*. Retrieved May 5, 2020, from http://sorting.at/