



Buchbauer Benjamin, BSc

Educational Computer Science Visualizations in Virtual Reality

Master's Thesis

Master's degree programme: Softwareengineering and Development

submitted to

Graz University of Technology

Supervisor

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

Co-Supervisor

Gütl Christian, Assoc.Prof. Dipl.-Ing. Dr.techn.

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

Graz, August 2019

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

Education in computer science is an ongoing topic of research nowadays and will continue to be, as adopting computational thinking skills gets more and more important in today's world. Therefore, students need to start learning the fundamentals even earlier. Computer science lectures are structured way better already with most of them focusing on problem-based approaches and group tasks. Furthermore, there is a lot of content freely available online, which can be used by students to learn individually. However, the new generations are easily distracted and need new and innovative forms of education to achieve the best possible success when dealing with such complicated topics. This master's thesis is about the design, implementation and evaluation of simulations of various computer science topics to support learning. One simulation has been implemented in both a web environment and a virtual reality environment. Due to that, it was possible to directly compare both environments with each other and get a better understanding of the effectiveness of the promising attributes of virtual reality regarding education. The evaluation was conducted by letting the participants test both environments and complete a survey with several questionnaires afterwards. The comparison of the results showed that the participants tend to favour the use of the virtual reality environment clearly, mostly because of its high engagement, increased motivation, and deep immersion.

Acknowledgments

First, I would like to express my deep gratitude and honest appreciation to my supervisor Johanna Pirker. She has guided and helped me through the whole process of writing this thesis due to her truly great support, huge patience and valuable expertise. Without her assistance, this work would not have been possible. Therefore, I honestly value her constant efforts to support me.

I would also like to thank Assoc.Prof. Dipl.-Ing. Dr.techn. Aichholzer Oswin for giving me useful advice and feedback about how to design simulations of algorithms and important features. It was a valuable influence on the development process of this work.

Finally, I want to thank my family and friends for their continuous support, encouragement, patience and love throughout my studies and in the process of this thesis. Special thanks go to my girlfriend Karin, for her enormous support and understanding. She always managed to encourage me, and without her, this thesis would not have been possible.

Contents

Abstract	iii
Acknowledgments	iv
1. Introduction	1
1.1. Goals and Objective	2
1.2. Methodology and Structure	2
2. Background and Related Work	5
2.1. Computer Science Education	5
2.1.1. Traditional Computer Science Education	6
2.1.2. Active Learning	7
2.1.3. Just-In-Time Teaching	8
2.2. Digital Learning	10
2.2.1. Videos and Animations	10
2.2.2. Computer Simulations and Visualizations	12
2.2.3. Collaborative E-Learning	15
2.2.4. Serious and Game-Based Education	18
2.2.5. Virtual Reality	20
2.3. Summary	24
3. Design and Requirements	27
3.1. Requirements and Objectives	27
3.1.1. Objectives and Target Group	27
3.1.2. Functional Requirements	28
3.1.3. Non-Functional Requirements	30
3.2. Technologies	31
3.3. Architecture	32
3.4. Summary	34
4. Implementation	35
4.1. Environment and Resources	35
4.2. Sorting	35
4.2.1. Graphical User Interface	36

Contents

4.2.2.	Game Field	38
4.2.3.	Algorithms	39
4.2.4.	Visualization Behaviour	41
4.2.5.	Input Interpreter	44
4.2.6.	Communication	45
4.3.	Pathfinding	45
4.3.1.	Graphical User Interface	46
4.3.2.	Game Field	48
4.3.3.	Algorithms	49
4.3.4.	Visualization Behaviour	50
4.3.5.	Communication	52
4.4.	Binary Search Tree	52
4.4.1.	Graphical User Interface	52
4.4.2.	Game Field	54
4.4.3.	Algorithms	55
4.4.4.	Visualization Behaviour	57
4.4.5.	Communication	58
4.5.	Website	58
4.6.	Sorting in Virtual Reality	58
4.6.1.	Graphical User Interface	60
4.6.2.	Virtual Reality Objects	61
4.7.	Summary	63
5.	Evaluation	65
5.1.	Research Methodology and Procedure	65
5.1.1.	Pre-Questionnaire	65
5.1.2.	Post-Questionnaire	66
5.1.3.	Post-Post-Questionnaire	68
5.2.	Participants	68
5.3.	Results	69
5.3.1.	Sorting in WebGL	69
5.3.2.	Sorting in Virtual Reality	72
5.3.3.	Comparison	75
5.4.	Discussion	75
6.	Lessons Learned	78
6.1.	Literature	78
6.2.	Development	78
6.3.	Evaluation	79

Contents

7. Future Work	81
7.1. Evaluation Results	81
7.2. Further Scenarios	81
8. Conclusion and Discussion	83
A. Computer Science Fundamentals	84
A.1. Essential Theoretical Fields of Computer Science	84
A.1.1. Algorithms	85
A.1.2. Data Structures	85
A.2. Application of Basic Fields of Computer Science	89
A.2.1. Sorting	89
A.2.2. Binary Search Trees	94
A.2.3. Pathfinding	96
Bibliography	100

List of Figures

1.1. Structure of the thesis	3
2.1. Active Learning-based Teaching Model	8
2.2. Just-In-Time Teaching - Step by Step	9
2.3. MergeSort with Folk Dance	11
2.4. How People Learn: Perspectives on Learning Environments	13
2.5. SimSE - Software Engineering Simulation	14
2.6. sorting.at	15
2.7. Sorting Algorithms in Open Wonderland	17
2.8. CodeMonkey	19
2.9. Head mounted display in use	21
2.10. Puzzle game	22
3.1. Architecture model	32
3.2. Example use case of a simulation	33
4.1. Sorting: User interface	36
4.2. Sorting: Provided code functions	38
4.3. Sorting: Algorithm Class Representation	39
4.4. Sorting: SortingVisualItem Class Representation	40
4.5. Sorting: Swapping of two elements	42
4.6. Sorting: Comparison of two elements	42
4.7. Sorting: Group by buckets	43
4.8. Sorting: Subarrays in MergeSort	43
4.9. Sorting: Single element saved in memory	44
4.10. Sorting: Algorithm in coding window	45
4.11. Pathfinding: User interface	46
4.12. Pathfinding: Multiple modified maps	48
4.13. Pathfinding: Class Tile	49
4.14. Pathfinding: Illustration of Executed Algorithms	51
4.15. Pathfinding: Greedy Best First Search Weakness	52
4.16. BST: User interface	54
4.17. BST: Class Node	55
4.18. BST: Structure of the BSTVisualItem Class	57

List of Figures

4.19. Website including <i>WebGL</i> builds	59
4.20. VR: Game Field	60
4.21. VR: Configuration Desk and Algorithm Cubes	61
4.22. VR: Area to Reset Algorithm Cubes	63
5.1. System Usability Scale questionnaire	66
5.2. Results of the Computer Emotion Scale for the Web Environment	69
5.3. Results of the Game Engagement Questionnaire for the Web Environment	70
5.4. Results of the Computer Emotion Scale for the Virtual Reality Environment	73
5.5. Results of the Game Engagement Questionnaire for the Virtual Reality Environment	73
5.6. Comparison of the Computer Emotion Scale between Web and VR	75
5.7. Comparison of the Game Engagement Questionnaire between Web and VR	76
A.1. Euclid's algorithm: Problem description and code implementation	86
A.2. Basic integer array	87
A.3. Graph Representation of a Tree Structure	89
A.4. Basic Sorting Problem	90
A.5. Anatomy of a Binary Search Tree	94
A.6. Deletion of a Node with Two Children	95
A.7. Grid Representation of a Map in Starcraft	97
A.8. Greedy Best First Search - Non Optimal Path	99

List of Listings

1.	Element colour adjustment	39
2.	SortingVisualType Enum	40
3.	SelectionSort code	41
4.	Swapping of two elements	44
5.	Introducing functions with Jint	44
6.	Vector Cross Product for Cleaner Paths	47
7.	Usage of Priority Queue	50
8.	Example Code of the Heuristic Function	50
9.	Priority used by A*	50
10.	Coroutine to visualize pathfinding algorithms	53
11.	Calculation of node positions	55
12.	Search Node Function	56

List of Tables

2.1. Comparison Between Digital Learning Methods	25
A.1. BubbleSort Complexity	90
A.2. SelectionSort Complexity	91
A.3. InsertionSort Complexity	91
A.4. ShellSort Complexity	91
A.5. MergeSort Complexity	92
A.6. QuickSort Complexity	92
A.7. HeapSort Complexity	93
A.8. RadixSort Complexity	93
A.9. GnomeSort Complexity	93

1. Introduction

In today's world, it is even more important to acquire computational thinking skills as early as possible (Horn et al., 2016). Learning computer science topics like algorithms, data structures or programming is a very challenging task (Ben-Ari, 1998). Therefore, it is essential for computer science educators to find ways to maximize students understanding and motivation to improve their learning success (O'Hara & Kay, 2003). Even older studies already show that didactic-based teaching methods, where course content is communicated to all the students, are suboptimal in conjunction with computer science (Jones, 1987). In further consequence, many different models and pedagogic approaches were introduced to support students in the best possible way. Constructivist methods and the problem-based construction of courses, where students are actively involved, show promising results (Hazzan, Lapidot, & Ragonis, 2014). By applying a strategy called Just-In-Time-Teaching, fewer drop-outs in an introductory class were reported (Gavrin, X. Watt, Marrs, & E. Blake, 2004). In more recent years, digital learning has become more popular. Due to the more extensive use of computers and the internet, the availability of computer simulations and visualizations increased dramatically. They became more goal-oriented, and studies show they are more effective than traditional teaching methods (Kathleen Smetana & Bell, 2012). Online tutorials in the form of videos are also a widely used form of learning nowadays. It allows individuals to study a variety of topics, even when underway. Educational games are a more common occurrence as well in recent years. They enable students to learn about topics in an enjoyable environment and are a good way to increase motivation (Kafai, 2001). While there is a decent amount of animations and simulations about computer science topics available, many lack further interactivity which lets users modify the content or even execute self-written code. Furthermore, still very limited resources regarding computer science in virtual reality exist. They show that interacting in such an environment leads to authentic learning and greater excitement (Madathil et al., 2017, 3). A comparison between the same simulation in a web environment and virtual reality can bring new aspects into the matter if virtual reality is a promising education environment for the future.

1.1. Goals and Objective

The main objectives in this thesis are the design, implementation and evaluation of several simulations of computer science topics considering the beforehand studied research lecture regarding computer science education.

- Design and implementation of a simulation that visualizes and compares several sorting algorithms
- Design and implementation of a simulation that visualizes and compares several pathfinding algorithms
- Design and implementation of a simulation that visualizes the standard functions search, add and delete of binary search trees
- A website that contains basic information and grants access to the aforementioned simulations
- A port of the sorting simulation into virtual reality
- Comparison of the same simulation in two different environments

The purpose of this work is to create a centralized collection of simulations that assist students, or people that want to get started in this subject area, in learning computer science. Its goal is to raise motivation and engagement by visually representing theoretical aspects of computer science which can be controlled at user-desired pace.

1.2. Methodology and Structure

The remainder of the thesis is organized in four main parts. The first part discusses the theoretical background and related work of computer science education (Chapter 2) and computer science fundamentals (Appendix A). The second part describes the design and requirements (Chapter 3). The third part focuses on the practical implementation (Chapter 4). The fourth part summarizes an evaluation of the implemented environments (Chapter 5). An overview of the structure and procedure of this thesis can be seen in Figure 1.1.

In Chapter 2, different approaches to computer science education are discussed. On one hand, traditional education methods that evolved to more active and problem-based approaches, and on the other hand, digital learning methods like computer simulations and game-based education, are described, compared, and their pros and cons outlined. Chapter 3 focuses on the practical design of the work. First, functional and non-functional requirements are pointed out, and then different components of the simulations and how they work together are explained. In Chapter 4, details about the implementation of the project are provided. Every single module of the sorting, pathfinding and binary search

1. Introduction

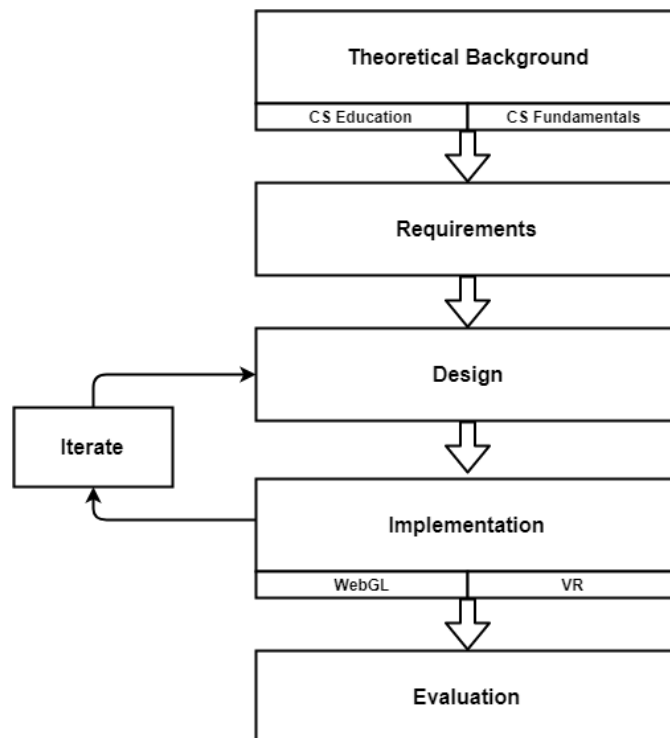


Figure 1.1.: Structure of the thesis: Research on theoretical background as a basis for the requirements, design and implementation

1. Introduction

tree simulation is discussed. Furthermore, it is stated how a website is created, which contains all the simulations and where people can access the content at all times. Following that, the port of the sorting simulation into a virtual reality environment is described. Chapter 5 summarizes the methodology and procedure of the carried out evaluation. Then further details about the participants and results are presented. Chapter 6 outlines the lessons that were learned during the literature research, development and evaluation process of this thesis. Chapter 7 gives a short overview about possible improvements and future work, which has been gathered through the given feedback of the evaluation and the authors own thoughts. In Chapter 8, a conclusion of this work is given. Last but not least, in Appendix A, all the theoretical background about computer science fundamentals is discussed, which has been mandatory knowledge for the design and implementation of this project.

2. Background and Related Work

It is evident that for a lot of students learning computer science is a very difficult task (Ben-Ari, 1998). Putnam, Sleeman, Baxter, and Kuspa (1986) experienced that even basic concepts like variables can be hard to understand for them because they constructed a consistent concept in their mind that happened to be unsuitable. Passive learning methods like lectures and presentations probably won't bring the desired accomplishments because every student's status of knowledge of a particular topic is different, and every student constructs new information in a different way (Ben-Ari, 1998). Nowadays the consensus seems to be that an active learning-based approach is the right way to go, as Silberman (1996) asserts "*Above all, students need to 'do it'—figure things out by themselves, come up with examples, try out skills, and do assignments that depend on the knowledge they already have or must acquire.*" The following sections will focus on giving background information about computer science education, interactive learning methods and simulations, visualizations or game-based tools that help to teach computer science.

2.1. Computer Science Education

Nowadays, we live in a world where having computational thinking skills is more and more important. Therefore, the computer science education community has focused on bringing students near these important topics at an even younger age (Horn et al., 2016). In computer science, educators always look for new input, methods and technologies to win students interests and motivate them. It is desirable to maximize their understanding and find incentives to promote their independent creative work. However, this process is quite challenging because of the amount of different pedagogical approaches and the large scope of computer science (O'Hara & Kay, 2003). Generally, it is important to focus on the principles and identify long-lasting concepts of the corresponding field so that students are endowed with fundamental skills that serve as a solid base for the continuous learning in the rest of their lives (Ghezzi & Mandrioli, 2006). McGettrick et al. (2005) list seven grand challenges that computer science education faces:

2. Background and Related Work

1. Perception: Computer science needs to be promoted so that it leads to a positive public image
2. Innovation: Simpler models of the discipline, and therefore higher quality in courses and broader student interest
3. Competencies: Students need to recognize that the currency and quality of skills in computer science are very important for their whole career
4. Programming issues: Thorough understanding so that programming knowledge and skills can be transferred effectively
5. Formalism: Ensure that students realize the importance of mathematical thinking and vocabulary
6. E-Learning: Establish e-learning as a viable alternative to traditional education approaches
7. Pre-university issues: Enable a smooth transition by providing information and influencing possible computer science students in a positive way

In the following sections, some approaches to computer science education are discussed as well as practical examples and how they deal with some of its challenges.

2.1.1. Traditional Computer Science Education

Computer science education has been an ongoing endeavour throughout almost all of its history. An increasing effort of teaching students in this broad field has happened after the first computer science departments have been introduced (Tucker, 1996). Traditional methods in computer science education include didactic teaching, like a lecture where information is communicated to all the attendees. In this case, the content is processed in batches. However, the broad topic of computer science is process-oriented, which becomes immediately clear to the students after their first encounters with different subjects. Teaching computer science in a process-oriented way where students are actively participating and experimenting is a lot more enjoyable and memorable than the traditional didactic teaching. Methods that include students being interactively involved help them to gain deeper thinking. They are confronted with different viewpoints and can express their positions, which gives them more motivation and makes the topic and the educational process more interesting (Jones, 1987). Research by Carroll, Paine, and Ivancevich (1972) and W. Newstrom (1980) illustrated that role-playing, games and case studies were considerably better than didactic methods like lectures or films when it comes to learning problem-solving skills. Jones (1987) created a list of participatory teaching methods that could be used in computer science classes which included brainstorming, directed dialogues, small group discussions, role-playing, games,

2. Background and Related Work

panel discussions, debates and Socratic dialogues with the favourable belief that they increase student motivation and interest, whereas the fact that interactive methods like that require way more time was a rather disadvantageous side effect.

2.1.2. Active Learning

Lectures should be constructed as teaching models so that it encourages a favourable learning experience of students in a supportive teaching environment. The focus of teaching is on using constructivist methods and active learning (Hazzan et al., 2014). Constructivism is a learning theory that suggests that knowledge is actively constructed by students from their experiences instead of passively acquired from lectures and books. New information is combined with existing ideas, and new cognitive structures are created. It is a recursive process to gain knowledge. Mental structures are developed step by step (Ben-Ari, 1998).

Hazzan et al. (2014) introduced an active-learning-based teaching model consisting of four stages that is shown in Figure 2.1:

1. **Trigger:** In the beginning, a new topic is presented to the students in a nonconventional way, some activity they are unfamiliar with, which enhances reasonable learning and can raise questions and ideas while following the constructivist approach. Newman, Daniels, and Faulkner (2003) propose open-ended group projects as an educational tool. Students are put into a team context and are faced with a problem for which there is no clear answer.
2. **Activity:** Students deal with the trigger individually, in pairs or groups. The duration of this stage can be variable, as it depends on the type of trigger and educational objectives.
3. **Discussion:** At this point, the students come together, and all their products, subjects and ideas from the activity stage are presented and discussed. Due to this, students elaborate on their understanding of concepts, thoughts and attitudes. Students are incited to respond by expressing their opinions and offering constructive criticism.
4. **Summary:** In contrast to the first three phases in this stage, the course instructor is the main actor. Concepts, ideas and other related subjects that were discussed in previous phases are summarized in different forms.

Bouchez-Tichadou (2018) describes how he switched the teaching method of his course from a classical approach to a problem-based approach where students work in groups and solve algorithmic issues. It resulted in him being

2. Background and Related Work

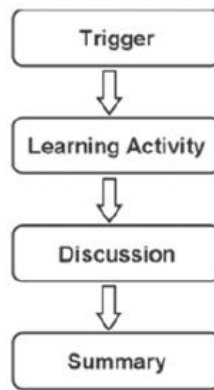


Figure 2.1.: Active Learning-based Teaching Model (Hazzan, Lapidot, & Ragonis, 2014)

more in contact with his students and being able to teach more meaningful content. They felt more responsible, showed more commitment and valued the course higher.

2.1.3. Just-In-Time Teaching

Just-In-Time Teaching (JITT) is a strategy that is founded on several principles of good practice in education (Chickering & Gamson, 1987). The basic idea is that instructors can make adjustments to the lecture on time. The teaching method is applicable to any area of study. Just-In-Time Teaching promotes active learning and ensures that students come well-prepared to class. Furthermore, it helps academic staff to recognize students strengths and weaknesses as well as their learning styles (Gavrin, 2006). A step-by-step overview of JITT can be seen in Figure 2.2. By applying Just-In-Time Teaching Gavrin et al. (2004) reported a 40 per cent decrease in students that received the grade 'D', 'F' or dropped out of an introductory physics class. Marrs, Gavrin, and Novak (2004) stated that it results in measurable cognitive gains and improves students habits in studying. Principles of Just-In-Time Teaching contain the following:

- Merger of high-tech and low-tech elements:
 - High-tech: Web-based platform to convey curricular materials and broaden the communication between students and faculty.
 - Low-tech: General classroom interaction between students, faculty and mentors.
- Quiz-like online exercises before class about conceptual material used in class.
- Making rapid adjustments according to feedback gathered from the web-based platform and classroom.

2. Background and Related Work

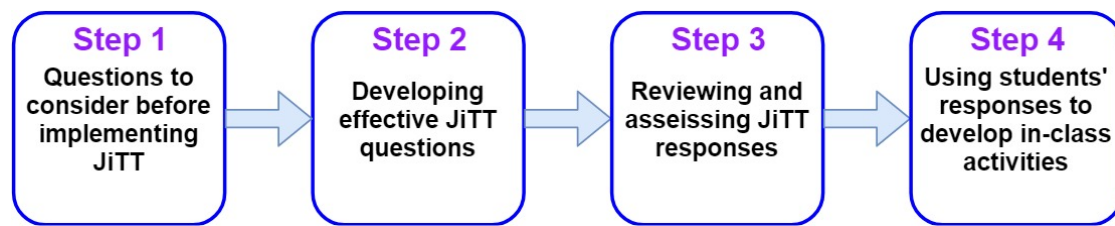


Figure 2.2.: Just-In-Time Teaching - Step by Step (Starting Point Project, 2012)

The advantage is that it encourages students to think about ideas themselves, which they connect to prior knowledge and, in further consequence, apply. Moreover, there is no need to use class time to administer the online quiz. That means there is more time in class to focus on topics students had problems with. Due to their prior examination of the subject, the class is more like a discussion to refine students understanding, it discourages passive note-taking and is learner-centred. Although Just-In-Time Teaching has basic IT infrastructure requirements and needs more time than traditional lectures, it has brought benefits to students education (Novak, Gavrin, & Wolfgang, 1999; Cashman & Eschenbach, 2003; Gavrin, 2006).

Elmaleh and Shankararaman (2017) implemented a flipped classroom model to compare the learning progress of students to a traditional educational model. In the traditional concept, students came to class unprepared and were taught programming theory in lectures. They were also given tasks to complete in class and additionally, homework exercises. Using this approach, many students failed to complete the tasks both in class and at home. In the flipped model students were required to watch video tutorials about the programming concepts and to undergo corresponding quizzes. In class, the students had to solve problems associated with the video tutorials and got live feedback from the instructors. Identically to the traditional approach, they were given additional homework exercises. The results showed that in the flipped model, more students were able to finish class exercises and homework, higher competencies were acquired, and better pass rates in the final exam were achieved. Worth mentioning is that the flipped education model required the students to spend more time on pre-class preparations and slightly less time post-class.

As the importance and relevance of basic computer science knowledge increases, a common occurrence in universities is the blending of major and non-major students in introductory computer science courses. Dawson, Allen, Campbell, and Valair (2018) state that while their course is effective in general, they noticed that non-CS major students had worse results with them mentioning that the workload was too high and the pace too fast. Due to this, a new introductory computer science course has been introduced, especially

2. Background and Related Work

for non-major students. Overall, learning goals were reduced compared to the original course, and the aim was to focus on how programming can be utilized in their corresponding academic subjects. Throughout the course, Just-In-Time Teaching had been used in lectures with students solving worksheets in groups most of the time and short discussions that focus on its key problems. In the end, each student had to design a project where they could freely choose a topic of interest. The results and evaluation of the new course showed that for non-CS major students, it improved their outcomes, attitude, interests and satisfaction compared to the original course. However, the results might have been overestimated because students who participated in the evaluations may have been more engaged than the others.

2.2. Digital Learning

One challenge of learning computer science is the effective usage of information and communication in the 21st century instead of the ineffective and outdated techniques of the previous century (McGettrick et al., 2005). Nowadays, people with computers and fast internet connections have quick and easy access to a vast amount of resources. Digital learning provides big opportunities to study and do research in various fields autonomously (Warschauer, 2007).

2.2.1. Videos and Animations

Animations are sequencing series of pictures in a fast succession, where each image is slightly different than the previous one, which results in a sort of illusion of motion (Wang, 2013). They are usually subject to mainly three types of changes, which include form, position, and inclusion (Lowe, 2003).

Videos are quite similar to animations as they also rely on the fact that the human eye is unable to spot differences between the rapid succession of separate images. Furthermore, video usually includes sound in addition to the pictures and is one of the major distributors of information (Wang, 2013). Studies also show that students have greater learning success through videos that include narrations than without narrations (Wittwer & Renkl, 2008).

Application Scenarios

Due to the longstanding usage of videos and animations, the application scenarios are very wide-ranging. Some examples include chemistry (Russell, Zohdy, Becker, & Russell, 2000), geology (Sangin, Molinari, Dillenbourg, Rebetz, & Bétrancourt, 2006), mathematics (Scheiter, Gerjets, & Catrambone, 2006) or

2. Background and Related Work



Figure 2.3.: MergeSort with Folk Dance (Katai & Toth, 2010)

animated pedagogical agents in general, which are lifelike characters that create learning interactions (Johnson, Rickel, & Lester, 2000).

Use Cases in Computer Science Education

An example use case for computer science is algorithms. The current state of the algorithm gets mapped into an image. Then is animated according to the operations between this state and the next state, to better illustrate its behaviour (Kerren & Stasko, 2002). A very creative and entertaining study by Katai and Toth (2010) investigates how the learning process in computer programming education is accelerated by including dance, rhythm, music and theatrical roleplaying into computer science classes. Their work shows how dance is used as an artistically enhanced multi-sensory learning strategy. Figure 2.3 presents an example of one of their show performances illustrating the sorting algorithm *MergeSort*. Further performances are uploaded to their *YouTube* channel¹ and include computer science topics like *InsertionSort*, *BubbleSort*, *SelectionSort*, *ShellSort*, *QuickSort*, *HeapSort*, *Linear Search*, *Binary Search* and *Backtracking Algorithm*.

Benefits

- Research shows that there is an increase in learning performance for environments where animation and narration are combined (Mayer & Pilegard, 2014).
- Learning opportunities are seemingly endless nowadays because a huge amount of tutorials and courses about various topics can be remotely

¹AlgoRythmics, 2011.

2. Background and Related Work

accessed on platforms like *YouTube*² or *FreeCodeCamp*³ for free.

- Videos and animations lead to more positive learning attitudes and better capability to apply learned principles (Moreno & Ortegado-Layne, 2008).
- Learners report increased motivation and satisfaction (Ainsworth, 2008).

Drawbacks

- Speed and flow of animations can not be controlled (Clark & Mayer, 2007).
- Might influence the way learners perceive the content but not increase their learning results (Swisher, 2007).
- Animations may lead to an illusion of understanding, which then conflicts with successful learning (Ainsworth, 2008).

2.2.2. Computer Simulations and Visualizations

According to Smith (1999), *"simulation is the process of designing a model of a real or imagined system and conducting experiments with that model. The purpose of simulation experiments is to understand the behavior of the system or evaluate strategies for the operation of the system. Assumptions are made about this system and mathematical algorithms and relationships are derived to describe these assumptions - this constitutes a "model" that can reveal how the system works"*. Many simulations additionally include interactive elements. O'Keefe (1987) asserts that *"a Visual Interactive Simulation (VIS) is a simulation which produces a dynamic display of the system model, and allows the user to interact with the running simulation"*. Interactivity is the possibility of the user to influence the outcome of a process or an event. Interactive simulation is a representation of such a process or event, which outcome can be influenced by the user (Smith, 2002). In the last decades, their availability has increased drastically, and they have become more and more powerful. A review of research literature on computer simulations concluded that they are more effective in several points than traditional education methods like lecture-based or textbook-based approaches (Kathleen Smetana & Bell, 2012). Holton (2010) uses the four lenses of the How People Learn framework by Bransford, Brown, and Cocking (2000), as shown in Figure 2.4, to review research on the right moment and right manner to support effective learning by using computer simulations for pedagogical purposes.

²YouTube, 2005.

³FreeCodeCamp, 2014.

2. Background and Related Work

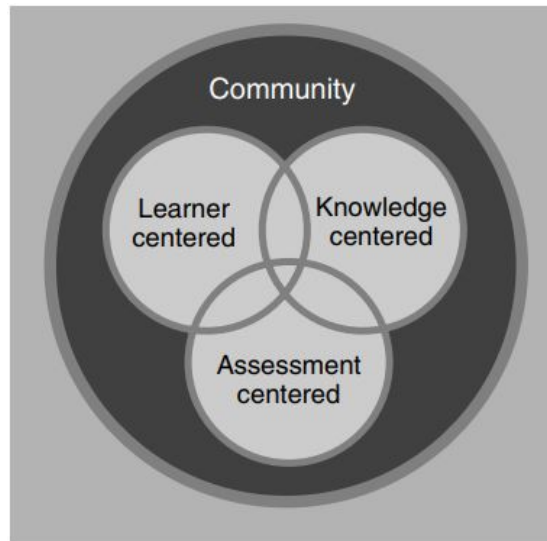


Figure 2.4.: How People Learn: Perspectives on Learning Environments (Bransford, Brown, & Cocking, 2000)

Application Scenarios

Due to the broadening use of this style of education, instructors can choose from a wide range of different computer simulations dependent on the specific subject. They are designed to support understanding and teaching by utilizing interaction with dynamic models and visualizations (de Jong & van Joolingen, 1998). Some examples of real-world applications include manufacturing and material handling systems (Rohrer, 2007), automobile industry (Ulgen & Gunal, 2007), logistics and transportation systems (Manivannan, 2007), health care (McGuire, 2007), and military simulations (Kang & Roland, 2007).

Use Cases in Computer Science Education

Alnoukari (2013) provides details about using simulations in computer science disciplines like computer networking, computer architecture and software engineering. He describes several tools and their usefulness to help the education process. For example SWANS, a scalable wireless network simulator (Barr et al., 2004) or SATSim, a superscalar architecture trace simulator using interactive animations (Wolff & Wills, 2000). Figure 2.5 shows SimSE, which is an educational software engineering simulation environment that allows students to practice a virtual software engineering process (Oh Navarro & van der Hoek, 2004). Sorting attempts to visualize and support to understand how some of the most famous sorting algorithms work, as shown in Figure 2.6 (Zapponi, 2014).

2. Background and Related Work

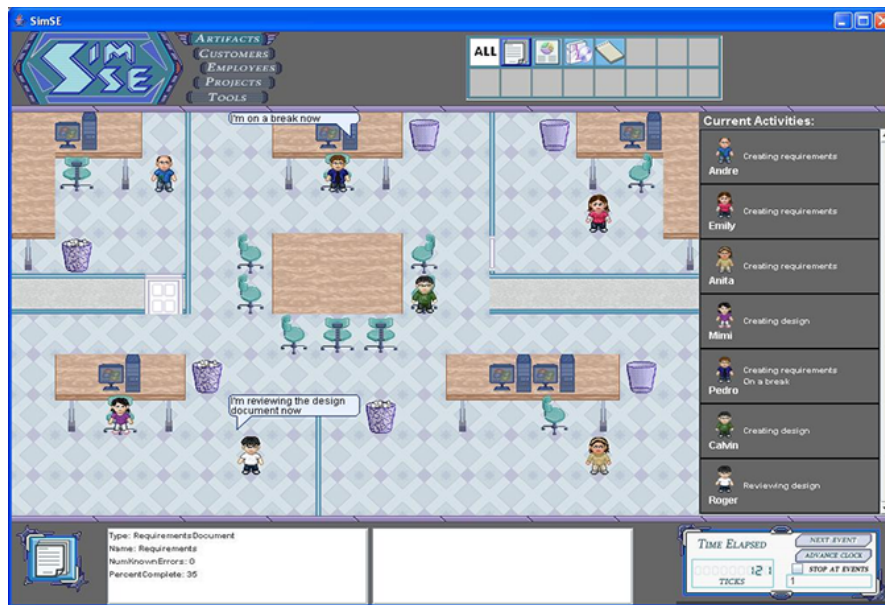


Figure 2.5.: SimSE - Software Engineering Simulation (Oh Navarro & van der Hoek, 2010)

Xueqiao (2011) has created an online demo that visually shows how pathfinding algorithms execute in a 2D environment.

Benefits

- Experimental studies show that learning of topics that students consider to be difficult improves by using integrated computer simulations (Webb, 2005).
- Wenglinsky (1998) finds higher student achievements in classrooms using computers for simulation and data research activities.
- Sarabando, Cravino, and Soares (2014) show that the use of computer simulations can help students to learn basic physic principles.
- It is feasible to simulate complex situations that are not possible to experience in the real world, for example, nuclear explosions (Xiannong, 1998).
- In a real experiment, interesting actions may happen too fast to witness accurately. In a simulation, such a situation can be replayed over and over again.
- Through simulations, it is possible to foresee problems and difficulties but also to make mistakes before introducing the real system (Moorthy, Vincent, & Darzi, 2005).
- Often flexible and easily modifiable.

2. Background and Related Work

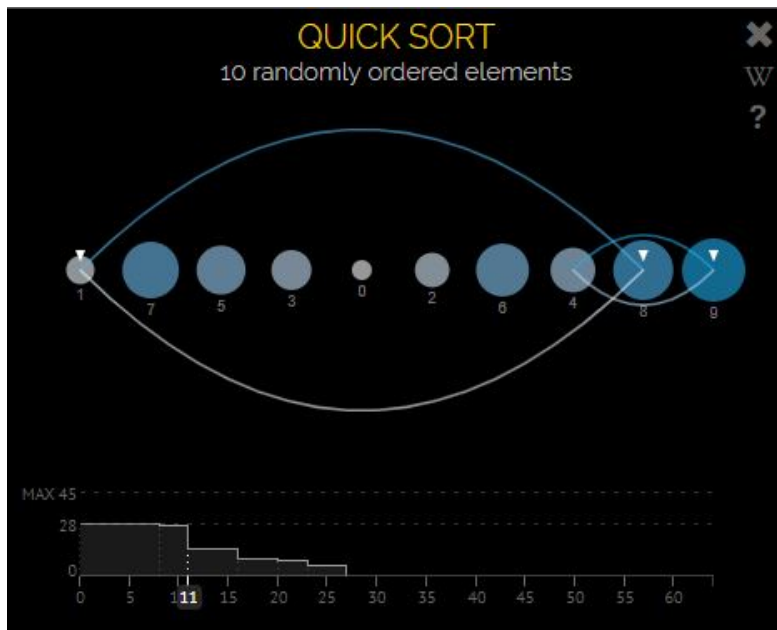


Figure 2.6.: sorting.at (Zapponi, 2014)

Drawbacks

- It may be difficult to interpret the results (Xiannong, 1998).
- Simulations can be inaccurate if there is insufficient real-world data available.
- Creation of simulations can be expensive and time-consuming (Xiannong, 1998).

2.2.3. Collaborative E-Learning

In recent years there has also been a focus on teaching using problem-solving activities combined with peer collaboration. The advantages of this method emerge due to the active character of the learning process, where a deep understanding is required (Dillenbourg, 1999). Pellas (2015) describes it as follows: *"The collaborative e-learning involves two key aspects of learning: (a) the situation of a collaborative process in which the learning process takes place by utilizing various communication forms between students, and (b) the interaction that takes place between group members, such as negotiation or cooperativeness"*. The widespread and sheer infinite possibilities to access the internet nowadays have offered new opportunities to support collaborative peer learning. It introduces tools that offer synchronous interaction and learning among peers (Voyiatzaki, Christakoudis, Margaritis, & Avouris, 2004). Another factor is the increased availability of open-

2. Background and Related Work

source software, which offers enormous benefits. As a result, it was possible to develop software or continue on existing projects that can be accessed by an international community. It resembles a world size laboratory and gives everyone the possibility to gain experience in large-scale software collaboration (O'Hara & Kay, 2003). Platforms that let students cooperate and speak to each other while being physically separated also offer big opportunities. A combination of virtual worlds and instant communication enables students to receive feedback from tutors immediately. Despite being in different physical locations, they can feel the presence of their classmates. Moreover, they can simultaneously view learning materials and take part in group discussions to actively share ideas and offer constructive criticism (Monahan, McArdle, & Bertolotto, 2008).

Application Scenarios

Examples of collaborative platforms like virtual worlds that can be used concerning education include Open Wonderland⁴, Open Simulator⁵ and Second Life⁶. Application scenarios include language learning (Ibáñez et al., 2011), medical and health care (Boulos, Hetherington, & Wheeler, 2007), virtual classrooms, virtual history tours, virtual archaeological tours, communications scenarios, virtual facilities, virtual laboratories, technical training and so on (Clifford, 2012).

Use Cases in Computer Science Education

Johanna Pirker (2013) has created a Virtual TEAL World in Open Wonderland, which is a collaborative and interactive virtual learning environment for students, as can be seen in Figure 2.7, an implemented scenario about sorting algorithms. Other use cases in computer science education include human computer interaction, wireless networking, routing (Miller et al., 2010), computer science principles and programming (Slator, Hill, & Del Val, 2004).

Benefits

- A feeling of immersion and presence in the virtual world (Christian Gütl, 2011).

⁴Open Wonderland, 2007.

⁵Open Simulator, 2007.

⁶Second Life, 2003.

2. Background and Related Work

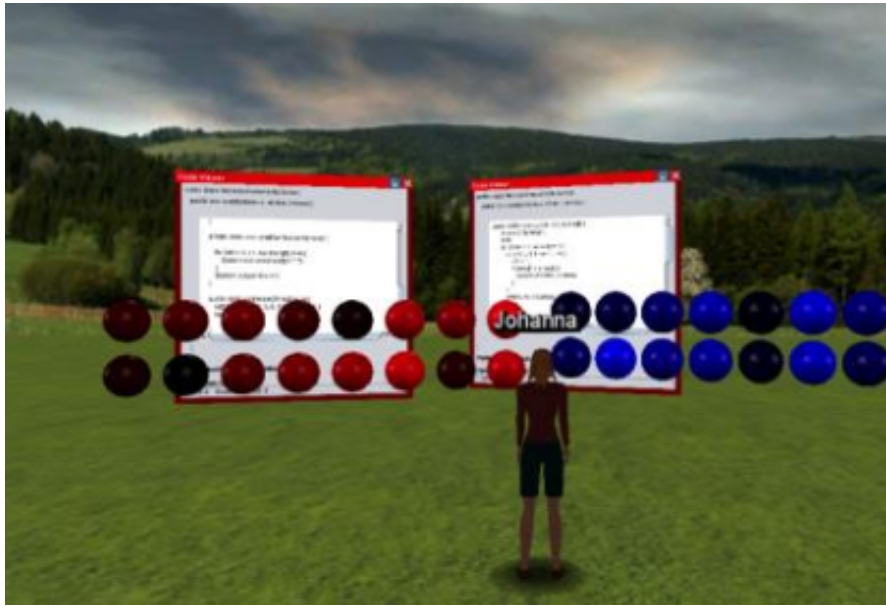


Figure 2.7.: Sorting Algorithms in Open Wonderland (Johanna Pirker & Gütl, 2015)

- Students with their own laptops are more involved in collaborative work, participate in more project-based activities and as a result have significantly better grades (Cengiz Gulek & Demirtas, 2005).
- Collaborative environments like virtual worlds are innovative tools to support exploratory learning and encourage student engagement (Miller et al., 2010).
- Higher social awareness (Christian Gütl, 2011).
- Students can feel the presence of their classmates (Monahan et al., 2008).

Drawbacks

- Good internet connection and a modern computer (C. Gütl & Pirker, 2011).
- Know-how about the interaction within the world which needs to be acquired through tutorial sessions (C. Gütl & Pirker, 2011).
- General awareness about the purpose of the environment (C. Gütl & Pirker, 2011).
- Support for the development, support for content creation and affordable service provision (Miller et al., 2010).

2. Background and Related Work

2.2.4. Serious and Game-Based Education

Game-based learning describes the merger of games and learning processes (Denk, Gabriel, Wernbacher, Pfeiffer, & Mayerhofer, 2018). Such approaches have become an increasingly popular way to raise interest and participation among students (Horn et al., 2016). Another sector of game-based learning originated from the introduction of serious games, which can be described as games with specific rules whose purposes go beyond pure entertainment. Such games are used, for example, in cybersecurity and information assurance for educational objectives (Amorim, Hendrix, Andler, & Gustavsson, 2013). The general views of students regarding games and fun have significant consequences on the learning process. Also, teachers play an essential role by influencing these views (Perry & Ballou, 1997). Computer games have the possibility of being very positive besides their entertainment value. Furthermore, there have been remarkable results when games are designed to tackle a specific issue (Griffiths, 2002).

Application Scenarios

Popular application scenarios include military implementations like Cyber-CIEGE, which has been created by the US Naval Postgraduate School (Irvine, Thompson, & Allen, 2005). Other games teach mathematics, promote strategical and anticipatory thinking in urban management, or deal with training personal behaviour where users are confronted with morally difficult decisions (Denk et al., 2018).

Use Cases in Computer Science Education

Many different game-based educational tools incorporate computer science topics, for example, DBSnap++ which is a block-based programming tool that integrates database queries to encourage the learning of data-driven coding (N. Silva, Nieuwenhuyse, G. Schenk, & Symons, 2018). A project called Iltis provides a web-based system that supports teaching logic in an interactive way where users have to answer questions and immediately get feedback (Geck et al., 2018). Sortko is a mobile learning application about sorting algorithms. It lets the users choose an algorithm and generates a random sequence of numbers. Sorting happens through specific interaction gestures. At the same time, Sortko informs the user about the current status, which makes it easier to progress (Boticki, Barisic, Martin, & Drljevic, 2012). CodeMonkey is a fun and educational game environment that teaches coding in a real programming language called CoffeeScript (Ashkenas, 2009). Users control a little monkey

2. Background and Related Work



Figure 2.8.: CodeMonkey (CodeMonkey Website, 2016)

and help him to catch bananas by writing lines of code while learning basic concepts of programming like objects, function calls, variables, conditions and loops (Schor, Schor, & Pinchover, 2014). A visual representation of the game can be seen in Figure 2.8. Furthermore, game-based tools and serious games appear in cyber security and information assurance (Amorim et al., 2013).

Benefits

- Studies show evidence that the use of computer games in education encourages the teaching and learning process (Pange, 2003).
- In many cases game-based learning increases motivation and gives a better understanding of the subject (Kafai, 2001).
- Interactive simulations and computer games lead to bigger cognitive gains and higher motivation than traditional lecture-based methods (Vogel et al., 2006).
- Replayability is a significant part and advantage of successful educational games, as the players are in a kind of loop where they have to judge situations, act accordingly, and receive feedback on their action (Peddycord-Liu, Cody, Michelle Barnes, F. Lynch, & Rutherford, 2017).
- Strengthens personal endurance and patience (Denk et al., 2018).
- Games improve the imagination and the ability to see things from a different perspective (Denk et al., 2018).
- Positive approach to technology. Educational games are perceived as tools for fun and relaxation (Denk et al., 2018).

2. Background and Related Work

Drawbacks

Despite the obvious benefits of educational games, there are also several challenges and drawbacks. In this context Klopfer, Osterweil, and Salen (2009) list several barriers that educational games encounter:

- **Barriers to adoption:** Curriculum requirements, attitudes towards games, limited time to access computers, support for teachers, different assessment of these skills, evidence of effectiveness, existing social and cultural structures slow down the integration of educational games
- **Barriers to design and development:** Development cost of games, rare collaboration with learning scientists in the development process, limited possibilities to test the game, funding
- **Barriers to sustainability:** Advancement of technology, maintenance and support of the games
- **Barriers to innovation:** Limited data, differences in approaches to pedagogy, limited research, too small ambitions which lead to no scale
- Formal nature of serious games. Even bigger benefits can be gained if these games were released in informal contexts (Le Compte, Elizondo, & Watson, 2015).

2.2.5. Virtual Reality

Virtual reality can be seen as a technology that extends 3D computer graphics and brings in additional devices that are used for more comprehensive handling. It is a virtual environment which a user can interact with and get a sense or feeling of reality (Jayaram, Connacher, & Lyons, 1997). Because the technology of virtual reality evolves quickly and devices get outdated soon and are replaced every few years, LaValle (2016) defines its general concept as following: *"Inducing targeted behavior in an organism by using artificial sensory stimulation, while the organism has little or no awareness of the interference"*. Subsequently LaValle (2016) explains four key components:

1. **Targeted behavior:** Having an experience like walking, flying, swimming and socializing.
2. **Organism:** Could be the users themselves or other life forms like animals or things.
3. **Artificial sensory stimulation:** Ordinary input of some senses is replaced by artificial stimulation.
4. **Awareness:** Organism is unaware of external noises and feels like being present and alive in a virtual world.

2. Background and Related Work



Figure 2.9.: Head mounted display in use (Huguen, 2018)

Usually nowadays head mounted displays like *Oculus Rift*⁷ or *HTC Vive*⁸, as shown in Figure 2.9, are used to give users an immersive feeling of being inside and interacting with the world themselves (Freina & Ott, 2015). Due to the characteristics of virtual reality, such as being able to communicate through far physical locations, being able to collaborate and being able to visualize, it includes elements that make its usage in education promising (Kumar, 2017). Even over 20 years ago, it had been asserted that virtual reality has a big potential to support students in learning complex topics like science (C. Byrne & Furness, 1994). Fernandez (2017) proposes six steps to adopt virtual reality in education. It includes the training of teachers of the subject, a conceptual prototype, the forming of a team, the production, training of the teacher to use the solution and finally, the implementation and use with students.

Application Scenarios

Several learning environments have been developed that simulate realistic settings and make use of the interactive nature of virtual reality (Curcio, Dipace, & Norlund, 2016). For example, CLEV-R, which includes education areas like biology and supports multimedia and instant communication among students (Monahan et al., 2008). U.S. Army soldiers learn about corrosion prevention

⁷Oculus Rift, 2016.

⁸HTC Vive, 2016.

2. Background and Related Work

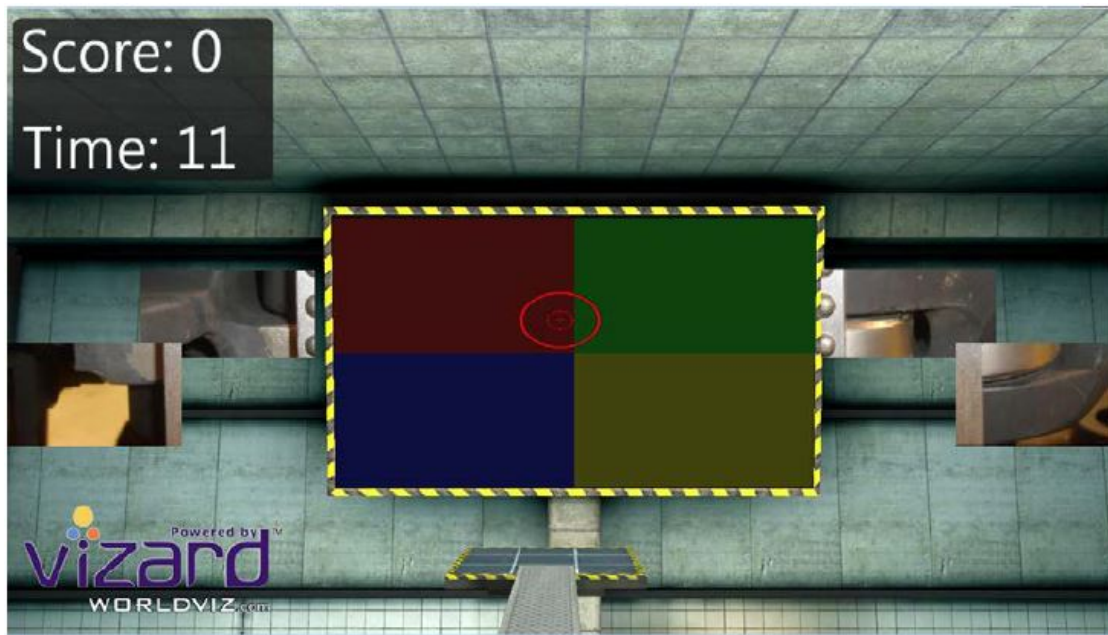


Figure 2.10.: Puzzle game (Webster, 2014)

and control. Figure 2.10 shows a part of it, a puzzle game where users have to put puzzle pieces to the correct position to reveal a type of corrosion (Webster, 2014). Further examples show the use of virtual reality environments in medical applications, astronomy, biology, geography, arithmetic (Curcio et al., 2016), to support the design of architectural spatial experiences (Angulo & Vásquez de Velasco, 2013), or to identify potential safety risks in manufacturing environments (Madathil et al., 2017, 3).

Use Cases in Computer Science Education

Virtual reality environments have been created for several computer science areas like cyber security (Jin, Tu, Kim, Heffron, & White, 2018), programming (Chandramouli, Zahraee, & Winer, 2014) and robotics (Crespo, García, & Quiroz, 2015). Furthermore, Akbulut, Catal, and Yıldız (2018) have developed an interactive teaching environment that is enhanced by virtual reality and shows the functionality of three sorting algorithms. It is a multi-user environment where they connect via Bluetooth or Wi-Fi.

Benefits

Due to the extensive research on the topic and related work, we can conclude several benefits from making use of virtual reality in education.

2. Background and Related Work

- An opportunity to experience, interact and live in situations which would not be possible in real life (Freina & Ott, 2015).
- Increase in critical thinking (Curcio et al., 2016).
- Efficient and enjoyable (Webster, 2014).
- Higher motivation and engagement (Freina & Ott, 2015; Curcio et al., 2016).
- Authentic learning and greater excitement (Madathil et al., 2017, 3).
- Possibility to practice dangerous situations in a secure environment (Freina & Ott, 2015).
- An immersive virtual learning environment is better in terms of learning and produces higher knowledge gains (Webster, 2014).
- Interactivity as part of virtual reality is very important (C. M. Byrne, 1996).
- Angulo and Vásquez de Velasco (2013) show positive learning effects.
- Madathil et al. (2017, 3) show that virtual reality improves the learning experience because of the active engagement of users.
- With further improved technology virtual reality will be effective in teaching (Du & Arya, 2015).
- Better results when used in addition to traditional material (Akbulut et al., 2018).

Drawbacks

On the other hand, various drawbacks need to be considered in virtual reality.

- Excessive use in an individual environment can lead to personal isolation (Fernandez, 2017).
- Mobility limited, and thus the whole virtual reality experience, because devices require cable connections (Abari, 2017).
- Unsatisfying physical comfort (Du & Arya, 2015).
- Requires expensive devices and high processing power (Fernandez, 2017).
- Lack of experience with virtual reality leads to difficulties in maneuvering and therefore slower learning effects compared to more ordinary methods (C. M. Byrne, 1996).
- C. M. Byrne (1996) couldn't show that virtual reality is superior to other teaching methods.
- It is a requirement for the instructors to be familiar with both the present virtual reality simulation and corresponding pedagogical considerations so that positive effects on students learning can be achieved (Vesisenaho et al., 2019).

2. Background and Related Work

2.3. Summary

In this section, background information about various education styles and possibilities were discussed. Learning computer science is a difficult task that needs focused and continuous attention to it. Previously discussed literature has shown that traditional methods to teach computer science, like a didactic approach, don't bring the desired learning effects. Teaching in a problem-based style seems to be more promising, as it encourages students to learn while being actively involved and show more commitment. Furthermore, active learning, paired with enhanced technological tools like computer simulations and visualizations, is more effective than traditional teaching methods and can lead to higher student achievements. Even higher student motivation can be observed when making use of game-based tools to teach specific topics. However, educational games require several considerations like meeting exact curriculum prerequisites and development costs, which means schools or universities can't realize it in a short time frame. Collaborative e-learning can raise the participation of students and in further consequence, their grades. Virtual worlds used for educational purposes made it possible to learn actively and have discussions in groups. However, they require certain minimal computer specifications and a stable internet connection. Ultimately, virtual reality enables new possibilities to experience simulations and learn inside environments that are close to reality. It is enjoyable and exciting but requires rather expensive devices and a little bit of previous knowledge to be able to handle its controls. Table 2.1 shows a final comparison between all the previously discussed methods for digital learning in Section 2.2.

Table 2.1.: Comparison Between Digital Learning Methods

	Videos	Simulations	Coll. E-Learning	Games	VR
Definition	Rapid succession of separate images including sound (Wang, 2013)	An exact model of a real system that is used for experiments (Smith, 1999)	Utilizing various communication forms and interaction between students (Pellas, 2015)	Merger of games and learning processes (Denk, Gabriel, Werbacher, Pfeiffer, & Mayerhofer, 2018)	Virtual environment users can interact with to get a feeling of reality (Jayaram, Connacher, & Lyons, 1997)
Applications	Chemistry, geology, mathematics, pedagogical agents, ... (Russell, Zohdy, Becker, & Russell, 2000; Sangin, Molinari, Dillenbourg, Rebetez, & Bétrancourt, 2006; Scheiter, Gerjets, & Catrambone, 2006; Johnson, Rickel, & Lester, 2000)	Manufacturing and material handling systems, automobile industry, logistics and transportation systems, health care, military (Rohrer, 2007; Ulgen & Gunal, 2007; Manivannan, 2007; McGuire, 2007; Kang & Roland, 2007)	Virtual worlds that include language learning, medical and health care, classrooms, facilities, laboratories, ... (Ibáñez et al., 2011; Boulos, Hetherington, & Wheeler, 2007; Clifford, 2012)	Military, mathematics, urban management, adult training, ... (Irvine, Thompson, & Allen, 2005; Denk, Gabriel, Werbacher, Pfeiffer, & Mayerhofer, 2018)	Biology, military, medical applications, astronomy, biology, geography, architecture, manufacturing, ... (Monahan, McArdle, & Bertolotto, 2008; Webster, 2014; Curcio, Dipace, & Norlund, 2016; Angulo & Vásquez de Velasco, 2013; Madathil et al., 2017, 3)
Use Cases CSE	Vast amount of Videos and animations of different computer science areas available (Kerren & Stasko, 2002; Katai & Toth, 2010)	Computer networking, architecture, software engineering, CS principles (Alnoukari, 2013; Zapponi, 2014)	Algorithms, human computer interaction, networking, programming (Johanna Pirker, 2013; Miller et al., 2010; Slator, Hill, & Del Val, 2004)	Programming, logic, algorithms, cyber security, information assurance (N. Silva, Nieuwenhuys, G. Schenk, & Symons, 2018; Geck et al., 2018; Amorim, Hendrix, Andler, & Gustavsson, 2013)	Cyber security, programming, algorithms (Jin, Tu, Kim, Heffron, & White, 2018; Chandramouli, Zahraee, & Winer, 2014; Akbulut, Catal, & Yildiz, 2018)
Benefits	Increase in learning performance and motivation, seemingly endless free online learning opportunities (Mayer & Pilegard, 2014; Ainsworth, 2008)	Learning can be improved, simulation of complex situations, can be replayed over and over again, foreseeable problems and difficulties (Webb, 2005; Xiannong, 1998; Moorthy, Vincent, & Darzi, 2005)	Feeling of immersion and presence, supports exploratory learning and encourages student engagement, higher social awareness (Christian Gütl, 2011; Miller et al., 2010)	Encourages the learning process, increases motivation, bigger cognitive gains, strengthens endurance and patience (Pange, 2003; Kafai, 2001; Vogel et al., 2006; Denk, Gabriel, Werbacher, Pfeiffer, & Mayerhofer, 2018)	Experience situations which would not be possible in real life, excitement, higher motivation and engagement, improves learning experience (Freina & Ott, 2015; Madathil et al., 2017, 3; Curcio, Dipace, & Norlund, 2016)
Drawbacks	May only influence perception, illusion of understanding, speed of animations uncontrollable (Clark & Mayer, 2007; Swisher, 2007; Ainsworth, 2008)	Difficult to interpret results, Creation can be expensive and time consuming (Xiannong, 1998)	Good connection, modern computer, tutorials, support for content creation, general awareness about the purpose (C. Gütl & Pirker, 2011; Miller et al., 2010)	Barriers to adoption, design and development, sustainability, and innovation (Klopfer, Osterweil, & Salen, 2009)	Limited mobility, modern computer, instructors need to be well prepared (Abari, 2017; Fernandez, 2017; Vesisenaho et al., 2019)

2. Background and Related Work

Due to the ever-rising demand of trained computer scientists and the fact that students struggle in learning its fundamental theory, we want to create tools which let users interactively engage the topics and increase the learning effects and motivation. Because of that, we focus on an approach to visualize several computer science topics and furthermore, extend one part to dive into the increasingly popular area of virtual reality. The next chapter will demonstrate the design used for this procedure.

3. Design and Requirements

In Appendix A, a general overview of fundamentals in computer science - algorithms and data structures - as well as several specific application areas - sorting algorithms, pathfinding and binary search trees - are introduced as a groundwork of this thesis. Furthermore, many different approaches to education, mostly related to computer science, were discussed in the previous chapter to get a better understanding of important factors that need to be considered when designing computer science topics for teaching.

3.1. Requirements and Objectives

In this section, the main objectives of this work as well as different requirements to meet our intended goals, based on the thorough research of various teaching approaches, are presented.

3.1.1. Objectives and Target Group

After analyzing the previously discussed education methods in combination to computer science topics which are relevant to us, and considering the proven effectiveness of computer simulations from Kathleen Smetana and Bell (2012), as well as several drawbacks of different approaches stated by Klopfer et al. (2009), our main objectives can be determined as follows:

- Design of visualizations of several specific computer science topics in *Unity3D*¹ that support students in learning their underlying concepts
- Utilization of a base user interface so that it's possible to gain control and move the simulations at will
- Creation of a web platform that is easily accessible at all times and contains an overview and explanation of implemented computer science fields and their simulations
- Port one visualization into a virtual reality environment
- Comparison of the same simulation in two different environments

¹Unity3D, 2005.

3. Design and Requirements

Our target groups are students in the beginning terms, pupils in upper classes that have chosen computer science as their main field and also all the people that want to start learning about CS and programming or want to improve their computational thinking skills. However, it can also be useful for graduates as well as teachers to refresh or restore their knowledge in the presented topics quickly. The visualizations are developed to demonstrate the functionality and behaviour of its underlying algorithms and concepts. It should be easily possible to understand the way of execution and to interact with the user interface to browse through all the available options.

3.1.2. Functional Requirements

Our functional requirements have been designed based on the research of educational methods in computer science, described in Section 2.1, and having considered our main target groups. Therefore, the following functional requirements can be concluded for the different visualizations:

Overall

- At all times in the visualizations we want the user to be able to have full control on what happens next:
 - The ability to pause and resume the visualization
 - The ability to move one step forward or backwards
 - The ability to move to the end or to the beginning
- It should be possible to modify the speed of the visualizations so that users can study at their own pace

Sorting Algorithms

- Spawn a set of random numbered and sized elements in a movable container
- Option to choose the amount of elements in a set (inside a certain supported range)
- Assigning sorting algorithms by drag and dropping over an element container
- Visualization of one or more containers with assigned sorting algorithms concurrently
- Self organization of element sets at runtime so that different scenarios can be simulated (e.g. best-case and worst-case scenarios)

3. Design and Requirements

- Operation counter on each element container so that it is possible to compare the performance of different algorithms
- Introduction of commands so that it is possible to interact with and swap elements
- Interpretation of self-created code that gets executed while the program is running, so that even basic algorithms can be realized

Pathfinding

- Automatic spawn of a map that contains a start and end point in its initial state
- It should be possible to customize and edit the map so that different scenarios and circumstances can be simulated
- Visual representation of the execution of implemented algorithms
- Display of the provisional steps needed at each visited tile
- Ability to spawn multiple maps where the customized content of former maps is copied
- Concurrent execution and visualization of assigned pathfinding algorithms
- Summary about the resulting path lengths of the executed algorithms
- Ability to toggle between various configuration options to only visualize the resulting path, to calculate the vector cross product, or to allow diagonal steps
- Possibility to modify movement costs for walkable tile types on the map

Binary Search Tree

- Spawn of a random binary search tree data structure in form of circles
- Perform the basic operations on a binary search tree:
 - Add a node by key
 - Search for a node by key
 - Delete a node by key
- Log window that is filled with messages explaining the ongoing activity presented in each step

Website

- Overview and short description of the implemented computer science topics

3. Design and Requirements

- Link to WebGL² builds of the visualizations so that users can run them directly in their browser

Sorting in Virtual Reality

- Option to choose the number of elements in a supported range
- Option to choose the order in which elements are spawned (random, nearly sorted, reversed)
- Representation of algorithms in form of cubes that can be grabbed, moved, and thrown
- If thrown at a specific area, cubes spawn a set of elements above them
- If previously spawned cubes get thrown at another specific area, they despawn and reset to their original positions
- Panel where the code of sorting algorithms is shown
- Controller to scroll between shown algorithm code
- Possibility to run one or more algorithms at the same time
- Panel where the current number operations of each running algorithm is shown

3.1.3. Non-Functional Requirements

In addition to functional requirements, which were listed in the previous section, non-functional requirements also play an important role when developing software. Some relevant factors to this work are the following:

- **Availability:** The simulation environment should be permanently available to users. By having created a website that is accessible without any real difficulties or technical requirements at all, it is assured that our work is available almost all the time from almost everywhere by simply having an internet connection.
- **Usability:** One of the most important aspects of this work is that our simulations are easy to use and simple to understand. Combined with a visually appealing and user-friendly interface, it is the key to avoiding frustration, raising motivation and supporting the user in learning the presented topics.
- **Scalability:** The system needs to be capable to remain stable when increasing the number of objects that are used in the simulations. In our case the number of containers and elements in the sorting project, the number of sections of a map in the pathfinding project or the number of nodes in a binary search tree.

²WebGL, 2011.

3. Design and Requirements

- **Extensibility:** It should be possible to extend our simulations with additional content like slightly modifying algorithms by changing criteria in the code or even adding completely new algorithms.
- **Reusability:** Parts of the system are easily reusable if we decide to implement another visualization of a different computer science topic.
- **Performance:** All the visualizations in the environment should run fluidly so that every single operation can be executed without interruption or lag. Performance hindering configurations, like, for example, spawning many maps in the pathfinding project with each having a very big amount of customizable map parts, are disabled by default.
- **Multi-platform:** It is desirable to reach out to as many users as possible. We want to be able to easily deploy the projects to several different platforms to make them available to almost everyone.

3.2. Technologies

After having determined the main objectives and the requirements of this work, the next step was to choose a proper framework/engine and programming language for the implementation. There were several candidates, but in the end, the decision fell between two of them, which are briefly discussed in this section.

- **HTML5 & JavaScript³:** One idea was to create and implement everything on a website using HTML5 and JavaScript. It offers many different libraries that are free to use and can be utilized to construct visualizations of all kind. It is simple, easy to learn and very popular.
- **Unity3D⁴:** Unity is a popular game engine that supports developing games in both 2D and 3D. One of the most practical benefits is its ability to export projects to many different platforms. It is easy to learn and provides many features. Unity is free for personal use if the funding does not exceed 100K dollars per year. However, there are also paid versions for professionals and studios that offer access to enhanced benefits. Furthermore, everyone can make use of the Unity asset store, where they offer a large collection of both free and paid assets that can be quickly integrated into projects. The supported languages of Unity are C# and, in earlier versions, UnityScript and Boo.

Due to the author's previous experience in Unity3D and especially C#, as well as having the aforementioned non-functional requirement *multi-platform* in mind,

³JavaScript, 1995.

⁴Unity3D, 2005.

3. Design and Requirements

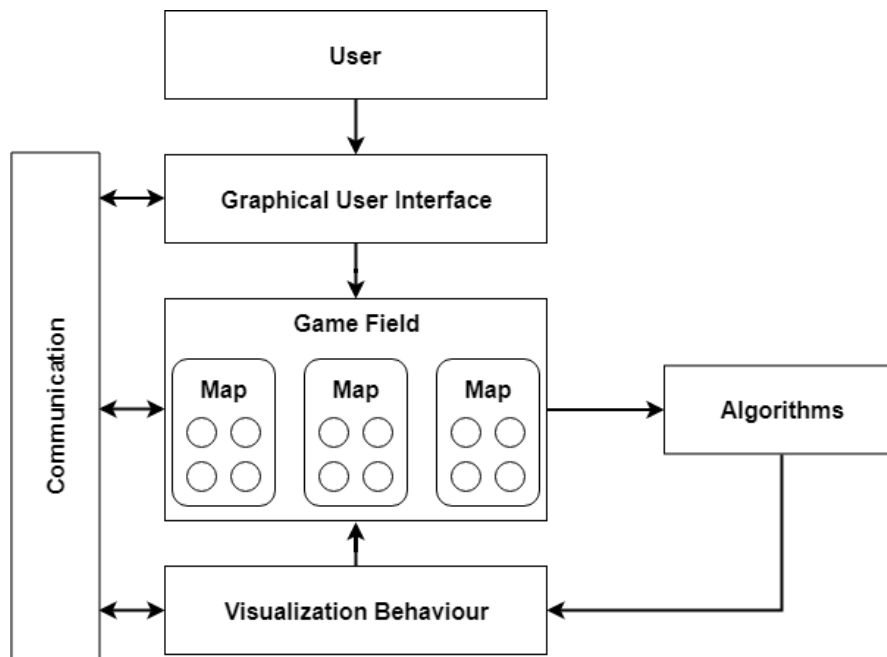


Figure 3.1.: Architecture model of the project show its components

the Unity engine will be used as a primary tool to create computer science visualizations. The obvious choice for the integrated development environment is VisualStudio⁵ as it offers a well-done integration to Unity, which is very useful for testing and debugging when facing various problems or errors, and again also due to the author being familiar with it.

3.3. Architecture

Based on the main objectives, as well as the functional and non-functional requirements, an architecture model is created to represent the different modules this project consists of. All the components are built from scratch and created in Unity3D. Figure 3.1 shows the concept of architecture containing the different modules. The components are implemented in every part of this project with slight modifications depending on the type and its specific distinctive requirements.

- **Graphical User Interface:** This module is responsible for receiving and delegating user input like spawning or customizing objects in the game field and executing algorithms. Its design mostly differs between each

⁵Visual Studio, 1997.

3. Design and Requirements

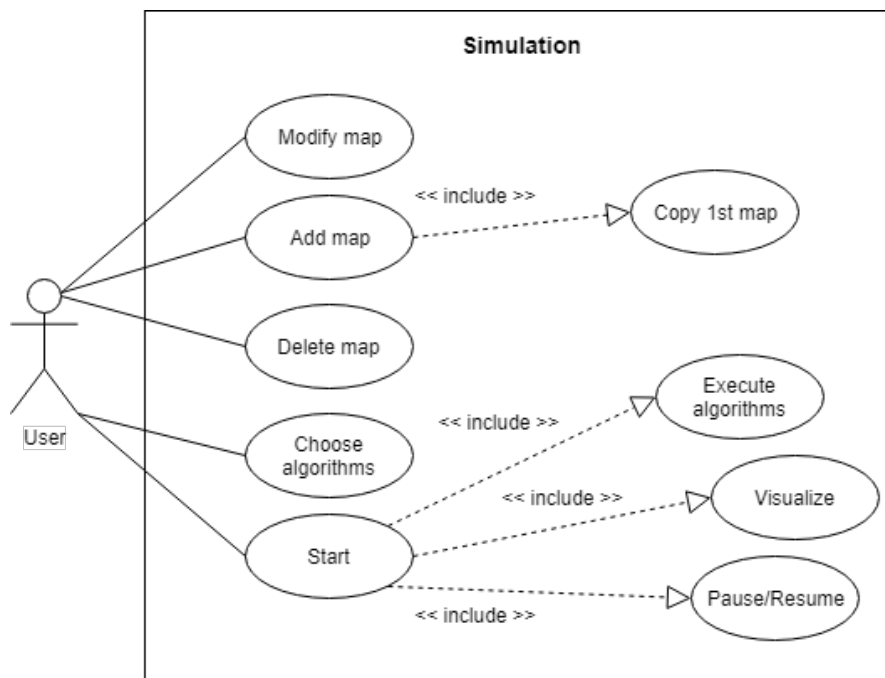


Figure 3.2.: Example use case of a simulation

computer science simulation except for one part that is responsible for controlling the visualizations.

- **Game Field:** The game field consists of one or more maps, depending on the actions of the users. It contains all the necessary information for further operations.
- **Algorithms:** This part is responsible for executing chosen algorithms on the given dataset. The result and relevant data, about its underlying functionality, is being transported to the *Visualization Behaviour* module.
- **Visualization Behaviour:** This module is the core component of the project as it is responsible for interpreting and visualizing the data given by the *Algorithms* module after an algorithm is executed.
- **Communication:** A key part is being able to have full control over the simulations. With this component, it is possible to communicate different game states between the modules to ensure proper behaviour.

Figure 3.2 shows an example use case of the simulations. The game field can be modified, parts can be added or removed, and algorithms can be chosen. Following that, the simulation can be started, which triggers the execution of algorithms on our data structure and visualization. Furthermore, it can be stopped, moved forward, moved backwards, and resumed.

3.4. Summary

In this chapter, we discussed the objectives and target group of this work, the derived requirements for all the simulations, as well as the overall architecture and design of the project. The objectives that we want to realise were decided after extensively studying related literature about both education methods in general and specifically related to computer science. The observations by Kathleen Smetana and Bell (2012) pointed the author into the selected direction. The functional and non-functional requirements were found after engaging related computer simulations mentioned in section 2.1 and considering certain drawbacks stated by Klopfer et al. (2009). With all these important factors in mind, the overall architecture for this project was created, as shown in Figure 3.1. It shows the main components which are needed for the successful implementation. It was decided to use the Unity engine⁶ for implementing our goals as it is easy to use and able to deploy our simulations to multiple platforms. Furthermore, it grants access to the Unity asset store⁷, which contains many free assets, some of them useful for our work, making it an obvious choice.

In the course of this work, several computer science topics will be implemented and visualized.

1. **Sorting:** Visualization of many sorting algorithms. Users have the possibility of comparing algorithms that run simultaneously. They can also move elements and run code themselves.
2. **Pathfinding:** Visualization of many pathfinding algorithms. Users can customize maps and run different scenarios.
3. **Binary Search Tree:** Visualization of a binary search tree data structure. Users can apply basic operations.

Important aspects of this work are its usefulness as a supportive learning tool, the ability of the user of having full control of the visualization, as well as the possibility to extend the simulations by further algorithms or options easily. The next chapter will be about the detailed description of implementing all the underlying components and the methods used to accomplish our objectives.

⁶Unity3D, 2005.

⁷Unity Asset Store, 2010.

4. Implementation

In this chapter, the methods to create the simulations are discussed in detail, screenshots of certain project parts are shown, and code snippets of important features are listed.

4.1. Environment and Resources

As previously mentioned in section 3.2, the engine used for creating this project is Unity3D¹ with Visual Studio² as integrated development environment, and the programming is done in C#. Furthermore, we make use of plugins and resources obtained from the Unity asset store³. For the movement of game objects, an asset called LeanTween⁴ is utilized. It is an efficient animation engine for Unity that lets you easily move objects to new positions. Due to the requirement of letting the user actively participate in the simulations, we have several user interface elements. Instead of the built-in Unity elements, we make use of the more powerful TextMeshPro⁵, which was an asset first but has been included in Unity since version 2018.2, for some parts. Last but not least, we want to be able to execute code at runtime for the Sorting simulation. To do that we include a plugin called Unity-Jint⁶, which is a port of Jint⁷, a Javascript⁸ Interpreter for .NET framework, for Unity.

4.2. Sorting

In this section, all the implementation details about the underlying components, the featured sorting algorithms, and the style of visualization of the Sorting

¹Unity3D, 2005.

²Visual Studio, 1997.

³Unity Asset Store, 2010.

⁴LeanTween, 2016.

⁵TextMesh Pro, 2017.

⁶Unity-Jint, 2016.

⁷Jint, 2014.

⁸JavaScript, 1995.

4. Implementation

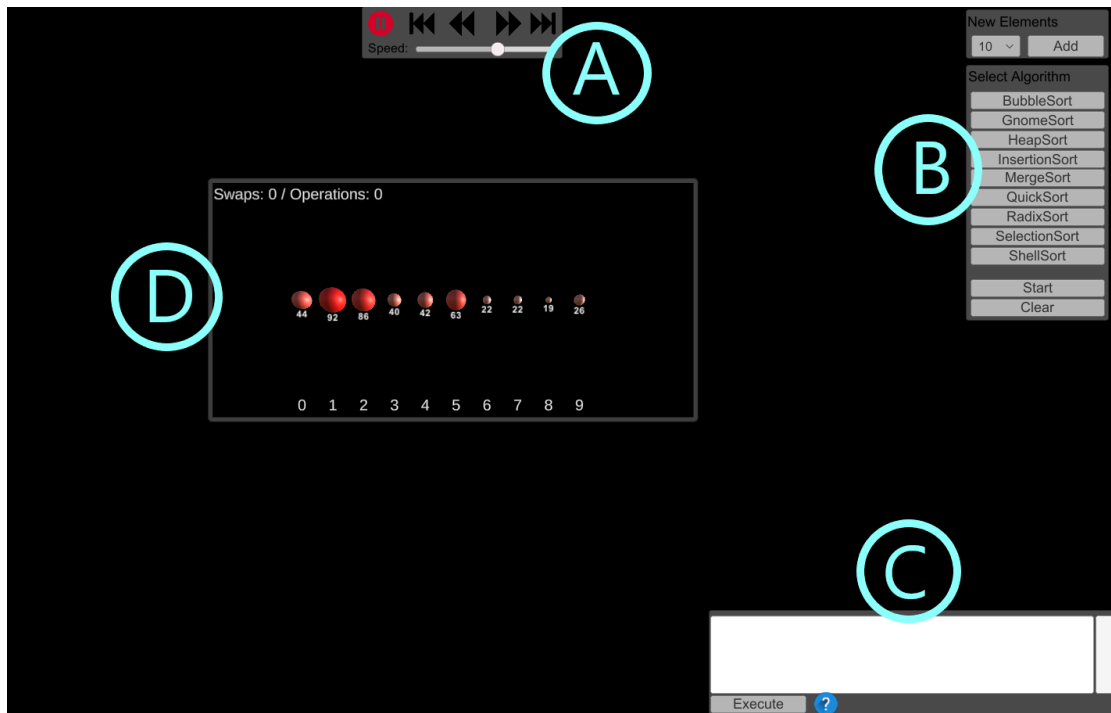


Figure 4.1.: User interface

project are discussed. An overview of the user interface of the Sorting simulation is represented in Figure 4.1.

4.2.1. Graphical User Interface

In this simulation, the user interface is divided into three parts, with each responsible for a different task.

1. **Controller:** The controller is the main user interface component of all our simulations. When visualizing algorithms and other computer science processes, it is important to give the users full control over the ongoing events. They must have the ability to stop and resume the visualization at will and move single steps forward or backwards so that they can learn and understand the presented concept at their own pace. As shown in Figure 4.1, sector A, the UI Controller consists of six options to intervene in the visualization.
 - a) Pause/Resume: lets the user stop and resume the ongoing visualization whenever needed
 - b) Step to begin: brings the simulation back to the begin game state

4. Implementation

- c) Step backwards: moves the visualization one step backwards
 - d) Step forwards: moves the visualization one step forward
 - e) Step to end: brings the simulation to the end game state
 - f) Speed: the user is able to control the speed of the visualization
2. **Main:** This is the main decision point of the user interface in this simulation. The corresponding options of this part are presented in Figure 4.1, sector B.
- New elements: On top, the user can choose a number in the predefined range from five to twenty. The *Add* button then spawns a new box that contains the previously chosen number of elements. If there is already an existing element container with the same number of elements in the game field, which will be described in the next section, then the newly spawned elements will be in the same randomly arranged order as the other container. On the other hand, if there is no other element container present or the chosen number of elements differs from the existing container, then a simple randomize function is called to determine the attached value of each element.
 - Select algorithm: Here the user can choose between several implemented sorting algorithms. By simply drag and dropping an algorithm over an element container, it gets attached to it. Algorithms can be applied to multiple containers. Pressing the *Start* button leads to the beginning of the visualization process of every single element container, that has an algorithm attached, at the same time. Hitting the *Clear* button removes all element containers that do not have a running visualization.
3. **Coding area:** In addition to the global architecture, the sorting simulation implements another module, the *Input Interpreter*, which will be discussed in a later subsection. It is responsible for interpreting user-entered code and getting it executed on an element container. Figure 4.1, sector C, shows the components of this part of the user interface. Arbitrary code can be entered into the big input field. Furthermore, some provided functions can be made use of, as shown in Figure 4.2. The panel that lists the functions can be displayed to the user by clicking on the blue question mark. By clicking on the *Execute* button, the entered code is executed on the selected element containers, which is recognizable by the differently coloured border.

After having explained our user interface component in detail and having given a rough overview of its appearance, we move over to the *Game Field* module and discuss it in the next subsection.

4. Implementation

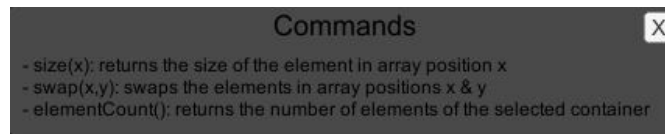


Figure 4.2.: Provided code functions

4.2.2. Game Field

The *Game Field* is the central point of the simulation, especially for the user, because almost every relevant or important action is taking place there and is visible to the user. As shown in Figure 4.1, sector D, it is filled by one or more containers with circular elements inside. A so-called *SortingBox* is stored as a *Prefab*, which is a complete game object with all its components, property values and child game objects. It is reusable and can be instantiated inside the simulation whenever needed. The *SortingBox* prefab consists of several components, a container, which holds the elements that get sorted, some text-based objects and scripts. The container is selectable by clicking on it, which is immediately recognizable by the users as it changes the colour of the border. Furthermore, containers are targets of drag and drop behaviour. Sorting algorithms can get attached to it. Their names get then shown in the top right corner. In the top left corner, the needed number of operations of the executed algorithm is displayed. The circular elements are spawned in the middle. They are the main objects of this simulation because they represent the animations of sorting algorithms. These elements have the following attributes:

- **Number:** A random number between 1 and 99 is assigned to each newly spawned element.
- **Colour:** The default colour of the elements is red. Depending on the resulting value the colour of the element is adjusted to represent the differences, as can be seen in Listing 1.
- **Size:** Similar to the colour attribute, the size of an element changes according to the number. There is a predefined value for both the minimum scale and the maximum scale.

The *SortingBox* has a script attached that is responsible for handling all the attribute values. Below the elements, we have text elements with numbers from zero to nine that represent the corresponding position in an array. That position is useful when using the *Coding Area*, which will be discussed in a later subsection. Last but not least, we have optional text elements on the left side. There is a *BucketScript* attached that is called when RadixSort is applied and started. It is responsible for displaying the different buckets. There, elements get grouped according to their digits.

4. Implementation

```
float multiplier = (255 - minGBColor) / maxScale;
float color = 1 - ((minGBColor + scale * multiplier) / 255);
...
elementTransform.GetComponent<Renderer> ().material.color =
    new Color (1, color, color);
```

Listing 1: Element colour adjustment

Algorithm
+ name: String
+ elementArray: GameObject[]
+ visualItems: List<SortingVisualItem>
+ GetElementSize(GameObject): int
+ Swap(int,int): void
+ Compare(GameObject,GameObject): void

Figure 4.3.: Algorithm Class Representation

4.2.3. Algorithms

In this subsection, the implemented sorting algorithms, and how we transport the data so that it is possible to visualize them in a way that is understandable and appealing, is discussed.

As most of the sorting algorithms behave in a similar way, an abstract base class is created, which can be seen in Figure 4.3. In the derived sorting algorithm classes, on one hand, the given element array is sorted and on the other hand, useful information about the way the specific algorithm behaves is saved in objects of a class called *SortingVisualItem*, which is presented in Figure 4.4. It contains all the necessary data to comprehend the processes at a later stage. An enum is used to differentiate between several types of *SortingVisualItems*, as can be seen in Listing 2. It is an enumerated type, with a restricted set of values, that is utilised to classify objects. All the obtained information at this point is crucial for visualizing each algorithm.

Due to the different characteristics of sorting algorithms, it is necessary to divide them into categories which lead to a similar style of visualization:

- **In-place comparison based:** In-place means that the algorithm does not need extra memory space in the sorting process. However, for variables, there is a small constant extra space allowed. Comparison based means

4. Implementation

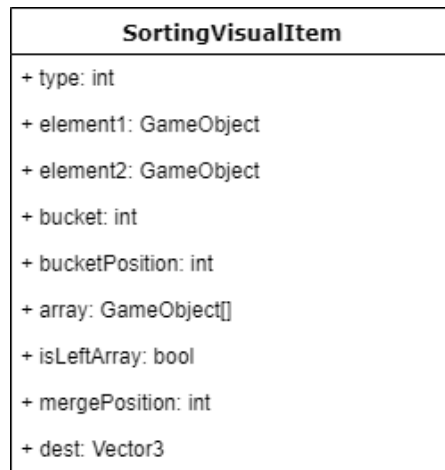


Figure 4.4.: SortingVisualItem Class Representation

```
enum SortingVisualType { Swap = 0, Comparison = 1, Radix = 2,
    MergeArray = 3, MergeMove = 4, MergeComparison = 5,
    MoveTo = 6, MoveMemory = 7 }
```

Listing 2: SortingVisualType Enum

that the algorithm simply compares the values of the elements in the array to each other. This type of sorting algorithm is featured the most in our simulation, as the following algorithms belong to this category:

- BubbleSort
- GnomeSort
- HeapSort
- InsertionSort
- QuickSort
- SelectionSort
- ShellSort

This is the most simple form of algorithms to visualize as the elements are only compared to each other, swapped, or moved to another position. Listing 3 shows the implementation of SelectionSort, how the algorithm sorts the elements, and which data we save for the visualization.

- **Out-of-place comparison based:** In this case, the algorithm also compares the elements to each other but needs extra memory space in the process. MergeSort falls into this category. It recursively splits the unsorted array into new smaller arrays that get sorted one after another. Therefore, we save information about the sub-arrays to be able to distinguish it from the

4. Implementation

```
void SelectionSort()
{
    for (int i = 0; i < elementArray.Length - 1; i++)
    {
        // Find the min element in the unsorted array
        int min_idx = i;
        for (int j = i + 1; j < elementArray.Length; j++)
        {
            visualItems.Add(new SortingVisualItem((int)SortingVisualType.Comparison,
            elementArray[j], elementArray[min_idx]));
            if (GetElementSize(elementArray[j]) < GetElementSize(elementArray[min_idx]))
                min_idx = j;
        }
        // Swap the found min element with the first element
        Swap(min_idx, i);
    }
}
```

Listing 3: SelectionSort code

other elements in the visualization.

- **Non-comparison based:** These algorithms don't compare the elements to each other but use other ways like RadixSort, which categorizes them according to the digits of the corresponding numbers. It can be implemented on the basis of least significant digit, which means that it starts from the least digit and moves towards the most significant digit, or the other way around, as is explained in appendix A.2. The saved information is a little bit different to the other categories as RadixSort groups elements into buckets according to their digits. These buckets are the centre of this visualization process.

In this section, we discussed the implemented sorting algorithms and the necessary information that is being kept track of and transported to the next module, which is the *Visualization Behaviour*.

4.2.4. Visualization Behaviour

After the algorithms have finished sorting the element arrays, the *Visualization Behaviour* module is called with relevant data. The saved information in the *Algorithms* module is interpreted here and decisive of the following actions. As mentioned previously, the different type of our *SortingVisualItem* objects is responsible for how the rest of the object data is interpreted and the resulting visualization.

4. Implementation

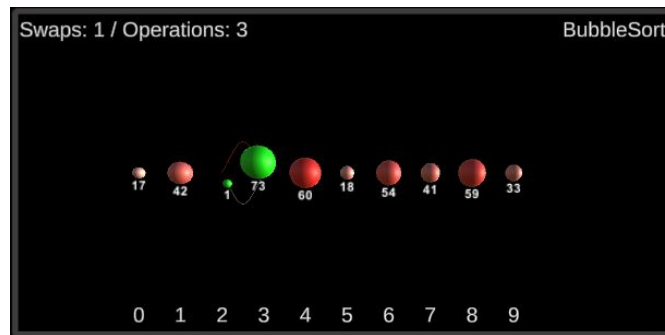


Figure 4.5.: Swapping of two elements

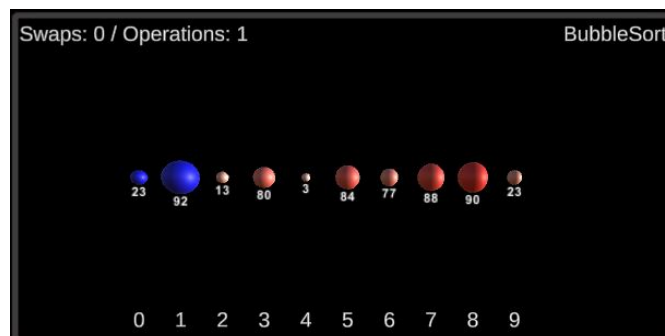


Figure 4.6.: Comparison of two elements

- **Swap:** Data consists of two elements that swap the place with each other. Both elements turn green as soon as the animation starts. A rotation point between them and two checkpoints are calculated to illustrate the trajectory, which is being kept track of, as good as possible. Figure 4.5 shows a simple swapping procedure.
- **Comparison:** The items contain this type when two elements are compared to each other. In this case, the elements turn blue to display that at this point in the algorithm, these two elements are compared, as shown in Figure 4.6.
- **Radix:** As the name implies, this type is used for moving elements in RadixSort. Figure 4.7 shows how the elements are grouped into the buckets, depending on the digit. After one digit has been handled, the elements are brought back to the original positions but in the new order. Then the next digit is processed, and so on. After all the digits have been handled, the elements are in a sorted and stable state, which means equal elements preserve their original order.
- **MergeArray:** As MergeSort is an out-of-place algorithm, it is necessary to illustrate the extra memory space appropriately. In this case, members of

4. Implementation

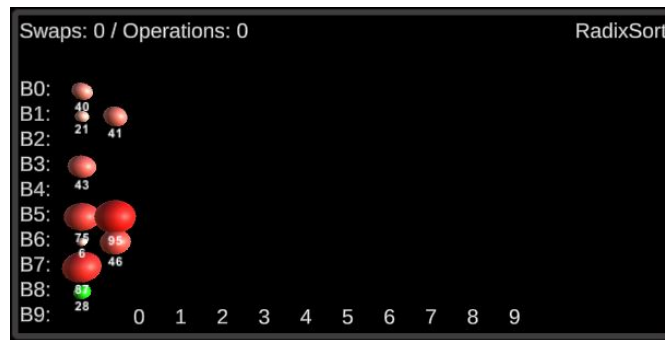


Figure 4.7.: Group by buckets

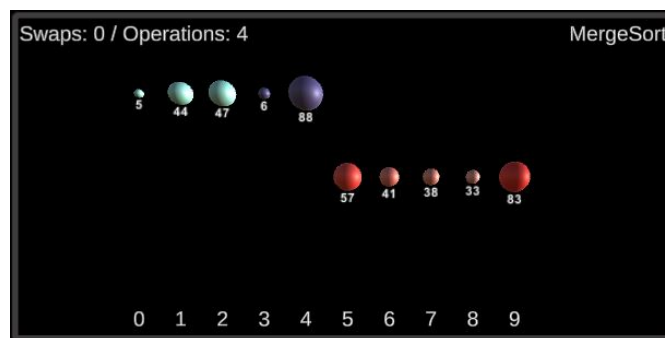


Figure 4.8.: Subarrays in MergeSort

extra arrays are coloured together clearly. Figure 4.8 shows two sub-arrays. Elements of the first sub-array are coloured teal, elements of the second array are coloured purple.

- **MergeMove:** After elements in the extra arrays of MergeSort are compared to each other, they move to their consequent position in the original array. While this animation is running the corresponding elements are turning green, like in all the movement scenarios.
- **MergeComparison:** Happens in MergeSort in the state when two extra arrays are displayed. In the course of this process, two elements are compared to each other and their colour changed to blue.
- **MoveTo:** This type is used when an element is moved to a specified position, colour is green while the element is on its way.
- **MoveMemory:** Similar to other movement cases but is particularly used when an algorithm takes an element of the array and saves it in a new local variable, which is later used to be compared. This behaviour is illustrated in Figure 4.9.

The previous itemization describes the behaviour and representation of different types of *SortingVisualItems*. The movement of elements itself is handled by the

4. Implementation

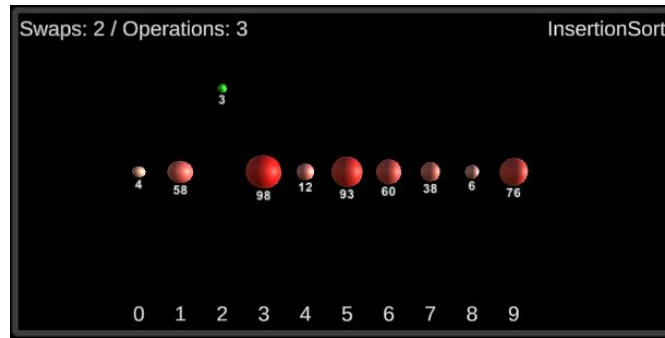


Figure 4.9.: Single element saved in memory

usage of an asset called *LeanTween*⁹. Listing 4 shows the simple process of swapping two elements. Apart from the destination points, it is possible to enter additional checkpoints to construct a cleaner trajectory.

```
LeanTween.move(element1,  
    new Vector3[] { dest2, temp1, temp1, dest1 },swapSpeed);  
LeanTween.move(element2,  
    new Vector3[] { dest1, temp2, temp2, dest2 }, swapSpeed);
```

Listing 4: Swapping of two elements

4.2.5. Input Interpreter

In the previous subsection, it was explained how the algorithms operate and how we visualize their behaviour. In this simulation, an additional module is used to handle user-entered code, the *Input Interpreter*. Due to the integration of a

```
JintEngine e = new JintEngine();  
e.SetFunction("swap",  
    new Jint.Delegates.Action<int, int>((a, b) => Swap(a, b)));  
...  
private void Swap(int a, int b) { ... }
```

Listing 5: Introducing functions with Jint

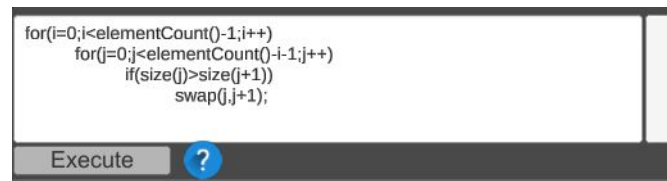
plugin called *Unity-Jint*¹⁰, it is possible to interpret *JavaScript*¹¹ code and execute

⁹LeanTween, 2016.

¹⁰Unity-Jint, 2016.

¹¹JavaScript, 1995.

4. Implementation



```
for(i=0;i<elementCount()-1;i++)
  for(j=0;j<elementCount()-i-1;j++)
    if(size(j)>size(j+1))
      swap(j,j+1);
```

Execute ?

Figure 4.10.: Algorithm in coding window

it on one or more *SortingBoxes*. Furthermore, we can introduce new functions that might be helpful for the users. In Listing 5, we show the introduction of a swap function, where two elements change their place in the same way as it does in the visualization for the implemented sorting algorithms. The provided functions, as previously mentioned in Figure 4.2, are as follows:

- **swap(x,y):** This function triggers the swapping behaviour where the elements x and y switch places with each other
- **size(x):** Returns the size of the element in array index position x
- **elementCount():** Returns the number of elements in the corresponding *SortingBox*

Aside from the self introduced functions, it is possible to execute loops and conditions, which offers the opportunity to try and test self-made algorithms. Figure 4.10 shows a simple adaption of the BubbleSort code to be executed by the *InputInterpreter*.

4.2.6. Communication

As we want the user to have full control over the ongoing events, the modules must communicate. If the pause button is pressed, everything should stop, if the step-to-end button is pressed, we want the *GameField* with its *SortingBoxes* and the *VisualizationBehaviour* module to jump to the end, if the visualization speed is changed, every component should know about it, and so on. Therefore a script is introduced to transport the relevant information to the other game object or scripts so that events are synchronized between each other at all times.

4.3. Pathfinding

In the previous section, we discussed the first simulation, Sorting. Now we will focus on Pathfinding, which is a topic that is a big deal in many different areas, especially games. Figure 4.11 shows the user interface of this simulation.

4. Implementation

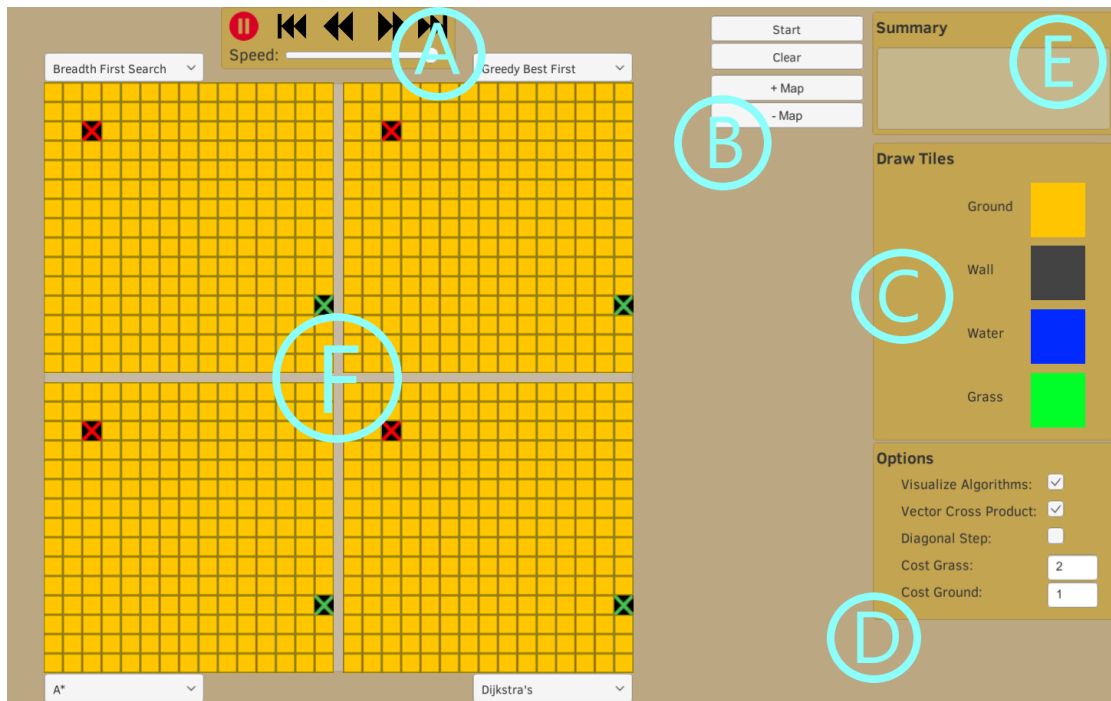


Figure 4.11.: User interface

4.3.1. Graphical User Interface

Similar to Sorting, the user interface is divided into several parts in this simulation.

1. **Controller:** The user interface controller, displayed in Figure 4.11 sector A, is the same for all our simulations and has been explained in detail in section 4.2.1.
2. **MapControls:** This simple part of the user interface is responsible for organizational actions, as can be seen in Figure 4.11, sector B.
 - Start: The *Start* button commands the execution of each selected algorithm in each map.
 - Clear: Clears the maps after a visualization has been finished so that the original map state is restored.
 - + Map: Adds an additional map up until a total of four maps.
 - - Map: Removes a map until there is only one remaining.
3. **Draw Tiles:** A tile is one small single component of the map. In this part of the user interface, it is possible to change the type of map tiles so that users can modify the map to their desired state and test algorithms in

4. Implementation

different scenarios. The available tile types, as shown in Figure 4.11 sector C, are as follows:

- Ground: This is the base tile. Using standard settings moving over a ground tile costs one step, but it can be changed by the user at will.
- Wall: It is not possible to step over a wall tile so all the algorithms will have to find paths around.
- Water: Same as the wall tile, it is not possible to move across water tiles.
- Grass: The grass tile can be passed, but it has increased movement costs. The level of movement costs can be changed by the user and is an excellent way of studying different scenarios.

To draw tiles on the map, the images of the different types need to be selected. After that simply clicking on the desired spots is enough to change the corresponding tile.

4. **Options:** Several options influence the behaviour of the simulation and can be chosen at runtime, see Figure 4.11, sector D.

- Visualize Algorithms: When activated all the tiles that are visited by the algorithm get visualized. Otherwise, only the final path is drawn. It is activated by default.
- Vector Cross Product: The vector cross product is an option for *Greedy Best First Search* and *A** to draw cleaner paths. Due to the square-based map, it often happens that multiple paths have the same length. Because the algorithms usually have a fixed order of visiting the neighbours, it happens that directions are rarely changed. The vector cross product prevents that and produces more logical paths. Listing 6 shows the implementation of the vector cross product.

```
protected float ComputeVectorCrossProduct(GameObject start,
GameObject goal, GameObject current)
{
    UIManager ui = GameObject.Find("UIManager").GetComponent<UIManager>();
    if (ui == null || !ui.vectorCrossProduct.isOn) return 0.0f;

    float dx1 = TileHelper.GetX(current) - TileHelper.GetX(goal);
    float dy1 = TileHelper.GetY(current) - TileHelper.GetY(goal);
    float dx2 = TileHelper.GetX(start) - TileHelper.GetX(goal);
    float dy2 = TileHelper.GetY(start) - TileHelper.GetY(goal);
    return System.Math.Abs(dx1 * dy2 - dx2 * dy1) * 0.001f;
}
```

Listing 6: Vector Cross Product for Cleaner Paths

4. Implementation

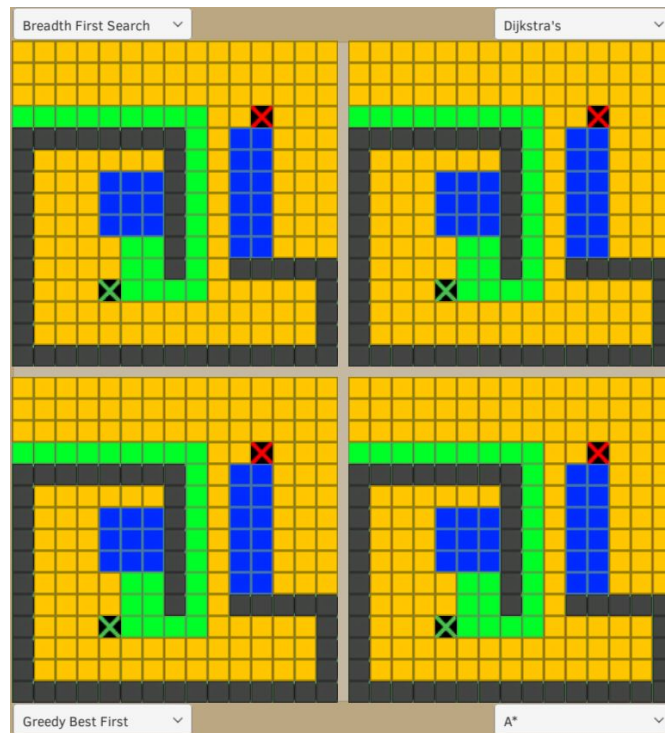


Figure 4.12.: Multiple modified maps

- Diagonal Step: This option allows our path to take diagonal steps towards the destination instead of only north, south, east and west.
 - Cost Grass: Setting for the movement costs of grass tiles.
 - Cost Ground: Setting for the movement costs of default ground tiles.
 - Visualization Delay: Handles the speed of the visualization.
5. **Summary:** This part of the user interface displays the result of algorithms after a path has been found. The required steps for each algorithm on the current map can be compared, as shown in Figure 4.11, sector E.

4.3.2. Game Field

The game field in this simulation, as can be seen in Figure 4.11 sector F, contains one to four maps with an attached dropdown field, where an algorithm for this map can be selected. In the beginning, the map consists of only ground tiles, one start point and one endpoint. The position of the start and endpoints can be changed to anywhere on the map, apart from wall and water tiles, by simply drag and dropping the tile. The maps can be modified and designed at free will. When adding more maps, the content of the first map is copied so that the user

4. Implementation

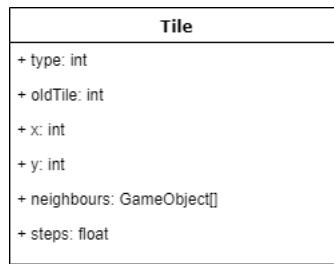


Figure 4.13.: Class Tile

does not have to redraw all the modifications. Figure 4.12 shows four modified maps with selected algorithms.

A map consists of a two-dimensional array of tile game objects. The structure of the tile class can be seen in Figure 4.13. It has a type attribute that is handled by an enumeration, similar to the approach in the sorting simulation. The *MapScript*, which is attached to each map, is responsible for setting all the reference to the neighbours of each tile. It is necessary information for all the pathfinding algorithms, which we discuss in the next subsection.

4.3.3. Algorithms

Similar to the algorithms in the sorting simulation, an abstract base class is created for the pathfinding algorithms. Again the algorithm classes are responsible for two things. First, the calculation of a path from the start to the endpoint and second, the storage of information about the underlying operations of the specific algorithm. Here we use a class called *AlgorithmStep*. An object of this class consists of a tile and a list of its neighbour tiles. That is enough to handle the visualization of the pathfinding process for all algorithms, which is explained in the next subsection. As follows are the implementation details of the provided pathfinding algorithms:

- **Breadth First Search:** This algorithm begins at the start point and puts all its neighbours in a queue. Then it visits the first item in the queue and again adds all the neighbours to the queue. This goes on until everything on the map is visited. Our implementation contains an early exit scenario by default, which cancels the algorithm as soon as the endpoint is found.
- **Dijkstra's Algorithm:** In contrast to Breadth First Search, this algorithm keeps track of movement costs for each visited tile. So for this purpose, we remove the queue and use a *PriorityQueue*¹² instead. This way, the movement costs can be taken into account when deciding how to evaluate

¹²PriorityQueue, 2013.

4. Implementation

locations. Listing 7 shows the definition of the *PriorityQueue* and the enqueueing of a tile with its corresponding priority.

```
SimplePriorityQueue<GameObject> frontier = new SimplePriorityQueue<GameObject>();
frontier.Enqueue(start, 0);
...
float newCost = costSoFar[currentTile] +
    map.GetCostByTileType(TileHelper.GetTileType(nextTile));
frontier.Enqueue(nextTile, priority);
```

Listing 7: Usage of Priority Queue

- **Greedy Best First Search:** As the previously mentioned pathfinding algorithms expand in all directions, Greedy Best First Search uses a heuristic function that calculates how close the current tile is towards the goal. This actual distance is used for the ordering of the *PriorityQueue*. Listing 8 shows the pseudocode of the heuristic function.

```
float Heuristic(Tile a, Tile b)
{
    return Math.abs(a.x - b.x) + Math.abs(a.y - b.y);
}
```

Listing 8: Example Code of the Heuristic Function

- **A*:** In A* the *PriorityQueue* is a combination of both the priority that gets derived from movement costs and the priority that is calculated by the heuristic function. The appropriate code from our work is shown in Listing 9.

```
float newCost = costSoFar[currentTile] +
    map.GetCostByTileType(TileHelper.GetTileType(nextTile));
...
priority = newCost + Heuristic(end, nextTile) +
    ComputeVectorCrossProduct(start, end, nextTile);
```

Listing 9: Priority used by A*

4.3.4. Visualization Behaviour

As the algorithms have done their work and stored the necessary information, the *Visualization Behaviour* module takes over and interprets the given data to start with the animation. To do that, the responsible script has to iterate through

4. Implementation

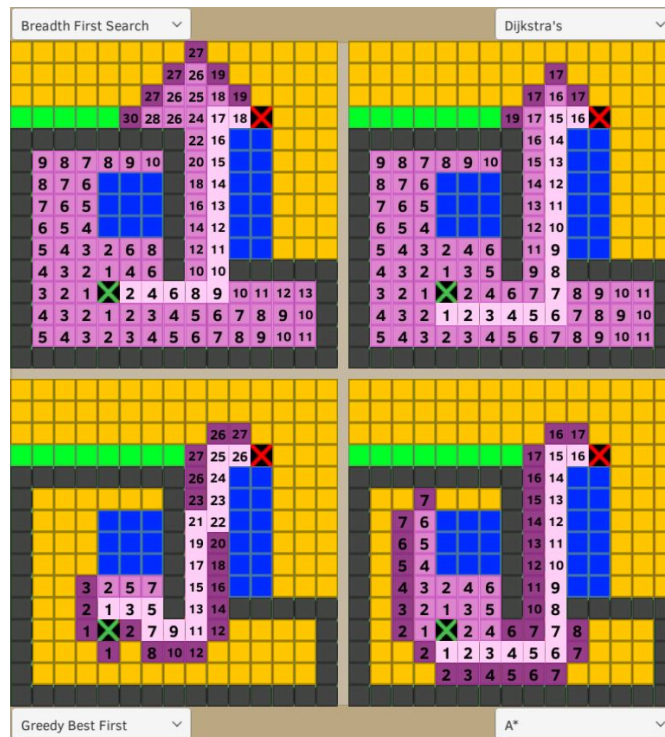


Figure 4.14.: Illustration of Executed Algorithms

the collected list of *AlgorithmStep* objects. For this purpose, a coroutine is used, which is like a function but can pause execution and return at a later time when the animation needs to be resumed. As previously mentioned, an *AlgorithmStep* object consists of a tile and a list of neighbour tiles. We use three different tile types for visualizing the algorithm behaviour:

1. PATH CURRENT: This colour represents a tile that is currently or was visited by the algorithm.
2. PATH NEXT: Tiles are coloured like that if they are about to be visited at a later time.
3. PATH: Tiles with this colour represent the final path.

Furthermore, a number of needed steps up until this point is displayed on every visited tile. Figure 4.14 shows the result of executed algorithms and the difference in their expansion. Due to having the ability to modify the maps quickly, it is easily possible to play through certain scenarios. It can be used to highlight the weak points of algorithms. For example, the weakness of Greedy Best First Search is that it is greedy and only wants to move in the most promising direction. Figure 4.15 shows such a scenario where Greedy Best First Search blindly moves into a dead end and as a result has to turn around and

4. Implementation

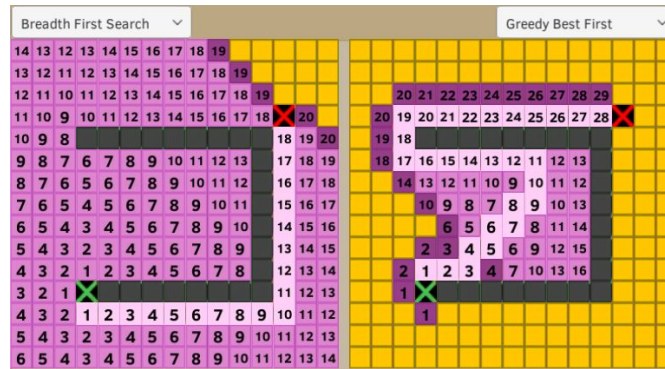


Figure 4.15.: Greedy Best First Search Weakness

take a much longer path than Breadth First Search, which has found the optimal path but is slower in general.

Due to the UI Controller, the user can decide on the flow of the visualization. The process of finding the path for the same map can be repeated as often as wished. The user can stop the visualization and continue in steps to better understand each algorithm. Listing 10 shows the approach to visualizing the pathfinding algorithms. A coroutine is used to replicate the corresponding algorithm step by step.

4.3.5. Communication

Similar to the *Communication* component in section 4.2.6, the same module in this simulation is needed so that the user interface, the game field with its maps, the algorithms and their behaviour work together.

4.4. Binary Search Tree

In this section, the implementation details about the simulation of a binary search tree data structure are discussed, which contains the user interface, the underlying algorithms, the game field, and the behaviour for visualization. Figure 4.16 shows the user interface of this simulation.

4.4.1. Graphical User Interface

The user interface in this simulation is held rather simple and is divided into three parts, which is enough to satisfy the required usability.

4. Implementation

```
IEnumerator DoVisualizeAlgorithms()
{
    for(;;_visualizationCounter < _algoSteps.Count; _visualizationCounter++)
    {
        AlgorithmStep algoStep = _algoSteps[_visualizationCounter];
        if (algoStep == null) continue;
        // set current tile
        if (algoStep.CurrentTile != start && algoStep.CurrentTile != end)
            SetTileType(algoStep.CurrentTile, (int)TILE_TYPE.PATH_CURRENT);

        foreach (GameObject next in algoStep.NeighbourTiles)
        {
            if (next != start && next != end)
            {
                SetTileType(next, (int)TILE_TYPE.PATH_NEXT);
                TileHelper.SetTileText(next);
            }
        }
        yield return new WaitForSeconds(GetVisualizationDelay());

        while (IsPaused() || _isBusy)
            yield return null;
    }
    _visualizationCounter--;
    DrawPath();
}
```

Listing 10: Coroutine to visualize pathfinding algorithms

1. **Controller:** As discussed in section 4.2.1, the *Controller* component of the user interface, displayed in Figure 4.16 sector A, is responsible to give the user full control over the simulation. The possibility to move through the algorithm step by step is an important aspect to fully understand the operating principle.
2. **Main:** Figure 4.16, sector B, shows the main part of the user interface in this simulation and consists of the following functionality:
 - Add: This button adds a new node with the entered key to the tree structure.
 - Search: A process is started to search for a specific key in all nodes of the tree.
 - Delete: Tries to delete the node with the entered key.
3. **Message Log:** This is a kind of event log where a summary of the performed operations is quoted. It is implemented in a way so that the latest message is in line with the current state of the visualization. An

4. Implementation

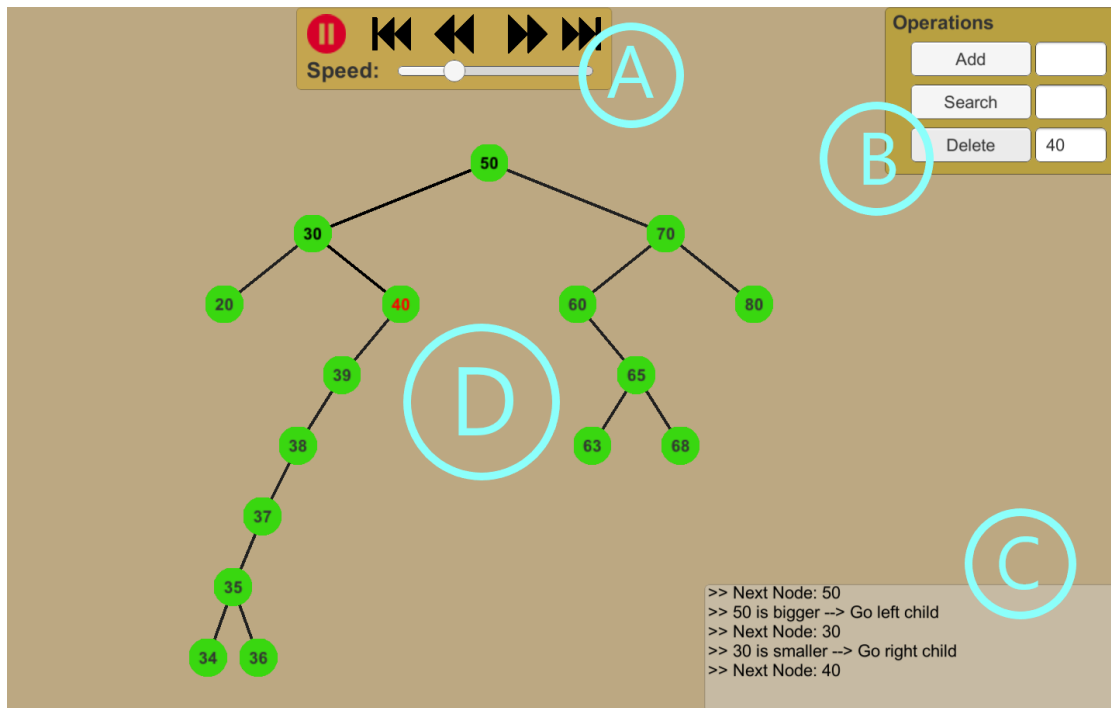


Figure 4.16.: User interface

illustration of the Message Log is shown in Figure 4.16, sector C.

4.4.2. Game Field

In this simulation, the game field is represented by the complete tree structure, as shown in Figure 4.16 sector D, that consists of so-called nodes and edges that symbolize a parent-child relationship between nodes. Figure 4.17 shows the structure of the node class. It consists of a level attribute that corresponds to its depth in the tree, a key that identifies the node and is visible in its centre and references to its parent, left child and right child nodes. The level attribute is important for the positioning of each node. As can be seen in Figure 4.16 sector D, the edges starting from the root node are longer and in a different angle than edges in deeper levels. Listing 11 shows the calculation to position each node correctly. The positions are calculated based on the nodes parent position, its level and static values for x and y differences. This principle is used when nodes are to be moved and change their positions after certain operations, for example when a node, that is in the middle of the tree and has only one child, is deleted and all the children need to be repositioned. In the class, which represents our whole tree structure, it is sufficient to only hold a reference to the

4. Implementation

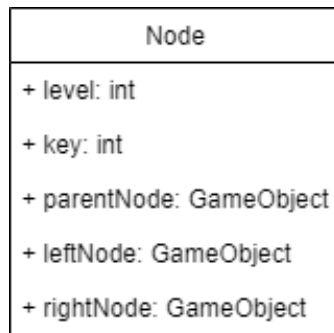


Figure 4.17.: Class Node

root node to perform all implemented operations. The next subsection discusses the algorithms that are visualized in this simulation.

```
if (level == 0)
    return gameObject.transform.localPosition = NodeManager.ROOT_POSITION;
float x = (isLeftNode) ? parentNode.transform.localPosition.x -
    NodeManager.X_DIFF / level : parentNode.transform.localPosition.x +
    NodeManager.X_DIFF / level;
Vector3 position = new Vector3(x, parentNode.transform.localPosition.y
    - NodeManager.Y_DIFF, 0.0f);
```

Listing 11: Calculation of node positions

4.4.3. Algorithms

As opposed to the algorithm implementations of previously discussed simulations, we execute all operations in the script of the tree data structure. The root node is used to initiate every single process and browse through the tree. Furthermore, information for visualization is stored at every relevant point of the operation in a class called *BSTVisualItem*. We have implemented the following operations on a binary search tree data structure according to our research in section A.2.2:

- **Search:** The search function takes the root node and the key, which should be searched for, as parameters. The key of the current node is compared to the wanted key. If it is equal, the searched node has been found, if it is null, then there is no node in the tree with the given key. Otherwise, the function is recursively called again with either the left child node or the right child node, depending on if the searched key is bigger or smaller than the key of the current node. Each step a result is displayed in the

4. Implementation

message log. In this process, we save the visited nodes and their edges that lead to the next node in a new *BSTVisualItem* object. The used algorithm and storage of objects for visualization can be seen in Listing 12.

- **Insert:** Similar to the search function, the tree is browsed through to find the appropriate location to insert a new node. It also takes a node, starting with the root node, and key as parameters. In addition to the saved objects in the search function, it is necessary to save an event for the spawning of a new node so that it can be visualized properly.
- **Delete:** First, the node that is to be deleted has to be found. This is done similarly, as described in the search function. After that, further actions have to be taken depending on the number of children this node has. If it is a leaf node, which means there are no children, then it just needs to be stored that this node is deleted. If the node has one child, then we additionally need to keep track of the child node because its position has to be updated after the parent node is removed. Otherwise, if the found node has two children, we have to find its inorder successor, which is the smallest node in the right subtree, and recursively call the delete function again. Therefore, we also save objects for finding the inorder successor, moving it, and changing the key of the original node.

```
private GameObject Search(GameObject node, int key)
{
    bstVisual.Items.Add(new BSTVisualItem(node, (int)VisualType.Node));
    // root null or root has key
    if (node == null || node.GetComponent<NodeScript>().Key == key)
    {
        bstVisual.Items.Add(new BSTVisualItem(node, (int)VisualType.FoundNode));
        return node;
    }
    // key > node key
    if (node.GetComponent<NodeScript>().Key < key)
    {
        bstVisual.Items.Add(new BSTVisualItem(node, (int)VisualType.RightArrow));
        return Search(node.GetComponent<NodeScript>().RightNode, key);
    }
    // key <= node key
    bstVisual.Items.Add(new BSTVisualItem(node, (int)VisualType.LeftArrow));
    return Search(node.GetComponent<NodeScript>().LeftNode, key);
}
```

Listing 12: Search Node Function

4. Implementation



Figure 4.18.: Structure of the BSTVisualItem Class

4.4.4. Visualization Behaviour

After the operations, which have been explained in the previous subsection, have finished running and stored the necessary data, then the *Visualization Behaviour* module of this simulation is called. It loops through the *BSTVisualItem* objects, whose structure can be seen in Figure 4.18, and decides on the resulting visualization depending on its type. In the following list, the action taken for different types of objects is briefly explained:

- **Node & InorderSuccessor:** Occurs when a node is visited. Changes its colour and logs the corresponding message.
- **LeftArrow & RightArrow:** Changes the colour of the edge to the left or right child node.
- **SpawnNode:** Spawns a new node with the set key as a left or right child of the given parent node.
- **DestroyNode:** Deletes a specific node. If the node has a child, then all nodes further down on this subtree get repositioned.
- **SetNodeKey:** Changes the key and the colour of the stored node.
- **InorderSuccessorMove:** This type of *BSTVisualItem* is saved in the process of deleting a node with two children. A temporary node is spawned with the key of the inorder successor and moved to the original node position. It should display the change of the original node key after the inorder successor has been found.

4. Implementation

- **InorderSuccessorFound:** Occurs when the inorder successor of a node has been found. Displays the corresponding message log entry.

Due to the UI Controller, the user can decide on the flow of the visualization, similar to the other simulations. It can be stopped, and steps can be repeated and reverted as often as wished. Therefore, coroutines are used to loop through the visualization objects that break on specific conditions and last as long for each step as stated in the speed slider option of the user interface.

4.4.5. Communication

Similar to the previous simulations, the modules call and communicate with each other to keep everything informed about the game state.

4.5. Website

In the previous sections, all the simulations were discussed in detail. Due to some of our functional and non-functional requirements in section 3.1, a website was created using the convenient *GitHub Pages*¹³. It contains an overview about the topics, images and links to the uploaded *WebGL*¹⁴ builds of each simulation, as can be seen in Figure 4.19. Because of that it is possible to access the simulations from anywhere with any device or browser with that it is possible to display *WebGL* content. Furthermore, we can add new content or simulations in a very simple and quick way.

4.6. Sorting in Virtual Reality

In this section, the port of the Sorting simulation into a virtual reality environment is discussed. Most of the background code base, like the algorithms and visualization data, can be kept in its original state but some other parts have to be replaced to make the simulation look visually appealing for a virtual reality environment.

Resources

For the virtual reality environment, some further resources were necessary for being able to develop and run the simulation. The head-mounted-display *HTC*

¹³GitHub Pages, 2008.

¹⁴WebGL, 2011.

4. Implementation

Computer Science Visualizations in Unity3D

This is a collection of visualizations in Unity3D of various computer science topics.

Sorting Algorithms

Sorting algorithms are algorithms that put randomly arranged elements of a list into a certain order, usually numeric or lexicographical order. In this project we visualize several sorting algorithms and compare their behaviour. Furthermore we can execute user entered code to swap elements and simulate different algorithms.

Sorting Algorithms.

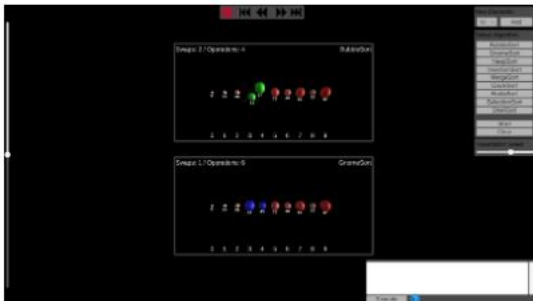


Figure 4.19.: Website including *WebGL* builds

4. Implementation

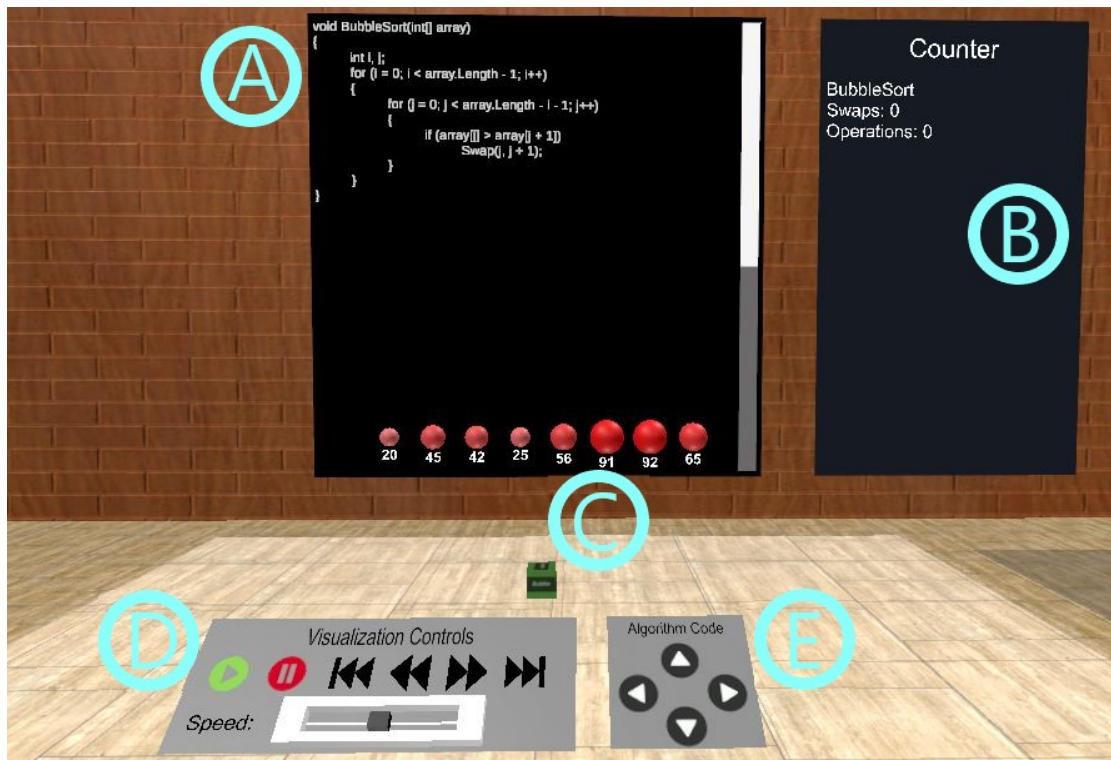


Figure 4.20.: VR: Game Field

*Vive*¹⁵ has been used combined with the virtual reality system *SteamVR*¹⁶, which acts as a software interface. Its system requirements are on the higher side requiring an Intel i5-4590, AMD FX 8350 equivalent or better as CPU, 4 GB or more RAM and a Nvidia GeForce GTX 970, AMD Radeon R9 290 equivalent or better graphics card. Due to that, it is not possible to run on rather old systems. For development, *VRTK*¹⁷ has been utilized, which is a collection of useful and reusable solutions to common problems in a virtual reality environment. It provides many example objects that can be modified and included in existing projects.

4.6.1. Graphical User Interface

Most of the user interface of the Sorting simulation, which has been discussed in section 4.2.1, is removed as in a virtual reality environment the user is actively engaged in the world by moving around and utilizing objects. Because of that, it

¹⁵HTC Vive, 2016.

¹⁶Steam VR, 2015.

¹⁷Extend Reality Ltd, 2018.

4. Implementation



Figure 4.21.: VR: Configuration Desk and Algorithm Cubes

would be counterproductive to keep all the user interface elements that are used by clicking. An exception is the *Controller* object which is responsible for starting, pausing, and stepping through the visualization, as well as controlling the speed. The look of the icons is kept the same, but they are replaced by images that users can trigger by using the virtual reality controller device. The slider to change the speed of the visualization is also substituted with a grabbable slider in VR style, as can be seen in Figure 4.20 sector D. Additionally, there is a panel that contains four usable images of arrows to handle the currently showed pseudocode of a sorting algorithm, shown in Figure 4.20 sector E. The left and right arrows change the displayed algorithm, the top and bottom arrows scroll the code up and down. In the remainder of the section, all the objects that were introduced in the span of the virtual reality implementation are described.

4.6.2. Virtual Reality Objects

In the beginning, the users spawn in front of the *Controller* object and also see other parts of the simulation in the background, like the game field. The actual starting point is on the left side to configure several options.

Configuration Desk

This object is located on the left side of the initial spawn point and contains two different configuration options that take place when new elements are created, as can be seen in Figure 4.21 sector A.

4. Implementation

1. **Slider:** This slider is used to choose the number of elements that are to be spawned in the game field. The small black piece in the middle can be grabbed and moved back and forth. The resulting number of elements changes according to the position and gets displayed in a text field in front of the object.
2. **Lever:** It is responsible for how new elements are spawned. The main reason it is introduced in this environment is that in contrast to the original Sorting simulation, it is not possible to quickly change the place of elements by entering and executing code. It can be used by grabbing the lever and pushing it back or pulling it forth. The text in front of the object displays the current option and changes according to the position of the lever. The system only generates a new numerical order, if currently, no set of elements is present, or if the configuration has been changed. It can have the following options:
 - **Random:** With this option a completely random order of elements is generated.
 - **Nearly sorted:** If this option is selected, an almost sorted list of elements is spawned. The difference to a sorted state is very small and only requires few swaps, depending on the number of elements.
 - **Reversed:** In this case, the elements spawn in a reverse sorted state. It is a worst case scenario for some of the sorting algorithms.

Algorithm Desk

The next object is the algorithm desk, shown in Figure 4.21 sector B. It contains many cubes, each of them representing a sorting algorithm. Users identify the type of algorithm by its looks, as the names of the corresponding algorithms are written on the cubes. They can be grabbed and thrown onto the game field to spawn a new set of elements.

Game Field

Figure 4.20 shows the Game Field, identified by a small area with a slightly brighter floor than the rest of the room. It has colliders attached to it so that an event gets triggered when a cube falls onto it. The simulation then creates a new set of elements, considering the configured options, above the point of collision between the cube and the game field (see Figure 4.20, sector C). Following that, the position of the elements can be changed by grabbing the cube and moving it to another location. The simulation starts by pressing the play button on the *Controller* object and visualizes the algorithms identical to the standard version, except for *RadixSort*, where additional cubes are spawned

4. Implementation



Figure 4.22.: VR: Area to Reset Algorithm Cubes

to represent the different buckets. Furthermore, behind the game field, there are two panels on the wall, which can be seen in Figure 4.20 sector A and B, respectively. The first one displays pseudocode of the implemented algorithms and is managed by four arrows in the *Controller* object. The second one shows the required operations and swaps of the currently executed algorithms. On the right side of the main game field, there is another small area which has the purpose to reset the cubes and their corresponding algorithms, as can be seen in Figure A. The procedure is similar to the spawn of elements, which means that users grab the cube and let it land on this small area. Consequently, the set of elements is deleted, and the cube returns to its original position in the simulation.

4.7. Summary

In this project simulations of three computer science topics - Sorting, Pathfinding and Binary Search Trees - were implemented in Unity3D¹⁸ with roughly the same core components, whereas one of them, the Sorting simulation, was additionally built in a second environment. Besides all the mentioned requirements, the focus of development and design was on usability and control. Research and

¹⁸Unity3D, 2005.

4. Implementation

tests have shown that the user having full control over the simulation regarding speed is a decisive factor for the usefulness of visualized algorithms. The user interface was purposely kept simple for all the simulations. Additionally, some extra configuration options were provided depending on the type of simulation, like the possibility to enter and execute code in the Sorting simulation or the different path visualization and movement cost parameters in the Pathfinding simulation. After having finished development of all the mentioned simulations, a website was created, which includes WebGL¹⁹ builds and a summary of each part. Due to that, it is possible to access the visualizations with a web browser from everywhere. Finally, the Sorting simulation has been ported into a virtual reality environment. Most of the core components could be retained but some, like for example the user interface, had to be transformed into objects that are more appropriate for a virtual reality environment. Therefore, small widgets have been provided that let the user actively engage in the virtual world.

Throughout the design and development process, there have only been minor issues. One example was a performance-related issue in the Pathfinding simulation. The execution and visualization of algorithms resulted in small lags when there were multiple complex maps with an increased amount of tiles. Because of that, the number of tiles on each map has been limited to a smaller number. Apart from that, no abnormalities were recognized as every simulation runs smoothly.

¹⁹WebGL, 2011.

5. Evaluation

To find out if our simulations have the potential to assist in learning computer science topics, an evaluation has been conducted and analyzed. This chapter explains the content and structure of the evaluation, describes its procedure, gives an overview of personal information about the participants, and finally shows the results.

5.1. Research Methodology and Procedure

Due to the fact that the sorting simulation has been implemented in two different environments, it enabled us to directly compare both education methods, in addition to evaluating each of them. Therefore, the procedure was as follows: First the participants had to answer a pre-questionnaire which collected some personal information and background. After that, they were asked to test the first simulation. One half started with the web version, the other half with the virtual reality simulation. After testing each simulation, they had to answer a post-questionnaire, which contained questions about the impression they got, motivational aspects, the System Usability Scale (Brooke, 1996), the Computer Emotion Scale (Kay & Loverock, 2008), and the Game Engagement Questionnaire (Fox & Brockmyer, 2013). In the end, they had to do a final questionnaire, which focused on the comparison between the two simulation environments. All questionnaires were created and performed with the tool LimeSurvey¹.

5.1.1. Pre-Questionnaire

The pre-questionnaire acquired some personal information about the participants like gender, age, education and profession. Furthermore, it questioned them about their previous experience with computer science, simulations, computer games and virtual reality.

¹LimeSurvey, 2003.

5. Evaluation

	Strongly disagree	Disagree	Undecided	Agree	Strongly agree
	0	1	2	3	4
I think that I would like to use this system frequently.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I found the system unnecessarily complex.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I thought the system was easy to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I think that I would need the support of a technical person to be able to use this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I found the various functions in this system were well integrated.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I thought there was too much inconsistency in this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I would imagine that most people would learn to use this system very quickly.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I found the system very cumbersome to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I felt very confident using the system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I needed to learn a lot of things before I could get going with this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 5.1.: System Usability Scale questionnaire

5.1.2. Post-Questionnaire

The post-questionnaire was carried out by using the System Usability Scale (Brooke, 1996), the Computer Emotion Scale (Kay & Loverock, 2008), the Game Engagement Questionnaire (Fox & Brockmyer, 2013), and asking several simulation specific questions.

System Usability Scale

The System Usability Scale (SUS) (Brooke, 1996) is a well-established and easy to use questionnaire, which is independent of technology and rates the usability of a system. It contains ten questions based on a Likert-scale:

- 0 (strongly disagree)
- 1 (disagree)
- 2 (undecided)
- 3 (agree)
- 4 (strongly agree)

The total points are added up and multiplied by 2,5. A result of 68 or higher usually illustrates good usability of the system. The maximum achievable points are 100 (Brooke, 1996). The SUS has been chosen to ensure that the implementation of our simulations in the light of usability has been done satisfyingly so that it is beneficial to the users. An illustration of the SUS can be seen in Figure 5.1.

Computer Emotion Scale

The Computer Emotion Scale (CES) is a reliable scale to rate emotions while learning new software on computers. It uses a total of 12 corresponding feelings out of four emotion constructs:

1. Anger

5. Evaluation

- Angry
 - Frustrated
 - Irritable
2. Anxiety
- Anxious
 - Helpless
 - Insecure
 - Nervous
3. Happiness
- Curious
 - Excited
 - Satisfied
4. Sadness
- Disheartened
 - Dispirited

Participants are asked about their feelings during testing the software. Their possible answers are based on a Likert-scale:

- 0 (None of the time)
- 1 (Some of the time)
- 2 (Most of the time)
- 3 (All of the time)

The results can be interpreted by calculating the mean score of each user for each emotion construct (Kay & Loverock, 2008). The purpose of using CES in the evaluation is to get a better understanding of the emotions of users while using the simulations. Due to that, it is possible to spot some eventual problematic parts of the implementation and improve it.

Game Engagement Questionnaire

The goal of the Game Engagement Questionnaire is to measure the psychological engagement of users when playing a game. It collects information about immersion, presence, flow, and absorption. The questionnaire includes 19 questions, and possible answers range from 'fully agree' to 'do not agree' (Fox & Brockmyer, 2013).

5. Evaluation

Simulation Specific Questions

Several simulations specific questions were asked after each simulation has been tested, to get a better understanding of the impression, motivation, and experience of the users. The types of possible answers to the questions contained both free text and Likert scales.

5.1.3. Post-Post-Questionnaire

After both post-questionnaires were answered, another short survey was conducted to compare both simulations to each other. The participants were asked which type they prefer, for which application they prefer it, and about their reasoning for that.

5.2. Participants

There were twelve participants aged from 15 to 34 ($M = 27$, $SD = 7.1$). Five of them were female (41.67%), and seven of them were male (58.33%). Only two of them (16.67%) had a visual impairment, glasses and contact lenses. Their professions made up as follows: six employed (50.00%), five students (41.67%), and one self-employed (8.33%). Their job titles compose of many different fields like a teacher, nurse, assistant professor, project manager, and process engineer. Related to that, their highest level of education ranged from high school graduates to PhD. On a scale from 1 to 5, about half of them rated their ability to use computers and video games above average, however eight (66.67%) regarded their skills with virtual reality usage as below average ($M = 2.17$, $SD = 1.27$). Their frequency of playing video games is quite balanced ($M = 3.25$, $SD = 1.76$). The majority of participants like role-playing, action, and adventure games the most. Seven (58.33%) of them have heard about Google Cardboard or Samsung GearVR environment, whereas all seven have known Oculus Rift and Samsung GearVR, and only one less HTC Vive. Two (16.67%) of the participants have experienced cyber sickness before. Most of them considered their knowledge about computer science as average or lower ($M = 2.5$, $SD = 1.09$), and they rated their knowledge about sorting algorithms even worse ($M = 2.25$, $SD = 1.26$).

5. Evaluation

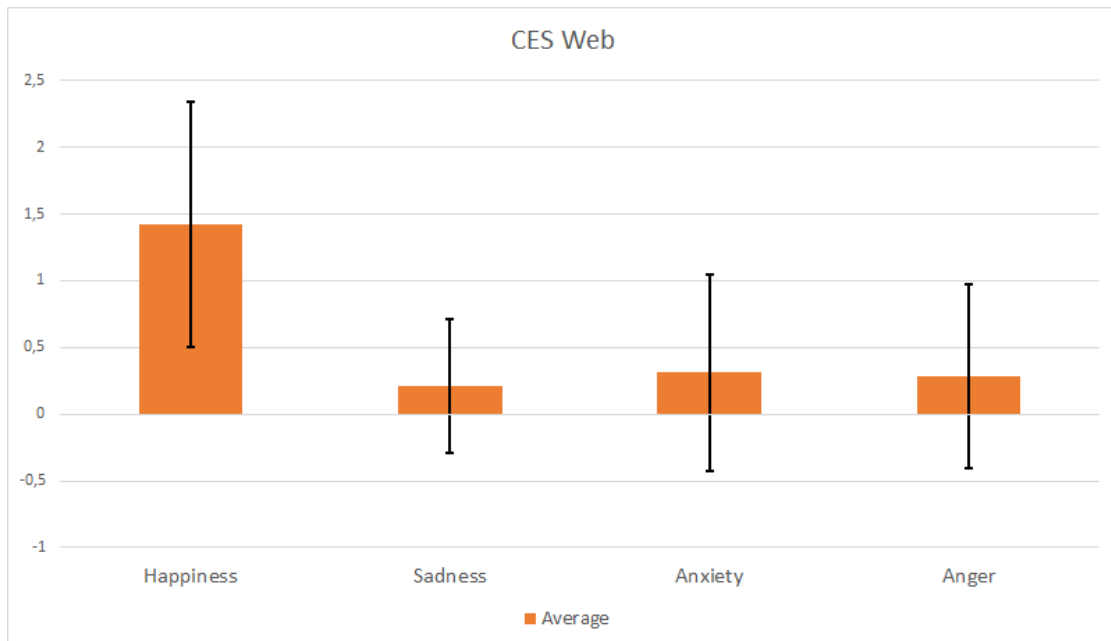


Figure 5.2.: Results of the Computer Emotion Scale for the Web Environment

5.3. Results

In the following sections, the results of the evaluation of each environment, as well as the comparison between both, are presented.

5.3.1. Sorting in WebGL

System Usability Scale

In the web version of the simulation there was only one person (8.33%) that thought the system was unnecessarily complex and difficult to use. However nine (75.00%) agreed that the system was easy to use. There was no one who found that the functions were badly integrated. One person thought there was too much inconsistency in the system. Two participants (16.67%) found the system cumbersome to use, whereas one of them admitted afterwards to having misunderstood the meaning of the word cumbersome. The overall score of the SUS is 69.49 with a standard deviation of 17.08. Worth mentioning are two outlier, which had a score of 35.7 and 43.35 respectively.

5. Evaluation

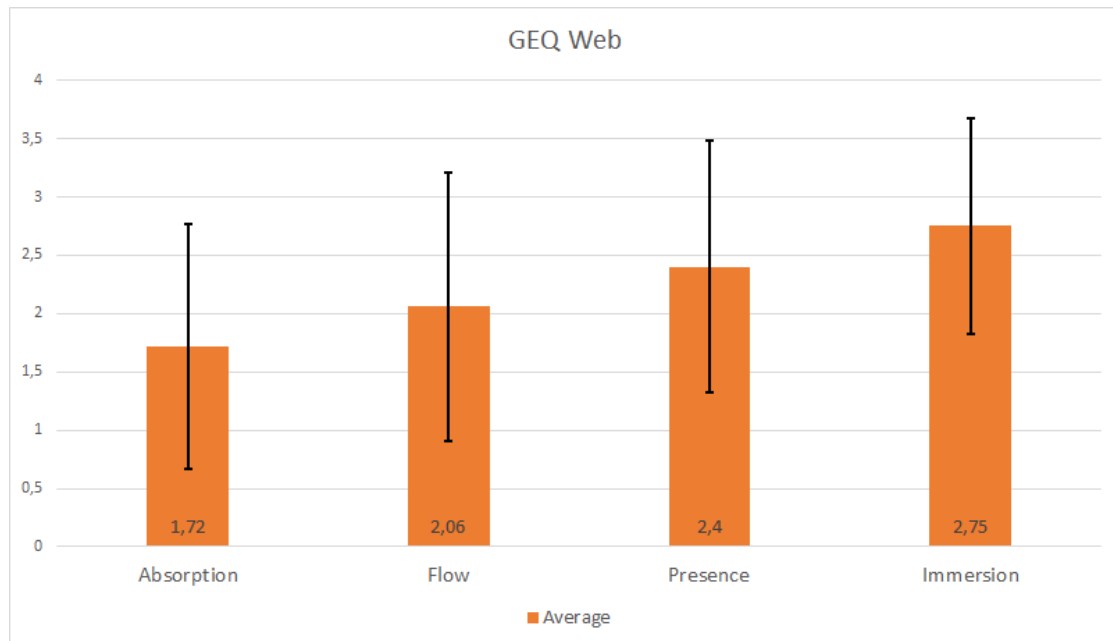


Figure 5.3.: Results of the Game Engagement Questionnaire for the Web Environment

Computer Emotion Scale

Figure 5.2 shows the results of the CES for the web environment. It is grouped by the four main emotions and displays the average score and standard deviation of all users per emotion. As explained previously, the scale ranges from 0 (none of the time) to 3 (all of the time). According to the results, sadness got the lowest score of 0.21 with a 0.5 standard deviation. There was one participant who felt disheartened some of the time and two participants who felt dispirited some of the time. Worth mentioning here is that one user felt dispirited most of the time. Anger received an average score of 0.28 (SD = 0.69). Here one participant felt irritable all the time, which is a clear outlier because the rest selected 0 (none of the time), despite one user who selected 1 (some of the time). Anxiety got a score of 0.31 (SD = 0.74). Nobody felt anxious and only one person felt nervous some of the time. However, two participants thought to be helpless all the time while testing the simulation. The highest score achieves the emotion happiness with an average score of 1.42, which is a bit lower than expected but has a quite high standard deviation of 0.92. It might have something to do with many participants not being familiar with computer science at all, and therefore having problems comprehending the content in the beginning.

5. Evaluation

Game Engagement Questionnaire

The GEQ collects information about absorption, flow, presence and immersion. Figure 5.3 shows the average score of all users combined for all the parameters. The lowest result occurs in the absorption part ($M = 1.72$, $SD = 1.05$), shortly followed by flow ($M = 2.06$, $SD = 1.15$). Presence is experienced by more users ($M = 2.4$, $SD = 1.08$) and immersion scores the highest result with the lowest standard deviation ($M = 2.75$, $SD = 0.92$), which was to be expected, as this item is the easiest to agree with.

Simulation Specific Questions

In this category, basic questions about their impression and motivation were asked. Most of the participants liked the simulation overall, with one stating that *"it has a big potential if you know the process behind it"*. Some features they liked are the style of visualization and that it is easy to use, however, some mentioned they did not like that there is no introduction and further explanation of the algorithms, and therefore hard to understand for people with no background in computer science. 75% of them would probably use the simulation for learning. One of them mentioned that achievements or quests could be added, making it more engaging and motivating. Most of the participants thought that it was good for learning, and some said, it would be good with some further additions. Moreover, they found the interaction in the simulation was quite good and self-explaining overall. The participants also mentioned some possible improvements:

- Introduction in the beginning
- Further explanation of the sorting algorithms
- More information on how to add loops or clauses in the coding window

Overall they rated their immersion of the experience above average ($M = 6$, $SD = 2.04$) on a scale from 1 (very little) to 10 (very deep). For the motivational questions, a scale from 1 (fully disagree) to 7 (fully agree) was used. Some meaningful results include, for example, answers to the statement that the sorting web application is a good supplement to regular learning, where 75% agreed with. Also, 75% agreed that they learned something with the simulation. Eight participants (66.67%) thought that it makes learning more engaging, while nine (75%) thought that it makes learning more interesting. Likewise, nine users agreed that learning with the sorting web application is more motivating than ordinary exercises and that it makes content more interesting. Furthermore, eight participants agreed that they would like to learn with the simulation in the classroom.

5.3.2. Sorting in Virtual Reality

System Usability Scale

The overall score of the SUS in the virtual reality simulation is 76.08 (SD = 12.04), which classifies it as a usable system according to Bangor, Kortum, and Miller (2009), where 70 is the minimum required score. Nobody found that the system is unnecessarily complex or that there was too much inconsistency. Ten participants (83.33%) thought that it was easy to use, and nine (75%) would imagine that most people would learn to use the system very quickly. Moreover, nobody felt unconfident or thought that they would need to learn a lot of things before they could get going with this system.

Computer Emotion Scale

In the CES, happiness got the highest score (M = 2.0, SD = 1.05) again of all four main emotions. As part of this emotion, seven participants (58.33%) were satisfied all the time. Anxiety is the second highest with an average score of 0.25 and a standard deviation of 0.44. One participant reported being helpless most of the time, whereas the other answers for this and the other emotions in this sector (anxious, insecure, nervous) only include 0 (none of the time) and 1 (some of the time). Sadness got a score of 0.125 (SD = 0.44). Nobody felt disheartened, but one user felt dispirited some of the time, and one most of the time. Anger got the lowest average score of 0.08 with a standard deviation of 0.28. Here two participants were frustrated some of the time, one was irritable some of the time, and nobody was angry. An overview of the result can be seen in Figure 5.4.

Game Engagement Questionnaire

The results of the GEQ for the virtual reality simulation, as shown in Figure 5.5, are as follows: Immersion got the highest score of 3.67 and a 0.89 standard deviation. Second highest is presence (M = 2.67, SD = 1.26). After that, flow is the next highest parameter (M = 2.33, SD = 1.26). The lowest score is absorption (M = 2.22, SD = 1.24). Worth mentioning is that for this GEQ result immersion scored way higher and with a far lower standard deviation than the other parameters. It seems like the virtual reality environment was a very immersive experience for most of the participants.

5. Evaluation

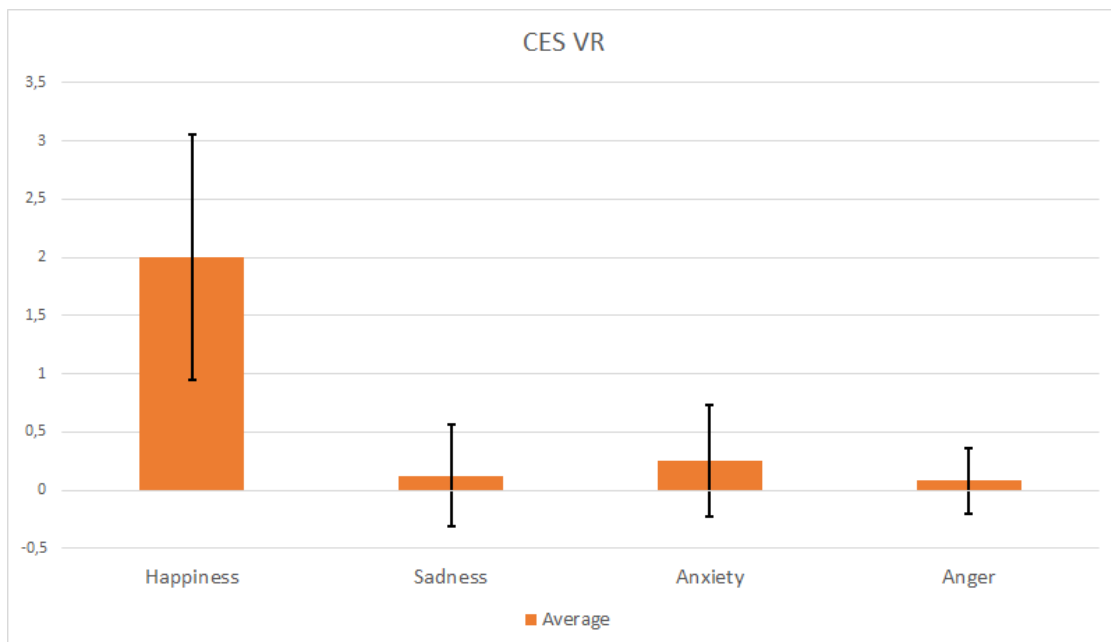


Figure 5.4.: Results of the Computer Emotion Scale for the Virtual Reality Environment

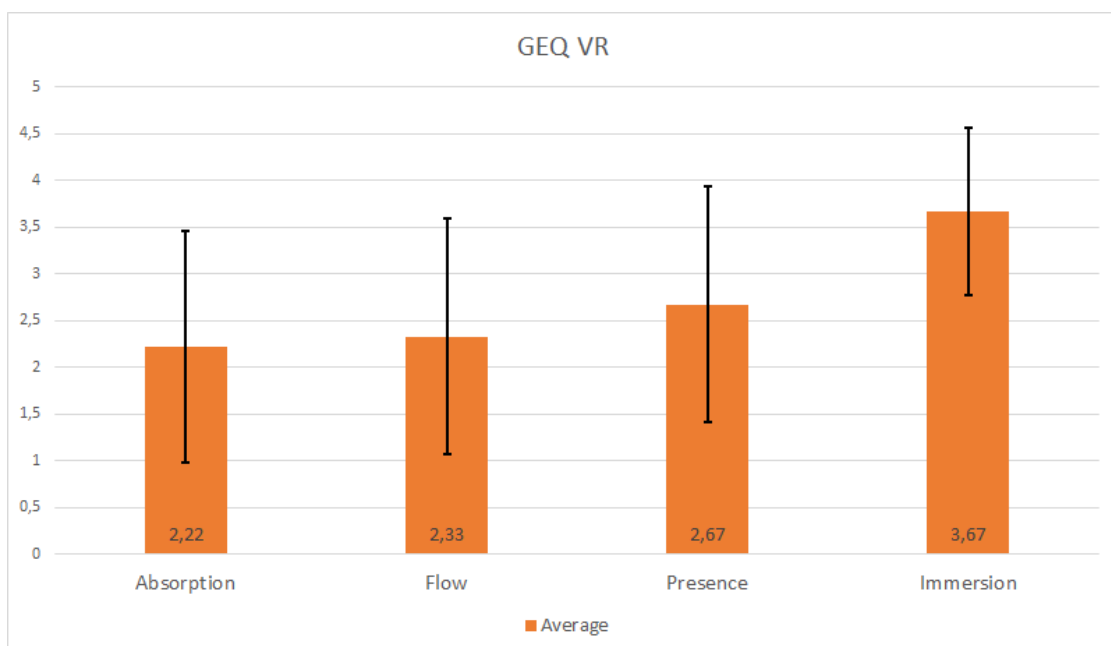


Figure 5.5.: Results of the Game Engagement Questionnaire for the Virtual Reality Environment

5. Evaluation

Simulation Specific Questions

The feedback in this part of the survey was overwhelmingly positive, as all the participants noted that they like it. They enjoyed various things, like one user wrote: *"I think the cubes are a good idea and I like the concept of throwing them in order to interact and work with them"*. Furthermore, they mentioned that they like the possibility to throw things around, the visualizations of the algorithms and that you can see the code, that it's fun to learn with, the ability to teleport and to be in a virtual world. On the other hand, there are only very few parts some of them did not like. For example, one participant mentioned that it was not always clear if a button was pressed or not. The vast majority of them found the simulation engaging and motivating and would use it for learning. Furthermore, everyone enjoyed the virtual reality experience and thought it was good for learning. Some users highlighted the interactivity of the simulation there. The suggestions for improvements were very constructive and even aimed into the direction of future work. They mentioned that some sort of tutorial would be beneficial, as well as further descriptions for the levers and buttons. And some said they would like to see more rooms of this kind with different backgrounds implemented. All the twelve participants thought that the controls were easy to use and rated the immersion, which ranged from 1 (very little) to 10 (very deep), with an average score of 7.67 (SD = 1.89). The results of the motivational questions, which go from 1 (fully disagree) to 7 (fully agree), are also continuously positive. Ten persons agreed that they would like to learn with Sorting in virtual reality (M = 5.67, SD = 1.03). All but one participant agreed, that the simulation makes the content more interesting (M = 6.25, SD = 1.16). Furthermore, all twelve users thought that Sorting in virtual reality makes learning more fun, with an average score of 6.33 (SD = 0.85). For the statement *"the simulation inspired me to learn more about computer science"* the results are a bit more varying, as three (25%) of them disagreed. However, eleven participants (91.67%) agreed that it makes the content more interesting. More than half of the users found regular computer science classes boring, which probably has something to do with the fact that the majority of them are not affiliated with computer science in any way. However, most of them agreed that the computer science simulation with the virtual reality head-mounted-display was interesting and engaging. The desire to use such an application on a smartphone seems to be rather low (M = 2.83, SD = 1.68). Finally, ten out of twelve participants (83.33%) agreed, that they would like to learn with the virtual reality simulation in the classroom (M = 5.17, SD = 1.52).

5. Evaluation

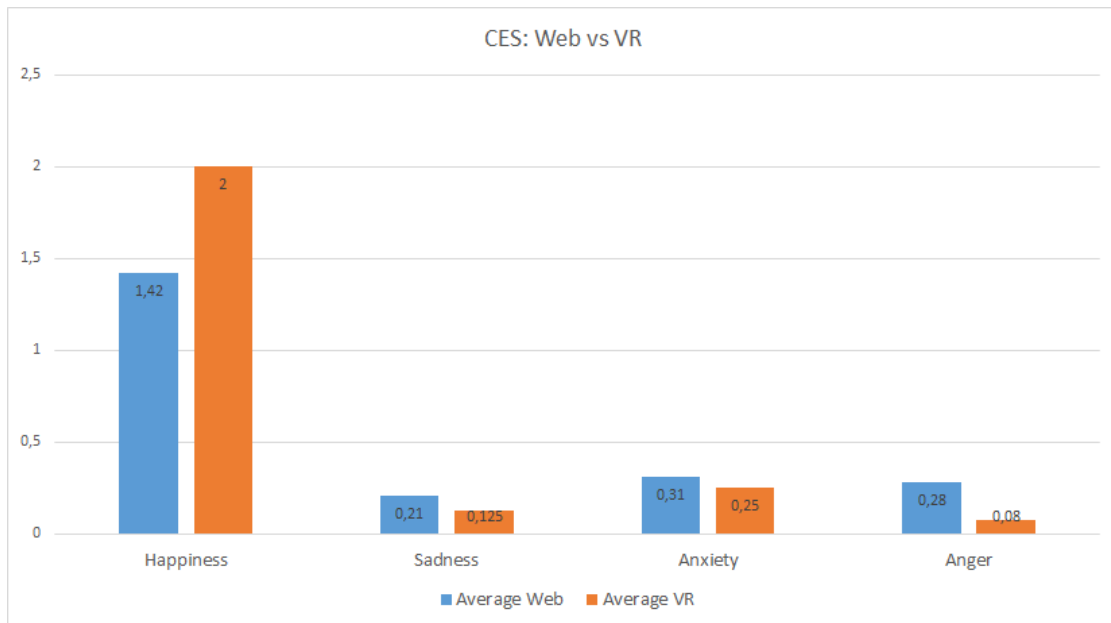


Figure 5.6.: Comparison of the Computer Emotion Scale between Web and VR

5.3.3. Comparison

In the end, the participants had to answer five more questions, that aimed to compare the web and virtual reality environment directly. Eleven users (91.67%) said that they prefer the virtual reality environment. Their reasoning was versatile as they mentioned it's more interactive, cooler, more user-friendly, more interesting, more fun, more exciting, more engaging and more motivating. The one participant, that preferred the web environment, based the decision on suffering from cybersickness. They would generally prefer the virtual reality version for mechanical engineering, chemistry, physics, medical applications, and further difficult real-life scenarios. On the other hand, they would use the web version independently at home for more theoretical content. The final question was about which device they would rather use for learning computer science. The answers are quite similar to the first question, as eleven out of twelve participants would choose the virtual reality environment with the same reasoning as above.

5.4. Discussion

The feedback of the participants in the study was very interesting and informative. It is worth mentioning that the majority of the attendees have no computer

5. Evaluation

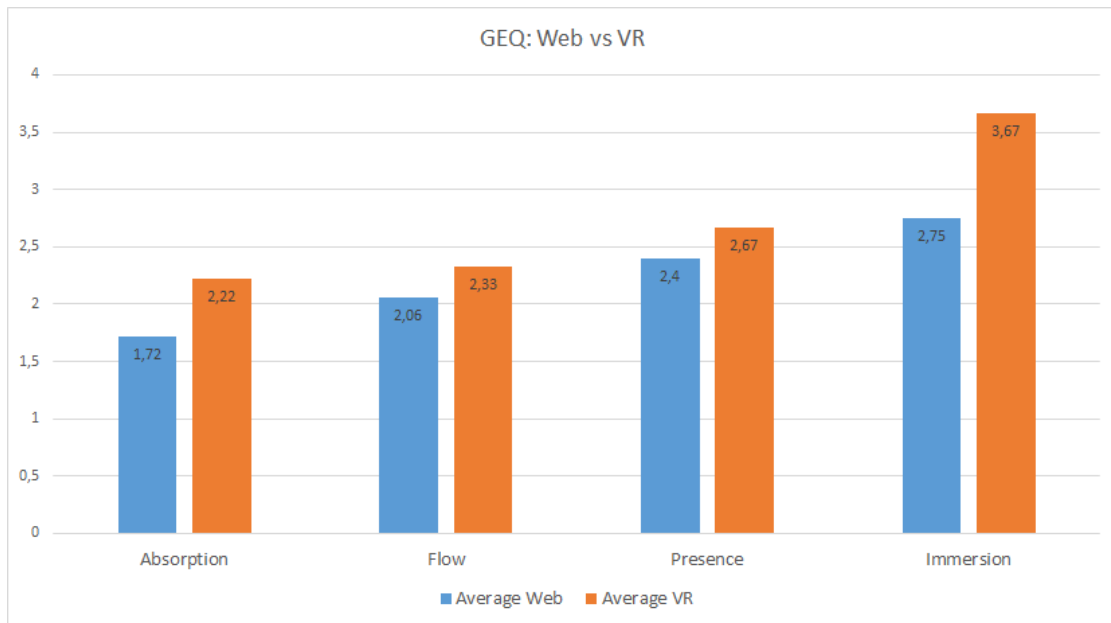


Figure 5.7.: Comparison of the Game Engagement Questionnaire between Web and VR

science background and only minor or no prior knowledge about computer science, or sorting algorithms specifically. Therefore, it felt that some of them had some difficulties in getting started, especially in the web version, which is confirmed by their feedback about possible improvements to add some sort of tutorial and further description of the algorithms. Some also noted that it would be beneficial to have quests or tasks and in further consequence, to obtain achievements. Furthermore, it seemed like a few participants occasionally misunderstood the meaning of some words, which has been confirmed by at least one person. This circumstance might have influenced the results in one way or another. However, it was to be expected, as there was not a single native English speaker participating in the survey. Generally, the majority of the participants enjoyed both simulations. They liked that the web version was easy to use and its style of visualization. Most of them agreed that using the web simulation makes learning more interesting and motivating than ordinary exercises. In the virtual reality version, they liked the enhanced interactivity, the virtual world, and that it is fun. Furthermore, they enjoyed the deep immersion, and that it makes the content more interesting and learning more fun. A bit unexpected is the rather big difference in rating between the web version and the virtual reality version. The VR environment achieved a 9.5% higher score in the SUS rating, the happiness emotion in the CES increased from 1.42 to 2, and the scores for the absorption, flow, presence, and immersion parameters in the GEQ are way higher for every single category, as can be seen in Figure 5.6

5. Evaluation

and Figure 5.7. While the simulations in both environments are not perfect and improvements can be made, the participants still had a clear trend in favour of the virtual reality environment, as they felt it is more engaging and exciting.

6. Lessons Learned

In this chapter, the challenges, issues and insights which were encountered in the course of the different project phases, namely literature research, development and evaluation, are summarized and discussed.

6.1. Literature

Since the goal of this work was to create a system which assists people in learning computer science topics, it was necessary to thoroughly study different approaches to teaching computer science, as well as refresh personal knowledge about the specific fields that were implemented.

In the beginning, traditional education methods were researched, and it was quickly obvious why such an approach is subpar nowadays. Lectures were more and more build in a problem-based way, and students were actively involved. With the start of the common computer era, many new techniques were introduced like computer visualizations, interactive simulations and educational games. These approaches have shown to have a big potential in raising students motivation and engagement. Another promising field was the introduction of virtual reality. It is still a little bit of a futuristic vision, and users are excited about themselves being actively involved with their movement in a virtual world. Therefore, it has been decided to create computer simulations and port one of them into virtual reality so that it is possible to compare a simulation in two different environments directly.

Before starting with the development, it was also important to refresh our understanding of computer science theory and some applications which can be seen in Appendix A and A.2 respectively. The knowledge achieved due to that was the base for the implementation of the Sorting, Pathfinding and Binary Search Tree simulation.

6.2. Development

After studying related simulations and educational games in the previously mentioned research literature, it was decided to develop simulations of specific

6. Lessons Learned

computer science topics in Unity3D¹. Unity was used as a development environment as it is easy to handle and supports multiple platforms. The simulations were designed by considering publication on a website so that users can have access from everywhere they want. This was accomplished by making use of the possibility to create WebGL² builds in Unity3D.

One major challenge was to visualize and illustrate the algorithms in a way so that users with no or little knowledge were able to understand the content. Feedback was crucial to advance the simulations to a useful state and to the point where they turned into beneficial assistance when learning the corresponding computer science theory. Another issue was the interpreter to execute code in the sorting visualization. Some tools were found and implemented successfully but weren't working in WebGL builds of the simulation. Due to that, in the end, Unity-Jint³ was used, which made it possible to execute code in this target build.

The second part of this study was to port the sorting simulation to a virtual reality environment and make it possible to directly compare them. This was done with the help of VRTK⁴, which contains many example objects and scripts. To get a better feeling in the virtual reality environment, many user interface elements had to be replaced by game objects that let users engage in the world actively. This whole process was an exciting phase of the project because it was the first time of the author to experience developing in a virtual reality environment.

6.3. Evaluation

The evaluation consisted of several standardized questionnaires like SUS, CES, and GEQ for a general assessment of the system, as well as simulation specific questions about motivation and impression. Due to that, the participants could freely enter their thoughts and give constructive criticism. In the end, five final questions were asked, which directly aimed at the comparison between the two different environments. The results are very interesting and informative as the author did not expect the participants to be so open towards virtual reality. But it seems like they enjoyed the enhanced engagement and interactivity in a virtual reality environment. The testing was conducted at two different locations. On the one hand, in a room at the Graz University of Technology, and on the other hand, at the home of the author. In hindsight it might have been better to do

¹Unity3D, 2005.

²WebGL, 2011.

³Unity-Jint, 2016.

⁴Extend Reality Ltd, 2018.

6. Lessons Learned

the testing in bigger rooms with a larger movement radius, as the movement by feet was a bit limited in the virtual reality environment and due to that, teleport was the primarily used method to cover even short distances. Furthermore, it might have been a good idea to conduct an additional phase of evaluation earlier in the development process to get early feedback from participants with a different background and level of knowledge about computer science. It would have enabled the early implementation of some of their concerns. However, the results of the evaluation show that the majority of the participants liked the simulations, especially in the virtual reality environment. Their extensive feedback has given valuable information about improvements and future work scenarios.

7. Future Work

This chapter discusses possible future improvements of this work, which consist of both user feedback received from the survey, and the authors own thoughts. Generally, future work will more likely focus on the virtual reality environment, due to the perceived fundamental and overwhelmingly positive attitude towards it.

7.1. Evaluation Results

One primary suggestion was the integration of an introduction or tutorial to get a quick overview of the purpose of the simulation and its basic functionality. This can be achieved by adding panels that include a detailed description, or in further consequence, by a pedagogical animated agent. It is a lifelike character, which the users can interact with, to obtain further information, as Johnson et al. (2000) explain. On another note, the participants stated that a further description of the interactable objects would be useful. It would mean both to denote objects, which can be interacted within the virtual reality environment, and add a short description of the type of interaction that is necessary like pressing or grabbing. Furthermore, it might be an idea to explain the functionality and idea of the sorting algorithms verbally besides the source code. Due to that, users with little or no experience in programming or computer science, in general, can understand the content easier.

7.2. Further Scenarios

As previously mentioned in the results of chapter 5, some users already provided feedback that falls into the category of future work. They mentioned that quests and achievements might be a way to further increase the engagement and motivation of the simulation in the virtual reality environment. This could also be achieved by lifelike pedagogical agents. Many online role-playing games have such interactable non-player characters. They signal newly available quests by having an exclamation mark above their heads and then proceed to describe the task. This can be combined with adding achievements like points

7. Future Work

for completed quests. They also requested to implement more rooms with computer science simulations in the virtual reality environment. The Binary Search Tree simulation, which is also published in the web environment, would, for example, be a suitable candidate to port into the virtual reality environment, as its underlying style of representation can be kept roughly the same. The Pathfinding simulation would need some adjustments so that its visualization of algorithms is more appropriate to a virtual reality environment. Apart from that, it is also an idea to implement learning scenarios for simple data structures like arrays, where users have to do quests and solve tasks.

8. Conclusion and Discussion

As technology advances, and adopting computational thinking skills and computer science knowledge gets more and more important in today's world, it is important for students to start learning the fundamentals at an even younger age. However, learning computer science is a very challenging task that can lead to frustration and demotivation. Therefore, it is essential to improve the learning conditions and environments, to assist people optimally, when dealing with such complex topics. Digital learning includes, for example, videos, simulations, educational games, virtual worlds, and virtual reality and shows promising results in many areas to increase the motivation and engagement of students. This work presented the design and implementation of Sorting, Pathfinding, and Binary Search Tree simulations in a web version. Furthermore, the Sorting simulation was also implemented in a virtual reality environment. The simulations help users to understand these computer science topics by visualizing their underlying algorithms and displaying further information. By having developed the same simulation in two different environments, it was possible to compare the possible learning achievements and user satisfaction. Therefore, an extensive survey was conducted, where twelve participants tested both the web version and the virtual reality version. First, they had to fill out a pre-questionnaire, which gave insight into some personal information and prior experience to computer science and games. After that, they tested a simulation, where half of the participants started with the web version, and the other half started with the virtual reality version. Then they had to fill out a post-questionnaire directed towards the corresponding environment, which was followed by testing the simulation in the other environment, including questionnaire. In the end, they were asked to answer five final questions, which aimed to compare both environments directly with each other to get information about their preference. The results were quite remarkable and interesting, as almost all of the participants leaned toward the virtual reality environment because they thought it was way more motivating, engaging, and interactive. They felt a quite deep immersion while being in the virtual world with head-mounted displays on their head. These results show that people are quite open about virtual reality and that it can be an effective way of learning in many different applications.

Appendix A.

Computer Science Fundamentals

Even up until today, scholars have different views about the definition of computer science, which means there is no consensus. There are many answers to the question "what is computer science?" (Hazzan et al., 2014). According to Denning (2005) "*computer science is the science of information processes and their interactions with the world. Computers are tools to implement, study, and predict them.*" Computer science is a very diverse area of expertise which contains scientific and engineering facets. Computational thinking skills are needed to deal with their underlying problem-solving processes. It is about searching, learning, planning, scheduling, and using a big amount of data to solve computer science problems (Wing, 2006).

A.1. Essential Theoretical Fields of Computer Science

Computer Science is a very broad subject area that consists of abstract topics like algorithms, data structures, programming et cetera. It is therefore difficult for students, especially for ones that lack skills in abstract thinking, to learn basic concepts of computer science in a didactic way without visualizations (Burbaite, Stukys, & Marcinkevicius, 2012). Building knowledge of algorithms and data structures is essential for any computer science disciple of study. Moreover, it's not only useful for students but for everyone that uses computers and wants to solve bigger problems (Sedgewick & Wayne, 2011). For example, computer-aided design systems have become a vital part of chip design processes. They rely on two important factors. Firstly, data structures that are used to representing appropriate information and switching between functions need to be compact. Secondly the algorithms, which do the work on the data structures, need to be efficient (Meinel & Theobald, 1998). The next sections introduce algorithms and data structures in general and list a few computer science areas where such concepts are essential to achieve specific goals.

A.1.1. Algorithms

According to Sedgewick and Wayne (2011) *"the term algorithm is used in computer science to describe a finite, deterministic, and effective problem-solving method suitable for implementation as a computer program. Algorithms are the stuff of computer science: they are central objects of study in the field"*. For instance, a specific algorithmic problem occurs. Every basic process to solve this problem, regardless of the discipline, starts with comprehending the requirements and ends with a simple draft of a sequence of instructions that ideally solves the problem, which is an algorithm usually coded in a programming language (Hazzan et al., 2014). Algorithmic thinking is an important type of computational thinking. It involves the ability to think about problems generally and to abstract common characteristics, which in further consequence leads to having a group of problems instead of a single one, and to create a series of actions as a solution (Harteveld, Smith, Carmichael, Gee, & Stewart-Gardiner, 2014). Euclid's algorithm is one of the first examples of algorithms, dating back to the Greek mathematician Euclid who described it in his work *Elements*. It is about a problem of finding the greatest common divisor of two numbers. The principle is that, if we subtract the smaller number from the bigger number, the greatest common divisor doesn't change. So if the smaller number, keeps getting subtracted from the bigger number, the result is the greatest common divisor. If the bigger number is way higher than the smaller number, then a lot of subtractions may be needed to reach the solution. There is a more efficient way to solve this issue by replacing the bigger number with its remainder when divided by the smaller number (see Figure A.1).

One aspect of most algorithms is how data that is part of the computation is organized. It brings us to another essential field of computer science, named data structures. Algorithms and data structures belong together. It is virtually impossible to successfully learn one of the two without also understanding the other (Sedgewick & Wayne, 2011). So the next section is about briefly introducing the theory of data structures.

A.1.2. Data Structures

Representing information is one of the essential parts of computer science. Information needs to be stored and retrieved with a significant emphasis laid on speed. Therefore, it is important to structure information so that efficient processing is provided, which is only possible through the study of data structures and algorithms, the fundamentals of computer science (Shaffer, 2013).

According to Aho, Hopcroft, and Ullman (1983) data structures are *"collections of variables, possibly of several different data types, connected in various ways"*. They

Appendix A. Computer Science Fundamentals

English-language description

Compute the greatest common divisor of two nonnegative integers p and q as follows: If q is 0, the answer is p . If not, divide p by q and take the remainder r . The answer is the greatest common divisor of q and r .

Java-language description

```
public static int gcd(int p, int q)
{
    if (q == 0) return p;
    int r = p % q;
    return gcd(q, r);
}
```

Euclid's algorithm

Figure A.1.: Euclid's algorithm: Problem description and code implementation (Sedgewick & Wayne, 2011)

are any data representation with related operations. The term is used to describe organizing of data item collections, for example, a sorted list of integer values. Furthermore data structures have a few requirements:

- Certain amount of space for items
- Certain amount of time for operations
- Certain amount of effort in programming

Computer science problems often have several constraints. Only after a thorough analysis, it is possible to choose the most suitable data structure for the solution (Shaffer, 2013).

With the right approach, many more possibilities come to light when tackling complex tasks where millions of objects are processed, like the potential to increase performance by huge factors (Sedgewick & Wayne, 2011). Computers become more and more powerful, which slightly aids when struggling with efficiency. However, problems nowadays also get larger and more complex, which is why program efficiency becomes an even bigger necessity. Computer scientists need to be taught to get an as deep as possible understanding of principles because using proper data structures can lead to a big difference in the efficiency of a solution (Shaffer, 2013).

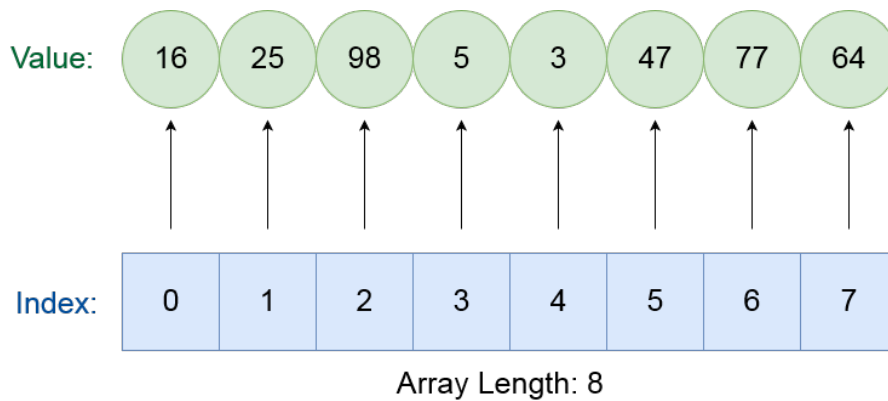


Figure A.2.: Basic integer array

Basic Data Structure

One of the most basic and most used data structures is an array. It usually consists of elements of the same type. All elements are equally quickly accessible by using their corresponding index and can also be selected randomly (Wirth, 2004). See Figure A.2 for a basic array containing integer values.

Fundamental Data Structures

The selection of a proper data structure is dependent on the magnitude of information that needs to be stored. If a program only needs to store a few simple things, without having to worry about any form of search, then the most effective way would be to put them in a list. Otherwise, if the organization and searching through a very large dataset is needed, then the use of more complex data structures becomes a requirement (Shaffer, 2013). In the following enumeration, a few of those fundamental data structures are briefly explained.

- **Lists:** A list contains a finite sequence of elements. Each element is at a specific position and has a certain data type. In simple list implementations, each element is of the same data type. It grants access to fields like length, begin and end (Shaffer, 2013). There are two basic types of list implementations:
 - **Array-Based List:** Elements are stored in neighbouring cells of an array, which means they are easily enumerated. New elements are added to the tail. The operations *insert* and *delete* require other elements to be shifted by one place (Aho et al., 1983).
 - **Linked List:** A linked list is a recursive data structure, a sequence of objects called nodes. Nodes contain a generic item that can have

any data type and a reference to another Node, which forms the linked list (Sedgewick & Wayne, 2011). Below code shows a very basic implementation for a linked list.

```
private class Node
{
    Node next;
    Item item;
}
```

- **Stacks:** Stacks are a structure similar to lists, however, elements can only be added and deleted from one end. That means that stacks are not as flexible as lists but for applications that only need their simple form of operations, they are efficient and easy to implement. Therefore in cases like that, it is preferable to use them over lists. Stacks can also be array-based and linked, quite similar to lists (Shaffer, 2013).
- **Queues:** Queues are a particular kind of lists where items are inserted at the tail and deleted at the front. It works under the principle of FIFO, which means *"first in first out"*. The operations are quite comparable to stacks. Again, it can be implemented array-based or linked. However, in this case, the standard array-based implementation is not as efficient which is why it is realised as a kind of circular array instead (Aho et al., 1983).
- **Dictionaries:** Dictionaries perform under the principle of key and value. It is very frequent in everyday life to store and retrieve information. For example a demand to look up a specific entry in a database. Usually, entries are defined as a kind of key-value, often id numbers. It is necessary for the keys to be comparable so that it's possible for a search to determine the entry with the corresponding key (Shaffer, 2013).
- **Trees:** According to Wirth (2004) *"a tree is either an empty structure or a node of type T with a finite number of associated disjoint tree structures of base type T, called subtrees"*. It is a hierarchical structure on a collection of elements that are called nodes. One of them is identified as the root node and is the starting point for the tree structure and its relations (Aho et al., 1983). Figure A.3 shows a graph representation of a tree structure with node A being the root node.

In this section, we explained the essential theoretical fields of computer science, like algorithms and data structures, which are fundamental for further engagement in this field of expertise. In the next section, we focus on several application areas that are relevant to this project.

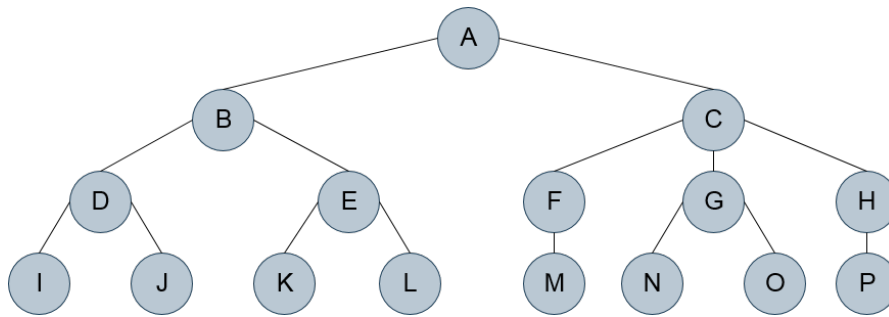


Figure A.3.: Graph Representation of a Tree Structure

A.2. Application of Basic Fields of Computer Science

Now that algorithms and data structures were briefly introduced, the next step is to discuss specific topics that make use of the previously mentioned fundamentals. In this case, we focus on sorting algorithms, binary search trees, and finding paths between points.

A.2.1. Sorting

Sorting is a prominent task in everyday life, be it card games, documents or spices. At the same time, it is one of the most commonly executed computing tasks. Therefore, sorting is an essential part of the world that has been studied full-scale for a long time (Shaffer, 2013). Research of sorting algorithms is rather old with analysis of *BubbleSort* even dating back to Demuth (1956). Still nowadays, new sorting algorithms get invented mostly because of the complexity to solve problems efficiently. A general definition in computer science is that sorting algorithms are algorithms that put items of a collection into a certain order, most of the time in numerical or lexicographical order (Durrani, Shreelakshmi, & Shetty, 2012). Figure A.4 shows a basic graphical representation of a numerical sorting problem. A very common practical example is the usage of sorting techniques in database systems for index creation, query operations and sorted output requested by the user. In addition to standard sorting algorithms, such modern systems also employ advanced techniques that improve the adaptive behaviour or the overall performance, like for example the simplification and reorder of comparison keys (Graefe, 2006). Below follows a list with brief descriptions of several popular sorting algorithms which have been analyzed in detail in Sedgewick and Wayne (2011), Aho et al. (1983), Wirth (2004), Shaffer (2013), Clément, Hien Nguyen Thi, and Vallée (2013), Hammad

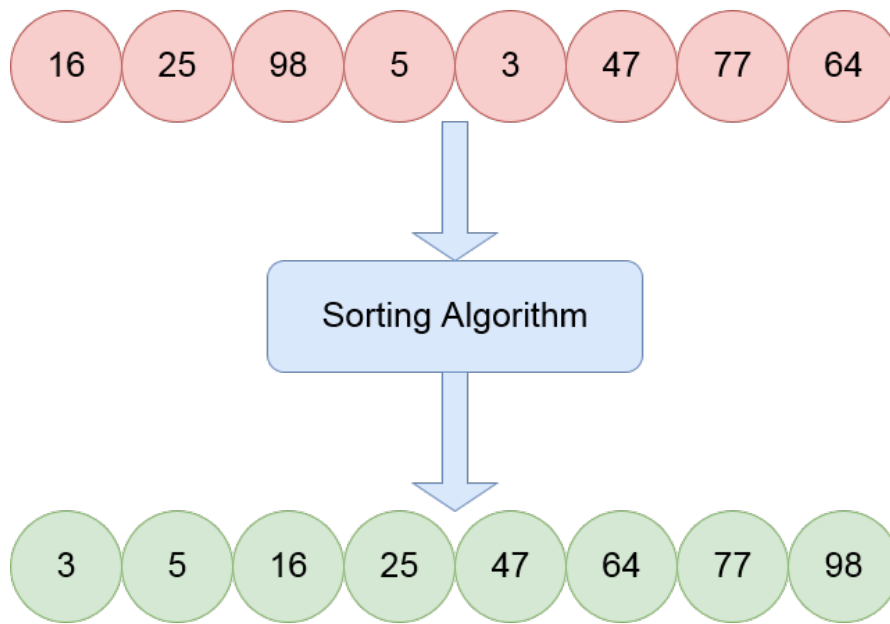


Figure A.4.: Basic Sorting Problem

(2015) and Durrani et al. (2012).

- **BubbleSort:** BubbleSort is one of the oldest and simplest sorting algorithms. It is a double for loop where every item in the list is compared with the item next to it and swapped if necessary. This process is repeated until it goes through the whole list without any swaps. It is easy to implement but very inefficient.

Table A.1.: BubbleSort Complexity

Time complexity	
Best case:	$\mathcal{O}(n)$
Worst case:	$\mathcal{O}(n^2)$
Average case:	$\mathcal{O}(n^2)$
Space complexity	
	$\mathcal{O}(1)$

- **SelectionSort:** The idea behind SelectionSort is related to human intuition. It finds the largest element and puts it into its correct spot in the list by switching places with the element at the end. It continues to do so until the array is sorted. SelectionSort is a simple in-place comparison sort that is inefficient on large collections.
- **InsertionSort:** This algorithm inserts each element into its proper place in the list. The easiest implementation uses two lists, one is the source

Appendix A. Computer Science Fundamentals

Table A.2.: SelectionSort Complexity

Time complexity	
Best case:	$\mathcal{O}(n^2)$
Worst case:	$\mathcal{O}(n^2)$
Average case:	$\mathcal{O}(n^2)$
Space complexity	$\mathcal{O}(1)$

list with the unsorted elements, and the other is the final sorted list. To save space, most of the time an implementation is used where the current element is moved behind the sorted elements and continuously swapped with its preceding element until it is in the proper place. InsertionSort performs about twice as well as BubbleSort, but its general complexity is $\mathcal{O}(n^2)$.

Table A.3.: InsertionSort Complexity

Time complexity	
Best case:	$\mathcal{O}(n)$
Worst case:	$\mathcal{O}(n^2)$
Average case:	$\mathcal{O}(n^2)$
Space complexity	$\mathcal{O}(1)$

- **ShellSort:** The basic idea is to make the collection almost sorted by breaking the collection into sublists, sort them, then combine the sublists again. It starts by choosing a gap sequence to sort pairs of elements that are far apart, then goes on by continuously reducing the gap. After that, the task is finished by InsertionSort because of its good runtime for mostly sorted lists. It is a good choice for large datasets. ShellSort is difficult to analyze because it heavily depends on the chosen gap sequence.

Table A.4.: ShellSort Complexity

Time complexity	
Best case:	$\mathcal{O}(n * \log n)$
Worst case (worst known gap sequence):	$\mathcal{O}(n^2)$
Average case:	depends on gap sequence
Space complexity	$\mathcal{O}(1)$

- **MergeSort:** MergeSort is an example of the divide-and-conquer style of sorting algorithms. It recursively breaks the list in two halves, sorts the

sublists and recombines them. It does a lesser number of comparisons than many other algorithms, but the space complexity is high.

Table A.5.: MergeSort Complexity

Time complexity	
Best case:	$\mathcal{O}(n * \log n)$
Worst case:	$\mathcal{O}(n * \log n)$
Average case:	$\mathcal{O}(n * \log n)$
Space complexity	
	$\mathcal{O}(n)$

- **QuickSort:** QuickSort is also a form of divide-and-conquer but with a different strategy than MergeSort. It is recursive and efficient but unstable, which means that the relative order of elements is not preserved. QuickSort chooses a pivot element from the original list. All elements that are smaller than the pivot element are put into the left sublist, whereas the bigger elements are put into the right sublist. The above steps are reapplied recursively, and the sublists get sorted. QuickSort has good average speed but poor worst-case performance.

Table A.6.: QuickSort Complexity

Time complexity	
Best case:	$\mathcal{O}(n * \log n)$
Worst case:	$\mathcal{O}(n^2)$
Average case:	$\mathcal{O}(n * \log n)$
Space complexity	
	$\mathcal{O}(\log n)$

- **HeapSort:** As the name suggests it uses a heap data structure which is binary tree-based. It is built so that either the biggest item, in a max heap, or the smallest item, in a min-heap, can be quickly accessed. In the beginning, a heap is built out of the unsorted array. Next, the first element is swapped with the last element and the heap size is reduced by one. After that, the heap is rebuilt, and the previously mentioned steps are repeated until the heap size is 1 and there is a sorted array. HeapSort is a good choice for large data because of it being non-recursive and in-place, but it works slower than divide-and-conquer algorithms with the same time complexity.
- **RadixSort:** RadixSort is a non-comparison-based, stable and out-of-place sorting algorithm which is based on BucketSort or CountingSort. It is assumed that the unsorted data only consists of keys with characters of an ending alphabet. For example, strings with characters from a to z, or

Appendix A. Computer Science Fundamentals

Table A.7.: HeapSort Complexity

Time complexity	
Best case:	$\mathcal{O}(n * \log n)$
Worst case:	$\mathcal{O}(n * \log n)$
Average case:	$\mathcal{O}(n * \log n)$
Space complexity	$\mathcal{O}(1)$

decimal numbers with base 10. It can be implemented based on least significant digit (LSD), which means that it starts from the least significant digit and moves towards the most significant digit (MSD), or the other way around. In the beginning, it takes the LSD of each key, then the keys are grouped based on that digit, while also taking the original order of keys into account, which makes RadixSort a stable sort. All the steps are repeated for each MSD. The grouping process is done by using BucketSort or CountingSort.

Table A.8.: RadixSort Complexity

Time complexity	
Worst case:	$\mathcal{O}(w * n)$... n is size of keys, w is length of keys
Space complexity	$\mathcal{O}(w + n)$

- **GnomeSort:** GnomeSort is a very simple and stable comparison-based sorting algorithm. It starts at the second element and always compares the current element with the previous one. If they are in the correct order, it will move one position ahead. Otherwise, it will swap the elements and go one step back. After that, the comparison procedure starts again, and everything gets repeated until the end of the array is reached. GnomeSort is good if the list is almost sorted but has a bad average and worst-case performance.

Table A.9.: GnomeSort Complexity

Time complexity	
Best case:	$\mathcal{O}(n)$
Worst case:	$\mathcal{O}(n^2)$
Average case:	$\mathcal{O}(n^2)$
Space complexity	$\mathcal{O}(1)$

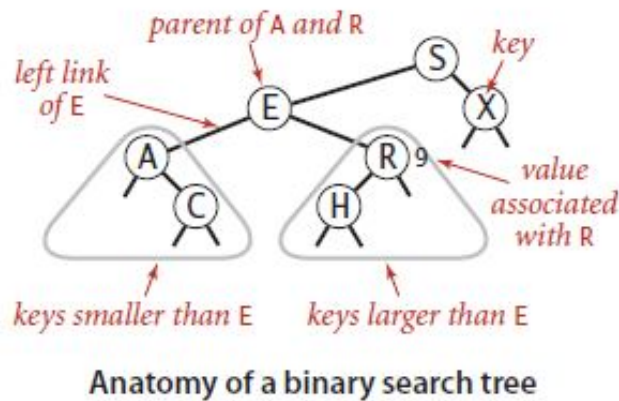


Figure A.5.: Anatomy of a Binary Search Tree (Sedgewick & Wayne, 2011)

A.2.2. Binary Search Trees

Dictionaries can be implemented based on sorted, or unsorted lists, however, both have their drawbacks. If we want to realize insertion in an unsorted list, it can be done quickly by simply putting a new element at the end of the list. But if we want to search a certain element, we have to go through all elements at worst case. Using sorted lists also brings problems. By having linked lists, we gain no speed when searching. If we use sorted array-based lists, searching can be done quickly with the binary search, but the insertion of elements now requires us to shift many other elements. A binary search tree (BST) provides an improved solution to it (Shaffer, 2013). The definition of a BST is explained by Sedgewick and Wayne (2011) as follows: "A binary search tree (BST) is a binary tree where each node has a Comparable key (and an associated value) and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node's left subtree and smaller than the keys in all nodes in that node's right subtree". Figure A.5 shows the anatomy of a binary search tree. A real-world scenario about the application of binary search trees occurs in the *RFC 791 Internet Protocol*. It requires that each packet sent by a host must contain a 16-bit identification field. As long as the datagram is active, the identification field needs to be unique for that source-destination pair (Postel, 1981). The Linux kernel remembers it by utilizing AVL trees, which are binary trees with an additional property that the height of the two subtrees of a node differ at most by one (Cormen, Leiserson, Rivest, & Stein, 2009). They are indexed by IP addresses and used as a second-level cache (Pfaff, 2004).

The basic operations on binary search trees include search, insertion, and deletion. Shaffer (2013) explains them as follows:

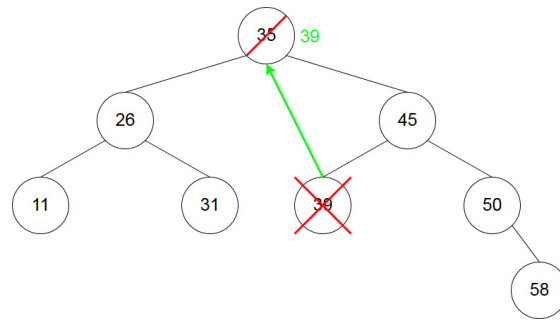


Figure A.6.: Deletion of a Node with Two Children

- **Search:** To find a node with key-value K , we have to begin at the root node and compare their keys. If they are equal, we are done. Otherwise, we have to go further down the tree. The good thing is that we only need to keep searching in one subtree because of the anatomy of BSTs. If K is bigger than the root node key, we take the right child node next. Otherwise, we take the left child node and repeat the above steps. If a leaf node is reached without encountering K , then no element in the tree exists with key K .
- **Insertion:** To insert a node with key-value K , we browse through the tree on the same principle as searching. As soon as we reach a leaf node, we can insert the new node, as a left child if the K is smaller and as a right child if K is bigger than the leaf nodes key.
- **Deletion:** Deleting a node from a BST is a bit trickier than the other two operations. First, we search the key in the tree. Then we have to figure out if this node has two children, only one child or is a leaf because it depends on which further operations we have to execute when removing the node.
 1. Leaf: If our node is a leaf node, we can remove it from the tree without further action.
 2. One Child: Replace the node by its child and delete the node.
 3. Two Children: In this case, we have to find the inorder successor of the node, which is the smallest element of the right subtree. Then replace the node with the inorder successor and delete it.

In Figure A.6, we can see the removal of a node with two children. Node with key 35 gets deleted. The fact that it has two children means that we need to find the inorder successor, which is key 39 in this case. Node with key 35 gets deleted, and Node with key 39 takes its position.

A.2.3. Pathfinding

The problem of finding the best path is a comprehensive issue in many areas be it computer games, artificial intelligence, navigation systems or computer simulations. A lot of fields have specific algorithms in place for reasons like performance or edge cases. However, most of the time, standard pathfinding algorithms can also be used successfully for common purposes. A definition of pathfinding would be that it is a process of finding a way between a start- and endpoint in a specific environment. Most of the time, the goal is to find the best path whereas in this context "best" can have various criteria like it being the shortest path, simplest path or cheapest path. (Zarembko & Kodors, 2015). Many computer games rely on pathfinding and have to face several problems. A challenging fact is that it needs to be solved in real-time with limited availability of memory and CPU power (Botea, Müller, & Schaeffer, 2004). A common genre where pathfinding is an important aspect is real-time strategy games. Most of the time, they use algorithms like A* because it can find the best possible way between two points in a reasonable short time frame. However, original A* does not support dynamic environments that are used for real-time strategy games. While a unit is travelling upon a previously calculated path, the environment can change in the meantime, for example, the path may be blocked, and it is necessary to recalculate the path. (Hagelbäck, 2016).

Figure A.7 shows a map of the game Starcraft as a big grid which is divided into many little squares. When the move command for a unit is issued (usually right-click) then the pathfinding algorithm is executed, and the unit walks along the resulting path one square at the time. At each square, it asks if the next square is occupied and either, continues to move, or waits for a very short time to check again. If the current path is still occupied, then a new path is generated by the algorithm. Now the new path could be a long roundabout way. Because of this functionality, the best way was to spam click the move command, so that every time the current best path is generated (Wyatt, 2013). To tackle problems like that Silver (2010) has presented three new algorithms, Koenig and Likhachev (2006) explain a way of accelerating A* by updating heuristics between searches, and Botea et al. (2004) use a hierarchical search to reduce complexity. Usually, a grid is overlaid over a map or region. Then some graph search is utilized to calculate the best path. Most of the time the grid consists of rectangles, called tiles, but it can also be Octa-based or Hexa-based. Most of the time, some form of A* is used to gain the best results (Yap, 2002). In this project, we focus on discussing the functionality of basic pathfinding algorithms on a tile-based grid, described by Sedgewick and Wayne (2011), Skiena (1998), Graham, McCabe, and Sheridan (2003), and Dijkstra (1959), which are used for our visualizations.

Appendix A. Computer Science Fundamentals



Figure A.7.: Grid Representation of a Map in Starcraft (Striketactics, 2017)

- **Breadth First Search:** Breadth First Search is often used to calculate the shortest path of unweighted graphs. It starts with the source node and explores equally in all directions. Every non-visited neighbour is added to the frontier, which is like an expanding ring, and marked as visited. BFS visits everything on the map. To be able to find the path, every node needs to remember its previous node. If information about the whole map is not needed, an early exit variant makes sure that BFS stops as soon as the destination point is reached.
- **Dijkstra's Algorithm:** Dijkstra's Algorithm, or also called Uniform Cost Search, is similar to BFS with the big difference that it tracks movement costs. Instead of adding a neighbour to the frontier if it has never been visited, the logic behind Dijkstra's Algorithm is to add it if the location is better than the previous best location. Furthermore, a priority queue is used, which changes the way the frontier expands so that more promising locations are visited earlier.
- **Greedy Best First Search:** Often it is the case to find the path to only one destination. A heuristic function is introduced that computes the proximity of the starting point to the endpoint, to make sure the frontier expands towards the destination. While Dijkstra's Algorithm uses the actual distance from the starting point to order the priority queue, Greedy Best First Search uses the estimated distance to the destination. The frontier moves quickly towards the destination using this method, but it has problems with complex maps, often not finding the best path, as can be seen in Figure A.8.
- **A*:** A* principally combines the positive aspects of Dijkstra's Algorithm and Greedy Best First Search and disposes of their inefficient features. Dijkstra's Algorithm can find the shortest path but spends time exploring unpromising locations. On the other hand, Greedy Best First Search quickly explores promising directions, but in further consequence often doesn't find the shortest path. So A* both considers the actual distance from the start, and the estimated distance to the destination point.

Bibliography

- Abari, O. (2017). Enabling high-quality untethered virtual reality. In *Proceedings of the 1st acm workshop on millimeter-wave networks and sensing systems 2017* (pp. 49–49). mmNets '17. Snowbird, Utah, USA: ACM. doi:10.1145/3130242.3131494
- Aho, A. V., Hopcroft, J. E., & Ullman, J. (1983). *Data structures and algorithms* (1st). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Ainsworth, S. (2008). How do animations influence learning?
- Akbulut, A., Catal, C., & Yildiz, B. (2018). On the effectiveness of virtual reality in the education of software engineering. *Computer Applications in Engineering Education*. doi:10.1002/cae.21935
- AlgoRythmics. (2011). [Online; accessed May 28, 2019]. Retrieved from <https://www.youtube.com/user/AlgoRythmics>
- Alnoukari, M. (2013). Simulation for computer sciences education. *Communications of the ACS*, 6.
- Amorim, J. A., Hendrix, M., Andler, S. F., & Gustavsson, P. M. (2013). Gamified training for cyber defence: Methods and automated tools for situation and threat assessment. doi:10.14339/STO-MP-MSG-111
- Angulo, A. & Vásquez de Velasco, G. (2013). Immersive simulation of architectural spatial experiences. (pp. 495–499). doi:10.5151/despro-sigradi2013-0095
- Ashkenas, J. (2009). Coffeescript. Retrieved from coffeescript.org
- Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *J. Usability Studies*, 4(3), 114–123. Retrieved from <http://dl.acm.org/citation.cfm?id=2835587.2835589>
- Barr, R., Haas, Z. J., Van Renesse, R., Tamtoro, K., Viglietta, B. S., Lin, C., ... Cheung, E. (2004). Swans scalable wireless ad hoc network simulator. [Online; accessed April 10, 2019]. Retrieved from <http://jist.ece.cornell.edu/index.html>
- Ben-Ari, M. (1998). Constructivism in computer science education. *SIGCSE Bull.* 30(1), 257–261. doi:10.1145/274790.274308
- Botea, A., Müller, M., & Schaeffer, J. (2004). Near optimal hierarchical path-finding. *Journal of Game Development*, 1, 7–28.
- Boticki, I., Barisic, A., Martin, S., & Drljevic, N. (2012). Teaching and learning computer science sorting algorithms with mobile devices: A case study.

Bibliography

- Computer Applications in Engineering Education*, 21(S1), E41–E50. Retrieved from <https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.21561>
- Bouchez-Tichadou, F. (2018). Problem solving to teach advanced algorithms in heterogeneous groups. In *Proceedings of the 23rd annual acm conference on innovation and technology in computer science education* (pp. 200–205). ITiCSE 2018. Larnaca, Cyprus: ACM. doi:10.1145/3197091.3197147
- Boulos, M. N. K., Hetherington, L., & Wheeler, S. (2007). Second life: An overview of the potential of 3-d virtual worlds in medical and health education. *Health Information & Libraries Journal*, 24(4), 233–245. doi:10.1111/j.1471-1842.2007.00733.x. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1471-1842.2007.00733.x>
- Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). How people learn: Brain, mind, experience, and school. Washington DC: National Academy Press.
- Brooke, J. (1996). Sus: A quick and dirty usability scale. *Usability Evaluation in Industry*, 194, 189–194.
- Burbaite, R., Stuikeys, V., & Marcinkevicius, R. (2012). The lego nxt robot-based e-learning environment to teach computer science topics. *Elektronika ir Elektrotechnika*, 18(9). Retrieved from <http://eejournal.ktu.lt/index.php/elt/article/view/2825>
- Byrne, C. & Furness, T. A. (1994). Virtual reality and education. In *Proceedings of the ifip tc3/wg3.5 international working conference on exploring a new partnership: Children, teachers and technology* (pp. 181–189). New York, NY, USA: Elsevier Science Inc. Retrieved from <http://dl.acm.org/citation.cfm?id=647119.717198>
- Byrne, C. M. (1996). *Water on tap: The use of virtual reality as an educational tool* (Doctoral dissertation, Seattle, WA, USA). UMI Order No. GAX96-30063.
- Carroll, S. J., Paine, F. T., & Ivancevich, J. J. (1972). The relative effectiveness of training methods: Expert opinion and research. *Personnel Psychology*, 25.
- Cashman, E. & Eschenbach, E. (2003). Active learning with web technology - just in time. (Vol. 1, T3F–9). doi:10.1109/FIE.2003.1263352
- Cengiz Gulek, J. & Demirtas, H. (2005). Learning with technology: The impact of laptop use on student achievement. *Journal of Technology, Learning, and Assessment*, 3.
- Chandramouli, M., Zahraee, M., & Winer, C. (2014). A fun-learning approach to programming: An adaptive virtual reality (vr) platform to teach programming to engineering students. *IEEE International Conference on Electro/Information Technology*, 581–586.
- Chickering, A. W. & Gamson, Z. F. (1987). Seven principles of good practice in undergraduate education. *AAHE Bulletin*, 3–7.

Bibliography

- Clark, R. C. & Mayer, R. E. (2007). *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning* (2nd). Pfeiffer & Company.
- Clément, J., Hien Nguyen Thi, T., & Vallée, B. (2013). A general framework for the realistic analysis of sorting and searching algorithms. application to some popular algorithms. *Leibniz International Proceedings in Informatics, LIPIcs*, 20. doi:10.4230/LIPIcs.STACS.2013.598
- Clifford, M. (2012). Top 20 uses of virtual worlds in education. [Online; accessed July 15, 2019]. Retrieved from <https://schoolleadership20.com/forum/topics/top-20-uses-of-virtual-worlds-in-education>
- CodeMonkey Website. (2016). Codemonkey. [Online; accessed April 12, 2019]. Retrieved from <https://www.playcodemonkey.com>
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms, third edition* (3rd). The MIT Press.
- Crespo, R., García, R., & Quiroz, S. (2015). Virtual reality simulator for robotics learning. In *2015 international conference on interactive collaborative and blended learning (icbl)* (pp. 61–65). doi:10.1109/ICBL.2015.7387635
- Curcio, I. D., Dipace, A., & Norlund, A. (2016). Virtual realities and education. *Research on Education and Media*, 8. doi:10.1515/rem-2016-0019
- Dawson, J. Q., Allen, M., Campbell, A., & Valair, A. (2018). Designing an introductory programming course to improve non-majors' experiences. In *Proceedings of the 49th acm technical symposium on computer science education* (pp. 26–31). SIGCSE '18. Baltimore, Maryland, USA: ACM. doi:10.1145/3159450.3159548
- de Jong, T. & van Joolingen, W. R. (1998). Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, 68(2), 179–201.
- Demuth. (1956). *H. electronic data sorting*. Stanford University.
- Denk, N., Gabriel, S., Wernbacher, T., Pfeiffer, A., & Mayerhofer, E. (2018). Game-based learning im unterricht. [Online; accessed July 15, 2019]. Julius Raab Stiftung. Retrieved from https://www.saferinternet.at/fileadmin/categorized/Materialien/Vademecum_Game_Based_Learning_fuer_den_Unterricht.pdf
- Denning, P. J. (2005). Is computer science science? *Commun. ACM*, 48(4), 27–31. doi:10.1145/1053291.1053309
- Dijkstra, E. (1959). *A note on two problems in connexion with graphs*. Springer-Verlag. doi:10.1007/BF01386390
- Dillenbourg, P. (1999). Collaborative learning: Cognitive and computational approaches. *Computers & Education*, 1. doi:10.1016/S0360-1315(00)00011-7

Bibliography

- Du, X. & Arya, A. (2015). Design and evaluation of a learning assistant system with optical head-mounted display (ohmd). In P. Zaphiris & A. Ioannou (Eds.), *Learning and collaboration technologies* (pp. 75–86). Cham: Springer International Publishing.
- Durrani, O. K., Shreelakshmi, V., & Shetty, S. (2012). Analysis and determination of asymptotic behavior range for popular sorting algorithms.
- Elmaleh, J. & Shankararaman, V. (2017). Improving student learning in an introductory programming course using flipped classroom and competency framework. In *2017 IEEE Global Engineering Education Conference (Educon)* (pp. 49–55). doi:10.1109/EDUCON.2017.7942823
- Extend Reality Ltd. (2018). Vrtk - virtual reality toolkit. Retrieved from <https://www.vrtk.io/>
- Fernandez, M. (2017). Augmented-virtual reality: How to improve education systems. *Higher Learning Research Communications*, 7, 1. doi:10.18870/hlrc.v7i1.373
- Fox, C. M. & Brockmyer, J. H. (2013). The development of the game engagement questionnaire: A measure of engagement in video game playing: Response to reviews. *Interacting with Computers*, 25(4), 290–293. doi:10.1093/iwc/iwto03
- FreeCodeCamp. (2014). Retrieved from <https://www.freecodecamp.org/>
- Freina, L. & Ott, M. (2015). A literature review on immersive virtual reality in education: State of the art and perspectives.
- Gavrin, A. (2006). Just-in-time teaching. *Published in Metropolitan Universities*, 17, 9–18.
- Gavrin, A., X. Watt, J., Marrs, K., & E. Blake, R. (2004). Just-in-time teaching (jitt): Using the web to enhance classroom learning. *Computers in Education Journal*, 14, 51–59.
- Geck, G., Ljulin, A., Peter, S., Schmidt, J., Vehlken, F., & Zeume, T. (2018). Introduction to iltis: An interactive, web-based system for teaching logic. doi:10.1145/3197091.3197095
- Ghezzi, C. & Mandrioli, D. (2006). The challenges of software engineering education. In P. Inverardi & M. Jazayeri (Eds.), *Software engineering education in the modern age* (pp. 115–127). Berlin, Heidelberg: Springer Berlin Heidelberg.
- GitHub Pages. (2008). Retrieved from <https://pages.github.com/>
- Graefe, G. (2006). Implementing sorting in database systems. *ACM Comput. Surv.* 38. doi:10.1145/1132960.1132964
- Graham, R., McCabe, H., & Sheridan, S. (2003). Pathfinding in computer games. *The ITB Journal*, 4. doi:10.21427/D7ZQ9J

Bibliography

- Griffiths, M. (2002). The educational benefits of videogames. *Education and Health, 20*, 47–51.
- Gütl, C. [C.] & Pirker, J. [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* (pp. 58–66). doi:10.1109/ICL.2011.6059548
- Gütl, C. [Christian]. (2011). The support of virtual 3d worlds for enhancing collaboration in learning settings. *IGI Global 2011*, 278–299.
- Hagelbäck, J. (2016). Hybrid pathfinding in starcraft. *IEEE Transactions on Computational Intelligence and AI in Games, 8*, 319–324.
- Hammad, J. (2015). A comparative study between various sorting algorithms. *IJCSNS International Journal of Computer Science and Network Security, 153*, 11.
- Harteveld, C., Smith, G., Carmichael, G., Gee, E., & Stewart-Gardiner, C. (2014). *A design-focused analysis of games teaching computer science*.
- Hazzan, O., Lapidot, T., & Ragonis, N. (2014). Guide to teaching computer science: An activity-based approach.
- Holton, D. (2010). How people learn with computer simulations. *ITLS Faculty Publications*. doi:10.4018/978-1-60566-782-9.ch029
- Horn, B., Clark, C., Strom, O., Chao, H., Stahl, A., Harteveld, C., ... Folajimi, Y. (2016). *Design insights into the creation and evaluation of a computer science educational game*. doi:10.1145/2839509.2844656
- HTC Vive. (2016). Retrieved from <https://www.vive.com/de/>
- Huguen, P. (2018). [Online; accessed May 13, 2019]. Retrieved from <https://www.nbcnews.com/mach/science/what-vr-devices-apps-turn-real-world-virtual-ncna857001>
- Ibáñez, M., García Rueda, J. J., Galán, S., Maroto, D., Morillo, D., & Delgado-Kloos, C. (2011). Design and implementation of a 3d multi-user virtual world for language learning. *Educational Technology & Society, 14*, 2–10.
- Irvine, C. E., Thompson, M. F., & Allen, K. (2005). Cybercieve: Gaming for information assurance. *IEEE Security Privacy, 3*(3), 61–64. doi:10.1109/MSP.2005.64
- JavaScript. (1995). A high-level, interpreted programming language. [Online; accessed April 15, 2019]. Retrieved from <https://www.javascript.com/about>
- Jayaram, S., Connacher, H. I., & Lyons, K. W. (1997). Virtual assembly using virtual reality techniques. *Computer-Aided Design, 29*(8), 575–584. Virtual Reality. doi:[https://doi.org/10.1016/S0010-4485\(96\)00094-2](https://doi.org/10.1016/S0010-4485(96)00094-2)
- Jin, G., Tu, M., Kim, T.-H., Heffron, J., & White, J. (2018). Game based cybersecurity training for high school students. In *Proceedings of the 49th acm*

Bibliography

- technical symposium on computer science education* (pp. 68–73). SIGCSE '18. Baltimore, Maryland, USA: ACM. doi:10.1145/3159450.3159591
- Jint. (2014). [Online; accessed April 15, 2019]. Retrieved from <https://github.com/sebastienros/jint>
- Johnson, W. L., Rickel, J. W., & Lester, J. C. (2000). Animated pedagogical agents: Face-to-face interaction in interactive learning environments. *International Journal of Artificial Intelligence in Education*, 11, 47–.
- Jones, J. S. (1987). Participatory teaching methods in computer science. *SIGCSE Bull.* 19(1), 155–160. doi:10.1145/31726.31751
- Kafai, Y. (2001). The educational potential of electronic games: From games-to-teach to games-to-learn.
- Kang, K. & Roland, R. J. (2007). Military simulation. In *Handbook of simulation* (Chap. 19, pp. 645–658). John Wiley & Sons, Ltd. doi:10.1002/9780470172445.ch19. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470172445.ch19>
- Katai, Z. & Toth, L. (2010). Technologically and artistically enhanced multi-sensory computer-programming education. *Teaching and Teacher Education*, 26, 244–251. doi:10.1016/j.tate.2009.04.012
- Kathleen Smetana, L. & Bell, R. (2012). Computer simulations to support science instruction and learning: A critical review of the literature. *International Journal of Science Education*, 34. doi:10.1080/09500693.2011.605182
- Kay, R. & Loverock, S. (2008). Assessing emotions related to learning new software: The computer emotion scale. *Computers in Human Behavior*, 24, 1605–1623. doi:10.1016/j.chb.2007.06.002
- Kerren, A. & Stasko, J. (2002). *Algorithm animation*. Springer-Verlag.
- Klopfer, E., Osterweil, S., & Salen, K. (2009). Moving learning games forward.
- Koenig, S. & Likhachev, M. (2006). Real-time adaptive a*. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems* (pp. 281–288). AAMAS '06. Hakodate, Japan: ACM. doi:10.1145/1160633.1160682
- Kumar, C. (2017). A new frontier: How can you profit from augmented and virtual reality? [Online; accessed May 10, 2019]. Retrieved from <https://www.business.com/articles/how-can-you-profit-from-augmented-and-virtual-reality/>
- LaValle, S. M. (2016). *Virtual Reality*. Cambridge University Press.
- Le Compte, A., Elizondo, D., & Watson, T. (2015). A renewed approach to serious games for cyber security. In *2015 7th international conference on cyber conflict: Architectures in cyberspace* (pp. 203–216). doi:10.1109/CYCON.2015.7158478
- LeanTween. (2016). Leantween, an efficient animation engine for unity. Retrieved from <https://github.com/dentedpixel/LeanTween>

Bibliography

- LimeSurvey. (2003). [Online; accessed July 20, 2019]. Retrieved from <https://www.limesurvey.org>
- Lowe, R. (2003). Animation and learning: Selective processing of information in dynamic graphics. *Learning and Instruction, 13*, 157–176. doi:10.1016/S0959-4752(02)00018-X
- Madathil, K. C., Frady, K., Hartley, R. S., Bertrand, J. W., Alfred, M., & Gramopadhye, A. K. (2017). An empirical study investigating the effectiveness of integrating virtual reality-based case studies into an online asynchronous learning environment. *Computers in Education, 8*.
- Manivannan, M. S. (2007). Simulation of logistics and transportation systems. In *Handbook of simulation* (Chap. 16, pp. 571–604). John Wiley & Sons, Ltd. doi:10.1002/9780470172445.ch16. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470172445.ch16>
- Marrs, K., Gavrin, A., & Novak, G. M. (2004). Just-in-time teaching in biology: Creating an active learner classroom using the internet. *Cell Biology Education, 3*, 1, 49–61.
- Mayer, R. & Pilegard, C. (2014). Principles for managing essential processing in multimedia learning: Segmenting, pre-training, and modality principles. (pp. 316–344). doi:10.1017/CBO9781139547369.016
- McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G., & Mander, K. (2005). Grand challenges in computing: Education—a summary. *The Computer Journal, 48*(1), 42–48. doi:10.1093/comjnl/bxh064
- McGuire, F. (2007). Simulation in healthcare. In *Handbook of simulation* (Chap. 17, pp. 605–627). John Wiley & Sons, Ltd. doi:10.1002/9780470172445.ch17. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470172445.ch17>
- Meinel, C. & Theobald, T. (1998). *Algorithms and data structures in vlsi design* (1st). Berlin, Heidelberg: Springer-Verlag.
- Miller, A., Allison, C., Mccaffery, J., Sturgeon, T., Nicoll, R., Getchell, K., ... Oliver, I. (2010). Virtual worlds for computer science education. *11 th Annual Conference of the Subject Centre for Information and Computer Sciences*, 239.
- Monahan, T., McArdle, G., & Bertolotto, M. (2008). Virtual reality for collaborative e-learning. *Computers & Education, 50*, 1339–1353. doi:10.1016/j.compedu.2006.12.008
- Moorthy, K., Vincent, C., & Darzi, A. (2005). Simulation based training. *BMJ, 330*(7490), 493–494. doi:10.1136/bmj.330.7490.493. eprint: <https://www.bmj.com/content/330/7490/493.full.pdf>
- Moreno, R. & Ortegado-Layne, L. (2008). Do classroom exemplars promote the application of principles in teacher education? a comparison of videos,

Bibliography

- animations, and narratives. *Educational Technology Research and Development*, 56(4), 449–465. doi:10.1007/s11423-006-9027-0
- N. Silva, Y., Nieuwenhuys, A., G. Schenk, T., & Symons, A. (2018). Dbsnap++: Creating data-driven programs by snapping blocks. (pp. 170–175). doi:10.1145/3197091.3197114
- Newman, I., Daniels, M., & Faulkner, X. (2003). Open ended group projects a 'tool' for more effective teaching. In *Proceedings of the fifth Australasian conference on computing education - volume 20* (pp. 95–103). ACE '03. Adelaide, Australia: Australian Computer Society, Inc. Retrieved from <http://dl.acm.org/citation.cfm?id=858403.858415>
- Novak, G. M., Gavrín, A., & Wolfgang, C. (1999). *Just-in-time teaching: Blending active learning with web technology* (1st). Upper Saddle River, NJ, USA: Prentice Hall PTR.
- Oculus Rift. (2016). Retrieved from <https://www.oculus.com/rift/>
- Oh Navarro, E. & van der Hoek, A. (2004). Simse: An interactive simulation game for software engineering education. (pp. 12–17).
- Oh Navarro, E. & van der Hoek, A. (2010). Simse. [Online; accessed April 10, 2019]. Retrieved from <https://www.ics.uci.edu/~emilyo/SimSE/Assets/images/screenshotSm.png>
- O'Hara, K. J. & Kay, J. S. (2003). Open source software and computer science education. *J. Comput. Sci. Coll.* 18(3), 1–7. Retrieved from <http://dl.acm.org/citation.cfm?id=771712.771716>
- O'Keefe, R. M. (1987). What is visual interactive simulation? (and is there a methodology for doing it right?) In *Proceedings of the 19th conference on winter simulation* (pp. 461–464). WSC '87. Atlanta, Georgia, USA: ACM. doi:10.1145/318371.318635
- Open Simulator. (2007). [Online; accessed July 20, 2019]. Retrieved from http://opensimulator.org/wiki/Main_Page
- Open Wonderland. (2007). [Online; accessed July 20, 2019]. Retrieved from <http://openwonderland.org>
- Pange, J. P. (2003). Teaching probabilities and statistics to preschool children. *Information Technology in Childhood Education Annual*, 1.
- Patel, A. (2003). Greedy best first search - non optimal path. [Online; accessed April 09, 2019]. Retrieved from <http://theory.stanford.edu/~amitp/game-programming/a-star/best-first-search-trap.png>
- Peddycord-Liu, Z., Cody, C., Michelle Barnes, T., F. Lynch, C., & Rutherford, T. (2017). The antecedents of and associations with elective replay in an educational game: Is replay worth it?
- Pellas, N. (2015). Open source virtual worlds for e-learning.

Bibliography

- Perry, E. L. & Ballou, D. J. (1997). The role of work, play, and fun in micro-computer software training. *SIGMIS Database*, 28(2), 93–112. doi:10.1145/264701.264708
- Pfaff, B. (2004). Performance analysis of bst's in system software. In *Proceedings of the joint international conference on measurement and modeling of computer systems* (pp. 410–411). SIGMETRICS '04/Performance '04. New York, NY, USA: ACM. doi:10.1145/1005686.1005742
- Pirker, J. [Johanna]. (2013). The virtual teal world-an interactive and collaborative virtual world environment for physics education.
- Pirker, J. [Johanna] & Gütl, C. [Christian]. (2015). Virtual worlds for 3d visualizations. *Workshop proceedings of the 11th international conference on intelligent environments*, 265–272.
- Postel, J. (1981). Rfc 791: Internet protocol.
- PriorityQueue. (2013). Retrieved from <https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp>
- Putnam, R. T., Sleeman, D., Baxter, J. A., & Kuspa, L. K. (1986). A summary of misconceptions of high school basic programmers. *Journal of Educational Computing Research*, 2(4), 459–472. doi:10.2190/FGN9-DJ2F-86V8-3FAU. eprint: <https://doi.org/10.2190/FGN9-DJ2F-86V8-3FAU>
- Rohrer, M. W. (2007). Simulation of manufacturing and material handling systems. In *Handbook of simulation* (Chap. 14, pp. 517–545). John Wiley & Sons, Ltd. doi:10.1002/9780470172445.ch14. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470172445.ch14>
- Russell, R., J. and Kozma, Zohdy, T., M. and Susskind, Becker, D., & Russell, C. (2000). Smv:chem (simultaneous multiple representations in chemistry). New York: John Wiley.
- Sangin, M., Molinari, G., Dillenbourg, P., Rebetez, C., & Bétrancourt, M. (2006). Collaborative learning with animated pictures: The role of verbalizations. In *Proceedings of the 7th international conference on learning sciences* (pp. 667–673). ICLS '06. Bloomington, Indiana: International Society of the Learning Sciences. Retrieved from <http://dl.acm.org/citation.cfm?id=1150034.1150131>
- Sarabando, C., Cravino, J., & Soares, A. (2014). Contribution of a computer simulation to students' learning of the physics concepts of weight and mass. *Procedia Technology*, 13, 112–121. doi:10.1016/j.protcy.2014.02.015
- Scheiter, K., Gerjets, P., & Catrambone, R. (2006). Making the abstract concrete: Visualizing mathematical solution procedures. *Comput. Hum. Behav.* 22(1), 9–25. doi:10.1016/j.chb.2005.01.009
- Schor, J., Schor, I., & Pinchover, Y. (2014). Codemonkey. CodeMonkey Studios Inc. Retrieved from <https://www.playcodemonkey.com/>

Bibliography

- Second Life. (2003). [Online; accessed July 20, 2019]. Retrieved from <https://secondlife.com/>
- Sedgewick, R. & Wayne, K. (2011). *Algorithms, 4th edition*. Addison-Wesley.
- Shaffer, C. (2013). *Data structures and algorithm analysis. edition 3.2 (java version)*. Dover Publications.
- Silberman, M. (1996). *Active learning: 101 strategies to teach any subject*. Allyn and Bacon. Retrieved from https://books.google.at/books?id=9x9T2%5C_WEAM8C
- Silver, D. (2010). Cooperative pathfinding. In *Proceedings of the first aaai conference on artificial intelligence and interactive digital entertainment* (pp. 117–122). AIIDE'05. Marina del Rey, California: AAAI Press. Retrieved from <http://dl.acm.org/citation.cfm?id=3022473.3022494>
- Skiena, S. S. (1998). *The algorithm design manual*. Berlin, Heidelberg: Springer-Verlag.
- Slator, B. M., Hill, C., & Del Val, D. (2004). Teaching computer science with virtual worlds. *IEEE Transactions on Education*, 47(2), 269–275. doi:10.1109/TE.2004.825513
- Smith, R. D. (1999). *Simulation: The engine behind the virtual world*.
- Smith, R. D. (2002). Interactive simulation. [Online; accessed July 10, 2019]. Retrieved from https://www.modelbenders.com/ein5255/01_InteractiveSim.pdf
- Starting Point Project. (2012). Just-in-time teaching - step by step. [Online; accessed April 12, 2019]. Retrieved from https://d320goqmya1dw8.cloudfront.net/images/introgeo/justintime/jitt_step-by-step_650.jpg
- Steam VR. (2015). Retrieved from <https://www.steamvr.com/en/>
- Striketactics. (2017). Starcraft brood war pathfinding grid. [Online; accessed April 07, 2019]. Retrieved from http://striketactics.net/sites/default/files/starcraft1_grid_pathfinding.jpg
- Swisher, D. (2007). Does multimedia truly enhance learning? moving beyond the visual media bandwagon toward instructional effectiveness.
- TextMesh Pro. (2017). [Online; accessed April 15, 2019]. Retrieved from <https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126>
- Tucker, A. B. (1996). Strategic directions in computer science education. *ACM Comput. Surv.* 28(4), 836–845. doi:10.1145/242223.246876
- Ulgen, O. & Gunal, A. (2007). Simulation in the automobile industry. In *Handbook of simulation* (Chap. 15, pp. 547–570). John Wiley & Sons, Ltd. doi:10.1002/9780470172445.ch15. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470172445.ch15>

Bibliography

- Unity Asset Store. (2010). [Online; accessed April 15, 2019]. Retrieved from <https://assetstore.unity.com/>
- Unity3D. (2005). Cross-platform game engine developed by unity technologies. [Online; accessed April 15, 2019]. Retrieved from <http://unity3d.com>
- Unity-Jint. (2016). Unity-jint, a port of jint for unity. Retrieved from <https://github.com/splhack/unity-jint>
- Vesisenaho, M., Juntunen, M., Häkkinen, P., Pöysä-Tarhonen, J., Fagerlund, J., Miakush, I., & Parviainen, T. (2019). Virtual reality in education: Focus on the role of emotions and physiological reactivity. *Journal For Virtual Worlds Research*, 12. doi:10.4101/jvwr.v12i1.7329
- Visual Studio. (1997). [Online; accessed April 15, 2019]. Retrieved from <https://visualstudio.microsoft.com>
- Vogel, J. J., Vogel, D. S., Cannon-Bowers, J., Bowers, C. A., Muse, K., & Wright, M. (2006). Computer gaming and interactive simulations for learning: A meta-analysis. *Journal of Educational Computing Research*, 34(3), 229–243. doi:10.2190/FLHV-K4WA-WPVQ-HoYM. eprint: <https://doi.org/10.2190/FLHV-K4WA-WPVQ-HoYM>
- Voyiatzaki, E., Christakoudis, C., Margaritis, M., & Avouris, N. (2004). Teaching algorithms in secondary education: A collaborative approach.
- W. Newstrom, J. (1980). Evaluating the effectiveness of training methods. *Personnel Administrator*, 25.
- Wang, X. (2013). A comparative analysis of performance and cognition in multimedia learning between esb and nesb students in higher education. Swinburne University of Technology.
- Warschauer, M. (2007). The paradoxical future of digital learning. *Learning Inquiry*, 1(1), 41–49. doi:10.1007/s11519-007-0001-5
- Webb, M. (2005). Affordances of ict in science learning: Implications for an integrated pedagogy. *International Journal of Science Education - INT J SCI EDUC*, 27, 705–735. doi:10.1080/09500690500038520
- WebGL. (2011). A cross-platform, royalty-free web standard for a low-level 3d graphics api. [Online; accessed April 15, 2019]. Retrieved from <https://www.khronos.org/webgl/>
- Webster, R. (2014). Corrosion prevention and control training in an immersive virtual learning environment. *NACE - International Corrosion Conference Series*.
- Wenglinsky, H. (1998). Does it compute?: The relationship between educational technology and student achievement in mathematics. *Princeton, NJ : Policy Information Center, Educational Testing Service*.
- Wing, J. M. (2006). Computational thinking. *Communications of the Acm*, 49(3), 33–35.

Bibliography

- Wirth, N. (2004). *Algorithms and data structures. oberon version*. Eidgenössische Technische Hochschule Zürich.
- Wittwer, J. & Renkl, A. (2008). Why instructional explanations often do not work: A framework for understanding the effectiveness of instructional explanations. *Educational Psychologist - EDUC PSYCHOL*, 43, 49–64. doi:10.1080/00461520701756420
- Wolff, M. & Wills, L. (2000). Satsim: A superscalar architecture trace simulator using interactive animation. doi:10.1145/1275240.1275249
- Wyatt, P. (2013). The starcraft path-finding hack. [Online; accessed April 09, 2019]. Retrieved from <https://www.codeofhonor.com/blog/the-starcraft-path-finding-hack>
- Xiannong, M. (1998). Simulation. [Online; accessed July 15, 2019]. Retrieved from <https://www.eg.bucknell.edu/~xmeng/Course/CS6337/Note/master/>
- Xueqiao, X. (2011). Pathfinding visualization. [Online; accessed April 12, 2019]. Retrieved from <https://qiao.github.io/PathFinding.js/visual/>
- Yap, P. (2002). Grid-based path-finding. In R. Cohen & B. Spencer (Eds.), *Advances in artificial intelligence* (pp. 44–55). Berlin, Heidelberg: Springer Berlin Heidelberg.
- YouTube. (2005). Retrieved from <https://www.youtube.com>
- Zapponi, C. (2014). Sorting, an attempt to visualize and help to understand how some of the most famous sorting algorithms work. [Online; accessed April 12, 2019]. Retrieved from <http://sorting.at>
- Zarembo, I. & Kodors, S. (2015). Pathfinding algorithm efficiency analysis in 2d grid.