Gotscha! - One Way To Support Learning

Franz Knobel

Bachelor Thesis
January 16, 2020

Supervising Professor:
Prof. Dr. Juraj Hromkovič

Supervisor:
Elizabeta Cavar

ETH zürich

Abstract

UNDER CONSTRUCTION

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

UNDER CONSTRUCTION

I want to thank my supervisor Elizabeta Cavar for supporting me in the last six months. Without her input and constant reminder of our deadlines, I would have had a lot more obstacles on the way. Also many thanks to Jaqueline Staub, Nicole Trachsler and Philippe Voinov for their help with specific topics on my work. Another very important role in my work played the debuggers, the ones who tested my program and searched for any kind of problems. Thank you for your time, effort and fun you had with my work: Dilana Kunz, Guido von Burg, Julia Badertscher, Ronja Koeppel, Jonathan Chen

Contents

1 Introduction **3**
1.1 Introduction . . . 3
1.2 Problem Statement . . . 3
1.3 Section . . . 4

2 Related Work **5**
2.1 Previous work . . . 5
2.2 Gamification and Game Based Learning (GBL) . . . 5
2.3 Existing Educational Software . . . 5
2.4 Checklist . . . 6

3 Requirements **7**
3.1 Boundary conditions . . . 7
3.2 Evaluation of possible solutions . . . 7
3.3 Checklist . . . 8

4 Design of the Learning Environment **9**
4.1 Phaser 3 . . . 9
4.1.1 Base Configuration . . . 9
4.1.2 Scenes . . . 11
4.1.3 Managers . . . 12
4.2 Object Generation . . . 12
4.3 Objects . . . 15
4.3.1 Object Storage . . . 15
4.3.2 Object Display . . . 16
4.3.3 Object Interaction . . . 16

4.4 Score Storage . . . 17
4.5 Introduction Animation Generation . . . 18
4.6 Concept of the Learning Environment . . . 18
4.6.1 Drop Down Menu . . . 18
4.6.2 Welcome Screen . . . 19
4.6.3 Level Menu . . . 19
4.6.4 Object Summary . . . 19
4.6.5 Sorting with one Category . . . 19
4.6.6 Sorting with one category under difficult conditions . . . 19
4.6.7 Sorting with restricted space . . . 20
4.6.8 Object pairing - easy version . . . 20
4.6.9 Object pairing - normal version . . . 21
4.6.10 Score Screen . . . 22
4.6.11 Introduction . . . 22
4.7 Code structure of the learning environment . . . 22
4.7.1 BaseScene . . . 22
4.7.2 PreloadAssets . . . 23
4.7.3 DropDownMenu . . . 24
4.7.4 WelcomeScene . . . 25
4.7.5 LevelMenuSceneScene . . . 25
4.7.6 IntroScene . . . 26
4.7.7 SortingScene . . . 26
4.7.8 PropertySortingScene . . . 28
4.7.9 RestrictedSortingScene . . . 29
4.7.10 GameScene . . . 29
4.7.11 ScoreScene . . . 32
4.8 Final Look of the Learning Environment . . . 32

5 Evaluation **45**
5.1 Debugging . . . 45
5.1.1 Devices . . . 45
5.1.2 Differences and obstacles . . . 45
5.2 Target Audience Test . . . 45
5.3 Various Audience Test . . . 46
5.4 Checklist . . . 46

6 Conclusion **47**
6.1 Future Improvements . . . 47
6.2 checklist . . . 47

Bibliography **49**

UNDER CONSTRUCTION

1

Introduction

1.1 Introduction

1.2 Problem Statement

the background on and motivation for your research - technological trends of the area - open problems - recent promising developments - introduce more specific terminology which is not widely known. - Provide good motivation for your work, - explaining its technological, research or economic importance - The motivation simply two or three good reasons are enough to make your research important.

a summary and outline of your paper, telling readers what they should expect to find in it.

- a problem description, which is slightly more detailed than in the abstract. - a description of your solution - some arguments on its impacts - key concepts and categorize its approach.

Close your introduction with a description of your pa- per outline - what sections it contains - what the reader will find in each.

A proper flow 1. context 2. present your proposal 3. provide the verification 4. conclusions.

As the importance of computer science is rising, certain games that support the developement of abstract thinking, analytical skills and decision making, are becoming more and more interesting at an early age. Through identification and describing relationships between items, children develop a foundation to early math skills and basic concepts of computer science (e.g. combinatorics of finite affine and projective spaces, the theory of error-correcting codes, etc.) . 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 should receive a tool for their pupils. Focused on classification of objects with certain properties, the pupils get introduced to a computer-based learning environment. There they can individually train and improve themselves in this field. In the mean time the teachers can concentrate on the majority of their pupils and have the opportunity to work with pupils on an individual basis, without feeling the pressure of having to support everyone at once.

Background

The key contribution of computer science to general school education is rooted in the concept of *algorithmic thinking* . One way of introducing kindergarten and primary school pupils to algorithmic thinking and it's concepts consists in making them solve problems with and without computers. This can be achieved using age- and knowledge-appropriate learning materials. Several papers with corresponding online learning environments have been proposed. This work is going to be added to the implementation of "INFORMATIK BIBER in KG und 1/2" by Jil Weber . The focus of this work 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.

Goals of the Thesis

The main objective of this thesis consists of planning, analyzing, implementing and testing a computer-based learning environment on the topic of classification. The student studies the already existing implementations of "INFORMATIK BIBER in KG und 1/2", analyses the capabilities of kindergarten kids and first graders, develops an interactive classification tool, implements it then on a platform compatible with the implementation of "INFORMATIK BIBER in KG und 1/2" and conducts an evaluation with test subjects. We expect the student to find a suitable implementation that integrates neatly into our existing system mentioned before. The outcome of this thesis is a well-documented, stable and reliable prototype, providing the functional elements to be used in schools.

1.3 Section

SECTION 0

2

Related Work

UNDER CONSTRUCTION

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

2.1 Previous work

Sonja Tabea Blum Kevin Tang Sarah Kamp Jil Weber

2.2 Gamification and Game Based Learning (GBL)

4 5 6 7 8

2.3 Existing Educational Software

pro: player didnt recognize he was learning something con: every level needed the same level of knowledge in every category

====

2.4 Checklist

X - list of research works that are related to your paper—necessary to show what has happened in this field. X - critique of the approaches in the literature—necessary to establish the contribution and importance of your paper.

X distinguish and describe all the different approaches to the problem.

Critiquing the major approaches of the background work will enable you to identify the limitations of the other works and show that your research picks up where the others left off. This is a great opportunity to demon- strate how your work is different from the rest; for ex- ample, show whether you make different assumptions and hypotheses, or whether your approach to solving the problem differs.

DIE FRAGE: ein neues framework, einfach zu lernen, ja nein? aufwand, vorteile, nachteile

irgendwo mus ine, dass au lehrer abgneigt sind gege elektronik im taegliche gebruch. jedoch isches fact, dass dmehrheit vode chinder scho im junge alter mit dere war konfrontiert wird. de taeglichi gebruch isch sowieso da. wieso noed gad als guets bispil vorusgah und ihne spil schmackhaft mache wo ihne spass mached und si demit passiv lerned. motivation isch hoecher chind choennd scho hoechduetsch, bevor si ueberhaupt schwizerduetsch richtig choennd, will si mit duetscher sprach im gebruch vo elektronik konfrontiert werded. warum noed gad bruche als au soettigs und dete de vorteil drus zieh?! mr chan als bispil voragah ide informatik da das ersch jetzt gross ufchunnt, bi etablierte lehrmittel wird das schwiriger vorallem wills denne oftmals heisst, mit dem hanich scho glernt und das het funktioniert

3

Requirements

UNDER CONSTRUCTION

3.1 Boundary conditions

3.2 Evaluation of possible solutions

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 has conditions as well. Those will be taken into consideration while evaluating a suitable framework.

Target Audience

Elements of Gamification

Additional Conditions

Framework

human - kinder koennen nicht lesen und schreiben, keine groesseren zahlen und keine buchstaben

Game - capturing name - objects categories nad subcategories - easy level - easy level under harder difficulties - limited sorting -> hashing - set easy - set hard - overview - gamification - track progress - time based - correct and wrong - visual gratification - have fun dont recognize you are learning - optics - randomization and more options for variety - sound - story like - menu - levelmenu - fullscreen - flow - exitbutton - returnbutton -

3.3 Checklist

- explicitly describe all the hypotheses and assumptions of the environment on which the problem will be stated. - Put good effort in realizing all explicit and implicit assumptions that you make, and clearly state them. It is important to provide support for your assumption choices. The more valid and acceptable your assumptions are, the more valid and acceptable your work will be. The system model section should always have a figure. The figure should demonstrate the parameters of your system model. Prepare the figure so that it can later be reused or enhanced to demonstrate your solution.

Often, this section is merged with the system model. State your problem clearly. Be as exact as possible into stating what the question of the problem is. It reflects poorly upon an author if he cannot describe or does not know what problem his solution addresses. But most im- portantly, it will be easier for successive researchers to classify your work.

4

Design of the Learning Environment

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

4.1 Phaser 3

Phaser [**??**] is an open source HTML5 [**??**] game framework created by Photon Storm [**??**]. It is a JavaScript/TypeScript [**??**] 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 [**??**], otherwise it seamlessly reverts to Canvas [**??**]. The current version is 3.17 and is used in this work. Version 4 is currently in development.

4.1.1 Base Configuration

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

For example:

Listing 4.1: *Game Setup File*

```
import 'phaser';
import GameConfig = Phaser.Types.Core.GameConfig;
import RenderConfig = Phaser.Types.Core.RenderConfig;
```

```
4       import {Scene1} from ./scene1';
5       import {Scene2} from ./scene2';
6
7       // Defining the renderer
8       const renderConfig: RenderConfig = {
9           antialias: true,
10          pixelArt: false
11      };
12
13      // Enforcing widescreen
14      let width: number = window.screen.width;
15      let height: number = window.screen.height;
16
17      if (window.screen.width <= window.screen.height) {
18          width = window.screen.height;
19          height = window.screen.width;
20      }
21
22      // Game Configuration
23      const config: GameConfig = {
24          title: 'TITLE',
25          parent: 'game',
26          type: Phaser.AUTO,
27          scene: [
28              scene1, scene2
29          ],
30          physics: {
31              default: 'arcade',
32              arcade: {
33                  debug: false
34              }
35          },
36
37          // Master background color
38          backgroundColor: '#000000',
39
40          render: renderConfig,
41
42          scale: {
43              parent: 'phaser-example',
44              mode: Phaser.Scale.FIT,
45              autoCenter: Phaser.Scale.CENTER_BOTH,
46              width: width,
47              height: height
48          }
49      };
50
51      export class GameName extends Phaser.Game {
52          constructor(config: GameConfig) {
53              super(config);
54          }
55      }
56
57      // Event handler for starting the game
58      window.onload = () => {
59          const game = new GameName(config);
```

```
60      };
```

4.1.2 Scenes

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

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

Lifecycle

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

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

Figure 4.1: *Scene Lifecycle [??]*

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 special care must be taken to set a known initial value on everything requiring one. Especially when loading/creating objects outside the preloader (animations, tweens, audiofiles, etc.).

Scenes here have 5 functions:

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

4.1.3 Managers

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

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

4.2 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 [4.3] scalable vector graphic (SVG) [**??**] files are generated [4.4]. After that they are converted to portable network graphic (PNG)[**??**] files with the GNU image manipulation program (GIMP) [**??**]. 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 [4.2] the last step is easily applicable to all 1000+ images.

Listing 4.2: *ImageMagick console command "mogrify"*

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

Listing 4.3: *SVG Generation*

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

filename = colordefault + shape + number + fillingname

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

Listing 4.4: *SVG Image File*

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

<svg height="1000" width="1000" viewbox="0 0 1000 1000" xmlns="http
    ://www.w3.org/2000/svg">
<defs>
        <pattern id="stripe" patternUnits="userSpaceOnUse" width="
            20%" height="20%">
            <path stroke="aqua" stroke-linecap="butt" stroke-width="
                50" d="M -20 -20 l1000 1000"/>
            <path stroke="aqua" stroke-linecap="butt" stroke-width="
                50" d="M -20 180 l1000 1000"/>
            <path stroke="aqua" stroke-linecap="butt" stroke-width="
                50" d="M -20 -220 l1000 1000"/>

        </pattern>
        <pattern id="dotted" enable-background="true" patternUnits="
            userSpaceOnUse" width="15%" height="15%">
            <circle cx="30" cy="30" r="25" fill="aqua" />
            <circle cx="105" cy="105" r="25" fill="aqua" />
        </pattern>
        <style>
        .button {

        stroke-width:5;
```

```
19              stroke:black;
20
21              }
22          </style>
23          </defs>
24
25      <g id="circle" display="none">
26          <circle cx="500" cy="500" r="300" class="button" fill="blue"/>
27          <circle cx="500" cy="500" r="300" class="button" fill="none"/>
28          <circle cx="500" cy="500" r="250" class="button" fill="none"/>
29      </g>
30
31      <g id="square" display="inherit">
32       <rect x="250" y="250" rx="20" ry="20" width="500" height="500"
            class="button" fill="blue"/>
33       <rect x="250" y="250" rx="20" ry="20" width="500" height="500"
            class="button" fill="none"/>
34       <rect x="300" y="300" rx="20" ry="20" width="400" height="400"
            class="button" fill="none"/>
35      </g>
36
37      <g id="triangle" display="none">
38       <polygon points="500,50 113.4,700 886.6,700" stroke-linejoin="
            round" class="button" fill="blue" />
39       <polygon points="500,50 113.4,700 886.6,700" stroke-linejoin="
            round" class="button" fill="none" />
40       <polygon points="500,150 200,650 800,650" stroke-linejoin="round"
            class="button" fill="none"/>
41      </g>
42
43      <g id="ellipse" display="none">
44       <ellipse cx="500" cy="500" rx="400" ry="250" class="button" fill="
            blue" />
45       <ellipse cx="500" cy="500" rx="400" ry="250" class="button" fill="
            none" />
46       <ellipse cx="500" cy="500" rx="350" ry="200" class="button" fill="
            none" />
47      </g>
48      <g id="octagon" display="none">
49       <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" />
50       <polygon points="400,250 600,250 750,400 750,600 600,750 400,750
            250,600 250,400" stroke-linejoin="round" class="button" fill="
            none" />
51       <polygon points="400,300 600,300 700,400 700,600 600,700 400,700
            300,600 300,400" stroke-linejoin="round" class="button" fill="
            none"/>
52      </g>
53
54      <g id="rhombus" display="none">
55       <polygon points="350,250 150,750 650,750 850,250" stroke-linejoin=
            "round" class="button" fill="blue" />
56       <polygon points="350,250 150,750 650,750 850,250" stroke-linejoin=
            "round" class="button" fill="none" />
57       <polygon points="390,300 230,700 620,700 770,300" stroke-linejoin=
```

```
            "round" class="button" fill="none"/>
      </g>


      <g id="one" display="none">
       <circle cx="500" cy="500" r="40" stroke="black" fill="black"/>
      </g>

      <g id="two" display="none">
       <circle cx="570" cy="500" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="500" r="40" stroke="black" fill="black"/>
      </g>

     <g id="three" display="none">
       <circle cx="570" cy="560" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="560" r="40" stroke="black" fill="black"/>
       <circle cx="500" cy="440" r="40" stroke="black" fill="black"/>
      </g>

     <g id="four" display="none">
       <circle cx="570" cy="430" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="430" r="40" stroke="black" fill="black"/>
       <circle cx="570" cy="570" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="570" r="40" stroke="black" fill="black"/>
     </g>
     <g id="five" display="none">
       <circle cx="570" cy="430" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="430" r="40" stroke="black" fill="black"/>
       <circle cx="570" cy="570" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="570" r="40" stroke="black" fill="black"/>
       <circle cx="500" cy="500" r="40" stroke="black" fill="black"/>
     </g>

     <g id="six" display="none">
       <circle cx="570" cy="400" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="400" r="40" stroke="black" fill="black"/>
       <circle cx="570" cy="600" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="600" r="40" stroke="black" fill="black"/>
       <circle cx="430" cy="500" r="40" stroke="black" fill="black"/>
       <circle cx="570" cy="500" r="40" stroke="black" fill="black"/>
     </g>

     Sorry, your browser does not support inline SVG.
     </svg>
```

4.3 Objects

4.3.1 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) file.

Listing 4.5: *geometrical_objects.json*

```
{
  "categories": [
    {
      "name": "cat1",
      "url": "color.png",
      "validElements": [
        {
          "name": "purple",
          "urls": [
            "purple1.png",
            "purple2.png",
            "purple3.png",
            "purple4.png",
            "purple5.png",
            "purple6.png"
          ]
        },
        ...
      ]
    },
    ...
  ],
  "images": [
    {
      "name": "purplesquareonefull.png",
      "cat1": "purple",
      "cat2": "square",
      "cat3": "one",
      "cat4": "full"
    },
    ...
  ]
}
```

4.3.2 Object Display

Displaying just the image of an object without tweaking it a little can seem boring for the user in long term. To add more diversity, objects gets a random rotation angle, size and position every time it is displayed/created. Of course within certain predefined boundaries.

4.3.3 Object Interaction

Whenever possible a generalization of the input code 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*

```
...
this.input.on('dragstart', function(pointer, gameObject) {
    ...
}, this);

this.input.on('drag', function(pointer, gameObject, dragX, dragY) {
    ...
}, this);

this.input.on('dragend', function(pointer, gameObject, dropped) {
    ...
    if (!dropped ...) {
        ...
    }
}, this);

this.input.on('drop', function(pointer, gameObject, dropZone) {
    ...
}, this);
...
```

4.4 Score Storage

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

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

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

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

```
...
if (typeof(Storage) !== "undefined") {
    if(window.localStorage.getItem('phaser_score_' + this.
        buttonToSceneMap(gameObject.name))){
        if (reset) {
            window.localStorage.setItem('phaser_score_' + this.
                buttonToSceneMap(gameObject.name), score);
        } else {
```

```
                    score = window.localStorage.getItem('phaser_score_' +
                        this.buttonToSceneMap(gameObject.name));
                }
            }
        } else {
            console.log("Sorry! No Web Storage support...");
        }
        ...
```

4.5 Introduction Animation Generation

To produce animated instruction for the task later on, the screen is recorded with the Open Broadcaster Software (OBS) Studio [**??**] while someone is playing the game. 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]). The Phaser framework animates those spreadsheets.

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

Figure 4.2: *Example: Spreadsheet*

4.6 Concept of the Learning Environment

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

4.6.1 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.

4.6.2 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.

4.6.3 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.

4.6.4 Object Summary

The Object summary allows the user to get a feeling for all the different properties an object can have. For that reason the number of objects per category is restricted to five, as there are over 1000 different objects. Objects are draggable and sortable by each category with a click on a respective button.

4.6.5 Sorting with one Category

Here the user has to sort the static 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 motivation, 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.

4.6.6 Sorting with one category under difficult conditions

Here 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 level. Each level represents a category. For motivation, 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, dummy objects are added. 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.

4.6.7 Sorting with restricted space

In this scene 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.

4.6.8 Object pairing - easy version

In this game, twelve objects with three categories are shown. the user has to select three objects which have to fulfil the following. For each category (color, shape and number of holes/dots) one of the following conditions has to hold:

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

Figure 4.3: *Example: Matching Set*

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

coloring the category symbols on the bar in green or red.

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

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

4.6.9 Object pairing - normal version

In this game, twelve objects with four categories are shown. the user has to select three objects which have to fulfil the following. For each category (color, filling, shape and number of holes/dots) one of the following conditions has to hold:

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

Figure 4.4: *Example: Matching Set*

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

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

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

4.6.10 Score Screen

After a task has been completed, there is going to be a score. This score is represented here with a displayed number of stars. The minimum of starts 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.

4.6.11 Introduction

Before each game starts, a sample animation of the task beforehand is being shown. The current scene is paused in the meantime, so that the user has enough time to find out what he has to do.

4.7 Code structure of the learning environment

Our learning environment consists of different scenes each one inheriting the base scene. The following subsections won't contain the whole code as some parts are trivial to understand. Instead a summary of the scenes purpose and crucial parts are shown and explained.

The full code can be accessed in the appendix.

Figure 4.5: *Scene Statediagram*

4.7.1 BaseScene

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

The transition is a mask in the form of a circle laid over a black rectangle. Masks have to be applied to every object they have to mask. So a simple solution was to lay a black rectangle over the whole scene and mask it. The animated transition is created by increasing or decreasing the radius of the circle mask according to the wanted animation (IN/OUT).

Listing 4.9: *BaseScene.ts*

```
...
private transitionInit(): void {
    // Shape of the graphical transition
    const circle: Phaser.GameObjects.Graphics = this.add.graphics();

    // Shape of the screen
    const rectangle: Phaser.GameObjects.Rectangle = this.add.
        rectangle(0, 0, this.cameras.main.width, this.cameras.main.
        height, 0x000000);
```

```
        // Define circle as the mask
        const mask: Phaser.Display.Masks.GeometryMask = circle.
            createGeometryMask();

        circle.setPosition(this.cameras.main.width / 2, this.cameras.
            main.height / 2);
        circle.fillCircle(0, 0, 0.1);
        circle.setDepth(0);

        mask.setInvertAlpha(true);

        rectangle.setDepth(1);
        rectangle.setOrigin(0, 0);
        rectangle.setMask(mask);

        circle.fillCircle(0, 0, 0.1);

        this.transition = [circle, rectangle];
    }
```

4.7.2 PreloadAssets

In the method "preload()" the files used later on can be loaded into the respective managers/-cache. If files are loaded in a method outside of the preload method, the loader has to be started manually (in the code). Now every asset loaded this way has a unique identifier and can be used in future scene as long as it is not removed explicitly. The asset can as such be loaded into the managers/cache in every scene, so that only the actual used assets are loaded. The advantage of this is that you can save time and storage. But the disadvantage is that you have to be connected to the internet the whole time. Should the connection be severed for only a short time, objects might not be displayed correctly. With a "preload" 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 [REFERENCE TO EXPLANATION TOP] can be accessed by filename.["fieldname"] or filename.fieldname:

Listing 4.10: *Example json file access*

```
    ...
    for (let category of loadedJsonObjectFile['categories']) {
        console.log("URL: " + category['url']);
        console.log("Name: " + category['validElements']['name']);
        console.log("ValidElement urls: " + category['validElements']['
            urls']);
    }

    for (let image of loadedJsonObjectFile['images']) {
        console.log("Name: " + image.name);
        console.log("Cat1: " + image.cat1);
        ...
    }
```

```
    ...
```

The way the objects files are preloaded into the respective managers is shown below.

Listing 4.11: *preloadAsset.ts*

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

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

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

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

Image Game Objects

Instead of representing an image as a normal image in game, sprites, a special texture, is used. A sprite game object can display both static and animated images in your game. They have input events, additional functions, fields and physics bodies. The main difference between a sprite and an image game object is that you cannot animate images.

4.7.3 DropDownMenu

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

The following code in every other scene ensures this:

Listing 4.12: *Send current scene to back*

```
...
create(): void {
    this.game.scene.sendToBack(this.getKey());
    ...
}
...
```

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

Listing 4.13: *Lock aquiring and freeing*

```
...
if (!this.lock) {
    this.lock = true;
    ...
}
...
onComplete: () => this.lock = false;
...
```

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 4.14: *Fetching current active scene*

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

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

4.7.4 WelcomeScene

A blinking finger is added to the screen so that the user knows that he can advance by clicking on the screen.

4.7.5 LevelMenuSceneScene

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

Figure 4.6: *Easy Level Icon*

Figure 4.7: *Medium Level Icon*

Figure 4.8: *Hard Level Icon*

4.7.6 IntroScene

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

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

4.7.7 SortingScene

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

Listing 4.15: *returnQuad() (sortingScene.ts)*

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

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

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

    horizontal.push(leftOffsite);

    vertical.push(topOffsite);

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

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

    switch (quadrantType) {
        case 3: {
```

```
                ret = [Phaser.Math.RND.between(horizontal[quadrant] +
                    spriteSizeHalf, horizontal[quadrant + 1] -
                    spriteSizeHalf), Phaser.Math.RND.between(vertical[0]
                     + spriteSizeHalf + this.cameras.main.height / 8,
                    vertical[1] - spriteSizeHalf - this.cameras.main.
                    height / 8)];
                break;
            }
            case 4: {
                if (quadrant < 2) {
                    ret = [Phaser.Math.RND.between(horizontal[quadrant]
                        + spriteSizeHalf, horizontal[quadrant + 1] -
                        spriteSizeHalf), Phaser.Math.RND.between(
                        vertical[0] + spriteSizeHalf, vertical[1] -
                        spriteSizeHalf)];
                } else {
                    ret = [Phaser.Math.RND.between(horizontal[quadrant
                        \% 2] + spriteSizeHalf, horizontal[(quadrant \%
                        2) + 1] - spriteSizeHalf), Phaser.Math.RND.
                        between(vertical[1] + spriteSizeHalf, vertical
                        [2] - spriteSizeHalf)];

                }
                break;
            }
            case 6: {
                if (quadrant < 3) {
                    ret = [Phaser.Math.RND.between(horizontal[quadrant]
                        + spriteSizeHalf, horizontal[quadrant + 1] -
                        spriteSizeHalf), Phaser.Math.RND.between(
                        vertical[0] + spriteSizeHalf, vertical[1] -
                        spriteSizeHalf)];
                } else {
                    ret = [Phaser.Math.RND.between(horizontal[quadrant
                        \% 3] + spriteSizeHalf, horizontal[(quadrant \%
                        3) + 1] - spriteSizeHalf), Phaser.Math.RND.
                        between(vertical[1] + spriteSizeHalf, vertical
                        [2] - spriteSizeHalf)];
                }
                break;
            }
            default: {
                break;
            }
        }
        return ret;
    }
```

4.7.8 PropertySortingScene

To specify the difficulty level, two fields are needed:

Listing 4.16: *Level fields (propertySortingScene.ts)*

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

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

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

4.7.9 RestrictedSortingScene

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

Listing 4.17: *equalityCheck (restrictedSortingScene.ts)*

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

4.7.10 GameScene

In the game scene three marked objects are checked for equality if the checker was not already run for those three objects.

Listing 4.18: *update (gameScene.ts)*

```
update(time: number): void {
    ...
    if (!this.checked && this.arrayMarked.getLength() >= 3) {
        this.checked = true;
        ...
    }
    ...
    if (timedata <= 0) {
        this.checked = true;

        ...
    } else {
        timedata -= this.timedataStepsize;
        this.timefluid.setData('timeY', timedata);
        this.timefluid.setScale(this.timefluid.getData('timeX'),
            timedata);
    }
}
```

As 'gameMax' is calculated with a quotient there is a rounding error in the floating point arithmetic. To even this error we add or subtract epsilon, the maximum relative error of the rounding procedure.

Listing 4.19: *updateProgressbar (gameScene.ts)*

```
...
if (this.points >= this.gamefluid.getData('gameMax') - Phaser.Math.
    EPSILON) {
    ...
}
```

To mark the helpers menu icons the checkEquality method is modified with a boolean to mark the icons while executing the check.

Listing 4.20: *checkEquality (gameScene.ts)*

```
...
private checkEquality(sprite1: Phaser.GameObjects.GameObject,
    sprite2: Phaser.GameObjects.GameObject, sprite3: Phaser.
    GameObjects.GameObject, inGame: boolean): boolean {
    if (sprite1 instanceof Phaser.GameObjects.Sprite &&
        sprite2 instanceof Phaser.GameObjects.Sprite &&
        sprite3 instanceof Phaser.GameObjects.Sprite
    ) {
        // Return value
        let replaceObjects: boolean = true;

        for (let categoryIndicator of this.arrayCategory.getChildren
            ()) {

            // Make sure your objects are sprites
            if (categoryIndicator instanceof Phaser.GameObjects.
                Sprite) {

```

```
                    // Clear tint
                    categoryIndicator.clearTint();

                    if (
                        sprite1.getData(categoryIndicator.name) ===
                            sprite2.getData(categoryIndicator.name) &&
                        sprite2.getData(categoryIndicator.name) ===
                            sprite3.getData(categoryIndicator.name) &&
                        sprite1.getData(categoryIndicator.name) ===
                            sprite3.getData(categoryIndicator.name)
                    ) {
                        if (inGame) {
                            categoryIndicator.setTintFill(0x00dd00);
                        }
                    } else if (
                        !(sprite1.getData(categoryIndicator.name) ===
                            sprite2.getData(categoryIndicator.name)) &&
                        !(sprite2.getData(categoryIndicator.name) ===
                            sprite3.getData(categoryIndicator.name)) &&
                        !(sprite1.getData(categoryIndicator.name) ===
                            sprite3.getData(categoryIndicator.name))
                    ) {
                        if (inGame) {
                            categoryIndicator.setTintFill(0x00dd00);
                        }
                    } else {
                        if (replaceObjects) {
                            replaceObjects = false;
                        }
                        if (inGame) {
                            // Mark category as red
                            categoryIndicator.setTintFill(0xdd0000);
                        }
                    }
                }
            }
            return replaceObjects;
        }
    }
```

To add another small gamification element in the form of a mini reward to the game, every time the user finds a matching pair, some additional is added to the timer. So the user is even more motivated find pairs even faster.

Listing 4.21: *updateProgressbar (gameScene.ts)*

```
    ...
    let timedata: number = this.timefluid.getData('timeY');
    timedata += this.timedataStepsize * 5000;
    if (timedata > this.timefluid.getData('timeYMax')) {
        timedata = this.timefluid.getData('timeYMax');
    }
    ...
```

As frustrating as it is to find no matching pair, it is ensured that there is at least one possible

pair every time a new object is added.

Listing 4.22: *INSERT (gameScene.ts)*

```
...
private rebuildDisplayedObjects(): void {
    ...
    for (let card of this.arrayDisplayed.getChildren()) {
        if (card instanceof Phaser.GameObjects.Sprite) {
            card.setVisible(false);
            this.arrayStack.add(card);
        }
    }

    this.arrayDisplayed.clear(false, false);

    this.initObjects();
}
...
```

4.7.11 ScoreScene

The most important and special part in the score scene is the way the score is saved, as explained in this section.

4.8 Final Look of the Learning Environment

This section contains the final look of the learning environment.

Figure 4.9: *Loading Screen*

Figure 4.10: *Title Screen*

Figure 4.11: *Level Menu*

Figure 4.12: *Level Menu showing Stars*

Figure 4.13: *Sorting Scene Category Mixture*

Figure 4.14: *Sorting Scene Category 1*

Figure 4.15: *Sorting Scene Category 2*

Figure 4.16: *Sorting Scene Category 3*

Figure 4.17: *Sorting Scene Category 4*

Figure 4.18: *Introduction Screen to Sorting Scene*

Figure 4.19: *Property Sorting Category 1*

Figure 4.20: *Property Sorting Category 2*

Figure 4.21: *Property Sorting Category 3*

Figure 4.22: *Property Sorting Category 4*

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

Figure 4.24: *Restricted Sorting Easy*

Figure 4.25: *Restricted Sorting 2 Hard*

Figure 4.26: *Introduction to Object Pairing Easy*

Figure 4.27: *Object Pairing Easy*

Figure 4.28: *Object Pairing Easy Helper Bar*

Figure 4.29: *Object Pairing Hard*

Figure 4.30: *Object Pairing Hard Helper Bar*

Figure 4.31: *Score Screen*

5

Evaluation

UNDER CONSTRUCTION

In this section the environment the game was tested in will be presented and analysed. First every game has to be tested on different devices for code errors or functional disbehaviour which was not intended. Second and third a target and various audience will test the game and significant reaction and behaviour will be stated.

5.1 Debugging

5.1.1 Devices

firefox, chrome, microsoft edge computer (apple, windows 10), tablet (apple, android), smartphone (apple, android)

5.1.2 Differences and obstacles

5.2 Target Audience Test

5 Test subjects age between 4 and 7 with less to none knowledge of use of electronic devices. 6 Test subjects age between 4 and 7 with strong knowledge of use of electronic devices.

5.3 Various Audience Test

11 Test subjects age between 18 and 40 equally distributed with daily usage of electronics (computer, tablet, smartphone) 13 Test subjects age between 30 and 70 equally distributed with only daily usage of their smartphone

5.4 Checklist

Depending on your budget and available time, you may have performed simulations or even some experiments. In either case, it is important to describe the environment of your experiments or simulations. This includes stating the parameters and conditions of the environment (sim- ulated or real), what measurements were taken and how they were taken. You need to establish the fact that your simulation or experiment results are statistically stable, meaning that they are representative of the space of possible results. Performing experiments and simulations is a subtle mat- ter, always putting the validity of your data at risk in many aspects. Before you perform the simulation or ex- periment, educate yourself on how to perform simula- tions, how to interpret the results and how to present them in graphs and figures. Each figure (or graph) should be well explained. Dedi- cate at least one paragraph for each figure. Describe what the reader sees in each figure and what he should notice. Moreover, reason on the results - are they the way they were expected to be? Avoid giving tables of numerical data as means of presenting your results. Compare the performance of your solution to the per- formance of one or two competing solutions. Usually, when you simulate or experiment on your solution, you simulate it in contrast to a competing solution. You have to make sure that the test scenarios are fair and make an argument about the fairness of your comparison in the paper. A special case of experiment is the usability test. Many times, although the performance of a solution can be im- pressive, the applicability of it can be minimal. Usability test is a type of experimentation, which determines the acceptance of a solution by end-users or its suitability for certain applications. Usually, papers on software product solutions contain usability tests.

Conclusion

UNDER CONSTRUCTION

limitation - design - knowledge of the framework and examples for complete funcitonality "der gedanke dahinter" -

disadvantages - lehrer mues iharbeite will beschrenkt durch keine zahlen und woerter - durch eine webaplikation schwer zu beschraenken auf gewissen geraeten (apple)

6.1 Future Improvements

6.2 checklist

The conclusions section, similar to the introduction and related work sections, serves two purposes. The first is to elaborate on the impacts of using your approach. The second is to state limitations or disadvantages of your so- lution, thus enabling you to provide directions for future research in the field.

Bibliography

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

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

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

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

Title of work (in block letters):

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

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

With my signature I confirm that

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

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

Place, date	**Signature(s)**

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