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 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. 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 meantime 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 meantime 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 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 opinion 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 engagement 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, leader board, virtual rewards, and story line or even quests. There is also an aspect to game elements that allow for social interaction between players and the acquisition 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 game play 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 game play 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 story line with a specific designed environment which includes mini games 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 mini games 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 game play, a more interesting story line 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 play field. The play field 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 using 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 adds 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 full screen 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 to detect and compare 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.

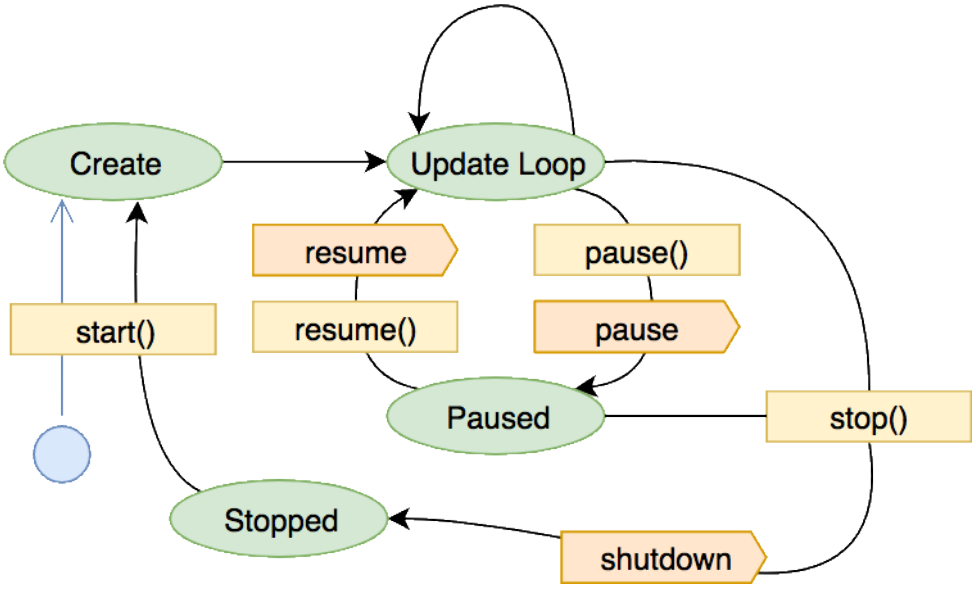

Figure 4.1.: *Scene Lifecycle[14]*

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 [4.7, 4.8].

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.

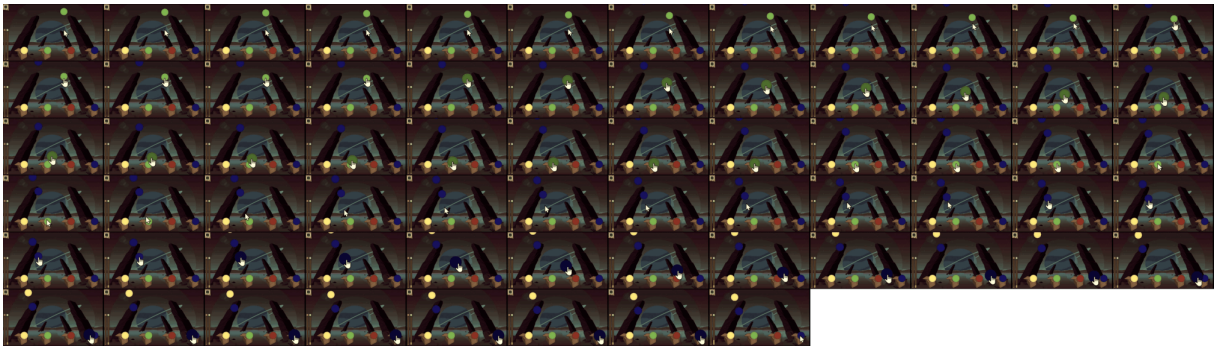

Figure 4.2.: *Example: Spreadsheet*

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 fulfill 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 fulfill 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 fulfill 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 fulfill the conditions, the play field will be generated anew.

Object pairing - easy version

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

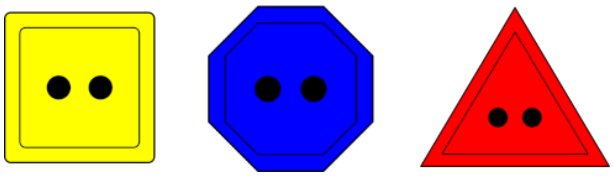

Figure 4.3.: *Example: Matching Set Easy*

Object pairing - hard version

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

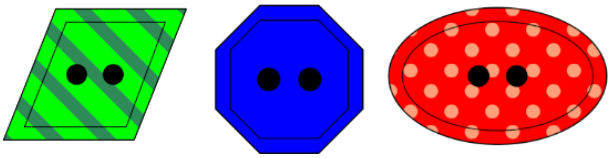

Figure 4.4.: *Example: Matching Set Hard*

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 [A] 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 acquiring 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 [4.6, 4.7, 4.8].

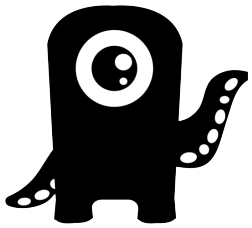

Figure 4.6.: *Easy Level Icon*

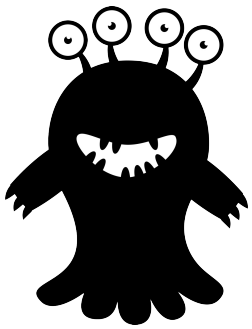

Figure 4.7.: *Medium Level Icon*

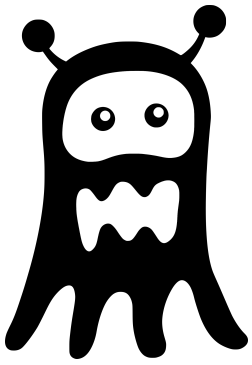

Figure 4.8.: *Hard Level Icon*

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 [4.16]. 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)*

```
...
private initInput(): void {
    this.input.on('pointerdown', function(pointer, currentlyOver) {
        const gameObject: any = currentlyOver[0];
        if (gameObject instanceof Phaser.GameObjects.Sprite) {
            ...
            gameObject.setData('clicked', true);
        }
    }, this);

    this.input.on('pointerup', function(pointer, currentlyOver) {
        const gameObject: any = currentlyOver[0];
        if (gameObject instanceof Phaser.GameObjects.Sprite &&
            gameObject.getData('clicked')) {
            this.buttonFunction(gameObject);
        }

        this.levelButtons.getChildren().forEach(function(gameObject)
            {
            if (gameObject instanceof Phaser.GameObjects.Sprite) {
                ...
                gameObject.setData('clicked', false);
            }
        }, this);
    }, this);
}
...
```

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)*

```
...
private returnQuad(quadrant: number, quadrantType: number): number[] {
    let ret: number[] = null;
    ...
    const leftOffsite: number = 100;
    const rightOffsite: number = 0;
    const topOffsite: number = 0;
    const bottomOffsite: number = 100;

    // Has entries dependant of
    const horizontal: number[] = [];

    // Has numberOfLines + 1 entries
    const vertical: number[] = [];

    horizontal.push(leftOffsite);

    vertical.push(topOffsite);

    switch (quadrantType) {
        case 3: {
            horizontal.push(leftOffsite + (this.cameras.main.width -
                leftOffsite - rightOffsite) / 3);
            horizontal.push(leftOffsite + (this.cameras.main.width -
                leftOffsite - rightOffsite) * 2 / 3);
            break;
        }
        case 4: {
            horizontal.push(leftOffsite + (this.cameras.main.width -
                leftOffsite - rightOffsite) / 2);
            vertical.push(topOffsite + (this.cameras.main.height -
               topOffsite - bottomOffsite) / 2);
            break;
        }
        case 6: {
            horizontal.push(leftOffsite + (this.cameras.main.width -
                leftOffsite - rightOffsite) / 3);
            horizontal.push(leftOffsite + (this.cameras.main.width -
                leftOffsite - rightOffsite) * 2 / 3);
            vertical.push(topOffsite + (this.cameras.main.height -
               topOffsite - bottomOffsite) / 2);
            break;
        }
        default: {
            break;
        }
    }

    horizontal.push(this.cameras.main.width - rightOffsite);
    vertical.push(this.cameras.main.height - bottomOffsite);

    switch (quadrantType) {
        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)*

```
    ...
    private setCat: number;
    ...
    private infinite: boolean;
    ...
```

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)*

```
    ...
        private equalityCheck(gameObject: Phaser.GameObjects.Sprite,
            dropZone: Phaser.GameObjects.Zone): boolean {
        ...
        let mergeArray: any[] = [];
        ...
        for (let cat of this.jsonObject['categories']) {
            mergeArray = [...mergeArray, ...cat['validElements']];
        }
        mergeArray.forEach((element, index, array) => array[index] =
            element.name);
        ...
        [...this.objZoneMap.filter((element, index) => this.zoneObjMap[
            index].name === dropZone.name), gameObject].forEach(function
             (element) {
            mergeArray = mergeArray.filter((x) => element.getData('
                properties').includes(x));
        });
        ...
        return (mergeArray.length > 0);
    }
    ...
```

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.

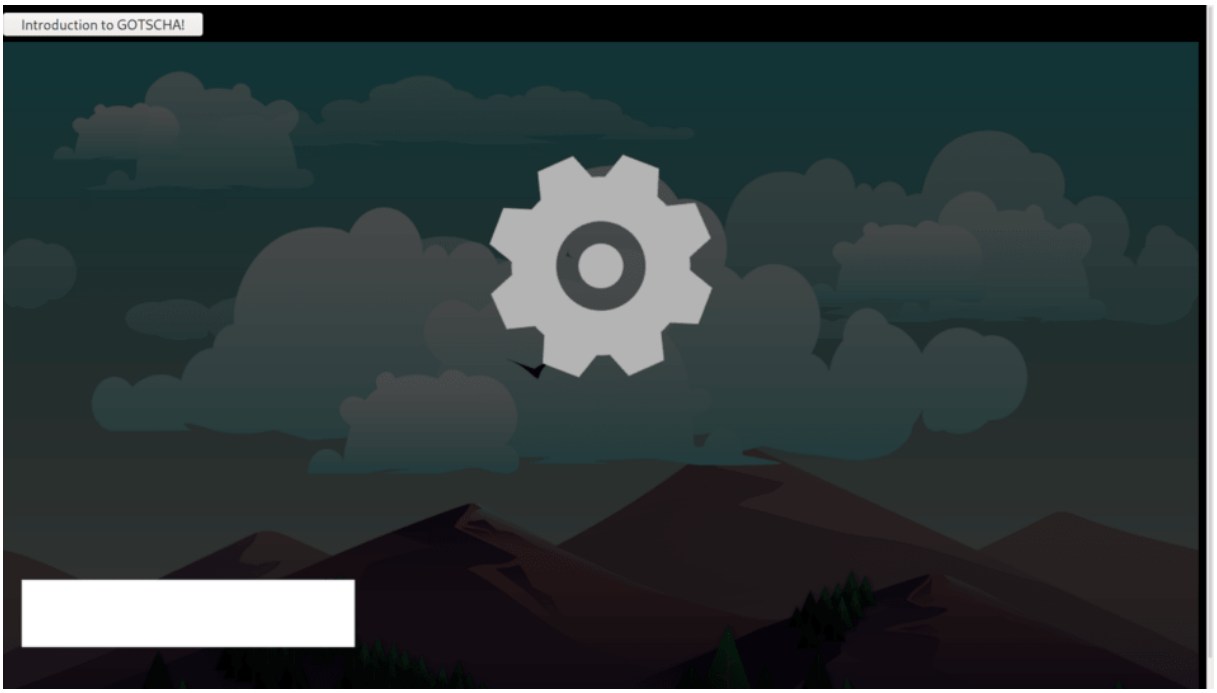

Figure 4.9.: *Loading Screen*

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*

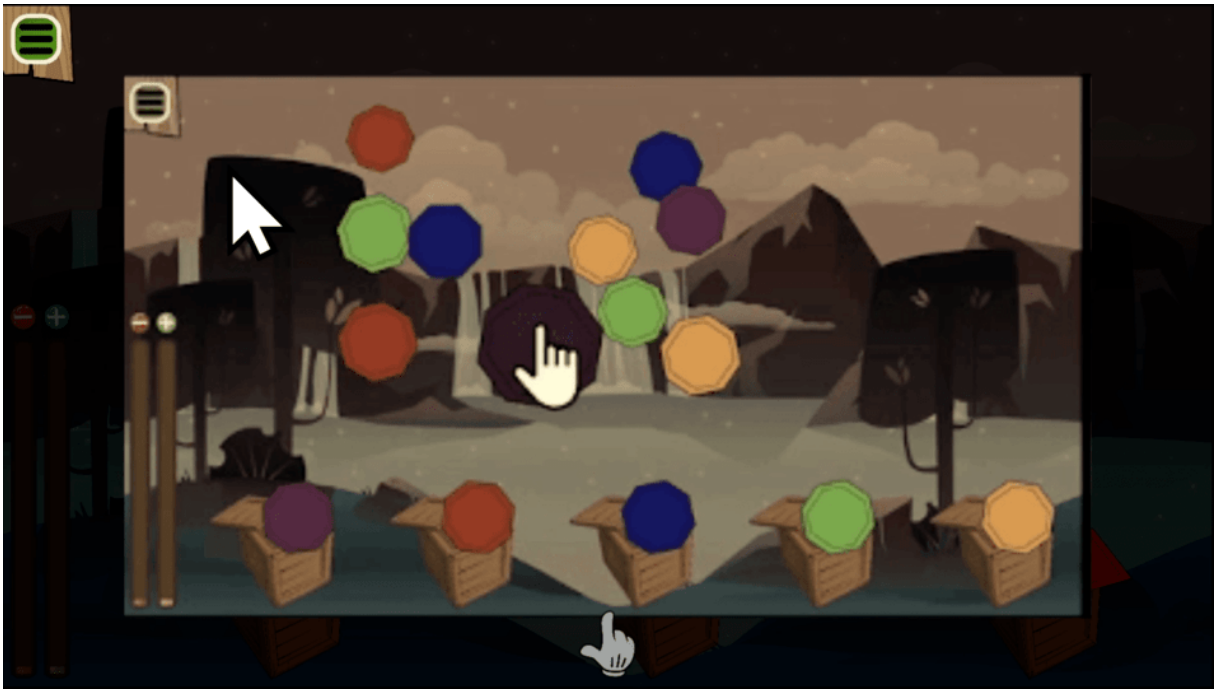

Figure 4.18.: *Introduction Screen to Sorting Scene*

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*

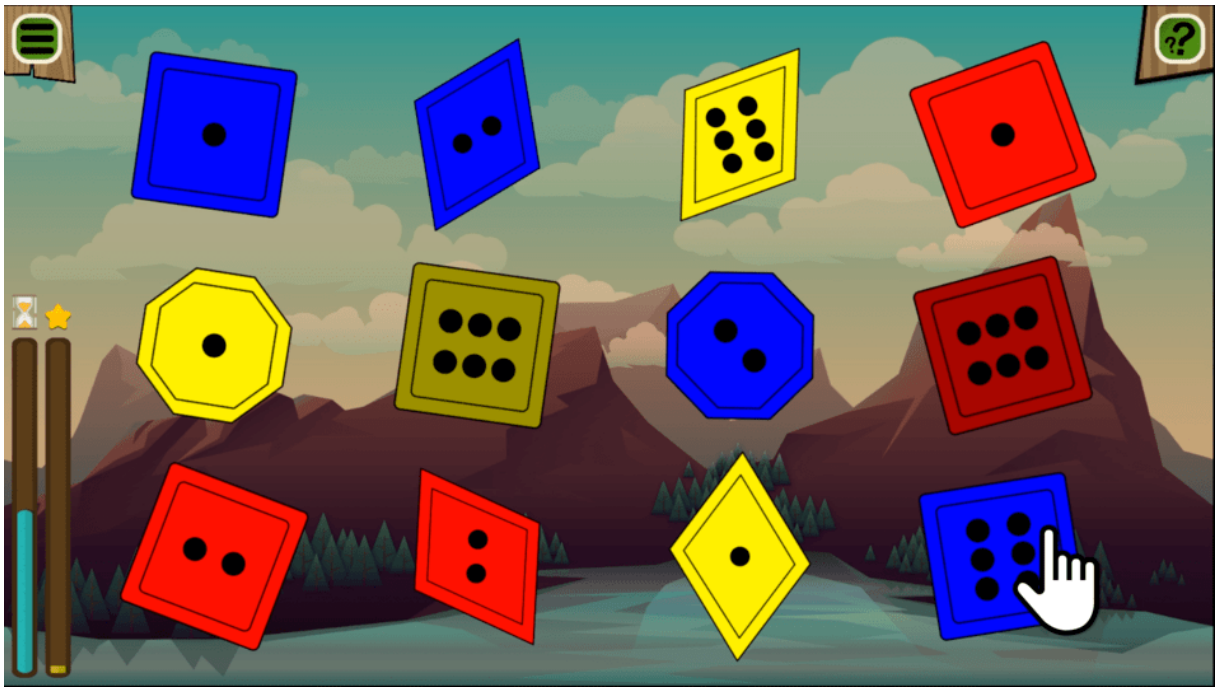

Figure 4.27.: *Object Pairing Easy*

Figure 4.28.: *Object Pairing Easy Helper Bar*

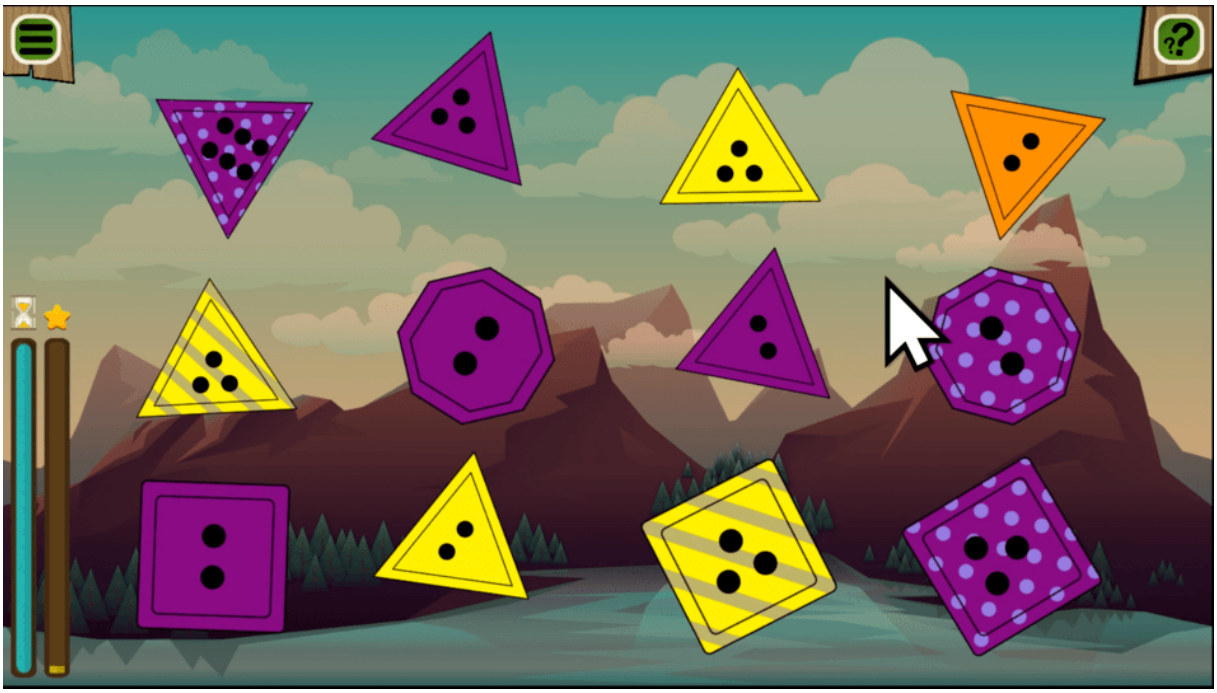

Figure 4.29.: *Object Pairing Hard*

Figure 4.30.: *Object Pairing Hard Helper Bar*

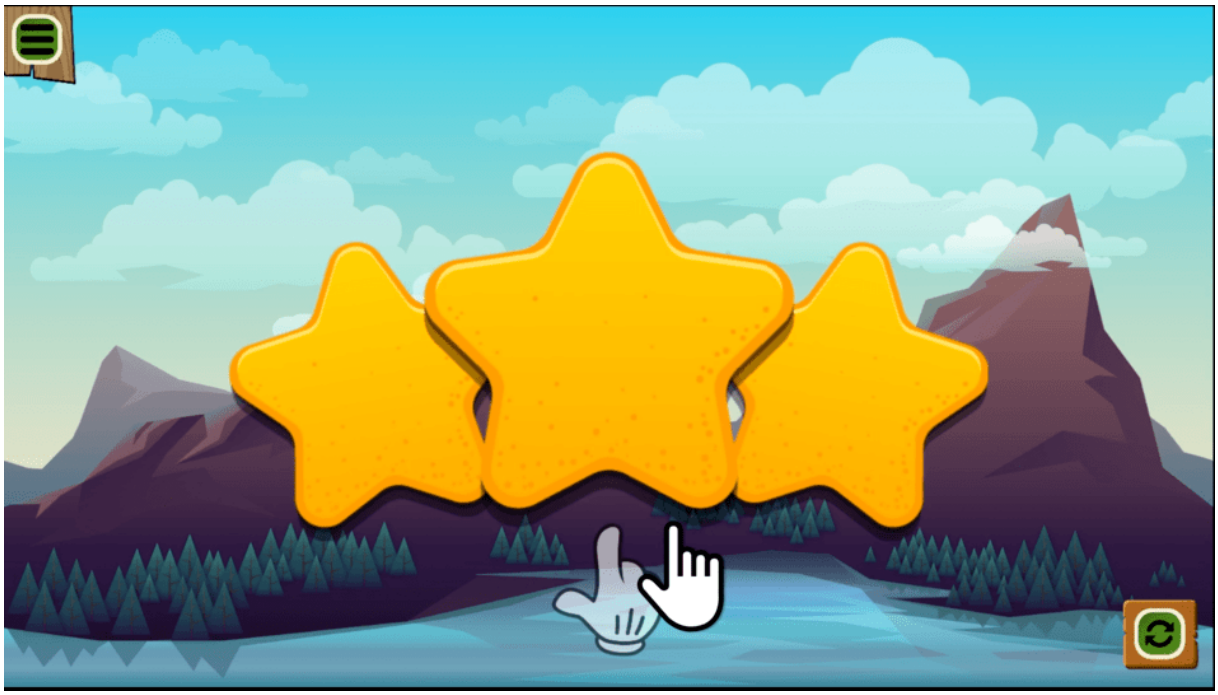

Figure 4.31.: *Score Screen*

5

Evaluation

In this section the environment the game was tested in will be presented and analyzed. 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 full screen option. As soon as the full screen 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 someone 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 kindergarten) who tested the game. One of them was not found of tablets in kindergarten 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).

A

Gotcha! – Full Code

A.1. baseScene.ts

Listing A.1: *baseScene.ts*

```
1  export class BaseScene extends Phaser.Scene {
2      /**
3       * Name of the scene
4       */
5      protected key: string;
6
7      /**
8       * Level of the scene
9       */
10     protected level: number;
11
12     /**
13      * Transition graphic
14      */
15     private transition: Phaser.GameObjects.GameObject[];
16
17     constructor(key: string) {
18         super({
19             key: key
20         });
21
22         this.key = key;
23         this.level = 0;
24         this.generateNewSeed();
25     }
26
```

```
27      /**
28       * Method for returning the key of this scene
29       */
30      public getKey(): string {
31          return this.key;
32      }
33
34      /**
35       * Method for returning the key of this scene
36       */
37      public getLevel(): number {
38          return this.level;
39      }
40
41      /**
42       * Method for generating a new seed so that pseudo randomness is
             guaranteed
43       */
44      private generateNewSeed(): void {
45          const rndStr: string = Phaser.Math.RND.realInRange(Math.pow(10,
                2), Math.pow(10,10)).toString();
46          Phaser.Math.RND.sow([rndStr]);
47      }
48
49      /**
50       * Method for initializing the shape, position and properties of the
              graphical scene transition
51       */
52      private transitionInit(): void {
53          // Shape of the graphical transition
54          const circle: Phaser.GameObjects.Graphics = this.add.graphics();
55
56          // Shape of the screen
57          const rectangle: Phaser.GameObjects.Rectangle = this.add.
                rectangle(0, 0, this.cameras.main.width, this.cameras.main.
                height, 0x000000);
58
59          // Define circle as the mask
60          const mask: Phaser.Display.Masks.GeometryMask = circle.
                createGeometryMask();
61
62          circle.setPosition(this.cameras.main.width / 2, this.cameras.
                main.height / 2);
63          circle.fillCircle(0, 0, 0.1);
64          circle.setDepth(0);
65
66          mask.setInvertAlpha(true);
67
68          rectangle.setDepth(1);
69          rectangle.setOrigin(0, 0);
70          rectangle.setMask(mask);
71
72          circle.fillCircle(0, 0, 0.1);
73
74          this.transition = [circle, rectangle];
75      }
```

```
76
77      /**
78       * Opening transition. Normally used to visually introduce a new
             scene
79       */
80      protected transitionIn(): void {
81          // Generating a new seed, so that randomness is guaranteed in
                every repetition of a scene
82          this.generateNewSeed();
83
84          this.transitionInit();
85
86          this.children.bringToTop(this.transition[1]);
87
88          const tween: Phaser.Tweens.Tween = this.add.tween({
89              targets: this.transition[0],
90              scale: 10 * 0.5 * Math.sqrt(Math.pow(this.cameras.main.width
                    , 2) + Math.pow(this.cameras.main.height, 2)),
91              ease: 'linear',
92              duration: 700
93          });
94
95          tween.on('start', () => this.sound.volume = 0);
96          tween.on('complete', () => this.introduction());
97          tween.on('update', () => this.sound.volume += 1/tween.duration);
98      }
99
100     /**
101      * Closing transition. Normally used to visually close or stop a
             scene.
102      * @param scene The scene you want to start next.
103      * @param data Additional data you want to give to the next scene.
104      */
105     protected transitionOut(scene: string, data?: any): void {
106         this.children.bringToTop(this.transition[1]);
107
108         const tween: Phaser.Tweens.Tween = this.add.tween({
109             targets: this.transition[0],
110             scale: 0,
111             ease: 'linear',
112             duration: 700
113         });
114
115         tween.on('complete', () => this.sceneChange(scene, data));
116         tween.on('update', () => this.sound.volume -= 1/tween.duration);
117     }
118
119     /**
120      * Helper method for starting a new scene and stopping the current
             one
121      * as the behaviour of the current scene when starting a new one is
122      * not clearly defined in the framework at this point of time.
123      * @param scene The scene you want to start next
124      * @param data Additional data you want to give to the next scene.
125      */
126     protected sceneChange(scene: string, data?: any): void {
```

```
127             this.sound.stopAll();
128             this.game.scene.start(scene, data);
129             this.game.scene.stop(this.key);
130         }
131
132         /**
133          * Helper method for playing the introduction and pause the current
                scene
134          */
135         protected introduction(): void {
136             this.scene.pause();
137             this.game.scene.start("IntroScene", {'pausedScene': this.getKey
                    (), 'level': this.getLevel()});
138         }
139
140         /**
141          * Returns the correct scaling factor for the wanted image size in
                relation to the real image size.
142          * @param wantedImageSize Image size you want to have for a
                dimension
143          * @param realImageSizeWidth The image width you want to scale
144          * @param realImageSizeHeight The image height you want to scale
145          * @param scaleToHeight Boolean for scaling height or width of image
                 to the wanted size. Default to false.
146          */
147         protected imageScalingFactor(wantedImageSize: number,
                realImageSizeWidth: number, realImageSizeHeight: number,
                scaleToHeight: boolean = false): number {
148             let ret: number;
149             if (scaleToHeight) {
150                 ret = Math.max(wantedImageSize / realImageSizeWidth,
                        wantedImageSize / realImageSizeHeight);
151             } else {
152                 ret = Math.min(wantedImageSize / realImageSizeWidth,
                        wantedImageSize / realImageSizeHeight);
153             }
154             return ret;
155         }
156 }
```

A.2. preloadAsset.ts

Listing A.2: *preloadAsset.ts*

```
1  import 'phaser';
2  import {BaseScene} from './baseScene';
3
4  export class PreloadAssets extends BaseScene {
5
6      constructor() {
7          super('PreloadAssets');
8      }
9
10     init(): void {
```

```
11
12      }
13
14      preload(): void {
15          this.load.setPath('assets/geometrical_objects/');
16          this.load.json('objects', 'geometrical_objects.json');
17
18          this.load.setPath('assets/ui/');
19          this.load.image('cogwheel', 'cogwheel.png');
20          this.load.image('background1', 'background1.png');
21      }
22
23      create(): void {
24          this.setBackground();
25          this.preLoadImages();
26          this.preLoadIntroFiles();
27          this.preLoadAudio();
28          this.initLoadingGraphics();
29          this.start();
30      }
31
32      /**
33       * Method for preloading all asset images
34       */
35      private preLoadImages(): void {
36          // Load category and object images
37          const jsonObject: any = this.cache.json.get('objects');
38
39          for (let category of jsonObject['categories']) {
40              this.load.setPath('assets/geometrical_objects/categories/');
41              this.load.image(category['name'], category['url']);
42
43              this.load.setPath('assets/geometrical_objects/images/');
44              for (let property of category['validElements']) {
45                  for (let url of property['urls']) {
46                      this.load.image(url, url);
47                  }
48              }
49          }
50
51          for (let image of jsonObject['images']) {
52              this.load.image(image['name'], image['name']);
53          }
54
55          // Load UI images
56          this.load.setPath('assets/ui/');
57          //this.load.image('background1', 'background1.png');
58          this.load.image('background2', 'background2.png');
59          this.load.image('background3', 'background3.png');
60          this.load.image('background4', 'background4.png');
61          this.load.image('background5', 'background5.png');
62
63          this.load.image('menubackground', 'menu_background.png');
64
65          this.load.spritesheet('fullscreenbuttonblack', '
                fullscreen_button_black.png', {frameWidth: 64, frameHeight:
```

```
            64});
66          this.load.image('menubutton', 'menu_button.png');
67          this.load.image('help', 'help.png');
68          this.load.image('exitbutton', 'exit_button.png');
69          this.load.image('replay', 'reload_button.png');
70          this.load.image('erase', 'erase_button.png');
71          this.load.image('return', 'return_button.png');
72
73          this.load.image('star_0', 'star_0.png');
74          this.load.image('star_1', 'star_1.png');
75          this.load.image('star_2', 'star_2.png');
76          this.load.image('star_3', 'star_3.png');
77
78          this.load.image('timefluid', 'timefluid.png');
79          this.load.image('gamefluid', 'gamefluid.png');
80          this.load.image('progressstar', 'star.png');
81          this.load.image('progressbar', 'progressbar.png');
82          this.load.image('progressbarGreen', 'progressbar_green.png');
83          this.load.image('progressbarRed', 'progressbar_red.png');
84          this.load.image('plus', 'plus.png');
85          this.load.image('minus', 'minus.png');
86
87
88          this.load.image('hourglass', 'hourglass.png');
89          this.load.image('finger', 'finger.png');
90
91
92          this.load.image('crate', 'crate_topview.png');
93          this.load.image('wooden_crate', 'wooden_crate.png');
94
95          this.load.image('title', 'title.png');
96
97          this.load.image('catButton', 'cat_button.png');
98          this.load.image('levelButton11', 'level11_button.png');
99          this.load.image('levelButton12', 'level12_button.png');
100         this.load.image('levelButton13', 'level13_button.png');
101         this.load.image('levelButton14', 'level14_button.png');
102         this.load.image('levelButton21', 'level21_button.png');
103         this.load.image('levelButton22', 'level22_button.png');
104         this.load.image('levelButton23', 'level23_button.png');
105         this.load.image('levelButton24', 'level24_button.png');
106         this.load.image('levelButton31', 'level31_button.png');
107         this.load.image('levelButton32', 'level32_button.png');
108         this.load.image('levelButton33', 'level33_button.png');
109         this.load.image('levelButton34', 'level34_button.png');
110     }
111
112     /**
113      * Method for preloading all introduction files
114      */
115     private preLoadIntroFiles(): void {
116         this.load.setPath('assets/introduction/');
117         this.load.image('intro_set', 'intro_set.png');
118         this.load.spritesheet('intro_sorting', 'intro_sorting.png', {
                frameWidth: 336, frameHeight: 189, endFrame: 150});
119         this.load.spritesheet('intro_falling', 'intro_falling.png', {
```

```
            frameWidth: 336, frameHeight: 189, endFrame: 68});
120         this.load.spritesheet('intro_restricted', 'intro_restricted.png'
                , {frameWidth: 336, frameHeight: 189, endFrame: 201});
121         this.load.spritesheet('intro_set_easy', 'intro_set_easy.png', {
                frameWidth: 336, frameHeight: 189, endFrame: 225});
122         this.load.spritesheet('intro_set_hard', 'intro_set_hard.png', {
                frameWidth: 336, frameHeight: 189, endFrame: 68});
123     }
124
125     /**
126      * Method for preloading all audio files
127      */
128     private preLoadAudio(): void {
129         this.load.setPath('assets/ui_audio/');
130         this.load.audio('back', 'back.mp3');
131         this.load.audio('battle', 'battle.mp3');
132         this.load.audio('exploration', 'exploration.mp3');
133         this.load.audio('fun', 'fun.mp3');
134         this.load.audio('loading', 'loading.mp3');
135         this.load.audio('lose', 'lose.mp3');
136         this.load.audio('pause', 'pause.mp3');
137         this.load.audio('select', 'select.mp3');
138         this.load.audio('space', 'space.mp3');
139         this.load.audio('sparkle', 'sparkle.mp3');
140         this.load.audio('welcome', 'welcome.mp3');
141         this.load.audio('win', 'win.mp3');
142     }
143
144     /**
145      * Method for initializing the loading graphics/animation
146      */
147     private initLoadingGraphics(): void {
148         // Loading graphics
149         const cogwheel: Phaser.GameObjects.Sprite = this.add.sprite(1/2*
                this.cameras.main.width, 1/3*this.cameras.main.height, '
                cogwheel');
150         cogwheel.setOrigin(0.5, 0.5);
151         const scale: number = this.imageScalingFactor(1/3*Math.min(this.
                cameras.main.height, this.cameras.main.width), cogwheel.
                width, cogwheel.height);
152         cogwheel.setScale(scale);
153
154         const cogTween: Phaser.Tweens.Tween = this.tweens.add({
155             targets: cogwheel,
156             angle: 360,
157             ease: 'Linear',
158             repeat: -1,
159             duration: 5000
160         });
161
162         // Progress bar
163         const progress = this.add.graphics();
164
165         this.load.on('progress', function (value) {
166             progress.clear();
167             progress.fillStyle(0xffffff, 1);
```

```
168             progress.fillRect(20, 4/5*this.cameras.main.height, (this.
                    cameras.main.width - 40) * value, 1/10*this.cameras.main
                    .height);
169         }, this);
170     }
171
172     /**
173      * Method for initializing the loading and action on completion
174      */
175     private start(): void {
176         this.load.on('complete', function(){
177             this.sceneChange('DropDownMenu');
178         }, this);
179
180         this.load.start();
181     }
182
183     /**
184      * Method for initializing the background
185      */
186     private setBackground(): void {
187         let background = this.add.sprite(0, 0, 'background1');
188         background.setOrigin(0, 0);
189         background.setDisplaySize(this.cameras.main.width, this.cameras.
                main.height);
190         background.setAlpha(0.3);
191     }
192 }
```

A.3. dropDownMenu.ts

Listing A.3: *dropDownMenu.ts*

```
1  import 'phaser';
2  import {BaseScene} from './baseScene';
3
4  export class DropDownMenu extends BaseScene {
5
6      /**
7       * Name of the paused scene
8       */
9      private key_paused_scene: string;
10
11     /**
12      * Lock for not messing up animations by clicking repeatedly without
            waiting for the animation to finish
13      */
14     private lock: boolean;
15
16     /**
17      * State of the drop down menu
18      */
19     private menuDown: boolean;
20
```

```
21      /**
22       * Size of the menu buttons
23       */
24      private buttonSize: number;
25
26      /**
27       * Menu button
28       */
29      private menuButton: Phaser.GameObjects.Sprite;
30
31      /**
32       * Exit button
33       */
34      private exitButton: Phaser.GameObjects.Sprite;
35
36      /**
37       * Fullscreen switch
38       */
39      private fullscreenButton: Phaser.GameObjects.Sprite;
40
41      /**
42       * Menu background
43       */
44      private menuBackground: Phaser.GameObjects.Sprite;
45
46      /**
47       * Pause background
48       */
49      private pauseBackground: Phaser.GameObjects.Container;
50
51      constructor() {
52          super('DropDownMenu');
53      }
54
55      init(): void {
56          // Initialize fields
57          this.pauseBackground = this.add.container(0, 0);
58          this.key_paused_scene = null;
59          this.lock = false;
60          this.menuDown = false;
61          this.buttonSize = 64;
62      }
63
64      preload(): void {
65
66      }
67
68      create(): void {
69          this.setPixelScreen();
70          this.setMenu();
71          this.initInput();
72      }
73
74      update(time: number, delta: number): void {
75
76      }
```

```
77
78      /**
79       * Method for initializing the menu buttons, background and action
80       */
81      private setMenu(): void {
82          let scale: number;
83
84          // Menubackground
85          this.menuBackground = this.add.sprite(10 + 64 + 25, 100, '
                menubackground');
86          this.menuBackground.setOrigin(1, 1);
87          this.menuBackground.setDisplaySize(200, 80 * 5);
88          this.menuBackground.setTint(0xeeeeee);
89
90          // ExitButton
91          this.exitButton = this.add.sprite(-64, 10 + 32 + 2 * (10 + 64),
                'exitbutton');
92          this.exitButton.setOrigin(0.5, 0.5);
93          this.exitButton.setName("exitButton");
94          this.exitButton.setData('clicked', false);
95
96          scale = this.imageScalingFactor(this.buttonSize, this.exitButton
                .width, this.exitButton.height);
97          this.exitButton.setScale(scale);
98
99          this.exitButton.setInteractive({cursor: 'pointer'});
100
101         // Fullscreen Button
102         this.fullscreenButton = this.add.sprite(-64, 10 + 32 + (10 + 64)
                , 'fullscreenbuttonblack', 0);
103         this.fullscreenButton.setOrigin(0.5, 0.5);
104
105         scale = this.imageScalingFactor(this.buttonSize, this.
                fullscreenButton.width, this.fullscreenButton.height);
106         this.fullscreenButton.setScale(scale);
107
108         this.fullscreenButton.setName("fullscreenButton");
109         this.fullscreenButton.setData('clicked', false);
110
111         this.fullscreenButton.setInteractive({cursor: 'pointer'});
112
113         // Enable key F for enabling/disabling fullscreen
114         const FKey: Phaser.Input.Keyboard.Key = this.input.keyboard.
                addKey('F');
115         FKey.on('down', function () {
116             this.scale.toggleFullscreen();
117             if (this.scale.isFullscreen) {
118                 this.fullscreenButton.setFrame(0);
119             } else {
120                 this.fullscreenButton.setFrame(1);
121             }
122
123         }, this);
124
125         // MenuButton
126         this.menuButton = this.add.sprite(32 + 10, 10 + 32, 'menubutton'
```

```
            );
127         this.menuButton.setOrigin(0.5, 0.5);
128
129         scale = this.imageScalingFactor(this.buttonSize, this.menuButton
                .width, this.menuButton.height);
130         this.menuButton.setScale(scale);
131
132         this.menuButton.setName("menuButton");
133         this.menuButton.setData('clicked', false);
134
135         this.menuButton.setInteractive({cursor: 'pointer'});
136
137         // StartGame
138         this.game.scene.start('WelcomeScene');
139     }
140
141     /**
142      * Method for initializing all input
143      */
144     private initInput(): void {
145         this.input.on('pointerdown', function (pointer, currentlyOver) {
146             const gameObject: any = currentlyOver[0];
147             if (gameObject instanceof Phaser.GameObjects.Sprite) {
148                 gameObject.setData('clicked', true);
149             }
150         }, this);
151
152         this.input.on('pointerup', function (pointer, currentlyOver) {
153             const gameObject: any = currentlyOver[0];
154             if (gameObject instanceof Phaser.GameObjects.Sprite &&
                    gameObject.getData('clicked')) {
155                 this.buttonFunction(gameObject);
156             }
157
158             this.menuButton.setData("clicked", false);
159             this.exitButton.setData("clicked", false);
160             this.fullscreenButton.setData("clicked", false);
161
162         }, this);
163     }
164
165     /**
166      * Method for assigning each button an event function
167      * @param gameObject GameObject on which you want the function on
168      */
169     private buttonFunction(gameObject: Phaser.GameObjects.Sprite): void
            {
170         switch (gameObject.name) {
171             case 'menuButton': {
172                 if (!this.lock) {
173                     // Acquire lock
174                     this.lock = true;
175                     this.menuAction();
176                 }
177                 break;
178             }
```

```
179
180             case 'fullscreenButton': {
181                 this.scale.toggleFullscreen();
182
183                 if (this.scale.isFullscreen) {
184                     gameObject.setFrame(0);
185                 } else {
186                     gameObject.setFrame(1);
187                 }
188                 break;
189             }
190
191             case 'exitButton': {
192                 this.menuAction();
193
194                 this.game.scene.getScenes(false).forEach(function (scene
                        ) {
195                     // @ts-ignore
196                     const sceneKey: string = scene.key;
197                     if (!(sceneKey === this.getKey()) && (this.game.
                            scene.isActive(sceneKey) || this.game.scene.
                            isPaused(sceneKey))) {
198                         scene.sound.stopAll();
199                         this.game.scene.stop(sceneKey);
200                     }
201                 }, this);
202
203                 if (this.key_paused_scene === 'LevelMenuScene' || this.
                        key_paused_scene === 'WelcomeScene') {
204                     this.game.scene.start('WelcomeScene');
205                 } else {
206                     this.game.scene.start('LevelMenuScene');
207                 }
208                 break;
209             }
210
211             default: {
212                 break;
213             }
214         }
215     }
216
217     /**
218      * Method which defines the graphical behaviour of the drop down
            menu
219      */
220     private menuAction(): void {
221
222         if (this.menuDown) {
223             // Animation
224             const menuButtonTween: Phaser.Tweens.Tween = this.tweens.add
                    ({
225                 targets: this.menuButton,
226                 angle: 0,
227                 ease: 'Cubic',
228                 duration: 700,
```

```
229                 onComplete: () => this.lock = false
230             });
231
232             const menuBackgroundTween: Phaser.Tweens.Tween = this.tweens
                    .add({
233                 targets: this.menuBackground,
234                 y: 100,
235                 ease: 'Cubic',
236                 duration: 500,
237                 delay: 200
238             });
239
240             const fullscreenButtonTween: Phaser.Tweens.Tween = this.
                    tweens.add({
241                 targets: this.fullscreenButton,
242                 x: -64,
243                 ease: 'Cubic',
244                 duration: 500,
245                 delay: 100
246             });
247
248             const exitButtonTween: Phaser.Tweens.Tween = this.tweens.add
                    ({
249                 targets: this.exitButton,
250                 x: -64,
251                 ease: 'Cubic',
252                 duration: 500
253             });
254
255             const pixelScreenTween: Phaser.Tweens.Tween = this.tweens.
                    add({
256                 targets: this.pauseBackground.getAll(),
257                 alpha: 0,
258                 ease: 'linear',
259                 duration: 700,
260                 delay: 0,
261                 onComplete: () => this.pauseBackground.setVisible(false)
262             });
263
264             this.menuDown = false;
265
266             // Resume current scene
267             this.game.scene.resume(this.key_paused_scene);
268
269         } else {
270             // Pause current scene
271             this.sound.add('pause').play();
272
273             // @ts-ignore
274             const key_paused_scene: string = this.game.scene.getScenes(
                    true)[0].key;
275             this.game.scene.pause(key_paused_scene);
276             this.key_paused_scene = key_paused_scene;
277
278             // Animation
279             const menuButtonTween: Phaser.Tweens.Tween = this.tweens.add
```

```
                ({
280                 targets: this.menuButton,
281                 angle: -90,
282                 ease: 'Cubic',
283                 duration: 700,
284                 onComplete: () => this.lock = false,
285             });
286
287             const menuBackgroundTween: Phaser.Tweens.Tween = this.tweens
                    .add({
288                 targets: this.menuBackground,
289                 y: 3 * (64 + 10) + 30,
290                 ease: 'Cubic',
291                 duration: 600
292             });
293
294             const fullscreenButtonTween: Phaser.Tweens.Tween = this.
                    tweens.add({
295                 targets: this.fullscreenButton,
296                 x: 10 + 32,
297                 ease: 'Cubic',
298                 duration: 500,
299                 delay: 100,
300             });
301
302             const exitButtonTween: Phaser.Tweens.Tween = this.tweens.add
                    ({
303                 targets: this.exitButton,
304                 x: 10 + 32,
305                 ease: 'Cubic',
306                 duration: 500,
307                 delay: 200
308             });
309
310             this.pauseBackground.setVisible(true);
311
312             const pixelScreenTween: Phaser.Tweens.Tween = this.tweens.
                    add({
313                 targets: this.pauseBackground.getAll(),
314                 alpha: 0.9,
315                 ease: 'linear',
316                 duration: 700,
317                 delay: 0
318             });
319
320             this.menuDown = true;
321         }
322     }
323
324     /**
325      * Method for setting the pixeled overlay and the hourglass
326      */
327     private setPixelScreen(): void {
328         const pixelScreen: Phaser.GameObjects.Grid = this.add.grid(0, 0,
                this.cameras.main.width, this.cameras.main.height, this.
               cameras.main.width / 100, this.cameras.main.width / 100);
```

```
329         pixelScreen.setOrigin(0, 0);
330         pixelScreen.setFillStyle(0x777777);
331         pixelScreen.setAltFillStyle(0x555555);
332         pixelScreen.setOutlineStyle(0x555555);
333         pixelScreen.setAlpha(0);
334         this.pauseBackground.add(pixelScreen);
335
336         const hourglass: Phaser.GameObjects.Sprite = this.add.sprite(
                this.cameras.main.width / 2, this.cameras.main.height / 2, '
                hourglass');
337         hourglass.setOrigin(0.5, 0.5);
338
339         const clockScale: number = this.imageScalingFactor(3 / 5 * this.
                cameras.main.height, hourglass.width, hourglass.height);
340         hourglass.setScale(clockScale);
341         hourglass.setAlpha(0);
342
343         this.pauseBackground.add(hourglass);
344
345         this.pauseBackground.setVisible(false);
346         this.children.sendToBack(this.pauseBackground);
347     }
348 }
```

A.4. introScene.ts

Listing A.4: *introScene.ts*

```
1  import 'phaser';
2  import {BaseScene} from './baseScene';
3  import AnimationManager = Phaser.Animations.AnimationManager;
4
5  export class IntroScene extends BaseScene {
6
7      /**
8       * Name of the paused scene the intro is currently playing for
9       */
10     private pausedScene: string;
11
12     /**
13      * Boolean, indicating if the paused scene should be resumed
14      */
15     private resume: boolean;
16
17     constructor() {
18         super('IntroScene');
19     }
20
21     init(data): void {
22         this.pausedScene = data.pausedScene;
23         this.level = data.level;
24         this.resume = false;
25     }
26
```

```
27      preload(): void {
28
29      }
30
31      create(): void {
32          // Bring MenuUI to the front and initialize transition
33          this.game.scene.sendToBack(this.getKey());
34          this.game.scene.moveUp(this.getKey());
35
36          if (this.getIntroData() == null) {
37              this.scene.resume(this.pausedScene);
38              this.scene.stop(this.getKey());
39          } else {
40              this.initIntro();
41          }
42
43      }
44
45      /**
46       * Method for initializing the introduction objects and animation
47       */
48      private initIntro(): void {
49          const background: Phaser.GameObjects.Rectangle = this.
                setBackground();
50          const intro: Phaser.GameObjects.Sprite = this.setIntro();
51
52          const finger: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width/2, this.cameras.main.height-10, 'finger')
                ;
53          const scale: number = this.imageScalingFactor(this.cameras.main.
                height/10, finger.width, finger.height);
54          finger.setScale(scale);
55          finger.setOrigin(0.5, 1);
56
57          if (this.pausedScene === "GameScene"){
58              intro.setX(3/4*this.cameras.main.width);
59              const intro1scale: number = this.imageScalingFactor(this.
                    cameras.main.width/2, intro.width, intro.height);
60              intro.setScale(intro1scale);
61
62              const intro2: Phaser.GameObjects.Sprite = this.add.sprite
                    (1/4*this.cameras.main.width, 0, 'intro_set');
63              intro2.setOrigin(0.5, 0.5);
64              const intro2scale: number = this.imageScalingFactor(this.
                    cameras.main.width/2, intro2.width, intro2.height);
65              intro2.setScale(intro2scale);
66
67              this.setAnimation(intro, background, finger, intro2);
68          } else {
69              this.setAnimation(intro, background, finger);
70          }
71      }
72
73      /**
74       * Method for the input setup
75       * @param intro Introduction image
```

```
76        * @param background Background image
77        * @param finger Finger image
78        * @param intro2 Introduction 2 image
79        */
80       private initInput(intro: Phaser.GameObjects.Sprite, background:
             Phaser.GameObjects.Rectangle, finger: Phaser.GameObjects.Sprite,
              intro2?: Phaser.GameObjects.Sprite): void {
81           this.input.on('pointerdown', function () {
82               this.input.on('pointerup', function () {
83                   this.resume = true;
84                   const introTweenOut: Phaser.Tweens.Tween = this.tweens.
                         add({
85                       targets: intro,
86                       y: -300,
87                       ease: 'linear',
88                       duration: 200,
89                       onComplete: function () {
90                           this.scene.resume(this.pausedScene);
91                           this.scene.stop(this.getKey());
92                       }.bind(this)
93                   });
94
95                   if (this.pausedScene === "GameScene") {
96                       const intro2TweenIn: Phaser.Tweens.Tween = this.
                             tweens.add({
97                           targets: intro2,
98                           y: -300,
99                           ease: 'linear',
100                          duration: 200
101                      });
102                  }
103
104                  const backgroundTweenOut: Phaser.Tweens.Tween = this.
                         tweens.add({
105                      targets: background,
106                      alpha: 0.01,
107                      ease: 'linear',
108                      duration: 200
109                  });
110
111                  const fingerTweenOut: Phaser.Tweens.Tween = this.tweens.
                         add({
112                      targets: finger,
113                      alpha: 0.01,
114                      ease: 'linear',
115                      duration: 200
116                  });
117
118              }, this);
119          }, this);
120      }
121
122      /**
123       * Method for generating the background object
124       */
125      private setBackground(): Phaser.GameObjects.Rectangle {
```

```
126         const background: Phaser.GameObjects.Rectangle = this.add.
                rectangle(0, 0, this.cameras.main.width, this.cameras.main.
                height);
127         background.setOrigin(0, 0);
128         background.setFillStyle(0x000000);
129         background.setAlpha(0.01);
130
131         return background;
132     }
133
134     /**
135      * Method for generating the introduction object
136      */
137     private setIntro(): Phaser.GameObjects.Sprite {
138         const data: [string, Phaser.Types.Animations.
                GenerateFrameNumbers] = this.getIntroData();
139
140         const introConfig = {
141             key: 'animateGif',
142             frames: this.anims.generateFrameNumbers(data[0], data[1]),
143             frameRate: 20,
144             repeat: -1
145         };
146
147         if(this.anims.exists(introConfig.key)) {
148             this.anims.remove(introConfig.key)
149         }
150
151         this.anims.create(introConfig);
152
153         const intro: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width / 2, this.cameras.main.height / 2, data
                [0]);
154         intro.setOrigin(0.5, 0.5);
155
156         const scale: number = this.imageScalingFactor(this.cameras.main.
                width * 4 / 5, intro.width, intro.height);
157         intro.setScale(scale);
158
159         intro.setY(0);
160
161         return intro;
162     }
163
164     /**
165      * Method for the animation setup
166      * @param intro The introduction gif/spritesheet
167      * @param background The background
168      * @param finger Finger image
169      * @param intro2 The 2nd introduction gif/spritesheet
170      */
171     private setAnimation(intro: Phaser.GameObjects.Sprite, background:
            Phaser.GameObjects.Rectangle, finger: Phaser.GameObjects.Sprite,
             intro2?: Phaser.GameObjects.Sprite) {
172         const introTweenIn: Phaser.Tweens.Tween = this.tweens.add({
173             targets: intro,
```

```
174             y: this.cameras.main.height / 2,
175             ease: 'linear',
176             duration: 500,
177             onComplete: function(){
178                 if (this.pausedScene === "GameScene") {
179                     this.initInput(intro, background, finger, intro2)
180                 } else {
181                     this.initInput(intro, background, finger)
182                 }
183             }.bind(this)
184         });
185 
186         if (this.pausedScene === "GameScene") {
187             const intro2TweenIn: Phaser.Tweens.Tween = this.tweens.add({
188                 targets: intro2,
189                 y: this.cameras.main.height / 2,
190                 ease: 'linear',
191                 duration: 500
192             });
193         }
194 
195         const backgroundTweenIn: Phaser.Tweens.Tween = this.tweens.add({
196             targets: background,
197             alpha: 0.8,
198             ease: 'linear',
199             duration: 500
200         });
201 
202         const fingerTween: Phaser.Tweens.Tween = this.tweens.add({
203             targets: finger,
204             alpha: 0.1,
205             ease: 'Linear',
206             repeat: 1000,
207             yoyo: true,
208             duration: 1000
209         });
210 
211         intro.play('animateGif');
212     }
213 
214     /**
215      * Method for retrieving the correct introduction for the current
            scene
216      */
217     private getIntroData(): [string, Phaser.Types.Animations.
            GenerateFrameNumbers] {
218         let ret: [string, Phaser.Types.Animations.GenerateFrameNumbers];
219 
220         switch (this.pausedScene + String(this.getLevel())) {
221             case 'PropertySortingScene1': {
222                 ret = ['intro_sorting', {start: 0, end: 150, first:
                        150}];
223                 break;
224             }
225 
226             case 'PropertySortingScene2': {
```

```
227                 ret = ['intro_sorting', {start: 0, end: 150, first:
                        150}];
228                 break;
229             }
230
231             case 'PropertySortingScene3': {
232                 ret = ['intro_sorting', {start: 0, end: 150, first:
                        150}];
233                 break;
234             }
235
236             case 'PropertySortingScene4': {
237                 ret = ['intro_sorting', {start: 0, end: 150, first:
                        150}];
238                 break;
239             }
240
241             case 'PropertySortingScene5': {
242                 ret = ['intro_falling', {start: 0, end: 68, first: 68}];
243                 break;
244             }
245
246             case 'PropertySortingScene6': {
247                 ret = ['intro_falling', {start: 0, end: 68, first: 68}];
248                 break;
249             }
250
251             case 'PropertySortingScene7': {
252                 ret = ['intro_falling', {start: 0, end: 68, first: 68}];
253                 break;
254             }
255
256             case 'PropertySortingScene8': {
257                 ret = ['intro_falling', {start: 0, end: 68, first: 68}];
258                 break;
259             }
260
261             case 'RestrictedSortingScene1': {
262                 ret = ['intro_restricted', {start: 0, end: 201, first:
                        201}];
263                 break;
264             }
265
266             case 'RestrictedSortingScene2': {
267                 ret = ['intro_restricted', {start: 0, end: 201, first:
                        201}];
268                 break;
269             }
270
271             case 'GameScene1': {
272                 ret = ['intro_set_easy', {start: 0, end: 225, first:
                        225}];
273                 break;
274             }
275
276             case 'GameScene2': {
```

```
277                 ret = ['intro_set_hard', {start: 0, end: 68, first:
                        68}];
278                 break;
279             }
280
281             default: {
282                 ret = null;
283                 break;
284             }
285         }
286         return ret;
287     }
288 }
```

A.5. welcomeScene.ts

```
1  import 'phaser';
2  import {BaseScene} from './baseScene';
3
4  export class WelcomeScene extends BaseScene {
5
6      constructor() {
7          super('WelcomeScene');
8      }
9
10     init(): void {
11
12     }
13
14     preload(): void {
15
16     }
17
18     create(): void {
19         // Bring MenuUI to the front and initialize transition
20         this.game.scene.sendToBack(this.getKey());
21         this.transitionIn();
22
23         this.setBackground();
24         this.setTitle();
25         this.initInput();
26         this.initAudio();
27     }
28
29     /**
30      * Method for initializing the background
31      */
32     private setBackground(): void {
33         let background = this.add.sprite(0, 0, 'background1');
34         background.setOrigin(0, 0);
35         background.setDisplaySize(this.cameras.main.width, this.cameras.
               main.height);
36         background.setInteractive({ cursor: 'pointer' });
37     }
```

```
38
39      /**
40       * Method for initializing title and animation
41       */
42      private setTitle(): void {
43          // Add title
44          const title: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width / 2, this.cameras.main.height / 2, 'title
                ');
45          const titleScale: number = this.imageScalingFactor(4/6*this.
                cameras.main.width, title.width, title.height);
46          title.setOrigin(0.5, 0.5);
47          title.setScale(titleScale);
48          title.setInteractive({ cursor: 'pointer' });
49
50          const titleTween: Phaser.Tweens.Tween = this.tweens.add({
51              targets: title,
52              alpha: 0.7,
53              ease: 'Linear',
54              repeat: 1000,
55              yoyo: true,
56              duration: 1000
57          });
58
59          // Add finger
60          const finger: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width / 2, 3/4*this.cameras.main.height, '
                finger');
61          const fingerScale: number = this.imageScalingFactor(1/6*this.
                cameras.main.height, finger.width, finger.height, true);
62          finger.setOrigin(0.5, 0.5);
63          finger.setScale(fingerScale);
64          finger.setInteractive({ cursor: 'pointer' });
65
66          const fingerTween: Phaser.Tweens.Tween = this.tweens.add({
67              targets: finger,
68              alpha: 0.1,
69              ease: 'Linear',
70              repeat: 1000,
71              yoyo: true,
72              duration: 1000
73          });
74      }
75
76      /**
77       * Method for initializing event actions
78       */
79      private initInput(): void {
80          this.input.on('pointerdown', function(){
81              this.input.on('pointerup', () => this.transitionOut('
                    LevelMenuScene'));
82          }, this);
83      }
84
85      /**
86       * Method for initializing sound effects
```

```
87          */
88         private initAudio(): void {
89             this.sound.add('welcome').play('', {loop: true});
90         }
91     }
```

A.6. levelMenuScene.ts

Listing A.5: *levelMenuScene.ts*

```
1  import 'phaser';
2  import {BaseScene} from './baseScene';
3
4  export class LevelMenuScene extends BaseScene {
5      /**
6       * Group with all level buttons
7       */
8      private levelButtons: Phaser.GameObjects.Group;
9
10     /**
11      * Global size of all buttons
12      */
13     private buttonSize: number;
14
15     constructor() {
16         super('LevelMenuScene');
17     }
18
19     init(): void {
20         // Initialize fields
21         this.levelButtons = this.add.group();
22
23         // Define button size dependant on the screen dimensions and the
               number of buttons
24         this.buttonSize = Math.min(this.cameras.main.width / (4 + 2),
               this.cameras.main.height / (3 + 2));
25     }
26
27     preload(): void {
28
29     }
30
31     create(): void {
32         // Bring MenuUI to the front and initialize transition
33         this.game.scene.sendToBack(this.getKey());
34         this.transitionIn();
35
36         this.setBackground();
37         this.setTitle();
38         this.setVisualLink();
39         this.setLevelButtons();
40         this.setStars();
41         this.initInput();
42         this.initAudio();
```

```
43      }
44
45      update(time: number): void {
46
47      }
48
49      /**
50       * Method for initializing background graphics
51       */
52      private setBackground(): void {
53          const background: Phaser.GameObjects.Sprite = this.add.sprite(0,
                0, 'background1');
54          background.setOrigin(0, 0);
55          background.setDisplaySize(this.cameras.main.width, this.cameras.
               main.height);
56      }
57
58      /**
59       * Method for initializing the level buttons and their onclick
            action
60       */
61      private setLevelButtons(): void {
62          const catButton: Phaser.GameObjects.Sprite = this.add.sprite(20,
                this.cameras.main.height - 20, 'catButton');
63          const eraseButton: Phaser.GameObjects.Sprite = this.add.sprite(
               this.cameras.main.width - 20, this.cameras.main.height - 20,
                'erase');
64          const levelButton11: Phaser.GameObjects.Sprite = this.add.sprite
               (1 / 5 * this.cameras.main.width, 1 / 4 * this.cameras.main.
               height, 'levelButton11');
65          const levelButton12: Phaser.GameObjects.Sprite = this.add.sprite
               (2 / 5 * this.cameras.main.width, 1 / 4 * this.cameras.main.
               height, 'levelButton12');
66          const levelButton13: Phaser.GameObjects.Sprite = this.add.sprite
               (3 / 5 * this.cameras.main.width, 1 / 4 * this.cameras.main.
               height, 'levelButton13');
67          const levelButton14: Phaser.GameObjects.Sprite = this.add.sprite
               (4 / 5 * this.cameras.main.width, 1 / 4 * this.cameras.main.
               height, 'levelButton14');
68          const levelButton21: Phaser.GameObjects.Sprite = this.add.sprite
               (4 / 5 * this.cameras.main.width, 2 / 4 * this.cameras.main.
               height, 'levelButton21');
69          const levelButton22: Phaser.GameObjects.Sprite = this.add.sprite
               (3 / 5 * this.cameras.main.width, 2 / 4 * this.cameras.main.
               height, 'levelButton22');
70          const levelButton23: Phaser.GameObjects.Sprite = this.add.sprite
               (2 / 5 * this.cameras.main.width, 2 / 4 * this.cameras.main.
               height, 'levelButton23');
71          const levelButton24: Phaser.GameObjects.Sprite = this.add.sprite
               (1 / 5 * this.cameras.main.width, 2 / 4 * this.cameras.main.
               height, 'levelButton24');
72          const levelButton31: Phaser.GameObjects.Sprite = this.add.sprite
               (1 / 5 * this.cameras.main.width, 3 / 4 * this.cameras.main.
               height, 'levelButton31');
73          const levelButton32: Phaser.GameObjects.Sprite = this.add.sprite
               (2 / 5 * this.cameras.main.width, 3 / 4 * this.cameras.main.
```

```
            height, 'levelButton32');
74          const levelButton33: Phaser.GameObjects.Sprite = this.add.sprite
                (3 / 5 * this.cameras.main.width, 3 / 4 * this.cameras.main.
                height, 'levelButton33');
75          const levelButton34: Phaser.GameObjects.Sprite = this.add.sprite
                (4 / 5 * this.cameras.main.width, 3 / 4 * this.cameras.main.
                height, 'levelButton34');
76
77          this.levelButtons.addMultiple([
78              levelButton11,
79              levelButton12,
80              levelButton13,
81              levelButton14,
82              levelButton21,
83              levelButton22,
84              levelButton23,
85              levelButton24,
86              levelButton31,
87              levelButton32,
88              levelButton33,
89              levelButton34
90          ]);
91
92          catButton.setOrigin(0, 1);
93          eraseButton.setOrigin(1, 1);
94
95          const scaleCatButton: number = this.imageScalingFactor(this.
                buttonSize / 1.5, catButton.width, catButton.height);
96          catButton.setScale(scaleCatButton);
97          catButton.setName('catButton');
98          catButton.setInteractive({cursor: 'pointer'});
99
100         const scaleEraseButton: number = this.imageScalingFactor(this.
                buttonSize / 1.5, eraseButton.width, eraseButton.height);
101         eraseButton.setScale(scaleEraseButton);
102         eraseButton.setName('eraseButton');
103         eraseButton.setInteractive({cursor: 'pointer'});
104
105         this.levelButtons.getChildren().forEach(function(gameObject) {
106             if (gameObject instanceof Phaser.GameObjects.Sprite) {
107                 gameObject.setName(gameObject.texture.key);
108                 gameObject.setOrigin(0.5, 0.5);
109
110                 const scale: number = this.imageScalingFactor(this.
                        buttonSize, gameObject.width, gameObject.height);
111                 gameObject.setScale(scale);
112
113                 gameObject.setData('clicked', false);
114
115                 gameObject.setInteractive({cursor: 'pointer'});
116             }
117         }, this);
118
119         this.levelButtons.add(catButton);
120         this.levelButtons.add(eraseButton);
121     }
```

```
122
123     /**
124      * Method for initializing all input
125      */
126     private initInput(): void {
127         this.input.on('pointerdown', function(pointer, currentlyOver) {
128             const gameObject: any = currentlyOver[0];
129             if (gameObject instanceof Phaser.GameObjects.Sprite) {
130                 gameObject.setTint(0xcccccc);
131                 gameObject.setData('clicked', true);
132             }
133         }, this);
134
135         this.input.on('pointerup', function(pointer, currentlyOver) {
136             const gameObject: any = currentlyOver[0];
137
138             if (gameObject instanceof Phaser.GameObjects.Sprite &&
                    gameObject.getData('clicked')) {
139                 this.buttonFunction(gameObject);
140             }
141
142             this.levelButtons.getChildren().forEach(function(gameObject)
                    {
143                 if (gameObject instanceof Phaser.GameObjects.Sprite) {
144                     gameObject.clearTint();
145                     gameObject.setData('clicked', false);
146                 }
147             }, this);
148         }, this);
149     }
150
151     /**
152      * Method for assigning each button an event function
153      * @param gameObject GameObject on which you want the function on
154      */
155     private buttonFunction(gameObject: Phaser.GameObjects.Sprite): void
            {
156         let name: string = this.buttonToSceneMap(gameObject.name);
157         let level: number = Number(name[name.length-1]);
158
159         this.sound.add('select').play();
160
161         switch (gameObject.name) {
162             case 'catButton': {
163                 this.transitionOut(name);
164                 break;
165             }
166
167             case 'eraseButton': {
168                 this.setStars(true);
169                 break;
170             }
171
172             case 'levelButton11': {
173                 name = name.substring(0, name.length-1);
174                 this.transitionOut(name, {'level': level});
```

```
175                 break;
176             }
177
178             case 'levelButton12': {
179                 name = name.substring(0, name.length-1);
180                 this.transitionOut(name, {'level': level});
181                 break;
182             }
183
184             case 'levelButton13': {
185                 name = name.substring(0, name.length-1);
186                 this.transitionOut(name, {'level': level});
187                 break;
188             }
189
190             case 'levelButton14': {
191                 name = name.substring(0, name.length-1);
192                 this.transitionOut(name, {'level': level});
193                 break;
194             }
195
196             case 'levelButton21': {
197                 name = name.substring(0, name.length-1);
198                 this.transitionOut(name, {'level': level});
199                 break;
200             }
201
202             case 'levelButton22': {
203                 name = name.substring(0, name.length-1);
204                 this.transitionOut(name, {'level': level});
205                 break;
206             }
207
208             case 'levelButton23': {
209                 name = name.substring(0, name.length-1);
210                 this.transitionOut(name, {'level': level});
211                 break;
212             }
213
214             case 'levelButton24': {
215                 name = name.substring(0, name.length-1);
216                 this.transitionOut(name, {'level': level});
217                 break;
218             }
219
220             case 'levelButton31': {
221                 name = name.substring(0, name.length-1);
222                 this.transitionOut(name, {'level': level});
223                 break;
224             }
225
226             case 'levelButton32': {
227                 name = name.substring(0, name.length-1);
228                 this.transitionOut(name, {'level': level});
229                 break;
230             }
```

```
231
232             case 'levelButton33': {
233                 name = name.substring(0, name.length-1);
234                 this.transitionOut(name, {'level': level});
235                 break;
236             }
237
238             case 'levelButton34': {
239                 name = name.substring(0, name.length-1);
240                 this.transitionOut(name, {'level': level});
241                 break;
242             }
243
244             case 'title': {
245                 this.transitionOut(name);
246                 break;
247             }
248
249             default: {
250                 break;
251             }
252         }
253     }
254
255     /**
256      * Method for initializing title and animation
257      */
258     private setTitle(): void {
259         // Add title
260         const y: number = (this.cameras.main.height / 4 - this.
                buttonSize / 2) / 2;
261         const title: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width / 2, y, 'title');
262         const titleScale: number = this.imageScalingFactor(y * 1.3,
                title.width, title.height, true);
263         title.setOrigin(0.5, 0.5);
264         title.setScale(titleScale);
265         title.setName('title');
266         title.setInteractive({cursor: 'pointer'});
267     }
268
269     /**
270      * Method for initializing the dashed line under the level buttons
271      */
272     private setVisualLink(): void {
273         const alpha: number = 0.5;
274
275         // Add lines
276         const dashedLine1: Phaser.GameObjects.Grid = this.add.grid(this.
                cameras.main.width / 2, 1 / 4 * this.cameras.main.height, 3
                / 5 * this.cameras.main.width, this.buttonSize / 6, this.
                buttonSize / 10, this.buttonSize / 6);
277         dashedLine1.setFillStyle(0x000000);
278         dashedLine1.setOrigin(0.5, 0.5);
279         dashedLine1.setAltFillStyle(0x000000, 0);
280         dashedLine1.setAlpha(alpha);
```

```
281
282         const dashedLine2: Phaser.GameObjects.Grid = this.add.grid(this.
                cameras.main.width / 2, 2 / 4 * this.cameras.main.height, 3
                / 5 * this.cameras.main.width, this.buttonSize / 6, this.
                buttonSize / 10, this.buttonSize / 6);
283         dashedLine2.setFillStyle(0x000000);
284         dashedLine2.setOrigin(0.5, 0.5);
285         dashedLine2.setAltFillStyle(0x000000, 0);
286         dashedLine2.setAlpha(alpha);
287
288         const dashedLine3: Phaser.GameObjects.Grid = this.add.grid(this.
                cameras.main.width / 2, 3 / 4 * this.cameras.main.height, 3
                / 5 * this.cameras.main.width, this.buttonSize / 6, this.
                buttonSize / 10, this.buttonSize / 6);
289         dashedLine3.setFillStyle(0x000000);
290         dashedLine3.setOrigin(0.5, 0.5);
291         dashedLine3.setAltFillStyle(0x000000, 0);
292         dashedLine3.setAlpha(alpha);
293
294         // Connecting half circles
295         const circle12 = this.add.graphics();
296         circle12.lineStyle(this.buttonSize / 6, 0x000000, 1);
297         circle12.beginPath();
298         circle12.arc(4 / 5 * this.cameras.main.width + this.cameras.main
                .height / 12, 1 / 4 * this.cameras.main.height + this.
                cameras.main.height / 8, this.cameras.main.height / 8, -
                Math.PI/2, Math.PI/2, false);
299         circle12.strokePath();
300         circle12.setAlpha(alpha);
301
302         const circle23 = this.add.graphics();
303         circle23.lineStyle(this.buttonSize / 6, 0x000000, 1);
304         circle23.beginPath();
305         circle23.arc(1 / 5 * this.cameras.main.width - this.cameras.main
                .height / 12, 2 / 4 * this.cameras.main.height + this.
                cameras.main.height / 8, this.cameras.main.height / 8, -
                Math.PI/2, Math.PI/2, true);
306         circle23.strokePath();
307         circle23.setAlpha(alpha);
308
309
310         // Triangle for indicating starting point
311         const triSize: number = this.buttonSize/3;
312         const startX: number = 1 / 5 * this.cameras.main.width - this.
                cameras.main.height / 8 + triSize - 10;
313         const startY: number = 1 / 4 * this.cameras.main.height +
                triSize;
314         const startTriangle1: Phaser.GameObjects.Triangle = this.add.
                triangle(startX, startY, 0, 0,- triSize, - triSize, -
                triSize, triSize, 0x000000, 1);
315         const startTriangle2: Phaser.GameObjects.Triangle = this.add.
                triangle(startX - 1/2*triSize, startY, 0, 0,- triSize, -
                triSize, - triSize, triSize, 0x000000, 1);
316         const startTriangle3: Phaser.GameObjects.Triangle = this.add.
                triangle(startX - 2/2*triSize, startY, 0, 0,- triSize, -
                triSize, - triSize, triSize, 0x000000, 1);
```

```
317
318             startTriangle1.setAlpha(alpha);
319             startTriangle2.setAlpha(alpha);
320             startTriangle3.setAlpha(alpha);
321
322             // Orange Square indicating the final level
323             const size: number = this.buttonSize*1.2;
324             const bossField: Phaser.GameObjects.Graphics = this.add.graphics
                    ();
325             bossField.fillStyle(0xfa7500, 0.3);
326             bossField.fillRoundedRect(4 / 5 * this.cameras.main.width - size
                    /2, 3 / 4 * this.cameras.main.height - size/2, size, size,
                    this.buttonSize/20);
327
328             const bossTween: Phaser.Tweens.Tween = this.tweens.add({
329                 targets: bossField,
330                 alpha: 0.1,
331                 ease: 'Linear',
332                 repeat: 1000,
333                 yoyo: true,
334                 duration: 1000
335             });
336         }
337
338         /**
339          * Method for initializing sound effects
340          */
341         private initAudio(): void {
342             this.sound.add('loading').play('', {loop: true});
343         }
344
345         /**
346          * Method for retrieving scene the respective button leads to with
                level at the end of the string
347          * @param buttonName Name of the button
348          */
349         private buttonToSceneMap(buttonName: string): string {
350             let ret: string = "";
351
352             switch (buttonName) {
353                 case 'catButton': {
354                     ret = 'SortingScene';
355                     break;
356                 }
357                 case 'levelButton11': {
358                     ret = 'PropertySortingScene1';
359                     break;
360                 }
361
362                 case 'levelButton12': {
363                     ret = 'PropertySortingScene2';
364
365                     break;
366                 }
367
368                 case 'levelButton13': {
```

```
369                 ret = 'PropertySortingScene3';
370                 break;
371             }
372
373             case 'levelButton14': {
374                 ret = 'PropertySortingScene4';
375                 break;
376             }
377
378             case 'levelButton21': {
379                 ret = 'PropertySortingScene5';
380                 break;
381             }
382
383             case 'levelButton22': {
384                 ret = 'PropertySortingScene6';
385                 break;
386             }
387
388             case 'levelButton23': {
389                 ret = 'PropertySortingScene7';
390                 break;
391             }
392
393             case 'levelButton24': {
394                 ret = 'PropertySortingScene8';
395                 break;
396             }
397
398             case 'levelButton31': {
399                 ret = 'RestrictedSortingScene1';
400                 break;
401             }
402
403             case 'levelButton32': {
404                 ret = 'RestrictedSortingScene2';
405                 break;
406             }
407
408             case 'levelButton33': {
409                 ret = 'GameScene1';
410                 break;
411             }
412
413             case 'levelButton34': {
414                 ret = 'GameScene2';
415                 break;
416             }
417
418             case 'title': {
419                 ret = 'WelcomeScene';
420                 break;
421             }
422
423             default: {
424                 break;
```

```
425                 }
426             }
427             return ret;
428         }
429
430         /**
431          * Method for initializing or resetting the (previous/default) score
432          */
433         private setStars(reset: boolean = false): void {
434             this.levelButtons.getChildren().forEach(function(gameObject) {
435                 if (gameObject instanceof Phaser.GameObjects.Sprite &&
                        gameObject.name != "catButton" && gameObject.name != '
                        eraseButton') {
436                     let score: string = 'star_0';
437                     if (typeof(Storage) !== "undefined") {
438                         if(window.localStorage.getItem('phaser_score_' +
                                this.buttonToSceneMap(gameObject.name))){
439                             if (reset) {
440                                 window.localStorage.setItem('phaser_score_'
                                        + this.buttonToSceneMap(gameObject.name)
                                        , score);
441                             } else {
442                                 score = window.localStorage.getItem('
                                        phaser_score_' + this.buttonToSceneMap(
                                        gameObject.name));
443                             }
444                         }
445                     } else {
446                         console.log("Sorry! No Web Storage support...");
447                     }
448
449                     const star: Phaser.GameObjects.Sprite = this.add.sprite(
                            gameObject.getBottomCenter().x, gameObject.
                            getBottomCenter().y, score);
450                     const scale: number = this.imageScalingFactor((
                            gameObject.getTopRight().x - gameObject.getTopLeft()
                            .x)/2, star.width, star.height);
451                     star.setScale(scale);
452                     star.setOrigin(0.5, 0.5);
453                 }
454             }, this);
455         }
456 }
```

A.7. sortingScene.ts

Listing A.6: *sortingScene.ts*

```
1 import 'phaser';
2 import {BaseScene} from './baseScene';
3
4 export class SortingScene extends BaseScene {
5     /**
6      * Object database with all image names, image paths and image
```

```
7           properties
8           */
9       private jsonObject: any;
10
11      /**
12       * Preselected objects
13       */
14      private selectedObjects: any[];
15
16      /**
17       * Remaining stack of objects
18       */
19      private arrayStack: Phaser.GameObjects.Container;
20
21      /**
22       * All category objects
23       */
24      private arrayCategory: Phaser.GameObjects.Group;
25
26      /**
27       * Size of displayed objects
28       */
29      private objectDisplaySize: number;
30
31      /**
32       * Number of objects per property
33       */
34      private objectsPerProperty: number;
35
36      /**
37       * The size of buttons
38       */
39      private buttonSize: number;
40
41      constructor() {
42          super('SortingScene');
43      }
44
45      init(): void {
46
47          // Initialize data from previous scene
48          this.jsonObject = this.cache.json.get('objects');
49
50          // Initialize fields
51          this.arrayStack = this.add.container(0, 0);
52          this.arrayCategory = this.add.group();
53          this.objectDisplaySize = 100;
54          this.objectsPerProperty = 5;
55          this.buttonSize = 64;
56      }
57
58      preload(): void {
59      }
60
61      create(): void {
            // Bring MenuUI to the front and initialize transition
```

```
62          this.game.scene.sendToBack(this.getKey());
63          this.transitionIn();
64
65          this.imagePreSelection();
66          this.setBackground();
67          this.setControlBar();
68          this.loadGameObjects();
69          this.exitButton();
70          this.initInput();
71          this.initAudio();
72      }
73
74      update(): void {
75      }
76
77      /**
78       * Method for pre-selecting a subgroup of images so that every
             property of each category has X representatives.
79       */
80      private imagePreSelection(): void {
81          // Select an manageable amount of images to be displayed
82          const selectiveArray: any[] = [];
83          const originArray: any[] = [...this.jsonObject['images']];
84
85          // Select category to add
86          for (let category of this.jsonObject['categories']) {
87              const temporaryArray: any[] = [];
88
89              // Select property to add
90              for (let property of category['validElements']) {
91                  Phaser.Math.RND.shuffle(originArray);
92                  // Check how many are needed in already selected
                        elements
93                  let missing: number = this.objectsPerProperty;
94                  for (let selectedImage of selectiveArray) {
95                      if (missing <= 0) {
96                          break;
97                      }
98
99                      if (selectedImage[category.name] === property.name)
                            {
100                         missing--;
101                     }
102                 }
103
104                 // Add number of needed images per property
105                 for (let image of originArray) {
106                     if (missing <= 0) {
107                         break;
108                     }
109
110                     if (image[category.name] === property.name) {
111                         temporaryArray.push(image);
112                         missing--;
113                     }
114                 }
```

```
115
116                 for (let image of temporaryArray) {
117                     let index: number = originArray.indexOf(image, 0);
118                     if (index > -1) {
119                         originArray.splice(index, 1);
120                     }
121                 }
122             }
123             temporaryArray.forEach((x) => selectiveArray.push(x));
124         }
125
126         this.selectedObjects = selectiveArray;
127     }
128
129     /**
130      * Method for initializing the background
131      */
132     private setBackground(): void {
133         const background: Phaser.GameObjects.Sprite = this.add.sprite(0,
                0, 'background4');
134         background.setOrigin(0, 0);
135         background.setDisplaySize(this.cameras.main.width, this.cameras.
              main.height);
136         background.setTint(0xffccaa);
137         background.setAlpha(0.9);
138     }
139
140     /**
141      * Method for initializing the control bar
142      */
143     private setControlBar(): void {
144         const controlbar: Phaser.GameObjects.Sprite = this.add.sprite(
              this.cameras.main.width / 2, this.cameras.main.height, '
              menubackground');
145         controlbar.setOrigin(0.5, 0.5);
146         controlbar.setAngle(-90);
147         controlbar.setScale(0.13, 0.20);
148
149         // Category indicator
150         let x: number = this.cameras.main.width / 2 - (controlbar.height
                * 0.24) / 2;
151         let countCategories: number = 0;
152
153         // Find out how many categories not null exist
154         for (let cat of this.jsonObject['categories']) {
155             if (cat.url === null) {
156                 continue;
157             }
158             countCategories++;
159         }
160
161         for (let cat of this.jsonObject['categories']) {
162             if (cat.url === null) {
163                 continue;
164             }
165
```

```
166             const validElements: any[] = [...cat['validElements']];
167             validElements.forEach((object, index, array) => array[index]
                    = object.name);
168
169             x += (controlbar.height * 0.24) / (countCategories + 1);
170             const name: string = cat.name;
171             const sprite: Phaser.GameObjects.Sprite = this.add.sprite(x,
                    this.cameras.main.height - controlbar.width * 0.13 / 5,
                    name);
172             sprite.setName(name);
173             sprite.setOrigin(0.5, 0.5);
174
175             sprite.setData('validElements', validElements);
176
177             const scale: number = this.imageScalingFactor(this.
                   buttonSize, sprite.width, sprite.height);
178             sprite.setScale(scale);
179
180             sprite.setVisible(true);
181
182             this.arrayCategory.add(sprite);
183
184             sprite.setInteractive({ cursor: 'pointer' });
185
186             sprite.on('pointerdown', function() {
187                 for (let item of this.arrayCategory.getChildren()) {
188                     if (item instanceof Phaser.GameObjects.Sprite) {
189                         item.clearTint();
190                     }
191                 }
192
193                 if (sprite instanceof Phaser.GameObjects.Sprite) {
194                     sprite.setTintFill(0x8dfd59);
195                     this.orderObjects(sprite.name, sprite.getData('
                           validElements'));
196                 }
197             }, this);
198         }
199     }
200
201     /**
202      * Method for loading all game objects
203      */
204     private loadGameObjects(): void {
205         for (let image of this.selectedObjects) {
206
207             let size = this.objectDisplaySize;
208
209             const name = image.name;
210             const cat1 = image.cat1;
211             const cat2 = image.cat2;
212             const cat3 = image.cat3;
213             const cat4 = image.cat4;
214
215             const sprite: Phaser.GameObjects.Sprite = this.add.sprite(
                   Phaser.Math.RND.between(100 + size / 2, this.cameras.
```

```
                main.width - size / 2), Phaser.Math.RND.between(size /
                2, this.cameras.main.height - 100 - size / 2), name);
216             sprite.setOrigin(0.5, 0.5);
217             sprite.setAngle(Phaser.Math.RND.angle());
218             sprite.setVisible(true);
219
220             const scale: number = Math.min(size / sprite.height, size /
                    sprite.width);
221             sprite.setScale(scale);
222
223             sprite.setName(name);
224
225             sprite.setData('cat1', cat1);
226             sprite.setData('cat2', cat2);
227             sprite.setData('cat3', cat3);
228             sprite.setData('cat4', cat4);
229             sprite.setData('scale', scale);
230
231             sprite.setInteractive({ cursor: 'pointer' });
232
233             this.arrayStack.add(sprite);
234             this.arrayStack.bringToTop(sprite);
235
236         }
237         this.children.bringToTop(this.arrayStack);
238     }
239
240     /**
241      * Method which initializes all global input actions
242      */
243     private initInput(): void {
244         // On start dragging
245         this.input.setDraggable(this.arrayStack.getAll());
246
247         this.input.on('dragstart', function(pointer, gameObject) {
248             if (gameObject instanceof Phaser.GameObjects.Sprite) {
249                 // Bring gameObject to top
250                 this.arrayStack.bringToTop(gameObject);
251
252                 // Set visual effects
253                 gameObject.clearTint();
254                 gameObject.setTint(0x999999);
255
256                 let scale: number = gameObject.getData('scale')*1.2;
257                 gameObject.setScale(scale);
258             }
259         }, this);
260
261         // On stop dragging
262         this.input.on('dragend', function (pointer, gameObject, dropped)
                {
263             // If not dropped set default visual effects
264             if (!dropped && gameObject instanceof Phaser.GameObjects.
                    Sprite) {
265                 gameObject.clearTint();
266
```

```
267                 let scale: number = gameObject.getData('scale');
268                 gameObject.setScale(scale);
269 
270                 let x: number = gameObject.x;
271                 let y: number = gameObject.y;
272                 let dist: number = Math.sqrt(Math.pow(gameObject.width*
                        gameObject.getData('scale'), 2) + Math.pow(
                        gameObject.height*gameObject.getData('scale'), 2))
                        /2;
273 
274                 if (x < 0) {
275                     x = 0 + dist;
276                 }
277 
278                 if (y < 0) {
279                     y = 0 + dist;
280                 }
281 
282                 if (x > this.cameras.main.width) {
283                     x = this.cameras.main.width - dist;
284                 }
285 
286                 if (y > this.cameras.main.height) {
287                     y = this.cameras.main.height - dist;
288                 }
289 
290                 gameObject.setPosition(x, y);
291             }
292         }, this);
293 
294         // While dragging update coordinates
295         this.input.on('drag', function(pointer, gameObject, dragX, dragY
                ) {
296             if (gameObject instanceof Phaser.GameObjects.Sprite){
297                 gameObject.setPosition(dragX, dragY);
298             }
299         }, this);
300     }
301 
302     /**
303      * Method for adding the exit button
304      */
305     private exitButton(): void {
306         const exitButton: Phaser.GameObjects.Sprite = this.add.sprite
                (10, this.cameras.main.height - 10, 'return');
307         exitButton.setOrigin(0,1);
308         exitButton.setInteractive({ cursor: 'pointer' });
309 
310         const scale: number = this.imageScalingFactor(this.buttonSize
                *1.5, exitButton.width, exitButton.height);
311         exitButton.setScale(scale);
312 
313         exitButton.on('pointerdown', function() {
314             exitButton.on('pointerup', function() {
315                 this.sound.add('back').play();
316                 this.transitionOut("LevelMenuScene");
```

```
317             }, this);
318         }, this);
319     }
320
321     /**
322      * Method for ordering objects by the properties of a category in a
            grid
323      * @param categoryName Name of the category
324      * @param validElements Properties of this category
325      */
326     private orderObjects(categoryName: string, validElements: string[]):
            void {
327         this.arrayStack.each(function(gameObject) {
328             if (gameObject instanceof Phaser.GameObjects.Sprite) {
329                 let coords: number[] = this.returnQuad(validElements.
                        indexOf(gameObject.getData(categoryName)),
                        validElements.length);
330                 gameObject.setPosition(coords[0], coords[1]);
331             }
332         }, this);
333     }
334
335     /**
336      * Method for initializing sound effects
337      */
338     private initAudio(): void {
339         this.sound.add('space').play('', {loop: true});
340     }
341
342     /**
343      * Returns random coordinates in the requested quadrant of total
            quadrants.
344      * Starting by 0, from left to right then from top to bottom.
345      * IMPORTANT: Sprite origin has to be (0.5, 0.5)!
346      * @param quadrant Number of the quadrant
347      * @param quadrantType Number of quadrants
348      */
349     private returnQuad(quadrant: number, quadrantType: number): number[]
            {
350         let ret: number[] = null;
351
352         if (quadrant >= quadrantType) {
353             console.log('ERROR: quadrant >= quadrantType');
354             return ret;
355         }
356
357         const spriteSizeHalf: number = this.objectDisplaySize / 2 + 20;
358
359         const leftOffsite: number = 100;
360         const rightOffsite: number = 0;
361         const topOffsite: number = 0;
362         const bottomOffsite: number = 100;
363
364         // Has entries dependant of
365         const horizontal: number[] = [];
366
```

```
367         // Has numberOfLines + 1 entries
368         const vertical: number[] = [];
369
370         horizontal.push(leftOffsite);
371
372         vertical.push(topOffsite);
373
374         switch (quadrantType) {
375             case 3: {
376                 horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) / 3);
377                 horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) * 2 / 3);
378                 break;
379             }
380             case 4: {
381                 horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) / 2);
382                 vertical.push(topOffsite + (this.cameras.main.height -
                       topOffsite - bottomOffsite) / 2);
383                 break;
384             }
385             case 6: {
386                 horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) / 3);
387                 horizontal.push(leftOffsite + (this.cameras.main.width -
                        leftOffsite - rightOffsite) * 2 / 3);
388                 vertical.push(topOffsite + (this.cameras.main.height -
                       topOffsite - bottomOffsite) / 2);
389                 break;
390             }
391             default: {
392                 break;
393             }
394         }
395
396         horizontal.push(this.cameras.main.width - rightOffsite);
397         vertical.push(this.cameras.main.height - bottomOffsite);
398
399         switch (quadrantType) {
400             case 3: {
401                 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)];
402                 break;
403             }
404             case 4: {
405                 if (quadrant < 2) {
406                     ret = [Phaser.Math.RND.between(horizontal[quadrant]
                           + spriteSizeHalf, horizontal[quadrant + 1] -
                           spriteSizeHalf), Phaser.Math.RND.between(
                           vertical[0] + spriteSizeHalf, vertical[1] -
                           spriteSizeHalf)];
```

```
407                     } else {
408                         ret = [Phaser.Math.RND.between(horizontal[quadrant
                                \% 2] + spriteSizeHalf, horizontal[(quadrant \%
                                2) + 1] - spriteSizeHalf), Phaser.Math.RND.
                                between(vertical[1] + spriteSizeHalf, vertical
                                [2] - spriteSizeHalf)];
409
410                     }
411                     break;
412                 }
413                 case 6: {
414                     if (quadrant < 3) {
415                         ret = [Phaser.Math.RND.between(horizontal[quadrant]
                                + spriteSizeHalf, horizontal[quadrant + 1] -
                                spriteSizeHalf), Phaser.Math.RND.between(
                                vertical[0] + spriteSizeHalf, vertical[1] -
                                spriteSizeHalf)];
416                     } else {
417                         ret = [Phaser.Math.RND.between(horizontal[quadrant
                                \% 3] + spriteSizeHalf, horizontal[(quadrant \%
                                3) + 1] - spriteSizeHalf), Phaser.Math.RND.
                                between(vertical[1] + spriteSizeHalf, vertical
                                [2] - spriteSizeHalf)];
418                     }
419                     break;
420                 }
421                 default: {
422                     break;
423                 }
424             }
425             return ret;
426         }
427 }
```

A.8. propertySortingScene.ts

Listing A.7: *propertySortingScene.ts*

```
1  import 'phaser';
2  import {BaseScene} from './baseScene';
3  import {LevelMenuScene} from './levelMenuScene';
4
5  export class PropertySortingScene extends BaseScene {
6
7      /**
8       * Object database with all image names, image paths and image
            properties
9       */
10     private jsonObject: any;
11
12     /**
13      * Array index number +1 of category to sort
14      */
15     private setCat: number;
```

```
16
17      /**
18       * Should the object fall (true) or be static (false)
19       */
20      private infinite: boolean;
21
22      /**
23       * All loaded objects
24       */
25      private arrayStack: Phaser.GameObjects.Container;
26
27      /**
28       * All displayed interactive objects with NO velocity
29       */
30      private arrayStatic: Phaser.GameObjects.Group;
31
32      /**
33       * All displayed interactive objects WITH velocity
34       */
35      private arrayFalling: Phaser.GameObjects.Group;
36
37      /**
38       * All DROPPED displayed interactive objects
39       */
40      private arrayDropped: Phaser.GameObjects.Group;
41
42      /**
43       * All category objects (images)
44       */
45      private arrayCategory: Phaser.GameObjects.Group;
46
47      /**
48       * All drop zones
49       */
50      private arrayDropZone: Phaser.GameObjects.Group;
51
52      /**
53       * Display size of displayed interactive objects
54       */
55      private objectDisplaySize: number;
56
57      /**
58       * Object of the number of correct categorized objects
59       */
60      private correctBar: Phaser.GameObjects.Sprite;
61
62      /**
63       * Object of the number of incorrect categorized objects
64       */
65      private wrongBar: Phaser.GameObjects.Sprite;
66
67      /**
68       * Number of correct categorized objects
69       */
70      private correctCount: number;
71
```

```
72      /**
73       * Number of incorrect categorized objects
74       */
75      private wrongCount: number;
76
77      /**
78       * Amount of properties in the respective category
79       */
80      private propertyCount: number;
81
82      /**
83       * Maximum reachable points
84       */
85      private maxPoints: number;
86
87      /**
88       * Object falling speed
89       */
90      private velocity: number;
91
92      /**
93       * Last time an object was emitted to fall
94       */
95      private lastEmitTime: number;
96
97      /**
98       * Time until the next object starts to fall
99       */
100     private delay: number;
101
102     /**
103      * How many objects of each category - category count
104      */
105     private numberOfObjectsEach: number;
106
107     /**
108      * Array of preselected objects with name for faster loading
109      */
110     private selectedElements: any[];
111     private selectedElementsName: string[];
112
113     /**
114      * Amount of dummies in play
115      */
116     private numberOfDummies: number;
117
118     /**
119      * Scaling value of the dropped object
120      */
121     private droppedObjectScale: number;
122
123     constructor() {
124         super('PropertySortingScene');
125     }
126
127     init(data): void {
```

```
128
129         // Initialize data from previous scene
130         this.jsonObject = this.cache.json.get('objects');
131         this.level = data.level;
132
133         // Set level parameter
134         this.infinite = false;
135         this.setCat = this.level;
136
137         if (this.setCat > 4) {
138             this.infinite = true;
139             this.setCat -= 4;
140         }
141
142         // Initialize fields
143         this.arrayStack = this.add.container(0, 0);
144         this.arrayCategory = this.add.group();
145         this.arrayStatic = this.add.group();
146         this.arrayFalling = this.add.group();
147         this.arrayDropped = this.add.group();
148         this.arrayDropZone = this.add.group();
149
150         this.correctCount = 0;
151         this.wrongCount = 0;
152         this.numberOfDummies = 0;
153
154         this.selectedElements = [];
155         this.selectedElementsName = [];
156
157         let numberOfProperties = 0;
158         for (let property of this.jsonObject['categories'][this.setCat -
                1]['validElements']) {
159             numberOfProperties++;
160         }
161
162         // Randomization of amount of properties to sort
163         this.propertyCount = Phaser.Math.RND.between(3,
               numberOfProperties);
164
165         // Debatable initializations
166         this.objectDisplaySize = 100;
167         this.droppedObjectScale = 0.4;
168         this.velocity = 150;
169         this.lastEmitTime = 0;
170         this.delay = 1500;
171         this.numberOfObjectsEach = Phaser.Math.RND.between(3, 5);
172     }
173
174     preload(): void {
175
176     }
177
178     create(): void {
179         // Bring MenuUI to the front and initialize transition
180         this.game.scene.sendToBack(this.getKey());
181         this.transitionIn();
```

```
182
183         this.preselectObjects();
184         this.setBackground();
185         this.addProgressbar();
186         this.loadObjects();
187         this.setDropzones();
188         this.initInput();
189         this.initFirstDrop();
190         this.initAudio();
191     }
192
193     update(time: number): void {
194         // If infinite, emit object after a specified amount of time
195         if (this.infinite) {
196             let diff: number = time - this.lastEmitTime;
197             if (diff > this.delay) {
198                 this.lastEmitTime = time;
199                 if (this.delay > 300) {
200                     this.delay -= 20;
201                 }
202                 if (this.arrayStatic.getLength() != 0) {
203                     const sprite: Phaser.Physics.Arcade.Sprite = Phaser.
                            Math.RND.pick(this.arrayStatic.getChildren());
204                     this.arrayStack.bringToTop(sprite);
205                     this.arrayStatic.remove(sprite);
206                     sprite.setVelocityY(this.velocity);
207                     sprite.setAngularVelocity(sprite.getData('spin'));
208                 }
209             }
210         }
211     }
212
213     /**
214      * Method for initializing the background
215      */
216     private setBackground(): void {
217         let background: Phaser.GameObjects.Sprite;
218         if (this.infinite) {
219             background = this.add.sprite(0, 0, 'background3');
220         } else {
221             background = this.add.sprite(0, 0, 'background2');
222         }
223         background.setOrigin(0, 0);
224         background.setDisplaySize(this.cameras.main.width, this.cameras.
                main.height);
225         background.setTint(0xffccaa);
226         background.setAlpha(0.9);
227     }
228
229     /**
230      * Methods for initializing the drop zones
231      */
232     private setDropzones(): void {
233         const leftBound: number = this.correctBar.getTopRight().x + 10;
234         const stepSize: number = (this.cameras.main.width - leftBound) /
                (this.propertyCount);
```

```
235         const zoneWidth: number = (this.cameras.main.width - leftBound)
                / (this.selectedElements.length);
236         const crateSize: number = Math.min(this.objectDisplaySize * 2,
                zoneWidth);
237         let iteration: number = 0.5;
238
239         for (let property of this.selectedElements) {
240             // Add crate
241             const crate: Phaser.GameObjects.Sprite = this.add.sprite(
                    leftBound + stepSize * iteration, this.cameras.main.
                    height - crateSize / 2, 'wooden_crate');
242             crate.setOrigin(0.5, 0.5);
243
244             const imageScalingFactor: number = this.imageScalingFactor(
                    crateSize, crate.width, crate.height);
245             crate.setScale(imageScalingFactor);
246
247             // Add zone around crate
248             const zone: Phaser.GameObjects.Zone = this.add.zone(crate.x,
                     crate.y, zoneWidth, crate.height * imageScalingFactor +
                     this.objectDisplaySize);
249             zone.setOrigin(0.5, 0.5);
250             zone.setRectangleDropZone(zone.width, zone.height);
251             zone.setName(property.name);
252
253             this.arrayDropZone.add(zone);
254
255             iteration++;
256         }
257     }
258
259     /**
260      * Method which initializes all input actions
261      */
262     private initInput(): void {
263
264         // On dragstart
265         this.input.on('dragstart', function(pointer, gameObject) {
266             if (gameObject instanceof Phaser.Physics.Arcade.Sprite) {
267                 if (gameObject.getData('active')) {
268                     gameObject.setTint(0x999999);
269                 }
270                 gameObject.setVelocityY(0);
271                 gameObject.setAngularVelocity(0);
272                 const zoomSpriteScale: number = gameObject.getData('
                        scale') * 1.5;
273                 gameObject.setScale(zoomSpriteScale);
274                 this.arrayStack.bringToTop(gameObject);
275             }
276         }, this);
277
278         this.input.on('drag', function(pointer, gameObject, dragX, dragY
                ) {
279             if (gameObject instanceof Phaser.Physics.Arcade.Sprite) {
280                 gameObject.setPosition(dragX, dragY);
281             }
```

```
282         }, this);
283
284         // On stop dragging
285         this.input.on('dragend', function(pointer, gameObject, dropped)
                {
286             // If not dropped set default visual effects
287             if (!dropped && gameObject instanceof Phaser.Physics.Arcade.
                    Sprite) {
288                 if (gameObject.getData('active')) {
289                     gameObject.clearTint();
290                 }
291
292                 let scale: number = gameObject.getData('scale');
293                 gameObject.setScale(scale);
294
295                 if (this.infinite) {
296                     gameObject.setVelocityY(this.velocity);
297                     gameObject.setAngularVelocity(gameObject.getData('
                            spin'));
298                 }
299
300                 let x: number = gameObject.x;
301                 let y: number = gameObject.y;
302                 let dist: number = Math.sqrt(Math.pow(gameObject.width *
                         gameObject.getData('scale'), 2) + Math.pow(
                        gameObject.height * gameObject.getData('scale'), 2))
                         / 2;
303
304                 if (x < 0) {
305                     x = dist;
306                 }
307
308                 if (y < 0) {
309                     y = dist;
310                 }
311
312                 if (x > this.cameras.main.width) {
313                     x = this.cameras.main.width - dist;
314                 }
315
316                 if (y > this.cameras.main.height) {
317                     y = this.cameras.main.height - dist;
318                 }
319
320                 gameObject.setPosition(x, y);
321             }
322         }, this);
323
324         this.input.on('drop', function(pointer, gameObject, dropZone) {
325             if (gameObject instanceof Phaser.Physics.Arcade.Sprite &&
                    dropZone instanceof Phaser.GameObjects.Zone) {
326                 let coords: number[] = [gameObject.input.dragStartX,
                        gameObject.input.dragStartY];
327                 let scale: number = gameObject.getData('scale');
328                 let point: number = -1;
329
```

```
330                 if (gameObject.name === dropZone.name && gameObject.
                        getData('active')) {
331                     coords = [dropZone.x + dropZone.width * 0.15,
                            dropZone.y - dropZone.height * 0.2];
332
333                     scale = this.imageScalingFactor(Math.min(dropZone.
                            width, dropZone.height) * this.
                            droppedObjectScale, gameObject.width, gameObject
                            .height);
334
335                     this.arrayDropped.add(gameObject);
336
337                     gameObject.disableInteractive();
338                     gameObject.setImmovable(true);
339
340                     point = +1;
341
342                 } else {
343                     if (this.infinite) {
344                         gameObject.setVelocityY(this.velocity);
345                         gameObject.setAngularVelocity(gameObject.getData
                                ('spin'));
346                     }
347                 }
348
349                 this.updateProgressbar(point);
350                 if (gameObject.getData('active')) {
351                     gameObject.clearTint();
352                 }
353                 gameObject.setScale(scale);
354                 gameObject.setPosition(coords[0], coords[1]);
355             }
356         }, this);
357     }
358
359     /**
360      * Method for initializing all game objects
361      */
362     private loadObjects(): void {
363         this.arrayStack.setDepth(1);
364
365         for (let propImage of this.selectedElements) {
366
367             // Create 10 of each property
368             for (let i = 0; i < this.numberOfObjectsEach; i++) {
369                 // RND size
370                 const size: number = Phaser.Math.RND.between(this.
                        objectDisplaySize, this.objectDisplaySize * 1.3);
371
372                 const sprite: Phaser.Physics.Arcade.Sprite = this.
                        physics.add.sprite(Phaser.Math.RND.between(100 +
                        this.objectDisplaySize / 2, this.cameras.main.width
                        - this.objectDisplaySize / 1.5), Phaser.Math.RND.
                        between(this.objectDisplaySize / 2, this.cameras.
                        main.height - this.objectDisplaySize * 2 - this.
                        objectDisplaySize / 2), this.selectedElementsName[
```

```
                    this.selectedElements.indexOf(propImage)]);
373                 sprite.setName(propImage.name);
374
375                 if (this.infinite) {
376                     sprite.setY(0 - 2 * this.objectDisplaySize);
377                 }
378
379                 sprite.setVelocity(0, 0);
380                 sprite.setOrigin(0.5, 0.5);
381
382                 // RND spin
383                 sprite.setAngle(Phaser.Math.RND.angle());
384
385                 sprite.setVisible(true);
386
387                 const spriteScale: number = this.imageScalingFactor(size
                        , sprite.width, sprite.height);
388                 sprite.setScale(spriteScale);
389
390                 sprite.setData('scale', spriteScale);
391                 sprite.setData('spin', Phaser.Math.RND.between(10, 50));
392                 sprite.setData('active', true);
393
394                 sprite.setInteractive({cursor: 'pointer'});
395
396                 this.arrayStatic.add(sprite);
397                 this.arrayStack.add(sprite);
398
399                 this.input.setDraggable(sprite);
400             }
401
402             let rndDummy: number = 0;
403
404             if (this.infinite) {
405                 rndDummy = Phaser.Math.RND.between(0, this.
                        numberOfObjectsEach / 2);
406             }
407
408             this.numberOfDummies += rndDummy;
409
410             for (let i = 0; i < rndDummy; i++) {
411                 // RND size
412                 const size: number = Phaser.Math.RND.between(this.
                        objectDisplaySize * 0.8, this.objectDisplaySize *
                        1.3);
413
414                 const sprite: Phaser.Physics.Arcade.Sprite = this.
                        physics.add.sprite(Phaser.Math.RND.between(100 +
                        this.objectDisplaySize / 2, this.cameras.main.width
                        - this.objectDisplaySize / 2), Phaser.Math.RND.
                        between(this.objectDisplaySize / 2, this.cameras.
                        main.height - this.objectDisplaySize * 2 - this.
                        objectDisplaySize / 2), this.selectedElementsName[
                        this.selectedElements.indexOf(propImage)]);
415                 sprite.setName(propImage.name);
416
```

```
417                 if (this.infinite) {
418                     sprite.setY(0 - 2 * this.objectDisplaySize);
419                 }
420
421                 sprite.setVelocity(0, 0);
422                 sprite.setOrigin(0.5, 0.5);
423
424                 sprite.setTintFill(0xffffff);
425
426                 // RND spin
427                 sprite.setAngle(Phaser.Math.RND.angle());
428
429                 sprite.setVisible(true);
430
431                 const spriteScale: number = this.imageScalingFactor(size
                        , sprite.width, sprite.height);
432                 sprite.setScale(spriteScale);
433
434                 sprite.setData('scale', spriteScale);
435                 sprite.setData('spin', Phaser.Math.RND.between(10, 50));
436                 sprite.setData('active', false);
437
438                 sprite.setInteractive({cursor: 'pointer'});
439
440                 this.arrayStatic.add(sprite);
441                 this.arrayStack.add(sprite);
442
443                 this.input.setDraggable(sprite);
444             }
445         }
446
447         this.maxPoints = this.arrayStack.length - this.propertyCount -
                this.numberOfDummies;
448
449         const floor: Phaser.Physics.Arcade.Sprite = this.physics.add.
                sprite(0, this.cameras.main.height + 2 * this.
                objectDisplaySize, 'background');
450         floor.setDisplaySize(this.cameras.main.width, 1);
451         floor.setTintFill(0x000000);
452         floor.setOrigin(0, 0);
453         floor.setImmovable(true);
454
455         this.physics.add.collider(this.arrayStack.getAll(), floor,
                function(gameObject1) {
456             if (gameObject1 instanceof Phaser.Physics.Arcade.Sprite) {
457                 if (gameObject1.getData('active')) {
458                     this.updateProgressbar(-1);
459                 }
460                 gameObject1.setVelocityY(0);
461                 gameObject1.setAngularVelocity(0);
462                 gameObject1.setPosition(Phaser.Math.RND.between(100 +
                        this.objectDisplaySize / 2, this.cameras.main.width
                        - this.objectDisplaySize / 2), 0 - 2 * this.
                        objectDisplaySize);
463                 this.arrayStatic.add(gameObject1);
464                 this.arrayFalling.remove(gameObject1);
```

```
465                 }
466             }, null, this);
467         }
468
469         /**
470          * Method for visually marking the drop zones
471          */
472         private initFirstDrop(): void {
473             let counterSet: string[] = [];
474
475             for (let sprite of this.arrayStack.getAll()) {
476                 if (sprite instanceof Phaser.Physics.Arcade.Sprite && sprite
                        .getData('active')) {
477                     const spriteName: string = sprite.name;
478                     if (!(counterSet.indexOf(spriteName) > -1)) {
479                         for (let dropZone of this.arrayDropZone.getChildren
                                ()) {
480                             if (dropZone instanceof Phaser.GameObjects.Zone)
                                    {
481                                 if (dropZone.name === spriteName) {
482                                     sprite.clearTint();
483                                     sprite.setPosition(dropZone.x + dropZone
                                            .width * 0.15, dropZone.y - dropZone
                                            .height * 0.2);
484
485                                     const imageScale: number = this.
                                            imageScalingFactor(Math.min(dropZone
                                            .width, dropZone.height) * this.
                                            droppedObjectScale, sprite.width,
                                            sprite.height);
486                                     sprite.setScale(imageScale);
487
488                                     this.arrayDropped.add(sprite);
489                                     sprite.disableInteractive();
490                                     this.arrayStatic.remove(sprite);
491
492                                     this.arrayStack.sendToBack(sprite);
493                                 }
494                             }
495                         }
496
497                         counterSet.push(spriteName);
498                     }
499
500                     // Return if every crate has an example
501                     if (counterSet.length >= this.propertyCount) {
502                         return;
503                     }
504                 }
505
506             }
507         }
508
509         /**
510          * Method for initializing the progressbar
511          */
```

```
512     private addProgressbar(): void {
513         const progressbarY: number = this.cameras.main.height - 10;
514         const progressbarCorrect: Phaser.GameObjects.Sprite = this.add.
                sprite(0, progressbarY, 'progressbar');
515         const multiplierX: number = 0.4;
516         const multiplierY: number = this.imageScalingFactor(this.cameras
                .main.height * 0.5, progressbarCorrect.height,
                progressbarCorrect.height);//0.3;
517
518         progressbarCorrect.setOrigin(0, 1);
519         progressbarCorrect.setScale(multiplierX, multiplierY);
520
521         const progressbarCorrectX: number = 10 * 2 + progressbarCorrect.
                width * multiplierX;
522         progressbarCorrect.setX(progressbarCorrectX);
523
524         const progressbarWrong: Phaser.GameObjects.Sprite = this.add.
                sprite(10, progressbarY, 'progressbar');
525         progressbarWrong.setOrigin(0, 1);
526         progressbarWrong.setScale(multiplierX, multiplierY);
527
528         const progressbarWrongX: number = 10;
529         progressbarWrong.setX(progressbarWrongX);
530
531         const plus: Phaser.GameObjects.Sprite = this.add.sprite(
                progressbarCorrectX, progressbarY - progressbarWrong.height
                * multiplierY - 10, 'plus');
532         const plusMultiplier: number = progressbarWrong.width *
                multiplierX / plus.width;
533         plus.setOrigin(0, 1);
534         plus.setScale(plusMultiplier);
535
536         const minus: Phaser.GameObjects.Sprite = this.add.sprite(
                progressbarWrongX, progressbarY - progressbarWrong.height *
                multiplierY - 10, 'minus');
537         const minusMultiplier: number = progressbarWrong.width *
                multiplierX / minus.width;
538         minus.setOrigin(0, 1);
539         minus.setScale(minusMultiplier);
540
541         this.correctBar = this.add.sprite(progressbarCorrectX +
                progressbarCorrect.width * multiplierX / 2 + 2, progressbarY
                 - 6, 'progressbarGreen');
542         this.correctBar.setOrigin(0.5, 1);
543         this.correctBar.setData('gameX', multiplierX);
544         this.correctBar.setData('gameY', 0.01);
545         this.correctBar.setData('gameMax', (progressbarCorrect.height *
                multiplierY - 6) / this.correctBar.height);
546         this.correctBar.setScale(multiplierX, 0.01);
547         this.correctBar.setAlpha(0.7);
548
549         this.wrongBar = this.add.sprite(progressbarWrongX +
                progressbarWrong.width * multiplierX / 2 + 2, progressbarY -
                 6, 'progressbarRed');
550         this.wrongBar.setOrigin(0.5, 1);
551         this.wrongBar.setData('gameX', multiplierX);
```

```
552             this.wrongBar.setData('gameY', 0.01);
553             this.wrongBar.setData('gameMax', (progressbarWrong.height *
                    multiplierY - 6) / this.wrongBar.height);
554             this.wrongBar.setScale(multiplierX, 0.01);
555             this.wrongBar.setAlpha(0.7);
556         }
557
558         /**
559          * Method for updating the progressbar
560          * @param point Number of points made (+1 or -1)
561          */
562         private updateProgressbar(point: number): void {
563
564             if (point > 0) {
565                 // Add to plus: max number of cards minus three
566                 this.correctCount += this.correctBar.getData('gameMax') /
                        this.maxPoints;
567                 this.correctBar.setScale(this.correctBar.getData('gameX'),
                        this.correctCount);
568
569
570             } else {
571                 // Add to minus: max not defined (lets say number of cards
                        minus three also...)
572                 this.wrongCount += this.wrongBar.getData('gameMax') / this.
                        maxPoints;
573                 this.wrongBar.setScale(this.wrongBar.getData('gameX'), this.
                        wrongCount);
574
575             }
576
577             if ((this.wrongCount >= this.wrongBar.getData('gameMax') -
                    Phaser.Math.EPSILON) || (this.correctCount >= this.
                    correctBar.getData('gameMax') - Phaser.Math.EPSILON)) {
578                 this.transitionOut('ScoreScene', {
579                     'score': this.correctCount / this.correctBar.getData('
                            gameMax') - this.wrongCount / this.wrongBar.getData(
                            'gameMax'),
580                     'previousScene': this.getKey() + String(this.level)
581                 });
582             }
583         }
584
585         /**
586          * Method for initializing sound effects
587          */
588         private initAudio(): void {
589             if (this.infinite) {
590                 this.sound.add('battle').play('', {loop: true});
591             } else {
592                 this.sound.add('space').play('', {loop: true});
593             }
594         }
595
596         /**
597          * Method for preselecting objects
```

```
598         */
599         private preselectObjects(): void {
600             // Preselect properties
601             for (let property of this.jsonObject['categories'][this.setCat -
                    1].validElements) {
602                 this.selectedElements.push(property);
603             }
604 
605             // Pick the elements
606             while (this.selectedElements.length > this.propertyCount) {
607                 this.selectedElements = Phaser.Math.RND.shuffle(this.
                        selectedElements);
608                 this.selectedElements.pop();
609             }
610 
611             // Get property images
612             let propLength: number = this.selectedElements[0].urls.length;
613             let rndIndex: number = Phaser.Math.RND.between(0, propLength -
                    1);
614             for (let prop of this.selectedElements) {
615                 const name = prop.urls[rndIndex];
616                 this.selectedElementsName.push(name);
617             }
618         }
619 }
```

A.9. restrictedSortingScene.ts

Listing A.8: *restrictedSortingScene.ts*

```
1  import 'phaser';
2  import {BaseScene} from './baseScene';
3 
4  export class RestrictedSortingScene extends BaseScene {
5 
6      /**
7       * Object database with all image names, image paths and image
            properties
8       */
9      private jsonObject: any;
10 
11     /**
12      * Global average display size of interactive objects
13      */
14     private objectSize: number;
15 
16     /**
17      * Two arrays which map the dropped objects to the respective zone
            and vice versa
18      */
19     private zoneObjMap: Phaser.GameObjects.Zone[];
20     private objZoneMap: Phaser.GameObjects.Sprite[];
21 
22     /**
```

```
23          * Array of preselected objects so that not all objects must be
                loaded
24          */
25         private preselectedObjects: any[];
26
27         /**
28          * Array of all displayed and not dropped objects
29          */
30         private displayedObjects: Phaser.GameObjects.Group;
31
32         constructor() {
33             super('RestrictedSortingScene');
34         }
35
36         init(data): void {
37
38             // Initialize data from previous scene
39             this.jsonObject = this.cache.json.get('objects');
40             this.level = data.level;
41
42             // Initialize fields
43             this.objZoneMap = [];
44             this.zoneObjMap = [];
45             this.preselectedObjects = [];
46             this.displayedObjects = this.add.group();
47             this.objectSize = 100;
48         }
49
50         preload(): void {
51         }
52
53         create(): void {
54             // Bring MenuUI to the front and initialize transition
55             this.game.scene.sendToBack(this.getKey());
56             this.transitionIn();
57
58             this.preselectObjects();
59             this.setBackground();
60             this.setDropZones();
61             this.setObjects();
62             this.initInput();
63             this.initAudio();
64         }
65
66         update(time: number): void {
67
68         }
69
70         /**
71          * Method which initializes the background graphics
72          */
73         private setBackground(): void {
74             const background: Phaser.GameObjects.Sprite = this.add.sprite(0,
                    0, 'background4');
75             background.setOrigin(0, 0);
76             background.setDisplaySize(this.cameras.main.width, this.cameras.
```

```
            main.height);
77          background.setTint(0xffccaa);
78          background.setAlpha(0.9);
79      }
80
81      /**
82       * Method which initializes the dropZones and their graphics
83       */
84      private setDropZones(): void {
85          const crate1: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width * (1 / 6), this.cameras.main.height * (3
                / 4), 'crate');
86          const crate2: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width * (3 / 6), this.cameras.main.height * (3
                / 4), 'crate');
87          const crate3: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width * (5 / 6), this.cameras.main.height * (3
                / 4), 'crate');
88
89          crate1.setOrigin(0.5, 0.5);
90          crate2.setOrigin(0.5, 0.5);
91          crate3.setOrigin(0.5, 0.5);
92
93          const scale: number = this.imageScalingFactor(Math.min(this.
                cameras.main.width / 3, this.cameras.main.height / 2) - 40,
                crate1.width, crate1.height);
94          crate1.setScale(scale);
95          crate2.setScale(scale);
96          crate3.setScale(scale);
97
98          for (let i: number = 0; i < 6; i++) {
99              let heightPosition: number = 0;
100             if (i > 2) {
101                 heightPosition = 1;
102             }
103
104             const x: number = crate1.x - crate1.width * scale / 2;
105             const y: number = crate1.y - crate1.height * scale / 2;
106             const zone: Phaser.GameObjects.Zone = this.add.zone(x + (i %
                    3) * crate1.width * scale / 3, y + heightPosition *
                    crate1.height * scale / 2, crate1.width * scale / 3,
                    crate1.height * scale / 2);
107             zone.setRectangleDropZone(crate1.width * scale / 3, crate1.
                    height * scale / 2);
108             zone.setOrigin(0, 0);
109             zone.setName('dropZone1');
110             this.zoneObjMap.push(zone);
111
112             // Display border of drop zones
113             const graphics: Phaser.GameObjects.Graphics = this.add.
                    graphics();
114             graphics.lineStyle(10, 0x000000);
115             graphics.strokeRect(zone.x, zone.y, zone.input.hitArea.width
                    , zone.input.hitArea.height);
116
117         }
```

```
118
119         for (let i: number = 0; i < 4; i++) {
120             let heightPosition: number = 0;
121             if (i > 1) {
122                 heightPosition = 1;
123             }
124
125             const x: number = crate2.x - crate2.width * scale / 2;
126             const y: number = crate2.y - crate2.height * scale / 2;
127             const zone: Phaser.GameObjects.Zone = this.add.zone(x + (i %
                    2) * crate2.width * scale / 2, y + heightPosition *
                    crate2.height * scale / 2, crate2.width * scale / 2,
                    crate2.height * scale / 2);
128             zone.setRectangleDropZone(crate2.width * scale / 2, crate2.
                    height * scale / 2);
129             zone.setOrigin(0, 0);
130             zone.setName('dropZone2');
131
132             this.zoneObjMap.push(zone);
133
134             // Display border of drop zones
135             const graphics: Phaser.GameObjects.Graphics = this.add.
                    graphics();
136             graphics.lineStyle(10, 0x000000);
137             graphics.strokeRect(zone.x, zone.y, zone.input.hitArea.width
                    , zone.input.hitArea.height);
138         }
139
140         if (this.level == 1) {
141             for (let i: number = 0; i < 2; i++) {
142                 const x: number = crate3.x - crate3.width * scale / 2;
143                 const y: number = crate3.y - crate3.height * scale / 2;
144                 const zone: Phaser.GameObjects.Zone = this.add.zone(x +
                        (i % 2) * crate3.width * scale / 2, y, crate3.width
                        * scale / 2, crate3.height * scale);
145                 zone.setRectangleDropZone(crate3.width * scale / 2,
                        crate3.height * scale);
146                 zone.setOrigin(0, 0);
147                 zone.setName('dropZone3');
148
149                 this.zoneObjMap.push(zone);
150
151                 // Display border of drop zones
152                 const graphics: Phaser.GameObjects.Graphics = this.add.
                        graphics();
153                 graphics.lineStyle(10, 0x000000);
154                 graphics.strokeRect(zone.x, zone.y, zone.input.hitArea.
                        width, zone.input.hitArea.height);
155             }
156         } else {
157             for (let i: number = 0; i < 5; i++) {
158                 let mod: number = 3;
159                 let heightPosition: number = 0;
160                 if (i > 2) {
161                     heightPosition = 1;
162                     mod = 2;
```

```
163                 }
164                 const x: number = crate3.x - crate3.width * scale / 2;
165                 const y: number = crate3.y - crate3.height * scale / 2;
166                 const zone: Phaser.GameObjects.Zone = this.add.zone(x +
                        (i % mod) * crate3.width * scale / mod, y +
                        heightPosition * crate3.height * scale / 2, crate3.
                        width * scale / mod, crate3.height * scale / 2);
167                 zone.setRectangleDropZone(crate3.width * scale / mod,
                        crate3.height * scale / 2);
168                 zone.setOrigin(0, 0);
169                 zone.setName('dropZone3');
170
171                 this.zoneObjMap.push(zone);
172
173                 // Display border of drop zones
174                 const graphics: Phaser.GameObjects.Graphics = this.add.
                        graphics();
175                 graphics.lineStyle(10, 0x000000);
176                 graphics.strokeRect(zone.x, zone.y, zone.input.hitArea.
                        width, zone.input.hitArea.height);
177             }
178         }
179
180     }
181
182     /**
183      * Method which initializes all input actions
184      */
185     private initInput(): void {
186         // On start dragging
187         this.input.on('dragstart', function (pointer, gameObject) {
188             if (gameObject instanceof Phaser.GameObjects.Sprite) {
189                 // Bring gameObject to top
190                 this.children.bringToTop(gameObject);
191
192                 // Set visual effects
193                 gameObject.clearTint();
194                 gameObject.setTint(0x999999);
195
196                 const scale: number = gameObject.getData('scale') * 1.2;
197                 gameObject.setScale(scale);
198             }
199         }, this);
200
201         // On stop dragging
202         this.input.on('dragend', function (pointer, gameObject, dropped)
                {
203             // If not dropped set default visual effects
204             if (!dropped && gameObject instanceof Phaser.GameObjects.
                    Sprite) {
205                 gameObject.clearTint();
206
207                 const scale: number = gameObject.getData('scale');
208                 gameObject.setScale(scale);
209
210                 let x: number = gameObject.x;
```

```
211                 let y: number = gameObject.y;
212                 const dist: number = Math.sqrt(Math.pow(gameObject.width
                        * gameObject.getData('scale'), 2) + Math.pow(
                    gameObject.height * gameObject.getData('scale'), 2))
                        / 2;
213
214                 if (x < 0) {
215                     x = dist;
216                 }
217
218                 if (y < 0) {
219                     y = dist;
220                 }
221
222                 if (x > this.cameras.main.width) {
223                     x = this.cameras.main.width - dist;
224                 }
225
226                 if (y > this.cameras.main.height) {
227                     y = this.cameras.main.height - dist;
228                 }
229
230                 gameObject.setPosition(x, y);
231
232                 // Check if the gameObject is already in a zone
233                 const index = this.objZoneMap.indexOf(gameObject);
234                 if (index > -1) {
235                     // Clear gameObject from this zone
236                     delete this.objZoneMap[index];
237                     this.displayedObjects.add(gameObject);
238                 }
239             }
240         }, this);
241
242         // While dragging update coordinates
243         this.input.on('drag', function (pointer, gameObject, dragX,
                dragY) {
244             if (gameObject instanceof Phaser.GameObjects.Sprite) {
245                 gameObject.setPosition(dragX, dragY);
246             }
247         }, this);
248
249         // On drop
250         this.input.on('drop', function (pointer, gameObject, dropZone) {
251             if (gameObject instanceof Phaser.GameObjects.Sprite &&
                    dropZone instanceof Phaser.GameObjects.Zone) {
252                 let scale: number = gameObject.getData('scale');
253                 let coords: number[] = [gameObject.input.dragStartX,
                        gameObject.input.dragStartY];
254
255                 // Check if there is already an object in the dropZone
                        and if the current gameObject fits with the other
                        elements
256                 let index1: number = this.zoneObjMap.indexOf(dropZone);
257                 let index2: number = this.objZoneMap.indexOf(gameObject)
                        ;
```

```
258                 if (typeof this.objZoneMap[index1] == 'undefined' &&
                        this.equalityCheck(gameObject, dropZone)) {
259                     // Check if the gameObject is already in a zone
260                     if (index2 > -1) {
261                         // Clear gameObject from this zone
262                         delete this.objZoneMap[index2];
263                         this.displayedObjects.add(gameObject);
264                     }
265
266                     // Set scale and coordinates
267                     scale = this.imageScalingFactor(Math.min(dropZone.
                            width, dropZone.height) * 0.9, gameObject.width,
                             gameObject.height);
268                     coords = [dropZone.getCenter().x, dropZone.getCenter
                            ().y];
269
270                     // Add object to the dropZone
271                     this.objZoneMap[index1] = gameObject;
272
273                     // Remove from displayed array
274                     this.displayedObjects.remove(gameObject);
275
276                     // If all elements are sorted, end game with score
277                     if (this.displayedObjects.getLength() <= 0) {
278                         this.transitionOut('ScoreScene', {
279                             'score': 1,
280                             'previousScene': this.getKey() + String(this
                                    .level)
281                         });
282                     }
283
284
285                 } else if (index2 > -1) {
286                     // Check if the gameObject was already in a zone
287                     let dropZoneOld: Phaser.GameObjects.Zone = this.
                            zoneObjMap[index2];
288                     scale = this.imageScalingFactor(Math.min(dropZoneOld
                            .width, dropZoneOld.height) * 0.9, gameObject.
                            width, gameObject.height);
289                 }
290
291                 // Set default visual effect and position
292                 gameObject.clearTint();
293                 gameObject.setScale(scale);
294                 gameObject.setPosition(coords[0], coords[1]);
295             }
296         }, this);
297     }
298
299     /**
300      * Method which initializes all the displayed object to sort
301      */
302     private setObjects(): void {
303         for (let image of this.preselectedObjects) {
304             const x: number = Phaser.Math.RND.between(100 + this.
                    objectSize / 2, this.cameras.main.width - this.
```

```
                objectSize / 2);
305             const y: number = Phaser.Math.RND.between(this.objectSize /
                    2, this.cameras.main.height / 2 - this.objectSize / 2);
306
307             const sprite: Phaser.GameObjects.Sprite = this.add.sprite(x,
                     y, image.name);
308
309             const size: number = Phaser.Math.RND.between(this.objectSize
                    , this.objectSize * 1.3);
310             const scale: number = this.imageScalingFactor(size, sprite.
                    width, sprite.height);
311
312             sprite.setScale(scale);
313             sprite.setOrigin(0.5, 0.5);
314             sprite.setVisible(true);
315
316             sprite.setName(image.name);
317             sprite.setData('scale', scale);
318             sprite.setData('properties', [image.cat1, image.cat2, image.
                    cat3, image.cat4]);
319
320             sprite.setInteractive({cursor: 'pointer'});
321             this.input.setDraggable(sprite);
322
323             this.displayedObjects.add(sprite);
324         }
325     }
326
327     /**
328      * Method for checking if there are mutual properties between the
             elements in the dropZone and the gameObject
329      * @param gameObject Current object you want to add to the dropZone
330      * @param dropZone Current dropZone the gameObject should be added
             to
331      */
332     private equalityCheck(gameObject: Phaser.GameObjects.Sprite,
            dropZone: Phaser.GameObjects.Zone): boolean {
333         // Initialize property-intersect-array
334         let mergeArray: any[] = [];
335
336         // Fill array with all property names
337         for (let cat of this.jsonObject['categories']) {
338             mergeArray = [...mergeArray, ...cat['validElements']];
339         }
340         mergeArray.forEach((element, index, array) => array[index] =
                element.name);
341
342         // Intersect valid elements of all elements already in dropzone
                plus current gameObject
343         [...this.objZoneMap.filter((element, index) => this.zoneObjMap[
                index].name === dropZone.name), gameObject].forEach(function
                 (element) {
344             mergeArray = mergeArray.filter((x) => element.getData('
                    properties').includes(x));
345         });
346
```

```
347         // Return false if there are no mutual properties
348         return (mergeArray.length > 0);
349     }
350
351     /**
352      * Method for initializing sound effects
353      */
354     private initAudio(): void {
355         this.sound.add('exploration').play('', {loop: true});
356     }
357
358     /**
359      * Method for preselecting the used objects
360      */
361     private preselectObjects(): void {
362         // Load accordingly to level
363         if (this.level === 1) {
364
365             // Copy category array
366             const categories: any[] = [...this.jsonObject['categories'
                    ]];
367
368             // Choose a random category
369             const rndCat: any = Phaser.Math.RND.shuffle(categories)[0];
370
371             // Select random fitting images
372             const images: any[] = [...this.jsonObject['images']];
373             Phaser.Math.RND.shuffle(images);
374
375             for (let property of Phaser.Math.RND.shuffle(rndCat['
                    validElements']).slice(0, 3)) {
376
377                 // Choose for one category 2, for the other 4 and for
                        the last 6 matching (in one property) images.
378                 let maxSize: number = 2;
379                 if (this.preselectedObjects.length === 2) {
380                     maxSize = 6;
381                 } else if (this.preselectedObjects.length === 6) {
382                     maxSize = 12;
383                 }
384
385                 // Iterate through the images until selecting criteria
                        is fulfilled
386                 for (let image of images) {
387                     // If selected enough images, break.
388                     if (this.preselectedObjects.length >= maxSize) {
389                         break;
390                     }
391
392                     // Load and add image if is has the same property as
                            the selected one
393                     if (image[rndCat.name] === property.name) {
394                         this.preselectedObjects.push(image);
395                     }
396                 }
397
```

```
398                 // Remove the selected images from the images array to
                        avoid duplicates
399                 this.preselectedObjects.forEach(function (element) {
400                     if (images.indexOf(element, 0) > -1) {
401                         images.splice(images.indexOf(element, 0), 1);
402                     }
403                 }, this);
404             }
405
406         } else {
407
408             // Copy category array
409             const categories: any[] = [...this.jsonObject['categories'
                    ]];
410
411             // Choose a random category
412             const rndCat: any = Phaser.Math.RND.shuffle(categories)[0];
413
414             // Select random fitting images
415             const images: any[] = [...this.jsonObject['images']];
416             Phaser.Math.RND.shuffle(images);
417
418             for (let property of Phaser.Math.RND.shuffle(rndCat['
                    validElements']).slice(0, 3)) {
419
420                 // Choose for one category 2, for the other 4 and for
                        the last 6 matching (in one property) images.
421                 let maxSize: number = 5;
422                 if (this.preselectedObjects.length === 5) {
423                     maxSize = 9;
424                 } else if (this.preselectedObjects.length === 9) {
425                     maxSize = 15;
426                 }
427
428                 // Iterate through the images until selecting criteria
                        is fulfilled
429                 for (let image of images) {
430                     // If selected enough images, break.
431                     if (this.preselectedObjects.length >= maxSize) {
432                         break;
433                     }
434
435                     // Load and add image if is has the same property as
                            the selected one
436                     if (image[rndCat.name] === property.name) {
437                         this.preselectedObjects.push(image);
438                     }
439                 }
440
441                 // Remove the selected images from the images array to
                        avoid duplicates
442                 this.preselectedObjects.forEach(function (element) {
443                     if (images.indexOf(element, 0) > -1) {
444                         images.splice(images.indexOf(element, 0), 1);
445                     }
446                 }, this);
```

```
447                 }
448             }
449         }
450 }
```

A.10. gameScene.ts

Listing A.9: *gameScene.ts*

```
1  import 'phaser';
2  import {BaseScene} from './baseScene';
3
4  export class GameScene extends BaseScene {
5
6      /**
7       * Object data file
8       */
9      private jsonObject: any;
10
11     /**
12      * Lock for not messing up animations by clicking repeatedly without
            waiting for the animation to finish
13      */
14     private lock: boolean;
15
16     /**
17      * State of the helper menu and data
18      */
19     private helpDown: boolean;
20     private buttonSize: number;
21
22     /**
23      * Lock for checking the marked objects
24      */
25     private checked: boolean;
26
27     /**
28      * Array of objects in play
29      */
30     private gameSet: any[];
31
32     /**
33      * Copy of the categories
34      */
35     private categorySet: any[];
36
37
38     /**
39      * Grid properties
40      */
41     private cellsX: number;
42     private cellsY: number;
43     private cellWidth: number;
44     private cellHeight: number;
```

```
45
46      /**
47       * Center coordinates of each grid cell
48       */
49      private arrayCoordinates: number[][];
50
51      /**
52       * All category objects
53       */
54      private arrayCategory: Phaser.GameObjects.Group;
55
56      /**
57       * All remaining (not yet discarded) objects
58       */
59      private arrayStack: Phaser.GameObjects.Group;
60
61      /**
62       * All displayed objects
63       */
64      private arrayDisplayed: Phaser.GameObjects.Group;
65
66      /**
67       * All marked objects
68       */
69      private arrayMarked: Phaser.GameObjects.Group;
70
71      /**
72       * All correctly indentified object sets
73       */
74      private arrayDropped: Phaser.GameObjects.Group;
75
76      /**
77       * Stats of already found sets
78       */
79      private points: number;
80      private maxPoints: number;
81
82      /**
83       * Timeprogressbar
84       */
85      private timefluid: Phaser.GameObjects.Sprite;
86
87      /**
88       * Gameprogressbar and data
89       */
90      private gamefluid: Phaser.GameObjects.Sprite;
91      private timedataStepsize: number;
92
93      constructor() {
94          super('GameScene');
95      }
96
97      init(data): void {
98
99          // Initialize data from previous scene
100         this.jsonObject = this.cache.json.get('objects');
```

```
101         this.level = data.level;
102
103         // Initialize fields
104         this.lock = false;
105         this.helpDown = false;
106         this.gameSet = [];
107         this.categorySet = [];
108         this.arrayCategory = this.add.group();
109         this.arrayStack = this.add.group();
110         this.arrayDisplayed = this.add.group();
111         this.arrayDropped = this.add.group();
112         this.arrayMarked = this.add.group();
113         this.checked = false;
114         this.points = 0;
115
116         // Initialize game parameters
117         this.buttonSize = 64;
118
119         this.maxPoints = 10;
120
121         this.timedataStepsize = 0.0001;
122
123         if (this.level > 1) {
124             this.timedataStepsize = 0.00001;
125         }
126
127         this.cellsX = 4;
128         this.cellsY = 3;
129
130         this.arrayCoordinates = [];
131         let offsetX = 100;
132         let offsetY = 30;
133         this.cellWidth = (this.cameras.main.width - 2 * offsetX) / (this
                .cellsX);
134         this.cellHeight = (this.cameras.main.height - 2 * offsetY) / (
                this.cellsY);
135         for (let x = 0; x < this.cellsX; x++) {
136             for (let y = 0; y < this.cellsY; y++) {
137                 this.arrayCoordinates.push([offsetX + this.cellWidth *
                        (0.5 + x), offsetY + this.cellHeight * (0.5 + y)]);
138             }
139         }
140
141
142     }
143
144     preload(): void {
145     }
146
147     create(): void {
148         // Bring MenuUI to the front and initialize transition
149         this.game.scene.sendToBack(this.getKey());
150         this.transitionIn();
151
152         this.preselectObjects();
153         this.setBackground();
```

```
154             this.setHelperMenu();
155             this.loadObjects();
156             this.initObjects();
157             this.setEqualityCheck();
158             this.setTimeProgressbar();
159             this.setGameProgressbar();
160             this.initAudio();
161         }
162 
163         update(time: number): void {
164 
165             // Check for correctness of selected cards
166             if (!this.checked && this.arrayMarked.getLength() >= 3) {
167                 this.checked = true;
168                 const objects: Phaser.GameObjects.GameObject[] = this.
                        arrayMarked.getChildren();
169                 this.replaceObject(this.checkEquality(objects[0], objects
                        [1], objects[2], true));
170             }
171 
172             // Update timeprogressbar
173             let timedata: number = this.timefluid.getData('timeY');
174             if (timedata <= 0) {
175                 this.checked = true;
176 
177                 // Endgame
178                 this.transitionOut('ScoreScene', {
179                     'score': this.points / this.gamefluid.getData('gameMax')
                            ,
180                     'previousScene': this.getKey() + String(this.level)
181                 });
182             } else {
183                 timedata -= this.timedataStepsize;
184                 this.timefluid.setData('timeY', timedata);
185                 this.timefluid.setScale(this.timefluid.getData('timeX'),
                        timedata);
186             }
187         }
188 
189         /**
190          * Method for preselecting objects
191          */
192         private preselectObjects(): void {
193             // Preselect objects and preload images
194             const selectedProperties: any[] = [];
195             // Choose three random properties of each category
196             for (let cat of this.jsonObject['categories']) {
197                 const selectedProperty: any[] = Phaser.Math.RND.shuffle(cat[
                        'validElements']).slice(0, 3);
198                 selectedProperty.forEach((object, index, array) => array[
                        index] = object.name);
199                 selectedProperties.push(selectedProperty);
200             }
201 
202             // Choose all image
203             for (let image of this.jsonObject['images']) {
```

```
204             if (
205                 selectedProperties[0].indexOf(image['cat1']) > -1 &&
206                 selectedProperties[1].indexOf(image['cat2']) > -1 &&
207                 selectedProperties[2].indexOf(image['cat3']) > -1
208             ) {
209                 // If level one, fix the last category
210                 if (this.level === 1) {
211                     if (image['cat4'] === 'full') {
212                         this.gameSet.push(image);
213                     }
214                 } else {
215                     this.gameSet.push(image);
216                 }
217             }
218         }
219 
220         // Preload category images
221         this.categorySet = [...this.jsonObject['categories']]; // Full
                copy the array instead of referencing
222 
223         // If level one, ignore the last category
224         if (this.level === 1) {
225             this.categorySet.pop();
226         }
227     }
228 
229     /**
230      * Function for initializing the background
231      */
232     private setBackground(): void {
233         const background = this.add.sprite(0, 0, 'background5');
234         background.setOrigin(0, 0);
235         background.setDisplaySize(this.cameras.main.width, this.cameras.
                main.height);
236         background.setTint(0xffccaa);
237         background.setAlpha(0.9);
238     }
239 
240     /**
241      * Function for creating the helper menu
242      */
243     private setHelperMenu(): void {
244         // Menu background
245         const backgroundY: number = 64 + 10 + 30;
246         const menuBackground: Phaser.GameObjects.Sprite = this.add.
                sprite(this.cameras.main.width - 64 - 10 - 30, backgroundY,
                'menubackground');
247         menuBackground.setAngle(180);
248         menuBackground.setOrigin(1, 0);
249         menuBackground.setDisplaySize(500, this.cameras.main.height +
                120);
250         menuBackground.setTint(0xdddddd);
251         menuBackground.setData('y', backgroundY);
252 
253         // Category indicator
254         let y: number = 16 + 32;
```

```
255         let countCategories: number = 0;
256
257
258         for (let cat of this.categorySet) {
259             if (cat.url === null || cat.name === null) {
260                 continue;
261             }
262             countCategories++;
263         }
264
265         for (let cat of this.categorySet) {
266             if (cat.url === null || cat.name === null) {
267                 continue;
268             }
269
270             y += (this.cameras.main.height - (16 + 32)) / (
                    countCategories + 1);
271             let name: string = cat.name;
272             const sprite: Phaser.GameObjects.Sprite = this.add.sprite(
                    this.cameras.main.width + 64, y, name);
273             sprite.setName(name);
274             sprite.setOrigin(0.5, 0.5);
275
276             const scale: number = this.imageScalingFactor(this.
                    buttonSize, sprite.width, sprite.height);
277             sprite.setScale(scale, scale);
278
279             sprite.setVisible(true);
280
281             this.arrayCategory.add(sprite);
282         }
283
284         // MenuButton
285         const menuButton: Phaser.GameObjects.Sprite = this.add.sprite(
                this.cameras.main.width - (10 + 32), 10 + 32, 'help');
286
287         const scale: number = this.imageScalingFactor(this.buttonSize,
                menuButton.width, menuButton.height);
288         menuButton.setScale(scale, scale);
289         menuButton.setInteractive({cursor: 'pointer'});
290
291         menuButton.on('pointerup', () => this.menuAction(menuButton,
                menuBackground));
292     }
293
294     /**
295      * Function fot loading all objects as sprites and data
            initialization
296      */
297     private loadObjects(): void {
298         for (let image of this.gameSet) {
299             const name: string = image.name;
300             const cat1: string = image.cat1;
301             const cat2: string = image.cat2;
302             const cat3: string = image.cat3;
303             const cat4: string = image.cat4;
```

```
304             const sprite: Phaser.GameObjects.Sprite = this.arrayStack.
                    create(200, 200, name);
305
306             sprite.setName(name);
307
308             sprite.setData('cat1', cat1);
309             sprite.setData('cat2', cat2);
310             sprite.setData('cat3', cat3);
311             sprite.setData('cat4', cat4);
312
313             sprite.setOrigin(0.5, 0.5);
314             sprite.setAngle(Phaser.Math.RND.angle());
315
316             sprite.setVisible(false);
317
318             const diag: number = Math.sqrt(Math.pow(sprite.height, 2) +
                    Math.pow(sprite.width, 2));
319             const scale: number = this.imageScalingFactor(Math.min(this.
                    cellWidth, this.cellHeight), diag, diag);
320             sprite.setScale(scale, scale);
321             sprite.setInteractive({cursor: 'pointer'});
322
323             sprite.on('pointerdown', function () {
324
325                 // If not already selected and there aren't already
                        three selected
326                 if (!this.arrayMarked.contains(sprite) && this.
                        arrayMarked.getLength() < 3) {
327
328                     // Mark card
329                     sprite.setTint(0x999999);
330
331                     // Add card to marked array
332                     this.arrayMarked.add(sprite);
333
334                     // Set checked to false
335                     this.checked = false;
336
337                 } else if (this.arrayMarked.contains(sprite)) {
338
339                     // Unmark card
340                     sprite.clearTint();
341
342                     // Remove from marked array
343                     this.arrayMarked.remove(sprite);
344
345                     this.resetHelp();
346
347                     // Set checked to false
348                     this.checked = false;
349                 }
350             }, this);
351         }
352     }
353
354     /**
```

```
355         * Function for initializing the first set of displayed objects
356         */
357        private initObjects(): void {
358
359            for (let coords of this.arrayCoordinates) {
360                const sprite: Phaser.GameObjects.Sprite = Phaser.Math.RND.
                       pick(this.arrayStack.getChildren());
361
362                sprite.setX(coords[0]);
363                sprite.setY(coords[1]);
364
365                sprite.setVisible(true);
366
367                this.arrayStack.remove(sprite);
368                this.arrayDisplayed.add(sprite);
369            }
370        }
371
372        /**
373         * Function for checking for set equality.
374         * All properties of one category have to be equal or inherently
                different.
375         * @param sprite1 First set object
376         * @param sprite2 Second set object
377         * @param sprite3 Third set object
378         * @param inGame Boolean: Shall the indicator for ingame help be
                marked?
379         */
380        private checkEquality(sprite1: Phaser.GameObjects.GameObject,
               sprite2: Phaser.GameObjects.GameObject, sprite3: Phaser.
               GameObjects.GameObject, inGame: boolean): boolean {
381            if (sprite1 instanceof Phaser.GameObjects.Sprite &&
382                sprite2 instanceof Phaser.GameObjects.Sprite &&
383                sprite3 instanceof Phaser.GameObjects.Sprite
384            ) {
385                // Return value
386                let replaceObjects: boolean = true;
387
388                for (let categoryIndicator of this.arrayCategory.getChildren
                       ()) {
389
390                    // Make sure your objects are sprites
391                    if (categoryIndicator instanceof Phaser.GameObjects.
                           Sprite) {
392
393                        // Clear tint
394                        categoryIndicator.clearTint();
395
396                        if (
397                            sprite1.getData(categoryIndicator.name) ===
                                   sprite2.getData(categoryIndicator.name) &&
398                            sprite2.getData(categoryIndicator.name) ===
                                   sprite3.getData(categoryIndicator.name) &&
399                            sprite1.getData(categoryIndicator.name) ===
                                   sprite3.getData(categoryIndicator.name)
400                        ) {
```

```
401                         if (inGame) {
402                             categoryIndicator.setTintFill(0x00dd00);
403                         }
404                     } else if (
405                         !(sprite1.getData(categoryIndicator.name) ===
                               sprite2.getData(categoryIndicator.name)) &&
406                         !(sprite2.getData(categoryIndicator.name) ===
                               sprite3.getData(categoryIndicator.name)) &&
407                         !(sprite1.getData(categoryIndicator.name) ===
                               sprite3.getData(categoryIndicator.name))
408                     ) {
409                         if (inGame) {
410                             categoryIndicator.setTintFill(0x00dd00);
411                         }
412                     } else {
413                         if (replaceObjects) {
414                             replaceObjects = false;
415                         }
416                         if (inGame) {
417                             // Mark category as red
418                             categoryIndicator.setTintFill(0xdd0000);
419                         }
420                     }
421                 }
422             }
423             return replaceObjects;
424         }
425     }
426
427     /**
428      * Function for replacing marked objects
429      * @param replaceObject
430      */
431     private replaceObject(replaceObject: boolean): void {
432         if (replaceObject) {
433             for (let oldSprite of this.arrayMarked.getChildren()) {
434                 if (oldSprite instanceof Phaser.GameObjects.Sprite) {
435                     oldSprite.clearTint();
436                     oldSprite.setVisible(false);
437                     this.arrayDisplayed.remove(oldSprite);
438                     this.arrayDropped.add(oldSprite);
439
440                     if (this.arrayStack.getLength() <= 0) {
441                         this.arrayDropped.getChildren().forEach((element
                                ) => this.arrayStack.add(element));
442                         this.arrayDropped.clear(false, false);
443                     }
444
445                     const newSprite: Phaser.GameObjects.Sprite = Phaser.
                            Math.RND.pick(this.arrayStack.getChildren());
446                     newSprite.setPosition(oldSprite.x, oldSprite.y);
447                     newSprite.setVisible(true);
448                     this.arrayStack.remove(newSprite);
449                     this.arrayDisplayed.add(newSprite);
450                 }
451             }
```

```
452
453             this.arrayMarked.clear(false, false);
454
455             this.resetHelp();
456
457             this.setEqualityCheck();
458
459             // Set checked to false
460             this.checked = false;
461
462             this.updateProgressbar();
463         }
464     }
465
466     /**
467      * Function for reseting the tint on the helper menu
468      */
469     private resetHelp(): void {
470         for (let cat of this.arrayCategory.getChildren()) {
471             if (cat instanceof Phaser.GameObjects.Sprite) {
472                 cat.clearTint();
473             }
474         }
475     }
476
477     /**
478      * Function for updating the progressbar
479      */
480     private updateProgressbar(): void {
481         this.points += this.gamefluid.getData('gameMax') / this.
                maxPoints;
482
483         // Add time but not more than max
484         let timedata: number = this.timefluid.getData('timeY');
485         timedata += this.timedataStepsize * 5000;
486         if (timedata > this.timefluid.getData('timeYMax')) {
487             timedata = this.timefluid.getData('timeYMax');
488         }
489
490         this.timefluid.setScale(this.timefluid.getData('timeX'),
                timedata);
491         this.timefluid.setData('timeY', timedata);
492
493         if (this.points >= this.gamefluid.getData('gameMax') - Phaser.
                Math.EPSILON) {
494             this.checked = true;
495
496             // Disable further interaction with the objects
497             this.arrayDisplayed.getChildren().forEach((gameObject) =>
                    gameObject.disableInteractive());
498
499             this.gamefluid.setScale(this.gamefluid.getData('gameX'),
                    this.points);
500
501             // End game
502             this.transitionOut('ScoreScene', {'score': 1, 'previousScene
```

```
                ': this.getKey() + String(this.level)});
503
504         }
505
506         this.gamefluid.setScale(this.gamefluid.getData('gameX'), this.
                points);
507     }
508
509     /**
510      * Function for checking if there is a occurrence of a set in the
            displayed objects
511      */
512     private setEqualityCheck(): void {
513         const objectSet: Phaser.GameObjects.GameObject[] = this.
                arrayDisplayed.getChildren();
514         const objectSetLength: number = objectSet.length;
515
516         for (let x = 0; x <= objectSetLength; x++) {
517             for (let y = x + 1; y <= objectSetLength - (x + 1); y++) {
518                 for (let z = y + 1; z <= objectSetLength - (y + 1); z++)
                        {
519                     if (this.checkEquality(objectSet[x], objectSet[y],
                            objectSet[z], false)) {
520                         return;
521                     }
522                 }
523             }
524         }
525
526         // Replace/add cards
527         this.rebuildDisplayedObjects();
528     }
529
530     /**
531      * Function for refreshing all displayed objects with new ones
532      * Usually used if there is no occurrence of a set.
533      */
534     private rebuildDisplayedObjects(): void {
535
536         // Replace all cards
537         for (let card of this.arrayDisplayed.getChildren()) {
538             if (card instanceof Phaser.GameObjects.Sprite) {
539                 card.setVisible(false);
540                 this.arrayStack.add(card);
541             }
542         }
543
544         this.arrayDisplayed.clear(false, false);
545
546         this.initObjects();
547
548     }
549
550     /**
551      * Function for initializing the animation on the helpers menu
552      * @param menuButton The helpers menu button
```

```
553         * @param menuBackground The helpers menu background
554         */
555        private menuAction(menuButton, menuBackground): void {
556            // ButtonAnimation
557            const menuButtonTween1: Phaser.Tweens.Tween = this.tweens.add({
558                targets: menuButton,
559                scale: 0.37,
560                ease: 'linear',
561                yoyo: true,
562                duration: 200
563            });
564
565            // Retract
566            if (this.helpDown) {
567                const menuBackgroundTween: Phaser.Tweens.Tween = this.tweens
                        .add({
568                    targets: menuBackground,
569                    y: menuBackground.getData('y'),
570                    x: this.cameras.main.width - 64 - 10 - 30,
571                    ease: 'Cubic',
572                    duration: 500,
573                    delay: 100
574                });
575
576                for (let helperButton of this.arrayCategory.getChildren()) {
577                    let helperButtonTween: Phaser.Tweens.Tween = this.tweens
                            .add({
578                        targets: helperButton,
579                        x: this.cameras.main.width + 64,
580                        ease: 'Cubic',
581                        duration: 300
582                    });
583                }
584
585
586                this.helpDown = false;
587
588                // Extend
589            } else {
590                let menuBackgroundTween: Phaser.Tweens.Tween = this.tweens.
                        add({
591                    targets: menuBackground,
592                    y: this.cameras.main.height + 90,
593                    x: this.cameras.main.width - 64 - 10 - 50,
594                    ease: 'Cubic',
595                    duration: 500
596                });
597
598                for (let helperButton of this.arrayCategory.getChildren()) {
599                    let helperButtonTween: Phaser.Tweens.Tween = this.tweens
                            .add({
600                        targets: helperButton,
601                        x: this.cameras.main.width - (16 + 32) + 4,
602                        ease: 'Cubic',
603                        duration: 300
604                    });
```

```
605                 }
606
607                 this.helpDown = true;
608             }
609
610         }
611
612         /**
613          * Function for initializing the progressbar for ingame game score
614          */
615         private setGameProgressbar(): void {
616             const progressbarY: number = this.cameras.main.height - 10;
617             const progressbar: Phaser.GameObjects.Sprite = this.add.sprite
                    (0, progressbarY, 'progressbar');
618             const multiplierX: number = 0.4;
619             const multiplierY: number = this.imageScalingFactor(this.cameras
                    .main.height * 0.5, progressbar.height, progressbar.height);
                    //0.3;
620             progressbar.setOrigin(0, 1);
621             progressbar.setScale(multiplierX, multiplierY);
622
623             const progressbarX: number = 10 * 2 + progressbar.width *
                    multiplierX;
624             progressbar.setX(progressbarX);
625
626             const progressstar: Phaser.GameObjects.Sprite = this.add.sprite(
                    progressbarX, progressbarY - progressbar.height *
                    multiplierY - 10, 'progressstar');
627             const starmultiplier: number = progressbar.width * multiplierX /
                     progressstar.width;
628             progressstar.setOrigin(0, 1);
629             progressstar.setScale(starmultiplier, starmultiplier);
630
631             this.gamefluid = this.add.sprite(progressbarX + progressbar.
                    width * multiplierX / 2 + 2, progressbarY - 6, 'gamefluid');
632             this.gamefluid.setOrigin(0.5, 1);
633             this.gamefluid.setData('gameX', multiplierX);
634             this.gamefluid.setData('gameY', 0.01);
635             this.gamefluid.setData('gameMax', (progressbar.height *
                    multiplierY - 6) / this.gamefluid.height);
636             this.gamefluid.setScale(multiplierX, 0.01);
637             this.gamefluid.setAlpha(0.7);
638         }
639
640         /**
641          * Function for initializing the game timer
642          */
643         private setTimeProgressbar(): void {
644             const progressbarY: number = this.cameras.main.height - 10;
645             const progressbar: Phaser.GameObjects.Sprite = this.add.sprite
                    (10, progressbarY, 'progressbar');
646             const multiplierX: number = 0.4;
647             const multiplierY: number = this.imageScalingFactor(this.cameras
                    .main.height * 0.5, progressbar.height, progressbar.height);
                    //0.3;
648             progressbar.setOrigin(0, 1);
```

```
649             progressbar.setScale(multiplierX, multiplierY);
650
651             const hourglass: Phaser.GameObjects.Sprite = this.add.sprite(10,
                    progressbarY - progressbar.height * multiplierY - 10, '
                    hourglass');
652             const starmultiplier: number = progressbar.width * multiplierX /
                    hourglass.width;
653             hourglass.setOrigin(0, 1);
654             hourglass.setScale(starmultiplier);
655
656             this.timefluid = this.add.sprite(10 + progressbar.width *
                    multiplierX / 2 + 2, progressbarY - 6, 'timefluid');
657             this.timefluid.setOrigin(0.5, 1);
658             this.timefluid.setData('timeX', multiplierX);
659             this.timefluid.setData('timeY', (progressbar.height *
                    multiplierY - 6) / this.timefluid.height);
660             this.timefluid.setData('timeYMax', (progressbar.height *
                    multiplierY - 6) / this.timefluid.height);
661             this.timefluid.setScale(this.timefluid.getData('timeX'), this.
                    timefluid.getData('timeY'));
662             this.timefluid.setAlpha(0.7);
663         }
664
665         /**
666          * Function for initializing sound effects
667          */
668         private initAudio(): void {
669             this.sound.add('fun').play('', {loop: true});
670         }
671     }
```

A.11. scoreScene.ts

Listing A.10: *scoreScene.ts*

```
1   import 'phaser';
2   import {BaseScene} from './baseScene';
3
4   export class ScoreScene extends BaseScene {
5       /**
6        * Game score
7        */
8       private score: number;
9
10      /**
11       * Name and level of the previous scene
12       */
13      private previousScene: string;
14
15      /**
16       * Standard size of a button
17       */
18      private buttonSize: number;
19
```

```
20      constructor() {
21          super('ScoreScene');
22      }
23
24      init(data): void {
25          // Initialize data from previous scene
26          this.score = data.score;
27          this.previousScene = data.previousScene;
28          this.buttonSize = 64;
29      }
30
31      preload(): void {
32
33      }
34
35      create(): void {
36          // Bring MenuUI to the front and initialize transition
37          this.game.scene.sendToBack(this.getKey());
38          this.transitionIn();
39
40          this.setBackground();
41          this.initUI();
42          this.initInput();
43          this.initAudio();
44      }
45
46      /**
47       * Method for initializing the background
48       */
49      private setBackground(): void {
50          const background: Phaser.GameObjects.Sprite = this.add.sprite(0,
                0, 'background5');
51          background.setOrigin(0, 0);
52          background.setDisplaySize(this.cameras.main.width, this.cameras.
               main.height);
53          background.setInteractive({ cursor: 'pointer' });
54      }
55
56      /**
57       * Method for initializing replaybutton and reward graphics
58       */
59      private initUI(): void {
60          // Add replay button
61          const replayButton: Phaser.GameObjects.Sprite = this.add.sprite(
               this.cameras.main.width - 100 + 34, this.cameras.main.height
                - 100 + 34, 'replay');
62          replayButton.setOrigin(0.5, 0.5);
63          const buttonScale: number = this.imageScalingFactor(this.
               buttonSize*1.5, replayButton.width, replayButton.height);
64          replayButton.setScale(buttonScale);
65          replayButton.setInteractive({ cursor: 'pointer' });
66          replayButton.on('pointerdown', function() {
67              replayButton.on('pointerup', function() {
68                  this.transitionOut(this.previousScene.substring(0, this.
                       previousScene.length-1), {level: Number(this.
                       previousScene[this.previousScene.length-1])});
```

```
69              }, this);
70          }, this);
71
72          let star: string;
73
74          if (this.score < 0.2) {
75              star = 'star_0';
76          } else if (this.score < 0.6) {
77              star = 'star_1';
78          } else if (this.score + Phaser.Math.EPSILON < 1) {
79              star = 'star_2';
80          } else {
81              star = 'star_3';
82          }
83          this.saveScore(star);
84
85          let sprite: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width / 2, this.cameras.main.height / 2, star);
86
87          sprite.setOrigin(0.5, 0.5);
88          const starScale: number = this.imageScalingFactor(this.cameras.
                main.width*3/5, sprite.height, sprite.width);
89          sprite.setScale(starScale);
90          sprite.setData('scale', starScale);
91          sprite.setInteractive({ cursor: 'pointer' });
92
93          const starTween: Phaser.Tweens.Tween = this.tweens.add({
94              targets: sprite,
95              ease: 'Linear',
96              scale: 1.1*sprite.getData('scale'),
97              repeat: 1000,
98              yoyo: true,
99              duration: 1000
100         });
101
102         // Add finger
103         const finger: Phaser.GameObjects.Sprite = this.add.sprite(this.
                cameras.main.width / 2, 6/7*this.cameras.main.height, '
                finger');
104         const fingerScale: number = this.imageScalingFactor(1/6*this.
                cameras.main.height, finger.width, finger.height, true);
105         finger.setOrigin(0.5, 0.5);
106         finger.setScale(fingerScale);
107         finger.setInteractive({ cursor: 'pointer' });
108
109         const fingerTween: Phaser.Tweens.Tween = this.tweens.add({
110             targets: finger,
111             alpha: 0.1,
112             ease: 'Linear',
113             repeat: 1000,
114             yoyo: true,
115             duration: 1000
116         });
117     }
118
119     /**
```

```
120         * Method which initializes all global input actions
121         */
122        private initInput(): void {
123            this.input.on('pointerdown', function() {
124                this.input.on('pointerup', function() {
125                    this.transitionOut('LevelMenuScene');
126                }, this);
127            }, this);
128        }
129
130        /**
131         * Method for initializing sound effects
132         */
133        private initAudio(): void {
134            if (this.score + Phaser.Math.EPSILON < 1) {
135                this.sound.add('lose').play();
136            } else {
137                this.sound.add('win').play();
138            }
139        }
140
141        /**
142         * Method for saving the score global
143         */
144        private saveScore(score: string): void {
145            if (typeof(Storage) !== "undefined") {
146                window.localStorage.setItem('phaser_score_' + this.
                       previousScene, score);
147            } else {
148                console.log("Sorry! No Web Storage support...");
149            }
150        }
151    }
```

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.