Gotscha! - One Way To Support Learning

Franz Knobel

Bachelor Thesis
January 21, 2020

Supervising Professor:
Prof. Dr. Juraj Hromkovič

Supervisor:
Elizabeta Cavar

ETH zürich

Abstract

The importance of computer science is rising. Games that support the developement of abstract thinking, analytical skills and decision making, are becoming more and more interesting at an early age. Through identification and describing relationships between items, children develop a foundation to early math skills and basic concepts of computer science. The pupils individual learning speed and lack of concentration, if not receiving the right amount of attention, is another challenge by itself. Without individual fostering, children are at high risk of losing interest. Through this thesis the teachers will be introduced to a tool for their pupils. Focused on classification of objects with certain properties, the pupils get introduced to a computer-based learning environment. There they can individually train and improve themselves in this field. In the mean time the teachers can concentrate on the majority of their pupils and have the opportunity to work with them on an individual basis, without feeling the pressure of having to support everyone at once. It has shown, that gamification and game based learning has enormous potential. It takes time to learn how to create games. Once overcome, creating further games is not as hard as one may think. In the created test environment the test subjects have shown more concentration, individual work behaviour and more willingness to learn than in the whole group.

Keywords : phaser 3, typescript, debugging, testing, from a topic to the game, how to tackle a new framework

Acknowledgment

I want to thank my supervisor *Elizabeta Cavar* for supporting me in the last six months. Without her input and constant reminder of our deadlines, I would have had a lot more obstacles on the way.

Also many thanks to *Jaqueline Staub, Nicole Trachsler and Philippe Voinov* for their help with specific topics on my work.

Another very important role in my work played the debuggers, the ones who tested my program and searched for any kind of problems. Thank you for your time, effort and fun you had with my work:

Dilana Kunz, Guido von Burg, Julia Badertscher, Ronja Koeppel, Jonathan Chen

Last but not least, many thanks to my proofreaders, who did not run away from my writing style and my gross negligence with characters and their ambiguous combinations.

ib

Contents

1

Introduction

1.1 Introduction

The importance of computer science is rising. The development of abstract thinking, analytical skills and decision making, are becoming more and more interesting at an early age. Through identification and describing relationships between items, children develop a foundation to early math skills and basic concepts of computer science (e.g. combinatorics of finite affine and projective spaces, the theory of error-correcting codes, hashing, etc.)[22]. The pupils individual learning speed and lack of concentration, if not receiving the right amount of attention, is another challenge by itself. Without individual fostering, children are at high risk of losing interest. Through this thesis the teachers will be introduced to a tool for their pupils. Focused on classification of objects with certain properties, the pupils get introduced to a computer-based learning environment or so called game based learning. There they can individually train and improve themselves in this field. In the mean time the teachers can concentrate on the majority of their pupils and have the opportunity to work with pupils on an individual basis, without feeling the pressure of having to support everyone at once.

1.2 Goals of the Thesis

The main objective of this thesis consists of planning, analyzing, implementing and testing a computer-based learning environment on the topic of classification. The student studies the already existing implementations of "INFORMATIK BIBER in KG und 1/2", analyses the capabilities of kindergarten kids and first graders, develops an interactive classification tool, implements it then on a platform compatible with the implementation of "INFORMATIK BIBER

in KG und 1/2"[28] and conducts an evaluation with test subjects. The implementation is going to be integrated into the existing system of "Einfach Informatik"[4]. The outcome of this thesis is a well-documented, stable and reliable prototype, providing the functional elements to be used in schools.

1.3 Problem Statement

Concerning computer science in schools, new educational material is in production and some is already distributed. With the printed teaching materials, the need for digitalized materials and exercises will come. In related work [2] the already published digitalized exercises and in which way they were realized is addressed. The book *"INFORMATIK BIBER in KG und 1/2"*[28] was published recently and only parts have been digitalized yet. In this thesis the part where children learn how to identify and compare properties of different objects is covered.

The focus though will not solely lie on the translation of the existing teaching material to a computer-based learning environment, but also on introducing learning methods in a gamified environment which not only complements the teachers with their teachings but also assisting them.

The main questions we are going to ask are:

- How can we reach and support our target audience?
- What is our target audience capable of?
- What kind of different digital environments are there?

1.4 Background

The key contribution of computer science to general school education is rooted in the concept of *algorithmic thinking*[23]. One way of introducing kindergarten and primary school pupils to algorithmic thinking and it's concepts consists in making them solve problems with and without computers. This can be achieved using age- and knowledge-appropriate learning materials.

1.5 Motivation

Computer science provides already an easier way than other topics for teaching in the digitalized world. As one can at least guess the huge potential to make use of the students daily enjoyments while still teaching them the necessary things, it would be a loss to the teaching world not to explore and evaluate this kind of teaching method. Especially game based learning as gamified learning is already being explored and evaluated. The aspect of integrating this method as a helper/assistant to the teacher and not only as an additional feature in their free time, is alone very tempting to explore in my opinion.

1.6 Outline

In the following chapters related work in the field of game based learning will be addressed, and the target audience as well as electronic devices will be analyzed for their requirements and capabilities. Furthermore the created learning environment based on game based learning is explained in detail, its evaluation and future applications and improvements.

2

Related Work

In this section we are going through some previous works in a the same area. Starting with mainly publications, in a second part we take look at gamification and game based learning and in a third part we look at existing educational software and highlight their pros and cons to build on the experiences made there.

2.1 Previous work

Several papers with corresponding online learning environments based on the teaching materials "Einfach Informatik" [25, 26, 27] have been proposed by ETH Students [33, 32, 30, 24].

The learning environments they created/extended are mainly based on adding gamification elements to digitalized tasks instead of creating a game based environment. This works to some extent as the texts and exercises can be used from the book.

Their environment is based on TypeScript, the Angular framework and accessible through the homepage **Einfach Informatik**[4].

It contains a lot of information for both students and teachers. Information about the task is mainly obtained via text. Thus one requirement is being able to read or being accompanied by someone who is able to.

Often teacher can upload exercises or create tasks. This can be seen as a disadvantage as then the teacher has to focus on doing something other than care for the majority of his class.

Exercises are sorted by textbook chapter, which is in my opition a valid strategy to use the learning environment as an addition to the textbook. Each topic contains exercises, as well as

exams, where students can solve them and sum up the skills they were supposed to acquire in this chapter. The division of the exercises into different difficulty levels offers each student level adjusted exercises by knowing in advance which exercises require a basic level of understanding and which ones are for the more advanced students. The main goal tries to focus on the engagment of the students interest on the subject which is good.

2.1.1 Angular

Angular is an open-source web application framework based on TypeScript. It is maintained by Google and offers lots of built-in functionalities. It uses HTML and CSS for the view of the application and TypeScript for the model and the controller. As Google is supervised the framework, it is contiguously developed and tested, which makes it a stable Framework giving lots of hands for fast development.

2.2 Gamification and Game Based Learning (GBL)

2.2.1 Gamification

Gamification reflects a social phenomenon arising with a generation of digitally literate population. It has been defined as the use of "game-based mechanics, aesthetics, and game thinking to engage people, motivate action, promote learning, and solve problems"[29]. This includes digital game mechanics, but is not limited to, avatars, badges, points, levels, leaderboard, virtual rewards, and storyline or even quests. There is also an aspect to game elements that allow for social interaction between players and the aquisition of critical thinking skills, which are essential in learning.

Digital video game elements that are used in the pedagogical context promote task engagement, increase motivation, and enforce desirable learning behaviour. The rationale behind deploying video game elements in an educational context is that they have already captured the attention of millions of people all over the world.

2.2.2 Game-Based-Learning

Game based learning describes an approach to teaching, where students explore relevant aspects of games in a learning context. Generally, it is designed to balance subject matter with gameplay and the ability of the player to retain and apply said subject matter to the real world.

Good game-based learning applications can draw the user into virtual environments that look and feel familiar and relevant. Within an effective game-based learning environment, it is common to work towards a goal, choosing actions and experiencing the consequences of those actions along the way. Making mistakes in a risk-free setting, and through experimentation, it is actively learned and practiced the right way to do things. This keeps the user highly engaged

in practicing behaviors and thought processes that he can easily transfer from the simulated environment to the real life[5].

2.2.3 Gamification vs. Game-Based-Learning

While game based learning is similar to gamification, it is a different breed of learning experience. Gamification takes game elements and applies them to a non-game setting. Game based learning is designed to balance subject matter with gameplay and the ability of the player to retain, and apply said subject matter to the real world.

2.3 Existing Educational Software

The idea of propagating games based on game based learning is often touched but rarely integrated in schools. Sometimes they have more in common with normal computer games and the learning aspect is only discovered when specific looked at. En good example of a such company is the learning company[18].

The learning company produced a grade-based system of learning software and tools to improve productivity. It was known for its games like Reader Rabbit[15] and OutNumbered![16].

Reader Rabbit is an educational game franchise and a series aimed at children from infancy to the age of eight.

OutNumbered! is an educational computer game software aimed at children ages seven to fourteen and is designed to teach children mathematical computation and problem solving skills.

Both games have a storyline with a specific designed environment which includes minigames and challenges covering various topics not only focusing on one subject.

The games teach language arts including basic skills in reading and spelling, and mathematics. They are designed to be played alone and not as an addition to the lessons at school but as additional homework or recreational activities over the holidays to not forget the freshly learned subjects.

It had great success depending on the perspective, as the kids were sitting at the computer over the holidays in summer and "learning" instead of playing outside. Kids see such games as what the games are trying to be, games without noticing the learning aspect. Though, it is difficult to create one game for a broad spectrum of users, as the level of the topics vary from country to county or even town to town. This becomes even more complicated if multiple topics from different school subjects are included in one game. If a part of a game is too easy or too hard for the user you are risking loosing his attention fast, depending on his the age.

In my opinion the rise and fall of those games was on one hand the integrated learning process in a story and minigames which were very satisfying to the normal user, as he did not notice that he was learning. Unfortunately, the games were not supporting the teacher in class and only supplementary. Why should I play a game related to school when there are multiple other games with better gameplay, a more interesting storyline and better visual effects? Yes, it is nearly

impossible to compete with the game industry if you are not , but if you cannot compete, change the playfield. The playfield normal games will never touch is the school, as they normally have no correlation to school topics.

3

Requirements

In this section we will capture the requirements for our learning environment. We will look into the cognitive and motor abilities of the target audience and some elements of gamification which will be analyzed and assessed. The existing environment, our work should be implemented in, has conditions as well. Those will be taken into consideration while evaluating a suitable framework.

3.1 Programming Environment

3.1.1 Boundary Conditions

The following conditions were extracted from the assignment:

- The game should be maintainable as it is not excluded that others are going to work on it.
- Modularity is an important aspect as one cannot assume the final design at this point in time.
- It should be possible to integrate it in a Typescript/JavaScript/HTML environment (e.g. as a canvas) as all similar work up until now is mostly written in Typescript in the Angular framework.
- The ability to run on computers and tablets is important as a lot of schools have at least one.
- The game should be self-explainable as the teacher has to keep his focus on the majority of his students.

- Easy to learn. Support is an important aspect as it is easier to start with something new. A huge community correlates in most cases with good support on free open source software.

3.1.2 Evaluation of Possible Solutions

Babylon.js

Babylon.js is a complete JavaScript framework for building 3D games. Using WebGL for graphics, the feature set for Babylon is somewhat extensive. Its community is said to be fairly active. The most significant features worth mentioning are:

- Support and exporting tools
- Game engine staples such as scene picking, collision handling, and scene graphs
- Particle and animation systems
- Performance optimizations such as frustum clipping, hardware scaling, and occlusion queries
- Shader, rendering, and texture systems
- Expansive mesh support

Babylon does not require to be installed as an internal entity on your computer. Thus, all development can happen within the browser/code editor itself[2].

Pixi.js

Pixi.js is a 2D game rendering engine intended for HTML5 games. There are benefits of its integrated hardware acceleration. Pixi's focus lies not on WebGL, yet utilizes rich game content, interactive displays, and apps that are supported on all platforms equally. It is said that it is the way that Pixi has been built that enables for it to be a smooth, rapid, and evenly interactive rendering engine[2].

Melon.js

Melon.js has a sprite-built JavaScript engine for 2D game development, is an independent project which does not require additional libraries to work, supports mobile type devices as well as all leading browsers, optimization for mobile devices for motion and hardware, in-built HTML5 audio support, a practical physics engine to reduce the CPU usage, a great deal of effects that would be required for creating a functional on-line game in the browser. Community forums is hosted on Google Groups, where you can quickly yield answers to your questions in regards to how Melon.js works or in the case of you experiencing bugs. The documentation features several dozens of demo applications built with Melon, some of which are open-source and can be used to learn different aspects of game development from[2].

Phaser

Phaser is a free and open source JavaScript/TypeScript framework which puts a focus on letting coders make games quickly. The system works primarily with Canvas and WebGL, letting programmers easily build substantial games for both desktop and mobile browsers.

It is said that phaser has an active community that regularly participates on the forum, Slack, and Discord channels.

One of the biggest benefits of this engine is that it is a fully-featured engine, so it isn't restricted to doing just one thing. Here's a list of some of the feature sets provided with Phaser:

- Built on WebGL and canvas
- Preloader system
- Physics features
- Sprite and animation handling
- Particle system
- Camera, input, and sound systems
- Tilemap support
- Device scaling support
- Plugin availability

Phaser's preloader makes it easy for developers to load their game assets and have them automatically handled. That way, one des not have to waste time writing extensive code for each part of the game[2].

Evaluation of the Frameworks

Considering the boundary conditions one can see that the TypeScript support, a lot of code examples, an active community for support and multiple browser support is most important for this work. It is trivial to see that the phaser framework fulfils these conditions to a fairly good extent.

3.2 Boundary Conditions of the Problem Statement

Considering the problem statement the user has to learn how to identify and compare properties of objects. As these object should just be placeholders, the simplest objects can be used for that purpose (e.g. geometrical objects). To make the task of comparing more difficult the objects can have more than one property. The user should train the task at hand on different levels of difficulty. There should be easy levels for inexperienced users and hard levels for more advanced users. The aspect of hashing in the context of sorting these objects with limited available space must be included.

3.3 Boundary Condition of the Target Audience

The target audience are going to be children between the age of 5 and 8. They cannot be expected to be able to read or write. Numbers from 1 to 5 should be possible though. The motoric abilities of children with tablets are rather advanced in contrast to useing a computer with a mouse. So a touch screen of any kind is not a problem but the handling with a keyboard of a mouse cannot be assumed.

3.4 Additional Conditions: Elements of Gamification

To enhance the user experience and to influence his behaviour the following elements of gamification should be implemented:

- Different levels for different experienced users.
- A reward system for instant gratification.
- Global progress tracking to keep track of your success.
- The gratification should be visualized and animated as this gives the user more satisfaction
- Tracking of correct and incorrect choices made to have a live tracking, which ads motivation and stress on the same time (higher difficulty).
- Time limits to put the user under stress.
- Sound can add a valuable replay factor.
- If possible there should be an aspect of a story/timeline. This has the potential to captivate the user on a whole different level.
- A level menu or a map is useful addition or replacement for a story.
- A fullscreen option helps to stay focused and not get distracted by other visuals from your environment
- A pause menu is important for the user as he can always be distracted. Without a pause menu the user may get frustrated and fed up with the game.

4

Design of the Learning Environment

In this section all implementation tools and approaches are explained. First the used framework will be explained, then how the different scenes and objects are generated and at last how these two are combined to the final solution: Gotscha! A learning environment based on game based learning. The user goes through multiple levels to learn detecting and comparing properties of objects under simple and more difficult situations.

4.1 Phaser 3

Phaser[12] is an open source HTML5[8] game framework created by Photon Storm[12]. It is a JavaScript/TypeScript[19] library designed to run on all major desktop browsers. A lot of focus was given to the performance inside of mobile web browsers. Important for the renderer is that if the device is capable, it uses WebGL[21], otherwise it seamlessly reverts to Canvas[3]. The current version is 3.17 and is used in this work. Version 4 is in development.

4.1.1 Base Configuration

In Phaser 3, games need a configuration file and a starting point [4.1].

Listing 4.1: *Game Setup File*

```
1    import 'phaser';
2    import GameConfig = Phaser.Types.Core.GameConfig;
3    import RenderConfig = Phaser.Types.Core.RenderConfig;
4    import {Scene1} from './scene1';
5    import {Scene2} from './scene2';
```

```
6
7       // Defining the renderer
8       const renderConfig: RenderConfig = {
9           antialias: true,
10          pixelArt: false
11      };
12
13      // Enforcing widescreen
14      let width: number = window.screen.width;
15      let height: number = window.screen.height;
16
17      if (window.screen.width <= window.screen.height) {
18          width = window.screen.height;
19          height = window.screen.width;
20      }
21
22      // Game Configuration
23      const config: GameConfig = {
24          title: 'TITLE',
25          parent: 'game',
26          type: Phaser.AUTO,
27          scene: [
28              scene1, scene2
29          ],
30          physics: {
31              default: 'arcade',
32              arcade: {
33                  debug: false
34              }
35          },
36
37          // Master background color
38          backgroundColor: '#000000',
39
40          render: renderConfig,
41
42          scale: {
43              mode: Phaser.Scale.FIT,
44              autoCenter: Phaser.Scale.CENTER_BOTH,
45              width: width,
46              height: height
47          }
48      };
49
50      export class GameName extends Phaser.Game {
51          constructor(config: GameConfig) {
52              super(config);
53          }
54      }
55
56      // Event handler for starting the game (starting point)
57      window.onload = () => {
58          const game = new GameName(config);
59      };
```

4.1.2 Scenes

Games in Phaser 3 are structured around scene objects. A scene is a collection of game objects and related logic because these two should be kept together. The objects will be drawn when the scene is rendered.

Where this gets special is that Phaser doesn't place any constraints on how many scenes need to be running. This means you can have 0, 1, or as many as you need running at once. You can communicate between them and each scene has a depth (z-index). With the z-index a UI scene, rendered above the play scene, which is always rendered above the background scene, is possible.

Lifecycle

A simplified model has a scene that moves between four states: *Create, Update Loop, Paused, and Stopped*. Transitions are initiated by a function call and emit a signal that can be listened for. This way, you can take action at specific point in the process.

The scene state transitions and fired events are summarized in the following state diagram. Functions which initiate a transition are in yellow and signals emitted are orange.

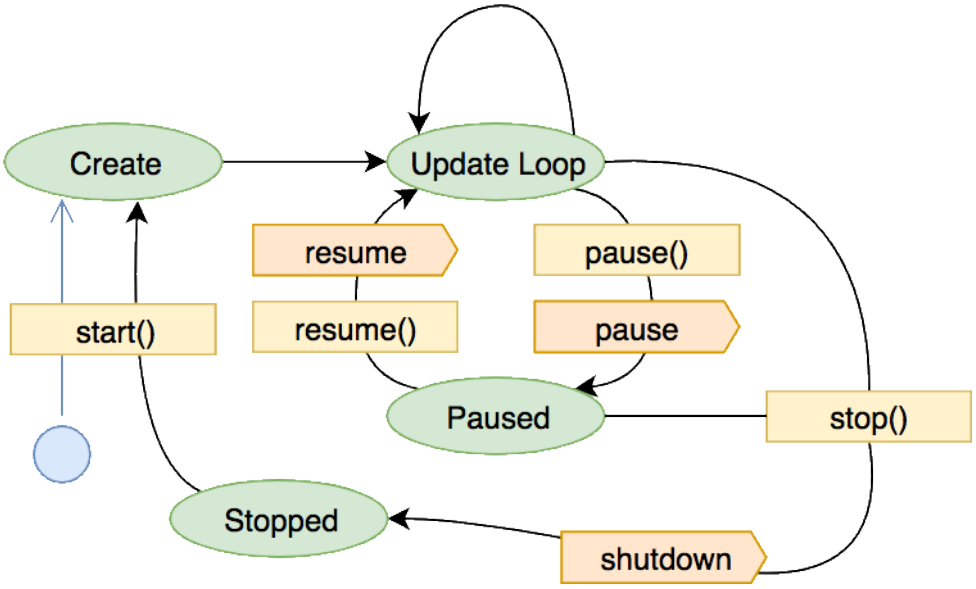

Figure 4.1: *Scene Lifecycle[14]*

An interesting behavior is that once a scene has been shut down it is not garbage collected. The scene can always be resumed by the start() method. When this happens the three creation functions get called once more. This means any state tracked in the scene class will be retained between a stopped state and the next create state. Thus one has to be careful with setting a known initial value on everything requiring one. Especially when loading/creating objects outside the preloader (animations, tweens, audiofiles, etc.).

Scenes here have 5 functions:

- **constructor()**: Run 1 time.
- **init()**: Run 1 time. Initialization of fields and passed on data by other scenes.
- **preload()**: Run 1 time. Loads up all the assets inside the scene like images and audio.
- **create()**: Run 1 time. Position and display the assets that are already preloaded, animations, physics, etc.
- **update()**: Run 1 time per frame, takes care of everything related to the game logic.

4.1.3 Managers

In Phaser 3 managers are a global system. Animations, scenes, images loaded/created within it are globally available to all game objects. They share the base data while managing their own timelines. This allows the definition of a single object once and its application to as many game objects as required. So a game object can be called in an completely other and unrelated scene. Examples for used managers in this work are:

- **Scene Manager** (scene.scenes). Contains all scenes of the game once created.
- **Texture Manager** (scene.textures). Contains all textures once loaded.
- **Audio Manager** (scene.sound). Contains all audio files once loaded.
- **Data Manager** (scene.data). Shared data manager. An event (listenable) is triggered when an event is stored/changed.
- **Cache Manager** (scene.cache). Contains all special files once loaded.
- **Animation Manager** (scene.anims). Contains all animations once created.
- **Tween Manager** (scene.tweens). Contains all tween objects once created. A Tween is able to manipulate the properties of one or more objects to any given value, based on a duration and type of ease.

4.2 Objects

4.2.1 Object Generation

Our objects can have up to four properties with exactly one from each of the following categories:

- **Geometrical shape** (square, triangle, circle, ellipse, rhombus, octagon)
- **Color** (yellow, orange, red, purple, green, blue)
- **Holes** or dots (one, two, three, four, five, six)
- **Filling** (filled, striped, dotted)

All possible objects with one, three and four properties, are needed.

With a python script [4.2] scalable vector graphic (SVG)[17] files are generated [4.3]. After that they are converted to portable network graphic (PNG) files with the GNU image manipulation program (GIMP)[6]. Additional image scaling and cropping is done for saving space and only displaying the actual image with as less empty space as possible around it. With the imagemagick command *mogrify* [4.4] the last step is easily applicable to all 1000+ images.

Listing 4.2: *SVG Generation*

```
...
imageString = imgStr(colordefault, colordark, square, circle,
    triangle, ellipse, octagon, rhombus, filling, one,
two, three, four, five, six)

filename = colordefault + shape + number + fillingname

with open("images/svg/" + filename + ".svg", "w+") as file:
file.write(imageString)
file.close()
...
```

Listing 4.3: *SVG Image File*

```
<?xml version="1.0" standalone="yes"?>

<svg height="1000" width="1000" viewbox="0 0 1000 1000" xmlns="http
    ://www.w3.org/2000/svg">
<defs>
<pattern id="stripe" patternUnits="userSpaceOnUse" width="20%"
    height="20%">
<path stroke="aqua" stroke-linecap="butt" stroke-width="50" d="M -20
     -20 l1000 1000"/>
...

</pattern>
<pattern id="dotted" enable-background="true" patternUnits="
    userSpaceOnUse" width="15%" height="15%">
<circle cx="30" cy="30" r="25" fill="aqua" />
...
</pattern>
<style>
.button {

stroke-width:5;
stroke:black;

}
</style>
</defs>

<g id="circle" display="none">
<circle cx="500" cy="500" r="300" class="button" fill="blue"/>
...
</g>

```

```
29      <g id="square" display="inherit">
30      <rect x="250" y="250" rx="20" ry="20" width="500" height="500" class
            ="button" fill="blue"/>
31      ...
32      </g>
33
34      <g id="triangle" display="none">
35      <polygon points="500,50 113.4,700 886.6,700" stroke-linejoin="round"
             class="button" fill="blue" />
36      ...
37      </g>
38
39      <g id="ellipse" display="none">
40      <ellipse cx="500" cy="500" rx="400" ry="250" class="button" fill="
            blue" />
41      ...
42      </g>
43      <g id="octagon" display="none">
44      <polygon points="400,250 600,250 750,400 750,600 600,750 400,750
            250,600 250,400" stroke-linejoin="round" class="button" fill="
            blue" />
45      ...
46      </g>
47
48      <g id="rhombus" display="none">
49      <polygon points="350,250 150,750 650,750 850,250" stroke-linejoin="
            round" class="button" fill="blue" />
50      ...
51      </g>
52
53
54      <g id="one" display="none">
55      <circle cx="500" cy="500" r="40" stroke="black" fill="black"/>
56      </g>
57
58      <g id="two" display="none">
59      <circle cx="570" cy="500" r="40" stroke="black" fill="black"/>
60      ...
61      </g>
62
63      <g id="three" display="none">
64      <circle cx="570" cy="560" r="40" stroke="black" fill="black"/>
65      ...
66      </g>
67
68      <g id="four" display="none">
69      <circle cx="570" cy="430" r="40" stroke="black" fill="black"/>
70      ...
71      </g>
72      <g id="five" display="none">
73      <circle cx="570" cy="430" r="40" stroke="black" fill="black"/>
74      ...
75      </g>
76
77      <g id="six" display="none">
78      <circle cx="570" cy="400" r="40" stroke="black" fill="black"/>
```

```
79      ...
80      </g>
81
82      Sorry, your browser does not support inline SVG.
83      </svg>
```

Listing 4.4: *ImageMagick console command "mogrify"*

```
1       mogrify -resize 50\% -trim -repage *.png
```

4.2.2 Object Storage

Our objects are images with different properties. These properties cannot be saved in the images itself. for that reason the image path with the respective properties are stored in a easily accessible "JavaScript Object Notation" (JSON) [4.5] file.

As the game should be just a template for further graphical enhancements the JSON File can be adapted in a simple way to other categories and images.

IMPORTANT: Objects in this game have three to six properties per category. This is just an example. Categories can have more than six properties but should not have less than three.

Listing 4.5: *JavaScript Object Notation File (geometrical_objects.json)*

```
1  {
2    "categories": [
3      {
4        "name": "cat1",
5        "url": "color.png",
6        "validElements": [
7          {
8            "name": "purple",
9            "urls": [
10             "purple1.png",
11             "purple2.png",
12             ...
13           ]
14         },
15         ...
16       ]
17     },
18     ...
19   ],
20   "images": [
21     {
22       "name": "purplesquareonefull.png",
23       "cat1": "purple",
24       "cat2": "square",
25       "cat3": "one",
26       "cat4": "full"
27     },
28     ...
29   ]
```

```
30 }
```

4.2.3 Object Display

Displaying the same set of objects or just the image of an object without tweaking it a little can seem boring for the user in long term. Thus in every game and level the set of objects is randomly selected and playing the same game or even level keeps being visually interesting for a longer time. To add even more diversity, objects gets a random rotation angle, size and position every time it is displayed/created. Of course within certain predefined boundaries. Those boundaries take into account that the minimum size should always be enough to touch it with a normal sized finger.

4.2.4 Object Interaction

Whenever possible a generalization of the input code [4.6] is used with every user-object interaction. Instead of giving each object an event task, the global event task is fetched and the object via the event parameters. This way the input interaction code is kept in one place and so less fragmented.

Listing 4.6: *Input code*

```
1     ...
2     this.input.on('dragstart', function(pointer, gameObject) {
3         ...
4     }, this);
5
6     this.input.on('drag', function(pointer, gameObject, dragX, dragY) {
7         ...
8     }, this);
9
10    this.input.on('dragend', function(pointer, gameObject, dropped) {
11        ...
12        if (!dropped ...) {
13            ...
14        }
15    }, this);
16
17    this.input.on('drop', function(pointer, gameObject, dropZone) {
18        ...
19    }, this);
20    ...
```

4.2.5 Modularity

To ensure the modularity and adaptability of this work for other designs, the objects with the property file as well as the background is completely replaceable by other images. The sub-

section "Object Storage" [4.2.2] explains the most difficult part of adapting, namely writing the property file. As a template the current property file should be used.

4.3 Score Storage

The achieved score is saved in the local storage[9] of the browser. The local storage remains unchanged until it is cleared in the browser settings. Thus the saved score is not lost on the same device until purposely deleted. Before accessing the local storage it is checked if there even is a storage [**??**].

Listing 4.7: *Storage access (scoreScene.ts)*

```
1     ...
2     private saveScore(score: string): void {
3         if (typeof(Storage) !== "undefined") {
4             window.localStorage.setItem('phaser_score_' + this.
                  previousScene, score);
5         } else {
6             console.log("Sorry! No Web Storage support...");
7         }
8     }
9     ...
```

Listing 4.8: *Storage access (levelMenuScene.ts)*

```
1     ...
2     if (typeof(Storage) !== "undefined") {
3         if(window.localStorage.getItem('phaser_score_' + this.
              buttonToSceneMap(gameObject.name))){
4             if (reset) {
5                 window.localStorage.setItem('phaser_score_' + this.
                      buttonToSceneMap(gameObject.name), score);
6             } else {
7                 score = window.localStorage.getItem('phaser_score_' +
                      this.buttonToSceneMap(gameObject.name));
8             }
9         }
10    } else {
11        console.log("Sorry! No Web Storage support...");
12    }
13    ...
```

4.4 Creating animated Introductions

To generate animated instructions for the task later on, the screen is recorded with the Open Broadcaster Software (OBS) Studio[10] while someone is playing the game. Phaser does not support video playback. Thus parts of the final video are then taken for the respective scenes and converted into single pictures. These single pictures are saved in one picture (called spreadsheet [4.2]). Those spreadsheets can then be animated with Phaser.

Is is important to note, that different devices have a different limit for the resolution of images. Tablets seem to have the lowest, namely 4096x4096 pixels[20]. The spreadsheet images are scaled down for that reason.

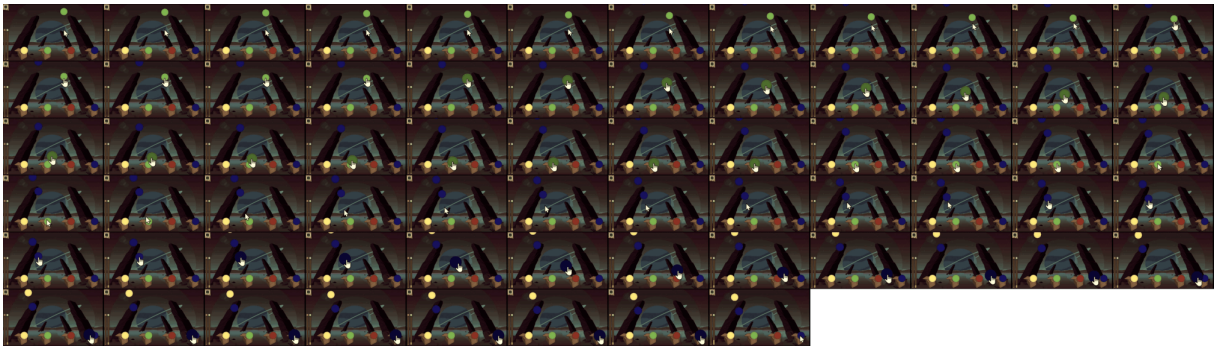

Figure 4.2: *Example: Spreadsheet*

4.5 Scene Concept of the Learning Environment

Through the given boundary conditions and requirements the environment is split into multiple parts/scenes.

4.5.1 Drop Down Menu

Throughout the whole experience, the user can open a menu with a button. This button is always visible. Once the menu is open, the user may close the menu and return to the current scene, exit the current scene and return to the level menu and go into fullscreen and back. The current scene is paused while the menu is open as it can be distracting for the user, if suddenly something pops up and covers parts of the visible and running scene. With blurring out the current scene the user is made aware of the current scene being paused and of the "open menu" state.

4.5.2 Welcome Screen

The welcome screen is the starting point of the user experience. Through this screen the user is greeted by showing the name of the game. Through a click he may commence to the level menu. To make the art of transition clear to the user a finger icon is displayed and animated.

4.5.3 Level Menu

In the level menu, the user is able to choose between different levels and games and can access the object summary. Through the stars below each button the user can track his progress/score. The progress can be resetted by the reset button.

4.5.4 Object Summary

The Object summary allows the user to get a feeling for all the different properties an object can have. The number of objects per category is restricted to five, as there are over 1000 different objects and with all of them the user would not be able to focus on the different properties. Objects are draggable and sortable by each category with a click on a respective button.

4.5.5 Sorting with one Category

Here the user has to sort the static objects with one category by the given category. It is important for users, inexperienced with sorting objects by their properties, to start at the lowest level possible. As the user should be able to experience all categories, they are split into different levels. Each level represents a category.

For motivational purposes, the user can track his progress. The Progress is defined by objects sorted the right and the wrong way.

The amount of objects to sort, as well as the amount of properties of the category is randomly selected each time you start the game.

4.5.6 Sorting with one Category under difficult Conditions

It is important to internalize learned skills under difficult conditions. This turns mental processes into automatisms, which means to execute processes correctly without thinking.

So in this game the user has to sort falling objects with one category by the given category. As the user should be able to experience all categories, they are split into different levels. Each level represents a category.

For motivational purposes, the user can track his progress. The progress is defined by objects missed, sorted the right and the wrong way. To make the task harder, objects get instead of a randomly selected rotation angle a randomly selected spin velocity. Dummy objects are added as well. Those objects look similar to the original ones but with a succinct characteristic. There are no negative points for missing such an object but negative points for sorting them in any way.

For more diversity, the amount of falling objects to sort, as well as the amount of properties of the category is randomly selected each time you start the game.

4.5.7 Sorting with restricted space

This game scene is specific designed for the users preparation to the computer science topic hashing or hash functions.

A hash function is any function that can be used to map data of arbitrary size to fixed-size values. The values returned by a hash function are called hash values, hash codes, or simply

hashes and are used to index a fixed-size table called a hash table. The use of a hash function to index a hash table is called hashing or scatter storage addressing.[31]

Broadly summarized, a hash function converts a given sequence of objects (e.g. text, symbols, etc.) with a certain length to a fixed sequence with always the same length within the same hash function. This function is not invertible.

This scene is an strong abstraction of a hash function. The user has to sort a given number of objects, with all properties shown, into boxes with limited space.

The objects in one box must have at least one property in common and can be put into the box and taken out an infinite amount of times.

To make the game more difficult, this level is split into two. The first level has boxes with the size 6, 4, 2 and the second one boxes with the size 6, 5, 4.

4.5.8 Object pairing

This game scene trains the user in the comparison and grouping of objects. In the process he must identify and isolate properties of different objects, compare them and remember the outcome. To finally compare objects and reach a conclusion if the objects can be grouped or not, the user has to remember multiple outcomes.

This lays a foundation of a rich mathematical structure linking it to the combinatorics of finite affine and projective spaces and the theory of error-correcting codes.[22]

The game is split into two versions. The easy one for adapting to the correct thinking and the normal one to strengthen it.

There are twelve objects being displayed. The user has to select three objects which have to fulfil the following rules.

For each category one of the following conditions has to hold:

- They must be the same (blue, blue, blue)
- They must be completely different (square, triangle, circle)

If the user needs help, there is a helper bar which can be accessed by a button with a question mark on it. The helper bar shows which categories fulfil the conditions and which do not by coloring the category symbols on the bar in green or red.

The game is time limited. The remaining time is shown by a bar. If you select three objects which fulfil the conditions, more time will be added. After a set amount of correct selected objects, the game will end.

If there are no three objects that fulfil the conditions, the playfield will be generated anew.

Object pairing - easy version

In the easy version objects have three categories: *color, shape and number of holes/dots*

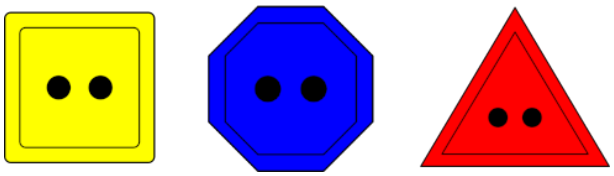

Figure 4.3: *Example: Matching Set Easy*

Object pairing - hard version

In the hard version objects have four categories: *color, shape, number of holes/dots and filling*

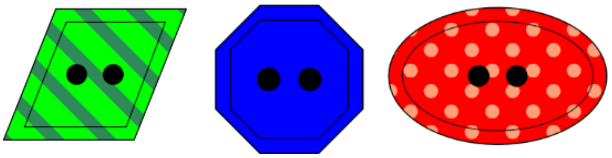

Figure 4.4: *Example: Matching Set Hard*

4.5.9 Score Screen

After the completion of a task/level/game, the user will be granted a score. The score is represented here with a displayed number of stars. The minimum of stars is zero and the maximum is three. If the user is unhappy with his results, he can replay the game by clicking on the replay button. To return back to the level menu the user has to click anywhere on the screen (excluding buttons). To make this action clear, a clicking finger is being displayed.

4.5.10 Introduction

As the task beforehand for each level may not be clear to every user an introduction is necessary. Before each game starts, an animation [4.4] of the task beforehand is being shown. This scene serves as a helper scene so that the running scene can be paused while the animated introduction runs. If the current scene is not paused in the meantime, the user may not take enough time to understand the task.

4.6 Code structure of the learning environment

Our learning environment consists of different scenes each one inherits the base scene. The following subsections contain a summary of crucial and not trivial parts of the code.

The full code can be accessed in the appendix [**??**] or on gitlab[1].

Figure 4.5: *Scene State Diagram*

4.6.1 Playing the Game

The game can be played here[7].

4.6.2 Running the Code

To run the code clone the gitlab repository[1] and run the following commands in the terminal in the cloned folder.

Listing 4.9: *BaseScene.ts*

```
1    npm run start
```

The game should now be playable on **"localhost:8080"** in a web browser of your choosing.

Please note that the gameplay video is not included in the repository.

4.6.3 BaseScene

This scene contains methods and fields which are necessary in multiple scenes (e.g. the scene transition, identifier of the scene, ...).

The Scene Transition

Transitions are simply made by laying a geometrical mask over an game object and animate the mask. As masks have to be applied to every object they have to mask, a simple solution was to lay a black rectangle over the whole scene and mask it. So the scene transition has the form of a circle cut out of a black rectangle [4.10]. The animation comes to life with increasing or decreasing the radius of the circle according to the wanted animation (IN/OUT).

Listing 4.10: *Transition Initialization Method (BaseScene.ts)*

```
1    ...
2    private transitionInit(): void {
3        // Shape of the graphical transition
4        const circle: Phaser.GameObjects.Graphics = this.add.graphics();
5
6        // Shape of the screen
7        const rectangle: Phaser.GameObjects.Rectangle = this.add.
             rectangle(0, 0, this.cameras.main.width, this.cameras.main.
             height, 0x000000);
8
```

```
9           // Define circle as the mask
10          const mask: Phaser.Display.Masks.GeometryMask = circle.
                createGeometryMask();
11
12          circle.setPosition(this.cameras.main.width / 2, this.cameras.
                main.height / 2);
13          circle.fillCircle(0, 0, 0.1);
14          circle.setDepth(0);
15
16          mask.setInvertAlpha(true);
17
18          rectangle.setDepth(1);
19          rectangle.setOrigin(0, 0);
20          rectangle.setMask(mask);
21
22          circle.fillCircle(0, 0, 0.1);
23
24          this.transition = [circle, rectangle];
25      }
26      ...
```

4.6.4 PreloadAssets

In the "preload()" method the files used later on can be loaded into the respective managers/-cache. Now every asset loaded this way has a unique identifier and can be used in future scene as long as it is not removed explicitly. The asset can as such be loaded into the managers/cache in every scene, so that only the actual used assets are loaded. The advantage of this is that you can save time and storage. But the disadvantage is that you have to be connected to the internet the whole time. Should the connection be severed for only a short time, objects might not be displayed correctly. With a "preloadAsset" scene in the beginning, all available assets are loaded, so that after the longer loading time the game can run fluently, without problems and most importantly without a connection to the internet.

The attributes and path of objects in the imported json file [4.2.1] can be accessed by filename.["fieldname"] or filename.fieldname [4.11].

Listing 4.11: *Example json file access*

```
1       ...
2       for (let category of loadedJsonObjectFile['categories']) {
3           console.log("URL: " + category['url']);
4           console.log("Name: " + category['validElements']['name']);
5           console.log("ValidElement urls: " + category['validElements']['
                urls']);
6       }
7
8       for (let image of loadedJsonObjectFile['images']) {
9           console.log("Name: " + image.name);
10          console.log("Cat1: " + image.cat1);
11          ...
12      }
13      ...
```

The way the objects files are preloaded into the respective managers is shown below [4.12]. It is important to note that if assets are loaded outside of the preload() method, the loader has to be started manually.

Listing 4.12: *preloadAsset.ts*

```
...
private preload(): void {
    this.load.setPath('assets/geometrical_objects/');
    this.load.json('objects', 'geometrical_objects.json');
    ...
}
...
private create(): void {
    ...
    this.preLoadImages();
    ...
    this.start();
}
...
private preLoadImages(): void {
    // Load category and object images
    const jsonObject: any = this.cache.json.get('objects');

    for (let category of jsonObject['categories']) {
        this.load.setPath('assets/geometrical_objects/categories/');
        this.load.image(category['name'], category['url']);

        this.load.setPath('assets/geometrical_objects/images/');
        for (let property of category['validElements']) {
            for (let url of property['urls']) {
                this.load.image(url, url);
            }
        }
    }

    for (let image of jsonObject['images']) {
        this.load.image(image['name'], image['name']);
    }
    ...
}
...
private start(): void {
    ...
    this.load.start();
}
...
```

Image Game Objects

Instead of representing an image as a normal image in game, sprites, a special texture, is used. But what is a sprite? A sprite is a game object which can display both static and animated images in your game. The big main difference between a sprite and an image game object is

that you cannot animate images. Additionally, sprites have input events, additional functions, fields and physics bodies.

4.6.5 DropDownMenu

In this scene the drop down menu is created. The scene is never stopped and is always on top of other scenes.

The following code [4.13] in every other scene ensures this:

Listing 4.13: *Send current scene to back*

```
1    ...
2    create(): void {
3        this.game.scene.sendToBack(this.getKey());
4        ...
5    }
6    ...
```

As the drop down animation takes time, it was necessary to create a boolean field as a lock. This ensures that the closing and opening animation won't interfere with each other. The lock is freed after the completion of an event/animation.

Listing 4.14: *Lock aquiring and freeing*

```
1    ...
2    if (!this.lock) {
3        this.lock = true;
4        ...
5    }
6    ...
7    onComplete: () => this.lock = false;
8    ...
```

While the menu is down/open, the current scene has to be paused. This is achieved by accessing the other scene by fetching the key of the only other active scene and pausing it. Important to note is, that the last started/activated scene is at position 0 in the array of all active scenes.

Listing 4.15: *Fetching current active scene*

```
1    ...
2    const key_paused_scene: string = this.game.scene.getScenes(true)[0].
         key;
3    this.game.scene.pause(key_paused_scene);
4    this.key_paused_scene = key_paused_scene;
5    ...
```

The same trick is used for closing all current scenes when exiting (without the dropDownMenu Scene).

4.6.6 LevelMenuSceneScene

Levels with the same difficulty are distinguished by numbers from one to four. Those with another difficulty are distinguished by images of monsters with different level of spookiness [**??**].

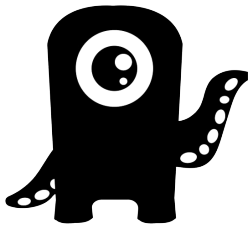

Figure 4.6: *Easy Level Icon*

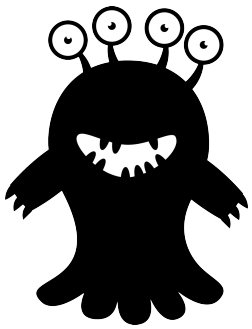

Figure 4.7: *Medium Level Icon*

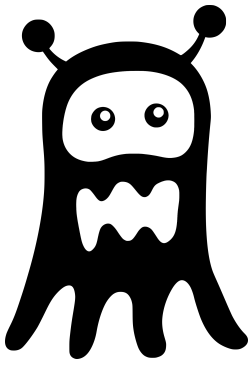

Figure 4.8: *Hard Level Icon*

As there are multiple buttons in this scene it is important to simulate a specific button behaviour. A click is split into "pointer up" and "pointer down" and thus can vary in its combinations on where those events happen.

1. pointer down and pointer up in the same button area
2. pointer down and pointer up in different button areas

3. pointer down and pointer up where only pointer down is in a button area
4. pointer down and pointer up where only pointer up is in a button area

To make sure that the behaviour is bug free, on pointer down on a button, this respective button is marked [**??**]. Now, only marked buttons can react to pointer up events. And upon any pointer up event all markers are removed.

Listing 4.16: *Button marking (levelMenuScene.ts), label=lst:buttonmarking*

```
1     ...
2     private initInput(): void {
3         this.input.on('pointerdown', function(pointer, currentlyOver) {
4             const gameObject: any = currentlyOver[0];
5             if (gameObject instanceof Phaser.GameObjects.Sprite) {
6                 ...
7                 gameObject.setData('clicked', true);
8             }
9         }, this);
10
11        this.input.on('pointerup', function(pointer, currentlyOver) {
12            const gameObject: any = currentlyOver[0];
13            if (gameObject instanceof Phaser.GameObjects.Sprite &&
                  gameObject.getData('clicked')) {
14                this.buttonFunction(gameObject);
15            }
16
17            this.levelButtons.getChildren().forEach(function(gameObject)
                  {
18                if (gameObject instanceof Phaser.GameObjects.Sprite) {
19                    ...
20                    gameObject.setData('clicked', false);
21                }
22            }, this);
23        }, this);
24    }
25    ...
```

4.6.7 IntroScene

To pause the current scene and still play the animated introduction, a separate scene is necessary. Each time an intro has to be played, the intro scene is started anew. The current scene is pauses itself when the intro scene is started and the name/key/identifier of the paused scene is given to the intro scene. That way the intro scene can resume the paused scene and then stop itself.

The respective intro material is selected via the name/key/identifier of the paused scene.

4.6.8 SortingScene

To sort the displayed objects by their respective subcategory by clicking on one of the category buttons, random coordinates [4.17] are needed dependant on the screen size and object size.

Listing 4.17: *returnQuad() (sortingScene.ts)*

```
1       ...
2       private returnQuad(quadrant: number, quadrantType: number): number[]
            {
3           let ret: number[] = null;
4           ...
5           const leftOffsite: number = 100;
6           const rightOffsite: number = 0;
7           const topOffsite: number = 0;
8           const bottomOffsite: number = 100;
9
10          // Has entries dependant of
11          const horizontal: number[] = [];
12
13          // Has numberOfLines + 1 entries
14          const vertical: number[] = [];
15
16          horizontal.push(leftOffsite);
17
18          vertical.push(topOffsite);
19
20          switch (quadrantType) {
21              case 3: {
22                  horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) / 3);
23                  horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) * 2 / 3);
24                  break;
25              }
26              case 4: {
27                  horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) / 2);
28                  vertical.push(topOffsite + (this.cameras.main.height -
                       topOffsite - bottomOffsite) / 2);
29                  break;
30              }
31              case 6: {
32                  horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) / 3);
33                  horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) * 2 / 3);
34                  vertical.push(topOffsite + (this.cameras.main.height -
                       topOffsite - bottomOffsite) / 2);
35                  break;
36              }
37              default: {
38                  break;
39              }
40          }
41
42          horizontal.push(this.cameras.main.width - rightOffsite);
43          vertical.push(this.cameras.main.height - bottomOffsite);
44
45          switch (quadrantType) {
46              case 3: {
```

```
47                  ret = [Phaser.Math.RND.between(horizontal[quadrant] +
                        spriteSizeHalf, horizontal[quadrant + 1] -
                        spriteSizeHalf), Phaser.Math.RND.between(vertical[0]
                         + spriteSizeHalf + this.cameras.main.height / 8,
                        vertical[1] - spriteSizeHalf - this.cameras.main.
                        height / 8)];
48                  break;
49              }
50              case 4: {
51                  if (quadrant < 2) {
52                      ret = [Phaser.Math.RND.between(horizontal[quadrant]
                            + spriteSizeHalf, horizontal[quadrant + 1] -
                            spriteSizeHalf), Phaser.Math.RND.between(
                            vertical[0] + spriteSizeHalf, vertical[1] -
                            spriteSizeHalf)];
53                  } else {
54                      ret = [Phaser.Math.RND.between(horizontal[quadrant
                            \% 2] + spriteSizeHalf, horizontal[(quadrant \%
                            2) + 1] - spriteSizeHalf), Phaser.Math.RND.
                            between(vertical[1] + spriteSizeHalf, vertical
                            [2] - spriteSizeHalf)];
55
56                  }
57                  break;
58              }
59              case 6: {
60                  if (quadrant < 3) {
61                      ret = [Phaser.Math.RND.between(horizontal[quadrant]
                            + spriteSizeHalf, horizontal[quadrant + 1] -
                            spriteSizeHalf), Phaser.Math.RND.between(
                            vertical[0] + spriteSizeHalf, vertical[1] -
                            spriteSizeHalf)];
62                  } else {
63                      ret = [Phaser.Math.RND.between(horizontal[quadrant
                            \% 3] + spriteSizeHalf, horizontal[(quadrant \%
                            3) + 1] - spriteSizeHalf), Phaser.Math.RND.
                            between(vertical[1] + spriteSizeHalf, vertical
                            [2] - spriteSizeHalf)];
64                  }
65                  break;
66              }
67              default: {
68                  break;
69              }
70          }
71          return ret;
72      }
```

4.6.9 PropertySortingScene

To specify the difficulty level, two fields are needed:

Listing 4.18: *Level fields (propertySortingScene.ts)*

```
1       ...
2       private setCat: number;
3       ...
4       private infinite: boolean;
5       ...
```

In contrast to other scenes, objects must have the type Phaser.Physics.Arcade.Sprite as only arcade sprites have the possibility of an acceleration in a direction.

As additional visual feature objects get a random spin velocity and the time an object spawns gets shorter over time.

4.6.10 RestrictedSortingScene

To check if objects in the same box have some property in common, the four properties of an object are taken as a list of strings and intersected with the list of strings from the other objects in the same box [4.19]. If finally there is no empty list, the objects have some property in common and thus this is a valid solution for one box. Which one does not matter to us in this case. That way, the possibility of multiple solutions, which were not intended but also correct, is open.

Listing 4.19: *equalityCheck (restrictedSortingScene.ts)*

```
1       ...
2           private equalityCheck(gameObject: Phaser.GameObjects.Sprite,
                dropZone: Phaser.GameObjects.Zone): boolean {
3           ...
4           let mergeArray: any[] = [];
5           ...
6           for (let cat of this.jsonObject['categories']) {
7               mergeArray = [...mergeArray, ...cat['validElements']];
8           }
9           mergeArray.forEach((element, index, array) => array[index] =
                element.name);
10          ...
11          [...this.objZoneMap.filter((element, index) => this.zoneObjMap[
                index].name === dropZone.name), gameObject].forEach(function
                 (element) {
12              mergeArray = mergeArray.filter((x) => element.getData('
                    properties').includes(x));
13          });
14          ...
15          return (mergeArray.length > 0);
16      }
17      ...
```

4.6.11 GameScene

In the game scene, three marked objects are checked for equality if the respective method was not already run for exactly those three objects [4.20].

Listing 4.20: *update (gameScene.ts)*

```
1     ...
2     update(time: number): void {
3         ...
4         if (!this.checked && this.arrayMarked.getLength() >= 3) {
5             this.checked = true;
6             ...
7         }
8         ...
9         if (timedata <= 0) {
10            this.checked = true;
11
12            ...
13        } else {
14            timedata -= this.timedataStepsize;
15            this.timefluid.setData('timeY', timedata);
16            this.timefluid.setScale(this.timefluid.getData('timeX'),
                  timedata);
17        }
18    }
```

The maximum score 'gameMax' is calculated with a quotient. Therefore there is a rounding error in the floating point arithmetic. To even this error epsilon, the maximum relative error of the rounding procedure, has to be added or subtracted.

Listing 4.21: *updateProgressbar (gameScene.ts)*

```
1     ...
2     if (this.points >= this.gamefluid.getData('gameMax') - Phaser.Math.
          EPSILON) {
3         ...
4     }
5     ...
```

The helpers menu icons have to be marked according to the selected three objects. Therefore the checkEquality method [4.22] is modified with a boolean to mark the icons while executing the check.

Listing 4.22: *checkEquality (gameScene.ts)*

```
1     ...
2     private checkEquality(sprite1: Phaser.GameObjects.GameObject,
          sprite2: Phaser.GameObjects.GameObject, sprite3: Phaser.
          GameObjects.GameObject, inGame: boolean): boolean {
3         if (sprite1 instanceof Phaser.GameObjects.Sprite &&
4             sprite2 instanceof Phaser.GameObjects.Sprite &&
5             sprite3 instanceof Phaser.GameObjects.Sprite
6         ) {
7             // Return value
8             let replaceObjects: boolean = true;
9
10            for (let categoryIndicator of this.arrayCategory.getChildren
                  ()) {
11
12                // Make sure your objects are sprites
```

```
13                  if (categoryIndicator instanceof Phaser.GameObjects.
                        Sprite) {
14
15                      // Clear tint
16                      categoryIndicator.clearTint();
17
18                      if (
19                          sprite1.getData(categoryIndicator.name) ===
                                sprite2.getData(categoryIndicator.name) &&
20                          sprite2.getData(categoryIndicator.name) ===
                                sprite3.getData(categoryIndicator.name) &&
21                          sprite1.getData(categoryIndicator.name) ===
                                sprite3.getData(categoryIndicator.name)
22                      ) {
23                          if (inGame) {
24                              categoryIndicator.setTintFill(0x00dd00);
25                          }
26                      } else if (
27                          !(sprite1.getData(categoryIndicator.name) ===
                                sprite2.getData(categoryIndicator.name)) &&
28                          !(sprite2.getData(categoryIndicator.name) ===
                                sprite3.getData(categoryIndicator.name)) &&
29                          !(sprite1.getData(categoryIndicator.name) ===
                                sprite3.getData(categoryIndicator.name))
30                      ) {
31                          if (inGame) {
32                              categoryIndicator.setTintFill(0x00dd00);
33                          }
34                      } else {
35                          if (replaceObjects) {
36                              replaceObjects = false;
37                          }
38                          if (inGame) {
39                              // Mark category as red
40                              categoryIndicator.setTintFill(0xdd0000);
41                          }
42                      }
43                  }
44              }
45              return replaceObjects;
46          }
47      }
```

To add another small gamification element in the form of a mini reward to the game, every time the user finds a matching pair, some additional is added to the timer [4.23]. So the user is even more motivated to find pairs even faster. But not only that, the user sets himself under stress and thus the game gets harder without making it actually harder.

Listing 4.23: *updateProgressbar (gameScene.ts)*

```
1       ...
2       let timedata: number = this.timefluid.getData('timeY');
3       timedata += this.timedataStepsize * 5000;
4       if (timedata > this.timefluid.getData('timeYMax')) {
5           timedata = this.timefluid.getData('timeYMax');
```

```
6       }
7       ...
```

It is frustrating to not find a matching pair, thus it is ensured [4.24] that there is at least one possible pair every time a new object is added.

Listing 4.24: *Refreshing the current set of objects (gameScene.ts)*

```
1       ...
2       private rebuildDisplayedObjects(): void {
3           ...
4           for (let card of this.arrayDisplayed.getChildren()) {
5               if (card instanceof Phaser.GameObjects.Sprite) {
6                   card.setVisible(false);
7                   this.arrayStack.add(card);
8               }
9           }
10
11          this.arrayDisplayed.clear(false, false);
12
13          this.initObjects();
14      }
15      ...
```

4.6.12 ScoreScene

The most important and special part in the score scene is the way the score is saved, as explained in the score storage section [4.3].

4.7 Final Look of the Learning Environment

This section contains the final look of the learning environment.

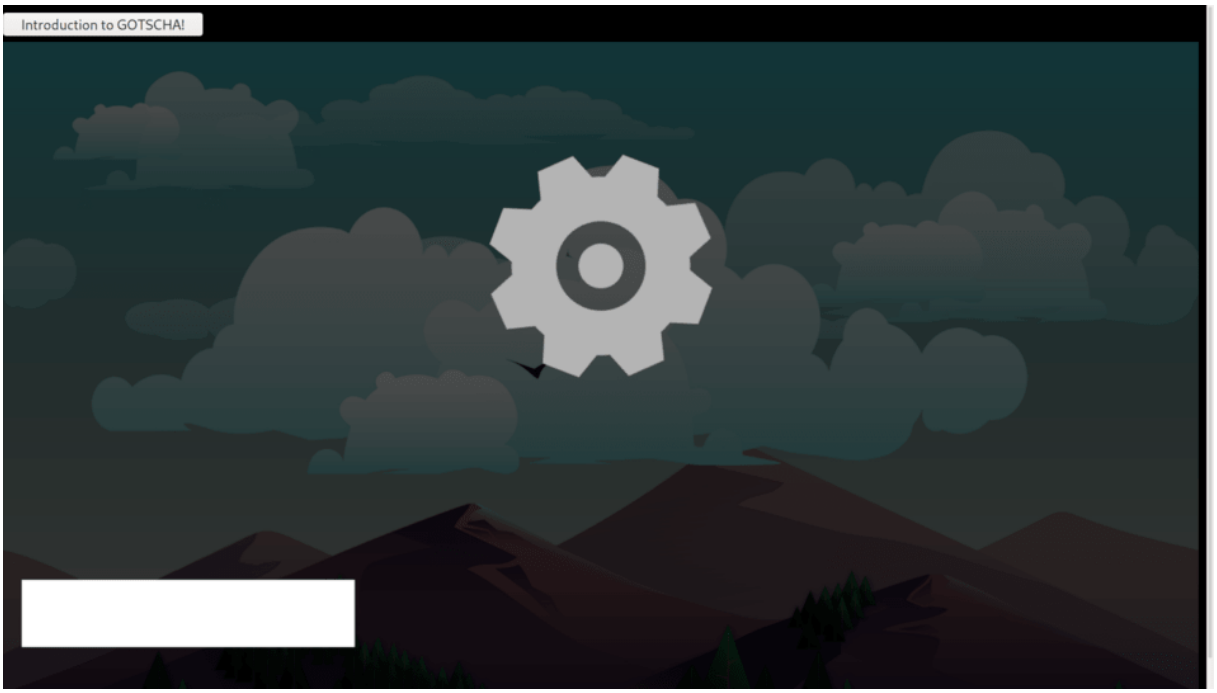

Figure 4.9: *Loading Screen*

Figure 4.10: *Title Screen*

Figure 4.11: *Level Menu*

Figure 4.12: *Level Menu showing Stars*

Figure 4.13: *Sorting Scene Category Mixture*

Figure 4.14: *Sorting Scene Category 1*

Figure 4.15: *Sorting Scene Category 2*

Figure 4.16: *Sorting Scene Category 3*

Figure 4.17: *Sorting Scene Category 4*

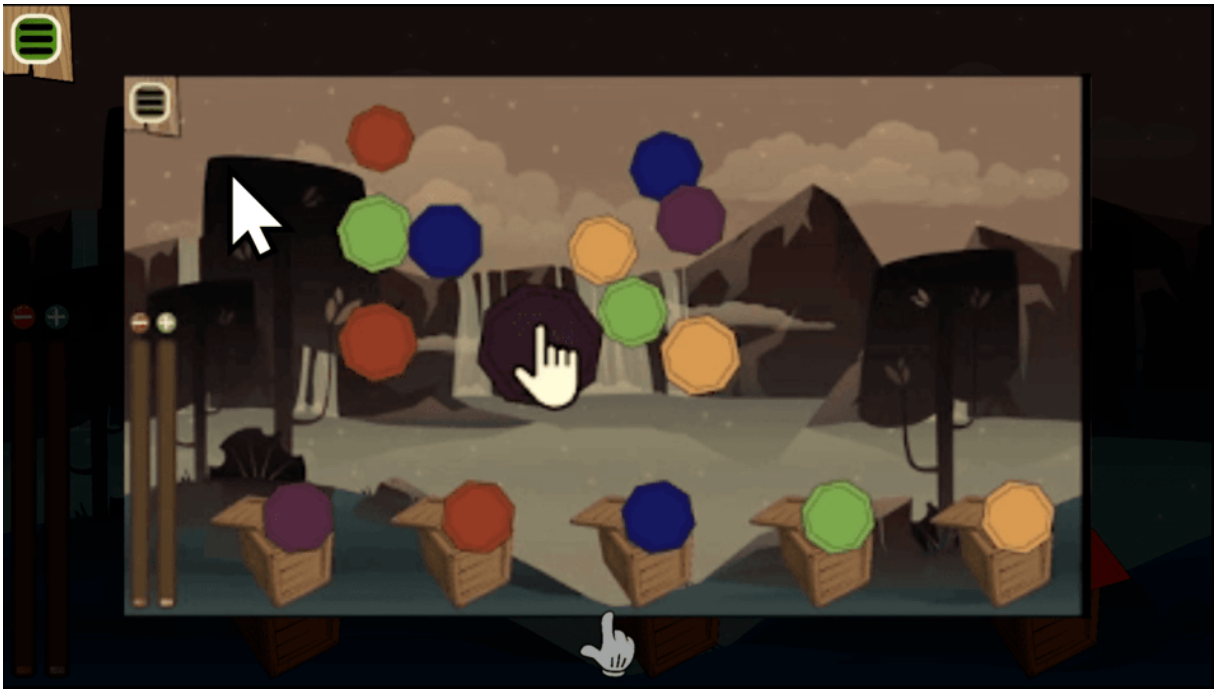

Figure 4.18: *Introduction Screen to Sorting Scene*

Figure 4.19: *Property Sorting Category 1*

Figure 4.20: *Property Sorting Category 2*

Figure 4.21: *Property Sorting Category 3*

Figure 4.22: *Property Sorting Category 4*

Figure 4.23: *Falling Property Sorting Category 1, 2–4 will be omitted*

Figure 4.24: *Restricted Sorting Easy*

Figure 4.25: *Restricted Sorting 2 Hard*

Figure 4.26: *Introduction to Object Pairing Easy*

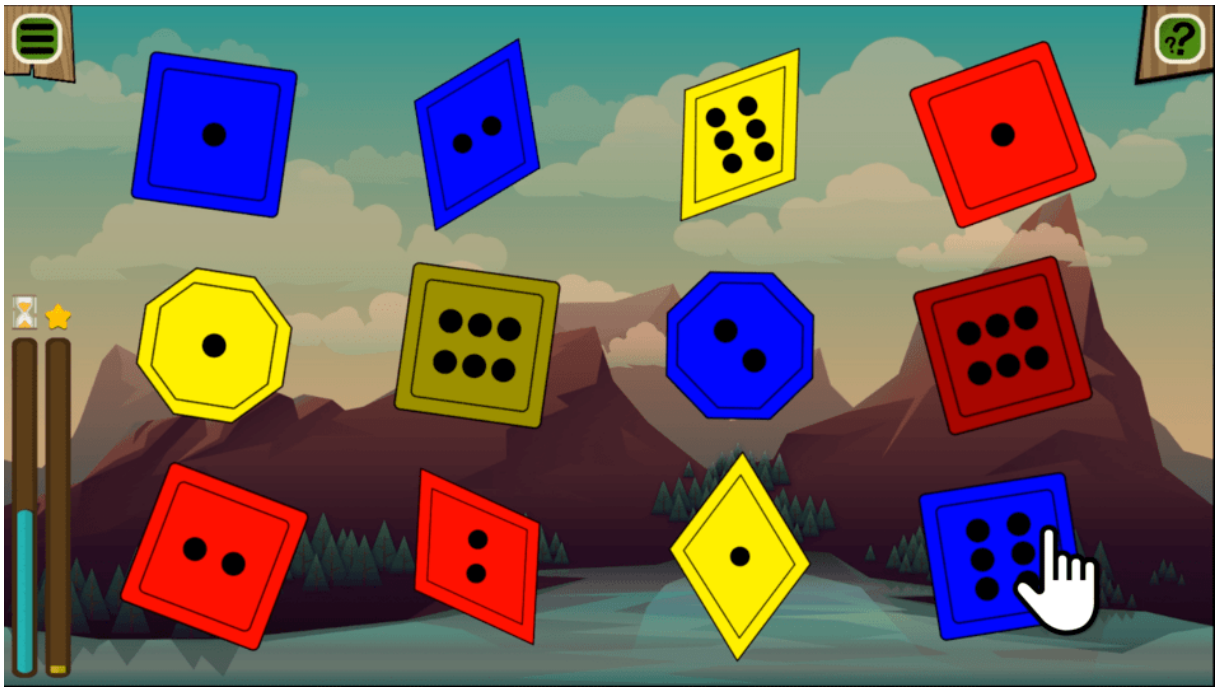

Figure 4.27: *Object Pairing Easy*

Figure 4.28: *Object Pairing Easy Helper Bar*

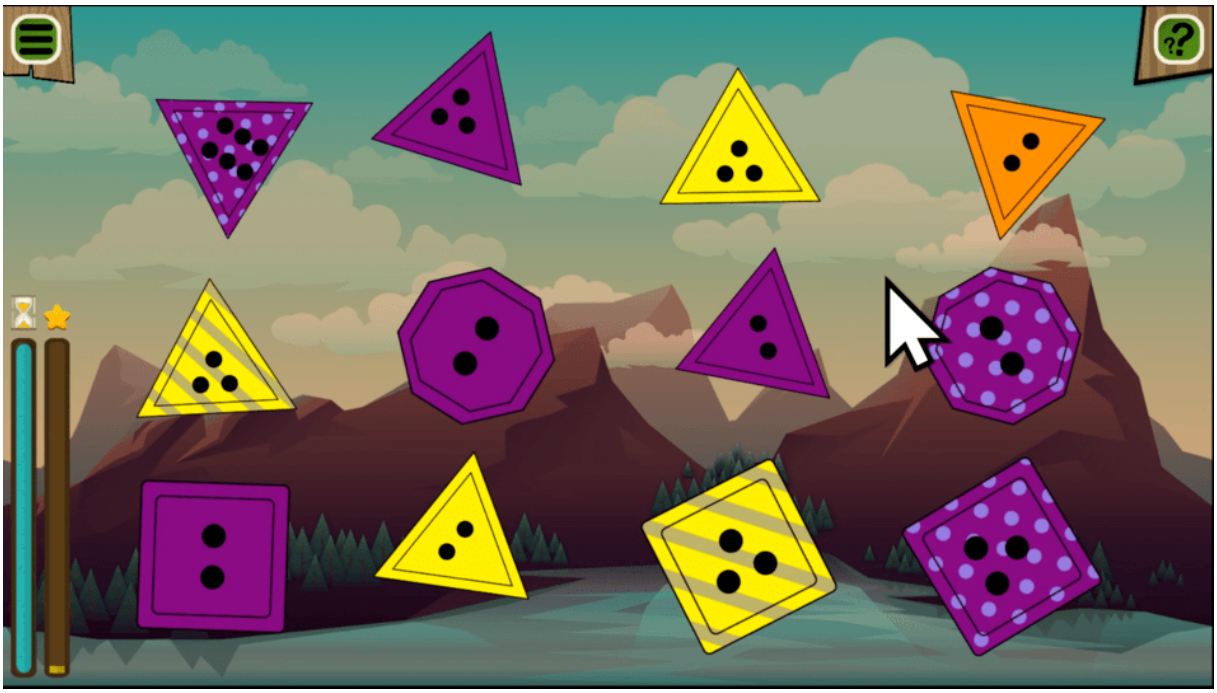

Figure 4.29: *Object Pairing Hard*

Figure 4.30: *Object Pairing Hard Helper Bar*

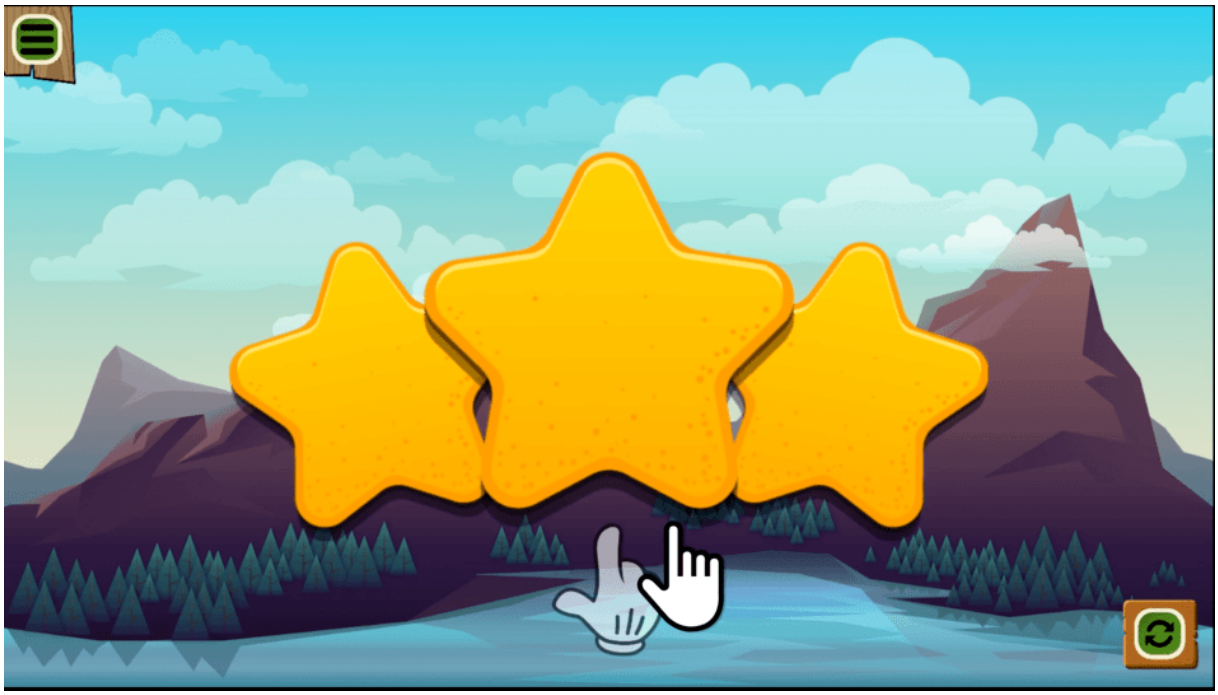

Figure 4.31: *Score Screen*

5

Evaluation

In this section the environment the game was tested in will be presented and analysed. First every game has to be tested on different devices for code errors or unintended behaviour. Second a target and various audience will test the game and significant reactions and behaviour will be addressed.

5.1 Debugging

5.1.1 Devices

The game was tested on the following devices and browsers:

Browsers

- Firefox 68.0
- Chrome 77
- Microsoft Edge 18 (only on devices with Windows OS)
- Safari 5.0.4 (only on iOS devices)

Devices

- Computer with Windows 10
- Computer with Apple iOS
- Tablet with Apple iOS

- Tablet with Android
- Smartphone with Android
- iPhone

5.1.2 Differences and obstacles

On Windows and Android devices the browser based application run without any troubles. On Apple devices there is an issue with the fullscreen option. As soon as the fullscreen is engaged, the touch functions of the operating system interferes with the touch input in the browser window. This is a common problem with apple devices and the only known solution to work so far is to just minimize the header bar. This is an acceptable solution considering all the safety restriction apple applies to its users and developers.

5.1.3 Unexpected Bugs

With the various test audience some unexpected bugs are found. The most fascinating are mentioned here.

One would not expect the irrational behaviour users can simulate if tasked with finding errors. For example dragging an object not to its intended destination but to the screen/window boundary and even further. To ensure the object can still be accessed through dragging, a restriction on dragging the object outside of the windows had to be added.

After a goal is reached and the scene transition sets in, one could expect the user to wait until a new scene is loaded. As this is not the case further restriction on the user input had to be added.

The users thinking becomes faster the longer he plays the game and so are his interaction/inputs. Thus different locks on animations had to be placed, so that while some animation is in progress not another animation can be triggered.

It was fascinating to see that users have a different thickness of their finger. For users with a thicker fingers, sometimes the object size was too small to touch and still observe the now dragged object. Thus adjustments had to be made to the size of objects.

5.2 Target Audience Test

The created learning environment was tested on two groups:

1. 6 test subjects; age between 4 and 7; advanced knowledge on handling electronic devices
2. 5 test subjects; age between 4 and 7; less to none knowledge on handling electronic devices

The first group had no problem at all interacting with the browser based application. The second group needed more time to get a feeling of the game mechanics but after the adjustment time

the difference of the two groups was negligible.

In the end both test groups were eager to play and explore the newly presented learning environment.

In the first levels the perfect score was almost always reached on the first try.

The younger ones had the most fun with the middle staged levels with the falling objects but were eager and curious to explore the harder levels. With the support of the older ones or even an adult playing the easy pairing game was possible and understandable.

The older ones were most eager to understand and beat the hardest level and sometimes found a possible solution faster than the adult test group.

What was fascinating that almost every test subject knew what to do in the level with restricted space but some had different tactics to approach it. The two tactics used were:

- Fixating a category and finding the solution with trial and error.
- Sorting the objects in the field above the boxes after a category.

Some test subjects wanted to play more even after playing all the levels and some of them even formed a group and were discussing possible solutions, sharing their thoughts.

5.3 Various Audience Test

1. 11 test subjects; age between 18 and 40; daily usage of electronic devices (computer, tablet, smartphone)
2. 13 test subjects; age between 30 and 70; only some knowledge in how to handle a smartphone

The first test group had no problem at all finding out how to interact with the browser based application. All of them figured out all tasks on their own. As in the first test run the time limit on the last level was somewhat too small, at first no one managed to beat it at all. Fascinating to see was that 5 test subjects did not want to stop playing until they beat the game, what they did in the end.

The second test group needed some adjustment time to figure out how to interact with the browser based application. On some one could see that the motoric abilities were not trained as well. One could assume that this were the older test subjects. On the contrary, yes there were two test subject in the older spectrum, but three younger test subjects had not used their smartphone in that way in a long time and some of their motoric functions deteriorated in a fascinating way. Furthermore almost all of them had trouble understanding the level with sorting objects in boxes with restricted space but not with the other levels. Though the ones who immediately understood the task on hand had problem understanding other tasks others had no problems with.

There were two teachers (one preschool and one kindergarden) who tested the game. One of them was not found of tablets in kindergarden and opposed the idea in the beginning. After she

found out that the application was not intended to replace some topics or ways they are teaching but to enhance their spectrum and ability to deal with their pupils on their respective levels, she wanted to give the idea a chance. The aspect of group work was also very welcomed.

6

Conclusion

The idea of not replacing but enhancing a teachers ability to teach pupils on their respective levels with game based learning has potential. It is an option worth evaluating in detail as the interest and curiosity of the test subjects (age independently) was on both sides in my opinion by far not negligible. The adapting behaviour concerning task recognition and group building has shown that electronic devices are not bad tools, only when used the wrong way.

Interesting to see is that an application intended for a very young age is able to foster and fascinate even adults.

6.1 Limitation

One big hurdle will be the time investment to create an almost ideal prototype to convince teachers that using this learning method is not a downgrade from the current way of learning in school/kindergarten.

Through my thesis I realized that a lot can be achieved by having the right background/story and design. For that the modularity comes into play. With the cornerstones laid the right environment can be created and hopefully integrated in my work.

As much as I would like to say I understood all the things I did, even in the end, I discovered new techniques and features of the phaser framework. Those new insights could further tighten, defragment and increase the readability and maintainability of my code. But to which extent I dare not to make a guess.

Further TypeScript support will be added with the upcoming fourth version of phaser.

6.2 Future Improvements

Future improvements are clearly the integration of the knowledge gained towards the end of my work as well as coming up with a specific design and environment for the topic the game is used in combination with the other learning material. Another aspect would be the same evaluation on bigger scale with a more specific application (technical design).

Bibliography

[1] Bachelor Thesis - Gotscha! https://gitlab.ethz.ch/ethz-projects/bachelor-thesis.git. Accessed: 2019-01-24.

[2] Best JavaScript Game Engines and Games to Download. https://code.tutsplus.com/articles/javascript-game-engines-for-your-next-project--cms-32311. Accessed: 2019-01-17.

[3] Canvas API Documentation. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. Accessed: 2019-01-17.

[4] Einfach Informatik - Lernumgebungen. https://einfachinformatik.inf.ethz.ch/. Accessed: 2019-01-17.

[5] Games and learning: an introduction. https://edtechreview.in/dictionary/298-what-is-game-based-learning. Accessed: 2019-01-24.

[6] GIMP - GNU Image Manipulation Program. https://www.gimp.org/. Accessed: 2019-01-17.

[7] Gotscha! https://n.ethz.ch/~knobelf. Accessed: 2019-01-24.

[8] HTML5. https://www.w3.org/TR/2017/REC-html51-20171003/. Accessed: 2019-01-17.

[9] HTML5 Web Storage. https://www.w3schools.com/html/html5_webstorage.asp. Accessed: 2019-01-17.

[10] Open Broadcaster Software (OBS) Studio. https://obsproject.com/. Accessed: 2019-01-17.

[11] Phaser 3 Documentation. https://photonstorm.github.io/phaser3-docs/. Accessed: 2019-01-17.

[12] Phaser 3 Framework. https://phaser.io/. Accessed: 2019-01-17.

[13] Phaser 3 Labratories. https://labs.phaser.io/. Accessed: 2019-01-17.

[14] Phaser 3 Series: Let's Talk About Scenes. https://github.com/jdotrjs/phaser-guides/blob/master/Basics/Part3.md. Accessed: 2019-01-17.

[15] Reader Rabbit - First Grade. https://www.mobygames.com/game/reader-rabbit-1st-grade. Accessed: 2019-01-17.

[16] Super Solvers: OutNumbered! https://www.mobygames.com/game/super-solvers-outnumbered. Accessed: 2019-01-17.

[17] SVG Tutorial. https://www.w3schools.com/graphics/svg_intro.asp. Accessed: 2019-01-17.

[18] The Learning Company. https://www.mobygames.com/company/learning-company. Accessed: 2019-01-17.

[19] TypeScript Documentation. https://www.typescriptlang.org/docs/home.html. Accessed: 2019-01-17.

[20] WebGL 2 Stats, MAX_TEXTURE_SIZE. https://webglstats.com/webgl2/parameter/MAX_TEXTURE_SIZE. Accessed: 2019-01-17.

[21] WebGL API Documentation. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API. Accessed: 2019-01-17.

[22] Benjamin Lent Davis and Diane Maclagan. The card game SET. *The Mathematical Intelligencer*, 25(3):33–40, 2003.

[23] Juraj Hromkovič, Tobias Kohn, Dennis Komm, and Giovanni Serafini. Algorithmic thinking from the start. *Bulletin of the EATCS*, 121.

[24] Jill Weber. Eine interaktive Lernplattform füralgorithmisches Denken, für 4-8 jährige Kinder. *Master Thesis, ETH Zürich, Department of Computer Science*, 2019.

[25] Juraj Hromkovič, Regula Lacher. *Einfach Informatik 5/6 – Lösungen finden*. KLETT, 2019.

[26] Juraj Hromkovič, Regula Lacher. *Einfach Informatik 7 – 9 Daten darstellen, verschlüsseln, komprimieren*. KLETT, 2019.

[27] Juraj Hromkovič, Regula Lacher. *Einfach Informatik 7 – 9 Strategien entwickeln*. KLETT, 2019.

[28] Juraj Hromkovič, Regula Lacher. *INFORMATIK BIER in KG und 1/2 - Ordnung, Suche, Zeichensrpachen und Programmieren*. KLETT, 2020.

[29] Karl M Kapp. *The gamification of learning and instruction fieldbook: Ideas into practice*. John Wiley & Sons, 2013.

[30] Kevin Tang. Graphs, trees and discrete optimizations: Computer-based learning envi-

ronment about combinatorial problems for secondary education. *Bachelor Thesis, ETH Zürich, Department of Computer Science*, 2019.

[31] Donald E Knuth. The art of computer programming, volume 3: sorting and searching, 1998.

[32] Sarah Eleonora Kamp. A platform independant, computer-based learning environment. *Bachelor Thesis, ETH Zürich, Department of Computer Science*, 2019.

[33] Sonja Tabea Blum. Eine interaktive Lernplattform füralgorithmisches Denken, für 4-8 jährige Kinder. `https://doi.org/10.3929/ethz-b-000312911`, 2018. Accessed: 2019-01-17.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.