

Gotscha! - One Way To Support Learning



Franz Knobel

Bachelor Thesis
January 26, 2020

Supervising Professor:
Prof. Dr. Juraj Hromkovič

Supervisor:
Elizabeta Cavar

ETH zürich

Abstract

The importance of computer science is gradually increasing. Games which 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 her support during 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 were very patient with me.

ib

Contents

1. Introduction	1
1.1. Introduction	1
1.2. Goals of the Thesis	1
1.3. Problem Statement	2
1.4. Background	2
1.5. Motivation	2
1.6. Outline	3
2. Related Work	5
2.1. Previous work	5
2.1.1. Angular	6
2.2. Gamification and Game Based Learning (GBL)	6
2.2.1. Gamification	6
2.2.2. Game-Based-Learning	6
2.2.3. Gamification vs. Game-Based-Learning	7
2.3. Existing Educational Software	7
3. Requirements	9
3.1. Programming Environment	9
3.1.1. Boundary Conditions	9
3.1.2. Evaluation of Possible Solutions	10
3.2. Boundary Conditions of the Problem Statement	11
3.3. Boundary Condition of the Target Audience	12
3.4. Additional Conditions: Elements of Gamification	12

4. Design of the Learning Environment	13
4.1. Phaser 3	13
4.1.1. Base Configuration	13
4.1.2. Scenes	15
4.1.3. Managers	16
4.2. Objects	16
4.2.1. Object Generation	16
4.2.2. Object Storage	19
4.2.3. Object Display	20
4.2.4. Object Interaction	20
4.2.5. Modularity	20
4.3. Score Storage	21
4.4. Creating animated Introductions	21
4.5. Scene Concept of the Learning Environment	22
4.5.1. Drop Down Menu	22
4.5.2. Welcome Screen	22
4.5.3. Level Menu	22
4.5.4. Object Summary	23
4.5.5. Sorting with one Category	23
4.5.6. Sorting with one Category under difficult Conditions	23
4.5.7. Sorting with restricted space	23
4.5.8. Object pairing	24
4.5.9. Score Screen	25
4.5.10. Introduction	25
4.6. Code structure of the learning environment	25
4.6.1. Running the Code	26
4.6.2. BaseScene	26
4.6.3. PreloadAssets	27
4.6.4. DropDownMenu	29
4.6.5. LevelMenuSceneScene	29
4.6.6. IntroScene	31
4.6.7. SortingScene	31
4.6.8. PropertySortingScene	33
4.6.9. RestrictedSortingScene	34
4.6.10. GameScene	34
4.6.11. ScoreScene	37
4.7. Final Look of the Learning Environment	37
5. Evaluation	51
5.1. Debugging	51
5.1.1. Devices	51
5.1.2. Differences and obstacles	52
5.1.3. Unexpected Bugs	52
5.2. Target Audience Test	52
5.3. Various Audience Test	53

6. Conclusion	55
6.1. Obstacles	55
6.2. Limitation	56
6.3. Future Improvements	56
A. Gotcha! – Full Code	57
A.1. baseScene.ts	57
A.2. preloadAsset.ts	60
A.3. dropDownMenu.ts	64
A.4. introScene.ts	71
A.5. welcomeScene.ts	77
A.6. levelMenuScene.ts	79
A.7. sortingScene.ts	88
A.8. propertySortingScene.ts	97
A.9. restrictedSortingScene.ts	110
A.10. gameScene.ts	120
A.11. scoreScene.ts	133
Bibliography	137

Introduction

1.1. Introduction

The importance of computer science is gradually increasing. 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

1. Introduction

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 digitized materials and exercises will come. In related work [2] the already published digitized 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 digitized 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 based on the concept of *algorithmic thinking*[23]. One way of introducing kindergarten and primary school pupils to algorithmic thinking and its concepts is to make them solve problems with and without computers. This can be achieved by using age- and knowledge-appropriate learning materials.

1.5. Motivation

Computer science provides already an easier way than other topics for teaching in the digitized 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.

1. Introduction

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 already 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

2. Related Work

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 supervising the framework, it is continuously developed and tested. This makes it a stable Framework with the possibility for a fast development.

2.2. Gamification and Game Based Learning (GBL)

2.2.1. Gamification

Gamification reflects a social phenomenon creating 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. It 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 gameplay and the ability of the player to retain and apply said subject matter to the real world.

Good game-based learning applications can draw the user into familiar virtual environments. Within an effective game-based learning environment, it is common to work towards a goal, choosing actions and experiencing the consequences of those actions along the way. Making mistakes in a risk-free setting, and through experimentation, it is actively learned and practiced the right way to do things. This keeps the user highly engaged in practicing behaviors and thought processes that he can easily transfer from the simulated environment to the real life[5].

2.2.3. Gamification vs. Game-Based-Learning

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

2.3. Existing Educational Software

The idea of propagating games based on game based learning is often touched but rarely integrated in schools. Sometimes they have more in common with normal computer games and the learning aspect is only discovered when specifically looked at. A 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 for children from infancy to the age of eight.

OutNumbered! is an educational computer game software aimed at children from the age 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. Not within the classroom 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. 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 one part of the game is too easy or too hard for the user you are risking losing his attention fast, depending on his 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 gameplay, a more interesting story line and better visual effects? Yes, it is nearly impossible to compete with the game industry. But if you cannot compete, change the play field. The play field normal games are not touching is the school, as they normally have no correlation to school topics.

2. *Related Work*

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.

3. Requirements

- 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 does 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.

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 objects 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 or 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 and motivation
- Live tracking on correct and incorrect choices made will add 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 a 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 learning how to detect and compare properties of objects under simple and more difficult circumstances.

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';
```

4. Design of the Learning Environment

```
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 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.

Life Cycle

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[14].

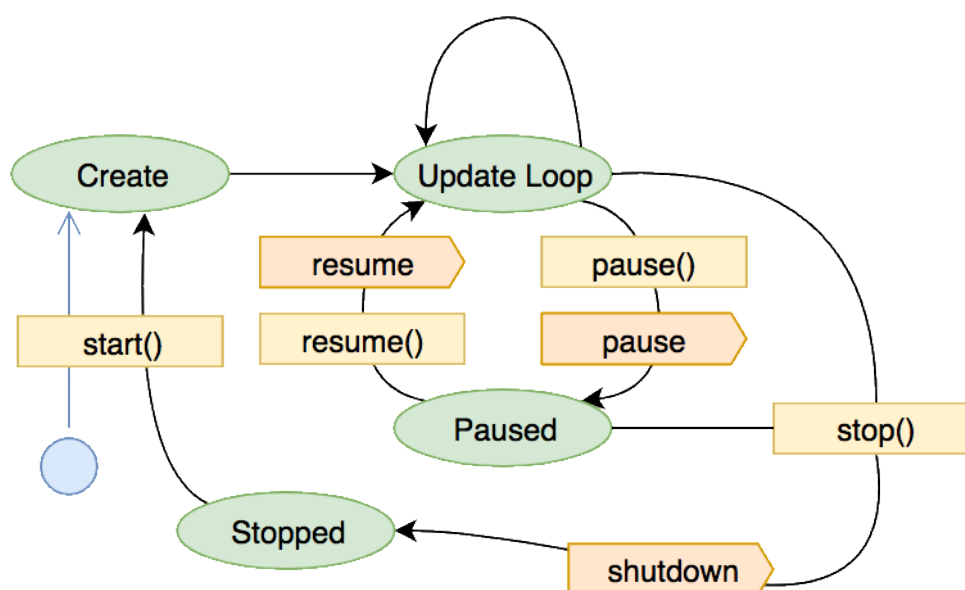


Figure 4.1.: Scene Life Cycle[14]

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.)[14].

Scenes here have 5 functions:

4. Design of the Learning Environment

- **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[14]. 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

```

1      ...
2      imageString = imgStr(colordefault, colordark, square, circle,
3                          triangle, ellipse, octagon, rhombus, filling, one,
4                          two, three, four, five, six)
5
6      filename = colordefault + shape + number + fillingname
7
8      with open("images/svg/" + filename + ".svg", "w+") as file:
9          file.write(imageString)
10         file.close()
11     ...

```

Listing 4.3: SVG Image File

```

1      <?xml version="1.0" standalone="yes"?>
2
3      <svg height="1000" width="1000" viewBox="0 0 1000 1000" xmlns="http
4          ://www.w3.org/2000/svg">
5          <defs>
6              <pattern id="stripe" patternUnits="userSpaceOnUse" width="20%"
7                  height="20%">
8                  <path stroke="aqua" stroke-linecap="butt" stroke-width="50" d="M -20
9                      -20 11000 1000"/>
10             ...
11             </pattern>
12             <pattern id="dotted" enable-background="true" patternUnits="
13                 userSpaceOnUse" width="15%" height="15%">
14                 <circle cx="30" cy="30" r="25" fill="aqua" />
15                 ...
16             </pattern>
17             <style>
18                 .button {
19
20                 stroke-width:5;
21                 stroke:black;
22
23                 }
24             </style>
25             </defs>
26
27             <g id="circle" display="none">
28                 <circle cx="500" cy="500" r="300" class="button" fill="blue"/>
29                 ...
30             </g>

```

4. Design of the Learning Environment

```
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   ]

```

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.

4. Design of the Learning Environment

It 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.

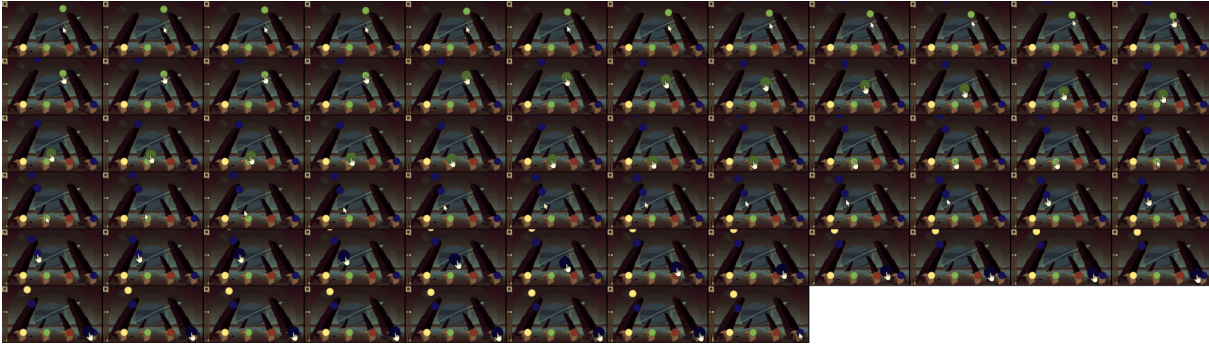


Figure 4.2.: Example: Spreadsheet

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

4. Design of the Learning Environment

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*

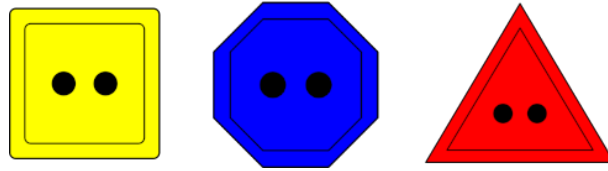


Figure 4.3.: Example: Matching Set Easy

Object pairing - hard version

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

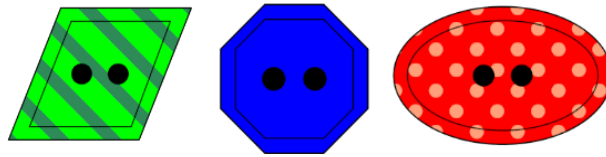


Figure 4.4.: Example: Matching Set Hard

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]. The game itself can be played here[7].

The following state diagram [4.5] displays our scenes and how they interact with each other.

4. Design of the Learning Environment



Figure 4.5.: Scene State Diagram

4.6.1. 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.

4.6.2. 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 a game object and animate the mask. As masks have to be applied to every object they want 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 desired 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     ...

```

4.6.3. 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 scenes as long as not explicitly removed/deleted. 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.

4. Design of the Learning Environment

Listing 4.12: *preloadAsset.ts*

```
1    ...
2    private preload(): void {
3        this.load.setPath('assets/geometrical_objects/');
4        this.load.json('objects', 'geometrical_objects.json');
5        ...
6    }
7    ...
8    private create(): void {
9        ...
10       this.preLoadImages();
11       ...
12       this.start();
13   }
14   ...
15   private preLoadImages(): void {
16       // Load category and object images
17       const jsonObject: any = this.cache.json.get('objects');
18
19       for (let category of jsonObject['categories']) {
20           this.load.setPath('assets/geometrical_objects/categories/');
21           this.load.image(category['name'], category['url']);
22
23           this.load.setPath('assets/geometrical_objects/images/');
24           for (let property of category['validElements']) {
25               for (let url of property['urls']) {
26                   this.load.image(url, url);
27               }
28           }
29       }
30
31       for (let image of jsonObject['images']) {
32           this.load.image(image['name'], image['name']);
33       }
34       ...
35   }
36   ...
37   private start(): void {
38       ...
39       this.load.start();
40   }
41   ...
```

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.

4.6.4. 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    ...

```

The drop down animation takes time. Thus, it was necessary to create a boolean field as a lock. With 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).

4.6.5. 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].

4. Design of the Learning Environment

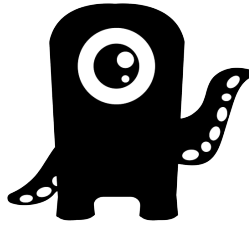


Figure 4.6.: Easy Level Icon

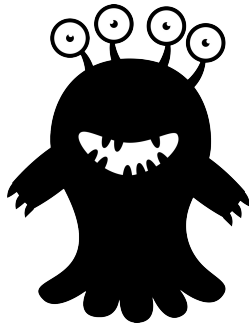


Figure 4.7.: Medium Level Icon

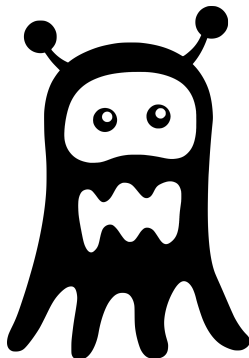


Figure 4.8.: Hard Level Icon

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)

```

1      ...
2      private initInput(): void {
3          this.input.on('pointerdown', function(pointer, currentlyOver) {
4              const gameObject: any = currentlyOver[0];
5              if (gameObject instanceof Phaser.GameObjects.Sprite) {
6                  ...
7                  gameObject.setData('clicked', true);
8              }
9          }, this);
10
11         this.input.on('pointerup', function(pointer, currentlyOver) {
12             const gameObject: any = currentlyOver[0];
13             if (gameObject instanceof Phaser.GameObjects.Sprite &&
14                 gameObject.getData('clicked')) {
15                 this.buttonFunction(gameObject);
16             }
17
18             this.levelButtons.getChildren().forEach(function(gameObject)
19                 {
20                     if (gameObject instanceof Phaser.GameObjects.Sprite) {
21                         ...
22                         gameObject.setData('clicked', false);
23                     }
24                 }, this);
25         }, this);
26     }
27     ...

```

4.6.6. IntroScene

To pause the current scene and still play the animated introduction, a separate scene is necessary. To play the respective intro, the intro scene has to be started anew and stopped again each time. The current scene 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.

4.6.7. SortingScene

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

Listing 4.17: returnQuad() (sortingScene.ts)

```

1      ...
2      private returnQuad(quadrant: number, quadrantType: number): number[]
3      {

```

4. Design of the Learning Environment

```
3      let ret: number[] = null;
4      ...
5      const leftOffsite: number = 100;
6      const rightOffsite: number = 0;
7      const topOffsite: number = 0;
8      const bottomOffsite: number = 100;
9
10     // Has entries dependant of
11     const horizontal: number[] = [];
12
13     // Has numberOfLines + 1 entries
14     const vertical: number[] = [];
15
16     horizontal.push(leftOffsite);
17
18     vertical.push(topOffsite);
19
20     switch (quadrantType) {
21         case 3: {
22             horizontal.push(leftOffsite + (this.cameras.main.width -
23                 leftOffsite - rightOffsite) / 3);
24             horizontal.push(leftOffsite + (this.cameras.main.width -
25                 leftOffsite - rightOffsite) * 2 / 3);
26             break;
27         }
28         case 4: {
29             horizontal.push(leftOffsite + (this.cameras.main.width -
30                 leftOffsite - rightOffsite) / 2);
31             vertical.push(topOffsite + (this.cameras.main.height -
32                 topOffsite - bottomOffsite) / 2);
33             break;
34         }
35         case 6: {
36             horizontal.push(leftOffsite + (this.cameras.main.width -
37                 leftOffsite - rightOffsite) / 3);
38             horizontal.push(leftOffsite + (this.cameras.main.width -
39                 leftOffsite - rightOffsite) * 2 / 3);
40             vertical.push(topOffsite + (this.cameras.main.height -
41                 topOffsite - bottomOffsite) / 2);
42             break;
43         }
44         default: {
45             break;
46         }
47     }
48
49     horizontal.push(this.cameras.main.width - rightOffsite);
50     vertical.push(this.cameras.main.height - bottomOffsite);
51
52     switch (quadrantType) {
53         case 3: {
54             ret = [Phaser.Math.RND.between(horizontal[quadrant] +
55                 spriteSizeHalf, horizontal[quadrant + 1] -
56                 spriteSizeHalf), Phaser.Math.RND.between(vertical[0]
57                     + spriteSizeHalf + this.cameras.main.height / 8,
58                     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         break;
57     }
58     case 6: {
59         if (quadrant < 3) {
60             ret = [Phaser.Math.RND.between(horizontal[quadrant]
                + spriteSizeHalf, horizontal[quadrant + 1] -
                spriteSizeHalf), Phaser.Math.RND.between(
                vertical[0] + spriteSizeHalf, vertical[1] -
                spriteSizeHalf)];
61         } else {
62             ret = [Phaser.Math.RND.between(horizontal[quadrant
                \% 3] + spriteSizeHalf, horizontal[(quadrant \%
                3) + 1] - spriteSizeHalf), Phaser.Math.RND.
                between(vertical[1] + spriteSizeHalf, vertical
                [2] - spriteSizeHalf)];
63         }
64         break;
65     }
66     default: {
67         break;
68     }
69 }
70 }
71 return ret;
72 }

```

4.6.8. PropertySortingScene

To specify the difficulty level, two fields are needed:

Listing 4.18: Level fields (propertySortingScene.ts)

```

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

```

In contrast to other scenes, objects must have the type `Phaser.Physics.Arcade.Sprite` instead of `Phaser.GameObjects.Sprite` as only arcade sprites can have an acceleration in specified directions.

To add another visual feature, objects get a random spin velocity and the time an object spawns gets shorter over time.

4.6.9. RestrictedSortingScene

To check if objects in the same box have some common property, 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 there is in the end no empty list, the objects have at least some property in common and thus this is a valid solution for one box. Which one does not matter to us now. with that, the possibility of multiple solutions, where some where not intended but also correct, is considered.

Listing 4.19: *equalityCheck (restrictedSortingScene.ts)*

```
1      ...
2      private equalityCheck(gameObject: Phaser.GameObjects.Sprite,
3                             dropZone: Phaser.GameObjects.Zone): boolean {
4          ...
5          let mergeArray: any[] = [];
6          ...
7          for (let cat of this.jsonObject['categories']) {
8              mergeArray = [...mergeArray, ...cat['validElements']];
9          }
10         mergeArray.forEach((element, index, array) => array[index] =
11             element.name);
12         ...
13         [...this.objZoneMap.filter((element, index) => this.zoneObjMap[
14             index].name === dropZone.name), gameObject].forEach(function
15             (element) {
16                 mergeArray = mergeArray.filter((x) => element.getData('
17                     properties').includes(x));
18             });
19         ...
20         return (mergeArray.length > 0);
21     }
22     ...
```

4.6.10. 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             } else {
13                 timedata -= this.timedataStepsize;
14                 this.timefluid.setData('timeY', timedata);
15                 this.timefluid.setScale(this.timefluid.getData('timeX'),
16                                         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.
3          EPSILON) {
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,
3                             sprite2: Phaser.GameObjects.GameObject, sprite3: Phaser.
4                             GameObjects.GameObject, inGame: boolean): boolean {
5          if (sprite1 instanceof Phaser.GameObjects.Sprite &&
6              sprite2 instanceof Phaser.GameObjects.Sprite &&
7              sprite3 instanceof Phaser.GameObjects.Sprite
8          ) {
9              // Return value
10             let replaceObjects: boolean = true;
11
12             for (let categoryIndicator of this.arrayCategory.getChildren
13                 ()) {
14                 // Make sure your objects are sprites
15                 if (categoryIndicator instanceof Phaser.GameObjects.
16                     Sprite) {

```

4. Design of the Learning Environment

```
14
15         // Clear tint
16         categoryIndicator.clearTint();
17
18         if (
19             sprite1.getData(categoryIndicator.name) ===
20             sprite2.getData(categoryIndicator.name) &&
21             sprite2.getData(categoryIndicator.name) ===
22             sprite3.getData(categoryIndicator.name) &&
23             sprite1.getData(categoryIndicator.name) ===
24             sprite3.getData(categoryIndicator.name)
25         ) {
26             if (inGame) {
27                 categoryIndicator.setTintFill(0x00dd00);
28             }
29             } else if (
30                 !(sprite1.getData(categoryIndicator.name) ===
31                 sprite2.getData(categoryIndicator.name)) &&
32                 !(sprite2.getData(categoryIndicator.name) ===
33                 sprite3.getData(categoryIndicator.name)) &&
34                 !(sprite1.getData(categoryIndicator.name) ===
35                 sprite3.getData(categoryIndicator.name))
36             ) {
37                 if (inGame) {
38                     categoryIndicator.setTintFill(0x00dd00);
39                 }
40             } else {
41                 if (replaceObjects) {
42                     replaceObjects = false;
43                 }
44                 if (inGame) {
45                     // Mark category as red
46                     categoryIndicator.setTintFill(0xdd0000);
47                 }
48             }
49         }
50     }
51     return replaceObjects;
52 }
```

To add another small gamification element in the form of a mini reward to the game, every time the user finds a matching pair, additional time 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 not finding 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     ...

```

4.6.11. 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.

4. Design of the Learning Environment

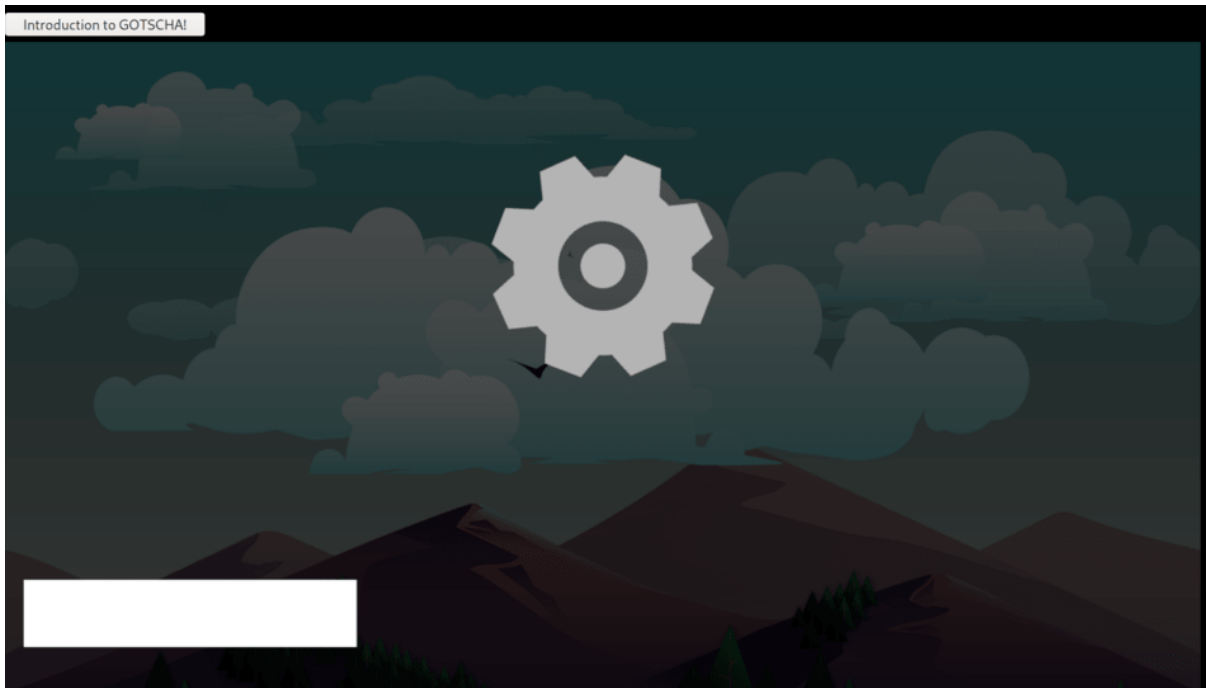


Figure 4.9.: Loading Screen



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

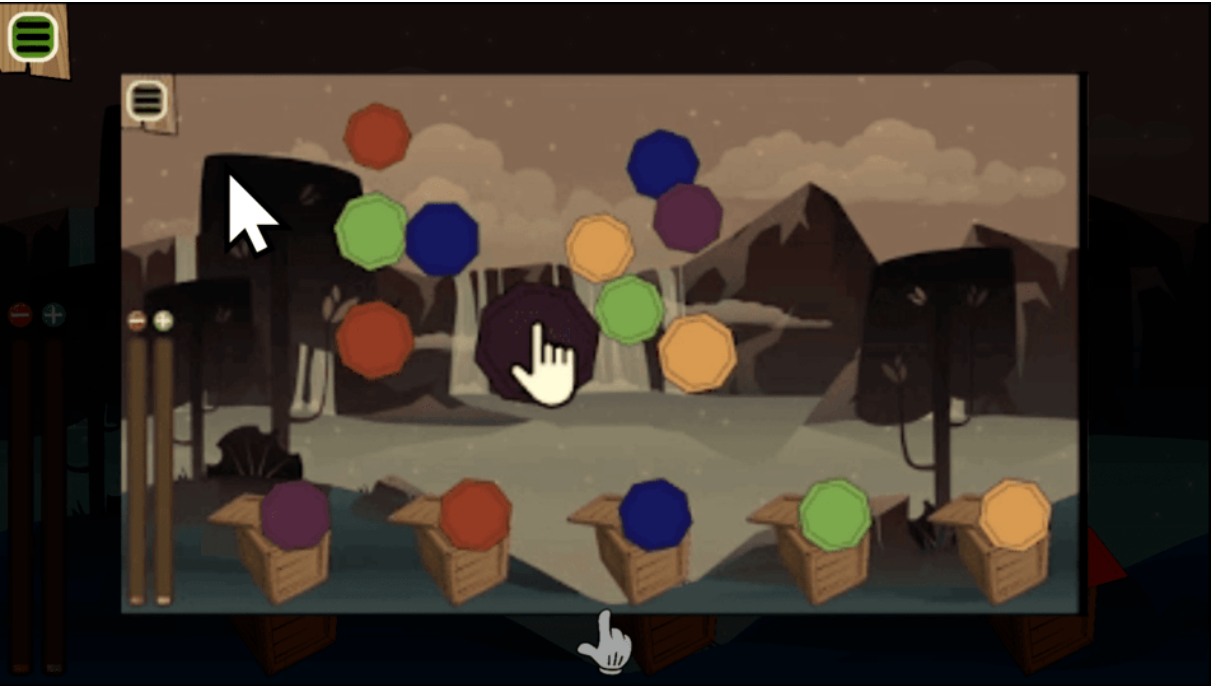


Figure 4.18.: Introduction Screen to Sorting Scene



Figure 4.19.: Property Sorting Category 1



Figure 4.20.: Property Sorting Category 2



Figure 4.21.: Property Sorting Category 3

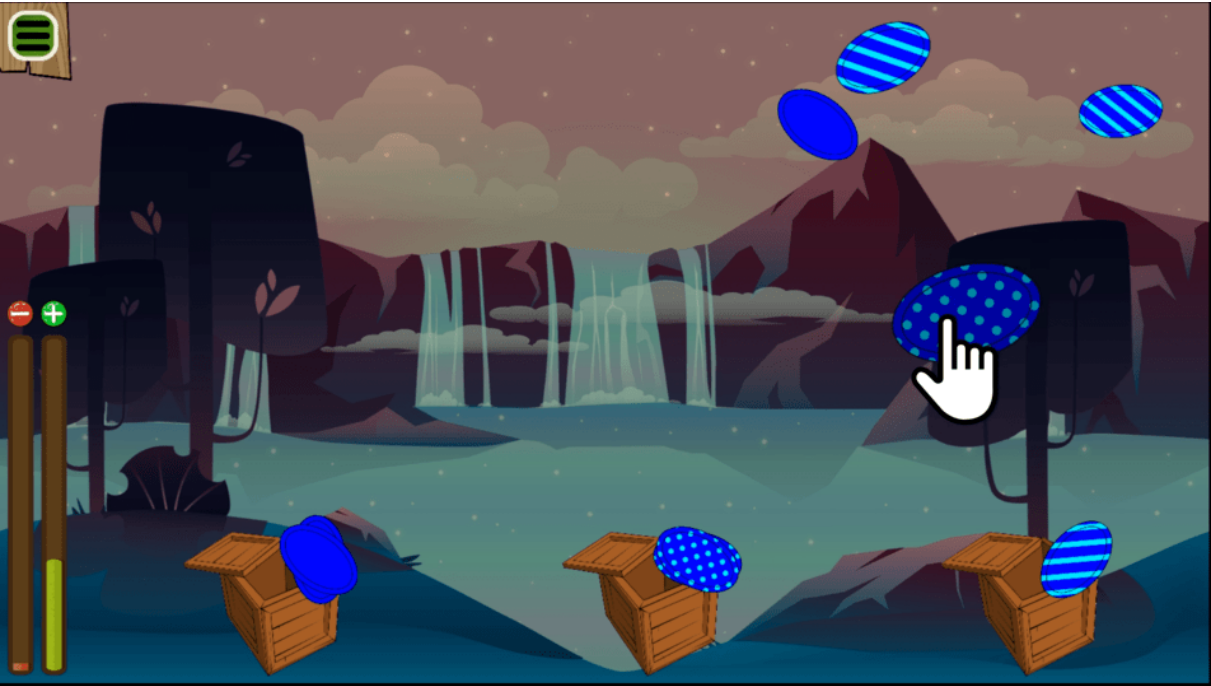


Figure 4.22.: Property Sorting Category 4



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

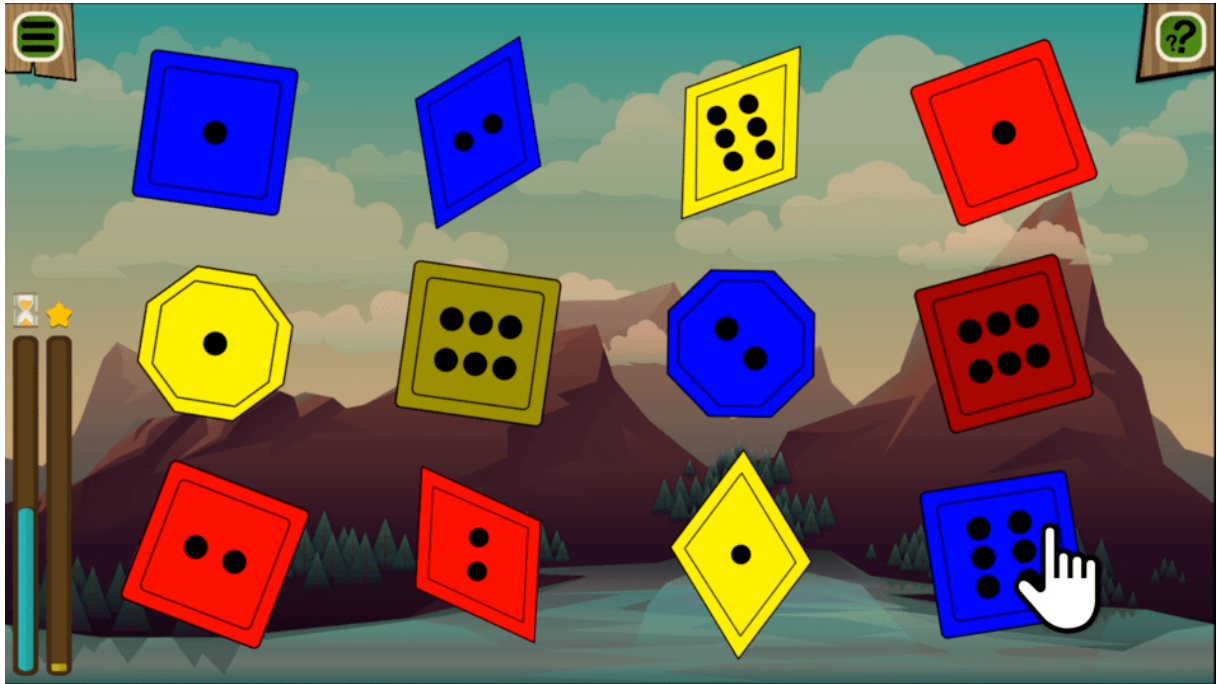


Figure 4.27.: Object Pairing Easy



Figure 4.28.: Object Pairing Easy Helper Bar

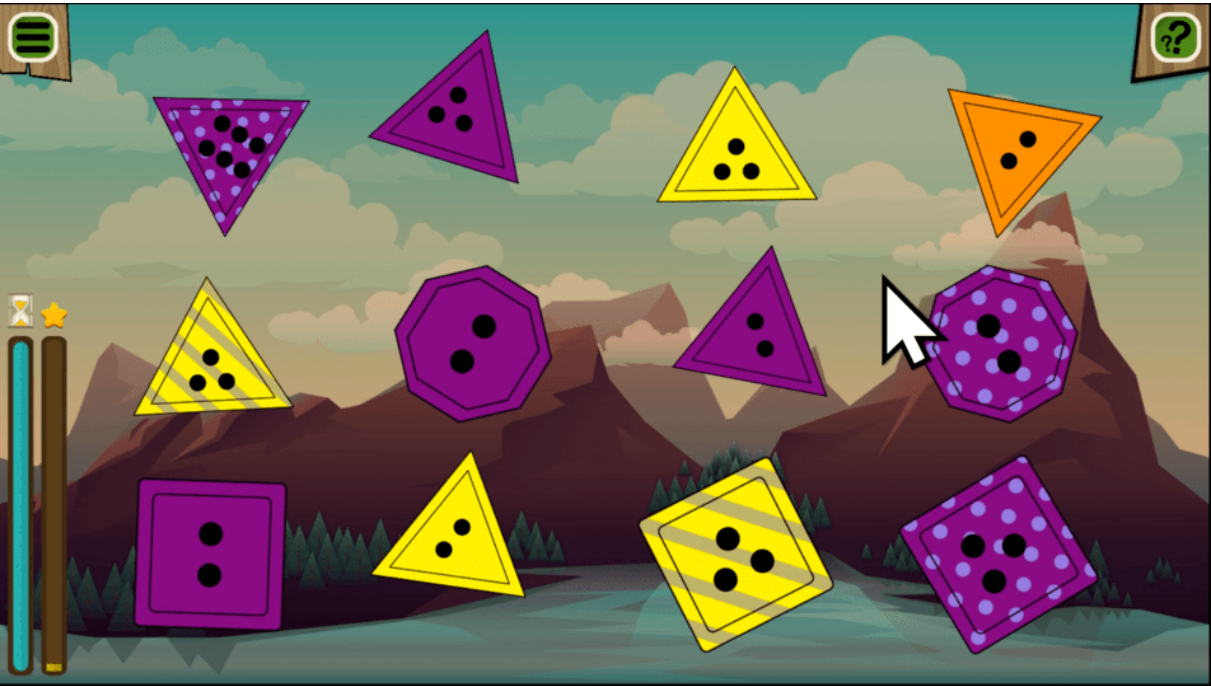


Figure 4.29.: Object Pairing Hard



Figure 4.30.: Object Pairing Hard Helper Bar



Figure 4.31.: Score Screen

4. *Design of the Learning Environment*

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

5. Evaluation

- 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 ignore the fullscreen button and just minimize the header bar (near fullscreen). 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 were found. The most fascinating are mentioned below.

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 user's 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 of a few minutes 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. They even formed groups of two to three kids by themselves and had no trouble in interacting with the environment together.

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, 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.

Most fascinating was that almost every test subject knew what to do in the level with restricted space and some had a different approach to get to the solution. 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 on even after having played and beaten all the levels. In self formed groups they were even discussing possible solutions and actively 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. In the end they did.

The second test group needed some adjustment time of a few minutes to figure out how to interact with the browser based application. On some test subjects the motoric abilities were not well trained. 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.

5. Evaluation

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.

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. Obstacles

For starters it is in my opinion a hurdle to understand the concept of the framework. The framework and the purpose of its parts is often clear but not the interplay of those parts. Even now I am still discovering new ways of optimizing my code or restructuring methods to make it more simple and understandable.

One of the biggest obstacles was to find a starting point. I like to give the foundation/structure of my code also called skeleton usually more thought than other people. This for the reason that I know, restructuring a skeleton can be very time consuming if not impossible in the time I am given. But a skeleton template for a specific framework in a specific environment is not guaranteed to exist.

6.2. 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.

Furthermore, additional TypeScript support will be added with the upcoming fourth version of phaser.

6.3. 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 }
```

A. Gotcha! – Full Code

```
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
43      * guaranteed
44      */
45     private generateNewSeed(): void {
46         const rndStr: string = Phaser.Math.RND.realInRange(Math.pow(10,
47             2), Math.pow(10,10)).toString();
48         Phaser.Math.RND.sow([rndStr]);
49     }
50
51     /**
52      * Method for initializing the shape, position and properties of the
53      * graphical scene transition
54      */
55     private transitionInit(): void {
56         // Shape of the graphical transition
57         const circle: Phaser.GameObjects.Graphics = this.add.graphics();
58
59         // Shape of the screen
60         const rectangle: Phaser.GameObjects.Rectangle = this.add.
61             rectangle(0, 0, this.cameras.main.width, this.cameras.main.
62             height, 0x000000);
63
64         // Define circle as the mask
65         const mask: Phaser.Display.Masks.GeometryMask = circle.
66             createGeometryMask();
67
68         circle.setPosition(this.cameras.main.width / 2, this.cameras.
69             main.height / 2);
70         circle.fillCircle(0, 0, 0.1);
71         circle.setDepth(0);
72
73         mask.setInvertAlpha(true);
74
75         rectangle.setDepth(1);
76         rectangle.setOrigin(0, 0);
77         rectangle.setMask(mask);
78
79         circle.fillCircle(0, 0, 0.1);
80
81         this.transition = [circle, rectangle];
82     }
```

```

76
77 /**
78  * Opening transition. Normally used to visually introduce a new
79   scene
80  */
81  protected transitionIn(): void {
82      // Generating a new seed, so that randomness is guaranteed in
83       every repetition of a scene
84      this.generateNewSeed();
85
86      this.transitionInit();
87
88      this.children.bringToTop(this.transition[1]);
89
90      const tween: Phaser.Tweens.Tween = this.add.tween({
91          targets: this.transition[0],
92          scale: 10 * 0.5 * Math.sqrt(Math.pow(this.cameras.main.width
93              , 2) + Math.pow(this.cameras.main.height, 2)),
94          ease: 'linear',
95          duration: 700
96      });
97
98      tween.on('start', () => this.sound.volume = 0);
99      tween.on('complete', () => this.introduction());
100      tween.on('update', () => this.sound.volume += 1/tween.duration);
101  }
102
103 /**
104  * Closing transition. Normally used to visually close or stop a
105   scene.
106  * @param scene The scene you want to start next.
107  * @param data Additional data you want to give to the next scene.
108  */
109  protected transitionOut(scene: string, data?: any): void {
110      this.children.bringToTop(this.transition[1]);
111
112      const tween: Phaser.Tweens.Tween = this.add.tween({
113          targets: this.transition[0],
114          scale: 0,
115          ease: 'linear',
116          duration: 700
117      });
118
119      tween.on('complete', () => this.sceneChange(scene, data));
120      tween.on('update', () => this.sound.volume -= 1/tween.duration);
121  }
122
123 /**
124  * Helper method for starting a new scene and stopping the current
125   one
126  * as the behaviour of the current scene when starting a new one is
127   not clearly defined in the framework at this point of time.
128  * @param scene The scene you want to start next
129  * @param data Additional data you want to give to the next scene.
130  */
131  protected sceneChange(scene: string, data?: any): void {

```

A. Gotcha! – Full Code

```
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
134      * scene
135      */
136     protected introduction(): void {
137         this.scene.pause();
138         this.game.scene.start("IntroScene", {'pausedScene': this.getKey
139             (), 'level': this.getLevel()});
140     }
141
142     /**
143      * Returns the correct scaling factor for the wanted image size in
144      * relation to the real image size.
145      * @param wantedImageSize Image size you want to have for a
146      * dimension
147      * @param realImageSizeWidth The image width you want to scale
148      * @param realImageSizeHeight The image height you want to scale
149      * @param scaleToHeight Boolean for scaling height or width of image
150      * to the wanted size. Default to false.
151      */
152     protected imageScalingFactor(wantedImageSize: number,
153         realImageSizeWidth: number, realImageSizeHeight: number,
154         scaleToHeight: boolean = false): number {
155         let ret: number;
156         if (scaleToHeight) {
157             ret = Math.max(wantedImageSize / realImageSizeWidth,
158                 wantedImageSize / realImageSizeHeight);
159         } else {
160             ret = Math.min(wantedImageSize / realImageSizeWidth,
161                 wantedImageSize / realImageSizeHeight);
162         }
163         return ret;
164     }
165 }
```

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:

```

A. Gotcha! – Full Code

```
        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', {
119         frameWidth: 336, frameHeight: 189, endFrame: 150});
120     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(0xffffffff, 1);

```

A. Gotcha! – Full Code

```
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
13      * waiting for the animation to finish
14      */
15     private lock: boolean;
16
17     /**
18      * State of the drop down menu
19      */
20     private menuDown: boolean;
```

```

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  }

```

A. Gotcha! – Full Code

```
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      }, this);
123
124      // MenuButton
125      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 {
171     switch (gameObject.name) {
172         case 'menuButton': {
173             if (!this.lock) {
174                 // Acquire lock
175                 this.lock = true;
176                 this.menuAction();
177             }
178             break;
179         }
180     }
181 }

```

A. Gotcha! – Full Code

```
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             ) {
196             // @ts-ignore
197             const sceneKey: string = scene.key;
198             if (!(sceneKey === this.getKey()) && (this.game.
199                 scene.isActive(sceneKey) || this.game.scene.
200                 isPaused(sceneKey))) {
201                 scene.sound.stopAll();
202                 this.game.scene.stop(sceneKey);
203             }
204             }, this);
205
206         if (this.key_paused_scene === 'LevelMenuScene' || this.
207             key_paused_scene === 'WelcomeScene') {
208             this.game.scene.start('WelcomeScene');
209         } else {
210             this.game.scene.start('LevelMenuScene');
211         }
212         break;
213     }
214
215     default: {
216         break;
217     }
218 }
219
220 /**
221  * Method which defines the graphical behaviour of the drop down
222  * menu
223  */
224 private menuAction(): void {
225     if (this.menuDown) {
226         // Animation
227         const menuButtonTween: Phaser.Tweens.Tween = this.tweens.add
228             ({
229                 targets: this.menuButton,
230                 angle: 0,
231                 ease: 'Cubic',
232                 duration: 700,
```



```

229         onComplete: () => this.lock = false
230     });
231
232     const menuBackgroundTween: Phaser.Tweens.Tween = this.tweens
233         .add({
234             targets: this.menuBackground,
235             y: 100,
236             ease: 'Cubic',
237             duration: 500,
238             delay: 200
239         });
240
241     const fullscreenButtonTween: Phaser.Tweens.Tween = this.
242         tweens.add({
243             targets: this.fullscreenButton,
244             x: -64,
245             ease: 'Cubic',
246             duration: 500,
247             delay: 100
248         });
249
250     const exitButtonTween: Phaser.Tweens.Tween = this.tweens.add
251         ({
252             targets: this.exitButton,
253             x: -64,
254             ease: 'Cubic',
255             duration: 500
256         });
257
258     const pixelScreenTween: Phaser.Tweens.Tween = this.tweens.
259         add({
260             targets: this.pauseBackground.getAll(),
261             alpha: 0,
262             ease: 'linear',
263             duration: 700,
264             delay: 0,
265             onComplete: () => this.pauseBackground.setVisible(false)
266         });
267
268     this.menuDown = false;
269
270     // Resume current scene
271     this.game.scene.resume(this.key_paused_scene);
272
273     } else {
274         // Pause current scene
275         this.sound.add('pause').play();
276
277         // @ts-ignore
278         const key_paused_scene: string = this.game.scene.getScenes(
279             true)[0].key;
280         this.game.scene.pause(key_paused_scene);
281         this.key_paused_scene = key_paused_scene;
282
283         // Animation
284         const menuButtonTween: Phaser.Tweens.Tween = this.tweens.add

```

A. Gotcha! – Full Code

```
280         ({
281             targets: this.menuButton,
282             angle: -90,
283             ease: 'Cubic',
284             duration: 700,
285             onComplete: () => this.lock = false,
286         });
287
288         const menuBackgroundTween: Phaser.Tweens.Tween = this.tweens
289             .add({
290                 targets: this.menuBackground,
291                 y: 3 * (64 + 10) + 30,
292                 ease: 'Cubic',
293                 duration: 600
294             });
295
296         const fullscreenButtonTween: Phaser.Tweens.Tween = this.
297             tweens.add({
298                 targets: this.fullscreenButton,
299                 x: 10 + 32,
300                 ease: 'Cubic',
301                 duration: 500,
302                 delay: 100,
303             });
304
305         const exitButtonTween: Phaser.Tweens.Tween = this.tweens.add
306             ({
307                 targets: this.exitButton,
308                 x: 10 + 32,
309                 ease: 'Cubic',
310                 duration: 500,
311                 delay: 200
312             });
313
314         this.pauseBackground.setVisible(true);
315
316         const pixelScreenTween: Phaser.Tweens.Tween = this.tweens.
317             add({
318                 targets: this.pauseBackground.getAll(),
319                 alpha: 0.9,
320                 ease: 'linear',
321                 duration: 700,
322                 delay: 0
323             });
324
325         this.menuDown = true;
326     }
327 }
328
329 /**
330  * Method for setting the pixelated overlay and the hourglass
331  */
332 private setPixelScreen(): void {
333     const pixelScreen: Phaser.GameObjects.Grid = this.add.grid(0, 0,
334         this.cameras.main.width, this.cameras.main.height, this.
335         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(
337         this.cameras.main.width / 2, this.cameras.main.height / 2, '
338         hourglass');
339     hourglass.setOrigin(0.5, 0.5);
340
341     const clockScale: number = this.imageScalingFactor(3 / 5 * this.
342         cameras.main.height, hourglass.width, hourglass.height);
343     hourglass.setScale(clockScale);
344     hourglass.setAlpha(0);
345
346     this.pauseBackground.add(hourglass);
347
348     this.pauseBackground.setVisible(false);
349     this.children.sendToBack(this.pauseBackground);
350 }
351 }

```

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

```

A. Gotcha! – Full Code

```
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.
50             setBackground();
51         const intro: Phaser.GameObjects.Sprite = this.setIntro();
52
53         const finger: Phaser.GameObjects.Sprite = this.add.sprite(this.
54             cameras.main.width/2, this.cameras.main.height-10, 'finger')
55             ;
56         const scale: number = this.imageScalingFactor(this.cameras.main.
57             height/10, finger.width, finger.height);
58         finger.setScale(scale);
59         finger.setOrigin(0.5, 1);
60
61         if (this.pausedScene === "GameScene"){
62             intro.setX(3/4*this.cameras.main.width);
63             const intro1scale: number = this.imageScalingFactor(this.
64                 cameras.main.width/2, intro.width, intro.height);
65             intro.setScale(intro1scale);
66
67             const intro2: Phaser.GameObjects.Sprite = this.add.sprite
68                 (1/4*this.cameras.main.width, 0, 'intro_set');
69             intro2.setOrigin(0.5, 0.5);
70             const intro2scale: number = this.imageScalingFactor(this.
71                 cameras.main.width/2, intro2.width, intro2.height);
72             intro2.setScale(intro2scale);
73
74             this.setAnimation(intro, background, finger, intro2);
75         } else {
76             this.setAnimation(intro, background, finger);
77         }
78     }
79
80     /**
81      * Method for the input setup
82      * @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 {

```

A. Gotcha! – Full Code

```
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
216      * scene
217      */
218     private getIntroData(): [string, Phaser.Types.Animations.
219         GenerateFrameNumbers] {
220         let ret: [string, Phaser.Types.Animations.GenerateFrameNumbers];
221
222         switch (this.pausedScene + String(this.getLevel())) {
223             case 'PropertySortingScene1': {
224                 ret = ['intro_sorting', {start: 0, end: 150, first:
225                     150}];
226                 break;
227             }
228             case 'PropertySortingScene2': {

```

A. Gotcha! – Full Code

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



```

277         ret = ['intro_set_hard', {start: 0, end: 68, first:
278             68}];
279         break;
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.
36             main.height);
37         background.setInteractive({ cursor: 'pointer' });
38     }
39 }

```

A. Gotcha! – Full Code

```
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
24         // number of buttons
25         this.buttonSize = Math.min(this.cameras.main.width / (4 + 2),
26                                     this.cameras.main.height / (3 + 2));
27     }
28
29     preload(): void {
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();

```

A. Gotcha! – Full Code

```
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,
54             0, 'background1');
55         background.setOrigin(0, 0);
56         background.setDisplaySize(this.cameras.main.width, this.cameras.
57             main.height);
58     }
59
60     /**
61      * Method for initializing the level buttons and their onclick
62      * action
63      */
64     private setLevelButtons(): void {
65         const catButton: Phaser.GameObjects.Sprite = this.add.sprite(20,
66             this.cameras.main.height - 20, 'catButton');
67         const eraseButton: Phaser.GameObjects.Sprite = this.add.sprite(
68             this.cameras.main.width - 20, this.cameras.main.height - 20,
69             'erase');
70         const levelButton11: Phaser.GameObjects.Sprite = this.add.sprite
71             (1 / 5 * this.cameras.main.width, 1 / 4 * this.cameras.main.
72             height, 'levelButton11');
73         const levelButton12: Phaser.GameObjects.Sprite = this.add.sprite
74             (2 / 5 * this.cameras.main.width, 1 / 4 * this.cameras.main.
75             height, 'levelButton12');
76         const levelButton13: Phaser.GameObjects.Sprite = this.add.sprite
77             (3 / 5 * this.cameras.main.width, 1 / 4 * this.cameras.main.
78             height, 'levelButton13');
79         const levelButton14: Phaser.GameObjects.Sprite = this.add.sprite
80             (4 / 5 * this.cameras.main.width, 1 / 4 * this.cameras.main.
81             height, 'levelButton14');
82         const levelButton21: Phaser.GameObjects.Sprite = this.add.sprite
83             (4 / 5 * this.cameras.main.width, 2 / 4 * this.cameras.main.
84             height, 'levelButton21');
85         const levelButton22: Phaser.GameObjects.Sprite = this.add.sprite
86             (3 / 5 * this.cameras.main.width, 2 / 4 * this.cameras.main.
87             height, 'levelButton22');
88         const levelButton23: Phaser.GameObjects.Sprite = this.add.sprite
89             (2 / 5 * this.cameras.main.width, 2 / 4 * this.cameras.main.
90             height, 'levelButton23');
91         const levelButton24: Phaser.GameObjects.Sprite = this.add.sprite
92             (1 / 5 * this.cameras.main.width, 2 / 4 * this.cameras.main.
93             height, 'levelButton24');
94         const levelButton31: Phaser.GameObjects.Sprite = this.add.sprite
95             (1 / 5 * this.cameras.main.width, 3 / 4 * this.cameras.main.
96             height, 'levelButton31');
97         const levelButton32: Phaser.GameObjects.Sprite = this.add.sprite
98             (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  }

```

A. Gotcha! – Full Code

```
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 &&
139                 gameObject.getData('clicked')) {
140                 this.buttonFunction(gameObject);
141             }
142
143             this.levelButtons.getChildren().forEach(function(gameObject) {
144                 if (gameObject instanceof Phaser.GameObjects.Sprite) {
145                     gameObject.clearTint();
146                     gameObject.setData('clicked', false);
147                 }
148             }, this);
149         }, this);
150     }
151
152     /**
153      * Method for assigning each button an event function
154      * @param gameObject GameObject on which you want the function on
155      */
156     private buttonFunction(gameObject: Phaser.GameObjects.Sprite): void
157     {
158         let name: string = this.buttonToSceneMap(gameObject.name);
159         let level: number = Number(name[name.length-1]);
160
161         this.sound.add('select').play();
162
163         switch (gameObject.name) {
164             case 'catButton': {
165                 this.transitionOut(name);
166                 break;
167             }
168
169             case 'eraseButton': {
170                 this.setStars(true);
171                 break;
172             }
173
174             case 'levelButton11': {
175                 name = name.substring(0, name.length-1);
176                 this.transitionOut(name, {'level': level});
177             }
178         }
179     }
180 }
```

```

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     }

```

A. Gotcha! – Full Code

```
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);

```

A. Gotcha! – Full Code

```
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         ();
326     bossField.fillStyle(0xfa7500, 0.3);
327     bossField.fillRoundedRect(4 / 5 * this.cameras.main.width - size
328         /2, 3 / 4 * this.cameras.main.height - size/2, size, size,
329         this.buttonSize/20);
330
331     const bossTween: Phaser.Tweens.Tween = this.tweens.add({
332         targets: bossField,
333         alpha: 0.1,
334         ease: 'Linear',
335         repeat: 1000,
336         yoyo: true,
337         duration: 1000
338     });
339 }
340
341 /**
342  * Method for initializing sound effects
343  */
344 private initAudio(): void {
345     this.sound.add('loading').play('', {loop: true});
346 }
347
348 /**
349  * Method for retrieving scene the respective button leads to with
350  * level at the end of the string
351  * @param buttonName Name of the button
352  */
353 private buttonToSceneMap(buttonName: string): string {
354     let ret: string = "";
355
356     switch (buttonName) {
357         case 'catButton': {
358             ret = 'SortingScene';
359             break;
360         }
361         case 'levelButton11': {
362             ret = 'PropertySortingScene1';
363             break;
364         }
365         case 'levelButton12': {
366             ret = 'PropertySortingScene2';
367             break;
368         }
369         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;

```

A. Gotcha! – Full Code

```
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 &&
436             gameObject.name !== "catButton" && gameObject.name !== '
437             eraseButton') {
438             let score: string = 'star_0';
439             if (typeof(Storage) !== "undefined") {
440                 if (window.localStorage.getItem('phaser_score_' +
441                     this.buttonToSceneMap(gameObject.name))) {
442                     if (reset) {
443                         window.localStorage.setItem('phaser_score_'
444                             + this.buttonToSceneMap(gameObject.name)
445                             , score);
446                     } else {
447                         score = window.localStorage.getItem('
448                             phaser_score_' + this.buttonToSceneMap(
449                             gameObject.name));
450                     }
451                 }
452             } else {
453                 console.log("Sorry! No Web Storage support...");
454             }
455
456             const star: Phaser.GameObjects.Sprite = this.add.sprite(
457                 gameObject.getBottomCenter().x, gameObject.
458                 getBottomCenter().y, score);
459             const scale: number = this.imageScalingFactor((
460                 gameObject.getTopRight().x - gameObject.getTopLeft().
461                 x)/2, star.width, star.height);
462             star.setScale(scale);
463             star.setOrigin(0.5, 0.5);
464         }
465     }, this);
466 }
```

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        // Initialize data from previous scene
47        this.jsonObject = this.cache.json.get('objects');
48
49        // Initialize fields
50        this.arrayStack = this.add.container(0, 0);
51        this.arrayCategory = this.add.group();
52        this.objectDisplaySize = 100;
53        this.objectsPerProperty = 5;
54        this.buttonSize = 64;
55    }
56
57    preload(): void {
58    }
59
60    create(): void {
61        // Bring MenuUI to the front and initialize transition

```

A. Gotcha! – Full Code

```
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
79  * property of each category has X representatives.
80  */
81 private imagePreSelection(): void {
82     // Select an manageable amount of images to be displayed
83     const selectiveArray: any[] = [];
84     const originArray: any[] = [...this.jsonObject['images']];
85
86     // Select category to add
87     for (let category of this.jsonObject['categories']) {
88         const temporaryArray: any[] = [];
89
90         // Select property to add
91         for (let property of category['validElements']) {
92             Phaser.Math.RND.shuffle(originArray);
93             // Check how many are needed in already selected
94             // elements
95             let missing: number = this.objectsPerProperty;
96             for (let selectedImage of selectiveArray) {
97                 if (missing <= 0) {
98                     break;
99                 }
100
101                 if (selectedImage[category.name] === property.name) {
102                     missing--;
103                 }
104             }
105
106             // Add number of needed images per property
107             for (let image of originArray) {
108                 if (missing <= 0) {
109                     break;
110                 }
111
112                 if (image[category.name] === property.name) {
113                     temporaryArray.push(image);
114                     missing--;
```

```

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,
134         0, 'background4');
135     background.setOrigin(0, 0);
136     background.setDisplaySize(this.cameras.main.width, this.cameras.
137         main.height);
138     background.setTint(0xffccaa);
139     background.setAlpha(0.9);
140 }
141
142 /**
143  * Method for initializing the control bar
144  */
145 private setControlBar(): void {
146     const controlbar: Phaser.GameObjects.Sprite = this.add.sprite(
147         this.cameras.main.width / 2, this.cameras.main.height, '
148         menubackground');
149     controlbar.setOrigin(0.5, 0.5);
150     controlbar.setAngle(-90);
151     controlbar.setScale(0.13, 0.20);
152
153     // Category indicator
154     let x: number = this.cameras.main.width / 2 - (controlbar.height
155         * 0.24) / 2;
156     let countCategories: number = 0;
157
158     // Find out how many categories not null exist
159     for (let cat of this.jsonObject['categories']) {
160         if (cat.url === null) {
161             continue;
162         }
163         countCategories++;
164     }
165
166     for (let cat of this.jsonObject['categories']) {
167         if (cat.url === null) {
168             continue;
169         }
170     }

```

A. Gotcha! – Full Code

```
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

```

A. Gotcha! – Full Code

```
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
323     grid
324  * @param categoryName Name of the category
325  * @param validElements Properties of this category
326  */
327 private orderObjects(categoryName: string, validElements: string[]):
328     void {
329     this.arrayStack.each(function(gameObject) {
330         if (gameObject instanceof Phaser.GameObjects.Sprite) {
331             let coords: number[] = this.returnQuad(validElements.
332                 indexOf(gameObject.getData(categoryName)),
333                 validElements.length);
334             gameObject.setPosition(coords[0], coords[1]);
335         }
336     }, this);
337 }
338
339 /**
340  * Method for initializing sound effects
341  */
342 private initAudio(): void {
343     this.sound.add('space').play('', {loop: true});
344 }
345
346 /**
347  * Returns random coordinates in the requested quadrant of total
348     quadrants.
349  * Starting by 0, from left to right then from top to bottom.
350  * IMPORTANT: Sprite origin has to be (0.5, 0.5)!
351  * @param quadrant Number of the quadrant
352  * @param quadrantType Number of quadrants
353  */
354 private returnQuad(quadrant: number, quadrantType: number): number[]
355 {
356     let ret: number[] = null;
357
358     if (quadrant >= quadrantType) {
359         console.log('ERROR: quadrant >= quadrantType');
360         return ret;
361     }
362
363     const spriteSizeHalf: number = this.objectDisplaySize / 2 + 20;
364
365     const leftOffsite: number = 100;
366     const rightOffsite: number = 0;
367     const topOffsite: number = 0;
368     const bottomOffsite: number = 100;
369
370     // Has entries dependant of
371     const horizontal: number[] = [];

```

A. Gotcha! – Full Code

```
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 -
377             leftOffsite - rightOffsite) / 3);
378         horizontal.push(leftOffsite + (this.cameras.main.width -
379             leftOffsite - rightOffsite) * 2 / 3);
380         break;
381     }
382     case 4: {
383         horizontal.push(leftOffsite + (this.cameras.main.width -
384             leftOffsite - rightOffsite) / 2);
385         vertical.push(topOffsite + (this.cameras.main.height -
386             topOffsite - bottomOffsite) / 2);
387         break;
388     }
389     case 6: {
390         horizontal.push(leftOffsite + (this.cameras.main.width -
391             leftOffsite - rightOffsite) / 3);
392         horizontal.push(leftOffsite + (this.cameras.main.width -
393             leftOffsite - rightOffsite) * 2 / 3);
394         vertical.push(topOffsite + (this.cameras.main.height -
395             topOffsite - bottomOffsite) / 2);
396         break;
397     }
398     default: {
399         break;
400     }
401 }
402
403 horizontal.push(this.cameras.main.width - rightOffsite);
404 vertical.push(this.cameras.main.height - bottomOffsite);
405
406 switch (quadrantType) {
407     case 3: {
408         ret = [Phaser.Math.RND.between(horizontal[quadrant] +
409             spriteSizeHalf, horizontal[quadrant + 1] -
410             spriteSizeHalf), Phaser.Math.RND.between(vertical[0]
411             + spriteSizeHalf + this.cameras.main.height / 8,
412             vertical[1] - spriteSizeHalf - this.cameras.main.
413             height / 8)];
414         break;
415     }
416     case 4: {
417         if (quadrant < 2) {
418             ret = [Phaser.Math.RND.between(horizontal[quadrant]
419                 + spriteSizeHalf, horizontal[quadrant + 1] -
420                 spriteSizeHalf), Phaser.Math.RND.between(
421                 vertical[0] + spriteSizeHalf, vertical[1] -
422                 spriteSizeHalf)];
423         }
424     }
425 }
```

```

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
9       * properties
10      */
11      private jsonObject: any;
12
13      /**
14       * Array index number +1 of category to sort
15      */
16      private setCat: number;

```

A. Gotcha! – Full Code

```
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 {

```

A. Gotcha! – Full Code

```
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 -
159     1]['validElements']) {
160     numberOfProperties++;
161 }
162
163 // Randomization of amount of properties to sort
164 this.propertyCount = Phaser.Math.RND.between(3,
165     numberOfProperties);
166
167 // Debatable initializations
168 this.objectDisplaySize = 100;
169 this.droppedObjectScale = 0.4;
170 this.velocity = 150;
171 this.lastEmitTime = 0;
172 this.delay = 1500;
173 this.numberOfObjectsEach = Phaser.Math.RND.between(3, 5);
174 }
175
176 preload(): void {
177
178 }
179
180 create(): void {
181     // Bring MenuUI to the front and initialize transition
182     this.game.scene.sendToBack(this.getKey());
183     this.transitionIn();
184 }
```



```

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.
204                     Math.RND.pick(this.arrayStatic.getChildren());
205                 this.arrayStack.bringToTop(sprite);
206                 this.arrayStatic.remove(sprite);
207                 sprite.setVelocityY(this.velocity);
208                 sprite.setAngularVelocity(sprite.getData('spin'));
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.
225             main.height);
226         background.setTint(0xffccaa);
227         background.setAlpha(0.9);
228     }
229
230     /**
231      * Methods for initializing the drop zones
232      */
233     private setDropzones(): void {
234         const leftBound: number = this.correctBar.getTopRight().x + 10;
235         const stepSize: number = (this.cameras.main.width - leftBound) /
236             (this.propertyCount);

```

A. Gotcha! – Full Code

```
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     ) {
280         if (gameObject instanceof Phaser.Physics.Arcade.Sprite) {
281             gameObject.setPosition(dragX, dragY);
282         }
283     });
284 }
```

```

282     }, this);
283
284     // On stop dragging
285     this.input.on('dragend', function(pointer, gameObject, dropped)
286     {
287         // If not dropped set default visual effects
288         if (!dropped && gameObject instanceof Phaser.Physics.Arcade.
289             Sprite) {
290             if (gameObject.getData('active')) {
291                 gameObject.clearTint();
292             }
293
294             let scale: number = gameObject.getData('scale');
295             gameObject.setScale(scale);
296
297             if (this.infinite) {
298                 gameObject.setVelocityY(this.velocity);
299                 gameObject.setAngularVelocity(gameObject.getData('
300                     spin'));
301             }
302
303             let x: number = gameObject.x;
304             let y: number = gameObject.y;
305             let dist: number = Math.sqrt(Math.pow(gameObject.width *
306                 gameObject.getData('scale'), 2) + Math.pow(
307                 gameObject.height * gameObject.getData('scale'), 2))
308                 / 2;
309
310             if (x < 0) {
311                 x = dist;
312             }
313
314             if (y < 0) {
315                 y = dist;
316             }
317
318             if (x > this.cameras.main.width) {
319                 x = this.cameras.main.width - dist;
320             }
321
322             if (y > this.cameras.main.height) {
323                 y = this.cameras.main.height - dist;
324             }
325
326             gameObject.setPosition(x, y);
327         }
328     }, this);
329
330     this.input.on('drop', function(pointer, gameObject, dropZone) {
331         if (gameObject instanceof Phaser.Physics.Arcade.Sprite &&
332             dropZone instanceof Phaser.GameObjects.Zone) {
333             let coords: number[] = [gameObject.input.dragStartX,
334                 gameObject.input.dragStartY];
335             let scale: number = gameObject.getData('scale');
336             let point: number = -1;
337         }
338     });

```

A. Gotcha! – Full Code

```
330         if (gameObject.name === dropZone.name && gameObject.  
331             getData('active')) {  
332             coords = [dropZone.x + dropZone.width * 0.15,  
333                     dropZone.y - dropZone.height * 0.2];  
  
334             scale = this.imageScalingFactor(Math.min(dropZone.  
335                 width, dropZone.height) * this.  
336                 droppedObjectScale, gameObject.width, gameObject.  
337                 height);  
  
338             this.arrayDropped.add(gameObject);  
  
339             gameObject.disableInteractive();  
340             gameObject.setImmovable(true);  
  
341             point = +1;  
342         } else {  
343             if (this.infinite) {  
344                 gameObject.setVelocityY(this.velocity);  
345                 gameObject.setAngularVelocity(gameObject.getData  
346                     ('spin'));  
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.  
371                 objectDisplaySize, this.objectDisplaySize * 1.3);  
  
372             const sprite: Phaser.Physics.Arcade.Sprite = this.  
373                 physics.add.sprite(Phaser.Math.RND.between(100 +  
374                     this.objectDisplaySize / 2, this.cameras.main.width  
375                     - this.objectDisplaySize / 1.5), Phaser.Math.RND.  
376                     between(this.objectDisplaySize / 2, this.cameras.  
377                     main.height - this.objectDisplaySize * 2 - this.  
378                     objectDisplaySize / 2), this.selectedElementsName[
```

```

373         this.selectedElements.indexOf(propImage));
374     sprite.setName(propImage.name);
375
376     if (this.infinite) {
377         sprite.setY(0 - 2 * this.objectDisplaySize);
378     }
379
380     sprite.setVelocity(0, 0);
381     sprite.setOrigin(0.5, 0.5);
382
383     // RND spin
384     sprite.setAngle(Phaser.Math.RND.angle());
385
386     sprite.setVisible(true);
387
388     const spriteScale: number = this.imageScalingFactor(size
389         , sprite.width, sprite.height);
390     sprite.setScale(spriteScale);
391
392     sprite.setData('scale', spriteScale);
393     sprite.setData('spin', Phaser.Math.RND.between(10, 50));
394     sprite.setData('active', true);
395
396     sprite.setInteractive({cursor: 'pointer'});
397
398     this.arrayStatic.add(sprite);
399     this.arrayStack.add(sprite);
400
401     this.input.setDraggable(sprite);
402 }
403
404 let rndDummy: number = 0;
405
406 if (this.infinite) {
407     rndDummy = Phaser.Math.RND.between(0, this.
408         numberOfObjectsEach / 2);
409 }
410
411 this.numberOfDummies += rndDummy;
412
413 for (let i = 0; i < rndDummy; i++) {
414     // RND size
415     const size: number = Phaser.Math.RND.between(this.
416         objectDisplaySize * 0.8, this.objectDisplaySize *
417         1.3);
418
419     const sprite: Phaser.Physics.Arcade.Sprite = this.
420         physics.add.sprite(Phaser.Math.RND.between(100 +
421             this.objectDisplaySize / 2, this.cameras.main.width
422             - this.objectDisplaySize / 2), Phaser.Math.RND.
423             between(this.objectDisplaySize / 2, this.cameras.
424                 main.height - this.objectDisplaySize * 2 - this.
425                 objectDisplaySize / 2), this.selectedElementsName[
426                 this.selectedElements.indexOf(propImage)]);
427     sprite.setName(propImage.name);
428 }

```

A. Gotcha! – Full Code

```
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(0xffffffff);
425
426         // RND spin
427         sprite.setAngle(Phaser.Math.RND.angle());
428
429         sprite.setVisible(true);
430
431         const spriteScale: number = this.imageScalingFactor(size
432             , sprite.width, sprite.height);
433         sprite.setScale(spriteScale);
434
435         sprite.setData('scale', spriteScale);
436         sprite.setData('spin', Phaser.Math.RND.between(10, 50));
437         sprite.setData('active', false);
438
439         sprite.setInteractive({cursor: 'pointer'});
440
441         this.arrayStatic.add(sprite);
442         this.arrayStack.add(sprite);
443
444         this.input.setDraggable(sprite);
445     }
446
447     this.maxPoints = this.arrayStack.length - this.propertyCount -
448         this.numberOfDummies;
449
450     const floor: Phaser.Physics.Arcade.Sprite = this.physics.add.
451         sprite(0, this.cameras.main.height + 2 * this.
452             objectDisplaySize, 'background');
453     floor.setDisplaySize(this.cameras.main.width, 1);
454     floor.setTintFill(0x000000);
455     floor.setOrigin(0, 0);
456     floor.setImmovable(true);
457
458     this.physics.add.collider(this.arrayStack.getAll(), floor,
459         function(gameObject1) {
460             if (gameObject1 instanceof Phaser.Physics.Arcade.Sprite) {
461                 if (gameObject1.getData('active')) {
462                     this.updateProgressbar(-1);
463                 }
464                 gameObject1.setVelocityY(0);
465                 gameObject1.setAngularVelocity(0);
466                 gameObject1.setPosition(Phaser.Math.RND.between(100 +
467                     this.objectDisplaySize / 2, this.cameras.main.width
468                     - this.objectDisplaySize / 2), 0 - 2 * this.
469                     objectDisplaySize);
470                 this.arrayStatic.add(gameObject1);
471                 this.arrayFalling.remove(gameObject1);
472             }
473         });
```

```

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
477             .getData('active')) {
478             const spriteName: string = sprite.name;
479             if (!(counterSet.indexOf(spriteName) > -1)) {
480                 for (let dropZone of this.arrayDropZone.getChildren
481                     ()) {
482                     if (dropZone instanceof Phaser.GameObjects.Zone)
483                     {
484                         if (dropZone.name === spriteName) {
485                             sprite.clearTint();
486                             sprite.setPosition(dropZone.x + dropZone
487                                 .width * 0.15, dropZone.y - dropZone
488                                 .height * 0.2);
489
490                             const imageScale: number = this.
491                                 imageScalingFactor(Math.min(dropZone
492                                     .width, dropZone.height) * this.
493                                     droppedObjectScale, sprite.width,
494                                     sprite.height);
495                             sprite.setScale(imageScale);
496
497                             this.arrayDropped.add(sprite);
498                             sprite.disableInteractive();
499                             this.arrayStatic.remove(sprite);
500
501                             this.arrayStack.sendToBack(sprite);
502                         }
503                     }
504                 }
505             }
506             counterSet.push(spriteName);
507         }
508     }
509
510     /**
511     * Method for initializing the progressbar
512     */

```

A. Gotcha! – Full Code

```
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 *
554         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') /
567             this.maxPoints;
567         this.correctBar.setScale(this.correctBar.getData('gameX'),
568             this.correctCount);
569
570     } else {
571         // Add to minus: max not defined (lets say number of cards
572         // minus three also...)
573         this.wrongCount += this.wrongBar.getData('gameMax') / this.
574             maxPoints;
575         this.wrongBar.setScale(this.wrongBar.getData('gameX'), this.
576             wrongCount);
577
578     }
579
580     if ((this.wrongCount >= this.wrongBar.getData('gameMax') -
581         Phaser.Math.EPSILON) || (this.correctCount >= this.
582         correctBar.getData('gameMax') - Phaser.Math.EPSILON)) {
583         this.transitionOut('ScoreScene', {
584             'score': this.correctCount / this.correctBar.getData('
585                 gameMax') - this.wrongCount / this.wrongBar.getData(
586                     'gameMax'),
587             'previousScene': this.getKey() + String(this.level)
588         });
589     }
590 }
591
592 /**
593  * Method for initializing sound effects
594  */
595 private initAudio(): void {
596     if (this.infinite) {
597         this.sound.add('battle').play('', {loop: true});
598     } else {
599         this.sound.add('space').play('', {loop: true});
600     }
601 }
602
603 /**
604  * Method for preselecting objects

```

A. Gotcha! – Full Code

```
598     */
599     private preselectObjects(): void {
600         // Preselect properties
601         for (let property of this.jsonObject['categories'][this.setCat -
602             1].validElements) {
603             this.selectedElements.push(property);
604         }
605
606         // Pick the elements
607         while (this.selectedElements.length > this.propertyCount) {
608             this.selectedElements = Phaser.Math.RND.shuffle(this.
609                 selectedElements);
610             this.selectedElements.pop();
611         }
612
613         // Get property images
614         let propLength: number = this.selectedElements[0].urls.length;
615         let rndIndex: number = Phaser.Math.RND.between(0, propLength -
616             1);
617         for (let prop of this.selectedElements) {
618             const name = prop.urls[rndIndex];
619             this.selectedElementsName.push(name);
620         }
621     }
622 }
```

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
8       * properties
9       */
10     private jsonObject: any;
11
12     /**
13      * Global average display size of interactive objects
14      */
15     private objectSize: number;
16
17     /**
18      * Two arrays which map the dropped objects to the respective zone
19      * and vice versa
20      */
21     private zoneObjMap: Phaser.GameObjects.Zone[];
22     private objZoneMap: Phaser.GameObjects.Sprite[];
23
24     /**
```

```

23     * Array of preselected objects so that not all objects must be
24     loaded
25     */
26     private preselectedObjects: any[];
27
28     /**
29     * Array of all displayed and not dropped objects
30     */
31     private displayedObjects: Phaser.GameObjects.Group;
32
33     constructor() {
34         super('RestrictedSortingScene');
35     }
36
37     init(data): void {
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,
75             0, 'background4');
76         background.setOrigin(0, 0);
77         background.setDisplaySize(this.cameras.main.width, this.cameras.

```

A. Gotcha! – Full Code

```
        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;

```

A. Gotcha! – Full Code

```
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     }, this);
240
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)
                ;

```

A. Gotcha! – Full Code

```
258         if (typeof this.objZoneMap[index1] == 'undefined' &&
259             this.equalityCheck(gameObject, dropZone)) {
260             // Check if the gameObject is already in a zone
261             if (index2 > -1) {
262                 // Clear gameObject from this zone
263                 delete this.objZoneMap[index2];
264                 this.displayedObjects.add(gameObject);
265             }
266
267             // Set scale and coordinates
268             scale = this.imageScalingFactor(Math.min(dropZone.
269                 width, dropZone.height) * 0.9, gameObject.width,
270                 gameObject.height);
271             coords = [dropZone.getCenter().x, dropZone.getCenter
272                 ().y];
273
274             // Add object to the dropZone
275             this.objZoneMap[index1] = gameObject;
276
277             // Remove from displayed array
278             this.displayedObjects.remove(gameObject);
279
280             // If all elements are sorted, end game with score
281             if (this.displayedObjects.getLength() <= 0) {
282                 this.transitionOut('ScoreScene', {
283                     'score': 1,
284                     'previousScene': this.getKey() + String(this
285                         .level)
286                 });
287             }
288
289             } else if (index2 > -1) {
290                 // Check if the gameObject was already in a zone
291                 let dropZoneOld: Phaser.GameObjects.Zone = this.
292                     zoneObjMap[index2];
293                 scale = this.imageScalingFactor(Math.min(dropZoneOld
294                     .width, dropZoneOld.height) * 0.9, gameObject.
295                     width, gameObject.height);
296             }
297
298             // Set default visual effect and position
299             gameObject.clearTint();
300             gameObject.setScale(scale);
301             gameObject.setPosition(coords[0], coords[1]);
302         }, this);
303     }
304
305     /**
306      * Method which initializes all the displayed object to sort
307      */
308     private setObjects(): void {
309         for (let image of this.preselectedObjects) {
310             const x: number = Phaser.Math.RND.between(100 + this.
311                 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

```

A. Gotcha! – Full Code

```
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
369             // Choose a random category
370             const rndCat: any = Phaser.Math.RND.shuffle(categories)[0];
371
372             // Select random fitting images
373             const images: any[] = [...this.jsonObject['images']];
374             Phaser.Math.RND.shuffle(images);
375
376             for (let property of Phaser.Math.RND.shuffle(rndCat['
377                 validElements']).slice(0, 3)) {
378
379                 // Choose for one category 2, for the other 4 and for
380                 // the last 6 matching (in one property) images.
381                 let maxSize: number = 2;
382                 if (this.preselectedObjects.length === 2) {
383                     maxSize = 6;
384                 } else if (this.preselectedObjects.length === 6) {
385                     maxSize = 12;
386                 }
387
388                 // Iterate through the images until selecting criteria
389                 // is fulfilled
390                 for (let image of images) {
391                     // If selected enough images, break.
392                     if (this.preselectedObjects.length >= maxSize) {
393                         break;
394                     }
395
396                     // Load and add image if is has the same property as
397                     // the selected one
398                     if (image[rndCat.name] === property.name) {
399                         this.preselectedObjects.push(image);
400                     }
401                 }
402             }
403         }
404     }
405 }
```

```

398         // Remove the selected images from the images array to
399         // avoid duplicates
400         this.preselectedObjects.forEach(function (element) {
401             if (images.indexOf(element, 0) > -1) {
402                 images.splice(images.indexOf(element, 0), 1);
403             }
404         }, this);
405     }
406 } else {
407
408     // Copy category array
409     const categories: any[] = [...this.jsonObject['categories'
410         ]];
411
412     // Choose a random category
413     const rndCat: any = Phaser.Math.RND.shuffle(categories)[0];
414
415     // Select random fitting images
416     const images: any[] = [...this.jsonObject['images']];
417     Phaser.Math.RND.shuffle(images);
418
419     for (let property of Phaser.Math.RND.shuffle(rndCat['
420         validElements']).slice(0, 3)) {
421
422         // Choose for one category 2, for the other 4 and for
423         // the last 6 matching (in one property) images.
424         let maxSize: number = 5;
425         if (this.preselectedObjects.length === 5) {
426             maxSize = 9;
427         } else if (this.preselectedObjects.length === 9) {
428             maxSize = 15;
429         }
430
431         // Iterate through the images until selecting criteria
432         // is fulfilled
433         for (let image of images) {
434             // If selected enough images, break.
435             if (this.preselectedObjects.length >= maxSize) {
436                 break;
437             }
438
439             // Load and add image if is has the same property as
440             // the selected one
441             if (image[rndCat.name] === property.name) {
442                 this.preselectedObjects.push(image);
443             }
444         }
445
446         // Remove the selected images from the images array to
447         // avoid duplicates
448         this.preselectedObjects.forEach(function (element) {
449             if (images.indexOf(element, 0) > -1) {
450                 images.splice(images.indexOf(element, 0), 1);
451             }
452         }, 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
13       * waiting for the animation to finish
14      */
15     private lock: boolean;
16
17     /**
18      * State of the helper menu and data
19      */
20     private helpDown: boolean;
21     private buttonSize: number;
22
23     /**
24      * Lock for checking the marked objects
25      */
26     private checked: boolean;
27
28     /**
29      * Array of objects in play
30      */
31     private gameSet: any[];
32
33     /**
34      * Copy of the categories
35      */
36     private categorySet: any[];
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');

```

A. Gotcha! – Full Code

```
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             ,
181             'previousScene': this.getKey() + String(this.level)
182         });
183     } else {
184         timedata -= this.timedataStepsize;
185         this.timefluid.setData('timeY', timedata);
186         this.timefluid.setScale(this.timefluid.getData('timeX'),
            timedata);
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']) {

```

A. Gotcha! – Full Code

```
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
222     // copy the array instead of referencing
223
224     // If level one, ignore the last category
225     if (this.level === 1) {
226         this.categorySet.pop();
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.
236             main.height);
237         background.setTint(0xffccaa);
238         background.setAlpha(0.9);
239     }
240
241     /**
242     * Function for creating the helper menu
243     */
244     private setHelperMenu(): void {
245         // Menu background
246         const backgroundY: number = 64 + 10 + 30;
247         const menuBackground: Phaser.GameObjects.Sprite = this.add.
248             sprite(this.cameras.main.width - 64 - 10 - 30, backgroundY,
249                 'menubackground');
250         menuBackground.setAngle(180);
251         menuBackground.setOrigin(1, 0);
252         menuBackground.setDisplaySize(500, this.cameras.main.height +
253             120);
254         menuBackground.setTint(0xdddddd);
255         menuBackground.setData('y', backgroundY);
256
257         // Category indicator
258         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
296  * initialization
297  */
298 private loadObjects(): void {
299     for (let image of this.gameSet) {
300         const name: string = image.name;
301         const cat1: string = image.cat1;
302         const cat2: string = image.cat2;
303         const cat3: string = image.cat3;
304         const cat4: string = image.cat4;

```

A. Gotcha! – Full Code

```
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.
        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                     ) {

```

A. Gotcha! – Full Code

```
401         if (inGame) {
402             categoryIndicator.setTintFill(0x00dd00);
403         }
404     } else if (
405         !(sprite1.getData(categoryIndicator.name) ===
406           sprite2.getData(categoryIndicator.name)) &&
407         !(sprite2.getData(categoryIndicator.name) ===
408           sprite3.getData(categoryIndicator.name)) &&
409         !(sprite1.getData(categoryIndicator.name) ===
410           sprite3.getData(categoryIndicator.name))
411     ) {
412         if (inGame) {
413             categoryIndicator.setTintFill(0x00dd00);
414         }
415     } else {
416         if (replaceObjects) {
417             replaceObjects = false;
418         }
419         if (inGame) {
420             // Mark category as red
421             categoryIndicator.setTintFill(0xdd0000);
422         }
423     }
424 }
425 return replaceObjects;
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             const newSprite: Phaser.GameObjects.Sprite = Phaser.
445                 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 resetting 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

```

A. Gotcha! – Full Code

```

        ': this.getKey() + String(this.level)'));
503
504     }
505
506     this.gamefluid.setScale(this.gamefluid.getData('gameX'), this.
        points);
507
508     }
509
510     /**
511      * Function for checking if there is a occurrence of a set in the
512      * displayed objects
513      */
514     private setEqualityCheck(): void {
515         const objectSet: Phaser.GameObjects.GameObject[] = this.
516             arrayDisplayed.getChildren();
517         const objectSetLength: number = objectSet.length;
518
519         for (let x = 0; x <= objectSetLength; x++) {
520             for (let y = x + 1; y <= objectSetLength - (x + 1); y++) {
521                 for (let z = y + 1; z <= objectSetLength - (y + 1); z++) {
522                     if (this.checkEquality(objectSet[x], objectSet[y],
523                         objectSet[z], false)) {
524                         return;
525                     }
526                 }
527             }
528         }
529
530         // Replace/add cards
531         this.rebuildDisplayedObjects();
532     }
533
534     /**
535      * Function for refreshing all displayed objects with new ones
536      * Usually used if there is no occurrence of a set.
537      */
538     private rebuildDisplayedObjects(): void {
539
540         // Replace all cards
541         for (let card of this.arrayDisplayed.getChildren()) {
542             if (card instanceof Phaser.GameObjects.Sprite) {
543                 card.setVisible(false);
544                 this.arrayStack.add(card);
545             }
546         }
547
548         this.arrayDisplayed.clear(false, false);
549
550         this.initObjects();
551     }
552
553     /**
554      * Function for initializing the animation on the helpers menu
555      * @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
568                 .add({
569                 targets: menuBackground,
570                 y: menuBackground.getData('y'),
571                 x: this.cameras.main.width - 64 - 10 - 30,
572                 ease: 'Cubic',
573                 duration: 500,
574                 delay: 100
575             });
576
577             for (let helperButton of this.arrayCategory.getChildren()) {
578                 let helperButtonTween: Phaser.Tweens.Tween = this.tweens
579                     .add({
580                     targets: helperButton,
581                     x: this.cameras.main.width + 64,
582                     ease: 'Cubic',
583                     duration: 300
584                 });
585             }
586
587             this.helpDown = false;
588
589             // Extend
590             } else {
591                 let menuBackgroundTween: Phaser.Tweens.Tween = this.tweens.
592                     add({
593                     targets: menuBackground,
594                     y: this.cameras.main.height + 90,
595                     x: this.cameras.main.width - 64 - 10 - 50,
596                     ease: 'Cubic',
597                     duration: 500
598                 });
599
600                 for (let helperButton of this.arrayCategory.getChildren()) {
601                     let helperButtonTween: Phaser.Tweens.Tween = this.tweens
602                         .add({
603                         targets: helperButton,
604                         x: this.cameras.main.width - (16 + 32) + 4,
605                         ease: 'Cubic',
606                         duration: 300
607                     });
608                 }
609             }
610         }

```

A. Gotcha! – Full Code

```
605         }
606
607         this.helpDown = true;
608     }
609 }
610
611 /**
612  * Function for initializing the progressbar for ingame game score
613  */
614 private setGameProgressbar(): void {
615     const progressbarY: number = this.cameras.main.height - 10;
616     const progressbar: Phaser.GameObjects.Sprite = this.add.sprite
617         (0, progressbarY, 'progressbar');
618     const multiplierX: number = 0.4;
619     const multiplierY: number = this.imageScalingFactor(this.cameras
620         .main.height * 0.5, progressbar.height, progressbar.height);
621     //0.3;
622     progressbar.setOrigin(0, 1);
623     progressbar.setScale(multiplierX, multiplierY);
624
625     const progressbarX: number = 10 * 2 + progressbar.width *
626         multiplierX;
627     progressbar.setX(progressbarX);
628
629     const progressstar: Phaser.GameObjects.Sprite = this.add.sprite(
630         progressbarX, progressbarY - progressbar.height *
631         multiplierY - 10, 'progressstar');
632     const starmultiplier: number = progressbar.width * multiplierX /
633         progressstar.width;
634     progressstar.setOrigin(0, 1);
635     progressstar.setScale(starmultiplier, starmultiplier);
636
637     this.gamefluid = this.add.sprite(progressbarX + progressbar.
638         width * multiplierX / 2 + 2, progressbarY - 6, 'gamefluid');
639     this.gamefluid.setOrigin(0.5, 1);
640     this.gamefluid.setData('gameX', multiplierX);
641     this.gamefluid.setData('gameY', 0.01);
642     this.gamefluid.setData('gameMax', (progressbar.height *
643         multiplierY - 6) / this.gamefluid.height);
644     this.gamefluid.setScale(multiplierX, 0.01);
645     this.gamefluid.setAlpha(0.7);
646 }
647
648 /**
649  * Function for initializing the game timer
650  */
651 private setTimeProgressbar(): void {
652     const progressbarY: number = this.cameras.main.height - 10;
653     const progressbar: Phaser.GameObjects.Sprite = this.add.sprite
654         (10, progressbarY, 'progressbar');
655     const multiplierX: number = 0.4;
656     const multiplierY: number = this.imageScalingFactor(this.cameras
657         .main.height * 0.5, progressbar.height, progressbar.height);
658     //0.3;
659     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

```

A. Gotcha! – Full Code

```
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,
51             0, 'background5');
52         background.setOrigin(0, 0);
53         background.setDisplaySize(this.cameras.main.width, this.cameras.
54             main.height);
55         background.setInteractive({ cursor: 'pointer' });
56
57     /**
58      * Method for initializing replaybutton and reward graphics
59      */
60     private initUI(): void {
61         // Add replay button
62         const replayButton: Phaser.GameObjects.Sprite = this.add.sprite(
63             this.cameras.main.width - 100 + 34, this.cameras.main.height
64             - 100 + 34, 'replay');
65         replayButton.setOrigin(0.5, 0.5);
66         const buttonScale: number = this.imageScalingFactor(this.
67             buttonSize*1.5, replayButton.width, replayButton.height);
68         replayButton.setScale(buttonScale);
69         replayButton.setInteractive({ cursor: 'pointer' });
70         replayButton.on('pointerdown', function() {
71             replayButton.on('pointerup', function() {
72                 this.transitionOut(this.previousScene.substring(0, this.
73                     previousScene.length-1), {level: Number(this.
74                     previousScene[this.previousScene.length-1])});
75             });
76         });
77     }
78 }
```

```

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    /**

```

A. Gotcha! – Full Code

```
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.

Bibliography

- [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 Laboratories. <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 für algorithmisches 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, Zeichensprachen 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 für algorithmisches 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):

Gotscha! - One Way To Support Learning

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

Knobel

First name(s):

Franz

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

24. Januar 2020

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.