Alexander Steinmaurer

Revising a Game-Based Learning Platform for Computational Skills in Education

Diploma Thesis

to achieve the university degree of

Magister der Naturwissenschaften

Teacher training programme: Computer Science and Computer Science Management

submitted to

Graz University of Technology

Supervisor

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

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

Graz, December 2019

Alexander Steinmaurer

Überarbeitung einer spielebasierten Plattform für Computational Skills im Unterricht

Diplomarbeit

zur Erlangung des akademischen Grades

Magister der Naturwissenschaften

Unterrichtsfach: Informatik und Informatikmanagement

eingereicht an der

Technische Universität Graz

Betreuer

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

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

Graz, December 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 upoaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum	Unterschrift

Abstract

This thesis describes the further development and introduction of a pedagogical concept in sCool. The sCool system consists of two components: a mobile video game for students and a web application for educators. It was initially developed in 2017 in a cooperation between Graz University of Technology and Westminster University (UK). The general approach of the game is divided into two parts: concept-learning and practical working. In the first part of the game the students learn concepts in a textual way and in the second they have to apply the concepts using the Python programming language. The creation of the content can be done in a web application that is designed especially for educators. The combination of both components provides a highly adaptive tool to teach computational skills.

To receive a tool with both an encouraging game environment a learner-centered pedagogical concept some improvements are necessary. The thesis discuss the pre-evaluation that was conducted in two (academic) secondary schools in order to develop requirements and a concept for a further version. In terms of this pre-evaluation aspects like game engagement, emotions, and game-related questions were asked. Based on this findings the development of a revised sCool version started. This version extends the existing game environment with different game element to provide additional concepts in computational skill teaching. The development also covers the work on the web application to make the new concepts as highly adaptive as possible.

A central part of this thesis is the evaluation of the further developed sCool version. In four workshops different aspects of the game has been evaluated in order to target the appropriate person group and to figure out if the new version lead to a better understanding of the concepts. One of the evaluations has been conducted with ten computer science teachers to receive feedback about the overall pedagogical concept. The other three experiments were taken in a primary school, an academic secondary school and a university.

The evaluation showed that the pedagogical concept is a key element for successful education. In the scope of this evaluation it seems that the appropriate age group are 10-14 years old children since they enjoyed both playing the game and learning with sCool. The revised version of the game helps to teach more concepts and improved some usability issues in order to get a more playable game.

Zusammenfassung

Die vorliegende Diplomarbeit beschreibt die Weiterentwicklung und Einführung eines didaktischen Konzepts in sCool. Es besteht aus zwei Komponenten: einem mobilen Videospiel und einer Webanwendung. Ursprünglich wurde das Spiel 2017 in einer Zusammenarbeit zwischen der Technischen Universität Graz und der Westminster University entwickelt. Das Spiel gliedert sich in zwei Teile: einem Konzeptteil und einem praktischen Teil. Im Konzeptteil lernen die SchülerInnen verschiedene Konzepte kennen, die sie im zweiten Teil mithilfe der Programmiersprache Python anwenden können. Die Erstellung der Inhalte erfolgt in einer Webanwendung, die speziell für PädagogInnen entwickelt wurde. Die Kombination beider Komponenten bietet ein äußerst anpassungsfähiges Werkzeug, zur Vermittlung von Computational Skills.

Damit das Spiel eine möglichst motivierende Spielumgebung aber auch ein lernerzentriertes didaktisches Konzept zur Verfügung stellt, sind einige Anpassungen notwendig. Zu Beginn der Diplomarbeit wird eine Vorevaluierung an zwei Schulen der Sekundarstufe I beschrieben. Diese dient als Basis für die Entwicklung einer weiteren Version. Im Rahmen dieser Vorevaluierung werden unterschiedliche Aspekte wie zum Beispiel Engagment, Emotionen und spielbezogene Fragen untersucht. Basierend auf diesen Erkenntnissen wird mit der Weiterentwicklung der bestehenden Spieleumgebung begonnen. Es werden zusätzliche Spielelemente eingeführt, um neue Konzepte zu unterstützen. Die Weiterentwicklung umfasst außerdem die Arbeit an der Webanwendung, um die neuen Konzepte möglichst anpassungsfähig zu gestalten.

Ein zentraler Bestandteil dieser Arbeit ist die Evaluierung der neuen sCool-Version. In vier Workshops wurden verschiedene Aspekte des Spiels evaluiert, um die Zielgruppe herauszufinden und festzustellen, ob die neue Version zu einem besseren Verständnis der Konzepte führt. Eine der Evaluationen wurde mit zehn InformatiklehrerInnen durchgeführt, um Feedback zum didaktischen Gesamtkonzept zu erhalten. Die anderen drei Experimente wurden an einer Volksschule, einer Sekundarstufe I und einer Universität durchgeführt.

Die Evaluierung ergab, dass das pädagogische Konzept ein zentrales Element beim Erlernen von Computational Skills mithilfe von sCool ist. Die Evaluierung zeigte außerdem, dass die Zielgruppe zwischen 10 und 14 Jahre alt ist, da diese SchülerInnen sowohl Spaß am Spielen mit sCool hatten, aber auch gerne etwas Neues lernten. Die weiterentwickelte Version des Spiels hilft dabei, zusätzliche Konzepte zu vermitteln und durch außerdem wurden einige Usability-Probleme behoben, damit das Spiel noch ansprechender wird.

Acknowledgement

Firstly I would like to express the deepest appreciation to my supervisor Assoc. Prof. Dr.techn. Christian Gütl. His support, professional experience and advice during all project phases guided and motivated me through the whole thesis. I also want to thank him for the great opportunity to do some of the research in Melbourne.

During my research stay at the Royal Melbourne Institute of Technology Christopher and France Cheong gave me a warm welcome and supported the project by providing a meaningful input. Thank you for this great time in Australia.

Finally I want to give special thanks to my parents - Ingrid and Ernst - for supporting me with a matter of course through my whole study, my girlfriend Vanessa for her saintly patience and of course my best friends Thomas and Patrick.

Contents

List of Figures

Listings

List of Tables

Abbreviations

API Application Programming Interface

CAS Computing At School

CES Computer Emotion Scale

CSS Cascading Style Sheets

ECDL European Computer Driving License

ER Entity Relationship

GEQ Game Engagement Questionnaire

HP Health Points

HTML HyperText Markus Language

HTTP Hypertext Transfer Protocol

ICT Information and Communications Technology

JS JavaScript

JSON JavaScript Object Notation

LMS Learning Management System

MVC Model, View, Controller

orm object-relational mapping

PBL problem-based-learning

PEP8 Python Enhancement Proposals

REST Representational State Transfer

RMIT Royal Melbourne Institute of Technology

SIMS Situational Motivation Scale

SQL Structured Query Language

STEM Science, Technology, Engineering , Mathematics

UI User Interface

XP Experience Points

1 Introduction

"Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" (Wing, 2006). This quote by Jeannette Wing is often cited regarding computational thinking. Wing demands to extend the traditional cultural techniques by computational skills. Former US president Barack Obama goes one step further and wants everyone from kindergarden to high school to learn computer science (Smith, 2016). The importance of computational skills is emphasized by politics, industry, and education. These are key abilities in a century were the significance of data is higher than ever before.

The approach of A. Kojic et al. (2018) was to create an educational video game based on procedural content generation (PCG) algorithms to make an encouraging game play. By using PCG algorithms the map, game objects and sound can be generated in a pseudo-random way which makes it more replayable. The game was created as a mobile video game to make it highly accessible in schools and to support micro-teaching sequences (Steinmaurer, Pirker, and Gütl (2019b)). Initially, it was designed for STEM education but the focus of the video game is on programming and computational skills since it has a code editor where coding in the Python programming language is possible.

1.1 Aims and Objectives

The main purpose of this thesis is to continue the work on the mobile video game sCool in a pedagogical perspective. sCool should provide educators with the possibility to encourage and motivate students to learn computational skills and coding in particular. Since the game's learning content is highly adaptive, educators can create individual courses for their classes. To provide the teachers as many possibilities in content creation and support the players with an user-centered learning environment an appropriate pedagogical concept is required. This thesis is divided into three main phases where this process is presented:

1. Conducting a pre-evaluation of sCool version 1 and a first pedagogical concept in order to analyse usability, game engagement, emotions, and the pedagogical concept.

2. Further development of sCool (version 1) to improve the game regarding the pre-evaluation and revise the pedagogical concept.
3. Evaluate version 2 of sCool with different target groups to analyse game-related aspects and the pedagogical concept.

The sCool system consists out of two components: a web platform for educators where they can create courses and analyse the student's progress and the mobile video game. Both components are connected via an application programming interface (API) in order to provide the video game with content and to transmit the learning analytics to the server. In the game the process of learning is divided into two fields. The first part are the concept-learning missions where the players explore a world and receive concepts in a textual form. The second part are the robot missions where the players have to apply the concepts to solve tasks. These tasks can be solved by using the Python programming language. The focus of this thesis is on the practical part of the game where concepts have to be applied. In order to enable a motivating game environment and to provide additional programming concepts the robot missions will be extended. The web application has to be considered as well in this further development, since it is used for content creation and learning analytics.

1.2 Methodology and Contribution

This research is based on the work of A. Kojic (2017) and M. Kojic (2017), which developed version 1 of sCool and the web application. In their thesis they made a first user study for both components. Further research was conducted (Steinmaurer, Pirker, and Gütl (2019a)) to make a pre-evaluation in an educational context. This evaluation included two experiments in different secondary schools. For the introduction of a pedagogical concept a comparison of related work should show common game-based learning frameworks in research. Similar approaches and tools should be compared in order to get an idea of possible concepts for the revision.

The result of the project should be a further developed version of sCool. The introduced pedagogical concept should help both students and teachers in terms of education. Since different experiments will be conducted in different phases of the project the in-game courses should consist of best practice examples and guide future users of the web application to create a meaningful course. The additional learning documents (worksheets, evaluation forms, etc.) should also be evaluated and revised in order to get examples for documents that can be used in class. The web application is the tool that educators use for creating an own course respectively individual concepts. To help them at this task a web application with multi-user support, a comprehensive content creation and a sophisticated evaluation of learning analytics should be provided.

1.3 Structure

This thesis consists of nine chapters that describe different phases of the project. Chapter 2 represents the theoretical part of the thesis. In this part different learning theories and central terms like *Computational Skills* and *Gamification* are described in terms of learning and the school system. Chapter 3 is about the mobile video game sCool. It covers the version 1 of the game and explains the architecture, implementation and game types. In chapter 4 the results of two pre-evaluation of version 1 are presented. The pre-evaluations lead to requirements and a concept for version 2. These aspects are introduced in chapter 5 of this thesis. Chapter 6 covers technical specifications on the implementation and the further-development of version 2. In chapter 7 the evaluation of version 2 will be presented and the related research questions are going to be answered. Chapter 8 and 9 summarize the learned lessons and give suggestions for future development on sCool.

2 Background and Related Work

This section discusses different perspectives about computational skills and the skills that are related to computer science. The key concepts will be introduced and described in terms of computational skills. This section also covers the current situation of computer science education in the Austrian school system and compares it with other countries. Another thematic priority is game-based learning and cognitive aspects of learning. At the end of the section five game-based tools were introduced. The tools focus on teaching computational skills but with different game elements and didactic approaches.

2.1 Cognitive Perspective

There are different models and hypothesis that aim to describe the complex process of learning. Each of these learning theories covers a theoretical approach that should explain an aspect of learning. The most common theories are behaviorism, cognitivism, and constructivism.

Behaviorism
The role of the learner in this theory is a passive one. The learner only responds to different stimuli (Siang and Rao, 2003). In behaviorism the mental processes are not considered - the learner is seen as a black box that response to a stimuli with a certain behavior. Based on this response future stimuli will be affected and so a new behaviour has been learned. Despite the decrease in behavioristic methods in traditional educational contexts they are still a common approach in video gaming. The player performs a certain action and based on this stimuli follows a response. This theory is dominated by two key concepts: classical conditioning (Pavlov and Anrep, 1927) and operant conditioning (Skinner, 1938).

The classical conditioning goes back to Pavlov (1897) who observed the behaviour of dogs *Pavlov's dog*. In Pavlov's theory learning is divided into three different phases. The first phase is before the conditioning is happening. An unconditioned stimulus (UCS) leads to an unconditioned response (UCR). The term unconditioned means that no learning took place in this situation to reach this response. There is also a neutral stimulus (NS) that leads to no certain behaviour. In the second phase (during conditioning) the neutral stimulus (NS) is followed by the unconditioned stimulus

(UCS) that leads to the unconditioned stimulus (UCR). The last phase takes place after the conditioning. The neutral stimulus (NS) was transformed to a conditioned stimulus (CS) that results in a conditioned response (CR). In a game-related context this could mean (see Figure 2.1): Every time a player faces an enemy (UCS) he attacks (UCR) him, but the player will not fire if he hears a certain noise (NS). During the conditioning the noise (NS) is rendered before the player can see the enemy (UCS). But as soon as the enemy is displayed the player will attack him. In this way the noise will warn the player against an enemy. After the conditioning the player will fire (CR) as a conditioned behaviour on the appearing noise (CS). (Green Wood, Wood, and Boyd, 2005)

According to Skinner (1938) learning occurs as a result of punishment and reinforcement. The behaviour can be changed (increased or decreased) with punishment and reinforcement (Becker, 2015). To decrease a behaviour it is necessary to add positive or remove negative punishment. Vice versa if a behaviour should be increased it is required to add a positive or remove a negative reinforcement. An example therefore is earning points for right and losing points for wrong behaviour (see Figure 2.2).

Cognitivism
The term cognitivism subsumes different cognitivistic trends. The common idea is that learning is an active process (not passive like in behaviorism). The behavioristic black box will be opened and the mental processes are focused. These processes are abilities like thinking, problem-solving, knowing or memorizing. A key concept in cognitivism is knowledge. It is seen as a schema and learning *"results when information is stored in memory in an organized, meaningful manner"* (Ertmer and Newby, 2008).

In terms of video games this means that the player's perception is the element of central attention in a game environment (Cardona-Rivera and Young, 2014). An important concept in this context is *affordance theory* (Gibson, 1979) that describes the possibilities that are given from an environment to indicate which actions are feasible without mental afford. Cardona-Rivera and Young (2014) describe three elements that are necessary for a game environment that support players: i) real affordances, ii) perceived affordances, and iii) feedback. Real affordance on the one hand means the actual possibilities that an environment can offer. This type of affordance is intended by the designer. On the other hand perceived affordances describe what the player thinks is possible. This is often depending on previous experiences in gaming situations (e.g. behaviour when a user press a button). The third element is feedback and is also manipulated by the designer. Feedback should help the player to get an idea of an environments capabilities (real affordance) by increasing the perceived affordances.

Constructivism
In this paradigm the learner constructs knowledge based on environment and experience (Warren and G. Jones, 2017). The task of the teacher is to engage students to learn by exploring. This exploration process involves a sociocultural dimension in where collaboration becomes a central element to share and organize knowledge with others.

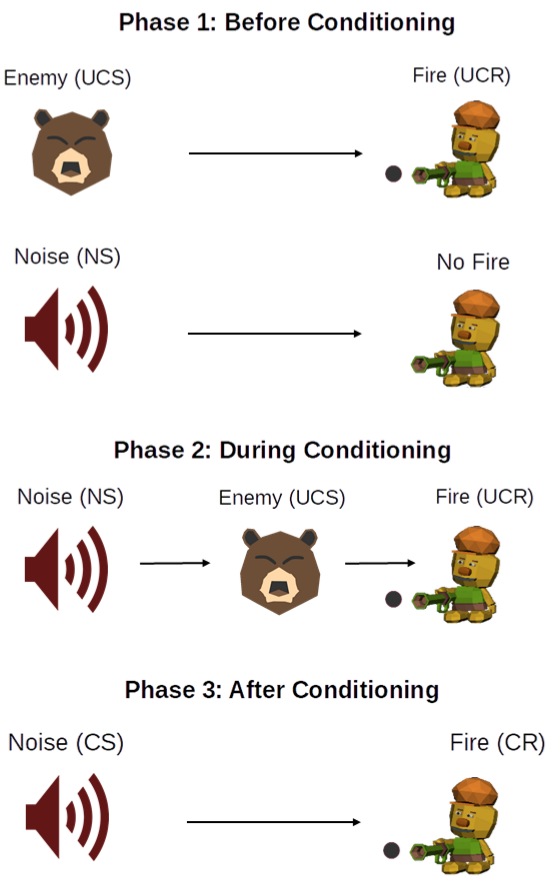


Figure 2.1: Example on Classic Conditioning adapted after Green Wood, Wood, and Boyd (2005)

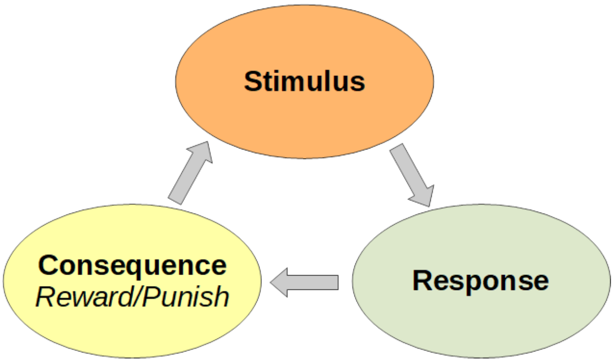


Figure 2.2: Example on Operand Conditioning adapted after Skinner (1938)

Since the central aspect is the construction of knowledge, constructivistic approaches are suitable in engineering class. Students can build their own solutions based on their knowledge and experience.

Constructivism is also related to *problem-based-learning (PBL)*. Learners are faced with a certain open-ended problem and they have to construct an appropriate solution. An important aspect is that the problem should be *"complex, ill-structured, and open-ended; to support intrinsic motivation, they must also be realistic and resonate with the student's experiences"* (Hmelo-Silver, 2004). In this context the role of the teacher has changed compared to the traditional context. The teacher gets the role of a facilitating tutor that provides an engaging environment and guides the student's learning process so they are provided with a higher level of autonomy (N. J. Kim, Belland, and Axelrod, 2019). Teachers have to find interesting problems that are manageable for the students with transparent learning goals (Allen, Donham, and Bernhardt, 2011). Good problem-solving learning is especially promoted when learners can make use of existing learning materials. Furthermore, collaborative working has a positive impact on the problem-solving process. Student learn to distribute the problem among other persons. This can be achieved especially with interdisciplinary problems. In this way students also realize social aspects of working in groups and learn to appreciate everyone's effort. Similar to real-world problems there should not by only one possible solution. The problem has to be complex enough that there are many possible solutions (Utecht, 2003). Hmelo-Silver (2004) states five fields in which students will be supported through PBL:

1. construct an extensive and flexible knowledge base
2. develop effective problem-solving skills
3. develop self-directed, lifelong learning skills
4. become effective collaborators
5. become intrinsically motivated to learn

The PBL approach can increase problem-solving skills and critical thinking. It also supports the transfer of knowledge to problems in a similar category. The students are confronted with the problem and have to invest much effort in the solution. This leads to a self-directed way of working where the students acquire different hard and soft skills (breaking down problems, collaborate, analytical thinking, ...). But the self-reliant way of working is also a point of criticism in PBL. When learners are not familiar with the field of problem or do not have prior knowledge, they cannot know what is important. This means that the teacher in the role of the tutor has to adjust the problem with the existing knowledge of the students.

2.2 Computational Skill Teaching

In today's society nearly everyone is using computers in certain ways. This already starts in preschool age when children are using toys with a basic circuit which accomplish a task based on simple calculations. In nearly every age group technology permeates our daily lives. A fundamental understanding for technology can be helpful to make use out of it. To achieve such an awareness and comprehension schools make a significant contribution in digital education. There is a large number of different approaches and tools to integrate computational skill teaching to school subjects. This variety of tools covers content for learners in a wide age range and can be even used in preschool age. (Janka, 2008; E.-C. Foerster, K.-T. Foerster, and Loewe, 2018)

2.2.1 Computational Skills

The term computational thinking was first defined by Wing (2006): *"Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science."*

Computational thinking covers a various set of skills that are necessary for solving complex problems in a structured way. This should not be an ability reserved to computer scientists, it is vital for everyone to perform analytical thinking and includes new strategies to think about problems. Especially in a century where the generation and collection of data is getting more and more important it is also unavoidable for today's school students to learn how to manage this amount of data and think about it in a critical way. This enormous amount of data faces society with new challenges. The so called *four C's of the 21st Century Skills* (critical thinking, creativity, collaboration, communication) are defining a set of skills that are necessary in the digital age. The direct use of technology (like computers) is not required for these skills because it is much more a higher-order way of thinking (Mueller et al., 2017). Riley and Hunt (2014) define problem-solving as a skill set compromising i) problem definition, ii) logical reasoning, iii) decomposition, and iv) abstraction. There is a huge number of different characteristics defining the different skills involved in computational thinking but the vital elements are similar. Code.org (2019b), BBC (2019), and Google (2019) define i) decomposition, ii) pattern recognition, iii) generalization and abstraction, and iv) algorithm design as four key techniques in computational thinking.

Decomposition
Decomposition means breaking a complex problem into smaller manageable parts. In this way each separated part can be solved in a specific way (divide and conquer). By solving many small problems the complex main problem can be understood better. This is a vital aspect in programming because a program can be parted in individual tasks that can be solved. A possible approach is for example dividing a problem into a set of similar subproblems and solve them recursively. The idea of decompose a large task into smaller units is a central ability in other subjects as well. Possible applications therefore could be language classes where it is necessary of breaking a large text into

more undestandable paragraphs or physics classes where an electric circuit is divided to smaller units. (Curzon and McOwan, 2017)

Pattern Recognition
Recognizing patterns is an essential skill in terms of computational thinking. After decomposing a certain problem there are patterns or similarities in information. If subproblems share the same characteristics, the task of solving problems can be simplified by a solution in a way that it fits for another pattern class or rather a subclass. A given pattern can be matched with a set of rules and according to them a certain instruction can be carried out. When new patterns appear, existing matches can be used to solve them. Finding patterns in information is essential in many parts of computer science to analyse data according to given features. (Curzon and McOwan, 2017)

Generalization and Abstraction
Abstraction is the ability of refining problems, and consists of two different aspects: removing detail and generalisation. The idea of removing details is to hide irrelevant parts and raise the focus on certain details. This also means filtering out some characteristics from an object in order to concentrate on the vital aspects and increase efficiency. The aim of abstracting is to receive a general concept. This can be achieved by detecting common features or properties in a model. Abstraction is used in different paradigms such as object-orientation, control abstraction, data abstraction, etc. Abstraction is a fundamental principle in computer science and software engineering to handle complex situations. Through abstraction a model can be created that gives an idea about the given problem. (Kramer, 2007)

Algorithm Design
Algorithm design is the ability to formulate instructions step by step to solve a given problem (Tucker, 2003). This step combines other computational skills (for example decomposition) to granulate a complex problem into a subset of smaller problems (refining solution). The algorithm design can be formulated in pseudo code or diagrams. Especially pseudo code is a notation that can be used in a form that even young or unexperienced learners can work with, because it can be seen as sequence of instructions used for example in recipes or manuals. According to Kant (1985) the algorithm designing process *"combines cleverness in problem solving, knowledge of specific algorithm design principles, and knowledge of the subject matter of the algorithm"*. This includes different sub processes in algorithm design (see Figure 2.3).

The above mentioned skills can be applied in various fields that are not just linked to computer science. They can be seen as a universal and structured way of solving problems. Some concepts can even be learned in a preschool age in easy ways. Most of these skills are part of the K-12 curriculum (prrimary and secondary education) and are taught in many schools. The Computer Science Teachers Association (2017) introduced standards for computer science education in K-12 (*CSTA K–12 CS Standards*) that are designed to start already in elementary school level. This standards lists five

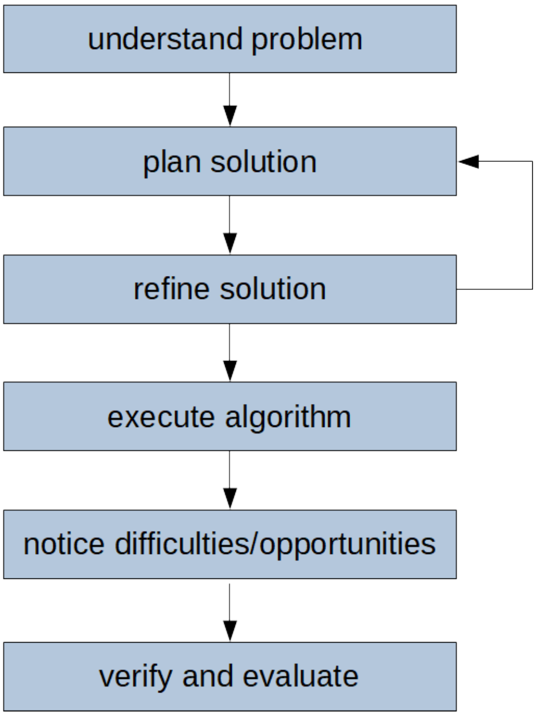


Figure 2.3: Algorithm Designing Process adapted after Kant (1985)

elementary concepts for students and elaborate them in each level according to their age and learn progress:

- Computing Systems
- Networks and the Internet
- Data and Analysis
- Algorithms and Programming
- Impact of Computing

The CSTA K–12 CS Standards are the base for the ACM Model Curriculum for K-12 Computer Science. In this model recommendations computer science is defined as *"the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society"* (Tucker, 2003). All above mentioned curricula (recommendations) also cover the impact of computing on society and culture in each level of the curriculum. Wing (2008) emphasizes the correlation between science, technology and society, since all these drivers depend on each other.

In literature the four computational skills that were initially defined by Wing (2006) are very common. Furthermore, there are other approaches to define a skill set for computational thinking that differ from Wing. Kazimoglu et al. (2012) created a game framework that teaches computational skills in a game-based way. For this purpose they defined a set of five different skills and assigned each skill a task in their framework.

This skills are more general than the previous stated approaches and can be seen as a complete workflow. These abilities are i) problem solving, ii) building algorithms, iii) debugging, iv) simulation, and v) socializing. In their model problem solving includes using computational models, identifying problems and deciding if the problem is solvable in a computational way. The algorithm should be abstract enough that it fits for different problems of the same class. Debugging includes the ability to identify and analyse issues in the created algorithm and re-think the strategies. In a next step these algorithms are used to create models for simulations. The simulations will help to consider the level of abstraction. During all of these steps socializing takes place to create strategies for working on problems together with other learners.

Another popular approach is the subdiversion of computational thinking into three dimensions (Table 2.1) that are vital in K-12 education (Lye and Koh, 2014; Weintrop et al., 2016). The first dimension (*computational concepts*) is about understanding main concepts in coding. The concepts do not depend on the actual programming language; they are universal concepts. These concepts are for example commands, conditions, loops, events, or data. The second dimension (*computational practices*) describes strategies that occur when solving problems like debugging, abstracting, or modularizing. The third dimension (*computational perspectives*) is the learner's understanding of technology and its impact and relationship on themselves and others.

Dimension	**Examples**
Concepts	Variables, Statements, Flow Control
Practices	Iteration, Recursion, Abstraction, Generalization, Modularity
Perspectives	Expressing and Questioning

Table 2.1: Dimensions of Computational Thinking (adapted after Weintrop et al. (2016))

2.2.2 Computer Science in Austrian School Education

Compared to school systems in other countries (European and international) the Austrian school systems differs in some aspects. This distinction covers for example the terms for the different school types (National Agency Erasmus+ Education, 2014). Figure 2.4 gives an overview of the current school system in Austria and uses the official terms from the Austrian Federal Ministry of Education, Science and Research. Austrian pupils attend primary school for four years. Compared to countries like Denmark, Latvia or Sweden the pupils have to attend school for nine years. Another difference is the total time of compulsory education: it is nine years in Austria whereas the majority of Europa provide children 10-12 years of education.

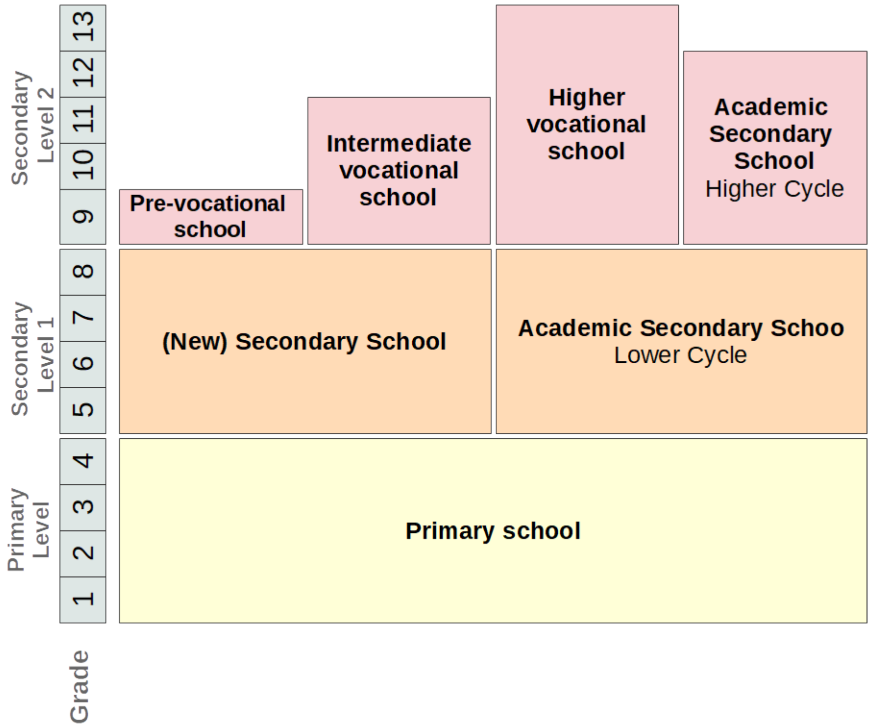


Figure 2.4: Austrian School System adapted after Austrian Federal Ministry of Education and Research (2016)

In Austrian secondary schools the subject computer science is mandatory in the ninth grade (secondary level 2). The curriculum's framework is regulated by the federal ministry. This curriculum provides two hours per week computer science class in both semesters. Furthermore, many secondary schools (especially schools with focus on STEM education) offer computer science as an elective subject in grade 6, 7, and 8. The curriculum defines different modules and competences that should be covered. It is up to the teacher to define the specific learning content of the subject in coordination with the school students.

The curriculum formulates four main aspects for both the mandatory and the elective subject that should be covered in computer science class. The following list contains the content of the ninth grade (Austrian Federal Ministry of Education and Research, 2000):

- **Computer Science, Human and Society**: School students learn the influence of computer systems on today's society and economy. This aspect also covers the knowledge of information security and copyright.
- **Computer Systems**: School students know about the structure of computer systems and are able to explain digital devices. They also know about the functionality of operating systems and have the ability to connect computers to networks.

- **Applied Computer Science**: School students know different standard software for creating documents, presentations, spreadsheets, and visualisation. Computers can be used to retrieve, analyse, structure, and process information.
- **Practical Computer Science**: School students know about basic concepts of computer science and have the ability to design algorithms and implement them into a programming language. They know about machines, data structures, and different algorithms and are able to create simple data models using a database.

2.2.3 Digital Basic Education in Austria

In the Austrian school system the value of *digitalization* has highly increased over the last years. Therefore, the Austrian federal ministry of education, science and research started to work on the so called *masterplan* for digitalization in summer 2018. Various experts and other federal ministries are involved in the implementation which is planned to be finished in 2023. The *masterplan* covers three major fields of action in the school education. (Austrian Federal Ministry of Education and Research, 2018)

The first part is about *teaching guidelines* and existing syllabi and aims to revise existing syllabi in all grades and all school types. It also includes the introduction of a new subject called *digital basic education* (*Digitale Grundbildung*) in Austrian secondary level 1 (new secondary school and academic secondars school). This subject was introduced in the school year 2017/18 with 2-4 hours per week in four years. The content can be taught as part of other subjects (*integrated teaching*) or as separate lessons. It covers a large number of abilities in different disciplines of computer science (Austrian Federal Ministry of Education and Research, 2019):

- social aspects through media change in digitalization
- digital and media competences
- operating systems and applications
- media design
- communication and social media
- security
- problem solving skills
- computational thinking

The second part will focus on the ICT infrastructure and a modern school administration. Therefore, the schools will be provided with suitable equipment (mobile devices, broadband, WiFi, electronic boards) and tools (learning management systems, eBooks, etc) to bring digitalization to all parts of education (including students, teachers and parents). The third part is about training and development of all teachers in the field of digitalization (Austrian Federal Ministry of Education and Research, 2019).

2.2.4 Computer Science in other countries

This section draws a comparison between computer science education in different countries in Europe and internationally. The following countries share a new approach of ICT/computational skill teaching in the curricula and are often supported from organisations to promote computer science education. The UK introduced the subject computing which covers a variety of computational skills. Sweden's national curriculum was revised in 2018 in order to focus on digital competences. Australia has two different subjects in the F-10 curriculum where the main concepts of computer science are taught. In addition chapter 7 covers an experiment that was conducted in an Australian University. The USA is pioneer in computer science education science since the initiative CS for all already began in 2016.

Since 2014 the UK implemented the subject *computing* for all national schools beginning in key stage 1 (5 years old) until key stage 4 (16 years old). Key stage 1 and 2 are at primary school and key stage 3 and 4 are secondary school. The subject is separated in three fields: computer science, information technology, and digital literacy. The aim of the subject is to teach computational thinking and related skills (decomposition, pattern recognition, abstraction, pattern generalisation, and algorithm design; Kamp, 2014). Besides new learning content a focus was on teacher education to bring knowledge to the pupils. Sentance and Csizmadia (2017) observed teaching strategies that teachers were using in this new subject. The top three strategies mentioned by the teachers are: unplugged strategies (learning away from the computer), reference to particular software and relational to real world activities. Besides the national curriculum, the community *Computing At School (CAS)* promotes computer science in schools. CAS is a part of the British Computer Society and supports teachers with different resources and guidelines. (Peyton-Jones, Mitchell, and Humphreys, 2013)

In 2018 the Swedish national curriculum was revised to bring digital competences and programming into K-9 school level (Heintz et al., 2017). Since 1994 the subject *technology* was taught in Sweden. The revised version is based on the European DigComp framework and concerns three main topics in the curriculum: *technological solutions*, *working methods for developing technical solutions*, and *technology, man, society and the environment*. The subject combines different aspects in technology and also covers computer hardware, construction, electric circuits, etc. (Skolverket, 2018)

The national curriculum F-10 (foundations until level 10) of Australia covers the learning area *technologies* each school year. This area covers two subjects: *digital technologies* and *design and technologies*. In both subjects the school students should learn how to create solutions, learn thinking skills (systems thinking, design thinking, and computational thinking) and project management. The main concepts in the subject digital concepts are: i) abstraction, ii) data collection, representation and interpretation iii) specification, algorithms and implementation, iv) digital systems, and v) interactions and impacts. (Australian Curriculum and Authority, 2015)

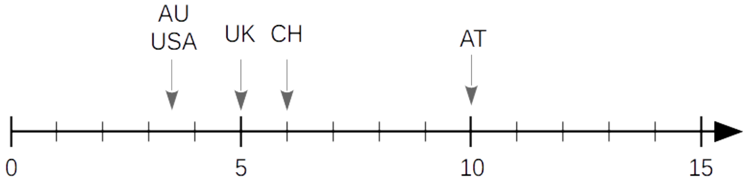


Figure 2.5: International Comparison of Computer Science Education (After Grandl and Ebner, 2017)

In 2016 president Obama started the *CS For All* initiative which should engage students from kindergarten to high school (K-12) to learn computer science. As part of the initiative a comprehensive funding, teacher training, etc was provided (Smith, 2016). Besides government measures many organisations also focused on engaging computer science education. These organisations also developed different curricula and recommendations for computer science class. The nonprofit organisation Code.org (2019a) has the vision *"that every student in every school has the possibility to learn computer science, just like biology, chemistry, or algebra"*. Code.org is supported by many companies like Microsoft, Facebook, Google, Amazon, etc. Together with other supporters like the Association for Computing Machinery or Computer Science Teachers Association the *K-12 Computer Science Framework* has been developed which has five core concepts: i) computing systems, ii) networks and the Internet, iii) data and analysis, iv) algorithms and programming, and v) impacts of computing (Science, 2019).

2.3 Gamification

Deterding et al. (2011) defines gamification as *"the use of game design elements in non-game contexts."* This means gaming elements are applied in situations that usually have no game context. Another central aim of gamification is to increase the engagement of the learner (S. Kim et al., 2018, p. 29). Figure 2.6 classifies gamification and differentiates it to similar areas. For this purpose the two dimensions playing/gaming and whole/parts were considered.

In order to distinguish gamification from game-based learning it just introduces game-like elements (elements or mechanics) into a non-gaming setting. This should make the traditional context more engaging. This should also lead to a better progress in learning and to a higher motivation to solve a given task (Huotari and Hamari, 2012). There is a large number of game mechanics that can be added in terms of gamification (Brull and Finlayson, 2016; S. Kim et al., 2018):

- Points
- Badges
- Levels
- Leaderboards
- Progression Bars

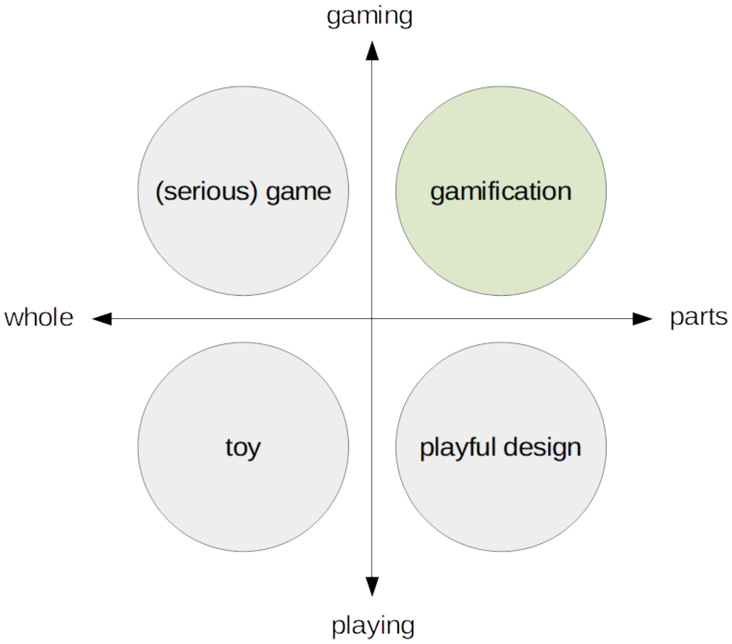


Figure 2.6: The dimensions of gaming/playing and whole/parts (After Deterding et al., 2011)

- Certificates
- Story
- Avatar (selection and customization)

2.3.1 Gamification in Education

The overall purposes of gamification is *"first to encourage desired learning behaviours [...] the latter to engage students in learning"* (Ibanez, Di-Serio, and Delgado-Kloos, 2014). According to Raymer (2011) there are three central aspects for a gamified environment:

- **Goals and objectives**: The environment should provide a balanced relation between skill and challenge in order to reach an adequate level of flow.
- **Feedback**: Feedback means a verbal or non-verbal message that provides information about the performance. There are two types of feedback: positive and negative. Positive feedback can be used to support, encourage and emphasize the strengths. Negative feedback is about weaknesses and insufficient performance. An important aspect when developing a gamified environment is to consider the way the feedback is given because it has a major impact on the learners. (S. Kim et al., 2018)
- **Rewards**: Rewards are used to keep the player's motivation high and appreciate their progress. According to S. Kim et al. (2018) rewarding elements can be:

points, levels, progression, badge, virtual/physical goods, gifts, virtual currency, etc.

In a literature review of gamification in different Science, Technology, Engineering , Mathematics (STEM) subjects, Ortiz, Chiluiza, and Valcke (2016) figured out that especially in the field of computer science gamified approaches are most common. The majority of these approaches is using a combination of different gamification elements (in particular points, badges, and leaderboards). In a computer science context this combination of badges, points and leaderboards is quite common (Narasareddygari, Walia, and Radermacher, 2018). Piteira, Costa, and Aparicio (2018) introduced a theoretical gamification framework for online learning coding courses consisting of following dimensions: "*target audience, general goals, learning outcomes, topics, contents, gamification, cognitive absorption, flow, and personality*".

The World Government Summit (2016) broke down gamification into a set of three elements: mechanical, personal, and emotional. Mechanical means clear defined goals and structured challenges. This should be an incremental process where the challenge is getting harder. Elements to acknowledge and reward the progress in an environment can be badges. Another vital gamification feature in terms of mechanical elements is onboarding. This is necessary to help the learners to become acquainted with the system and it also relieves the teachers. Since the first minutes on a new system are the most important it is rather necessary to engage them (Zichermann and Cunningham, 2011). Personal elements in a gamified system can be avatars, leaderboards and social engagement loops. The usage of avatars (self-representation within a system) emphasizes the creativity and the individuality of the players. In an educational context it can be seen as an *alter ego* used to have a different form of identity. Leaderboards are used to show the rank of a user in a gamified environment. They can show the name, avatar, rank, scores and other in-game metrics. Since leaderboards compare with other users the usage in an educational context is controversial. The last mentioned category concerns the player's emotions. This covers mainly the flow while performing an activity (see Section 2.4.3) (World Government Summit, 2016).

2.3.2 Tools for Gamification

Since gamification means the use of gaming elements in a non-gaming context it is not necessary to use tools. Nevertheless, there is a large number of different tools that are applied in school education. This section will introduce three different tools that can be used in terms of gamification in (school) education.

The web-based tool ClassCraft[1] is a gamified Learning Management System (LMS). It has a fantasy theme in which the players create an avatar and work together in teams to make achievements. At the beginning teachers create a class and explain the

[1] https://www.classcraft.com/de/

rules. When school students reach different goals they gain *Experience Points (XP)* and for their behaviour they either gain or lose *Health Points (HP)* . XPs can be assigned for getting a question right or helping others. It is up to the teachers to define the purpose of the points. For instance XP can be used to level up or to unlock real-life powers like additional time for homework or eating in class. All school students play in teams and so they are engaged to collaborate in order to achieve success. The tool also provides teachers with different tools that can gamify various tasks in class. They can use a sound volume indicator to record the noise level in the class to reward or punish the teams. (Glod, 2017) ClassCraft also can map the experience points to make grading easier and transparent. The tool can also be used for submitting homework or assignments and giving feedback. All in-game metrics (XP, HP, achievements) are transmitted in real-time so teachers can work with ClassCraft during class (Papadakis and Kalogiannakis, 2017).

SoloLearn[2] is a platform (web, Android and iOS) for learning coding. They provide different languages like Python, Java, or HTML in individual courses. Each course contains lessons which represent different concepts and quizzes that are taken after a lesson. When all questions of the quiz are answered correctly a new lesson is unlocked. In contrast to similar coding platforms SoloLearn has added some gamification elements. After completing a lesson the players receive a certain amount of XPs. These points are also a virtual currency that can be used to skip questions or get hints. The players may get rewarded with badges for different achievements and level up. Since each user has a public profile (avatar) the achievements of other players can be seen. SoloLearn also provides a leaderboard where the *top learners* and users with fewer points but close to the high score are displayed. (SoloLearn, 2019)

The platform Kahoot![3] is a platform where teachers can create quizzes and surveys or use already existing templates. There are different types of questions (regular quiz, true or false, puzzle, slide, etc.) that can include different media types (images, videos, etc.). Each single question can be assigned to a maximum number of points. When the quiz is created players can assign to the quiz with a pin. There are two main possibilities to access the quiz: i) live ii) as homework. When playing the game live in class a quiz can be joined by entering the pin in the app or the web browser. When the teacher starts the quiz the questions are displayed on the computer (respectively video projector). The learners have to answer the questions on their devices. Depending on the answer and the response time all players receive points. After each question the current leaderboard is displayed. (Tan, Ganapathy, and Mehar Singh, 2018) Besides the live mode it is also possible to create a *homework challenge*. Teachers can create a quiz with an expiration date and hand out the pin. The players receive points for right questions and are also displayed in a leaderboard. When using the app it is also possible to unlock achievements (badges) in the game. (Kahoot! 2019)

[2] https://www.sololearn.com/

[3] https://kahoot.com/

In Table 2.2 the above mentioned tools are listed and the most common gamification elements are compared.

Tool	Points	Badges	Levels	Leaderboards	Progression Bars	Certificates	Story	Avatars
ClassCraft	x	x	x	x	x	x	x	x
SoloLearn	x	x	x	x	x	x		x
Kahoot! (live)	x	x		x	x			

Table 2.2: Comparison of gamification elements in analysed tools.

2.4 Game-Based Learning

The idea to use games in school education has already been there for decades. A comprehensive number of tools in nearly every field of education makes it more and more easy for teachers to use them in school. These approaches help both teachers and students in an educational context. The teachers can be supported with already existing best-practice example (like curricula for example) and meaningful learning analytics. Learners are supported thtough extrinsic motivation to learn new concepts in a playful way. Tools can also provide students immediately with additional information and make the assessment easy as well.

2.4.1 Game Design

Playing and games are a natural way for humans to learn. Learning is something that happens the whole life and it does not depend on what is learned. In contrast, the term education is socially and culturally defined. Playing not always pursues an educational purpose, but it is always connected to learning (Becker, 2015, p. 6). Education does not claim to be entertainment or fun. But when combining playing and education it can be an engaging possibility for the learner (Rishipal, Saraff, and Kumar, 2019). According to Becker (2015) and Reinders (2012) a game has following characteristics:

- interaction,
- rules,
- one or more goals and objectives,
- narrative,
- outcome and feedback,

- conflict and competition
- quantifiable measure of progress,
- recognizable ending.

Lazzaro (2004) observed the emotions that were occurring during playing and connected these emotions to different experiences:

- **Hard fun**: People who enjoy hard fun play games to challenge themselves. They play to see how good they play the game and want to beat the high score. They are mainly driven by failure and success to enjoy playing.
- **Easy fun**: This type of fun is related to simply enjoying the game. Players are curious and want to explore the game. They want to dive into the narrative and are immersed by the story.
- **Altered states**: This experience is related to the emotions of a player while being in the game. The game helps to change the internal experience and they feel different when playing. Games are played to feel better for example.
- **People factor**: The key to the playing experience in this group is enjoying the social component while playing the game. These group of players also play even if they do not like the game just to spend time with other people.

This means that not every player feel the same way while playing. The players can also be classified based on their interests while playing the game. Bartle (1996) introduced four different groups of players regarding their interest (see Figure 2.7. The x-axis has two characteristics and goes from players to the world (or environment). On the other hand the y-axis goes from acting to interacting. Each player type has a stronger interest in the graph. The killers have a high tendency to the acting-axes and the players-axes. Their focus is competition and conflict with other players; mainly to demonstrate ability and see others lose. This amount of players that fall in this category is less than 1%. The second category with a high tendency in acting are the achievers. Compared to the killers they have a low interest in players and are more interested in the environment. Their focus is about rewards and prestige in the game. This could be for example by getting to a higher level, collecting badges/items or leading the ranking. Bartle (1996) estimates that about 10% of players are part of this category. The third group are the socialites with a strong tendency to players and interacting. They are driven by a social aspect and want to collaborate with other players to achieve something together. They mostly enjoy the social interaction that takes place in the game or in various communication channels (newsfeed, chat, etc). About 80% of all players belong to this group. The last group (about 10% of players) are the explorers that are more into the game's environment and the interaction. Their focus is on exploring as much as possible and to discover new secrets. They get engaged by hidden achievements and are willing to also accomplish repetitive tasks in order to unlock something new (Zenn, 2017; Schneider et al., 2016).

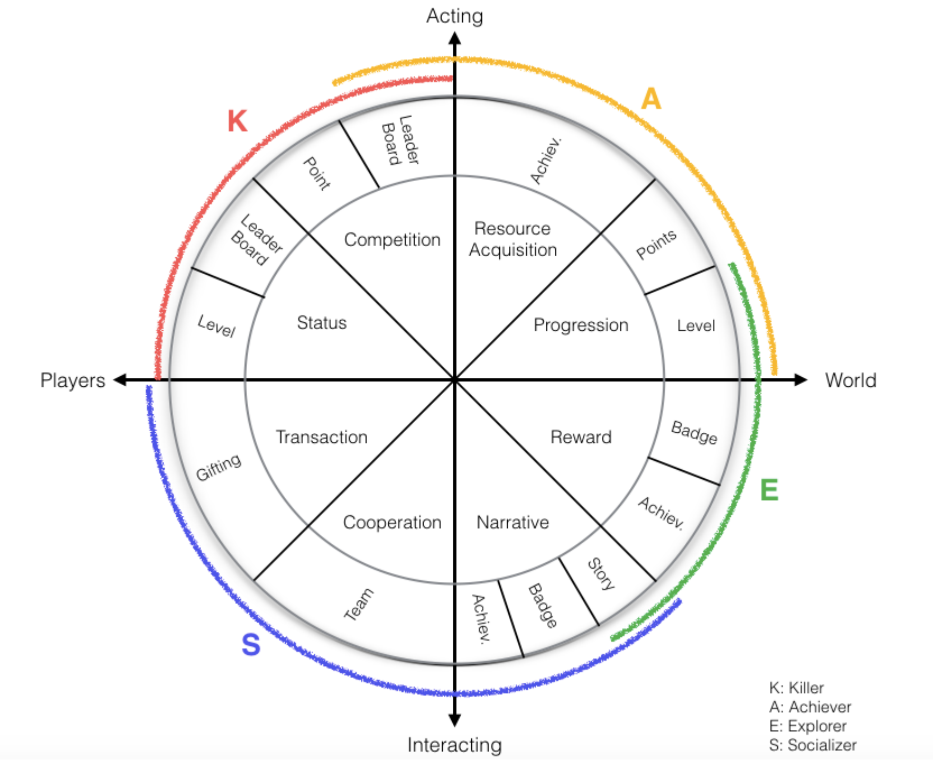


Figure 2.7: Four types of players (Arkün Kocadere and Çağlar Özhan, 2018)

2.4.2 Game Frameworks

Game frameworks can be used to identify game design elements in a video game. A model-driven approach is the MDA (mechanics, dynamics, and aesthetics) framework (Hunicke, Leblanc, and Zubek, 2004). The MDA framework is a link between the designer and the player but seen in a different perspective. The designers begin with the game's mechanics and will then work on the dynamics aspect of the game. The exterior level is the aesthetics that are on the surface. From a player's perspective the model is seen in a reverse view: The first layer of interaction is the experience of the aesthetic part. After the players are getting familiar with a game they are able to understand the dynamics and in further consequence also the mechanics. When designing a game it can be useful to consider both the player and the designer aspect. (Hunicke, Leblanc, and Zubek, 2004; Bohyun, 2015)

- **Mechanics**: Are the basic components of a game that reflect the general rules. All possible basic actions (represented by algorithms and data structures) in the game are part of the mechanics.
- **Dynamics**: The dynamics focus on the interaction of the players with the game mechanics. The mechanics are about the behaviour on input and output over time. Examples for dynamics are for example time pressure or sudden consequences during the game.

- **Aesthetics**: This component refers to the player's experience when interacting with the game. Hunicke, Leblanc, and Zubek (2004) list a (optionally expandable) taxonomy of types of aesthetics: sensation, fantasy, narrative, challenge, fellowship, discovery, expression, and submission.

2.4.3 Learning Theories in GBL

The term *flow* goes back to Csikszentmihalyi, Abuhamdeh, and Nakamura (2014): *"Flow is a subjective state that people report when they are completely involved in something to the point of forgetting time, fatigue, and everything else but the activity itself. [...] The defining feature off low is intense experiential involvement in moment-to-moment activity. Attention is fully invested in the task at hand, and the person functions at his or her fullest capacity."*

Csikszentmihalyi defines three different kinds of experiences: flow, boredom, and anxiety. The full capacity is given when a person is in flow. This state can be reached when the appropriate balance between skill and challenge is reached. If one of both parameters strongly exceeds, the person gets anxious or bored. In total Csikszentmihalyi brings up eight different states (see Figure 2.8) in four quadrants that can appear in any activity. Especially in the field of education a state of flow would mean the best outcome for the learner. The state of flow occurs mostly when a person is highly intrinsically motivated and does not feel required to learn new content because of external factors (G. M. Jones, 1998). An area where flow has a relevant character are video games. For this purpose Radoff (2011) adapted the eight states according to gaming: flow, arousal, control, relaxation, anxiety, worry, apathy, and boredom. Equal to Csikszentmihalyi's model Radoff declares flow as *"optimal mental state in which there is a balance between the challenge of the game and the player's skill"*. Additionally G. M. Jones (1998) defines eight characteristics for a high level of flow in gaming:

- a task that can be completed
- the ability to concentrate on the task
- that concentration is possible because the task has clear goals
- that concentration is possible because the task provides immediate feedback
- the ability to exercise a sense of control over actions
- a deep but effortless involvement that removes awareness of the frustrations of everyday life
- concern for self disappears, but sense of self emerges stronger afterwards
- the sense of the duration of time is altered

This model of elements involving the level of flow was adopted by Sweetser and Wyeth (2005) regarding gaming (*GameFlow model*). The model consists of eight different elements: i) the game, ii) concentration, iii) challenge player skills, iv) control, v) clear goals, vi) feedback, vii) immersion, and viii) social interaction. If the game is too complicated or too easy the player is not in a state of flow.

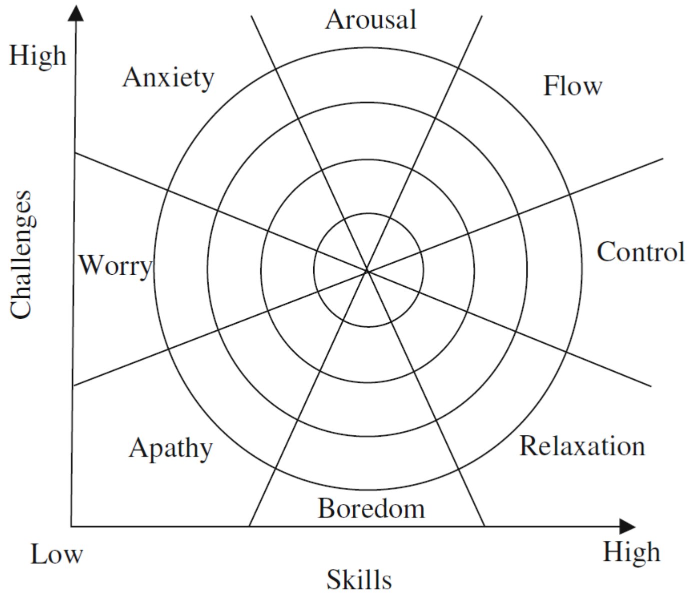


Figure 2.8: Challenge/Skill Diagram (After Csikszentmihalyi, Abuhamdeh, and Nakamura, 2014)

Csikszentmihalyi also showed that it is necessary to provide immediate feedback and specify clear goals. It is also important to enable self-determination and let the learners make decisions to reach a flow engaging state. De Freitas and Neumann (2008) developed the *exploratory learning model* (Figure 2.9) containing five steps that occur while exploratory learning in a game-based learning context. The model is based on Kolb's (1984) learning cycle where learning starts with a concrete experience. Afterwards the learners should get engaged to reflect the experience regarding learning. This step leads to think about the concepts, abstract and conceptualise the thoughts. The last step is the active experiment where the learners apply the learned knowledge. De Freitas and Neumann (2008) describe a cycle that is a similar approach but expands it by learning in immersive environments. The advantage of such an environment is the possibility for social interactions with other learners which leads to *social interactive learning*.

2.5 Game-Based Learning in Computer Science

In computer science and particular in coding class there are many game-based learning tools available. There are mainly three categories of games (Combéfis, Beresneviuius, and Dagiene, 2016):

- **Coding**: The focus is on learning *how* to code. A central part is to understand the language and syntactical features. Objectives are for example learning how to fix broken code or writing code that fulfills a certain task. This code is usually

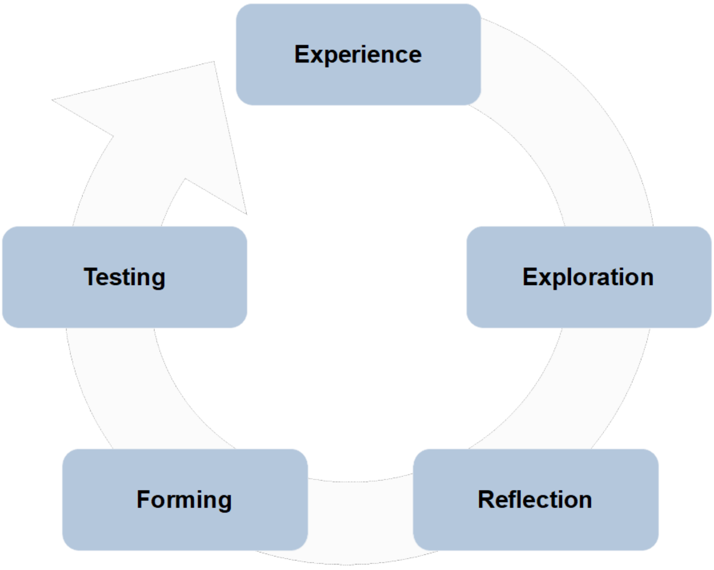


Figure 2.9: Exploratory Learning Model (adapted after De Freitas and Neumann, 2008)

submitted and the system provides feedback. The feedback can be a notification showing whether the task is passed or failed, but it can also be on a high elaborated level.

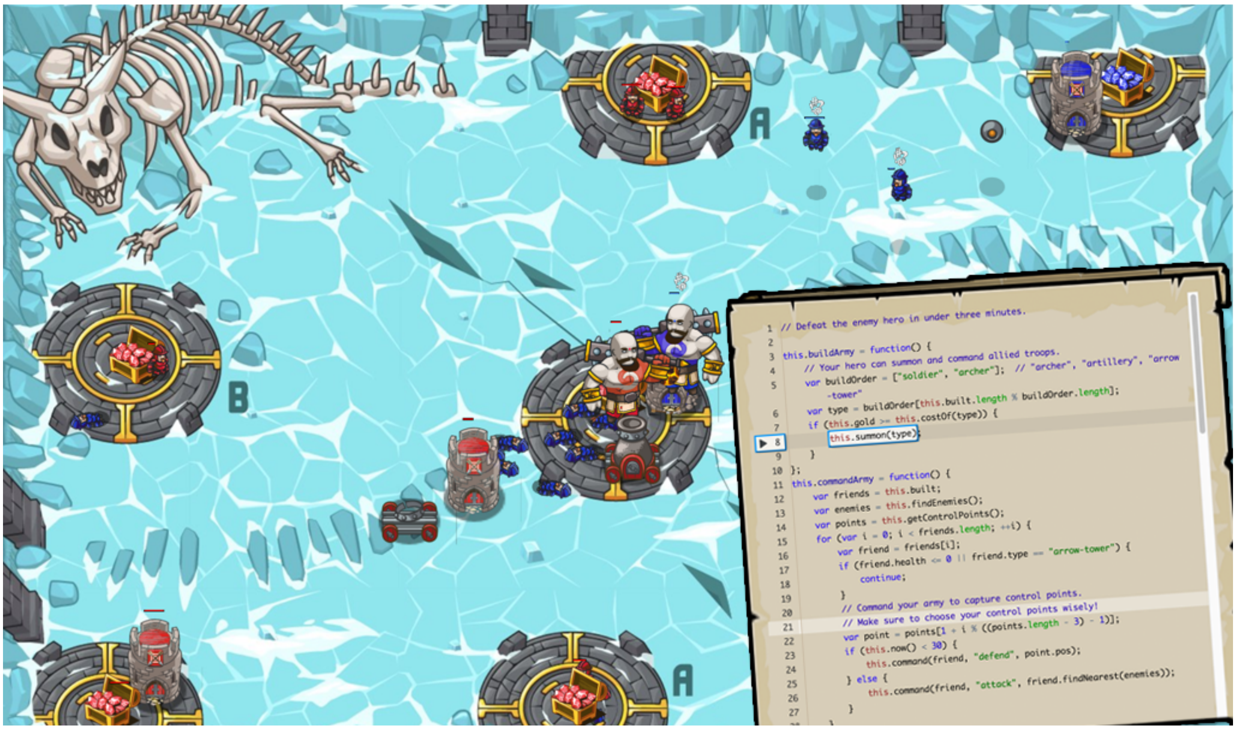
- **Algorithmic thinking**: In algorithmic thinking the focus is not on learning a particular programming language and relating concepts. The system provides various problems that have to be solved in a technical way. This can be reached by using a programming language (or a technical notation) or other concepts like command blocks. The main objective is to solve a given problem (searching, sorting, path finding).
- **Creating games**: Learners create a game using a technical notation (for example a programming language or blocks). In this way they learn computational skills that are necessary to create a game.

In the next section some tools were described that follow a game-based learning approach for teaching compuational skills. These tools share some similarities regarding game design and didactic concepts. The usage and the target audience is slightly different in the following tools. These key features of all games will be observed and analysed. Table 2.3 shows the main characteristics for each game and compares them with each other.

2.5.1 CodeCombat

CodeCombat[4] is an open-browser-based game where players can learn coding with various programming languages (for example Python or JavaScript) and the fundamentals

[4] https://codecombat.com/

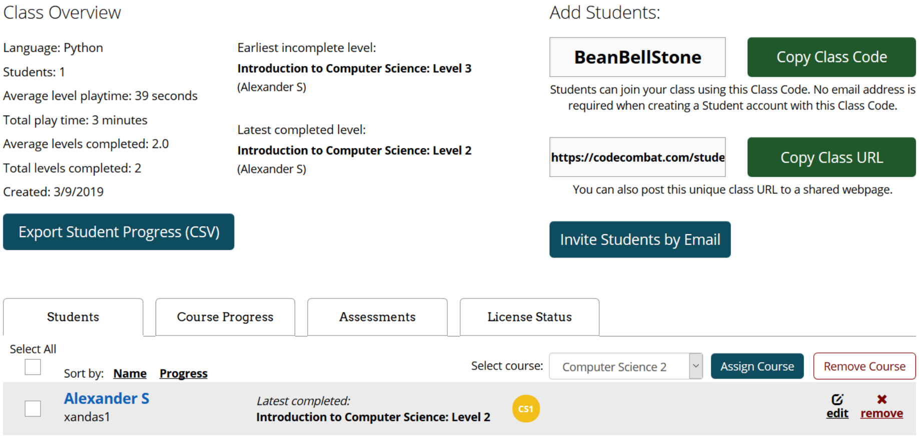


```
// Defeat the enemy hero in under three minutes.

this.buildArmy = function() {
    // Your hero can summon and command allied troops.
    var buildOrder = ["soldier", "archer"];  // "archer", "artillery", "arrow
        -tower"
    var type = buildOrder[this.built.length % buildOrder.length];
    if (this.gold >= this.costOf(type)) {
        this.summon(type);
    }
};
this.commandArmy = function() {
    var friends = this.built;
    var enemies = this.findEnemies();
    var points = this.getControlPoints();
    for (var i = 0; i < friends.length; ++i) {
        var friend = friends[i];
        if (friend.health <= 0 || friend.type == "arrow-tower") {
            continue;
        }
        // Command your army to capture control points.
        // Make sure to choose your control points wisely!
        var point = points[1 + i % ((points.length - 3) - 1)];
        if (this.now() < 30) {
            this.command(friend, "defend", point.pos);
        } else {
            this.command(friend, "attack", friend.findNearest(enemies));
        }
```

Figure 2.10: CodeCombat - User Interface

of computer science (S. Kim et al., 2018, p. 127). It is a two-dimensional role-playing game (see Figure 2.10) where the players control a character by programming it. There is no clear defined narrative in the game but the environment and the enemies of the player (dessert, ice, dungeon) change. Each concept is represented by a map of a certain world with different locations that are represented by levels. The two major elements in a level are the graphical representation of the maze and the coding editor (see Figure 2.12). Players learn concepts like algorithms, loops, conditions or objects. Besides coding the students can also create their own games and levels and share them with other players. In different levels students learn the concepts step by step. The game also includes different gamification elements like experience or ranking. With increasing experience the player is also rewarded with new items and can upgrade their avatar. The tool provides visual feedback and hints to support the player.

CodeCombat provides a web-based tool for educators called *Teacher Dashboard*. The teacher dashboard at Figure 2.15 illustrates the possibilities to manage courses and students. Teachers can create different courses for each class and adapt them according to their needs. This tool also makes it possible to receive assessment and see the individual progress of each student.



Class Overview

Language: Python
Students: 1
Average level playtime: 39 seconds
Total play time: 3 minutes
Average levels completed: 2.0
Total levels completed: 2
Created: 3/9/2019

Earliest incomplete level:
Introduction to Computer Science: Level 3
(Alexander S)

Latest completed level:
Introduction to Computer Science: Level 2
(Alexander S)

Export Student Progress (CSV)

Add Students:

BeanBellStone Copy Class Code

Students can join your class using this Class Code. No email address is required when creating a Student account with this Class Code.

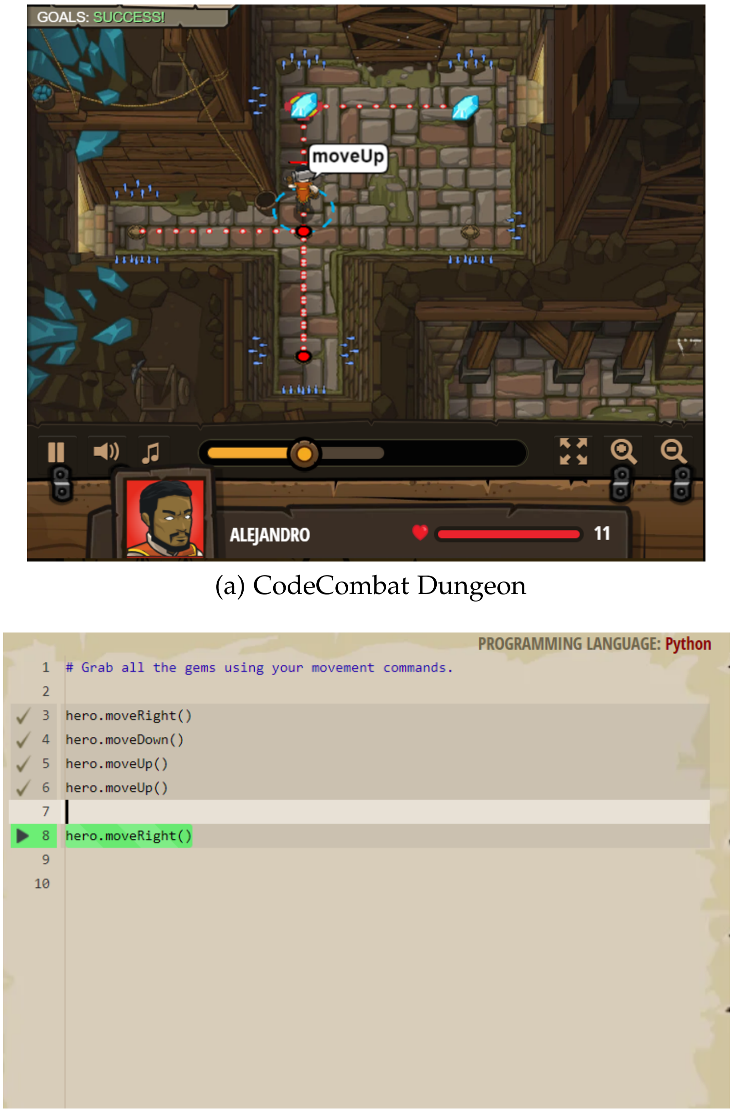
https://codecombat.com/stude Copy Class URL

You can also post this unique class URL to a shared webpage.

Invite Students by Email

□	**Alexander S** xandas1	*Latest completed:* **Introduction to Computer Science: Level 2**	CS1	edit remove

Figure 2.11: CodeCombat - Teacher's Dashboard

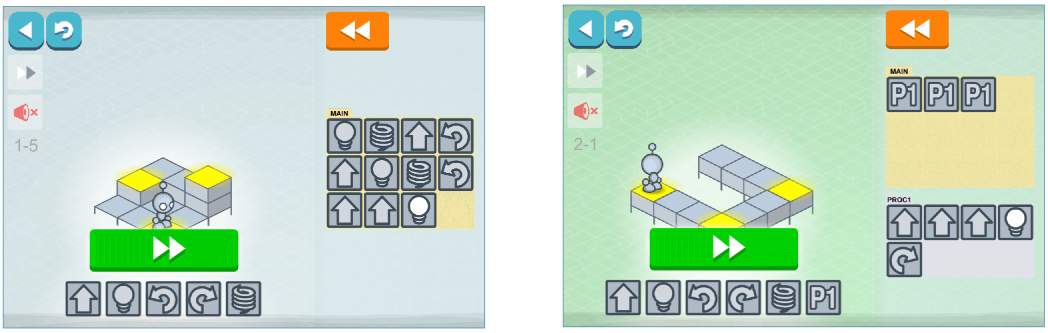


(a) CodeCombat Dungeon

```
# Grab all the gems using your movement commands.

hero.moveRight()
hero.moveDown()
hero.moveUp()
hero.moveUp()

hero.moveRight()
```

(b) Programming Editor in Python

Figure 2.12: CodeCombat Game elements



(a) Game Board

(b) Programming Editor

Figure 2.13: Lightbot Maze Game

2.5.2 LightBot

LightBot[5] is an educational puzzle game for learning computational skills. It can be played on both mobile devices and web browsers. The main idea is that the player controls a robot with different command blocks without writing a single line of programming code. Each block represents a certain command that the robot should fulfill. The aim is to navigate over a field, overcome obstacles and illuminate all necessary fields. According to Yaroslavski (2014) the game can be used to learn sequencing, overloading, procedures, recursive loops, and conditionals. Similar levels are grouped together to courses where different concepts can be learned. LightBot has seven command blocks: moving forward, turn left, turn right, light field, jump, call procedure 1, and call procedure 2. The players drag and drop the blocks into an editor and the commands are executed successively. There are additional editor slots for two procedures. In this way it is possible to write a function and call it in the code. This allows to create a loop-like structure (recursive procedures). Figure 2.13 demonstrates two missions with different levels of difficulty and available command blocks.

When learning with LightBot it is not necessary to create a user. It can be played in the browser or on a mobile device. All levels and concepts are already predefined and so it is not possible for educators to adapt the content. The game is also fully client based and so there are no learning analytics for teachers.

2.5.3 CodeMonkey

CodeMonkey[6] is a comprehensive web-based educational game; teaching coding and computational skills. It enables to learn different concepts among the K-12 curriculum with various modules. CodeMonkey supports curricula like CSTA K-12 curriculum or

[5] https://lightbot.com/
[6] https://www.codemonkey.com/

the national curriculum of England. Students can learn the programming languages Python and CoffeeScript. It covers the following topics in computer science: *"objects, function calls, arguments, loops, variables, arrays, for loops, function definitions, boolean conditions, until loops, if and if-else conditions, boolean operators, keyboard and mouse events"* (*CodeMonkey* 2019).

Currently CodeMonkey supports the following game modes:

- **Coding Adventure**: Students learn programming with CoffeeScript (see Figure 2.18). The aim of the game is to navigate a mokey over a two-dimensional grid to reach bananas. They have to pass obstacles by using programming concepts. This mode includes three different courses: fundamentals (statements, loops, objects, arrays, and statements), functions and conditions (functions, conditions, bool logic), and logic and events (operators, return values, triggers).
- **Banana Tales**: In this course the players have to bring a baby monkey bananas with the programming language Python. During these levels they learn programming concepts and other skills in computer science (classes, strings, lists, sorting). The game also provides hints and supports the players in different ways (code completion, numbered two-dimensional grid, ...).
- **Coding Chatbots**: This course teaches how to program a chatbot in Python. It is suitable for kids at the age of 13 years and older. Apart from basic concepts like functions, control structures, and primitive data structures players also learn advanced skills like complex data structures and server side programming. Figure 2.16 illustrates a comprehensive example of a sophisticated chatbot in Python.
- **Challenge Builder**: Students can create their own challenges with a level editor and share them. In this way the students are engaged to work together with others and solve different mazes. When creating a level students have to think about the concepts and game elements that others can use. So they move from learning to creating. The shared levels are also solved in CoffeeScript similar to the coding adventure mode.
- **Dodo Does Math**: This course deals with 2nd-4th grade skills in maths and coding. The course is divided into three different parts: distances, angles, and multiplication. By using a very basic CoffeeScript syntax the students solve maths problems.
- **CodeMonkey Jr.**: This mode is especially for young learners (pre-school age) because it motivates them to deal with computational thinking. CodeMonkey Jr. is an app for iOS and Android where the skills are learned in a block-based approach. Compared to the other game modes no knowledge on CoffeeScript or Python is necessary since the game deals with a small set of command blocks.
- **Game Builder**: The game builder supports three different game types (platformer, frogger, and sprite animation) to teach creating games. School students learn concepts like event handling, parameters, loops, sprites, and animations. They design and create games that can be shared with other players.

The tool also provides a large functionality for educators. It supports a full classroom management system for teachers which includes the creation of a curriculum and automatic grading. The students can be assigned to classrooms and teachers can manage them. Figure 2.15 shows the progress of the students that are enrolled in the selected course. Teachers can export the results or receive a more detailed analytic.

2.5.4 Grasshopper

The mobile video game Grasshopper[7] (see Figure 2.17) is an app for learning JavaScript. It is mainly designed for adult learners. Grasshoppper covers the fundamentals of JavaScript (instructions, functions, variables, loops, arrays, objects, ...) and animations (shapes, D3 library, callback functions, and animations). Each concept is represented by different tasks a player has to solve. This tasks are mainly:

- **Drawing**: The library D3 is used to draw different shapes (for example flags)
- **Coding**: By reference to an explanation and an example solution a certain code should be written.
- **Concept-learning**: A certain concept is explained with text, code examples or images.
- **Single-choice**: A single-choice question is asked based on previous learned concepts. This question is often a code listing and a corresponding question.

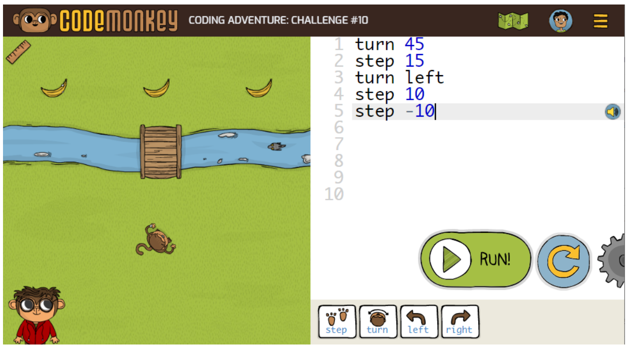
There are many gamification elements in the game that can help the player to stay engaged. The players can unlock different skills and receive various achievements. The app also supports micro-learning sequences since every course covers tiny lessons. To keep the player's attention and promote consistent learning, it is also possible to set a daily reminder.

Despite Grasshopper being a game for mobile devices, the keyboard was optimized in a convenient way. The players can pick code suggestions from a predefined list. All declared variables are contained in this list so it is not necessary to type much code. Each group of different language elements is represented in a different color. So it is possible to distinguish between variables, strings or control structures. All available commands are linked to a brief description of their functionality.

2.5.5 sCool

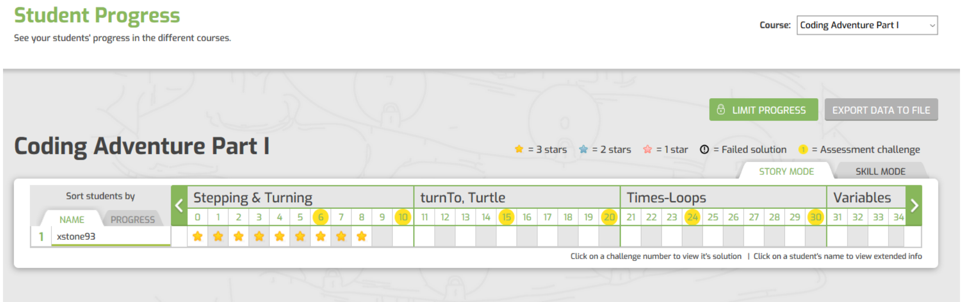
The mobile video game sCool was initiated in 2017 as a collaboration between Graz University of Technology and Westminster University (A. Kojic et al., 2018). It is a game-based learning tool for computational skills. sCool is mainly designed for mobile devices (currently Android devices) but is also available for Microsoft Windows. There are two different parts of the game that pursue different objectives. The game has

[7] https://grasshopper.codes/



```
turn 45
step 15
turn left
step 10
step -10
```

Figure 2.14: CodeyMonkey - Coding Adventure: Using CoffeeScript to reach the bananas



Student Progress

See your students' progress in the different courses.

Course: Coding Adventure Part I

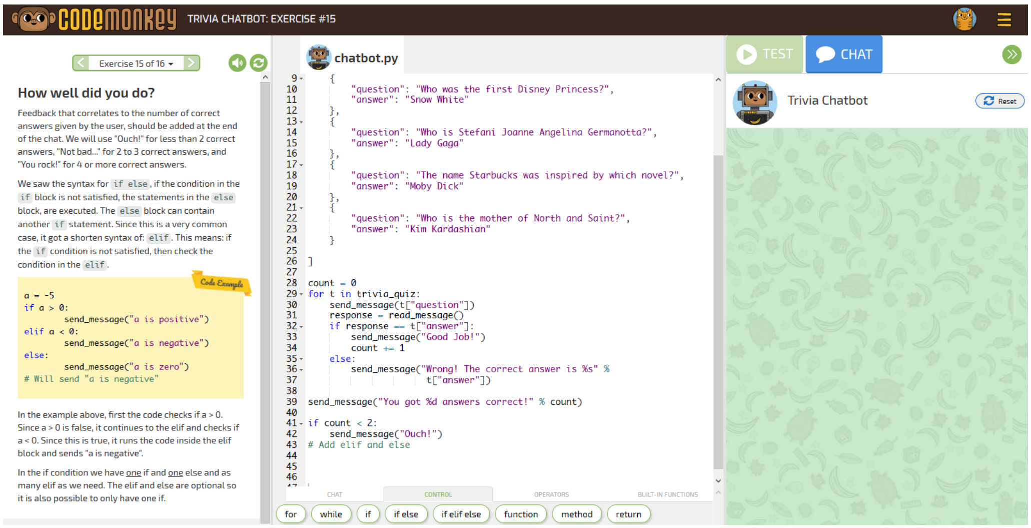
Coding Adventure Part I

★ = 3 stars ★ = 2 stars ★ = 1 star ⓘ = Failed solution ● = Assessment challenge

Sort students by		Stepping & Turning											turnTo, Turtle										Times-Loops										Variables			
NAME	PROGRESS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
1	xstone93	★	★	★	★	★	★	★	★	★																										

Click on a challenge number to view it's solution | Click on a student's name to view extended info

Figure 2.15: CodeMonkey - Teacher's Dashboard: Student's progress



How well did you do?

Feedback that correlates to the number of correct answers given by the user, should be added at the end of the chat. We will use "Ouch!" for less than 2 correct answers, "Not bad..." for 2 to 3 correct answers, and "You rock!" for 4 or more correct answers.

We saw the syntax for `if else`, if the condition in the `if` block is not satisfied, the statements in the `else` block, are executed. The `else` block can contain another `if` statement. Since this is a very common case, it got a shorten syntax of: `elif`. This means: if the `if` condition is not satisfied, then check the condition in the `elif`.

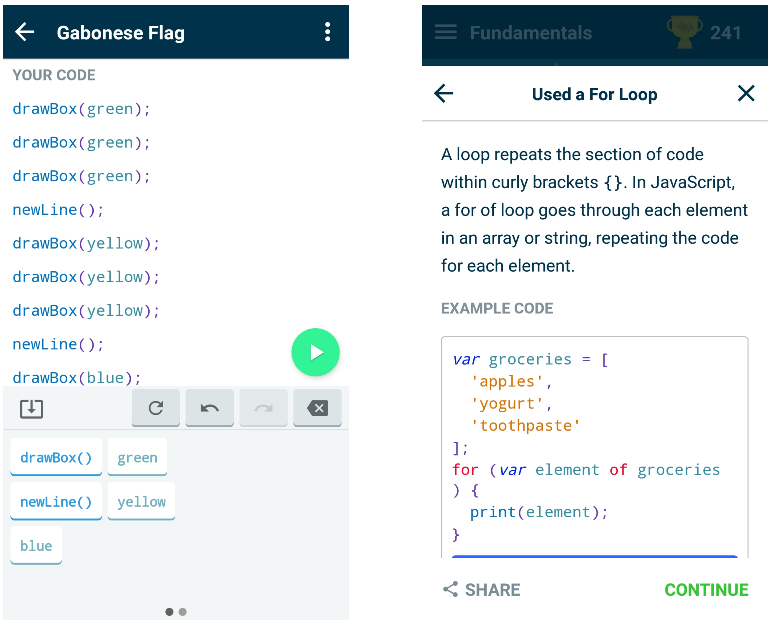
```
a = -5
if a > 0:
        send_message("a is positive")
elif a < 0:
        send_message("a is negative")
else:
        send_message("a is zero")
# Will send "a is negative"
```

In the example above, first the code checks if a > 0. Since a > 0 is false, it continues to the elif and checks if a < 0. Since this is true, it runs the code inside the elif block and sends "a is negative".

In the if condition we have one if and one else and as many elif as we need. The elif and else are optional so it is also possible to only have one if.

```
    {
        "question": "Who was the first Disney Princess?",
        "answer": "Snow White"
    },
    {
        "question": "Who is Stefani Joanne Angelina Germanotta?",
        "answer": "Lady Gaga"
    },
    {
        "question": "The name Starbucks was inspired by which novel?",
        "answer": "Moby Dick"
    },
    {
        "question": "Who is the mother of North and Saint?",
        "answer": "Kim Kardashian"
    }

]

count = 0
for t in trivia_quiz:
    send_message(t["question"])
    response = read_message()
    if response == t["answer"]:
        send_message("Good Job!")
        count += 1
    else:
        send_message("Wrong! The correct answer is %s" %
                     t["answer"])

send_message("You got %d answers correct!" % count)

if count < 2:
    send_message("Ouch!")
# Add elif and else
```

Figure 2.16: CodeMonkey - Coding Chatbots: Programming a chatbot using Python



```
drawBox(green);
drawBox(green);
drawBox(green);
newLine();
drawBox(yellow);
drawBox(yellow);
drawBox(yellow);
newLine();
drawBox(blue);
```

A loop repeats the section of code within curly brackets {}. In JavaScript, a for of loop goes through each element in an array or string, repeating the code for each element.

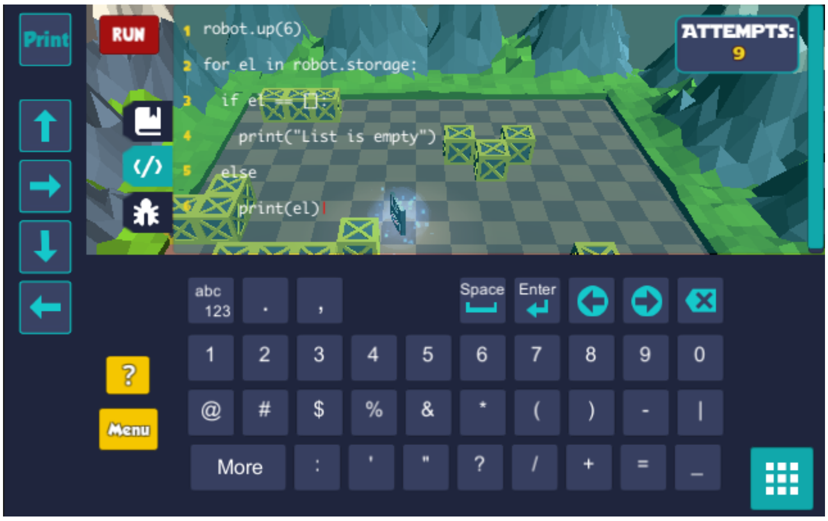
```
var groceries = [
  'apples',
  'yogurt',
  'toothpaste'
];
for (var element of groceries
) {
  print(element);
}
```

(a) Code Editor (b) Concept-learning

Figure 2.17: Learning Coding with Grasshopper

a coherent narrative structure that combines both modes together. A space mission failed and the shuttle crashed on a foreign planet. The players are in the role of a space team member and have to support the crew to repair the shuttle and escape from the planet. In the concept-learning part the players are introduced to new concepts. This is achieved by a procedural generated game world where the players have to find disks that represents concepts. The disks are guarded by enemies that have to be defeated. After collecting all disks the concepts are presented in a textual form. After reading the text the learners have to answer a related question. The second playing mode are the practical missions where the players have to apply the previous learned concepts. The playground is represented via a two-dimensional grid where they have to control a robot. The overall goal in each level is to reach a disk and solve different tasks on the way. To control the robot the players have to use certain instructions that represent commands in the Python programming language. To simplify the task of coding, blocks can be dragged into an editor. After dragging the block is converted into a command in Python that can be executed by the robot. Next to this existing code blocks it is possible to use a virtual keyboard to write one's own code (Steinmaurer, Pirker, and Gütl, 2019b).

A key feature of sCool is its adaptive content that can be modified by the educators. Therefore, a web platform is used where each course is represented in a hierarchical skill-tree. The teachers can define the content for both the concept-learning part and the practical missions. In this way they can provide a very individual learning experience that can be geared to the needs of the course. This web platform also



```
robot.up(6)
for el in robot.storage:
  if el == []:
    print("List is empty")
  else
    print(el)
```

Figure 2.18: sCool - Practical mode

provides a comprehensive assessment and analytics tool for the courses. The educators can analyze the learning progress of each individual player and can receive information regarding the communication with the system.

2.5.6 Overview

After considering the different educational video games it can be concluded that each tool has its certain purpose and focus on learning. Table 2.3 shows that the games distinguish in the designated educational use and the level of complexity. In nearly every game the fundamental idea is similar, since the player is controlling an avatar (monkey, robot or warrior) with the use of programming skills and solve different tasks. The major purpose of LightBot for example is to focus on logical skills like sequencing or functions. Therefore, no certain programming language was used and the instructions were based on blocks. The tasks always remain the same but the difficulty increases the more concepts are learned. Grasshopper is more complex in both graphics and content and so the aim of the game is different. Compared to other games it does not focus on a story or a learning avatar. With its clear and handy tasks it mainly addresses adult learners. The players have to write simple code in JavaScript that is linked to a certain task (for examples drawing a flag). CodeMonkey and CodeCombat have the most complex narrative of the observed games. Based on this narratives different tasks can be solved while programming. Due to the complexity the games are available on a web browser because it is often necessary to type more lines of code. The approach of sCool is to split concept-learning and practical learning into separate parts. In this way it is possible for students to explore and learn concepts and apply them afterwards in a practical mission.

Game	Platform	Language	Concepts	Dashboard
CodeCombat	browser	Python and JavaScript	Fundamentals, Control Flow, Objects, Algorithms	yes
LightBot	browser, Android, iOS	block based	Fundamentals, Control Flow, Recursion	no
CodeMonkey	browser	Python and CoffeeScript	Fundamentals, Control Flow, Algorithms, Objects, Events	yes
Grasshopper	Android, iOS	JavaScript	Fundamentals, Control Flow, Animations, Graphics	no
sCool	Android, Windows	Python	Fundamentals, Control Flow, Algorithms, Objects	yes

Table 2.3: Comparing different game-based coding tools.

Another difference is the possibility of assessment and evaluating. Lightbot does not support any feedback when failing and allows endless repetitions. There are no hints when the players do not know further. Grasshopper gives the user visual feedback and provides hints on the basis of the input. sCool provides the player with textual feedback or interpreter output to make aware of errors. CodeCombat and CodeMonkey have a comprehensive dashboard for educators. In this dashboard it is possible to define courses and see the student's progress. The learning objectives are always predefined by the system and the educators do not have a possibility to define their own content and see the student's input and metrics for a particular task. The most distinctive feature of sCool is its adaptive content that can be modified according to the teacher's concept.

2.6 Summary

There is a major advantage in using educational video games in a learning situation in school. Especially when it comes to computational skills and computer science there is a large number of tools and approaches. In literature there are mainly four skills that are related to computational thinking (decomposition, pattern recognition, generalization and abstraction, and algorithm design). These skills are also covered in many countries K-12 curricula. Some authors suggest to add another (social) component that describes the socializing that takes place to create problem solving strategies with others.

When it comes to learning there is a large number of theories that try to explain different aspects of human learning. The most common approaches are behaviorism, cognitivism and constructivism. The introduced video games are based on one (or more) of these learning theories. The objective in *CodeCombat* for example is to explore a fantasy world. Exploring and building are central aspects in the constructivistic approach where the students have to solve a certain task. Another central aspect of learning is being involved in an activity. A high level of flow can be reached when there is a good balance between challenge and skill. When the state of flow is reached learning can be seen as an engaging activity.

The theories and approaches that were considered in this chapter represent the basis for the further development of sCool. To reach a higher level of player′s engagement, additional gamification and flow increasing elements were developed. The redesign of the pedagogical concept was adapted to the learning activities. This should motivate students to remain engaged when playing the game and impart knowledge in a learner-centered way.

3 Game-Based Learning with sCool

This chapter will introduce the educational mobile game sCool and explain the game design and implementation. The system consists of several components interacting with each other to provide an adaptive learning environment.

3.1 Game Design

The mobile video game sCool was developed in cooperation between Graz University of Technology and Westminster University (A. Kojic et al., 2018). It is a game-based tool to engage school students in learning computational skills. sCool consists of two different components: the mobile game and the web application. Figure 3.1 shows the architecture of the sCool environment. The mobile game consists of two game modes, each with a certain aim: i) concept-learning mode and ii) practical mode. The web application takes an important role since it provides the game with the learning content and all collected data concerning learning is sent to the server. All communication between the mobile game and the web application is transmitted over a Representational State Transfer (REST) API in JavaScript Object Notation (JSON) format. After each level all game-related data will be synchronised with the server. In this way educators have access to the learning analytics in real-time.

The game has a coherent story that links the different game types within the game. The overall game theme is space and exploring a foreign planet. A space shuttle crashed on a foreign planet and parts of it got lost. The players slip into the role of a space crew member and have to repair the shuttle. For this purpose they have to explore the planet and collect different pieces of information. Each piece of information represents a learning content that helps the players to understand a certain concept. When the players passed the concept-learning missions and understand this content, they apply their knowledge in the practical missions. In this game type they have to control a robot by using the programming language Python and understand computational thinking skills.

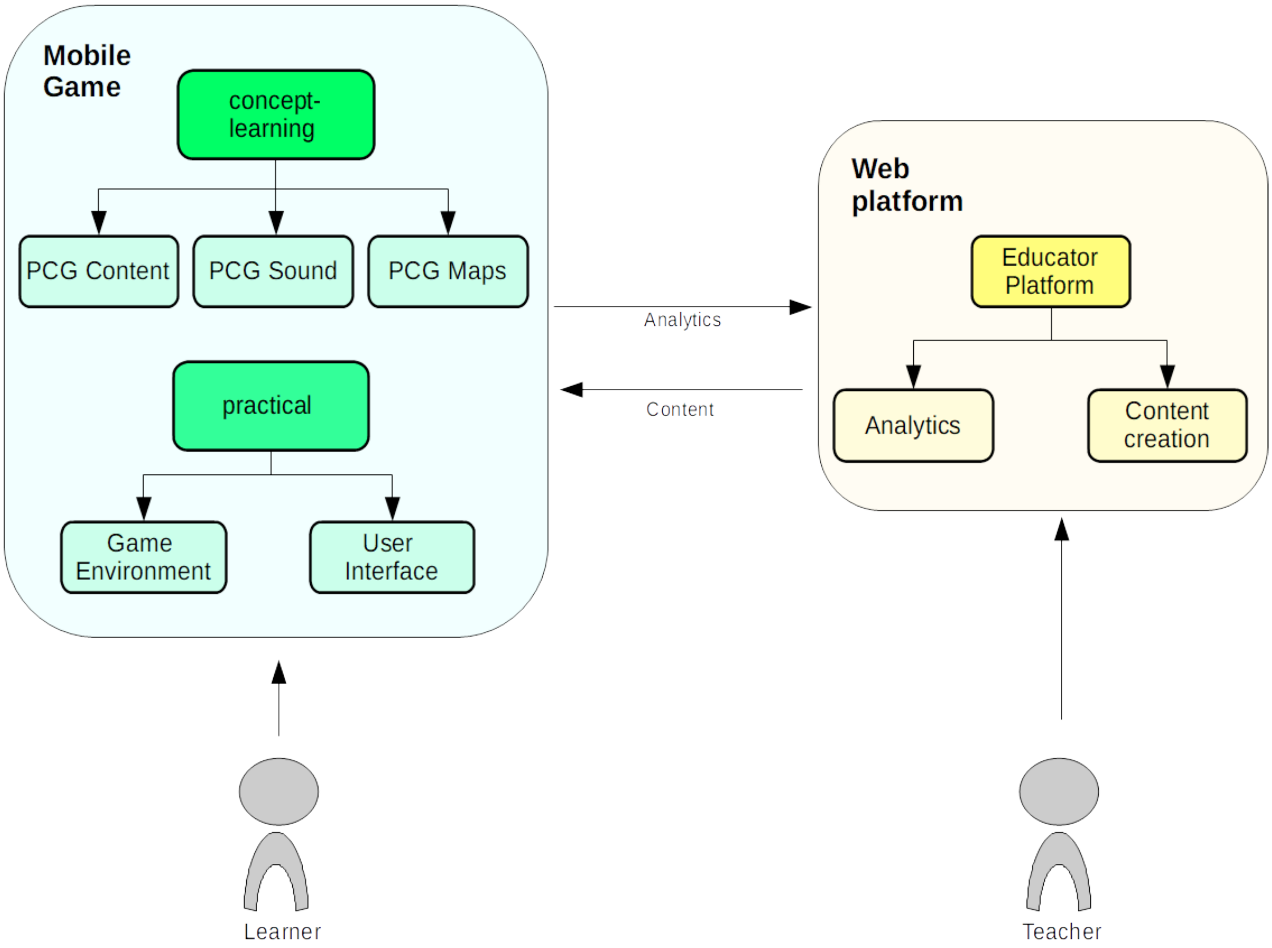


Figure 3.1: sCool - System Architecture (adapted after Steinmaurer, Pirker, and Gütl (2019b))

3.1.1 Concept-learning Mode

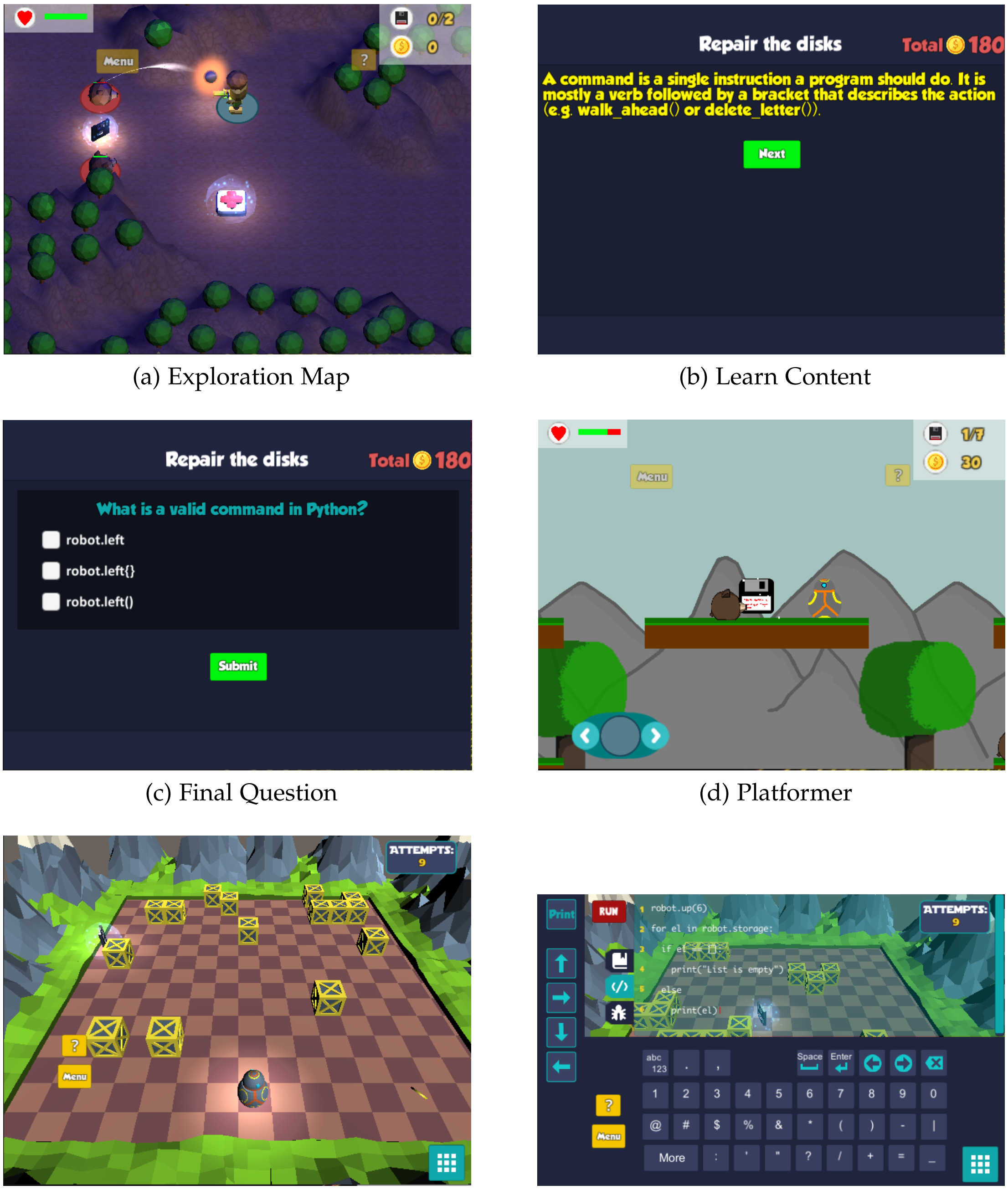
In the concept-learning mode the students are faced with the mission of discovering a hostile planet (A. Kojic et al., 2018). In this three-dimensional game mode the players are controlling a character over a game map (see Figure 3.2a). The map consists of four interactive game elements: character, enemies, disks, and first aid boxes. The aim is to collect different pieces of information in form of disks. These disks contains different information that is needed to escape the planet. Each map has at least two disks that are defended by enemies. Both the game map and the playground are generated based on procedural content generation algorithms (Cellular Automata and Perlin Noise) so that they are different every time, which makes the game more re-playable. The level of difficulty can be pre-defined in the web application, so the missions will get more complex. When the level of difficulty is increased the size of the playable map will be extended and the number of collectable disks will change as well. After collecting all disks new content is presented in textual form (see Figure 3.2b). The player has to go through this text and answering a following single choice question (see Figure 3.2c). When this question is answered correctly - the mission qualifies as passed - otherwise the whole level has to be repeated until it is done right. In this way the learning concepts are studied by completing the levels. When a task is passed the players receive a certain amount of virtual currency (coins). They can use it to answer questions they got wrong before or buy something in the in-game store.

Besides this game type there is another mode called *platformer* (Pöckelhofer, 2019). It is up to the players to decide which type they rather want to play since the learning content is equal in each type. The concept of the two-dimensional game map is based on the above mentioned game type. The players have to explore a world with different platforms and search for disks (see Figure 3.2d). The disks and platforms are protected by two types of enemies (movable and static) that can harm the player. After successfully passing the level the concept is presented in the same (textual) form with a final question.

3.1.2 Practical mode

In the practical mode the players have to apply the previous learned concepts (A. Kojic et al., 2018). There are two main components: the game environment and the user interface. The game environment is represented by the squared playground where the robot is placed (see Figure 3.2e). The environment has three interactive game objects: a robot, obstacles and a disk. Starting from the player's position the disk has to be reached. On the playing field a number of obstacles (boxes) is placed that have to be avoided.

The user interface enables the interaction with the game environment by giving the robot certain instructions. The robot can be controlled via different command



(a) Exploration Map

Repair the disks

Total 180

A command is a single instruction a program should do. It is mostly a verb followed by a bracket that describes the action (e.g. walk_ahead() or delete_letter()).

(b) Learn Content

Repair the disks

Total 180

What is a valid command in Python?

□ robot.left

□ robot.left{}

□ robot.left()

(c) Final Question

(d) Platformer

(e) Practical Mode

```
robot.up(6)
for el in robot.storage:
  if el == []:
    print("List is empty")
  else
    print(el)
```

(f) Coding in Practical Mode

Figure 3.2: sCool Game Types

blocks that get dropped into a code editor (see Figure 3.2f). Each block represents an instruction in Python that is executed by the robot. There are four different block types: print, variable declaration, move, and control structures. It is also possible to change the code in the editor via a virtual keyboard to generate custom output. The editor also includes a debugging tool that analyses the code and passes the interpreter's output to the user. This makes it possible to program in the practical mode.

3.1.3 Web Application

The .NET-based web application is a platform for educators (A. Kojic et al., 2018). Teachers can use it to provide new educational content, create new courses or edit existing courses. Each course is represented as hierarchical skill-tree. These skills are superior elements for all concept-learning and practical tasks. Different concepts can be mapped to a specific skill tree. The content and the degree of difficulty can be declared in the web application and the maps are generated based on these parameters. A distinction is also made between concept-learning and practical tasks on the web platform. In the concept-learning part educators can define learning content and the corresponding single choice question for the players. Additionally, the difficulty level for the exploration mode is appointed, so the map is generated according to that value. For the practical tasks, educators can define the tasks' goal and their reference output. This solution represents the expected output of the Python code to successfully finish the level. In this section it is also possible to define the accessible command blocks in the levels.

The web platform also provides an assessment and analytics tool for educators to analyse the student's learning process and gain a detailed evaluation of the course. Educators can see different metrics for each user. This makes it possible to see number of attempts, provided code, time duration, etc.

3.1.4 Gamification Elements

sCool implements a couple of elements that make the game more re-playable. These gamification elements should engage the players to make more achievements in the game (S. Kim et al., 2018, p. 127).

- **Virtual Currency**: After successfully passing a concept-learning or practical task the players receive coins.
- **Avatars**: The coins can be used to customize the avatar. The players can change the avatar's appearance (color, face, etc.) and buy additional equipment like weapons or clothes.
- **Leaderboard**: A highscore list with a course wide ranking will display the user's name and points.

- **Experience Points**: After passing a level the experience points are increased, which can lead to a new level in the game.

3.2 Implementation

This section covers selected aspects of the implementation of sCool (A. Kojic, 2017; M. Kojic, 2017). In terms of this thesis the focus will be on implementation features that are part of the further development in the didactical concept. The mobile video game is built with the Unity game engine and written in C#. The web application was built using ASP.NET MVC web framework. The UI was developed on the base of HyperText Markus Language (HTML), Cascading Style Sheets (CSS), and JavaScript (JS). Several libraries and frameworks were used in addition: jQuery and Bootstrap for the functionality of the platform and design, Flot UI and Morris.js was used for displaying the learning analytics and charts. The communication between web platform and video game is done by ASP.NET Web API framework. This framework supports the REST interface and the serialization with JSON.

3.2.1 Practical mode

In the practical mode the players can apply the learned concepts from the explorative (concept-learning) mode (M. Kojic, 2017). Therefore, they have to solve tasks programmatically. The coding tasks are done with the programming language Python. To use Python within the game environment the open-source implementation IronPython[1] is used. This implementation is fully written in C# and provides access to Python in the .NET framework or Mono. The practical mode is divided into two main component groups: game environment and user interface system. Since sCool is mainly developed for mobile devices, certain design decisions were taken that will be explained in this section.

The UI system consists of the following components (see Figure 3.3a):

- Code blocks
- Editor
- Virtual Keyboard
- Scrollbar
- Constraint Label
- Tabs
  - Description Tab
  - Code Tab

[1] https://ironpython.net/

- Output Tab
- Buttons
  - Run Button
  - Tutorial Button
  - Menu Button
  - Open/Close Button

The communication between the environment and the player happens in the editor window. In total there are three different editor behaviours that differ in the application area. The *description tab* formulates the task's objective according to the content that the teachers define at the web platform. The actual programming takes place in the *code tab* where the players are able to interact with the code editor. There are two ways to write code into the editor: code blocks and virtual keyboard. The code blocks are a placeholder for a code that will be displayed in the editor when a button is dragged and dropped. Each block represents a command in the Python programming language. In total there are four categories of blocks: print command, variable declaration, move commands (left, right, up, down), and control flow (loops and conditions). Beside using blocks the players can write Python code with the virtual keyboard. The virtual keyboard's layout is orientated on a native keyboard so that all users are familiar with the handling. Figure 3.3b shows all possible keys that can be used on the virtual keyboard. By pressing the shift button the characters appear in upper case. There is also a comprehensive range on special characters that are necessary in terms of programming (for example for using logical expressions). The *output tab* is opened when the users press the run button. Before the code is executed it is passed to the Python interpreter that performs a syntax check first. The corresponding message from the interpreter (true/success or false/fail) is displayed in the editor window (see Figure 3.3c). Listing 3.1 shows how the code is processed before it is passed to the Python interpreter. First it is formatted into an interpretable string and in line 12 the code is passed to the IronPython interpreter. According to the result when interpreting the code (succeed or error) the output is presented into the output editor.

The game environment consists of a 15x15 grid presented in the playing field, and its elements can be accessed by programming (M. Kojic, 2017). On top of the playground there are three interactive components:

- **Robot**: The robot can be controlled by giving commands in the language Python.
- **Disk**: To finish a given level the disk has to be reached.
- **Blocks**: The robot has to avoid these obstacles on the way to the disk.

The spawning point of the robot is static in the lower middle of the playing field (M. Kojic, 2017). In every practical mission the number of boxes is 15. The position is generated randomly using the .NET Random library. The position of the disk depends on the level of difficulty (see Listing 3.2). The disk's x position is randomly generated between position 1 to 15. The y position is set on base of the given level of difficulty. The playground is divided into thirds to place the disk.

```
public void ContentForCompiler()
{
  typedCode = "";
  RemoveRedSeparator();
  Text[] text = GetComponentsInChildren<Text>();
  foreach (Text t in text) {
    if(t.transform.parent.name != "CodeLineNumber")
      typedCode += GetIndent(t) + t.text + "\n";
  }
  string[] res = PythonBase.Instance.Run(typedCode);
  outputTxt.text = errorTxt.text = "";
  if (res[0] == "true") {
    playBtn.SetActive(true);
    outputTxt.text = res[1];
  } else {
    playBtn.SetActive(false);
    errorTxt.text = res[1];
  }
}
```

Listing 3.1: Processing Python Code

```
int AdjustY()
{
  int y;
  if (PracticeManager.Instance.practiceTask.Difficulty < 50)
  {
    y = Random.Range(10, 14);
  }
  else if
    (PracticeManager.Instance.practiceTask.Difficulty >= 50 &&
    PracticeManager.Instance.practiceTask.Difficulty < 80)
  {
    y = Random.Range(5,10);
  }
  else
  {
    y = Random.Range(0,5);
  }
  return y;
}
```

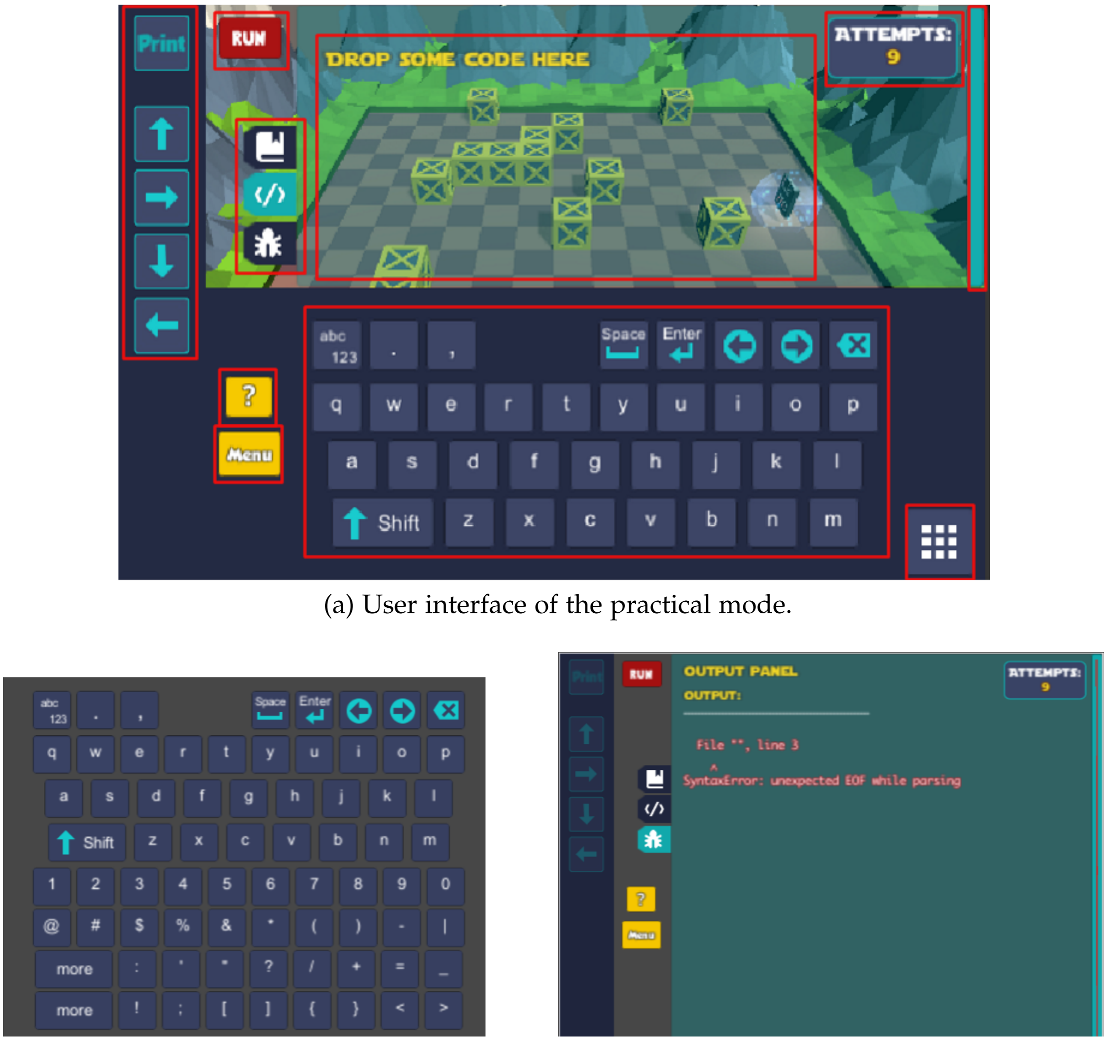
Listing 3.2: Generation of Disk Position

In every practical task it is necessary to reach the disk. Additionally the teachers can determine that a certain output has to be provided by the players. When the disk is reached (or rather the box collider is triggered) a *CompareStrings* method is called that compares the given output and the reference string with each other.

Apart from reserved language elements in Python it is possible to call additional objects. For the robot to be able to move on the playing field the players need to use one of four moving commands (robot.up, robot.down, robot.left, robot.right). The object *robot* is a C# object that is passed into the IronPython environment. The method *SetVariable* passes the object *robot* into IronPython and makes it accessible in the code editor (see Listing 3.3). In this way properties and methods (especially moving the robot) of this object can be accessed.

```
PythonBase.Instance.Scope.SetVariable("robot",
  value: Robot.Instance);
```

Listing 3.3: Pass Objects to IronPython



```
File "", line 3
^
SyntaxError: unexpected EOF while parsing
```

(a) User interface of the practical mode.

(b) Virtual keyboard with all possible (lower case) keys.

(c) Output tab with syntax errors.

Figure 3.3: Practical mode

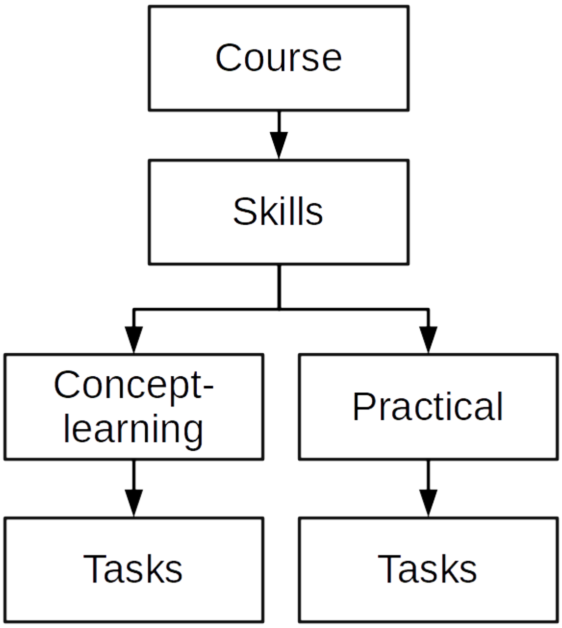


Figure 3.4: Hierarchical Course Tree

3.2.2 Web Application

The application is written in ASP.NET MVC web framework[2], HTML5, CSS, and different JavaScript libraries (A. Kojic, 2017). The application is hosted on Microsoft's cloud computing platform *Azure*[3]. As central storing technique a Microsoft SQL database system is used. This database stores all relevant information regarding the game content and the analytics. The courses are structured in a hierarchical tree (see Figure 3.4). Each course consists of one or more skills that represent concepts. A skill can have either a concept-learning part or a practical mission. Different aspects of this skill can be learned in several concept-learning levels where the learners receive information about this concept. The educators can create tasks to apply these concepts in a practical way.

Web Platform

The web application provides educators with the opportunity to manage and analyze all courses and users (A. Kojic, 2017). With the ASP.NET Model, View, Controller (MVC) framework a separation of data and business logic, user interface, and user interaction handling is possible. ASP.NET Core MVC supports routing which allows to define URL patterns that are routed to a certain controller. Listing 3.4 shows that all URLs must have a certain pattern: {*controller*}/{*action*}/{*id*}. The first parameter after the

[2] https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-3.0

[3] https://azure.microsoft.com/en-gb/

domain name has to be the name of the controller, the second one defines the action that has to be performed and the third is a identifier for the action.

```
public static void RegisterRoutes(RouteCollection routes)
{
  routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

  routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index",
      id = UrlParameter.Optional }
  );
}
```

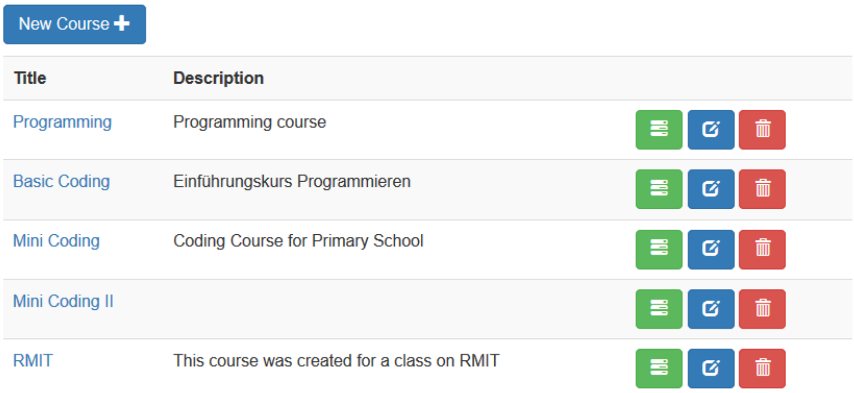
Listing 3.4: Configuration of Routing in MVC.

The generation of the *View* component is done with the Razor view engine[4]. Razor provides a markup syntax that can be embedded into C# code. In this way a template with corresponding data generates HTML output. Following routes can be used in order to generate views:

- Account
- Courses
- Home
- Manage
- PracticeTask
- Skills
- Students
- TheoryTask

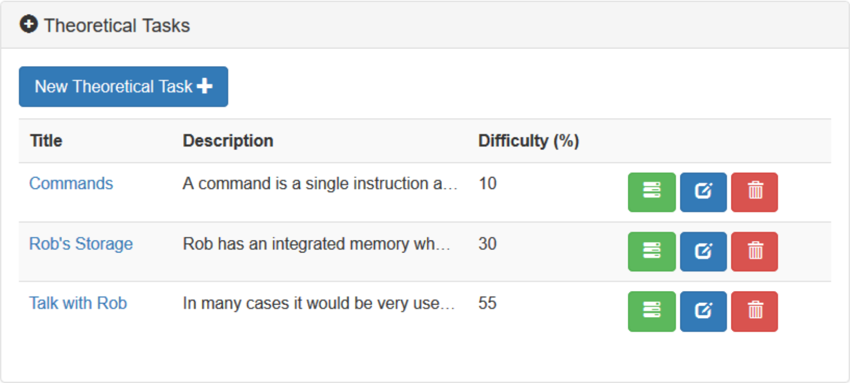
The course management enables to create, modify or delete courses (A. Kojic, 2017). Figure 3.5 shows the *New Course* button to create a new one. When pressing this button the educators are asked to fill in a title and description. This course is assigned to the educator and is only accessible within the account. Each course consists of a certain number of skills that are based on each other. If at least 66% of all tasks within a skill is reached the next skill will be unlocked. Every skill may involve concept-learning and/or practical tasks. They are listed under the corresponding skill (see Figure 3.6 and 3.7) and can be created individually. As Figure 3.8 shows each concept-learning mission has a title, description, task, answers, and a level of difficulty. The practical missions (see Figure 3.9) consist out of a title, description, task, solution, level of difficulty, and enabled blocks.

[4] https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-3.0



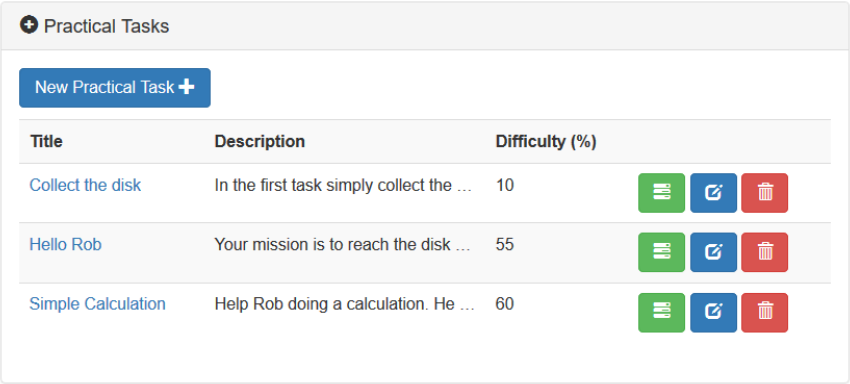
Title	Description
Programming	Programming course
Basic Coding	Einführungskurs Programmieren
Mini Coding	Coding Course for Primary School
Mini Coding II	
RMIT	This course was created for a class on RMIT

Figure 3.5: This view shows the *New Course* button to create a new course and displays a list with all available courses for a given user.



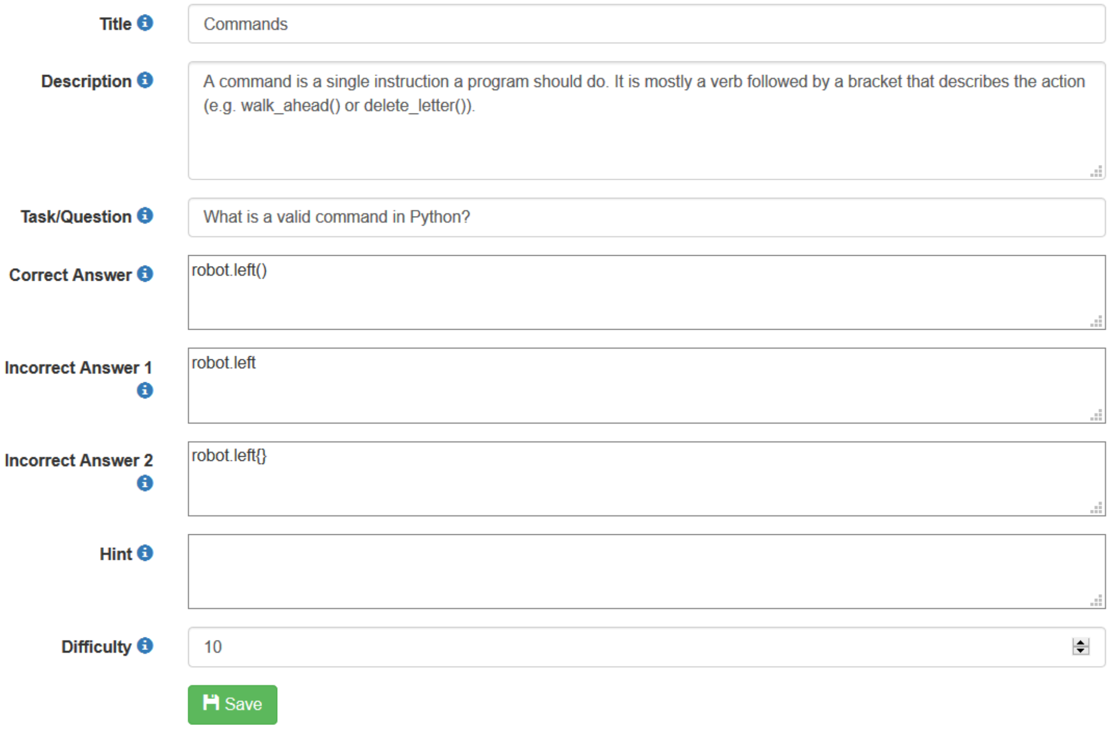
Title	Description	Difficulty (%)
Commands	A command is a single instruction a...	10
Rob's Storage	Rob has an integrated memory wh...	30
Talk with Rob	In many cases it would be very use...	55

Figure 3.6: Overview of the concept-learning tasks in the corresponding skill.



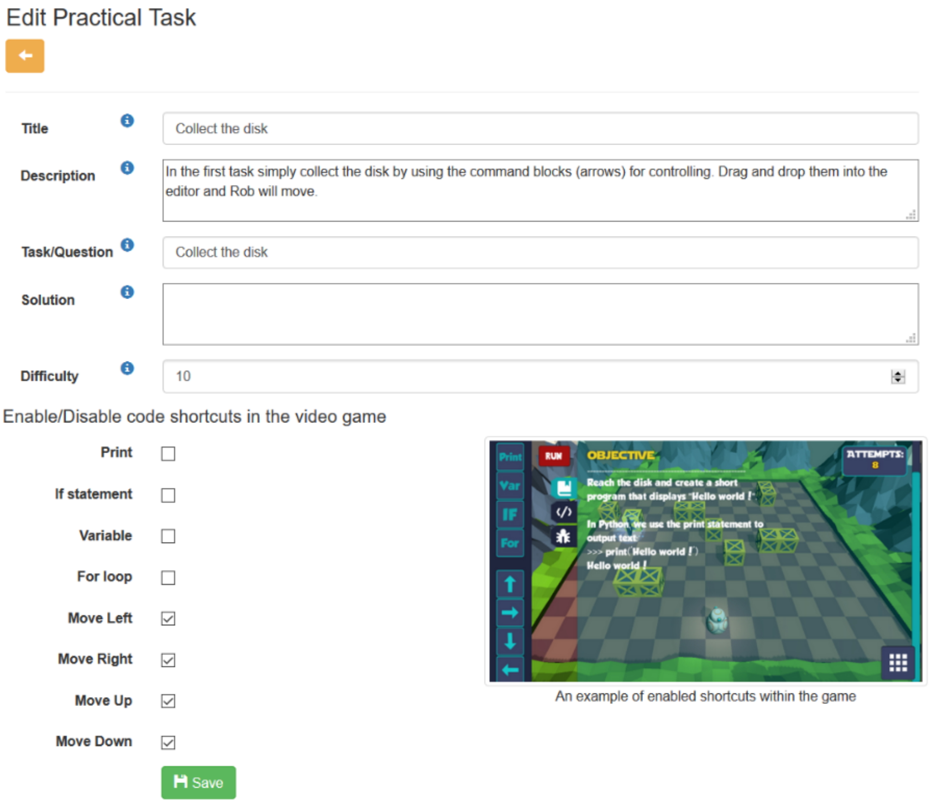
Title	Description	Difficulty (%)
Collect the disk	In the first task simply collect the ...	10
Hello Rob	Your mission is to reach the disk ...	55
Simple Calculation	Help Rob doing a calculation. He ...	60

Figure 3.7: Overview of the practical tasks in the corresponding skill.



Title	Commands
Description	A command is a single instruction a program should do. It is mostly a verb followed by a bracket that describes the action (e.g. walk_ahead() or delete_letter()).
Task/Question	What is a valid command in Python?
Correct Answer	robot.left()
Incorrect Answer 1	robot.left
Incorrect Answer 2	robot.left{}
Hint	
Difficulty	10

Figure 3.8: Form to create a new concept-learning mission.



Edit Practical Task

Title	Collect the disk
Description	In the first task simply collect the disk by using the command blocks (arrows) for controlling. Drag and drop them into the editor and Rob will move.
Task/Question	Collect the disk
Solution	
Difficulty	10

Enable/Disable code shortcuts in the video game

Print	□
If statement	□
Variable	□
For loop	□
Move Left	☑
Move Right	☑
Move Up	☑
Move Down	☑

OBJECTIVE

Reach the disk and create a short program that displays "Hello world !"

In Python, we use the print statement to output text

```
>>> print("Hello world !")
Hello world !
```

An example of enabled shortcuts within the game

Figure 3.9: Creation of practical mission.

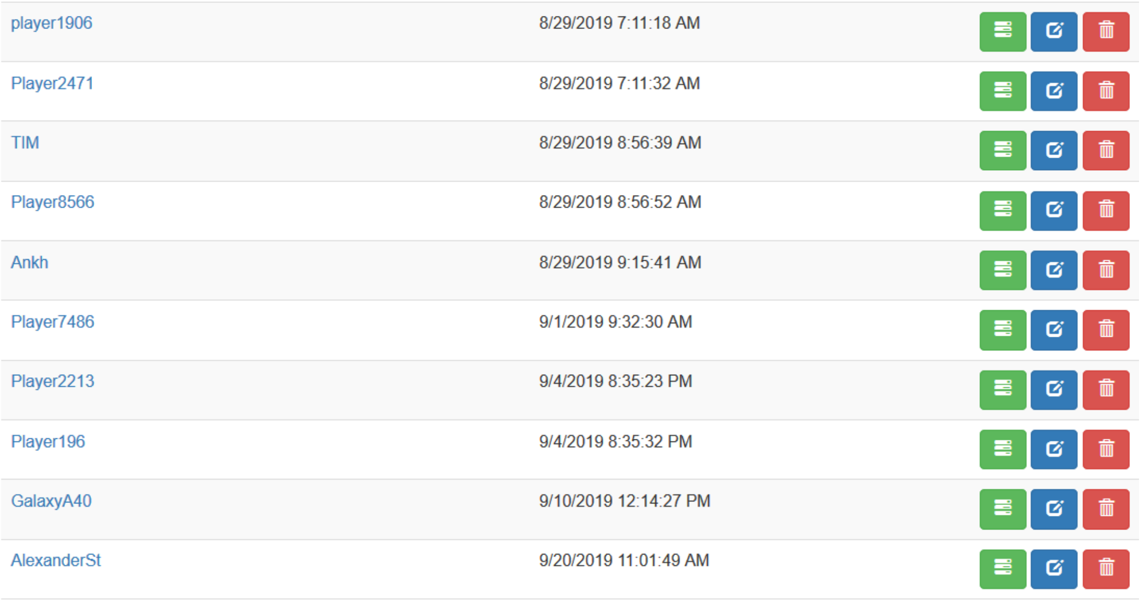
In the course and student analysis tool various statistics can be used to see the learning progress of each student. Figure 3.10 lists all users that are enrolled in the course. For a more detailed overview all statistics to concept-learning and practical tasks can be accessed (see Figure 3.11). In this way it is possible to analyse the overall performance in the game. Every single attempt (provided code and answers) can be seen on the web platform.

Interface
The communication between the client and server takes place over a REST API and is using the HTTP methods GET, PUT, and POST for receiving and manipulating data (A. Kojic, 2017). There are two types of requests: those which deal with the data (courses, tasks, highscores, etc.) and those which handle the student objects (register, list students, assign courses, etc.). Table 3.1 gives an overview of all possible API requests.

Since the data is stored in a relational database the JSON objects that are used for the communication are created for each request. The web platform implements an *Entity Framework* approach for object-relational mapping. This enables the usage of a complex JSON data structure within an object class. Each model class is mapped to a corresponding table in the database. All model classes are derived from *BaseEntity* which extends all objects with two properties: *CreatedAt* and *UpdatedAt*. This makes it possible to automatically add and update date and time. Listing 3.5 shows the model class for a course which consists of different properties. Figure 3.13 shows the JSON object that is serialized with the content of the Microsoft SQL database. Figure 3.12 shows the entity *Course* and the corresponding entity in an Entity Relationship (ER) diagram.

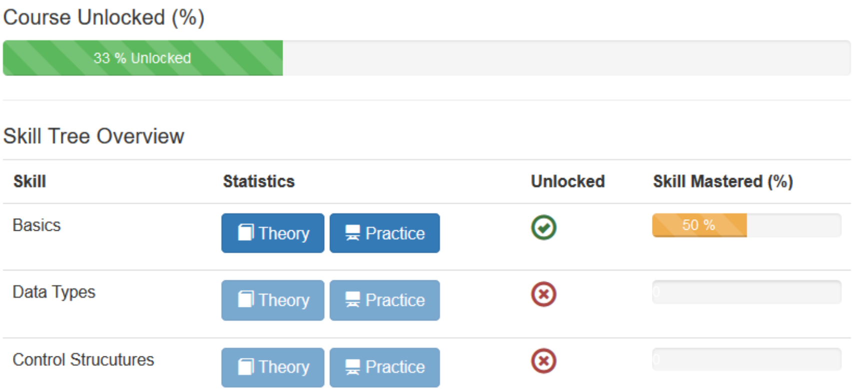
```
public class Course : BaseEntity
{
  public Course()
  {
    Skills = new List<Skill >();
    Enrolleds = new List<Enrolled >();
  }
  [Key]
  public int CourseId { get; set; }
  [Required]
  public string Title { get; set; }
  public string Description { get; set; }
  public virtual ApplicationUser User { get; set; }
  public virtual ICollection<Skill> Skills { get; set; }
  public virtual ICollection<Enrolled> Enrolleds { get; set; }
}
```

Listing 3.5: Sample class model for the course model.



player1906	8/29/2019 7:11:18 AM	
Player2471	8/29/2019 7:11:32 AM	
TIM	8/29/2019 8:56:39 AM	
Player8566	8/29/2019 8:56:52 AM	
Ankh	8/29/2019 9:15:41 AM	
Player7486	9/1/2019 9:32:30 AM	
Player2213	9/4/2019 8:35:23 PM	
Player196	9/4/2019 8:35:32 PM	
GalaxyA40	9/10/2019 12:14:27 PM	
AlexanderSt	9/20/2019 11:01:49 AM	

Figure 3.10: List of all enrolled users in the course.



Course Unlocked (%)

33 % Unlocked

Skill Tree Overview

Skill	Statistics	Unlocked	Skill Mastered (%)
Basics	Theory Practice	✓	50 %
Data Types	Theory Practice	⊗	0
Control Strucutures	Theory Practice	⊗	0

Figure 3.11: Learning progress of a single student.

Students		
api/Students/:id	GET	Receive the student object with all enrolled courses and the related skill tree. The tree stores the learning data and all concept-learning and practical tasks.
api/Students/:id	PUT	Modify a student object.
api/Students/	POST	Create a new student object and enroll it in all courses of all administrators.
Data		
api/Data/GetTheoryTask	GET	Receive information about a user's practical-learning tasks. This will include all tasks and the level of knowledge.
api/Data/AnswerTheoryTask	POST	Calculate various values regarding learning and unlocking new skills (if threshold is exceeded).
api/Data/GetPraticeTask	GET	Get all practical tasks related to a certain user.
api/Data/AnswerPracticeTask	POST	Calculate the given data from the practical missions and update the learning objects.
api/Data/Highscores	GET	Generates an ordered list (name and points) of all students that are enrolled in a certain course.
api/Data/JoinCourse	GET	Enroll for a certain course, based on a given token.
api/Data/SaveDetails	GET	Update name or email address of a certain user based on the student id.

Table 3.1: This table lists all possible API calls and includes the paths, http method, methods and an explanation of the behaviour.

The database consists of the following entities that are used in terms of the sCool system:

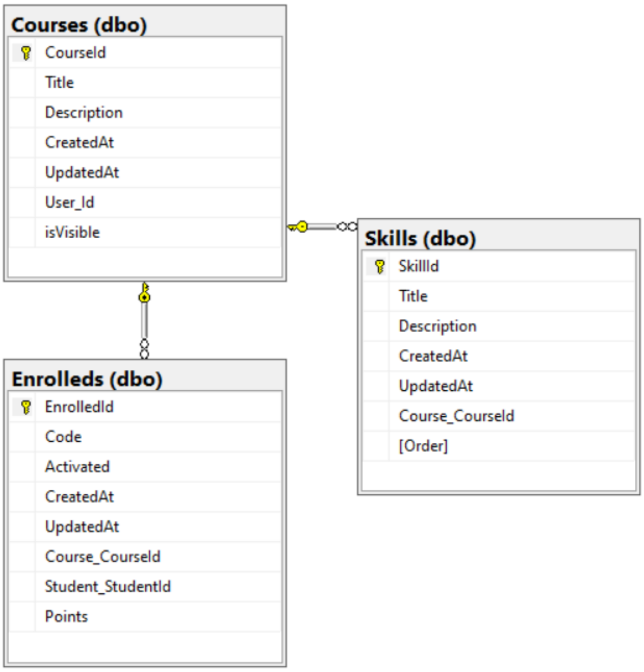
- **Courses**: Stores all titles and descriptions of courses.
- **Enrolleds**: Joining table that assigns the enrolled courses to the according users.
- **Learnings**: Contains all parameters regarding learning and analytics.
- **Logs**: JSON objects are stored with different information of the sCool video game.
- **PracticeStatistics**: All relevant information regarding the practical mode (answers, disk position, entered code, etc.) is stored.
- **PracticeTask**: Definition of each practical mission with objectives and various settings.
- **Skills**: Record the skills and matches them to the corresponding course.
- **Students**: Name and email address of all registered students.
- **TheoryStatistics**: All relevant information regarding the concept-learning mode (various positions, results, points, etc.) is stored.
- **TheoryTask**: Stores information to generate the concept-learning levels.

3.3 Summary

sCool has a comprehensive architecture that consists of a mobile video game and a web platform. The mobile video game has a concept-learning part and a practical part. Each part serves a certain purpose in terms of learning. In the concept-learning part learners acquire new skills in an explorative way. In the practical missions the learned skills have to be applied using the programming language Python to control a robot on a playing field.

The learning content is highly adaptive and can be modified by the educator using the web platform. The web platform includes a course management and a learning analytics tool. In the course management the learning content and all corresponding tasks can be created and modified. The learning analytics can help the educator to see the progress in the game and see the overall performance.

The communication between the mobile video game and the server happens over a REST API. This API is using HTTP methods (GET, PUT and POST) and JSON objects to send and receive data.

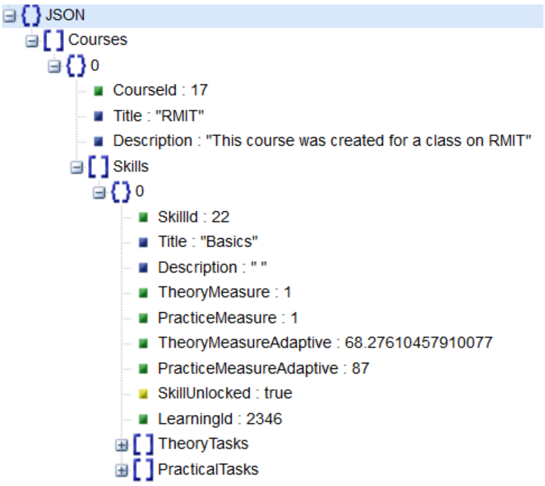


Courses (dbo)
- CourseId
- Title
- Description
- CreatedAt
- UpdatedAt
- User_Id
- isVisible

Skills (dbo)
- SkillId
- Title
- Description
- CreatedAt
- UpdatedAt
- Course_CourseId
- [Order]

Enrolleds (dbo)
- EnrolledId
- Code
- Activated
- CreatedAt
- UpdatedAt
- Course_CourseId
- Student_StudentId
- Points

Figure 3.12: Database model of the *Course* entity and related entities.



```
{} JSON
  [] Courses
    {} 0
      CourseId : 17
      Title : "RMIT"
      Description : "This course was created for a class on RMIT"
      [] Skills
        {} 0
          SkillId : 22
          Title : "Basics"
          Description : " "
          TheoryMeasure : 1
          PracticeMeasure : 1
          TheoryMeasureAdaptive : 68.27610457910077
          PracticeMeasureAdaptive : 87
          SkillUnlocked : true
          LearningId : 2346
        [] TheoryTasks
        [] PracticalTasks
```

Figure 3.13: JSON object of a sample course.

4 Pre-Evaluation

The introduced mobile video game sCool was evaluated in two Austrian schools. The aim of this evaluation was to receive information about engagement, emotions, motivation, usability, and didactic aspects. The first section covers the game's pedagogical concept and framework. It also covers the study design, the instruments and the procedure. Afterwards both experiments, the study participants and the results will be introduced. Finally, the findings of both experiments regarding all investigated aspects will be summarized. Some parts of this chapter are based on the case studies of Steinmaurer, Pirker, and Gütl (2019a) and Steinmaurer, Pirker, and Gütl (2019b).

4.1 Pedagogical Concept

Before working with sCool in an educational context it was necessary to develop a pedagogical concept. For this purpose two schools were selected to perform a pre-evaluation. The aim was to develop and test this new concept and evaluate the usage in classroom. Therefore, a user-centered survey regarding usability, game engagement, motivation, emotions, and gender-specific aspects was conducted. Another vital aspect was to find out if the video game can help school students to learn computational skills. For the practical part a real world example was used to create a game that engages its players. The following list contains the research questions that were related to the study:

- RQ1: Can sCool be used in school regarding usability and acceptability?
- RQ2: Do school students get motivated for coding by using sCool?
- RQ3: Are school students able to abstract the learned concepts in sCool and apply them to similar problem classes?

Settings and Instruments
A number of standardised questions were included in the final questionnaire to provide a comprehensive evaluation of different aspects related to sCool.

- Game Engagement Questionnaire (GEQ) (Brockmyer et al., 2009)
- Computer Emotion Scale (CES) (Kay and Loverock, 2008)
- Situational Motivation Scale (SIMS) (Guay, Vallerand, and Blanchard, 2000)
- gender-related aspects

	Experiment 1	Experiment 2
School Type	New Secondary School	Academic Secondary School (Lower Cycle)
Grade	7th	8th
Participants	18 pupils (11 girls, 7 boys)	12 pupils (6 girls, 6 boys)
Groups	9 groups	8 groups
Age	12-14 (M=12.72; SD=0.73)	13-15 (M=13.75; SD=0.59)

Table 4.1: This table gives an overview of all relevant information for each experiment (adapted after Steinmaurer, Pirker, and Gütl, 2019b).

- game-related questions
- open-ended questions

Both experiments were conducted in a workshop in schools with pupils. The sCool course was created in advance in coordination with the computer science teachers using the sCool web platform. The game-related data was sent to the sCool web application while students were playing the game. It is stored in a Microsoft SQL database and can be analysed by the web platform. All user-related data was collected using Google Forms[1]. After the experiment the questions related to the Game Engagement Questionnaire, the Computer Emotion Scale, and the Situational Motivation Scale was analysed by the open-source programming language R. It was mainly used to categorize all factors and evaluate the mean and standard deviation on the likert-based data. The game-related questions were analysed using Google Forms and Excel. The open-ended questions were examined and classified according to common features.

Procedure
The framework conditions for both groups were equal. Both classes had to do the same course in sCool. The time limitation in each class was 100 minutes (double lessons) for the whole experiment. At the beginning of the class the project and sCool was introduced briefly. After the introduction the school students had to form groups of two with at least one Android device. The game was installed using a download link of the game. After installing the game all groups were handed out an introduction worksheet (see Appendix) containing the first instructions and a brief explanation of their tasks. This sheet introduced the character *Rob the Robot* that acts as a guide through the game. Table 4.1 gives an overview of the general information for each experiment.

The school students had a total of 50 minutes for playing the sCool game. Within the given time they should complete five concept-learning and three practical missions. In the concept-learning part the following concepts were introduced: commands,

[1] https://www.google.com/forms/about/

#	Task	Concepts	Difficulty
	Concept-Learning Tasks		
1	Which command is used that Rob can say "Hello you"?	Print	20%
2	How can Rob store the word "Monday" in a variable?	Variables, Strings	30%
3	What is the result of the following calculation? x = (31 - (2*3))	Variables, Arithmetic	40%
4	How can Rob print the value of the variable 'donut'?	Print, Strings, Variables	50%
5	What code is used to count from 1 to 10?	Print, Loops	60%
	Practical Tasks		
1	Print the name of the Robot.	Print	20%
2	Store the result of the calculation 21+(3*3) in a variable and print it.	Print, Variables, Arithmetic	40%
3	Count from 20 to 30.	Print, Loops	55%

Table 4.2: This table shows all given tasks in the course *Basic Coding* (adapted after Steinmaurer, Pirker, and Gütl, 2019b).

variables, strings, basic arithmetic, and loops. The practical tasks were designed to review these concepts and apply them. Table 4.2 shows all tasks and the objectives in the game. After working with the game for 50 minutes the school students were asked to solve a worksheet (see Appendix) with a coding task. This task was similar to the previous one in the game. The aim of this worksheet was to figure out if the school students were able to transform the learned concepts and apply them to a similar field. The worksheet's task was to fix the robot's broken calculation module. They had to count from 3 to 30 in steps of 3 (three times table) in the programming language Python. The school students were informed that they do not have to pay attention to accurate syntax. (Steinmaurer, Pirker, and Gütl, 2019b)

The concept-learning and practical tasks cover different basic computational skills. After the students have passed the corresponding level they receive a new concept in a text form. The system asks them a question relating to the concept they have just learned. The concepts are embedded into the narrative and the space theme. In this way students should realize that the concepts can be applied in a coding context and that learning does fulfill a certain purpose. The school students should learn:

- How to print strings and variables,
- the idea of variables and assigning variables,
- fundamental arithmetic operations with variables, and
- repetition of certain commands.

With the aim to make students aware of syntactical features and capabilities of the code, snippets are attached to the textual concept. The level of difficulty is increased at 10 percentage for each concept-learning task, thereby the number of disks (and enemies) is also increased as well and so challenge and skill stay balanced. The practical tasks are designed to apply the knowledge of the previous learned tasks and solve a given problem. The overall goal in all sCool's practical missions is to reach the disk. In the first level the robot's name has to be printed out as well. This simple task is feasible with basic knowledge and should engage students to get familiar with the user interface (especially the blocks and code editor). The second level combines the concepts commands, variables and arithmetic. The aim is to make a calculation using a variable and print its result. The final task combines all previous learned concepts and also includes loops. In the last concept-learning mission the students learn how to count from one to ten. In the practical mission they are asked to do a similar task but count from 20 to 30. (Steinmaurer, Pirker, and Gütl, 2019a)

After the school students completed the worksheet they were asked to fill in an online questionnaire. All questionnaires were provided on Google Forms and answered individually. The questions covered some general questions, motivation, engagement, emotions, gender-specific questions and open-ended questions (see Appendix).

In cooperation with the school's computer science teacher a didactic consideration was created to formulate the workshop's goals, its didactic concept, and the procedure:

After a brief welcome and project presentation the distributed and signed consent documents will be handed in again. Afterwards the teachers will explain the installation of the game and provide the students with a handout. This handout should also give an overview of the two game types and introduce the character *Rob the Robot*. Since the game is only available on Android devices, groups of two will be formed so there will be at least one device per group. The students should also work together to explore and learn together. The group building will not be done by the teachers so students feel more comfortable with their peers. As soon as every group is finished with the installation the menu and user interface will be explained briefly. After this introduction the students are free to play the game for 50 minutes. If questions appear the students are asked to discuss them within the class and if it is not possible to resolve them the teachers will help them. After 50 minutes all students have to solve the worksheet individually. Thereby, they are faced with a coding task that is built on the previous learned concepts. This will show if it is possible to abstract the learned concepts and apply them in a similar problem class. After 10 minutes the students are asked to hand in the worksheet and do the prepared questionnaire. Since the class is in the ICT room the questionnaire will be provided via Google Forms. After completing all questions a conclusive discussion round will be held where all students can tell their experiences before they leave class. Table 4.3 gives an overview of the planned scheduling for the workshop. [2]

[2] This didactic consideration was developed together with the computer science teacher.

Time	Task	Method	Media
5′	welcome, introduction of the project, hand in the consent documents	frontal	-
10′	installation of sCool, hand out introduction worksheet	in plenum	handout
10′	introduce sCool and game modes, form groups	frontal, in plenum	smartphone
50′	play sCool and support if necessary	group	smartphone
10′	work on worksheets	individual work	worksheet
10′	answer questionnaire	individual work	questionnaire, computer
5′	final discussion round and goodbye	in plenum	-

Table 4.3: This table shows the schedule of the experiment.

The following didactical objectives were defined:

- Learn basic skills in programming
- Improve algorithmic/computational thinking
- Encourage social learning
- Consider practical aspects of computer science
- See smartphones and apps as a strategy to acquire learning content

4.2 Experiment 1: New Secondary School

The experiment was conducted at a 7th grade class on the New Secondary School in Steinerkirchen (Upper Austria) on 1st March 2018. In total the group consisted out of 18 school students (11 girls and 7 boys). Since there is no regular computer science class in the school, the students attended the elective course *ECDL*, where they practice for the European Computer Driving License. The class is taught in a double lesson (in total 100 minutes). The school does not have a WiFi connection available so it was necessary to organize a WiFi hotspot for the class. In advance it was also necessary to hand out the consent form for the parents. (Steinmaurer, Pirker, and Gütl, 2019a)

Results and Discussion
The results of the *Game Engagement Questionnaire* (Brockmyer et al., 2009) show that the school students were highly immersed (M=3.78, SD=1.55) while playing the game (see Figure 4.3). Another important factor in terms of learning is the level of flow (M=3.53, SD=1.50) which is the optimal experience and indicates a good balance between challenge and skill (Csikszentmihalyi, Abuhamdeh, and Nakamura, 2014). The Computer

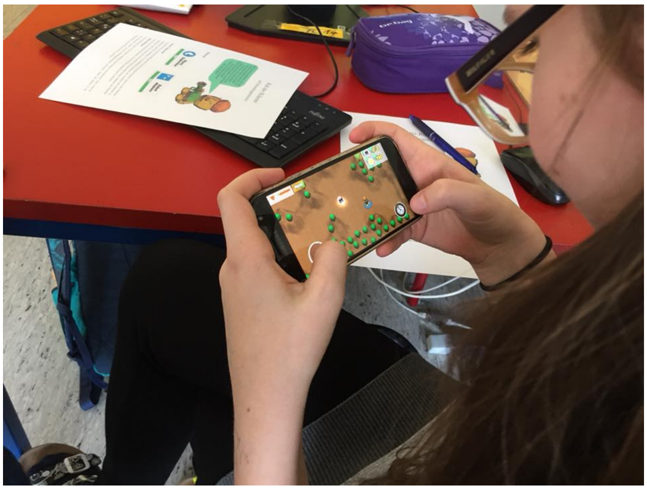


Figure 4.1: Student while playing the concept-learning part of the game.

Figure 4.2: Two students going through the game tutorial.

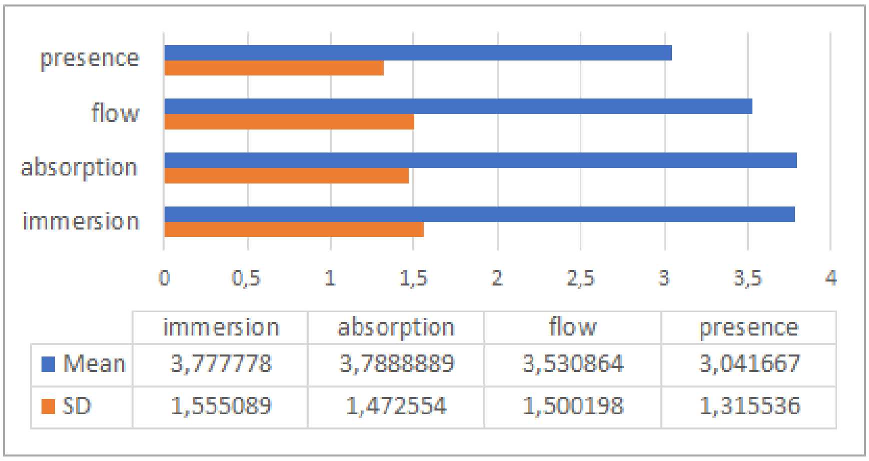
Emotion Scale (Kay and Loverock, 2008) is a twelve item questionnaire which results in the four base emotions that appear while learning a software: happiness, anger, anxiety, and sadness (see Figure 4.4). Playing sCool was mainly related with positive feelings (happiness, M=3.39, SD=0.63) while negative emotions (anger, anxiety, and sadness) are minor. The Situational Motivation Scale (Guay, Vallerand, and Blanchard, 2000) defines four types of motivation that are drivers for people to act (see Figure 4.5). The identified regulation is related to a form of extrinsic motivation. This means a person has identified and accepted something as important which influences the behaviour (Ryan and Deci, 2000). Its high level (M=5.26, SD=1.85) can be traced back to the overall context of the experiment. Since the experiment was taken in class the whole setting was not just self-regulated. Nevertheless, the level of identified regulation and intrinsic motivation (M=5.72, SD=1.27) is much higher than amotivation and external regulation which means that the motivation is not fully driven by external factors.

According to Steinmaurer, Pirker, and Gütl (2019a) all groups were able to succeed at all five concept-learning tasks. Only 33.39% of the groups were able to pass the task at the first try which can be traced back to the fact that many students skipped reading the instructions. Eight out of nine groups (88.89%) were able to pass the first practical mission. Another seven groups (77.78%) could master the second level were they had to do a simple calculation. The third task was successfully passed by just three groups (33.34%). After completing all tasks in the game the groups were asked to solve the worksheet. No group was able to solve the task and just one group had an approximate solution approach. The groups stated that they were not able to apply the concepts to another area.

Table 4.4 shows the results of the game related questions in the questionnaire. Most of the school students answered that they learned something new while playing the game and that they are encouraged to learn more about programming. Despite all school students were German-speaking they answered that the game's language was understandable. Most of the open-questioned feedback (5 out of 18) was related to the robot missions: *"Make the control at the robot missions easier.", "The robot missions were to complicated, it would be great to improve it."*

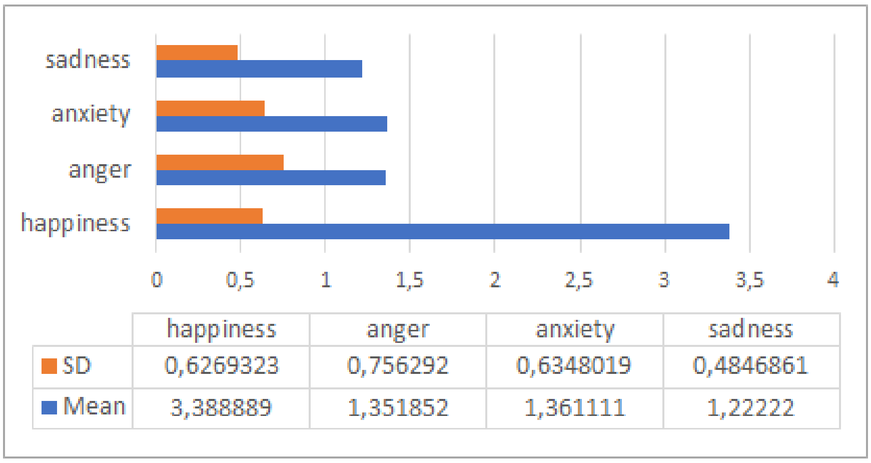
4.3 Experiment 2: Academic Secondary School

After running the first experiment (pilot test) in a New Secondary School the second pre-evaluation was conducted on the 20th June 2018 at a 8th grade Academic Secondary School in Graz (Styria). The group consisted out of 12 students (6 boys and 6 girls). Compared to the school students of the first group they were more familiar with game-based learning approaches in school since they attended the elective course *computer science* where they tried different (mobile) tools for learning. Three out of twelve (25%) students also answered that they already have some experience in programming



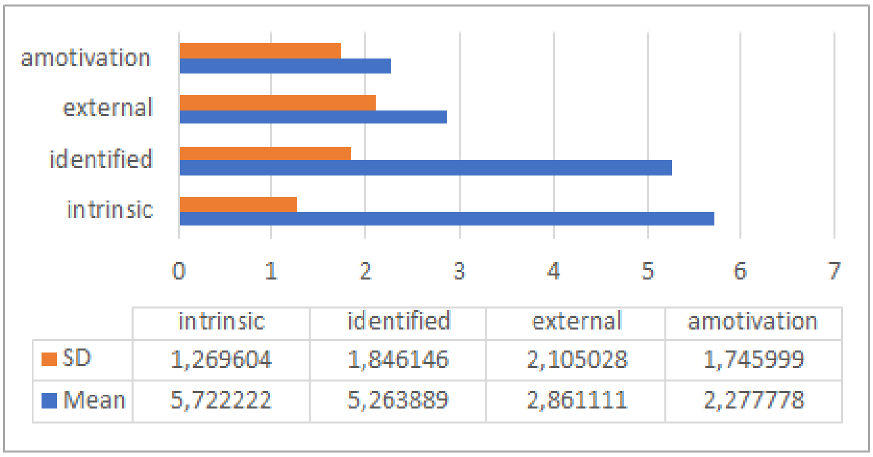
	immersion	absorption	flow	presence
🟦 Mean	3,777778	3,7888889	3,530864	3,041667
🟧 SD	1,555089	1,472554	1,500198	1,315536

Figure 4.3: Results of Game Engagement Questionnaire at Experiment 1.



	happiness	anger	anxiety	sadness
🟧 SD	0,6269323	0,756292	0,6348019	0,4846861
🟦 Mean	3,388889	1,351852	1,361111	1,22222

Figure 4.4: Results of Computer Emotion Scale at Experiment 1.



	intrinsic	identified	external	amotivation
🟧 SD	1,269604	1,846146	2,105028	1,745999
🟦 Mean	5,722222	5,263889	2,861111	2,277778

Figure 4.5: Results of Situational Motivation Scale at Experiment 1.

	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
The game's structure is coherent	7	5	5	1	0
The game's control is easy to use	5	5	6	2	0
The usage of the keyboard was easy	13	3	1	1	0
The language is understandable	4	7	7	0	0
The levels were easy to master	8	6	3	1	0
I learned something while playing the game	10	5	3	0	0
sCool encouraged me to learn more about programming	12	3	2	1	0
The work sheet was easy to solve	7	2	6	2	1

Table 4.4: This table shows the results of the game related questions of Experiment 1.

(Scratch and Java). The school had a better IT infrastructure and in this way the school students could use the school's WiFi. (Steinmaurer, Pirker, and Gütl, 2019b)

The framework conditions remain the same than in the first experiment. The lesson lasted 100 minutes and the procedure is equal to the schedule in table 4.3. Since the majority of the students had Android devices there were eight groups in total.

Results and Discussion

The results of the *Game Engagement Questionnaire* (Brockmyer et al., 2009) show that the school students were less engaged than the students in the first experiment (see Figure 4.6). The class is more experienced with game-based learning tools than the first group. This could be the cause for the lower level of engagement. When considering the Computer Emotion Scale (Kay and Loverock, 2008) the predominantly feelings are linked to happiness (M=2.556, SD=0.97) (see Figure 4.7). The results of the Situational Motivation Scale (Guay, Vallerand, and Blanchard, 2000) show a high level of external regulated (M=3.98, SD=2.17) motivation (see Figure 4.8). Similar to the first experiment this could be caused by the experiment's educational context.

All groups were able to pass each concept-learning task in the game (Steinmaurer, Pirker, and Gütl, 2019b). They could also finish the missions quite quickly since the first group has finished after 10 minutes. All groups could manage to solve the given practical tasks. The solution approach of the third task was different by the used concepts: Five groups used a *for loop* to count from 20 to 30. Three groups wrote the whole output into the print command. After the practical tasks the groups were

	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
The game's structure is coherent	1	6	4	1	0
The game's control is easy to use	2	2	4	3	1
The usage of the keyboard was easy	2	4	1	3	2
The language is understandable	2	2	1	5	2
The levels were easy to master	1	3	5	1	2
I learned something while playing the game	1	5	4	0	2
sCool encouraged me to learn more about programming	1	1	1	4	5
The work sheet was easy to solve	0	1	4	3	4

Table 4.5: This table shows the results of the game related questions of Experiment 2.

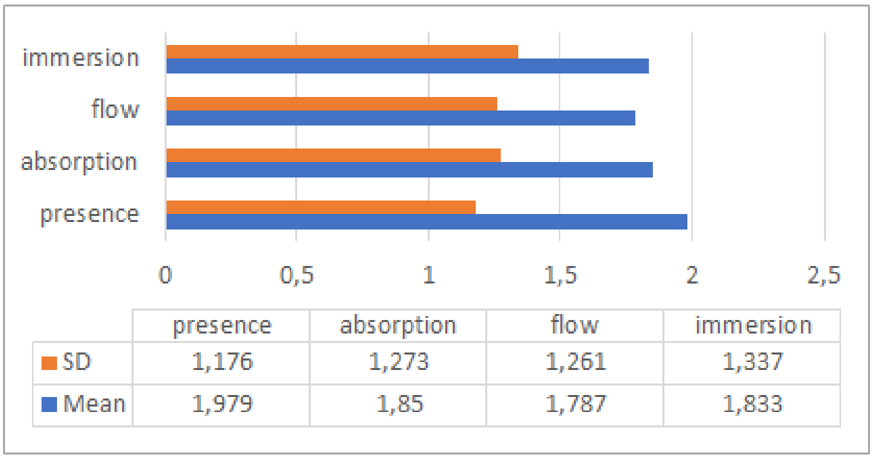
handed out the worksheet with the programming task. Just one group had a solution approach that contained a (for) loop. All others groups could not find an appropriate solution.

Table 4.5 shows the results of the game related questions. When playing the game (especially the robot missions) the groups had issues with the control and the keyboard: *"In my opinion the control is not easy.", "I would improve the control since it is stuck or not working well."* Most questions during the lesson occurred in relation to the control or the keyboard. The group also had problems with the game's language and asked for a german game version: *"Please add a german synchronisation", "german translation"*. A female student had a gender-specific response: "it would be cool to rename Rob or change the gender".

4.4 Findings and Limitations

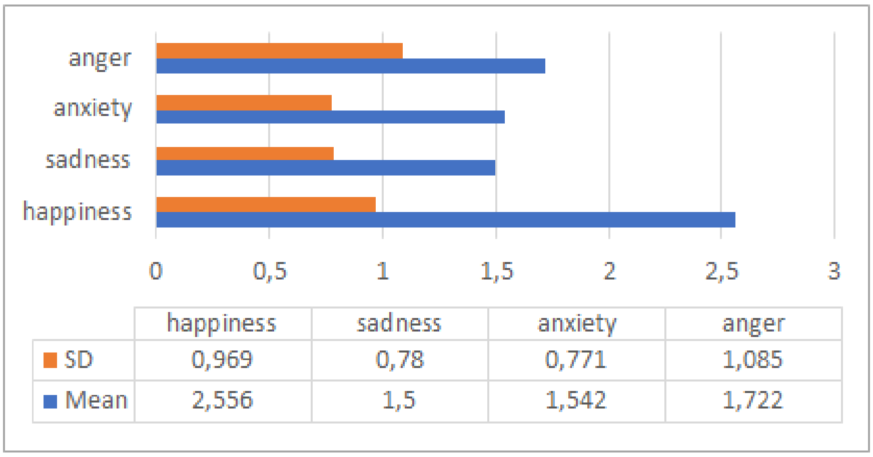
The findings of both pre-evaluations will be the foundation for the further development of sCool and its pedagogical concept. During both experiments the users and their interaction with the system was observed and all results are gathered on base of:

- game-related questions
- open-ended questions



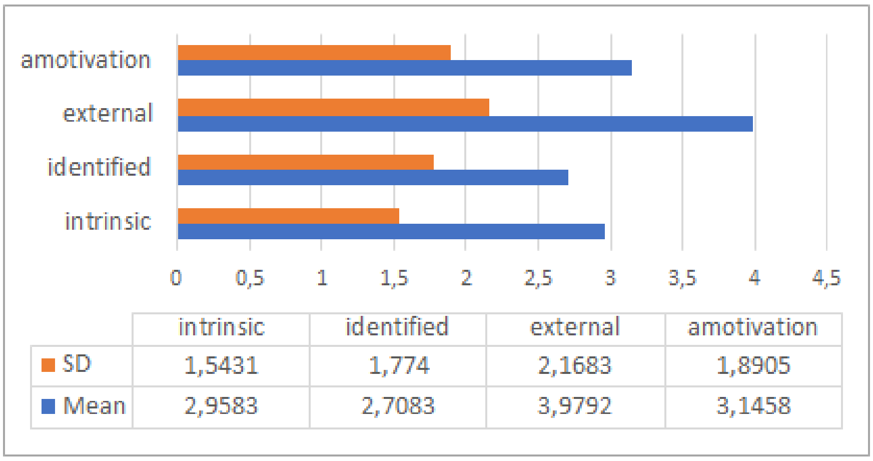
	presence	absorption	flow	immersion
🟧 SD	1,176	1,273	1,261	1,337
🟦 Mean	1,979	1,85	1,787	1,833

Figure 4.6: Results of Game Engagement Questionnaire at Experiment 2.



	happiness	sadness	anxiety	anger
🟧 SD	0,969	0,78	0,771	1,085
🟦 Mean	2,556	1,5	1,542	1,722

Figure 4.7: Results of Computer Emotion Scale at Experiment 2.



	intrinsic	identified	external	amotivation
🟧 SD	1,5431	1,774	2,1683	1,8905
🟦 Mean	2,9583	2,7083	3,9792	3,1458

Figure 4.8: Results of Situational Motivation Scale at Experiment 2.

- feedback while playing
- discussion round

When comparing both experiment groups the overall game performance of the Academic Secondary School was better since all school students were able to complete both concept-learning and practical tasks. The results of the worksheets were similar as no group was able to do the calculation without help. The results in game engagement, emotions, and motivation were also divergent. The flow factor of the group in experiment 1 was around M=3.53 (SD=1.50) whereas the second group had a mean flow value of M=1.78 (SD=1.26). Both groups associated mainly positive feelings (happiness) with sCool. The level of anger in the second group was the second highest feeling (M=1.722, SD=1.09). While working with the school students it could be observed that they often got stuck or had questions regarding control and keyboard which led to a higher level of frustration, anger and irritability. The dominating motivation at the first group was intrinsic (M=5.72, SD=1.27) followed by identified regulation (M=5.26, 1.85). In the second group the level of intrinsic motivation (M=2.958, SD=1.54) was rather small in comparison of amotivation (M=3.14, SD=1.89) and external regulation (M=3.97, SD=2.17).

Table 4.6 illustrates the results of the feedback from the open-ended questions. It contains the result of both experiments. The feedback is categorized in the following categories:

- **Controls**: This category contains feedback regarding the control in the game (or certain game types) or the keyboard. This category compromises especially feedback concerning issues with the virtual keyboard.
- **Design**: One answer was given regarding the design which requested "better graphic".
- **Game Play**: The feedback contains general suggestions for more (and harder) levels but also specific propositions like additional enemies.
- **Instructions**: All response to this category is related to the robot missions. The students had problems to understand the tasks and the user interface.
- **Language**: All feedback related to to language asked for a german translation of the game, since it would make the game more understandable.
- **User Interface**: The structure of the game (menu and navigation) was not intuitive enough.
- **Questionnaire**: A single person asked for a shorter questionnaire.
- **Gender**: A girl requested to change the name of the robot or its gender.

The overall evaluation shows that the acceptance is high since most of the students stated that they were encouraged to learn more about programming and learned something new. The evaluation also shows that there is room for improvement regarding usability since many issues occurred regarding the controls and instructions (RQ1). The school students have predominantly positive feelings connected with sCool. The level of negative emotions was lower than the positive ones in both experiments. When considering the situational motivation the findings in both experiments diverge. As a result of the school (respectively the workshop) setting the situational motivation seem to be regulated external since the students have to attend. For that reason the Situational Motivation Scale will not be used in future experiments conducted in schools (RQ2). No group was able to apply the learned concepts to a different problem class. In terms of the experiment it was not able to solve the calculation task on the worksheet. This can be traced back to the fact that the task was to hard for the students and that it was not possible to learn certain concepts within 50 minutes. For future workshops it would be useful to have more room for learning and practicing (RQ3).

Du, Wimmer, and Rada (2018) and Wangenheim et al. (2017) conducted related experiments and came to similar results. Coding in class with a game-based approach increases the motivation for programming and computer science. To receive an encouraging and sustainable experience for students it is necessary to present the concepts in an understandable way and provide a system (in terms of the user interface) that is intuitive. To achieve a high motivation and a long lasting understanding in programming it is necessary to spend more time in coding and learning concepts.

4.4.1 Game Mechanics and User Interface

The evaluation showed that many problems and questions occurred related to the game mechanics and the user interface. The school students seemed a bit lost when they started playing the game. This was noticeable since there were many questions regarding the first steps in the game. They had less questions about the explorative game type. The students often failed at the final questions at the first attempt because they skipped the textual explanation of the concepts (Steinmaurer, Pirker, and Gütl, 2019b).

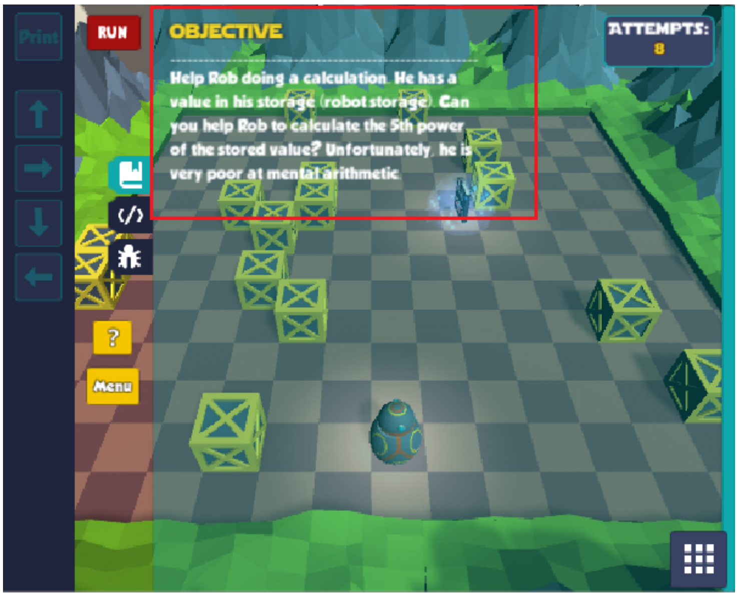
The most issues occurred during the programming tasks in the robot missions. The *objective screen* shows the task in the corresponding mission (see Figure 4.9). The students answered that they were not sure what to do exactly. They had problems to understand what they should do in particular. The students also stated that they did not know how to do the coding and what to type. In terms of the user interface the drag and drop mechanism was not intuitive and they missed feedback respectively support from the system.

4.4.2 Pedagogical Concept

The pedagogical concept was created together with a secondary school teacher and included the worksheets and the certain learning content in the game. The students answered that the introduction sheet was helpful for a first orientation. Many students were able to pass all missions and apply the learned concepts but they could not

#	Feedback	Game Mode	Category
1	I really like the game and will continue playing it at home.	sCool	-
2	better keyboard	Robot Missions	Controls
3	In my opinion the control is not easy.	Robot Missions	Controls
4	Make the control easier.	Robot Mission	Controls
5	I would improve the control since it is stuck or not working will.	sCool	Controls
6	Make the control at the robot missions easier.	Robot Mission	Controls
7	better graphic	sCool	Design
8	more shooting	Exploration	Game Play
9	other enemies	Exploration	Game Play
10	That not that many bears shoot you and you can hide.	Exploration	Game Play
11	make the game more interesting.	sCool	Game Play
12	more levels	sCool	Game Play
13	the levels should be longer and harder	sCool	Game Play
14	It would be cool to rename Rob or change the gender.	sCool	Gender
15	shorten the theory and make it callable while programming (it is hard to recognize everything)	Robot Mission	Instructions
16	Programming in the robot missions is to complicated	Robot Mission	Instructions
17	Make the robot missions easier and not so complicated	Robot Mission	Instructions
18	Open the programming again to make it more noticeable	Robot Mission	Instructions
19	The robot missions were to complicated for me.	Robot Mission	Instructions
20	please add a German synchronisation	sCool	Language
21	German translation	sCool	Language
22	German translation	sCool	Language
23	Select between German and English	sCool	Language
24	German translation	sCool	Language
25	make the questionnaire shorter	-	Questionnaire
26	simpler structure of game and app	sCool	User Interface

Table 4.6: Results of open-ended questions of both experiments.



OBJECTIVE

Help Rob doing a calculation. He has a value in his storage (robot storage). Can you help Rob to calculate the 5th power of the stored value? Unfortunately, he is very poor at mental arithmetic.

Figure 4.9: Robot Missions with the tasks objective.

transfer the concepts to a similar problem on another medium (paper instead of smartphone). Some students respond that they could not recognize the concepts and asked for a recall: *"make it callable while programming (its hard to recognize everything)"*.

4.5 Summary

The pre-evaluation of sCool covers two different school types. The first experiment was conducted at a New Secondary School and the second at an Academic Secondary School. The performance and the results of the questionnaires were different in both school types. Group 1 was not familiar with game-based approaches in education whereas group 2 was experienced with different tools. For this purpose the students of group 2 were less engaged while playing the game but achieved the better results in terms of the tasks.

Both groups provided constructive feedback for possible improvements in both the game mechanics and the pedagogical concept. Since many questions occurred while programming in the robot missions improvements of this game type are recommended. The school students answered that they had questions regarding the user interface and their objectives.

Hardly any group was able to solve the worksheet with the practical task. Therefore, the pedagogical concept has to be altered accordingly.

5 Requirements and Concepts

Based on both pre-evaluations new requirements and a revised pedagogical concept was created. The new concept should make it easier and more intuitive to introduce concepts and make them understandable for school students. Since a major part of the actual programming takes place in the robot missions it was decided to improve this particular game mode. The results of the pre-evaluation showed that many students did not know what they had to do and they also had questions regarding the user interface as well. The current pedagogical scope of the game covers just a few computational skills and so new concepts should be introduced in addition.

5.1 Requirements

In this section the functional and non-functional requirements regarding the game will be mentioned. The requirements can be grouped into functional and non-functional requirements. The functional requirements provide information about the functionality of a system and about the behaviour in certain situations. The non-functional requirements are about constraints of the system. (Sommerville, 2010)The requirements mainly cover the robot missions and the web application as well.

5.1.1 Functional Requirements

In the following list all functional requirements are stated that have to be considered for the further development of the pedagogical concept. The list includes requirements for both robot missions and the web application.

- **Tutorial**: Since many of the given feedback was concerning the game's instructions and the user interface, a tutorial should help players to get an orientation in the game. It should be triggered by events and occur only the first time when a certain behaviour is happening. The tutorial should fit in the already existing user interface and be intuitive as well.
- **Simple course creation**: The current web application allows educators to create courses in an easy way. Nevertheless, it is necessary to make slightly changes in order to extend the platform for new requirements. The tutorial system should not be part of the web application since it is part of the game itself.

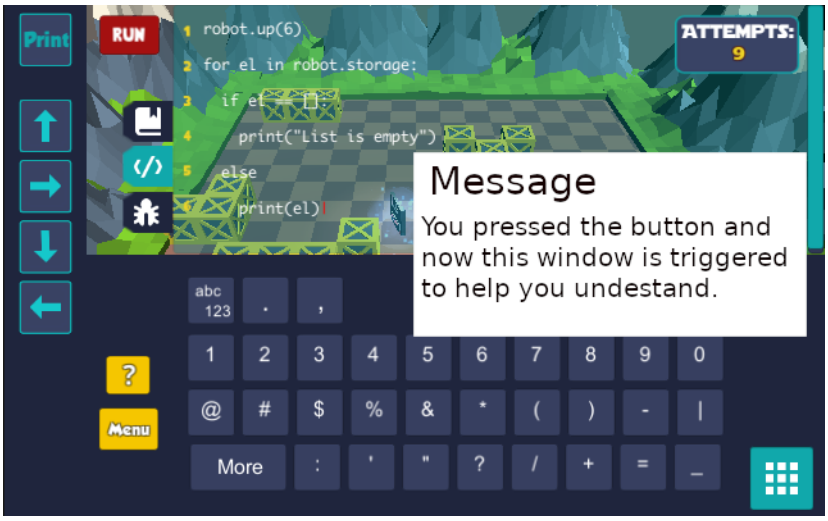
- **Lightweight Application Programming Interface (API)**: The extension of new functionality in the game (for example new field types) makes it necessary to extend the existing JSON API. To satisfy the non-functional requirements and to keep the usability and response time on an appropriate level it is vital to provide a lightweight API.
- **Valid Python commands**: In terms of didactics the commands in the robot missions should be at least valid Python commands. The commands should be oriented towards common Python coding standards. The Python Enhancement Proposals (PEP8)[1] covers the *Style Guide for Python Code*: *"Function names should be lowercase, with words separated by underscores as necessary to improve readability. Variable names follow the same convention as function names."*
- **Coordinates**: To support the usage of arrays, each field on the playfield should be addressable. Cartesian coordinates are recommended for an understandable navigation on the field.

5.1.2 Non-Functional Requirements

There are a few aspects to consider to keep the code base and the system architecture clean and lightweight. In terms of collaboration a readable and modular code base is important for future development. The redesign of the pedagogical concept should improve usability and make it more intuitive. The following list contains the most important non-functional requirements and extends the requirements of A. Kojic (2017).

- **Usability**: One of the most important non-functional requirements is the usability since many of the given feedback was about issues with it. A clearly designed user interface supports the players interacting with the system. Usability is a key requirement for the sCool platform because an understandable user interface enhance the players satisfaction with the system.
- **Modularity**: The already existing code base should stay modular and clean. A modular structure should minimise the maintenance afford and make collaborative working easier. The components should be created in an modular way to be opened for extension.
- **Security**: Since the whole sCool architecture contains data of students it is important to keep the system secure. This concerns on the one hand the web platform and on the other hand the game itself. The sCool game should also be protected from attempts to cheat within the game.
- **Level of Difficulty**: In order to create a motivating and engaging game experience an even balance between skill and challenge should be given. The educators can control this aspect by setting a certain level of difficulty for each task. The robot missions should be generated according to this value.

[1] https://www.python.org/dev/peps/pep-0008/



```
robot.up(6)
for el in robot.storage:
  if el == []:
    print("List is empty")
  else
    print(el)
```

Message

You pressed the button and now this window is triggered to help you undestand.

Figure 5.1: Sketch of sCool's onboarding system.

5.2 Pedagogical Concepts

The revision of the pedagogical concept covers three major changes. An onboarding system was introduced which should help players to understand the robot mission's user interface. The introduction of different field types and an interactive environment (playground, robot, etc.) brings a number of additional possibilities. In this way it is possible to work with concepts like arrays, loops and conditions in a meaningful way. The third concept is the ability to access different attributes and methods of the robot. This can help students to get an understanding for object-orientated aspects.

5.2.1 Onboarding

Some of the feedback was concerning the user interface and missing instructions. To help students reach a better understanding for the robot missions an onboarding should be included. Depending on the needs of the user the instructions should be displayed. The messages should only appear the first time a certain behaviour occurs. This means that all messages should be event triggered. Figure 5.1 shows a sketch of a message that is displayed when a certain event is triggered.

The onboarding should cover an explanation of the user interface and the basic interaction with the system. For this purpose all messages are part of the sCool environment and will not be adaptable in the web application.

5.2.2 Field Types

To reach a better understanding of data types so called *field types* are introduced. Currently there is one (blank) type of field on the playboard. The following elements can be placed on the fields:

- Robot
- Disk
- Box

It is not possible to interact with the environment and ask for certain types of fields. The introduction of field types has the advantage to use various concepts in an intuitive way. Different types of fields will also change the way how players will interact with the system. There are two practical ways to address a certain field:

1. **Cartesian coordinates**: With the usage of Cartesian coordinates all fields can be asked before runtime with the playing field's x- and y-coordinates (e.g. fieldType(x,y)). The advantage is to consider the exact path and solution approach beforehand. The playing field is seen as a two dimensional array and so the students learn how to access elements in an array, learn about different data types and array-related algorithms (searching a field, looping over a sequence, etc.). The disadvantage is, that coordinates make the interaction static since the player can not make decisions during runtime (for example avoid an obstacle).
2. **Adjacent fields**: The usage of adjacent fields enable an enormous flexibility during runtime. The player can ask for the field type of an adjacent field (e.g. fieldType(left)). The advantage is to enable dynamic decisions during runtime since the players can interact with the environment (avoid boxes). The disadvantage is that it is more difficult to integrate this approach into a pedagogical concept.

After considering both approaches the first one (Cartesian coordinates) was chosen: Before the robot makes the moves the whole code is interpreted first. In this way it is not possible to make dynamic decisions during coding.

Table 5.1 lists six (possible) additional field types and shows concepts that can be integrated in the courses.

5.2.3 Robot Interaction

All commands that are executed by the robot are part of the *robot object*. The available commands are: Left(), Right(), Up(), and Down(). To cover other concepts as well a few additional commands were added:

Field	Description	Concept
Hidden	This field is invisible for the player and punish for passing it.	Array, Conditions
Teleport	When the robot is entering this field he is placed to the opposite teleport field.	Array
Bonus	When passing this field the player receives a reward (coins).	Array, Loops
Slow	The number of steps are decreased when passing the field.	Array, Loops
Fake	There are fake disks on the playing field that punish the player.	Array, Conditions
Invert	This field inverts the moving commands.	Conditions, Sequencing

Table 5.1: This table shows possible field types.

- **Robot.storage**: The players can access the robot's storage which acts as a memory for external values. In this way educators can store different information in the memory that can be used for a large number of tasks. The values can be both elementary data types (integers, characters, etc.) or complex types (lists). This also gives the possibility to give meaningful tasks in the robot missions since the educators can decide how the input and the output should be.
- **Directions**: When analysing the pre-evaluation it showed that some players tried to pass the number of steps as parameters in the moving commands (e.g. *Robot.Left(3)* for three steps to the left). This parameter can also support beginners to use an easier way of looping in contrast to a for/while loop.
- **Randomization**: Another important concept in computer science is the generation of random numbers. This can be useful for different tasks on the playing field. The function *Robot.rand_number(int min, int max)* provides the possibility that the robot can compute random numbers.

Figure 5.2 shows an UML diagram with the most important attributes and methods of the robot object.

5.3 Summary

This chapter gives an overview of the requirements and concepts that are the results of the pre-evaluation. Some of the mentioned requirements are an extension of already existing requirements (A. Kojic, 2017; M. Kojic, 2017) and some are new. Significant functional requirements are for example the tutorial (or onboarding) system and a

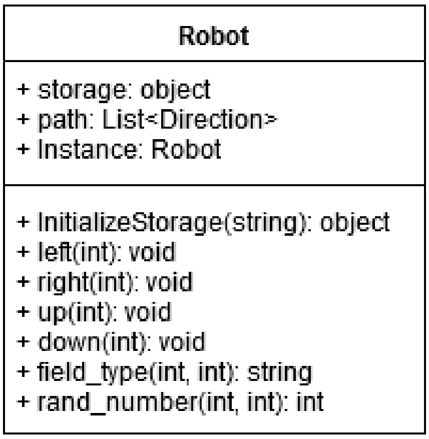


Figure 5.2: UML diagram of the robot object.

simple course creation. The onboarding system should be an flexible, event triggered system that can be used for all needed components. In order to have a flexible and fast communication between the web application and the sCool video game a lightweight API is necessary. The transmitted object should have less overhead and transfer just relevant data.

Usability is a central non-functional requirement because it contributes to the acceptance of the users. It is important to provide an easy user interface and give clear instructions that are understandable for the players. The players should also enjoy the game and not get unchallenged while working on the tasks. A balanced level of difficulty should challenge the school students in an appropriate way. sCool is a flexible and modular tool and open for further extension. For this reason a clean code base and a modular approach is central to extend additional concepts and game types.

The further development of the pedagogical concept covers three different parts: onboarding, field types and object interaction. This should help to improve the current concept, extend the abilities of the system and help the players to get a better understanding for computational skills and coding in general. The tutorial should help beginners with the first steps in the game and guide them. It should explain the user interface of the robot missions and the basic interaction with the system. The coordinates and object interaction brings new capabilities into the game. It helps to create meaningful tasks with conditions, arrays, loops, variables, and objects.

6 Development

This chapter covers the further development and implementation of the revised pedagogical concept. This concept is the result of two pre-evaluations that were conducted at two secondary schools. During the course creation a few issues could be found in both the web platform and the sCool game. This issues had to do with usability aspects, where the participants had particular questions or affected some functionality that did not work like expected.

This chapter is divided into two sections: *Robot Missions* and *Web Application and API*. The first section deals with the revised pedagogical concept and solutions for minor issues in the scope of usability. Both aspects are described in chapter 5. The second part covers all changes related to the web application and the API. This chapter shows essential code listings and figures that explain the further development of sCool.

All used technologies are already predetermined by the given sCool environment. These technologies are described in chapter 3, A. Kojic (2017) and M. Kojic (2017). The video game is based on Unity 3D and the C# programming language. On the server-side the web application was initially created with the ASP.NET MVC web framework and the C# programming language. As central storing technique a Microsoft SQL database is used. Video game and web platform are connected via a REST API. The exchange of the data occurs via JSON. The whole web platform is hosted on the Microsoft Azure cloud computing platform.

To avoid ambiguity and get a clear separation between both game versions, the sCool version before re-design and further development will be called *version 1*. The game version that will be introduced in this chapter is referenced as *version 2*.

6.1 Robot Mission

The extension and improvement of the robot mission is a central aspect of the pedagogical concept since the actual programming takes place in this part of sCool. This section introduces field types and shows implementation details and its pedagogical advantages. The introduction of an onboarding system was the result of several user-related questions concerning the user interface. The text-based onboarding system guides first-time players through the programming workflow. The players are using a virtual keyboard for programming that is part of the sCool user interface. Some of the experiments' response was related to usability aspects of the keyboard. The two-dimensional playground is a key element for the player because the interaction with the robot and the environment happens here. Version 1 has some minor issues, that can confuse the player. The last introduced concept is the robot's storage where educators can pre-define some external data that can be used in the game.

6.1.1 Field Types

In the robot missions of version 1 was no distinction between field types (see Chapter 3). A field can be occupied by the player, a box or the disk. The fields have no specific type and there are just two possibilities of interacting with the objects:

1. **Robot collides with box**: When the robot hits the box the *OnCollisionEnter* event is triggered. In this case the robot stops and the players get a visual feedback that a collision happened (see Listing 6.1). In this case the number of attempts is decremented by one.

```
void OnCollisionEnter(Collision collision)
{
  if (collision.gameObject.name == ”Robot”)
  {
    SoundEffects.Instance.ObstacleSound();
    Robot.Instance.Stop();
    ShakeScreen.shakeTime = 1f;
    collision.collider.enabled = false;
  }
}
```

Listing 6.1: Robot collides with a box on the playing field.

2. **Robot collides with disk**: When the disk is reached the disk's *OnTriggerEnter* event is called. In this case the state of the game is set to *finished* and the given output is compared with the reference output provided from the server (see Listing 6.2).

```
void OnTriggerEnter(Collider other)
{
  if(other.name == ”Robot”)
    PlaygroundManager.Instance.FinishReached(true);
}
```

Listing 6.2: Robot collides with the disk on the playing field.

In order to implement new pedagogical concepts an extension of the field types was necessary. Figure 6.1 shows the hierarchy of the field types. Each different field is

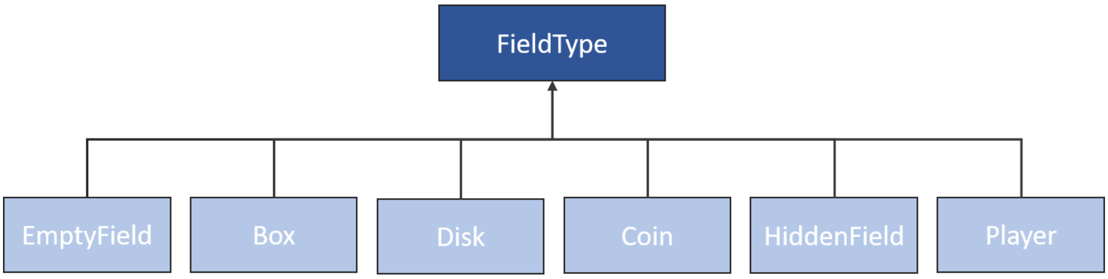


Figure 6.1: Hierarchy of the field types.

derived from the *FieldType*. The playground is represented by a two-dimensional array of the type *FieldType*. When the playground is generated the fields are populated (see Listing 6.3). The populating algorithm randomly selects a field on the playground and checks if the field is free or is already occupied. In version 1 the playground was created with one disk, one player and a total of 15 boxes. Now it is possible to let the educator decide how many fields of a certain type should appear on the map. The method *PopulateFields(FieldType.FieldTypes, GameObject, int)* is called for three types of fields: boxes, coins, and hidden fields.

```
public void PopulateFields(FieldType.FieldTypes type,
  GameObject go, int number)
{
  for (int i = 0; i < number; i++)
  {
    int x, y;
    do
    {
      x = Random.Range(0, 14);
      y = Random.Range(0, 14);
    }while(fieldType[x, y] != FieldType.FieldTypes.EmptyField);
    fieldType[x, y] = type;

    GameObject newGo = Instantiate(go);
    Vector3 pos = newGo.transform.position;
    pos.x = (x - 7) * 2;
    pos.z = (7 - y) * 2;

    newGo.transform.position = pos;
    newGo.SetActive(true);
  }
}
```

Listing 6.3: Populating algorithm for the playground.

EmptyField

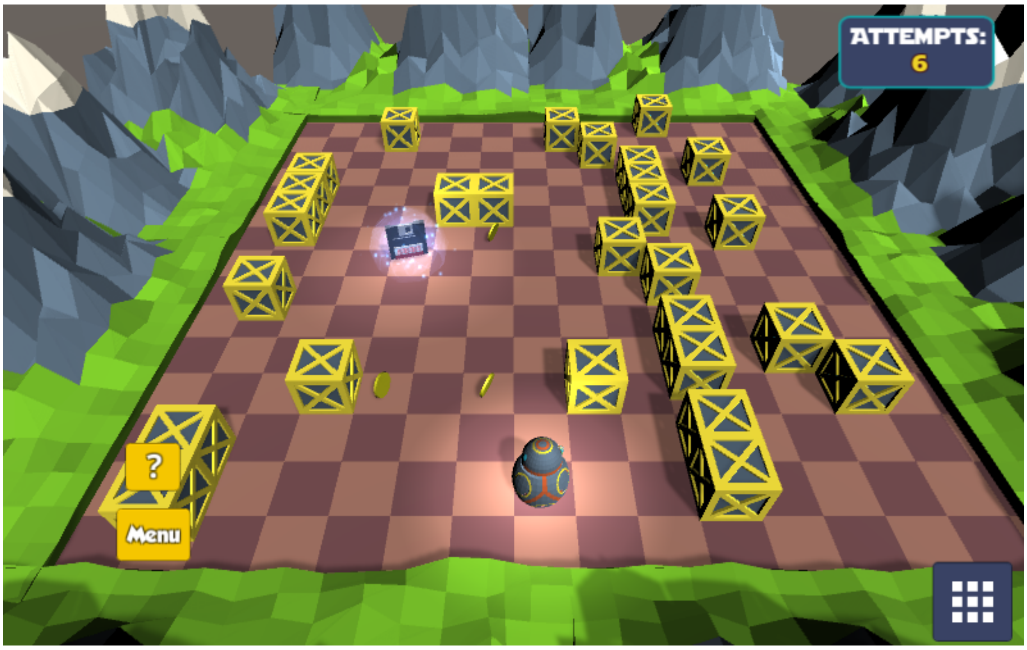


Figure 6.2: Playground with all possible field types enabled.

Per default every field on the playground is an empty field. If it is assigned differently the type is changed. All fields that remain empty after the field population are walkable fields without any specific behaviour.

Box
A box is an obstacle that can be placed on any non-occupied field. The number of boxes can be regulated by the educators over the web application. When a box is reached the player gets a visual feedback and the number of attempts is decreased by one.

Disk
Each mission has exactly one disk that is placed according to the level of difficulty. When the disk is reached the player's output is compared with the reference output. If both values are equal the mission is passed.

Coin
This field type extends the rewarding system. The number of coins can be configured by the web application. When a coin is collected the number of total coins is increased by 30 and the coin disappears from the playground (see Figure 6.2).

HiddenField
The hidden field is not visible for the player on the field. This field punish the player for passing it, by dividing the number of total coins in half. The player can find the field only by using the robot's *field_type(int,int)* function.

Player
The player object is the only movable object on the playground. It can be controlled by

the moving commands. All other objects do have colliders that are activated when the robot object collides with them.

To receive information about a certain field on the playground the player can use the *Robot.field_type(x,y)* function. This function is part of the robot object and is accessible in the Python environment. The method returns the type of a certain field in relation to its value at the playground array (see Listing 6.4).

```
public string field_type(int x, int y)
{
  return PlaygroundManager.Instance.fieldType[x,y].ToString();
}
```

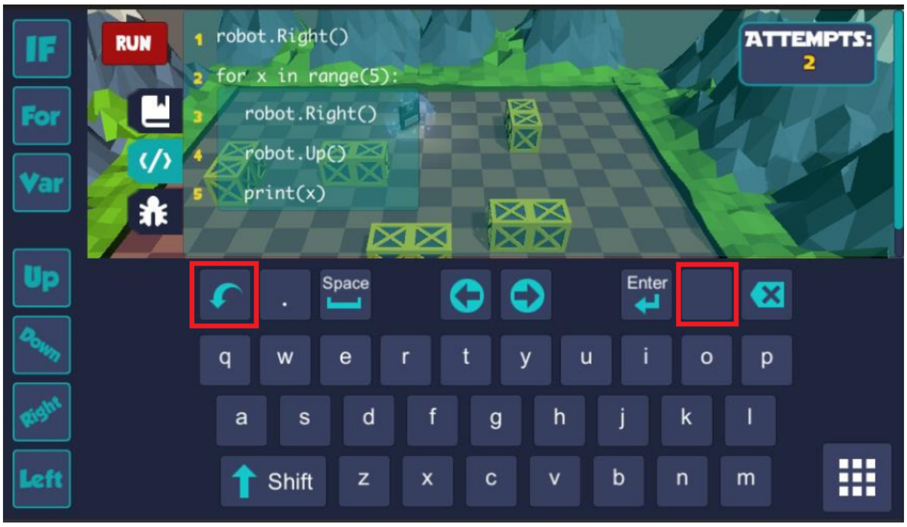
Listing 6.4: Receiving the type of a certain field with the coordinates.

6.1.2 Keyboard

The virtual keyboard is a key element for tasks in the robot missions. It is used to write the actual code in the coding editor. The keyboard is based on a regular *qwerty* layout and has the following buttons on the default view:

- lower case letters (a-z)
- special characters
- dot
- space
- horizontal cursor
- enter
- semicolon
- delete
- shift

This makes it possible to write text in a simple way into the editor. Figure 6.3 shows the keyboard of the game screen in version 1. Two issues occurred with the screen during pre-evaluation: i) it was confusing how to switch to special characters and ii) the ambiguous functionality of the blank button on the second last column in the first row. It was decided to make a more meaningful label for i) to provide information about the button. Since the button should change between a view with letters and a view with numbers and special characters the label of the button should be renamed regarding this functionality (see Figure 6.4). The blank button in ii) represents a semicolon but the label on the button is not visible. Figure 6.4 shows the enabled labeling for the button.



```
robot.Right()
for x in range(5):
    robot.Right()
    robot.Up()
    print(x)
```

Figure 6.3: The keyboard in the robot missions in version 1 (after A. Kojic (2017)).

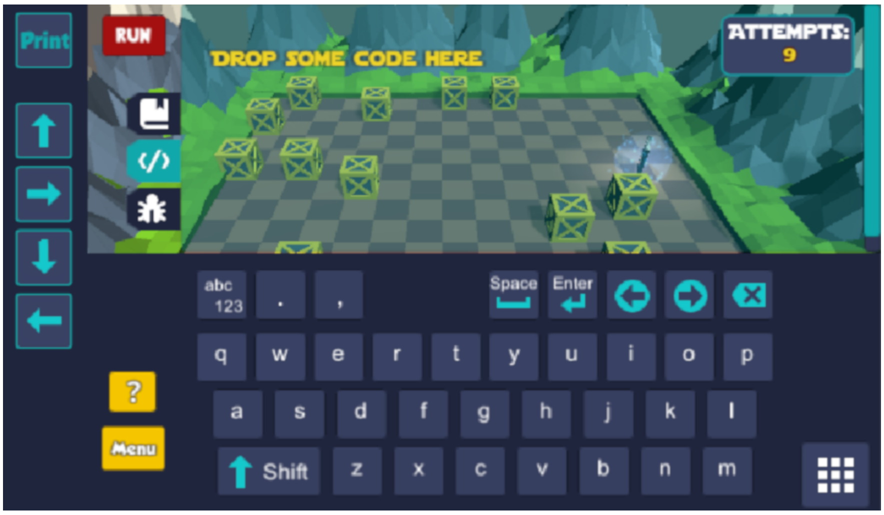


Figure 6.4: The keyboard with the changed labels in version 2.

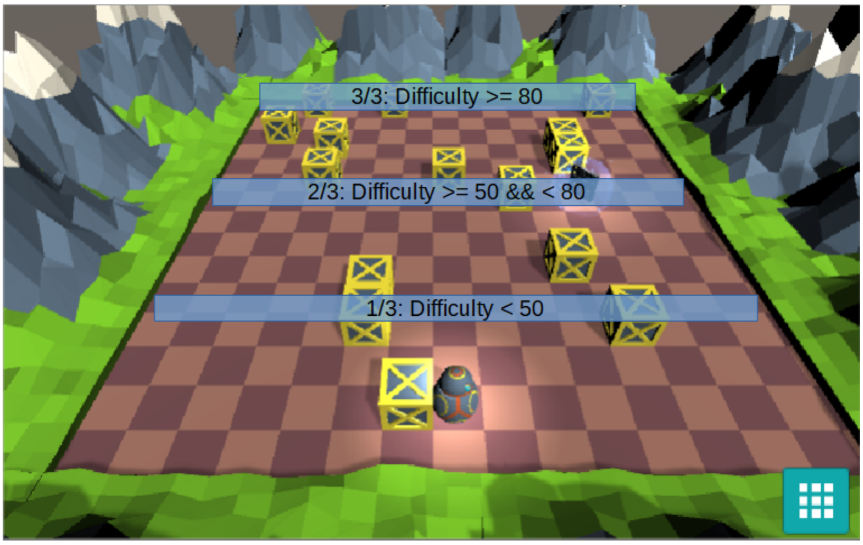


Figure 6.5: Calculation of disk's position related to the level of difficulty (adapted after A. Kojic (2017)).

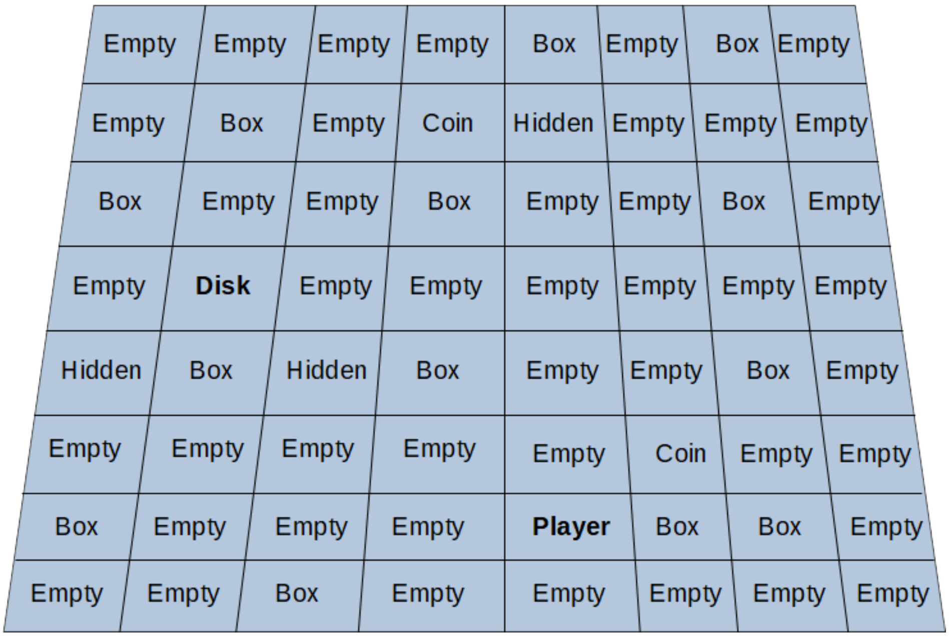
6.1.3 Playground

Besides the introduction of the above mentioned field types other changes have been made on the playground as well. During the pre-evaluation some students gave the feedback that the disk position remains simple, despite the higher level of difficulty. The x-coordinate of the disk is generated randomly between 1 and 15. Listing 6.5 shows the updated generation of the y-coordinate. Depending on the level of difficulty the position is set in the related third (see Figure 6.5).

```
int AdjustY()
{
  int y;
  if (PracticeManager.Instance.practiceTask.Difficulty < 50) {
    y = Random.Range(10, 14);
  } else if (PracticeManager.Instance.practiceTask.Difficulty
    >= 50 && PracticeManager.Instance.practiceTask.Difficulty
    < 80) {
    y = Random.Range(5, 10);
  } else {
    y = Random.Range(0, 5);
  }
  return y;
}
```

Listing 6.5: Calculating the disk's y-coordinate.

Related to the map generation another issue was reported by some students: A box was placed on the disk field which made the mission unsolvable. When inventing field



Empty	Empty	Empty	Empty	Box	Empty	Box	Empty
Empty	Box	Empty	Coin	Hidden	Empty	Empty	Empty
Box	Empty	Empty	Box	Empty	Empty	Box	Empty
Empty	**Disk**	Empty	Empty	Empty	Empty	Empty	Empty
Hidden	Box	Hidden	Box	Empty	Empty	Box	Empty
Empty	Empty	Empty	Empty	Empty	Coin	Empty	Empty
Box	Empty	Empty	Empty	**Player**	Box	Box	Empty
Empty	Empty	Box	Empty	Empty	Empty	Empty	Empty

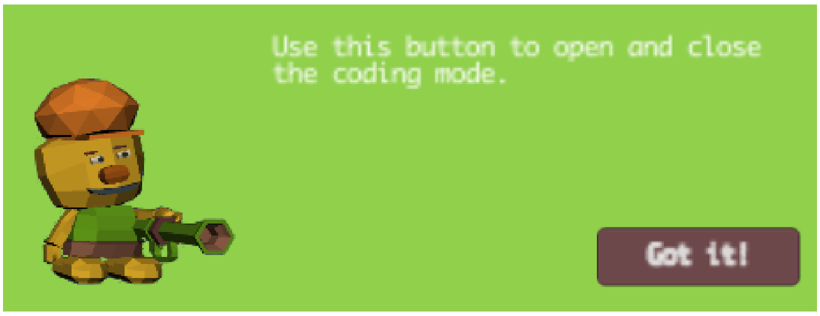
Figure 6.6: Visual representation of a sample field type array.

types the populating algorithm was revised in order to check if the desired position is free or occupied. If the position is free the coordinate in the two-dimensional array is set to the field. Figure 6.6 shows an example of a graphical representation of the field type array. This two-dimensional array must have at least a player and a disk field. If the educators do not declare a certain specification via the API all positions remain *EmptyField*s.

6.1.4 Onboarding

The onboarding system should provide further instructions and help players. It should be a guide for the first steps and an introduction to the user interface and the programming environment. The messages should be triggered by certain events and open for extension in a flexible way. Figure 6.7 shows the general tutorial screen that appears when a certain event is triggered.

The *TutorialManager* component handles the behaviour of the instruction messages and can be added to any game object (e.g. buttons or colliders). It stores all text in a dictionary of type *TutorialEntry*. A TutorialEntry consists of a text and an optional key to a next TutorialEntry object. In this way it is possible to create a linked list among related instructions. When a certain event is triggered the *TriggerEvent(string)* method is called with the dictionaries key (see Listing 6.6). The entry is added to a general list in the *GameController* where the index of all previous displayed messages is stored. When pressing the button *Got it!* the popup is closed and will not be showed again. All information regarding onboarding is stored locally and not transmitted to the server.



Use this button to open and close the coding mode.

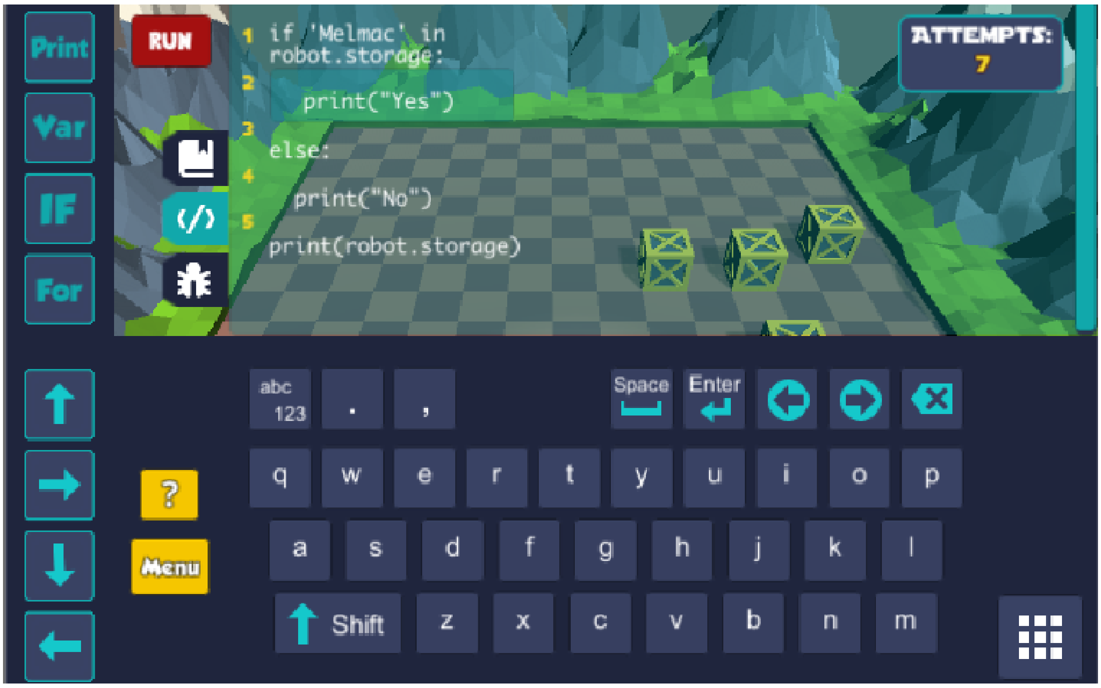
Figure 6.7: General view of the onboarding popup.

```
public void TriggerEvent(string eventName)
{
  if (!IsPractitcalTutorialShown(eventName))
  {
    animator.SetBool(''IsOpen'', true);
    getTutorialByKey(eventName);
    AddPracticalTutorialShown(eventName);
  }
}

public void CloseEvent()
{
  animator.SetBool(''IsOpen'', false);

  string followUp = getFollowingElement(currentTutorial);
  if (followUp != null)
  {
    this.TriggerEvent(followUp);
  }
}
```

Listing 6.6: Tutorial messaging system.

6.1.5 Robot Storage

The robot storage is a way to pass external data to the game. In this way educators can define some input that is available in the game. This input can be specified in the web application and is passed over the REST API to the game. It is stored as an attribute of type *object* in the robot instance. Since the value has the type *object*, it can have any data type. In this way it is possible to pass lists, strings, numbers, letters, etc. Listing 6.7 shows how the storage is initialized regarding the API's input. The storage should close an educational gap in order to support concepts that were not usable in a



```
if 'Melmac' in
robot.storage:
    print("Yes")
else:
    print("No")
print(robot.storage)
```

Figure 6.8: Screenshot of an example where the robot storage is used.

meaningful context in version 1 (for example conditions). The robot storage makes it possible to work with the following concepts:

- **Data Types**: The value (or values) in the storage can be both elementary or complex data types. Since it is an *object* list it can contain any data type. In this way the list can be treated as a regular list which makes it possible to ask for the data type of a single element or perform operations on a specific data type (e.g. concatenation of items, arithmetic operations on integer numbers).
- **Conditions**: In version 1 the usage of condition was problematic since there were less application areas. When using the storage, conditions can be applied to the values in the store (e.g. find a maximum value in a list, comparing a value in the storage with a calculated value).
- **Repetitions**: There are several domains where it make sense to repeat an action (for example moving the robot a given number of times). Using lists in combination with the robot storage extends the application area of loops since they can be iterated (e.g. checking if an element is in a list or summing up all numbers in a list).
- **Lists**: Lists are a common concept in computer science and Python as well. A predefined list makes it possible to use already existing lists and perform operations on them (e.g. indexing lists, slice lists, add/remove elements).

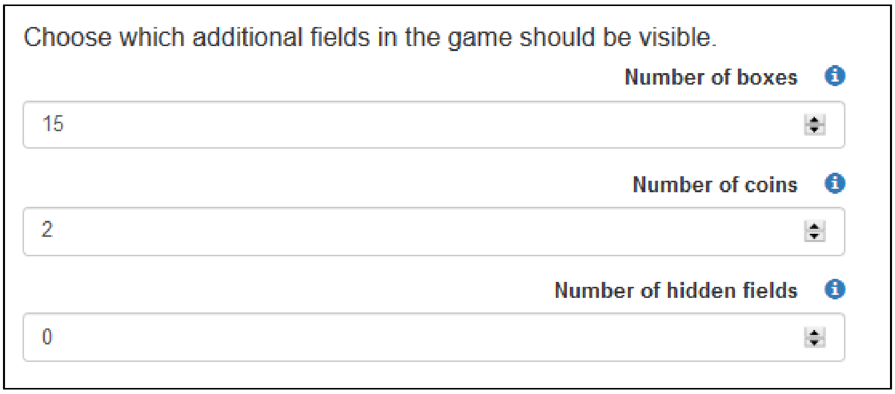
```
object InitializeStorage(string value)
{
  List<object> tempStorage = new List<object>();
  string[] strList = value.Split(',');
  foreach (var str in strList)
  {
    try
    {
      tempStorage.Add(Convert.ToInt32((str.Trim())));
    } catch (FormatException) {
       try
       {
         tempStorage.Add(Convert.ToDouble((str.Trim())));
       } catch (FormatException) {
         tempStorage.Add(str.Replace('\'', ' ')
           .Replace('\''', ' ').Trim());
       }
    }
  }

  if(strList.Length == 1)
  {
    return tempStorage[0];
  }
  return tempStorage;
}
```

Listing 6.7: Initialization of the robot's storage.

Figure 6.8 shows that the value in the storage can be accessed via the attribute (*robot.storage*). There is a distinction between integers, floating-point numbers, strings and lists. The elements are internally parsed into the related data type and in this way it is possible to perform all operations on the specific data type.

6.2 Web Application and API

The re-design of the pedagogical concept made it necessary to make some improvements on the web application and API since it provides the educators more capabilities. This section also describes issues that occurred while preparing and creating courses and presents solutions.



Choose which additional fields in the game should be visible.

Number of boxes: 15

Number of coins: 2

Number of hidden fields: 0

Figure 6.9: Screenshot of the web application's back-end with field configuration.

6.2.1 Field Types

The introduction of different field types brings new possibilities for educators to generate the playground. There are three field types that can be pre-defined in the web application: boxes, coins and hidden fields. Figure 6.9 shows the configuration of the field types. According to the number of occurrence the fields are generated randomly on the playing field.

To provide a meaningful and comprehensive analysis in the web application it is necessary to log the field types into the database. Each time the robot collides with a certain field (box, hidden field, coin, finish) the *OnCollisionEnter* method adds a *RobotCollision* object to the *RobotCollisions* list. This object stores the position, time and name of the obstacle (see Listing 6.8). Listing 6.9 shows an example of the logging after the robot collided with a coin object.

```
public class RobotCollision
{
  public string Time;
  public string Position;
  public string Obstacle;
}
```

Listing 6.8: Robot collision model.

```
Assets.WebServices.Models.Request.RobotCollision rc =
  new Assets.WebServices.Models.Request.RobotCollision();
rc.Position = transform.position.ToString();
rc.Obstacle = ''Coin'';
rc.Time = System.DateTime.UtcNow.ToString();
PracticeManager.Instance.practiceResponse.
  RobotCollisions.Add(rc);
```

Listing 6.9: Add collided object to the RobotCollisions list.

6.2.2 Consent Message

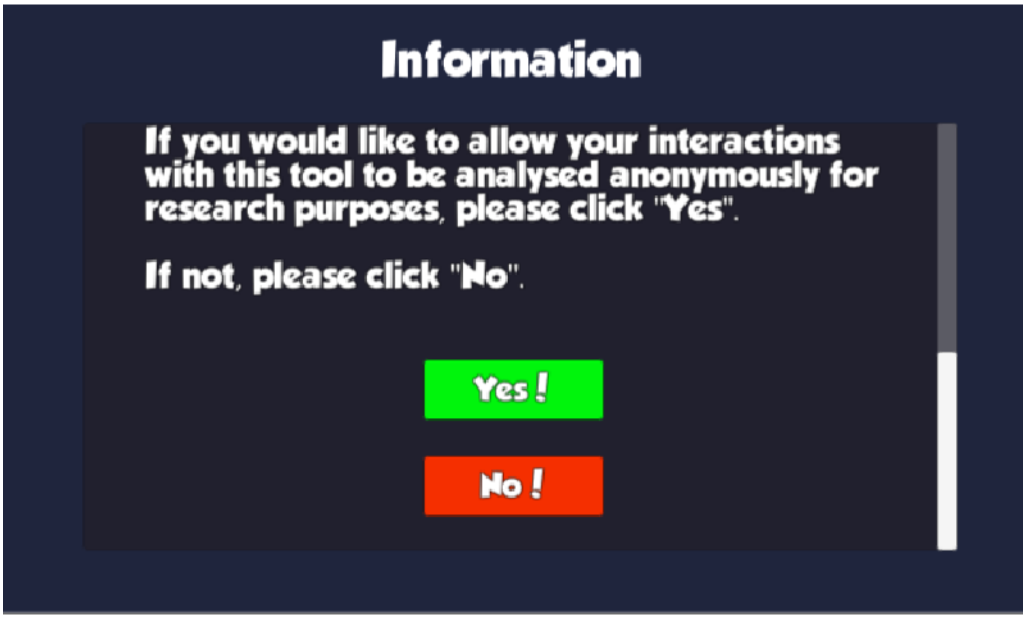
The consent message was a constraint for the experimental usage of sCool on the Royal Melbourne Institute of Technology (RMIT). Within the scope of the *Ethics Approval Application* it was necessary to inform the participants of the experiment about the collection of data and provide an opt-out. The participants can choose if they allow the collection of data or if they disagree. In order to let the players decide what happens to their data this option is also part of the sCool game version 2. In this way the players can play the game without being worried about any privacy concerns.

The consent message screen appears straight after the sCool registration popup (see Figure 6.10). According to the players response (yes or no) a flag in *Students.IsConsentGiven* is set. When the flag is set to *false* it is not allowed to analyse the data.

Listing 6.10 shows the server-side registration of the consent data for a certain user. The *Student* object is updated according to the provided boolean value. The data is transmitted via HTTP GET to the REST API.

```
[HttpGet]
[Route(''api/Data/GiveConsent'')]
public IHttpActionResult GiveConsent(int studentID,
  bool IsConsentGiven)
{
    Models.Student student = db.Students.Find(studentID);
    if (student == null)
        return NotFound();

    student.IsConsentGiven = IsConsentGiven;
    db.SaveChanges();

    return Ok('' '');
}
```

Listing 6.10: Server-sided processing of the consent data.



Information

If you would like to allow your interactions with this tool to be analysed anonymously for research purposes, please click "Yes".

If not, please click "No".

Figure 6.10: Consent message that appears after registration.

6.2.3 Skill Unlocking

During the pre-evaluation a major issue occurred since sCool did not unlock the next concepts (see Figure 6.11). The next concept should be unlocked when the player reached more than 66% in the concept-learning or the practical missions (see Listing 6.11). In the table *Skills* each skill has an internal order that appears in the web application and the game. By increasing the number by one it acts as a pointer to the following skill (Skills.Order + 1). If a certain skill is updated the order of the selected skill is overridden with zero which destroys the ordering mechanism. Figure 6.12 shows the column *Order* where order is overridden with zero at the updated rows. This issue could be resolved by keeping the order when updating the data record.

```
if ((type == ''theory'' && learning.Practice
  .GetMeasure(db, learning.UpdatedAt) > unlockLimit) ||
  (type == ''practice'' && learning.Theory
  .GetMeasure(db, learning.UpdatedAt) > unlockLimit))
{
  UnlockNextSkill(db, learning);
  al.Ability = B;
  al.ReInit();
}
```

Listing 6.11: Skill unlocking condition.

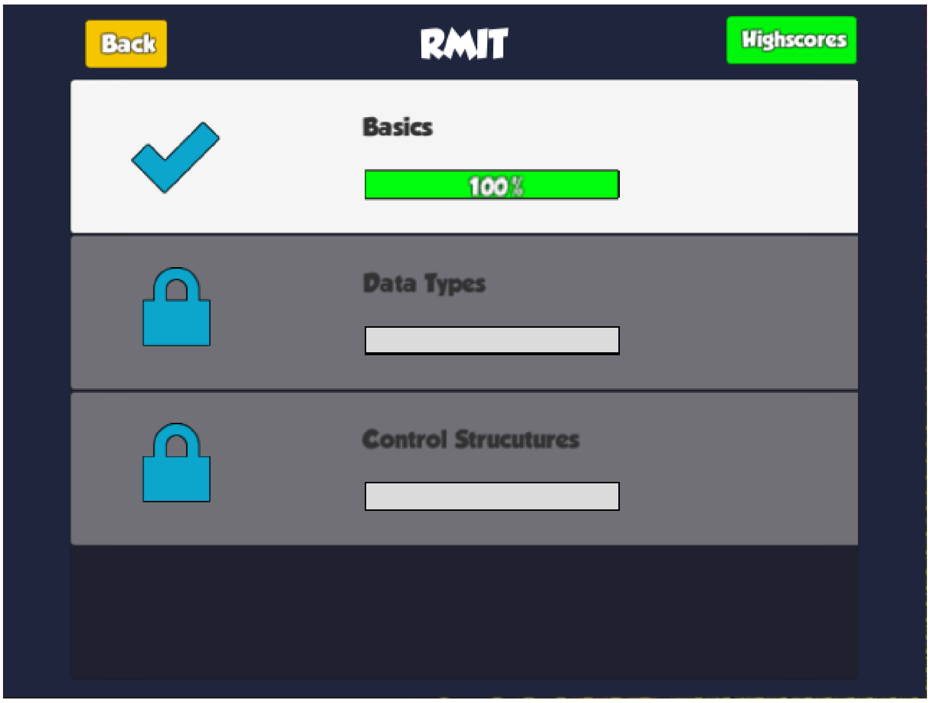
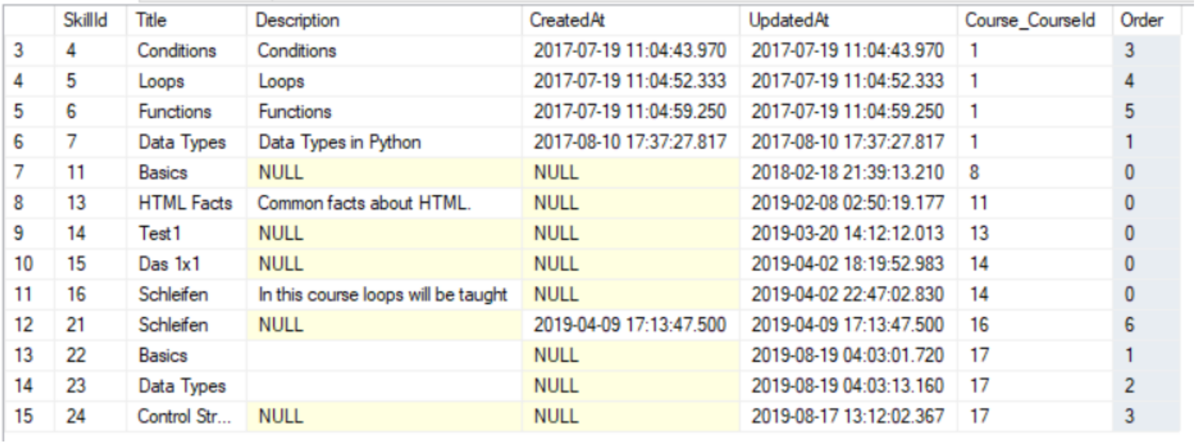


Figure 6.11: Issue with skill unlocking when the player reached 100%.



	SkillId	Title	Description	CreatedAt	UpdatedAt	Course_CourseId	Order
3	4	Conditions	Conditions	2017-07-19 11:04:43.970	2017-07-19 11:04:43.970	1	3
4	5	Loops	Loops	2017-07-19 11:04:52.333	2017-07-19 11:04:52.333	1	4
5	6	Functions	Functions	2017-07-19 11:04:59.250	2017-07-19 11:04:59.250	1	5
6	7	Data Types	Data Types in Python	2017-08-10 17:37:27.817	2017-08-10 17:37:27.817	1	1
7	11	Basics	NULL	NULL	2018-02-18 21:39:13.210	8	0
8	13	HTML Facts	Common facts about HTML.	NULL	2019-02-08 02:50:19.177	11	0
9	14	Test1	NULL	NULL	2019-03-20 14:12:12.013	13	0
10	15	Das 1x1	NULL	NULL	2019-04-02 18:19:52.983	14	0
11	16	Schleifen	In this course loops will be taught	NULL	2019-04-02 22:47:02.830	14	0
12	21	Schleifen	NULL	2019-04-09 17:13:47.500	2019-04-09 17:13:47.500	16	6
13	22	Basics		NULL	2019-08-19 04:03:01.720	17	1
14	23	Data Types		NULL	2019-08-19 04:03:13.160	17	2
15	24	Control Str...	NULL	NULL	2019-08-17 13:12:02.367	17	3

Figure 6.12: Table *Skills* with the corrupted *Order* column.

6.2.4 Robot Storage

The robot storage allows educators to pass various textual data into the game. When creating a practical mission the input field *Robot Storage* allows textual input. Figure 6.13 shows that it is possible to process either single or complex values (lists that are separated by commas). Figure 6.14 illustrates that the data is transmitted as string over the interface.

6.2.5 Web Application Multiuser Usage

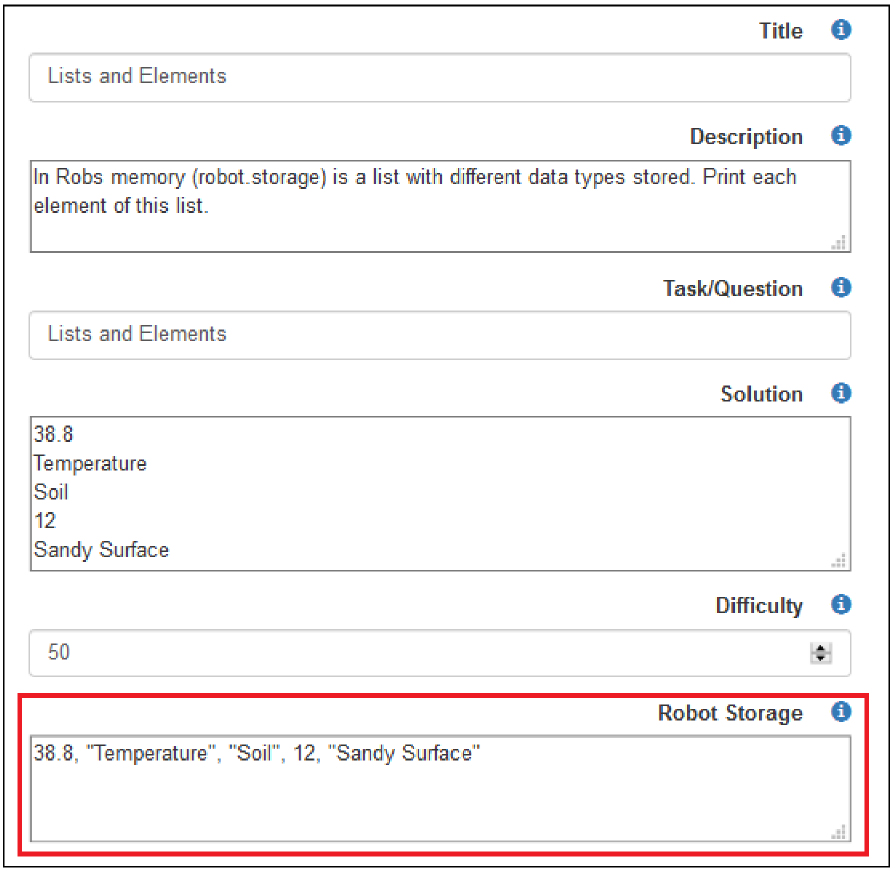
The web application is supposed to be used by several educators. This makes it necessary that every user can create own courses and manage the students. The creation of a new web user can be done at *Register* in the web application. The educator has to fill out a form with the email address and a password. After the form is completed the registration has finished and the user can work with the web application.

Until the user is not assigned as administrator the creation of new courses is possible but they will not appear in the students course list in sCool. The students should be enrolled to all courses of all administrators if the courses are not set as invisible.

This approach led to an issue since the course enrollment worked just for one educator with the admin flag. Listing 6.12 shows the resolving by assigning all new registered sCool users the courses of all administrators.

```
var adminUsers = db.Users.Where(u => u.IsAdmin == true);
var courses = adminUsers.SelectMany(a => a.Courses);

foreach (var c in courses)
{
  if (c.IsVisible)
  {
    Enrolled e = new Enrolled();
    e.Activated = true;
    e.Course = c;
    e.Student = student;
    db.Enrolleds.Add(e);
  }
}

  db.SaveChanges();
```

Listing 6.12: Enrolling students to courses.



Title

Lists and Elements

Description

In Robs memory (robot.storage) is a list with different data types stored. Print each element of this list.

Task/Question

Lists and Elements

Solution

38.8
Temperature
Soil
12
Sandy Surface

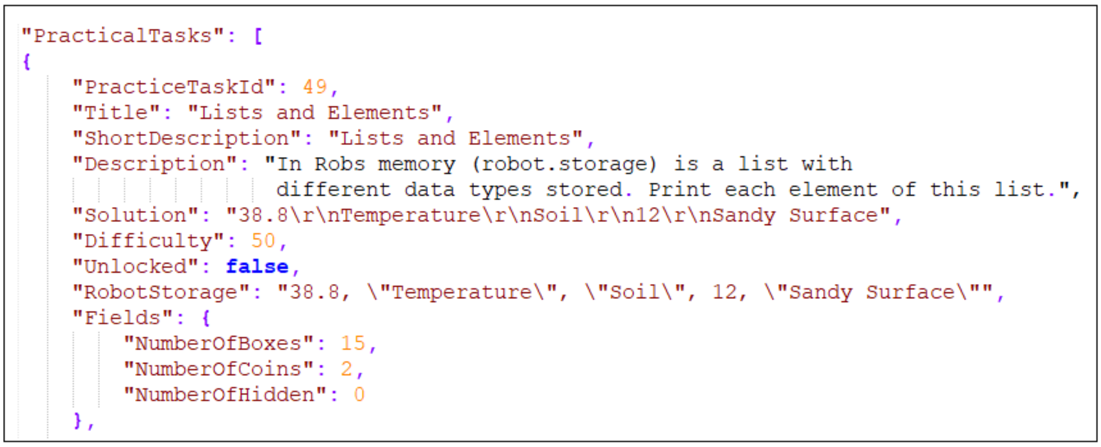
Difficulty

50

Robot Storage

38.8, "Temperature", "Soil", 12, "Sandy Surface"

Figure 6.13: Add data to the robot storage via web application.



```
"PracticalTasks": [
{
    "PracticeTaskId": 49,
    "Title": "Lists and Elements",
    "ShortDescription": "Lists and Elements",
    "Description": "In Robs memory (robot.storage) is a list with
                    different data types stored. Print each element of this list.",
    "Solution": "38.8\r\nTemperature\r\nSoil\r\n12\r\nSandy Surface",
    "Difficulty": 50,
    "Unlocked": false,
    "RobotStorage": "38.8, \"Temperature\", \"Soil\", 12, \"Sandy Surface\"",
    "Fields": {
        "NumberOfBoxes": 15,
        "NumberOfCoins": 2,
        "NumberOfHidden": 0
    },
```

Figure 6.14: Transmitting the robot storage over the REST API.

6.3 Summary

This chapter introduced the further development of the pedagogical concept in sCool. The improvements in the robot missions make it possible to apply new computational skills. The introduction of field types can lead to a better understanding for concepts like loops, conditions or arrays. The different field types should also increase the game's diversity and lead to a more engaging game type. They are represented by a hierarchical structure where each field has a different behaviour. The web application enables a configuration of the number of field types which can help the educators to create adaptive content. Version 1 of sCool had a few issues at the playing field (e.g. placing the disk). In version 2 these issues were fixed to offer the players a game where they can fully focus on the tasks in the game. The onboarding system should make the functionality of the user interface more understandable and reduce regarding questions. This should be achieved by an event-triggered tutorial system. Depending on the complexity of an instruction it can be a single message or a sequence of messages. Another way to help students understanding the user interface is the simplification of the keyboard. The order of the buttons was changed and some ambiguities were resolved.

The mobile video game sCool is attached to the web application since there is a permanent communication. This makes it necessary to adapt the web platform as well. A key requirement of the overall sCool platform is to provide a highly adaptive video game. To achieve this goal the new concepts had to be integrated into the web application as well. The web application is now capable to make some configurations regarding the playground fields. The educators can also pass external data into the video game via the API. Since the web application combines course creation and analysing of learning analytics it should but easy to use. Version 1 of the web application had issues when it comes to multiuser support. This issue was resoled to make sCool available for a broader user group.

A central aspect of the further development was to extend and improve the existing code base and work with the same technologies. The ASP.NET MVC web framework made an extension of the existing application easy. A high level of abstraction could be reached by using the object-relational mapping (orm) framework *Entity Framework*. This framework provides a straightforward handling for serialization and deserialization of the game objects.

7 Evaluation

This chapter covers the evaluation of the further development on sCool (version 2). Four evaluations will be presented with different target groups. All experiments included a questionnaire with questions answering aspects of the corresponding research focus. Each survey included at least some general and game-related questions (see Chapter 4), to see how the game has been improved from version 1 to version 2.

The first section covers the general scope of the evaluation and introduces the research questions. The used instruments for data collection and analysis will be presented in detail. In the following section the target groups and their participants are described. The central aspect of this chapter is the evaluation of all experiments where each is described in detail and the gathered data is going to be analysed to receive a meaningful result. All collected information is considered in terms of the research questions to get a general view of the evaluation.

7.1 Scope

The general structure was similar in all evaluations. The overall duration was between two and four hours were the participants had to solve different tasks. In all experiments the students had between 50 to 60 minutes for working with sCool and solving both the concept-learning and practical tasks. Since there is a huge variety between the participating educational institutes the introduced concepts had to be presented in a suitable way for each group of students. The general learning content was the same in all experiments and covered commands and loops. The presentation of the content in terms of phrasing and simplicity was based on the age group. There are three research questions that should be answered for the evaluation of the further developed sCool video game:

- RQ1: Is the chosen pedagogical concept suitable for the particular target group?
- RQ2: What is the appropriate age group for sCool?
- RQ3: Will the game's re-design lead to a better understanding of the game?

7.2 Instruments and Setup

Since the target groups of the evaluation were different, the instruments had to be adjusted for each experiment. Some general questions regarding gender, age, and smartphone usage were asked in each experiment. In the secondary school and on RMIT two standardised questionnaires were used: Game Engagement Questionnaire (Brockmyer et al., 2009) and Computer Emotion Scale (Kay and Loverock, 2008). Game-related questions were asked in all evaluations. In addition, all questionnaires provided the possibility to write an open-ended answer about the overall game experience and give feedback.

The way of data collection depended on the age group. The pupils in primary school got a sheet of paper with all questions on it (see Appendix). The data was manually transferred to a Microsoft Excel Spreadsheet and analysed. In secondary school the data was collected using Google Forms. The students could answer with their own mobile devices or the computers in the school's ICT room. The data from Google Forms was converted into a comma-separated file and added to a spreadsheet as well. Due to the RMIT's privacy policy it was necessary to use the approved web application *Qualtrics*[1]. The tool is hosted inside the RMIT infrastructure. The data was exported into a Microsoft Excel Spreadsheet for further analysis. The evaluation of the standardised questionnaires (GEQ and CES) was done using the programming language R. Listing 7.1 shows the code for analysing the Game Engagement Questionnaire.

```
absorption <- colSums(Data[Data[,1] \%in\%
  c('4', '13', '3', '8', '9'),][,-c(1,2,8)])
flow <-colSums(Data[Data[,1] \%in\%
  c('19', '10', '6', '18', '5', '7', '11', '14', '15'),]
  [,-c(1,2,8)])
presence <- colSums(Data[Data[,1] \%in\%
  c('2', '12', '16', '1'),][,-c(1,2,8)])
immersion <- colSums(Data[Data[,1] \%in\%
  c('17'),][,-c(1,2,8)])

df = data.frame(cbind(absorption,flow,presence,immersion))
for (i in seq(1,4)) {
  print(mean(c(rep(5,df[1,i]),rep(4,df[2,i]),
    rep(3,df[3,i]),rep(2,df[4,i]),rep(1,df[5,i]))))
  print(sd(c(rep(5,df[1,i]),rep(4,df[2,i]),
    rep(3,df[3,i]),rep(4,df[3,i]),rep(5,df[4,i]))))
}
```

Listing 7.1: Data analysis for Game Engagement Questionnaire in R

[1] https://www.qualtrics.com

7.2.1 Game Engagement Questionnaire

The Game Engagement Questionnaire (Brockmyer et al., 2009) consists of 19 items. Each item is related to a certain kind of engagement: absorption, flow, presence, and immersion. The 19 items are rated on a likert scale from 1 to 5.

- **Absorption**: This means to be totally engaged in an activity with an altered state of consciousness.
- **Flow**: The term flow goes back to Csikszentmihalyi, Abuhamdeh, and Nakamura (2014) and means a balance between skill and challenge. Being into the state of flow has a positive impact on learning.
- **Presence**: Presence is linked to a regular state of consciousness and being aware of the virtual environment that the person is in.
- **Immersion**: Immersion is a state of consciousness where the person is totally engaged in a virtual environment.

7.2.2 Computer Emotion Scale

The Computer Emotion Scale (Kay and Loverock, 2008) is a survey that covers 12 different emotions. The participants were asked how often a certain feeling occurred while interacting with a system: None of the time, Some of the time, Most of the time, All of the time. This number of occurrence can be mapped to values from 1 to 4 and represented on a likert scale. The twelve emotions are grouped together to superior emotions:

- **Happiness**: Satisfied, Excited, Curious
- **Sadness**: Disheartened, Dispirited
- **Anxiety**: Anxious, Insecure, Helpless, Nervous
- **Anger**: Irritable, Frustrated, Angry

7.2.3 Game-related questions

In every experiment the same game-related questions were asked. This questions cover game and learning experience with the sCool video game.

- The game's structure is coherent
- The game's control is easy to use
- The usage of the keyboard was easy
- The language is understandable
- The levels were easy to master
- I learned something while playing the game
- sCool encouraged me to learn more about programming

- The work sheet was easy to solve
- The type of game is fun
- The character Rob is appealing
- The theme space is interesting
- Playing the game was pleasing
- Programming was fun

7.3 Participants

The evaluation of the mobile video game sCool was done with several participants through multiple age and educational groups. The youngest group were pupils of a 4th grade primary school in Graz. In total 14 pupils (9-11 years old) attended this experiment. The experiment was also taken in a 7th grade academic secondary school in Graz with 28 pupils (12-14 years old). The last evaluation was done on the Royal Melbourne Institute of Technology (RMIT) in Australia with 25 students.

7.4 Experiments and Results

In this section four evaluations will be introduced regarding sCool as a tool for learning computational skills. The first evaluation was in the scope of a workshop with ten computer science teachers. The video game was presented and the teachers could give feedback based on their professional experience. In a cooperation between Graz University of Technology and the Viktor Kaplan primary school two workshops were held. The aim was to motivate pupils for computer science and coding using sCool. The next experiment took place at the Graz International Bilingual School where sCool was used with two groups in computer science class. The last experiment covered the usage of sCool in tertiary education on the Royal Melbourne Institute of Technology (RMIT). In the bachelor's programme of the School of Business IT and Logistics a learning activity was conducted where students worked with sCool as revision on previous learned concepts.

7.4.1 Teacher's Evaluation

The teacher's evaluation was part of the event *Tag der Informatik Fachdidaktik 2019* hosted by the University of Teacher Education Styria on March, 23th 2019. This event is an exchange for all teachers that are interested in the field of computer science to attend workshops with different topics.

The title of the hosted workshop was *sCool - Game-based learning in Computer Science Class*. In 90 minutes sCool was introduced to ten teachers of different school types. The goal of the workshop was to present the mobile video game sCool, the web platform and to evaluate the game and the teaching documents with teachers.

In the first 15 minutes sCool was introduced. They were also shown the teaching documents for the different school types and the worksheets in order to discuss the pedagogical usage. After this introduction they could play the game on both windows computers and their mobile devices. During these 45 minutes the teachers played the course that was created for the secondary schools in the pre-evaluation (see Chapter 4). In the last 30 minutes the teachers answered a survey regarding their experience in programming class and sCool as game-based tool for coding. The workshop ended with a discussion about coding in school and the teachers opinion on using sCool in class.

Six teachers (60%) are teaching at a new secondary school and the other four participants (40%) teach at vocational schools. The relation between female (50%) and male (50%) teachers was even. Figure 7.1 shows that the teachers professional experience is highly divergent. Seven teachers (70%) respond that they use apps in class and eight teachers (80%) are teaching programming in computer science class. The teachers that use apps in class also answered what requirements they have for an educational app (see Figure 7.2). Figure 7.3 illustrates which programming languages are used in school. Five teachers (62.5%) are using a block-based approach (Scratch or PocketCode) and also work with the programming language JavaScript. Only two teachers (25%) are working with the Python programming language. Another question regarding programming is aimed to figure out what concepts in programming are hard to understand for students. Seven teachers (87.5%) respond that abstraction and generalisation are hard to understand followed by decomposition and algorithm design (75%). Just two teachers (25%) think that data types and recursion are hard to understand and a single person (12.5%) respond that loops are hard to understand (see Figure 7.4).

The teachers were also asked questions regarding sCool in school: Eight teachers (80%) respond that they would use sCool in class. All participants answered that they would use it in order to introduce programming and computational skills and motivate students in this way. All teachers who would use sCool in school (80%) agreed that sCool would fit best for secondary school lower cycle (5th to 8th grade), and two of them respond that they would use it for primary school or higher cycle too. All participants answered that they would use provided teaching documents and worksheets in class. Two participants respond with textual feedback to the open-ended questions:

- *If you have multiple access and the internet connect is poor (like in most schools), the communication with the server takes too long.*
- *Great game for the beginning!*

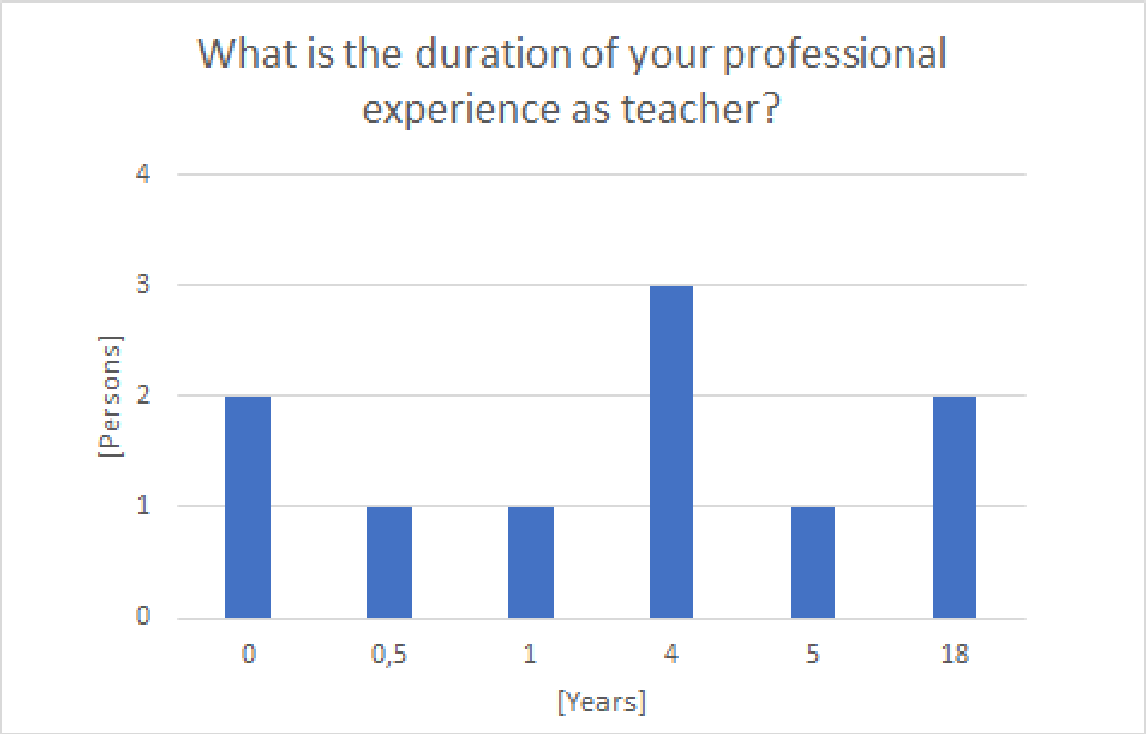


Figure 7.1: Duration of the teacher's professional experience.

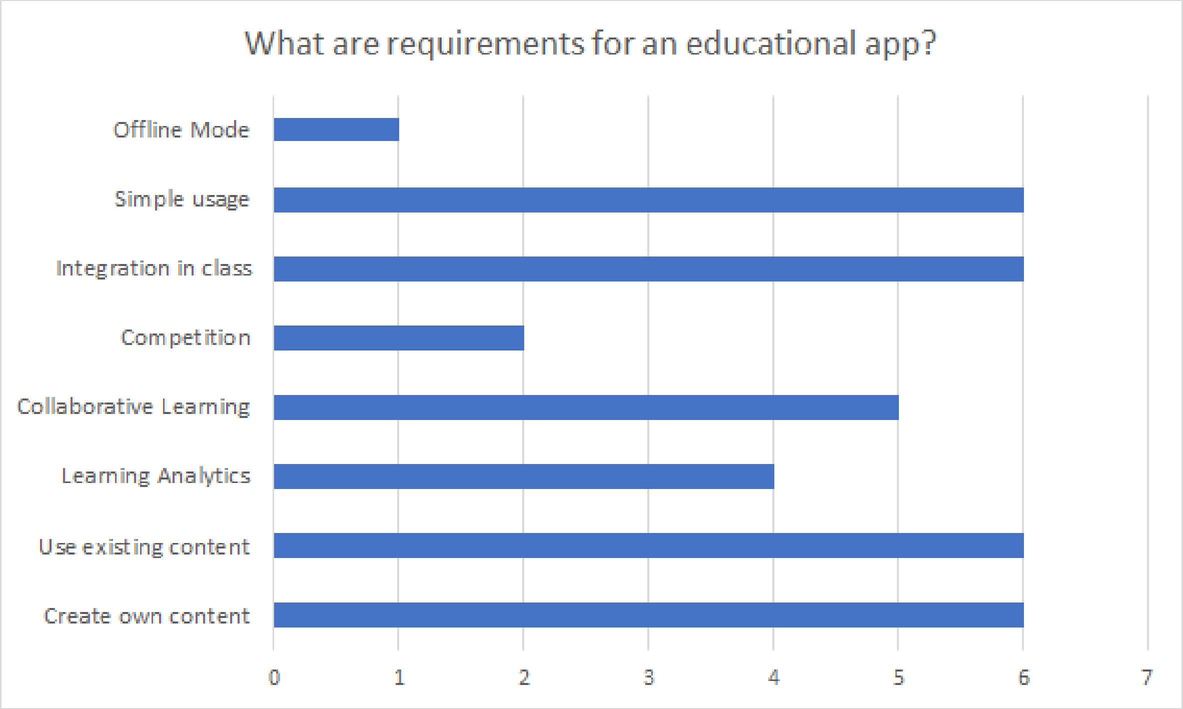


Figure 7.2: Requirements for an educational app in class.

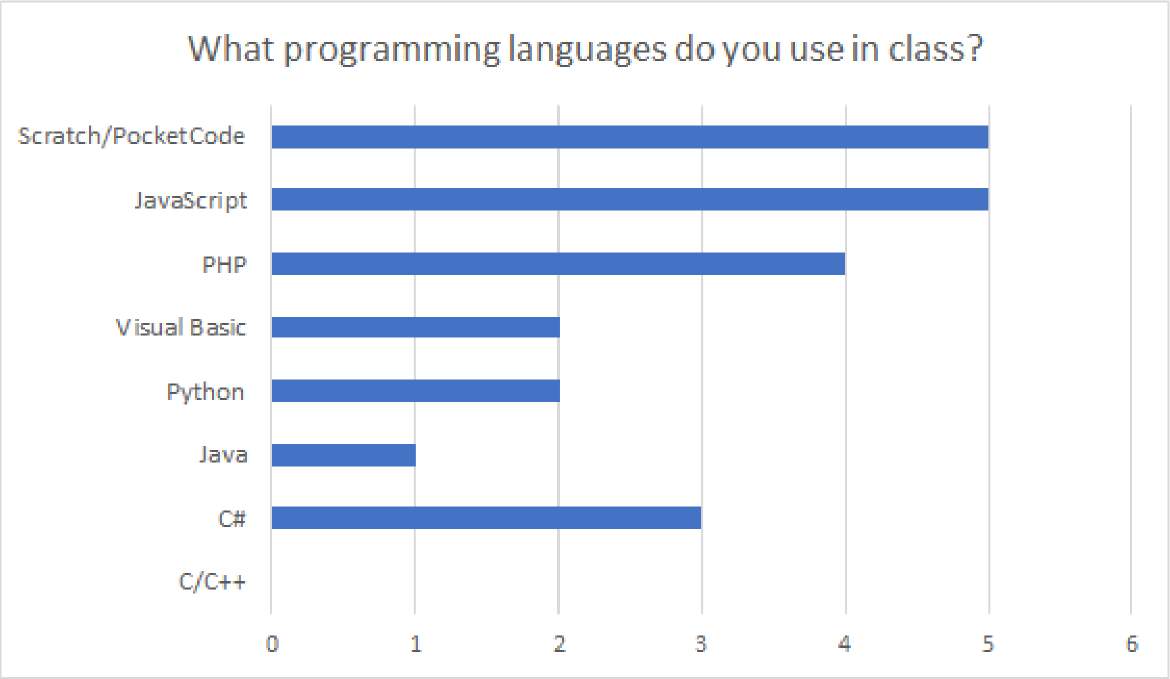


Figure 7.3: Overview of programming languages that are used in school.

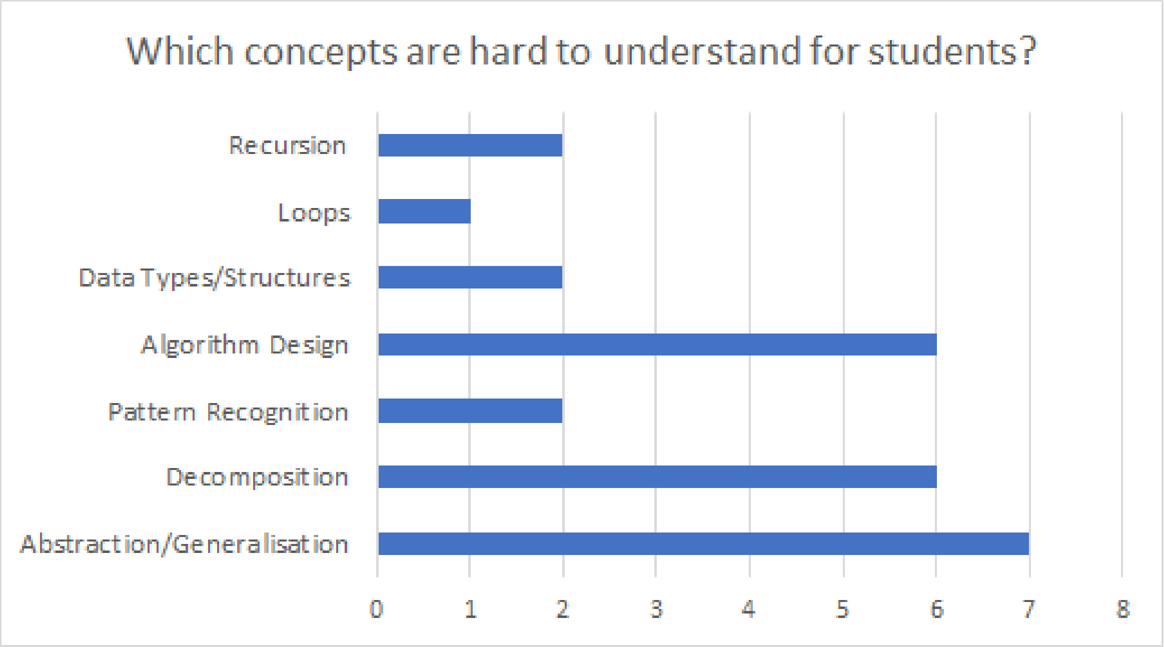


Figure 7.4: List of programming concepts that are hard to understand for students.

7.4.2 Experiment 1: Primary School

The experiment in primary school was conducted at the *Viktor Kaplan* school in Graz. The computer science teacher of the school works with pupils in an elective course on different technological topics. In terms of this cooperation two workshops on sCool and computational skills was scheduled for two school lessons. The 4th grade class consisted of 14 pupils (7 girls and 7 boys) in the age of 9 to 11. The children had no further knowledge on coding or computational skills.

The first workshop was about fundamental concepts in programming (commands and sequences) with the goal to motivate the children to work with sCool. Figure 7.5 shows the printed two-dimensional playing field of sCool with all relevant game elements. For about one hour sCool was introduced in an unplugged way without using mobile devices. In this way several concepts like moving on the playing field, commands, and loops could be explained simple. Each student had to solve a basic task on the printed playing field before the class was allowed to use mobile devices. This was done to ensure that everyone understands the commands and the concept of moving. The second half of the first workshop was about getting into sCool and solving basic tasks.

The second workshop was about coding in sCool. Before the students were allowed to work with the game they had to do a pre-test (see Appendix) in small groups. In this pre-test the pupils had to move the robot on a two-dimensional field and solve two tasks. They were allowed to use the commands in any notation they want. Afterwards the children could start playing the game for 60 minutes. After this time they had to solve a post-test (see Appendix) with a similar task to the pre-test but with obstacles and a larger playing field.

The research interest of this experiment was to figure our if the students are able to solve the post-test in a meaningful way which shows if they learned something while playing the game. This means the focus was on transferring the learned concepts onto the post-test and not on the performance in the game itself. Another aspect was to figure out how the pupils liked the game. Therefore, the game-related questions were asked in an adapted questionnaire for primary school (see Appendix).

Course Design
At the beginning of the workshop the pupils had to work with a printed 9x9 playing field. This playing field is extended by arrow cards, repeating cards, box cards a robot card, and a disk card. Each student had to do a task on this playing field first to understand the main principles of the commands. Afterwards the students could play missions in sCool to get familiar with the control. They did not play the practical missions, they were just playing the concept-learning tasks. In the next workshop the students started with a pre-test where they had to solve a simple task: Rob the Robot has deliver two astronauts with food on a 10x10 playing field. Therefore, they were allowed to use any notation they want. In the next step the students played twelve

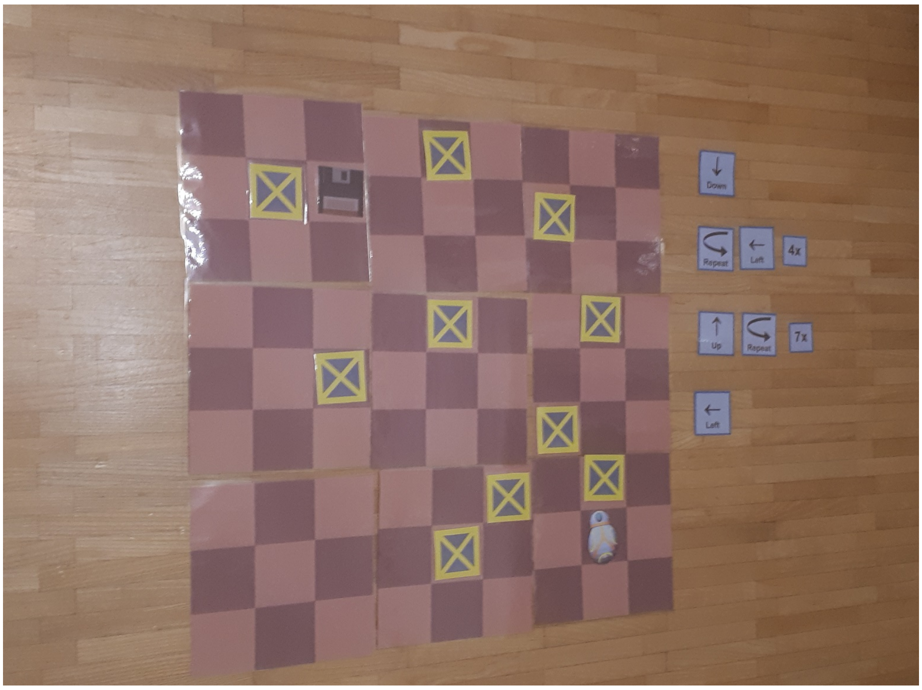


Figure 7.5: Printed version of the sCool playing field for primary school.

sCool missions (seven concept-learning and five practical missions) described in table 7.1. After playing the game they had to solve the post-test and answer the questionnaire. In contrast to the pre-test the students had to deal with a larger playing field (14x14) and different obstacles. They had to collect three different items (screw-nut, lightbulb, and a screw) and bring them to the space shuttle.

Results
The second workshop started with the pre-test that was done in small groups with two or three pupils. All seven groups had 10 minutes to work on the task. All groups were able to solve the pre-test in a meaningful way. The pupils did not have to use any certain notation for the commands. Five groups (71.43%) used loop-like commands where they wrote the number of repetitions and the direction next to each other (see Figure 7.6). The other groups (28.57%) wrote all commands down. After the pre-test the students played the sCool course for one hour. Within the given time every group was able to solve all tasks in the game. Especially the boys were highly motivated through the in-game store and wanted to buy better equipment. To reach more virtual currency they even repeated different missions (in particular those with higher difficulty to receive more coins). Since the scope of the task in the post-test is more complex than in the pre-test the pupils were encouraged to use loops. Despite the students did not have to use a certain notation for the commands all groups wrote Python code. Seven groups used *for loops* (see Figure 7.7) and one group used the *step* argument of the moving commands. All groups were able to solve the given tasks in the programming language Python. The pupils were allowed to use their phones to check the commands.

#	Task	Concepts	Difficulty
Concept-Learning Tasks			
1	What has Rob to collect?	Commands	15%
2	In how many directions can Rob go?	Commands	20%
3	Rob receives three moving commands, how many steps did he take?	Commands	25%
4	Which of these commands can Rob execute?	Commands	40%
5	Recap - Can you remember what the valid command is?	Commands	30%
6	What command is used to repeat a command?	Commands, Loops	60%
7	Which loop repeats a command seven times?	Commands, Loops	70%
Practical Tasks			
1	Reach the disk using the arrow keys.	Commands	30%
2	The disk is very far away. Nevertheless, try to reach it.	Commands	55%
3	The disk is still far away.	Commands, Loops	65%
4	Reach the disk using loops, so you do not have to use the arrows for each step.	Commands, Loops	75%
5	The disk is at its furthest points - try to reach it using loops, it is worth it!	Commands, Loops	95%

Table 7.1: This table shows the course design of the sCool course in primary school.

Table 7.2 shows the results of the game-related questions. 11 pupils (78.57%) respond that they like the type of game and think that the structure is coherent. 9 persons (64.29%) think that programming was fun. The number of pupils that answered *"I learned something while playing the game"* is low (4 students; 23.38%) and six pupils (42.86%) said that they did not learn anything while playing the game. This can be traced back to the circumstance that the pupils did not know exactly that they learned something while playing. All groups were able to solve the post-test with loops, which shows that they actually learned something. The questionnaire included a section for open-ended feedback as well. The following feedback was given:

- Aim for enemies
- More levels, more characters, more weapons
- More advanced enemies per level, stronger weapons, other maps
- More enemies, colors for the characters, different weapons, more colorful levels
- Hide from the enemies, less shooting range for enemies, build houses
- Rob should have more enemies, otherwise it is getting boring! It should be more action. The weapons that are more expensive should be better.
- More faces, colours and clothes, for the character; Less shooting range for the enemies because then you can hide. More levels.
- More levels

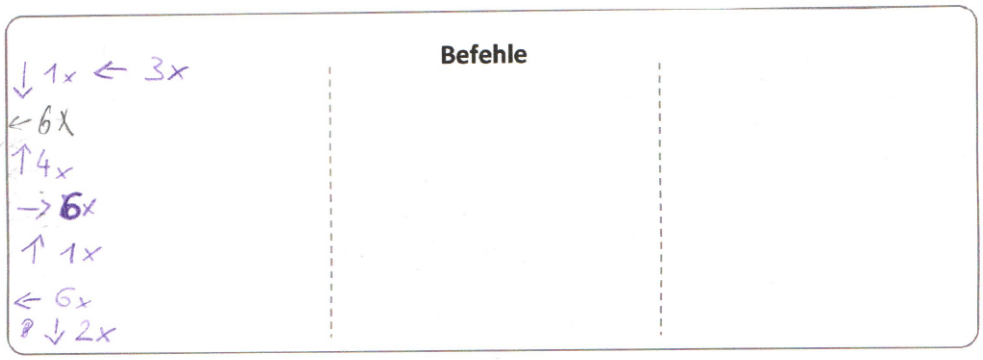
7.4.3 Experiment 2: Secondary School

The experiment was conducted at the *Graz International Bilingual School* (GIBS) in Graz. In total 28 pupils (16 girls, 12 boys) in the age of 12-14 attended the experiment within the computer science class. The computer science class is splitted into two groups with two different teachers. In each group the experiment was scheduled for a double lesson (100 minutes). The school had Android tablets were sCool was installed beforehand in order to use this devices. The ICT infrastructure made it also possible to use the school's WiFi. For this experiment a modified schedule of the pre-evaluation (see Chapter 4.1) was used but the time for playing was increased since the game was already installed. After playing the game a worksheet (see Appendix) had to be solved where the students had to do a basic task using loops. Due to the school's privacy policy the attendance of the final questionnaire was voluntary. In total 12 students filled out the questions provided via Google Forms.

Course Design
The content of the course was created in order to provide a motivating introduction to computational skills and coding. Table 7.3 shows that there were mainly two concepts that were introduced to the class: commands and loops. The course consisted of eight concept-learning missions (five about commands and three about loops) and seven practical tasks (three about commands and four about loops). The level of difficulty

	Agree	Undecided	Disagree
The game's structure is coherent	11	3	0
The game's control is easy to use	4	8	2
The usage of the keyboard was easy	8	4	2
The language is understandable	12	2	0
The levels were easy to master	7	6	1
I learned something while playing the game	4	4	6
sCool encouraged me to learn more about programming	6	3	5
It was easy to solve the worksheet after playing the game	8	6	0
This type of game is fun	11	3	0
The character Rob is appealing	10	1	3
The theme space is interesting	6	4	4
Programming was fun	9	3	2

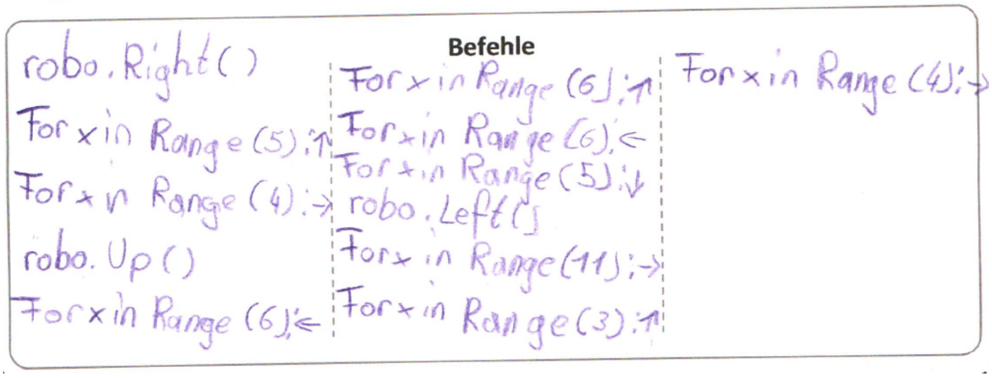
Table 7.2: This table shows the results of the game-related questions in primary school.



Befehle

↓ 1x ← 3x
← 6x
↑ 4x
→ 6x
↑ 1x
← 6x
? ↓ 2x

Figure 7.6: Pre-test of a randomly chosen group consisting out of three girls.



Befehle

```
robo.Right()
For x in Range(5):↑
For x in Range(4):→
robo.Up()
For x in Range(6):←
```

```
For x in Range(6):↑
For x in Range(6):←
For x in Range(5):↓
robo.Left()
For x in Range(11):→
For x in Range(3):↑
```

```
For x in Range(4):→
```

Figure 7.7: Post-test of a randomly chosen group consisting out of three girls.

#	Task	Concepts	Difficulty
	Concept-Learning Tasks		
1	Rob is looking for ...	Commands	20%
2	How can Rob print some text?	Commands	40%
3	In which directions is Rob able to walk?	Commands	25%
4	You tell Rob: "go left", "go left", "go right" and "go right". How many steps did Rob?	Commands	26%
5	What command do Rob understand?	Commands	27%
6	Can you remember a valid command?	Loops	30%
7	What is the command for a loop?	Loops	31%
8	Which loop is repeating a command seven times?	Loops	33%
	Practical Tasks		
1	Use the arrow keys to reach the disk.	Commands	20%
2	Reach the disk again	Commands	50%
3	Reach the disk and print "Rob"	Commands	82%
4	Use the for loop to reach the disks.	Loops	72%
5	The disk is far away again - maybe you should use some loops again?	Loops	81%
6	Repeat Robs name five times and reach the disk	Loops	85%
7	Since we used loops for repeating commands maybe Robs steps could also be counted? Play around and try to counting the steps!	Loops	90%

Table 7.3: This table shows the learning content of the experiment at the Graz International Bilingual School.

was chosen according to the corresponding concept and should guide the students to use loops in more advanced levels.

Results

Overall 23 groups worked on the tasks in sCool. Table 7.4 shows how the groups had performed in the game. Nearly every group was able to solve the concept-learning tasks of the first concept (commands). The hardest level seemed to be the second one (91.30%; 21 groups), where they had to answer how a text is printed in Python. Both first practical levels could be solved by everyone. The last one was only passed by 69.57% (16 groups) of both groups. In this mission they had to reach a disk and print the robot's name. The concept-learning missions at the second concept (loops) could be solved by at least 86.96% (20 groups). The table also shows that the majority was able to solve the first and the second level. The fourth level was just passed by 47.83%

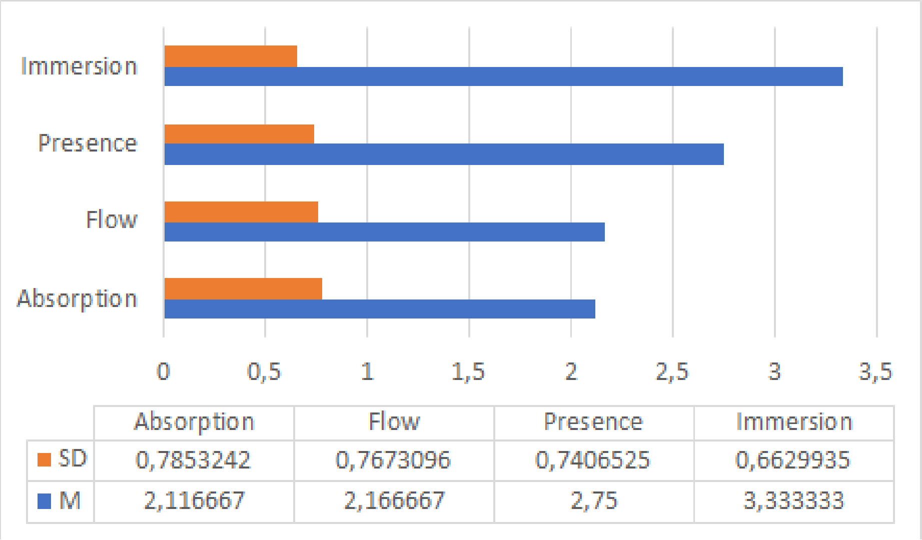
Commands				
Concept-Learning				
Level 1	Level 2	Level 3	Level 4	Level 5
100%	91.30%	100%	100%	100%
Practical				
Level	Level 2	Level 3		
100%	100%	69.57%		
Loops				
Concept-Learning				
Level 1	Level 2	Level 3		
86.96%	91.30%	86.96%		
Practical				
Level 1	Level 2	Level 3	Level 4	
95.65%	86.96%	52.17%	47.83%	

Table 7.4: This table shows the learning content of the experiment at the Graz International Bilingual School.

(11 groups). In this level they had to use loops and optionally count the steps before reaching the goal. In total 12 groups (52.17%) used *for loops* for the practical tasks in a meaningful way.

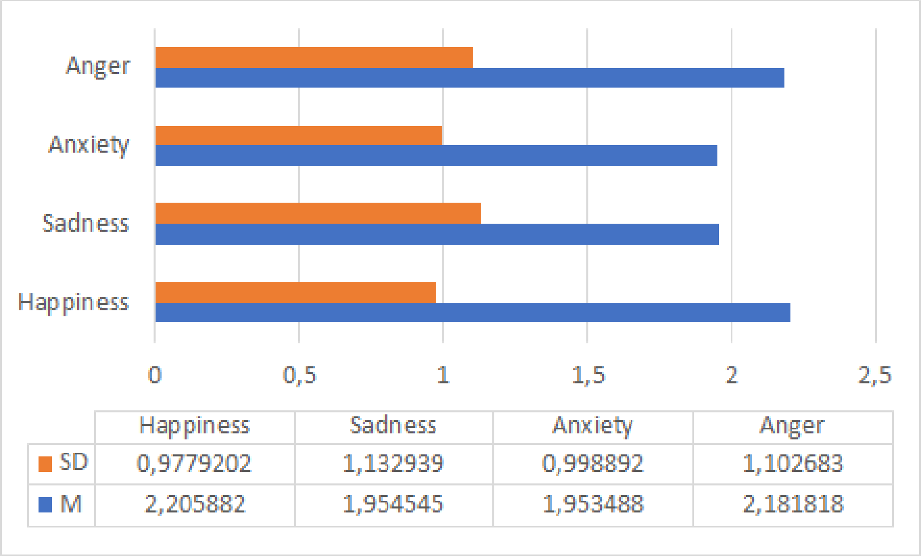
Figure 7.8 shows the results of the Game Engagement Questionnaire. The level of immersion is highest (M=3.33; SD=0.66) which means that the players were engaged in the virtual environment. The level of presence (M=2.75; SD=0.74) is lower than the immersion which means that the students were aware that they are in this kind of environment. The level of flow (M=2.17; SD=0.77) is an indicator for the process of learning. Related to the feelings that came up to the players the Computer Emotion Scale was used for analysing (see Figure 7.9). When it comes to positive feelings (satisfaction, excitement, and curiosity) they occurred the most (M=2.35; SD=0.81). Feelings related to anger (M=0.65; SD=0.73) appear most commonly when it comes to negative feelings.

In terms of game-related questions table 7.5 shows the results. Seven students (58.33%) agreed that the control is easy to use and that the levels were easy to master. Compared to the results of the learning analytics in table 7.4 this value is reasonable. The number of students which answered that they were encouraged by sCool to learn more about programming is low (4 persons; 33.33%). The work sheet could just be solved by three groups (25%) and all of them used *for loops*. Despite this low success rate the number of students (9 persons; 75%) that answers that the work sheet was easy to solve is high.



	Absorption	Flow	Presence	Immersion
🟧 SD	0,7853242	0,7673096	0,7406525	0,6629935
🟦 M	2,116667	2,166667	2,75	3,333333

Figure 7.8: Result of Game Engagement Questionnaire at the Graz International Bilingual School.



	Happiness	Sadness	Anxiety	Anger
🟧 SD	0,9779202	1,132939	0,998892	1,102683
🟦 M	2,205882	1,954545	1,953488	2,181818

Figure 7.9: Result of Computer Emotion Scale at the Graz International Bilingual School.

	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
The game's structure is coherent	4	3	4	1	0
The game's control is easy to use	3	4	3	2	0
The usage of the keyboard was easy	2	3	4	2	1
The language is understandable	3	3	3	2	1
The levels were easy to master	3	4	4	0	1
I learned something while playing the game	1	4	2	1	4
sCool encouraged me to learn more about programming	1	3	3	1	4
The work sheet was easy to solve	3	6	2	0	1
This type of game is fun	0	12	2	0	2
Programming was fun	2	6	6	0	2

Table 7.5: This table shows the results of the game-related questions at the experiment at the Graz International Bilingual School.

7.4.4 Experiment 3: Royal Melbourne Institute of Technology

The third experiment was conducted at the Royal Melbourne Institute of Technology (RMIT) in Australia. As part of the Bachelor's course *Information Systems Solutions and Design* the learning activity with sCool was part of a revision of previous learned concepts.

The course is compulsory for the undergraduate programme of various study programmes at the School of Business IT and Logistics (BITL) on RMIT. Overall 118 students were enrolled to this course. Within the scope of the course the students learn programming in Python and during the semester they create a business application with a graphical user interface (using TKinter) and connect it to a SQLite database. The course consists of an one hour lecture and a two hour lab. In the lab students must fulfill different tasks to learn the course contents and have the possibility to ask either a professor or the student assistant for advice. In the previous course lessons they learned the fundamentals of programming in Python (functions, conditions, loops, functions) and started with object-oriented programming. In week five the students had to submit an assignment that covered all the fundamentals of programming which means all enrolled students had some experience in programming before the experiment.

As part of this course in academic week six (August 26th to 30th, 2019) a learning activity was conducted to revise the programming fundamentals with sCool. To engage as many students as possible the experiment was personally introduced in week five in the lab and the students were also informed via RMIT's learning management system *Canvas*. The learning activity was held in all five lab classes in week six. Due to organizational reasons the activity was planned for 75 minutes in total. According to the schedule the first 15 minutes were for introducing sCool and installing the software on the devices. Since not every student had an Android device they could use their own notebooks with Microsoft Windows as well. After the students got the instructions they had 50 minutes to work with the game. In the last 10 minutes the students completed the survey.

In total 34 students in five lab classes played the video game and 25 persons conducted the survey. The attendance of the activity was voluntary so the students could leave at any time.

Course Design
The course's content was in coordination with the learning content of the lecture *Information Systems Solutions and Design*. Table 7.6 shows the design of the practical missions in the course. In total the course consisted of eight concept-learning and eight practical tasks. These tasks are parts of three concepts: basics, data types, and control structures. The *basic* skill should introduce the players into the sCool environment and Python. The goal is to get familiar with the robot's moving command, its storage and the print command. This tasks cover sequencing, printing, and basic arithmetic operations. After the players have finished this concept they unlock the skill *data*

types. This skill consists of three tasks as well. The learning objectives are lists, data types and field types. Field types are the in-game equivalent of data types and should help learners to understand the idea of data types. The students also had to apply conditions and loops in a simple way to interact with lists. The third skill was about *control structures* in a more complex way because it was necessary to use both loops and conditions. The last task was to write an algorithm to find the maximum out of a given list of integer numbers.

Results
The attendance at the learning activity and the questionnaire was voluntary for the students. In contrast to the experiments in school the way of working was fully self-determined. Many students finished after a few minutes or worked on other projects parallel. In this way a comparison of the students is hardly possible.

In terms of engagement (see Figure 7.10) the Game Engagement Questionnaire (Brockmyer et al., 2009) showed that there was a high level of presence (M=2.86, SD=0.71) and immersion (M=2.8, SD=0.64). This means that the students were both aware of the virtual environment and engaged into it. The level of flow (M=2.41, SD=0.74) provides information about the degree of challenge and skill which is an indicator for an optimal learning experience. The Computer Emotion Scale (Kay and Loverock, 2008) shows that the players connect sCool mainly with the positive emotion happiness (M=2.35, SD=0.81). Negative feelings like sadness (M=1.46, SD=0.65), anxiety (M=1.44, SD=0.72), and anger (M=1.65, SD=0.73) are nearly equal (see Figure 7.11).

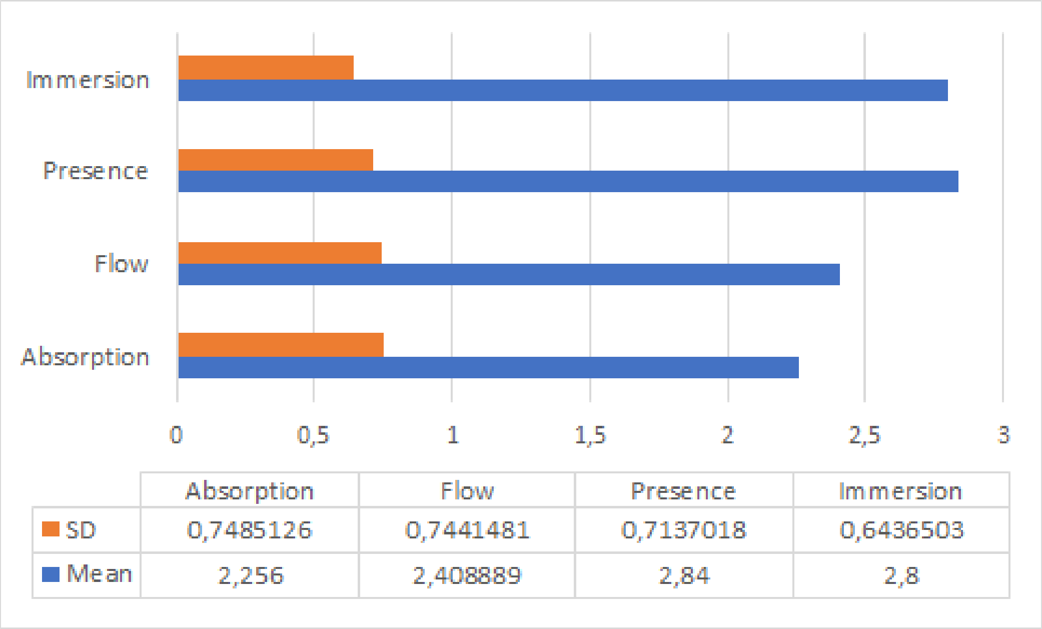
Table 7.7 shows the response to the game-related questions. 18 students (72%) agreed that they learned something while they were working with sCool. Since the sCool course was designed as a revision of already learned concepts the students could learn something in the game. Another 13 students (52%) answered that they were encouraged by sCool to learn more about programming and also 13 persons (52%) stated that programming was fun for them. This evaluation also shows that approximately half of the students (12 persons; 48%) were not satisfied with the keyboard.

Nevertheless, table 7.8 shows the performance of all 34 students. Every student was able to pass the first concept-learning task successfully. The other tasks were also completed by the majority of the students. When it comes to the practical missions just 76.47% (26 students) could solve the first task. In the second task it was just 67.64% (23 students) and the last task could be solved by more than the half of the students (52.94%; 18 students). The second skill was harder for the students since just 32.35% (11 students) solved all concept-learning levels. In the practical missions the second level was the easiest, since 32.35% (11 students) solved it as well. The first task was solved by just 14.70% (5 students) and the last level was completed by just a single student (2.94%). No student reached the third concept (control structures).

The results of the open-ended questions were grouped into five different categories. Figure 7.12 shows that most of the feedback was regarding the game's control (and

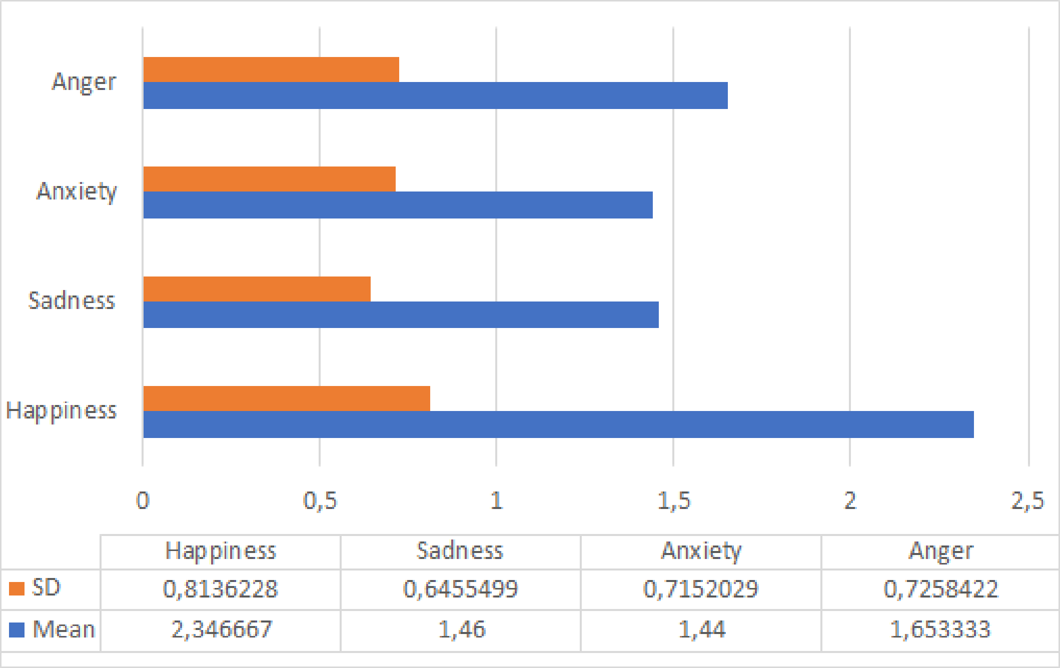
Basics		
1	Collect the Disk	In the first task simply collect the disk by using the command blocks (arrows) for controlling. Drag and drop them into the editor and Rob will move.
2	Hello World	Your mission is to reach the disk and let Rob print "Hello Rob". Therefore, you can use the print command block. Hint: If the way seems to be very long, you could probably use a parameter for the moving commands.
3	Calculating	Help Rob doing a calculation. He has a value in his storage (robot.storage). Can you help Rob to calculate the 5th power of the stored value? Unfortunately, he is very poor at mental arithmetic.
Data Types		
1	Working with lists	Rob stores a list with all planets that he already observed. Could you check if he has already explored the planet "Melmac"? If he was already there, Rob should print "Yes", if he wasn't print "No".
2	Lists and elements	In Robs memory (robot.storage) is a list with different data types stored. Print each element of this list.
3	Field Types	On the playfield are so called "hiddenFields". Rob should avoid these fields because when he travels this field, you will lose half of your coins. Sadly, these fields are not visible, so you should probably scan the playfield for them. When you are done reach the disk and print the total number of all appearing "hiddenFields". Hint: You can access the type of a field via the command robot.field_type(x_coordinate, y_coordinate).
Control Structures		
1	Counting up	Unfortunately, the calculation module of Rob got destroyed during has crash. Can you help him rewrite this module so he can count from 0 to 100 in steps of 5 (i.e. 0 5 10 15 ... 95 100) and print this before reaching the disk? Hint: The command range(min, max, steps) takes three arguments.
2	Tiny calculation	Rob made a few temperature measurements and stored all in his internal memory (robot.storage). Can you write a program to calculate the highest temperature in the memory (using control structures) and print this temperature?

Table 7.6: This table shows the course design at the sCool course on RMIT.



	Absorption	Flow	Presence	Immersion
🟧 SD	0,7485126	0,7441481	0,7137018	0,6436503
🟦 Mean	2,256	2,408889	2,84	2,8

Figure 7.10: Results of the Game Engagement Questionnaire at RMIT.



	Happiness	Sadness	Anxiety	Anger
🟧 SD	0,8136228	0,6455499	0,7152029	0,7258422
🟦 Mean	2,346667	1,46	1,44	1,653333

Figure 7.11: Results of the Computer Emotion Scale at RMIT.

	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
The game's structure is coherent	3	14	3	5	0
The game's control is easy to use	3	12	1	6	3
The usage of the keyboard was easy	1	8	4	8	4
The language is understandable	5	12	6	2	0
The levels were easy to master	4	9	8	4	0
I learned something while playing the game	7	11	4	3	0
sCool encouraged me to learn more about programming	5	8	9	2	1
This type of game is fun	4	12	5	1	2
The character Rob is appealing	2	10	9	3	1
The theme space is interesting	4	13	7	0	1
Playing the game was pleasing	4	8	11	2	0
Programming was fun	4	9	9	2	1

Table 7.7: This table shows the results of the game-related questions at RMIT.

Basics					
Concept-Learning			*Practical*		
Level 1	Level 2	Level 3	Level 1	Level 2	Level 3
100%	85.29%	91.17%	76.47%	67.64%	52.94%
Data Types					
Concept-Learning			*Practical*		
Level 1	Level 2	Level 3	Level 1	Level 2	Level 3
32.35%	32.35%	32.35%	14.70%	32.35%	2.94%

Table 7.8: This table shows the results of the game performance.

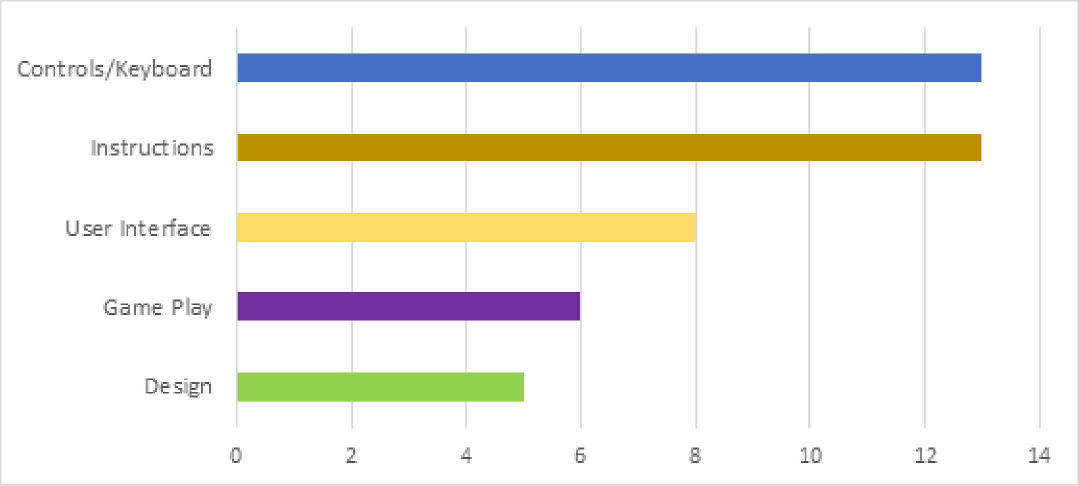


Figure 7.12: Evaluation of user feedback on RMIT experiment.

keyboard) and instructions. Despite the onboarding system many students had questions regarding the user interface. Many of the asked questions were explained at the tutorial. Since many students had to work with Microsoft Windows devices the virtual keyboard was hard to use. The most common request was a *hint* function that can be called if the player has no idea what to do in a task. Despite the students were familiar with Python programming they respond that they had problems with understanding the error messages from the Python interpreter. Table 7.9 includes the summary of the students feedback with the corresponding groups.

In total more than half of all participants were able to complete the *basic* skill (in 50 minutes). Some of the participants quitted earlier since attending was voluntary. About a third of all participants could complete the concept-learning missions of the concept *data types*. The same percentage of students was also able to solve the second task where they had to output a given list. Just one student was able to reach and pass level three (search on game board). Not a single student started with the third concept (control structures).

7.5 Discussion and Limitations

Based on the evaluation of the three conducted experiments in total 67 students were asked different questions regarding sCool. The evaluation also covers the results of the teacher's workshop where 10 teachers where asked about sCool. Since the experiments were done with different person groups a comparison of the overall performance would not be expedient. The aim of the evaluation was to to figure out how the pedagogical concept in each group helps to understand the learning content in sCool better (RQ1). This aspect is related to the question regarding the target group for

Controls/Keyboard	• virtual keyboard was hard to use (especially on Windows) • confusing keyboard layout • keyboard is overlapping parts of the screen
Instructions	• missing description of the navigation (menu) • description of levels • onboarding/Tutorial • hints in the game • error messages (from Python interpreter) hard to read • spelling errors
User Interface	• navigation in the menu • code editor (especially indentation) • renaming of buttons to make it more understandable (quit dialogue)
Game Play	• some concept-learning missions are unsolvable • additional game types • more (and more complex) levels • more programming during game (in some cases they seem put-on)
Design	• improvements of graphics • static design (especially in platformer levels)

Table 7.9: This table contains the summary of the user's feedback at RMIT.

sCool (RQ2). The introduction of a pedagogical concept and the further development of the practical missions should make the game more intuitive and help players understanding computational skills. This aspect brings up the question if version 2 of the game is more understandable than version 1 (RQ3).

The set up of the workshops was adapted for each particular group. The pupils in primary school did not have any previous knowledge in programming. This made it necessary to lead them to this topic in a more playful and child-orientated way. The concepts in the game were presented with *Rob the Robot* who accompanies the players through the whole game. The unplugged approach should also help the children to understand the fundamental ideas of coding in a playful way. The coherent usage of the space theme through the game and all learning documents are a common theme and make it easier to apply the concepts in different situations. When comparing pre-test and post-test it was obvious that the students were able to learn something new. Compared to the overall concept in primary school it was necessary to adapt the content for the secondary school in order to create an age-appropriate course. Instead of using an unplugged approach the students should learn the necessary skills in the game. Since some students in secondary school already had prior knowledge in coding a worksheet was provided instead of pre- and post-tests. The concepts were introduced in the game's concept-learning part. In total 23 groups attended the experiment but just three groups were able to find a solution for the final worksheet. At the experiment on RMIT sCool was used as a revision of the learned concepts. Therefore, the focus was on applying the concepts rather than introducing them. The game's content was created in coordination with the lecture notes. Just a third of all students solved all concept-learning missions of the second skill. In this context it should be mentioned that the learning activity was voluntary which also had an impact on the willingness to finish the experiment. In summary the best results were achieved when it comes to a combination of different media to explain concepts (RQ1).

At the teacher's workshop the questionnaire contained an estimation in which type of school they think sCool would fit best. 80% of the teachers answered that it would be best in secondary school lower cycle (5th to 8th grade). Students in this grades are usually between 10 and 14 years. When comparing the game-related questions regarding enjoyment of playing sCool the answers between the schools and university is similar. The students on RMIT were encouraged most to learn more about programming (52%) compared to primary school (42.86%) and secondary school (33.3%). The primary school children (78.57%) and secondary school (75%) enjoyed the type of game more than RMIT students (66.6%). The responds in primary school was the highest when answering the questions *programming was fun* (64.29%). In secondary school (50%) and on RMIT (52%) the results are nearly even. The results show that students of all age groups are enjoying the game and learned something while playing. In primary school the focus was more on the playing aspect of the game since the children spent most of the time in the planet exploration mode. In secondary school the students played both modes and also were able to apply the learned concepts in the game and even some could solve the worksheet. The students on RMIT also enjoyed playing sCool but more in terms of diversion in the regular lab class. Since sCool was not evaluated in a secondary school higher cycle this group of students cannot be minded in this consideration. According to the estimation at the teachers workshop and the evaluation, probably the most appropriate group are students between 10 and 14 years (RQ2).

A major aspect of sCool's re-design was to make the interaction with the robot mission's user interface easier. For this purpose an event-triggered onboarding system was created (see Chapter 6). At the experiment on RMIT the sCool video game was not introduced before playing to figure out how self-explaining the system is. When analysing the game-related questions 37.5% answered that the controls are not easy to understand and 48% responded that the keyboard is not easy to use. The evaluation of the open-ended answers regarding improvements showed that most requests wished for an improvement of the controls and the keyboard and more instructions and assistance in the game. During the experiments it could be observed that many students simply skip the text-based tutorial. Some general questions regarding the control (for example problems with finding the number or special characters) could be resolved in version 2. Since many students played with Microsoft Windows notebooks they requested for a keyboard support. In summary the usage of the user interface was improved slightly but there is room for improvements in future versions (RQ3).

8 Lessons Learned

This chapter summarizes the issues and experiences during the whole project. It covers literature, development, and didactics

8.1 Literature

One of the most important parts of the further development on sCool was the introduction of a pedagogical concept. There is a large number on different game-based tools and platforms with the aim to teach coding or computational skills in general. To receive a comprehensive overview of the most common tools, related papers and studies were considered. The results of this search were gathered and all tools were analysed in order to evaluate the relevance in terms of this work. The approaches of these tools vary greatly in the pedagogical concept. Some of the tools provide a complex environment and support students, teachers, and parents. Other tools are much simpler and have a focus on a certain concept. For the creation of the pedagogical concept in sCool these tools were analysed and were considered in the concept. When it comes to the presentation of certain concepts in programming there is a lack of literature. Nevertheless, there are lots of case studies that introduce or evaluate certain tools. A comprehensive general study is missing which gives an overview of teaching concepts in programming.

To get an understanding which skills should be part of the pedagogical concept several definitions were considered. In literature there are many classifications on the terms computational think and computational skills. Despite this great amount of definitions the core concepts always remain similar. For the development of the pedagogical concept the skills decomposition, pattern recognition, generalization and abstraction and algorithm design were considered during the design phase.

8.2 Development

The sCool platform consists of two separate components that are connected with each other over a REST API. The mobile video game sCool is written in C# and uses the Unity game engine. The game has two game modes with different purposes. The concept-learning part presents the players new concepts in a textual way. In the robot missions they have to apply the previous learned concepts in a practical way. The main focus of this work was to improve the practical mode to improve the system's capability in coding. For the extension of some game elements it was necessary to create models with the open-source 3D computer graphics software Blender. Some of the improvements also led to an extension of the programming models in order to support the learning analytics of the new characteristics as well.

The second component of the sCool platform is the web application. It is written in C# using the web framework ASP.NET. The infrastructure is hosted on Microsoft's Azure cloud computing platform. Some of the new concepts in the video game required some modification on the web application and the interface as well. This made it necessary to change the existing code and migrate the database according to the new requirements. While conducting the experiments in school some performance issues occurred while connecting to the server which slowed down the game. Three reasons caused this problem: i) the WiFi connection in schools is often very slow since it has a high network traffic, ii) the REST API transmits to many data overhead, and iii) due to scaling mechanisms the Microsoft Azure platform handles the first requests slow before vertical scaling.

During the development process other students worked on sCool as well. This made it necessary to think about an efficient way of collaboration. The web-based application Bitbucket supports version control via Git and was used initially for sCool. The amount of users per repository is limited in this application which made it necessary to move the code base to Gitlab. All involved developers worked on different parts of the game in this way merge conflicts could be avoided. In order to keep the master always clean and buildable it was decided to use a feature branch workflow.

8.3 Didactics

In many phases of the project didactic aspects had to be considered. In scope of the pre-evaluation a first concept was introduced. During the experiments some valuable observations could be made. Textual concepts or instructions are often skipped since the students want to start playing immediately. This led to a lack of understanding the concepts and the user interface. The consequence was to simplify the user interface and make the learning content more appealing. Despite the introduction of the onboarding system some questions regarding interaction with the user interface still could not be resolved.

When an experiment is conducted as part of a school lesson the workshop's structure is a key element for motivation and a good performance. The best results could be achieved when different media was combined in order to explain concepts in several ways. It is important for students to understand the meaning of a concept and its purpose. Another relevant factor is the presentation of the concepts and the tasks. Constantly reflection of the instructions and pretesting can avoid misconceptions beforehand.

9 Conclusion and Future Work

This chapter concludes the key elements and characteristics of this project. It will also give an outlook and will discuss future improvements in terms of the sCool video game, the web application and the pedagogical concept.

9.1 Conclusion

The overall aim of this project was to introduce a pedagogical concept and to improve the already existing sCool platform. In the first phase of the project a pre-evaluation was conducted in two secondary schools to figure out how students get motivated by sCool and if they can apply the learned concepts. Therefore, a course was created and different worksheets that should support students while learning. The result of this evaluation was that it is necessary to improve some of the game mechanics and the pedagogical concept. In a next step the requirements were considered and the concept was developed. Key elements of this improvement are the introduction of an onboarding system and the extension of the robot missions to make the system more capable for meaningful coding. Based on this concept the implementation of the concepts started. In the practical part of the game the students have to apply the learned concepts. Therefore, sCool receives tasks over a REST API which are defined by educators beforehand. The creation of a highly adaptive content is one of sCool's core features. The extension of the existing system provides more possibilities to include additional concepts in the practical part of the game. In order to keep the system as adaptive as possible the interface has to be adjusted as well. Apart from the additional concepts some minor improvements in terms of user interaction were necessary to increase the level of usability and make the system more intuitive. After the process of development further evaluations were scheduled. These experiments were conducted with three different groups: primary school students, secondary school students and university students. The general objectives were: analysing if the chosen pedagogical concept is suitable for the particular group, targeting the appropriate age group for sCool and evaluating the re-designed version of sCool. The pedagogical concept was different for each group of students and the results were described and compared.

Version 2 of sCool still provides a highly adaptive platform where course content can be created and analysed. The web application in version 1 had some issues with the usage of multiple educators. In order to open the system in the future for schools and educators it was necessary to make some improvements. The extended concept of version 2 brings many additional concepts into sCool which makes it possible to introduce new concepts in the the practical mode and to create more comprehensive tasks.

9.2 Future Work

Despite the work on sCool's usability there can be additional improvements. The results of the evaluation showed that there are still issues with the user interface. The most feedback was given related to the controls of the code editor and the keyboard. Instead of sCool's virtual keyboard a native keyboard would lead to a more naturally interaction with the system since the players are more familiar with it. There is also room for additional functionalities like setting the cursor position without the keyboard or using additional code blocks to simplify coding. The results also showed that it is necessary to provide more instructions in the game and add a *help* function for additional hints. This can prevent frustration when a player is stuck in the game.

The experiments also showed that there is a high request on using sCool for other platforms as well. sCool version 2 is currently available for Android and Microsoft Windows. A platform independent version of the game would give everyone the ability to work with sCool. Some of the design decisions have been made for mobile devices (for example the virtual keyboard and dragging and dropping the code blocks) which should be reconsidered when using sCool on other platforms as well.

The ICT infrastructure of schools can be very different and cause problems like slow response times or connection issues. Before sCool can be used effectively in class it is necessary to improve both interface and connection. The current version of sCool requires a permanent connection to the server to synchronise the data. Since not every school is provided with a WiFi connection or has a satisfying internet connection an asynchronous communication between game and web application is recommended.

When evaluating the experiments the impact of a pedagogical workshop design on the success rate can be seen. Therefore, it is necessary to improve not only technical but also social aspects. A well considered workshop design can help students to understand concepts and explain them by different media. In terms of a successful workshop for all participating students it is also of prime importance to mind a gender equal education.

Bibliography

Allen, Deborah E., Richard S. Donham, and Stephen A. Bernhardt (2011). "Problem-based learning." In: *New Directions for Teaching and Learning* 2011.128, pp. 21–29. ISSN: 02710633. DOI: 10.1002/tl.465.

Arkün Kocadere, Selay and Şeyma Çağlar Özhan (July 2018). "Gamification from Player Type Perspective: A Case Study." In: *Educational Technology & Society* 21, pp. 1436–4522.

Australian Curriculum, Assessment and Reporting Authority (2015). *F-10 curriculum. Technologies*. URL: https://www.australiancurriculum.edu.au/f-10-curriculum/technologies/introduction/.

Austrian Federal Ministry of Education, Science and Research (2000). *Lehrpläne der AHS*. URL: https://bildung.bmbwf.gv.at/schulen/unterricht/lp/lp_ahs.html.

Austrian Federal Ministry of Education, Science and Research (Aug. 2016). *Education in Austria*. URL: https://bildung.bmbwf.gv.at/enfr/school/bw_en/bildungswege2016_eng.pdf?6kdmda.

Austrian Federal Ministry of Education, Science and Research (Sept. 2018). *Masterplan Digitalisierung*. URL: https://bmbwf.gv.at/fileadmin/user_upload/Aussendung/Masterplan_Digitalisierung/Masterplan_Digitalisierung_Presseinformation.pdf.

Austrian Federal Ministry of Education, Science and Research (Jan. 2019). *Digitale Grundbildung*. URL: https://bildung.bmbwf.gv.at/schulen/schule40/dgb/index.html (visited on 01/14/2019).

Bartle, Richard (1996). "Hearts, clubs, diamonds, spades: Players who suit MUDs." In: *Journal of MUD research* 1.1.

BBC (Jan. 2019). *Introduction to computational thinking*. URL: https://www.bbc.com/bitesize/guides/zp92mp3/revision/1 (visited on 01/27/2019).

Becker, Katrin (Jan. 2015). *Choosing and Using Digital Games in the Classroom – A Practical Guide*. ISBN: 9783319122229.

Bohyun, Kim (2015). "Chapter 3. Game Mechanics, Dynamics, and Aesthetics." In: vol. 51. 2. DOI: https://doi.org/10.5860/ltr.51n2.

Brockmyer, Jeanne H. et al. (2009). "The development of the Game Engagement Questionnaire: A measure of engagement in video game-playing." In: *Journal of Experimental Social Psychology* 45.4, pp. 624–634.

Brull, Stacey and Susan Finlayson (Aug. 2016). "Importance of Gamification in Increasing Learning." In: *The Journal of Continuing Education in Nursing* 47.8, pp. 372–375. DOI: https://doi.org/10.3928/00220124-20160715-09.

Cardona-Rivera, Rogelio E. and R. Michael Young (Aug. 2014). "A Cognitivist Theory of Affordances for Games." In: *DiGRA &#3913 - Proceedings of the 2013 DiGRA International Conference: DeFragging Game Studies*. ISBN: ISSN 2342-9666. URL: http://www.digra.org/wp-content/uploads/digital-library/paper_74b.pdf.pdf.

Code.org (2019a). *About us*. URL: https://code.org/about.

Code.org (Jan. 2019b). *CS Fundamentals Unplugged*. URL: https://code.org/curriculum/unplugged (visited on 01/27/2019).

CodeMonkey (2019). URL: https://app.codemonkey.com/faqs (visited on 09/18/2019).

Combéfis, Sébastien, Gytautas Beresneviuius, and Valentina Dagiene (2016). "Learning Programming through Games and Contests: Overview, Characterisation and Discussion." In: vol. 10. Vilnius University Institute of Mathematics and Informatics, pp. 39–60. DOI: 10.15388/ioi.2016.03.

Computer Science Teachers Association (2017). *CSTA K-12 Computer Science Standards*. URL: http://www.csteachers.org/standards (visited on 08/21/2019).

Csikszentmihalyi, Mihaly, Sami Abuhamdeh, and Jeanne Nakamura (2014). "Flow." In: *Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi*. Dordrecht: Springer Netherlands, pp. 227–238. ISBN: 978-94-017-9088-8. DOI: 10.1007/978-94-017-9088-8_15. URL: https://doi.org/10.1007/978-94-017-9088-8_15.

Curzon, Paul and Peter W. McOwan (2017). *The Power of Computational Thinking*. WORLD SCIENTIFIC (EUROPE). ISBN: 978-1-78634-183-9. DOI: 10.1142/q0054.

De Freitas, Sara and Tim Neumann (2008). "The use of 'exploratory learning' for supporting immersive learning in virtual environments." In: *Computers & Education* 52, pp. 343–352. DOI: 10.1016/j.compedu.2008.09.010.

Deterding, Sebastian et al. (2011). "From Game Design Elements to Gamefulness: Defining "Gamification"." In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. MindTrek '11. Tampere, Finland: ACM, pp. 9–15. ISBN: 978-1-4503-0816-8. DOI: 10.1145/2181037.2181040.

Du, Jie, Hayden Wimmer, and Roy Rada (Feb. 2018). "Hour of Code: A Case Study." In: vol. 16. 1, pp. 51–60. URL: http://proc.iscap.info/2017/pdf/4343.pdf.

Ertmer, Peggy and Timothy Newby (Oct. 2008). "Behaviorism, Cognitivism, Constructivism: Comparing Critical Features From an Instructional Design Perspective." In: *Performance Improvement Quarterly* 6, pp. 50–72. DOI: 10.1111/j.1937-8327.1993.tb00605.x.

Foerster, Emmy-Charlotte, Klaus-Tycho Foerster, and Thomas Loewe (Apr. 2018). "Teaching Programming Skills in Primary School Mathematics Classes: An Evaluation using Game Programming." In: *9th IEEE Global Engineering Education Conference (EDUCON 2018)*. DOI: 10.1109/EDUCON.2018.8363411. URL: http://eprints.cs.univie.ac.at/5494/.

Gibson, J.J. (1979). *The ecological approach to visual perception*. Boston, MA, US: Houghton Mifflin.

Glod, Gilles (2017). *Gamification: Using game design and game elements in the EFL classroom*. URL: https://files.classcraft.com/classcraft-assets/research/gamification-gilles-glod-2017.pdf.

Google (2019). Computational Thinking for Educators. URL: https://computationalthinkingcourse.withgoogle.com (visited on 08/21/2019).

Grandl, Maria and Martin Ebner (June 2017). "Informatische Grundbildung – ein Ländervergleich." deutsch. In: *Medienimpulse* 2017.2, pp. 1–9. ISSN: 2307-3187.

Green Wood, Ellen, Samuel E. Wood, and Denise Boyd (2005). *The World of Psychology*. 5th ed. Boston, MA: Allyn and Bacon. ISBN: 0-205-43055-4. URL: http://catalogue.pearsoned.co.uk/samplechapter/0205361374.pdf.

Guay, Frédéric, Robert J. Vallerand, and Céline Blanchard (Sept. 2000). "On the Assessment of Situational Intrinsic and Extrinsic Motivation: The Situational Motivation Scale (SIMS)." In: *Motivation and Emotion* 24.3, pp. 175–213.

Heintz, Fredrik et al. (2017). "Introducing Programming and Digital Competence in Swedish K-9 Education." In: *Informatics in Schools: Focus on Learning Programming*. Ed. by Valentina Dagienė and Arto Hellas. Cham: Springer International Publishing, pp. 117–128. ISBN: 978-3-319-71483-7.

Hmelo-Silver, Cindy E. (Sept. 2004). "Problem-Based Learning: What and How Do Students Learn?" In: *Educational Psychology Review* 16.3, pp. 235–266. ISSN: 1573-336X. DOI: 10.1023/B:EDPR.0000034022.16470.f3. URL: https://doi.org/10.1023/B:EDPR.0000034022.16470.f3.

Hunicke, Robin, Marc Leblanc, and Robert Zubek (2004). "MDA: A formal approach to game design and game research." In: *In Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence*. Press, pp. 1–5.

Huotari, Kai and Juho Hamari (2012). "Defining Gamification: A Service Marketing Perspective." In: *Proceeding of the 16th International Academic MindTrek Conference*. MindTrek '12. Tampere, Finland: ACM, pp. 17–22. ISBN: 978-1-4503-1637-8. DOI: 10.1145/2393132.2393137. URL: http://doi.acm.org/10.1145/2393132.2393137.

Ibanez, M., A. Di-Serio, and C. Delgado-Kloos (July 2014). "Gamification for Engaging Computer Science Students in Learning Activities: A Case Study." In: *IEEE Transactions on Learning Technologies* 7.3, pp. 291–301. DOI: 10.1109/TLT.2014.2329293.

Janka, Pekárová (2008). "Using a Programmable Toy at Preschool Age: Why and How?" In: *Workshop proceedings of International Conference on Simulation, Modeling and Programming for Autonomous Robots*. Venice, Italy, pp. 112–121.

Jones, Gary Marshall (1998). "Creating Electronic Learning Environments: Games, Flow, and the User Interface." In: URL: https://files.eric.ed.gov/fulltext/ED423842.pdf.

Kahoot! (2019). *Kahoot!* URL: https://kahoot.com/ (visited on 10/23/2019).

Kamp, Peter (2014). *Computing in the national curriculum. A guide for secondary teachers*. URL: https://www.computingatschool.org.uk/data/uploads/cas_secondary.pdf.

Kant, E. (Nov. 1985). "Understanding and Automating Algorithm Design." In: *IEEE Transactions on Software Engineering* 11.11, pp. 1361–1374. ISSN: 0098-5589. DOI: 10.1109/TSE.1985.231884.

Kay, Robin H. and Sharon Loverock (2008). "Assessing emotions related to learning new software: The computer emotion scale." In: *Computers in Human Behavior* 24.4, pp. 1605–1623.

Kazimoglu, Cagin et al. (2012). "Learning Programming at the Computational Thinking Level via Digital Game-Play." In: *Procedia Computer Science* 9, pp. 522–531. ISSN: 18770509. DOI: 10.1016/j.procs.2012.04.056.

Kim, Nam Ju, Brian R. Belland, and Daryl Axelrod (2019). "Scaffolding for Optimal Challenge in K–12 Problem-Based Learning." In: *Interdisciplinary Journal of Problem-Based Learning* 13.1. DOI: 10.7771/1541-5015.1712.

Kim, Sangkyun et al. (2018). *Gamification in Learning and Education: Enjoy Learning Like Gaming*. Advances in Game-Based Learning. Cham: Springer International Publishing. ISBN: 9783319472829. DOI: 10.1007/978-3-319-47283-6.

Kojic, Aleksandar (2017). "Design and Implementation of an Adaptive Multidisciplinary Educational Mobile Game." Master's Thesis. Graz University of Technology.

Kojic, Aleksandar et al. (2018). "sCool - A Mobile Flexible Learning Environment." In: pp. 72–84. DOI: 10.3217/978-3-85125-609-3-11.

Kojic, Milos (2017). "Procedural Content Generation in a Multidisciplinary Educational Mobile Game." Graz University of Technology.

Kolb's, David (Jan. 1984). *Experiential Learning: Experience As The Source Of Learning And Development*. Vol. 1. ISBN: 0132952610.

Kramer, Jeff (2007). "Is abstraction the key to computing?" In: *Communications of the ACM* 50.4, pp. 36–42. ISSN: 00010782. DOI: 10.1145/1232743.1232745.

Lazzaro, Nicole (2004). "Why We Play Games: Four Keys to More Emotion Without Story." In: URL: http://www.xeodesign.com/xeodesign_whyweplaygames.pdf (visited on 09/03/2019).

Lye, Sze Yee and Joyce Hwee Ling Koh (2014). "Review on teaching and learning of computational thinking through programming: What is next for K-12?" In: *Computers in Human Behavior* 41, pp. 51–61. ISSN: 0747-5632. DOI: https://doi.org/10.1016/j.chb.2014.09.012. URL: http://www.sciencedirect.com/science/article/pii/S0747563214004634.

Mueller, Julie et al. (2017). "Assessing Computational Thinking Across the Curriculum." In: *Emerging Research, Practice, and Policy on Computational Thinking*. Ed. by Peter J. Rich and Charles B. Hodges. Cham: Springer International Publishing, pp. 251–267. DOI: 10.1007/978-3-319-52691-1_16. URL: https://doi.org/10.1007/978-3-319-52691-1_16.

Narasareddygari, Mourya Reddy, Gursimran Singh Walia, and Alex David Radermacher (Aug. 2018). "Gamification in Computer Science Education: a Systematic Literature Re-view Gamification in Computer Science Education: A Systematic Literature Review." In:

National Agency Erasmus+ Education (2014). URL: https://www.bildungssystem.at/en/ (visited on 08/22/2019).

Ortiz, Margarita, Katherine Chiluiza, and Martin Valcke (July 2016). "Gamification in Higher Education and STEM: A Systematic Review of Literature." In: pp. 6548–6558. DOI: 10.21125/edulearn.2016.0422.

Papadakis, Stamatios and Michail Kalogiannakis (Oct. 2017). "Using Gamification for Supporting an Introductory Programming Course. The Case of ClassCraft in a Secondary Education Classroom." In:

Pavlov, I.P. and G.V. Anrep (1927). *Conditioned Reflexes: An Investigation Of The Physiological Activity Of The Cerebral Cortex*. Dover Publications. ISBN: 9780486430935.

Peyton-Jones, Simon, Bill Mitchell, and Simon Humphreys (2013). *Computing at school in the UK*. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/ComputingAtSchoolCACM.pdf.

Piteira, Martinha, Carlos Costa, and Manuela Aparicio (Apr. 2018). "Computer Programming Learning: How to Apply Gamification on Online Courses?" In: *Journal of Information Systems Engineering & Management* 3. DOI: 10.20897/jisem.201811.

Pöckelhofer, Oliver (2019). "sCool - Procedurally Generated Levels For A New Platforming Game Mode." Bachelor's Thesis. Graz University of Technology.

Radoff, Jon (2011). *Game on: Energize your business with social media games*. Indianapolis, Ind.: Wiley. ISBN: 9780470936269. URL: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10513718.

Raymer, Rick (Sept. 2011). "Gamification: Using Game Mechanics to Enhance eLearning." In: *eLearn* 2011.9. ISSN: 1535-394X. DOI: 10.1145/2025356.2031772. URL: http://doi.acm.org/10.1145/2025356.2031772.

Reinders, Hayo (2012). *Digital games in language learning and teaching*. New language learning and teaching environments. Basingstoke: Palgrave Macmillan. DOI: 10.1057/9781137005267. URL: http://www.palgraveconnect.com/pc/doifinder/10.1057/9781137005267.

Riley, David D. and Kenny A. Hunt, eds. (2014). *Computational thinking for the modern problem solver*. Chapman & Hall/CRC textbooks in computing. ISBN: 978-1-4665-8779-3.

Rishipal, Sweta Saraff, and Raman Kumar (2019). "A Gamification Framework for Redesigning the Learning Environment." In: *RECENT ADVANCES IN COMPUTATIONAL INTELLIGENCE*. Ed. by Raman Kumar and Uffe Kock Wiil. Vol. 823. Studies in Computational Intelligence. [S.l.]: SPRINGER NATURE, pp. 93–105. ISBN: 978-3-030-12499-1. DOI: 10.1007/978-3-030-12500-4{\textunderscore}6.

Ryan, Richard M. and Edward L. Deci (2000). "Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions." In: *Contemporary Educational Psychology* 25.1, pp. 54–67. ISSN: 0361-476X. DOI: https://doi.org/10.1006/ceps.1999.1020. URL: http://www.sciencedirect.com/science/article/pii/S0361476X99910202.

Schneider, Marvin Oliver et al. (2016). In: pp. 617–623. URL: http://www.sbgames.org/sbgames2016/downloads/anais/157721.pdf.

Science, K12 Computer (2019). *K12 Computer Science*. URL: https://k12cs.org/.

Sentance, Sue and Andrew Csizmadia (Mar. 2017). "Computing in the curriculum: Challenges and strategies from a teacher's perspective." In: *Education and Information Technologies* 22.2, pp. 469–495. ISSN: 1573-7608. DOI: 10.1007/s10639-016-9482-0. URL: https://doi.org/10.1007/s10639-016-9482-0.

Siang, Ang Chee and Radha Krishna Rao (Dec. 2003). "Theories of learning: a computer game perspective." In: *Fifth International Symposium on Multimedia Software Engineering, 2003. Proceedings.* Pp. 239–245. DOI: 10.1109/MMSE.2003.1254447.

Skinner, B.F. (1938). *The Behavior of Organisms: An Experimental Analysis*. Oxford, England: Appleton-Century.

Skolverket (2018). *Curriculum for the compulsory school, preschool class and school-age educare*. URL: https://www.skolverket.se/getFile?file=3984.

Smith, Megan (2016). *Computer Science For All*. URL: https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all.

SoloLearn (2019). *SoloLearn*. URL: https://www.sololearn.com (visited on 10/23/2019).

Sommerville, Ian (2010). *Software Engineering*. 9th. USA: Addison-Wesley Publishing Company. ISBN: 0137035152, 9780137035151.

Steinmaurer, Alexander, Johanna Pirker, and Christian Gütl (2019a). "sCool – Game-Based Learning in Computer Science Class: A Case Study in Secondary Education." In: *International Journal of Engineering Pedagogy (iJEP)* 9.2, p. 35. DOI: 10.3991/ijep.v9i2.9942.

Steinmaurer, Alexander, Johanna Pirker, and Christian Gütl (Apr. 2019b). "sCool – Game-Based Learning in Computer Science Class: A Case Study in Secondary Education." In: *International Journal of Engineering Pedagogy* 9.2, pp. 26–50. DOI: https://doi.org/10.3991/ijep.v9i2.9942.

Sweetser, Penelope and Peta Wyeth (July 2005). "GameFlow: A Model for Evaluating Player Enjoyment in Games." In: *Comput. Entertain.* 3.3, pp. 3–3. ISSN: 1544-3574. DOI: 10.1145/1077246.1077253. URL: http://doi.acm.org/10.1145/1077246.1077253.

Tan, Debbita, Malini Ganapathy, and Manjet Kaur Mehar Singh (Mar. 2018). "Kahoot! It: Gamification in Higher Education." In: *Pertanika Journal of Social Science and Humanities* 26, pp. 565–582.

Tucker, Allen (2003). *A Model Curriculum for K–12 Computer Science: Final Report of the ACM K–12 Task Force Curriculum Committee*. Tech. rep. ACM Order No.: 104043. New York, NY, USA.

Utecht, Jeffrey R. (2003). *Problem-Based Learning in the Student Centered Classroom*. URL: http://www.jeffutecht.com/docs/PBL.pdf (visited on 08/28/2019).

Wangenheim, Christiane Gresse von et al. (2017). "Teaching Computing in a Multidisciplinary Way in Social Studies Classes in School: A Case Study." In: *International Journal of Computer Science Education in Schools* 1.2, p. 3. DOI: 10.21585/ijcses.v1i2.9.

Warren, Scott and Greg Jones (2017). *Learning Games: The Science and Art of Development*. Advances in Game-Based Learning. Cham and s.l.: Springer International Publishing. ISBN: 9783319468273. DOI: 10.1007/978-3-319-46829-7.

Weintrop, David et al. (Feb. 2016). "Defining Computational Thinking for Mathematics and Science Classrooms." In: *Journal of Science Education and Technology* 25.1, pp. 127–147. ISSN: 1573-1839. DOI: 10.1007/s10956-015-9581-5. URL: https://doi.org/10.1007/s10956-015-9581-5.

Wing, Jeannette (Mar. 2006). "Computational Thinking." In: *Commun. ACM* 49.3, pp. 33–35. ISSN: 0001-0782. DOI: 10.1145/1118178.1118215. URL: http://doi.acm.org/10.1145/1118178.1118215.

Wing, Jeannette (Nov. 2008). "Computational thinking and thinking about computing." In: *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 366, pp. 3717–25. DOI: 10.1098/rsta.2008.0118.

World Government Summit (2016). *Gamification and the Future of Education*. URL: https://www.worldgovernmentsummit.org/api/publications/document?id=2b0d6ac4-e97c-6578-b2f8-ff0000a7ddb6.

Yaroslavski, Danny (July 2014). *How does Lightbot teach programming*. URL: https://lightbot.com/Lightbot_HowDoesLightbotTeachProgramming.pdf (visited on 09/18/2019).

Zenn, Jacqueline (Nov. 2017). *Understanding Your Audience – Bartle Player Taxonomy*. URL: https://gameanalytics.com/blog/understanding-your-audience-bartle-player-taxonomy.html.

Zichermann, Gabe and Christopher Cunningham (2011). *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. 1st. O'Reilly Media, Inc. ISBN: 1449397670, 9781449397678.

Appendix

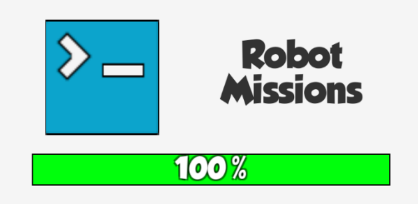
A.1 Intro Worksheet

Rob the Robot

Hello, I am Rob the Robot. I am a research robot and explore foreign planets. During a space mission I had a quite bumpy landing. At this crash all previous collected information was scattered around the whole planet. Can you help me collect all missing disks so I can continue my missions?

Missions

Help Rob on his exploration missions on the foreign planet to collect all disks. Beware of the enemies trying to protect the floppy disks. If you are injured try to find a first aid kit, it will help you. When gathering the floppy disks, you will learn different commands and concepts to control the robot.



In the *Robot Missions* you have to help the Robot collect a single disk. To do this, you must apply all the theoretical knowledge from the *Planet Exploration* missions. Simply drag and drop the command blocks into the workspace to control the robot.

Good luck on your mission!

A.2 Revision Worksheet

Rob the Robot

and the delayed departure

Ready for take-off in …

Now it is done! You finally helped Rob to find all missing disks, so he can fly back to the earth – thank you so much! Unfortunately, there was an error while reading the disks and so the start sequence could not be loaded correctly – it seems like Rob cannot start. Apparently, there is a problem with the calculation module that is needed for the flight back. Rob has to do different calculations but is not possible to solve them. He needs a program that performs different multiplications. The broken program was responsible for calculating the three times table. Can you help Rob to re-write the program, so he can finally escape from the planet and take back home all collected research results?

Output

Rob's output should look like this:

```
3, 6, 9, 12, 15, 18, 21, 24, 27, 30

Calculating finished
```

Code

Hint

Here are some useful commands for your mission:

Print

```
print("Hello Rob!")
```

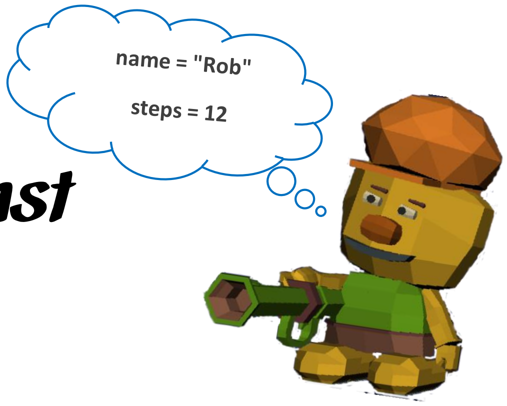
Var

```
steps = 12
name = "Rob"
```

For

```
for x in range(5)
  print(x)
  robot.Left()
```

A.3 Variables Worksheet



Robs fight against forgetfulness

I am sure you know the lowercase letters (x, y, z) from math class. The so-called placeholders (or variables) are written instead of numbers. The value of these variables can also change they does not remain constant. This is very similar in computer science. A variable can store any value (numbers, characters, words or even whole sentences). The name of the variable remains, but the value can change at any time.

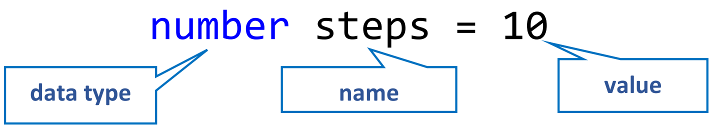
Rob the Robot needs also variables. With these variables he can remember certain information, such as his name or the number of steps he has already taken.

Imagine …

Rob wants to take different measurements on a foreign planet. However, he can only make a total of 10 steps in any direction before recharging his battery. Charing the battery will then take a few hours and means he needs to get enough sunlight. So, he has to device these steps well.

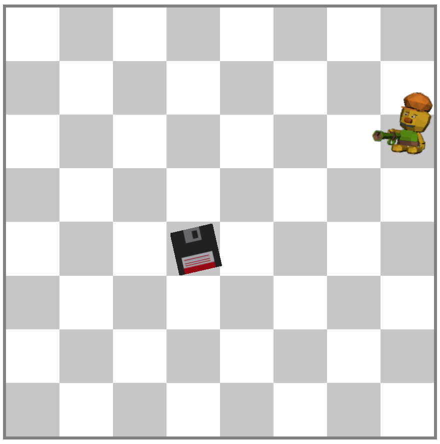
Each variable consists out of three parts:

- **name**: What is the variables name?
- **data type**: What is stored in the variable? A number? A letter? A word?
- **value**: What is the variables content?



```
number steps = 10
```

Can you complete the missing data for Rob so he can arrive his destination?

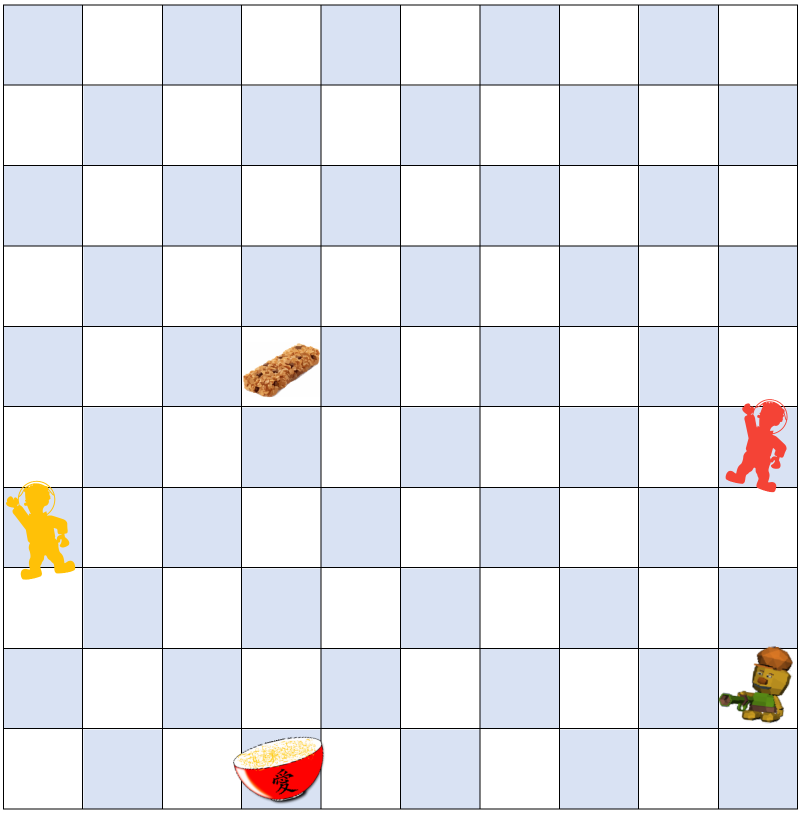


```
steps = 7
robot.Left()
steps = 6
robot.Left()
steps = ___
robot.Left()
_______ = ____
robot.Left()
______________
robot.Down()
______________
robot.Down()
______________
print("Rob has " + _______ + " steps left")
```

A.4 Pre-Test Primary School

Robs Delivery Service

The crew of the spaceship has been on a foreign planet for a few days now. The food supplies are slowly running out and it is time to get some food. The team commits Rob to collect the food on the planet and bring it back to the astronauts. Can you help him pick up the food first and then bring it to the right person? The cereal bar belongs to the yellow astronaut and the noodle bowl to the red one.

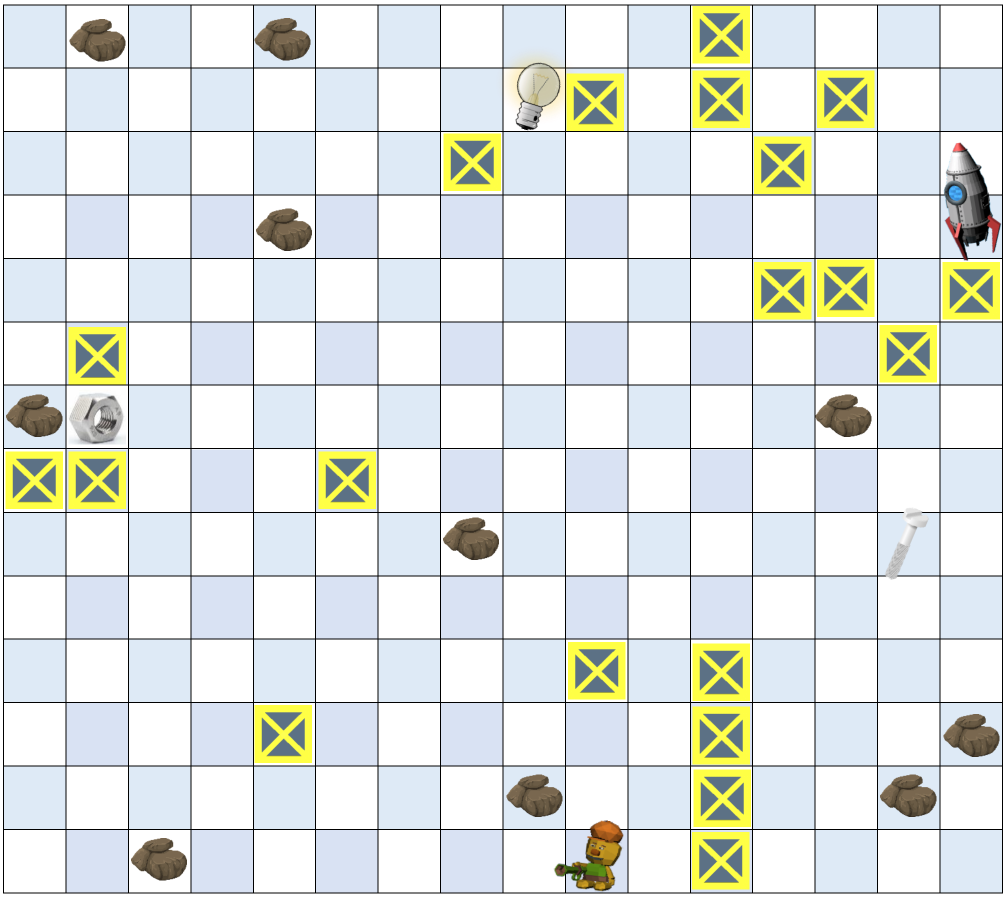


Commands

A.5 Post-Test Primary School

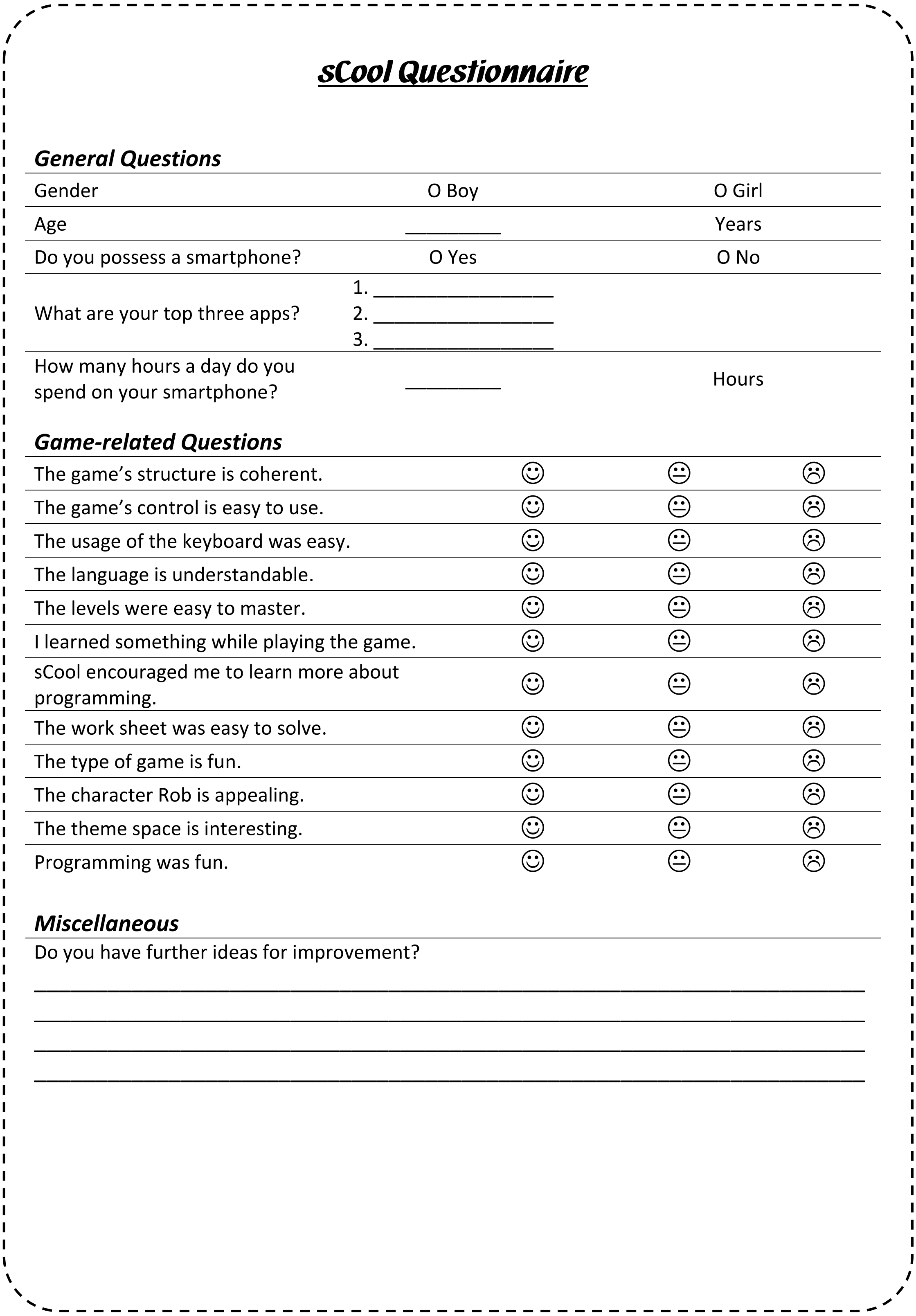
Rob the Mechanic

During the space shuttles crash some parts got lost. All these parts are widely scattered now, and they seem to be lost. Can you help Rob to collect the pieces and then bring them back to the spaceship? Try to avoid the obstacles in the field (rocks and boxes) because Rob cannot pass them.



Commands

A.6 Questionnaire for Primary School



sCool Questionnaire

General Questions

Gender	O Boy	O Girl
Age	_________	Years
Do you possess a smartphone?	O Yes	O No
What are your top three apps?	1. _________________ 2. _________________ 3. _________________	
How many hours a day do you spend on your smartphone?	_________	Hours

Game-related Questions

The game's structure is coherent.	☺	😐	☹
The game's control is easy to use.	☺	😐	☹
The usage of the keyboard was easy.	☺	😐	☹
The language is understandable.	☺	😐	☹
The levels were easy to master.	☺	😐	☹
I learned something while playing the game.	☺	😐	☹
sCool encouraged me to learn more about programming.	☺	😐	☹
The work sheet was easy to solve.	☺	😐	☹
The type of game is fun.	☺	😐	☹
The character Rob is appealing.	☺	😐	☹
The theme space is interesting.	☺	😐	☹
Programming was fun.	☺	😐	☹

Miscellaneous

Do you have further ideas for improvement?

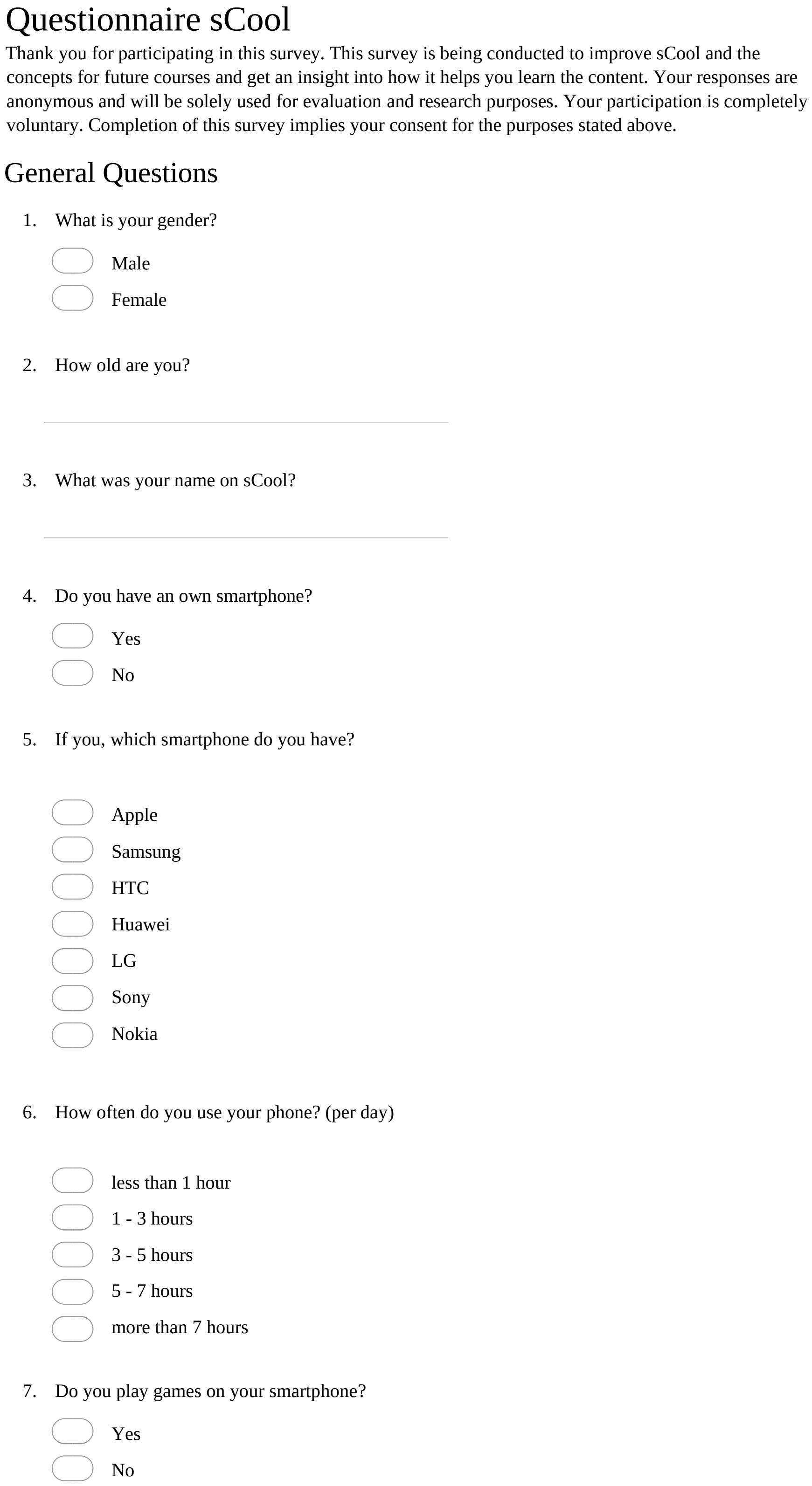
____

____

____

____

A.7 Questionnaire Pre-Evaluation



Questionnaire sCool

Thank you for participating in this survey. This survey is being conducted to improve sCool and the concepts for future courses and get an insight into how it helps you learn the content. Your responses are anonymous and will be solely used for evaluation and research purposes. Your participation is completely voluntary. Completion of this survey implies your consent for the purposes stated above.

General Questions

1. What is your gender?

   ○ Male
   ○ Female

2. How old are you?

   ____

3. What was your name on sCool?

   ____

4. Do you have an own smartphone?

   ○ Yes
   ○ No

5. If you, which smartphone do you have?

   ○ Apple
   ○ Samsung
   ○ HTC
   ○ Huawei
   ○ LG
   ○ Sony
   ○ Nokia

6. How often do you use your phone? (per day)

   ○ less than 1 hour
   ○ 1 - 3 hours
   ○ 3 - 5 hours
   ○ 5 - 7 hours
   ○ more than 7 hours

7. Do you play games on your smartphone?

   ○ Yes
   ○ No

A.8 Experiment Documentation Primary School

Experiment Documentation

General Information

School type: Primary School

Grade: 4th Grade

Concepts: Commands, Sequences, Loops

Time: Two double lessons (200 min total)

Students: 7 boys and 7 girls

Goals: The goal is to introduce the children in a motivating way into programming and computer science. In this workshop they should learn what commands are and see that there are real-life examples for commands to. The students should also know that commands can be grouped together and that the order of execution matters. It is also important that they understand the idea of repeating certain commands.

Schedule

Time	Task	Method	Media
Day 1			
10’	welcome, introducing the project	frontal	-
5’	explaining *sCool unplugged*	frontal	sCool unplugged
5’	solving a sample task together	in plenum	sCool unplugged
50’	solving different tasks	in plenum	sCool unplugged
5’	hand out devices	in plenum	smartphone
20’	work with sCool in the concept-learning part to make pupils familiar with user interface	group	smartphone
5’	hand in devices, goodbye	in plenum	-
Day 2			
5’	welcome	frontal	
10’	running pre-test	group	pre-test
60’	work on the sCool course	group	smartphone
10’	running post-test	group	post-test, smartphone
10’	fill out questionnaire	individual	questionnaire
5’	goodbye	in plenum	-

Concept-learning tasks

<table>
<tr><th colspan="3">Basics</th></tr>
<tr><td>#1</td><td>Rob the Robot</td><td>15%</td></tr>
<tr><td colspan="3">**Description**
Rob is a research robot, crashed on a foreign planet. Help him collecting all disks to escape.
Question
What has Rob to collect?
Answers
- Disks
- CDs
- USB sticks</td></tr>
<tr><td>#2</td><td>Control Rob</td><td>20%</td></tr>
<tr><td colspan="3">**Description**
Rob can be controlled with the arrow buttons. When you are playing the ‘Robot Missions’ you simply have to drag and drop the block buttons on the left. Rob can move up, down, left, and right.
Question
In how many directions can Rob walk?
Answers
- four directions
- five directions
- two directions</td></tr>
<tr><td>#3</td><td>Instructions</td><td>25%</td></tr>
<tr><td colspan="3">**Description**
Rob can understand simple commands like ‘move to the right field’. Each single line represents an instruction. The commands will be executed sequentially.
Question
Rob receives three moving commands, how many steps did he take?
Answers
- 3
- 2
- 1</td></tr>
<tr><td>#4</td><td>Give Rob instructions</td><td>40%</td></tr>
<tr><td colspan="3">**Description**
Rob can understand many commands. All commands do have the same structure: robot.command(). The first part is the word ‘robot’ followed by a dot (robot.). In this way, Rob know that he has to do something. After the dot is the instruction that should be executed. Each command is closed by two brackets. Caution: Rob is very exact when it comes to the right spelling – be case sensitive.
Question
Which of these commands can Rob execute?
Answers
- robot.left()
- robotLeft()
- robot.left</td></tr>
<tr><th colspan="3">Loops</th></tr>
<tr><td>#5</td><td>Recap</td><td>30%</td></tr>
<tr><td colspan="3">**Description**
In the previous lessons you learned that Rob is able execute different commands. In some cases, it is very useful to repeat commands.
Question
Can you remember what the valid command is?</td></tr>
</table>

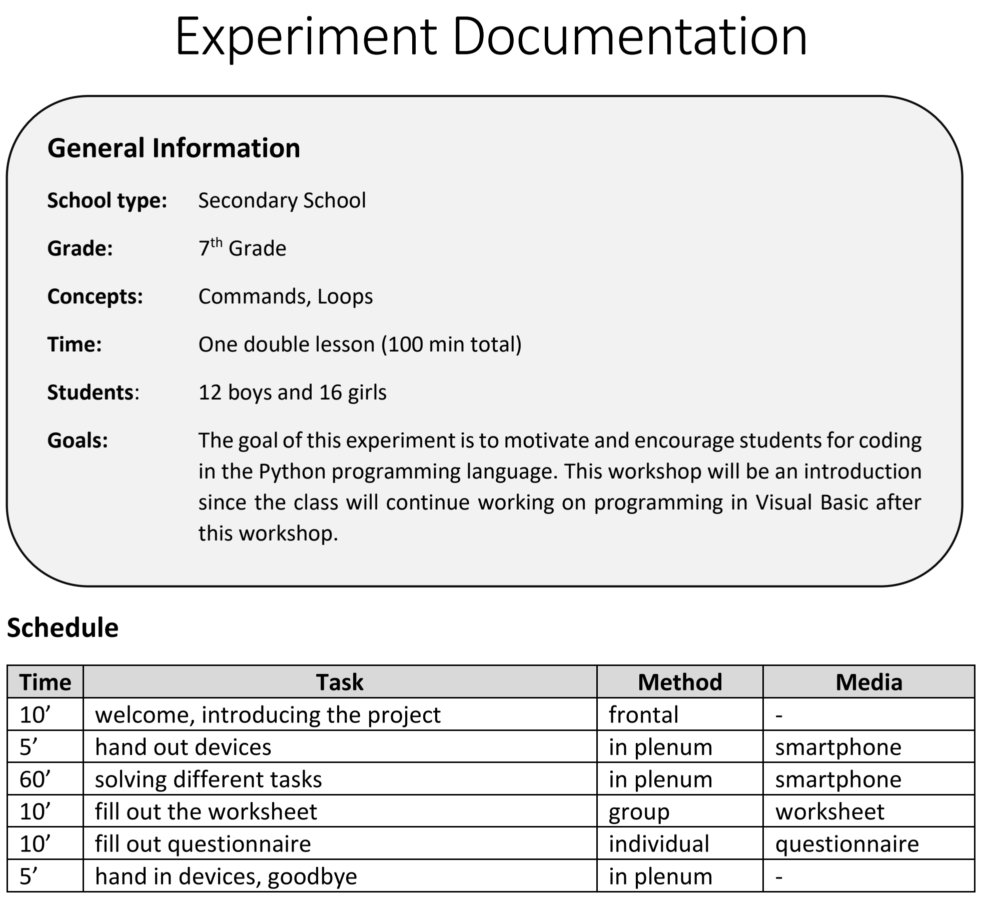
<table>
<tr><td colspan="3">**Answers**
- robot.Left()
- robot.left()
- robot.LEFT</td></tr>
<tr><td>#6</td><td>Loops</td><td>60%</td></tr>
<tr><td colspan="3">**Description**
To repeat certain commands, it is possible to use loops. Loops are used for repeating various commands. Therefore, you can use the command ‘for’.
Question
What command is used to repeat a command?
Answers
- for
- für
- schleife</td></tr>
<tr><td>#7</td><td>for loops</td><td>70%</td></tr>
<tr><td colspan="3">**Description**
A for loop looks like this: ‘for x in range(5):’. In this way the command is repeated five times. Each loop is closed by a colon. This colon means that each command after it will be repeated.
Question
Which loop repeats a command seven times?
Answers
- for x in range(7):
- for x in range(6):
- for x in range(8):</td></tr>
</table>

Practical tasks

<table>
<tr><th colspan="3">Basics</th></tr>
<tr><td>#1</td><td>First attempts of walking</td><td>30%</td></tr>
<tr><td colspan="3">**Task**
Reach the disk using the arrow keys.
Available Commands
Left, Right, Up, Down</td></tr>
<tr><td>#2</td><td>Hiking Day</td><td>55%</td></tr>
<tr><td colspan="3">**Task**
The disk is very far away. Nevertheless, try to reach it.
Available Commands
Left, Right, Up, Down</td></tr>
<tr><td>#3</td><td>Hiking Again</td><td>65%</td></tr>
<tr><td colspan="3">**Task**
The disk is still far away – you know what to do.
Available Commands
Left, Right, Up, Down</td></tr>
<tr><th colspan="3">Loops</th></tr>
<tr><td>#4</td><td>Marching</td><td>75%</td></tr>
<tr><td colspan="3">**Task**
Reach the disk using loops, so you do not have to use the arrows for each step.
Available Commands
Left, Right, Up, Down, For</td></tr>
<tr><td>#5</td><td>Far Journey</td><td>95%</td></tr>
<tr><td colspan="3">**Task**
The disk is at its furthest points - try to reach it using loops, it is worth it!</td></tr>
</table>

Available Commands
Left, Right, Up, Down, For

A.9 Experiment Documentation Secondary School



Experiment Documentation

General Information

School type: Secondary School

Grade: 7th Grade

Concepts: Commands, Loops

Time: One double lesson (100 min total)

Students: 12 boys and 16 girls

Goals: The goal of this experiment is to motivate and encourage students for coding in the Python programming language. This workshop will be an introduction since the class will continue working on programming in Visual Basic after this workshop.

Schedule

Time	Task	Method	Media
10’	welcome, introducing the project	frontal	-
5’	hand out devices	in plenum	smartphone
60’	solving different tasks	in plenum	smartphone
10’	fill out the worksheet	group	worksheet
10’	fill out questionnaire	individual	questionnaire
5’	hand in devices, goodbye	in plenum	-

Concept-learning tasks

Basics		
#1	Rob the Robot	20%
Description Rob is a research robot, that crashed on a foreign planet. By finding all disks he can escape the planet. Can you help him? **Question** Rob is looking for ... **Answers** - disks - USB sticks - CDs		
#2	Print	40%
Description In many cases it would be very useful that Rob would say something. Therefore, he can use the command "print("Hello")". Watch out: Quotes are necessary to write a text. **Question** How can Rob print some text? **Answers** - print("Hi, my name is Rob") - print(Hi, my name is Rob) - shout("Hi, my name is Rob")		
#3	Moving Around	25%
Description Rob can be controlled with blocks. Each block represents a direction (up, down, left and right) that Rob can walk in the "Robot Missions". To let Rob walk simply drag and drop the blocks into the large area (editor). **Question** In which directions is Rob able to walk? **Answers** - up, down, left and right - north, east, south and west - just up and down		
#4	Commands	26%
Description With different commands like "go right" you can tell Rob what to do. Each command is represented into a single line of text. The commands are executed one by one. **Question** You tell Rob: "go left", "go left", "go right" and "go right". How many steps did Rob? **Answers** - Four - Three - Two		
#5	Controlling Rob	27%
Description Rob can understand a few commands. All these commands do have a similar structure (like grammar in other languages): robot.DoSomething(). The first part of the command is always the word "robot" followed up by a dot (robot.). So, Rob knows that you talk to him. Next to the dot is the certain instruction that Rob should execute. Each command is completed by parenthesis: robot.Up(). Watch out: Rob is very case sensitive! (robot.Left() vs. robot.left())		

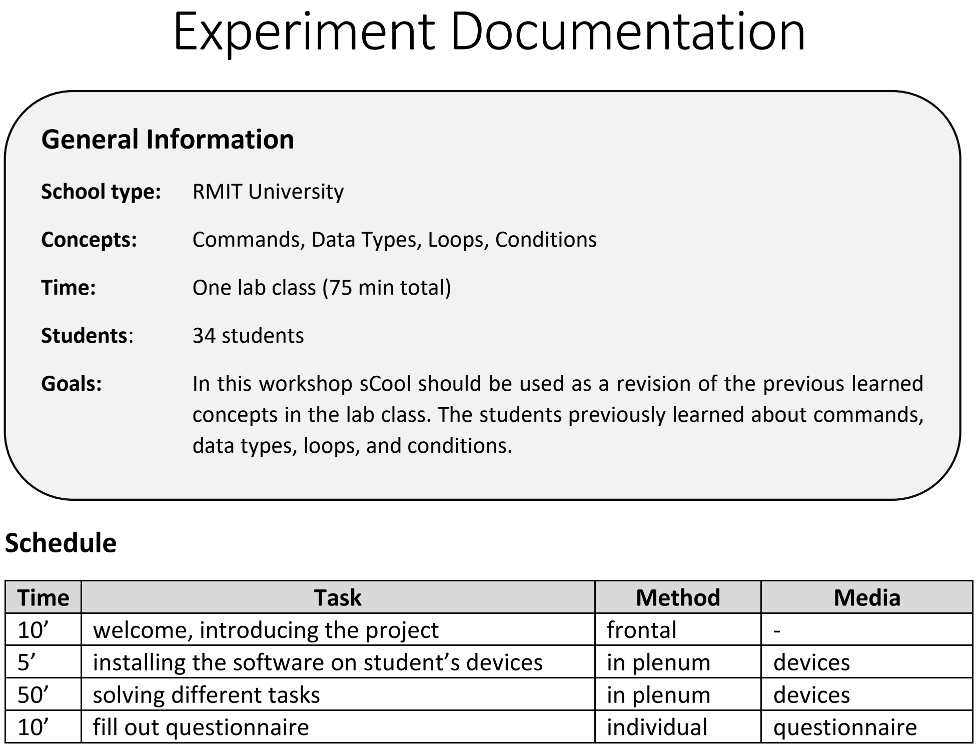
Question What command do Rob understand? **Answers** - robot.Left() - robotLeft() - robot.left()		
Loops		
#6	Revision	30%
Description At the first missions you learned that Rob can execute certain commands. In some cases, it is very usefull to repeat them. **Question** Can you remember a valid command? **Answers** - robot.Left() - robot.left() - ROBOT.LEFT		
#7	Loops	31%
Description To repeat certain commands Rob is using loops. In each loop the number of repetitions have to be declared. To repeat something Rob uses the word "for". **Question** What is the command for a loop? **Answers** - for - repeat - loop		
#8	for loop	33%
Description Here you can see a for loop: "for x in range(5):" With this command a command will be repeated five times. The last character is a colon, that tells Rob to repeat the following instructions. **Question** Which loop is repeating a command seven times? **Answers** - for x in range(7): - for x in range(6): - for x in range(8):		

Practical tasks

Basics		
#1	Attempts At Walking	20%
Task Use the arrow keys to reach the disk. **Available Commands** Left, Right, Up, Down		
#2	Go Hiking	50%
Task The disk seems to be far away - let's try to reach it again! **Available Commands** Left, Right, Up, Down		

Loops		
#3	Walk with Rob	72%
Task Use the for loop to reach the disks. **Available Commands** Left, Right, Up, Down, For		
#4	Long Journey	81%
Task The disk is far away again - maybe you should use some loops again? **Available Commands** Left, Right, Up, Down, For		
#5	Talk with Rob	82%
Task When Rob reaches the disk he should print his name (remember, he is very case sensitive). Reach the disk and print "Rob" **Available Commands** Left, Right, Up, Down, For		
#6	Repeating	85%
Tasks Rob can also repeat his name a certain number of times. Can you say his name five times before he reaches the disk? **Available Commands** Left, Right, Up, Down, For, Print		
#7	Count Steps	90%
Tasks Since we used loops for repeating commands maybe Robs steps could also be counted? Play around and try to counting the steps! **Available Commands** Left, Right, Up, Down, For, Print		

A.10 Experiment Documentation RMIT



Experiment Documentation

General Information

School type: RMIT University

Concepts: Commands, Data Types, Loops, Conditions

Time: One lab class (75 min total)

Students: 34 students

Goals: In this workshop sCool should be used as a revision of the previous learned concepts in the lab class. The students previously learned about commands, data types, loops, and conditions.

Schedule

Time	Task	Method	Media
10’	welcome, introducing the project	frontal	-
5’	installing the software on student’s devices	in plenum	devices
50’	solving different tasks	in plenum	devices
10’	fill out questionnaire	individual	questionnaire

Concept-learning tasks

<table>
<tr><th colspan="3">Basics</th></tr>
<tr><td>#1</td><td>Commands</td><td>10%</td></tr>
<tr><td colspan="3"><b>Description</b>
A command is a single instruction a program should do. It is mostly a verb followed by a bracket that describes the action (e.g. walk_ahead() or delete_letter()).
<b>Question</b>
What is a valid command in Python?
<b>Answers</b>
- robot.left()
- robot.left
- robot.left{}</td></tr>
<tr><td>#2</td><td>Rob’s Storage</td><td>30%</td></tr>
<tr><td colspan="3"><b>Description</b>
Rob has an integrated memory where sometimes vital information is stored. This memory can be accessed via robot.storage. It can contain strings, lists or numbers.
<b>Question</b>
How can Rob's memory be accessed?
<b>Answers</b>
- robot.storage
- storage
- robot.memory</td></tr>
<tr><td>#3</td><td>Talk with Rob</td><td>55%</td></tr>
<tr><td colspan="3"><b>Description</b>
In many cases it would be very useful that Rob would say something. Therefore, he can use the command print("Hello"). It is also possible to print the content of variables. In this case no quotes within the brackets are needed.
<b>Question</b>
How can Rob print some text?
<b>Answers</b>
- print("Hi, my name is Rob")
- print(Hi, my name is Rob)
- shout("Hi, my name is Rob")</td></tr>
<tr><th colspan="3">Loops</th></tr>
<tr><td>#4</td><td>Data Types</td><td>40%</td></tr>
<tr><td colspan="3"><b>Description</b>
Each variable has a certain data type that describes what kind of value is stored (string, number, list). With the command type() you can get the type of this variable. In addition to variables each field of the game board has also a type that can be asked with the command robot.field_type().
<b>Question</b>
Which command will print the data type of its variable?
<b>Answers</b>
- print(type(var))
- type(var)
- print(var)</td></tr>
<tr><td>#5</td><td>Lists</td><td>45%</td></tr>
<tr><td colspan="3"><b>Description</b>
A list is a collection where many different objects (even with different data types) can be stored. Such a list could store a collection of measurements (temperature, brightness, ...) that Rob made on a planet. In Python the items of a list are stored between square brackets.</td></tr>
</table>

Question

What is a valid list declaration?

Answers

- measurements = [50, 0.3, 6.2, "high"]
- measurements = <50, 0.3, 6.2, "high">
- measurements = {50, 0.3, 6.2, "high"}

#6 Access Lists 55%

Description

The elements of a list can be accessed via its index or can be iterated through a 'for' loop. >> for element in lst: Hint: It is also a good idea to use the for loop to repeat a command (like moving on a playfield). It is also possible to check if a certain element is in a list with the membership operator 'in'. >> if 'a' in lst:

Question

What is the output? lst = ["a", "b", "c"] print("a" in lst)

Answer

- **True**
- a
- “False”

Control Structures

#7 Conditions 60%

Description

With the keyword 'if' it is possible to evalute the logical condition of an expression. If the result of the expression is 'True' the indented block is entered. If the result is 'False' the block will be skipped, or an 'else' block can be used. if 2 == 2: >>print("Numbers are equal!") else: >>print("Numbers are not equal!")

Question

This is an alternative way to formulate conditions. What is the output? print('True') if (2+3 <= 3) else print('False')

Answers

- False
- True
- Equal

#8 Looping 80%

Description

Another important concept to control the flow of a program are loops. With the 'for' loop it is possible to iterate over a given sequence of numbers. For this purpose you could use the 'range' function. for x in range(5) >> print(x)

Question

What is the output of the following loop? for num in range(1,3): print(num)

Answers

- 1 2
- 1 2 3
- num

Practical tasks

Basics

#1 Collect the disk 10%

Task

In the first task simply collect the disk by using the command blocks (arrows) for controlling. Drag and drop them into the editor and Rob will move.

<table>
<tr><td colspan="3"><b>Available Commands</b>
Left, Right, Up, Down</td></tr>
<tr><td>#2</td><td>Hello Rob</td><td>55%</td></tr>
<tr><td colspan="3"><b>Task</b>
Your mission is to reach the disk and let Rob print "Hello Rob". Therefore, you can use the print command block. Hint: If the way seems to be very long, you could probably use a parameter for the moving commands.
<b>Available Commands</b>
Left, Right, Up, Down, Print</td></tr>
<tr><td>#3</td><td>Simple Calculation</td><td>60%</td></tr>
<tr><td colspan="3"><b>Task</b>
Help Rob doing a calculation. He has a value in his storage (robot.storage). Can you help Rob to calculate the 5th power of the stored value? Unfortunately, he is very poor at mental arithmetic.
<b>Robot Storage</b>
18
<b>Available Commands</b>
Left, Right, Up, Down, Print</td></tr>
<tr><td colspan="3"><b>Data Types</b></td></tr>
<tr><td>#4</td><td>Working with Lists</td><td>40%</td></tr>
<tr><td colspan="3"><b>Task</b>
Rob stores a list with all planets that he already observed. Could you check if he has already explored the planet "Melmac"? If he was already there, Rob should print "Yes" , if he wasn’t print "No".
<b>Robot Storage</b>
"Earth", "Mars", "lmc", "X12", "Jupiter", "Grux", "B1252", "Soilyj", "Switq", "Knoedl", "Grz", "Winq", "Ljop9"
<b>Available Commands</b>
Left, Right, Up, Down, For, If, Variables, Print</td></tr>
<tr><td>#5</td><td>Lists and Elements</td><td>50%</td></tr>
<tr><td colspan="3"><b>Task</b>
In Robs memory (robot.storage) is a list with different data types stored. Print each element of this list.
<b>Robot Storage</b>
38.8, "Temperature", "Soil", 12, "Sandy Surface"
<b>Available Commands</b>
Left, Right, Up, Down, For, Variables, Print</td></tr>
<tr><td>#6</td><td>Field Types</td><td>70%</td></tr>
<tr><td colspan="3"><b>Task</b>
On the playfield are so called "hiddenFields". Rob should avoid these fields because when he travels this field, you will lose half of your coins. Sadly, these fields are not visible, so you should probably scan the playfield for them. When you are done reach the disk and print the total number of all appearing "hiddenFields". Hint: You can access the type of a field via the command robot.field_type(x_coordinate, y_coordinate).
<b>Available Commands</b>
Left, Right, Up, Down, For, If, Variables, Print</td></tr>
<tr><td colspan="3"><b>Control Structures</b></td></tr>
<tr><td>#7</td><td>Counting Up</td><td>30%</td></tr>
<tr><td colspan="3"><b>Tasks</b>
Unfortunately, the calculation module of Rob got destroyed during has crash. Can you help him re-write this module so he can count from 0 to 100 in steps of 5 (i.e. 0 5 10 15 ... 95 100) and print this before reaching the disk? Hint: The command range(min, max, steps) takes three parameters.</td></tr>
</table>

Available Commands Left, Right, Up, Down, For, If, Variables, Print		
#8	Tiny Calculation	50%
Tasks Rob made a few temperature measurements and stored all in his internal memory (robot.storage). Can you write a program to calculate the highest temperature in the memory (using control structures) and print this temperature? **Robot Storage** 5,30,8,3,80,64,99,2,30,5,78,98,5,45,6,85,15,90 **Available Commands** Left, Right, Up, Down, For, If, Variables, Print		

8. If yes, which games do you prefer?

	always	sometimes	never
Adventures	○	○	○
Action	○	○	○
Arcade	○	○	○
Board Games	○	○	○
Puzzles	○	○	○
Card Games	○	○	○
Educational Games	○	○	○
Quizzes	○	○	○
Racing Games	○	○	○
Role Play Games	○	○	○
Sports	○	○	○
Strategy Games	○	○	○

9. Have you ever played an educational game on your smartphone?

○ Yes

○ No

10. If yes, which game?

____

11. How do you like the usage of apps in school?

○ This would encourage me.

○ This would not have any impact on my motivation.

○ This would discourage me.

12. Which media do you use for learning?

□ Schoolbook

□ Books

□ Educational Games (without computers)

□ Educational Games (on computers)

□ Social Media (Facebook, Twitter,...)

□ WhatsApp

□ YouTube Other:

□ ______________________________

13. Do you have experience in programming?

○ Yes

○ No

14. If yes, in which programming language?

____

15.

	Strongly Agree	*Agree*	*Undecided*	*Disagree*	*Strongly Disagree*
The game's structure is coherent					
The game's control is easy to use					
The usage of the keyboard was easy					
The language is understandable					
The levels were easy to master					
I learned something while playing the game					
sCool encouraged me to learn more about programming					
The work sheet was easy to solve					

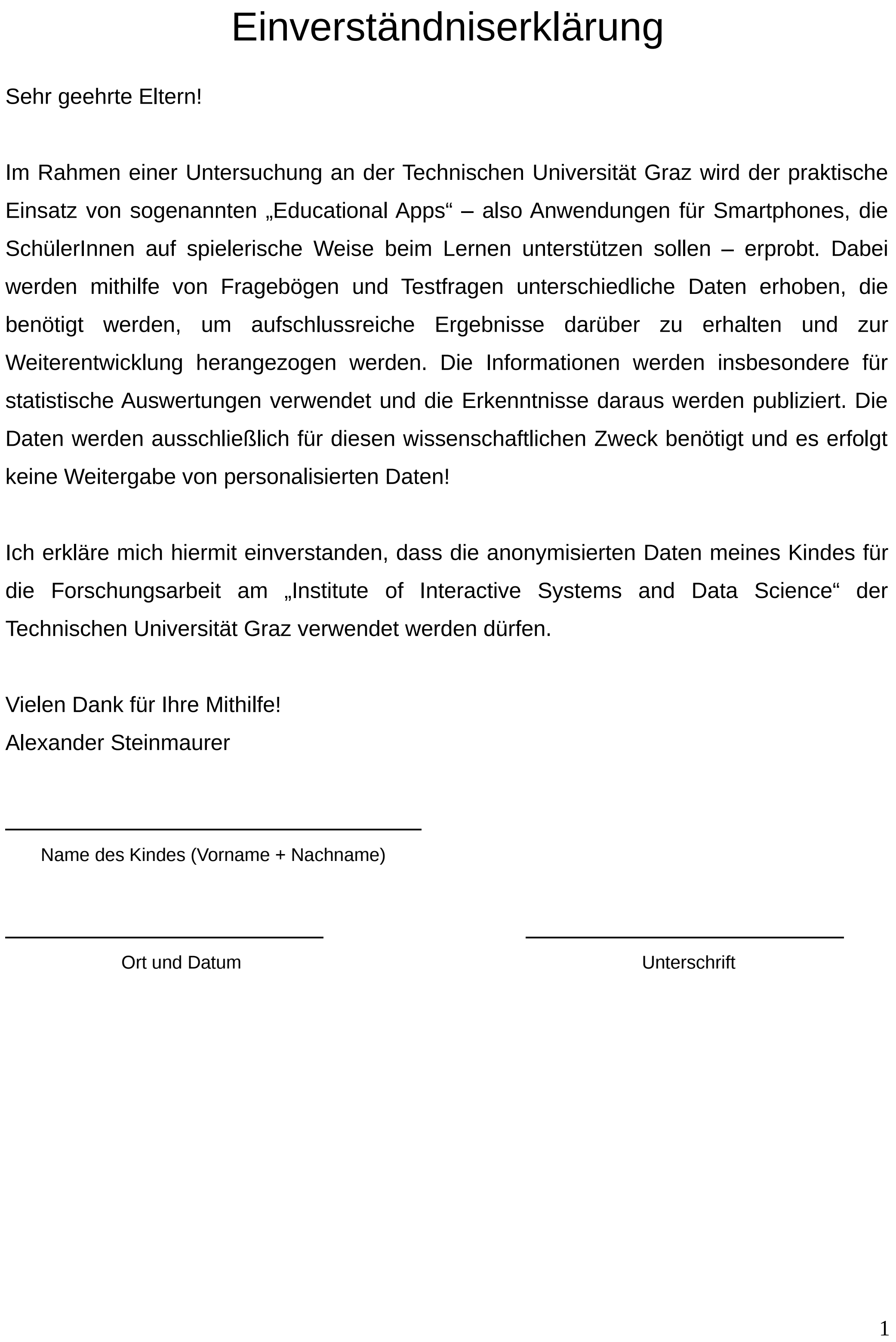
16. Do you have further suggestions for improvement?

17. Questions about sCool

	Strongly Agree	*Agree*	*Undecided*	*Disagree*	*Strongly Disagree*
The type of game is fun					
The character Rob is appealing					
The theme ‘space’ is interesting					
Playing the game was pleasing					
Programming was fun					

18. How could the game get more fun?

A.11 Consent Form



Einverständniserklärung

Sehr geehrte Eltern!

Im Rahmen einer Untersuchung an der Technischen Universität Graz wird der praktische Einsatz von sogenannten „Educational Apps“ – also Anwendungen für Smartphones, die SchülerInnen auf spielerische Weise beim Lernen unterstützen sollen – erprobt. Dabei werden mithilfe von Fragebögen und Testfragen unterschiedliche Daten erhoben, die benötigt werden, um aufschlussreiche Ergebnisse darüber zu erhalten und zur Weiterentwicklung herangezogen werden. Die Informationen werden insbesondere für statistische Auswertungen verwendet und die Erkenntnisse daraus werden publiziert. Die Daten werden ausschließlich für diesen wissenschaftlichen Zweck benötigt und es erfolgt keine Weitergabe von personalisierten Daten!

Ich erkläre mich hiermit einverstanden, dass die anonymisierten Daten meines Kindes für die Forschungsarbeit am „Institute of Interactive Systems and Data Science“ der Technischen Universität Graz verwendet werden dürfen.

Vielen Dank für Ihre Mithilfe!
Alexander Steinmaurer

____

Name des Kindes (Vorname + Nachname)

________________________ ________________________

Ort und Datum Unterschrift