MonstAR Documentation

Andreas Riedel, Daniel Peters, Michael Kronester

0.Initial Thoughts and Planning

0.1 Initial Idea

The goal of the app is to train monsters and have them fight each other. Each monster has a unique game card that can be tracked through image recognition. When the game card is tracked, the monster is displayed in a 3D model on the game card. It is also possible to upload the image of the game card on a smartwatch in order to access AR functionality and inspect/check on the monsters on the go. Once two cards are placed so that they face each other, the monsters can fight each other through actions on the screen. Each monster has several development possibilities. They can be trained/pushed by connecting them to a fitness app. Through your walking, standing and fitness actions, the monsters can be then

leveled up and potentially evolve. Furthermore, all collected monsters can be viewed in detail in a separate portal. When you enter this portal, you will see a list of all the monsters you have in real "human" size.

0.2Timetable

Time	Deliverable
01.11. - 05.11.	Personas, User Stories, Requirements
06.11. - 26.11.	Prototype: Image tracking with minimal functions Portal with minimal Content inside
01.11. - 05.11.	Proof of work
27.11. - 10.12. (postponed 21.12)	Usability Test on 30.11. & 01.12.
until 10.12. (postponed 2 weeks=	Iteration for improving Prototype
10.12.-23.12. (postponed until 6.1.22)	Final Development
24.12. (postponed until 6.1.22)	Code freeze! Working on Documentation
16.01.	Documentation Hand-In

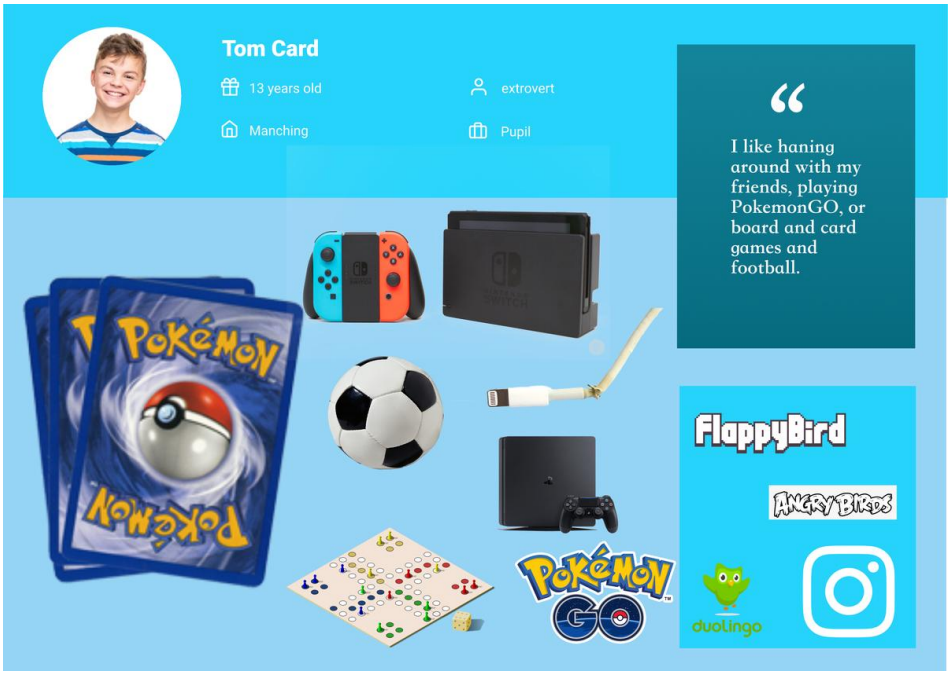
0.3 User Stories

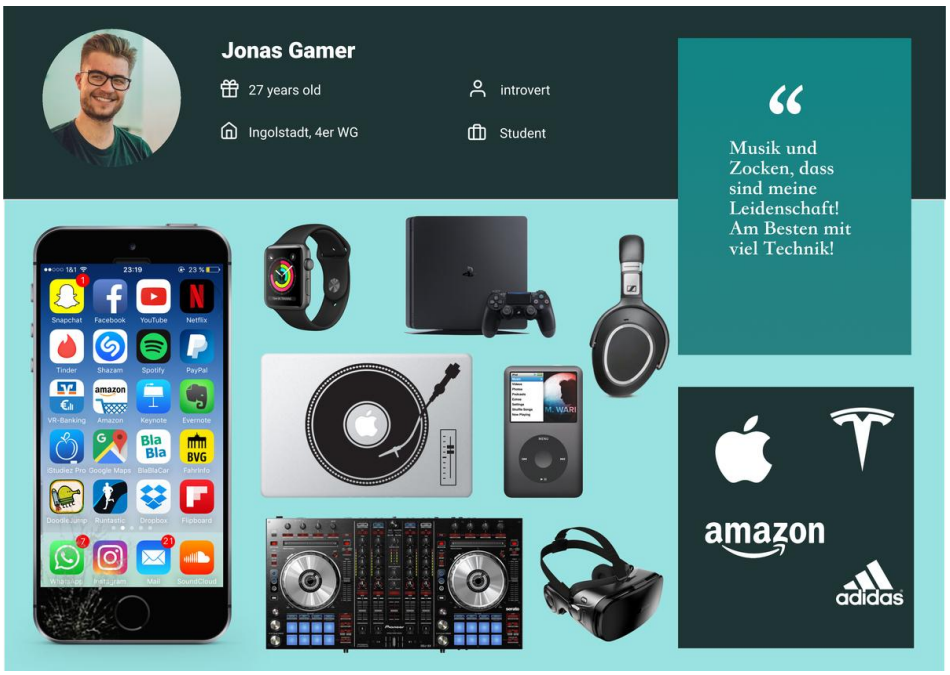
After the initial ideation phase and the rough concept planning, we sat down and defined 5 User-Stories to implement in the App:

1. US 001 - Choosing Monster
   As a young student I want to choose my monster via card to play with my friends at school.
   Requirements:
   - Projection on card
2. US 002 - Choosing Monster on Smartwatch
   As a student I want to impress my friends with my Monster Projection on my Smartwatch.
   Requirements:
   - Being able to choose your monster
   - Sending it to the Smartwatch
   - Projection on the Smartwatch
3. US 003 - Fighting
   As a User I want to win against my friends Monsters.
   Requirements:
   -Fighting Card/Card Watch/Watch Card/Watch
   - Animations
   - Able to win against each other
4. US 004 - Habitat
   As a student I want to procrastinate by looking at my habitat full of monsters.
   Requirements:
   - Projecting the habitat portal in your room
5. US 005 - Train
   As a student I want to train my monster to make it beat every other monster.
   Requirements:
   -Able to train my monster

0.4 Personas

To start our project development with a kind of User-centered process we developed 2 fictional personas, depicting the two main user groups.





1. About MonstAR

MonstAR´s base idea is to generate an augmented turn-based card play experience merging the analog and digital world. You can collect Monsters in reality with the analog cards and let them fight against the Monsters of your friend. You also can have a look at your monsters in real life size and train them by doing sports.

As there is no practical and commonly used AR Headset right now, MonstAR was developed as a proof of concept mobile app with interactions suitable to that medium. To have the best experience MonstAR should therefore be later used with an AR-Headset when the headsets are technologically mature and widely commercially available.

1.1 Concept

At first, our concept was based on having your monster on a peripheral device like a smartwatch or similar. But as we tried to track the cards on a smartwatch we had several problems with tracking and positioning.
So we decided to build the experience based on analog cards to overcome reflection problems of glass displays (smartwatch or smartphone) and therefore have better tracking. We didn't investigate further on the use of other mediums as simple hard paper cards seemed like the best compromise of easy use and low-cost prototyping.

As mentioned before, the concept of the App is based all around collecting cards, training your monsters, and fighting against your friends or KPIs.

The following paragraphs explain the Components of the App. Therefore we name the component and explain its function as well as the constraints and requirements we thought out.

1.2 Components

MonstAR consists of 3 main components - The inspector, the portal, and the battling. Every component mentioned was built functionally in the frame of the project.
Some features like the walking to train the monster or the actual connection to a database for accounts and monster management were just partially implemented.

1.2.1 Inspector Scene

The Inspector can be reached by just tracking one of our MonstAR cards. this submenu consists of 3 main functions:

1. Look at the Monster
With Vuforia it is possible to track the monster into reality and position it accordingly to the orientation of the card placed on a plane or similar.

2. Try out the Evolutions

As you can train your monster just with sports, we decided to give an opportunity to foreshadow the next evolutions in the inspector. Therefore you need to just click the “evolve” button. Every evolution will be displayed with a custom particle system developed by us.

3. Try out the Attacks
Of course, it should be possible to try out the attacks, that's why we also decided to implement the attack animations and particle systems. By clicking on the respective button the attack is triggered. Critical attacks will also be displayed with a particle system animation.

1.2.2 Portal Scene

The second main function is the portal. The portal basically is an AR-Portal showcasing all monsters in real life-size. It can be triggered by clicking on the “portal” button. Within this scene it is possible to track the floor with plane tracking, projecting the portal in front of you and foreshadowing the environment behind it. It is possible to walk through, which triggers the monsters to walk right in line in front of you. So you can easily have a look at the monsters from all sides and distances.

1.2.3 Battle Scene

The third and heart of the game's main function is the Battle mode. It can be triggered by clicking the “Battle” button. When entered you will be instructed to lay your card on a projected position in front of the opponent's card. After successful execution, the “fight” button shows up. By clicking it, you will enter the fight mode. Here you can fight against the opponent's monster by clicking the respective attack button. After the battle, you will receive points based on victory or loss. As mentioned in 1. The point function as well as the actual implementation of a multiplayer was faked and therefore not implemented for our prototype.

1.3 Gameplay

The Gameplay mainly consists of the sport leveling function and the battle function. In the following, we will briefly describe the concepts behind obtaining monsters, leveling up, and battling.

1.3.1 Obtaining Monsters

If starting playing MonstAR for the first time the user will receive the first stage of your monster - the egg. By doing sports like walking, jogging, or bicycling you will be able to unlock the 4 evolutions of your chosen monster. Every Monster, therefore, has one egg state at the beginning and 3 Evolutions.

1.3.2 Leveling Up

The level-up button will be unlocked after you reach the threshold of the respective next evolution. After the 3 evolutions, it will not be possible to evolve your monster to a next stage but gain monster levels.
From the egg to the 3rd evolution the threshold of sport points you´ll need to level up will increase. A screenshot of the particle System triggered by leveling up is shown in fig. 1.

fig.1: Level-Up Particle System

1.3.3 Battling

Battling is meant to make the game more interactive and help to connect people battling against each other. At the finished stage it should not only be possible to battle against humans but also against KI (wild) Opponents in real life. To make it even more tactical, monsters have attributes granting them advantages or disadvantages in fights, so it is vital to choose the right monster against the chosen opponent's monster.

1.4 Monsters

Our application is all about monsters. Their design is extremely important to create a bond for the user and thus to strengthen and maintain the motivation to train and level them up. This could already be seen in the app Pokemon Go from Niantic, which triggered a lot of hype in the past few years. Since the designs of several monsters turned out to be extremely time-consuming, since they had to be modeled, animated, and covered with textures and furthermore did not lead to a satisfying result, we decided to use already existing monster assets for our application. We found an asset pack from Meshtint Studio in the Unity Asset Store suitable for our concept, which we bought during a Black Friday deal. This asset pack contains some monster evolution series which contain three evolutions of a line each of which is already rigged and feature some standard animations.

1.4.1 Evolutions

As a core mechanic of the application is to level up and battle with monsters, we decided to include multiple evolution stages of one monster in order to provide variety as well as long-term motivation. When a monster evolves it gets a huge boost in its stats compared to a level up. Whenever an evolution takes place a particle system will be activated to highlight and smoothen the transition between the two monster models.

Eggs
Eggs are the first stage a player obtains from an evolution line. Because we want to increase the excitement and curiosity we do not show the name of the monster yet. Thus, a surprise effect should arise during the hatching of the egg. Every egg of an evolution line has a unique texture that gives a hint about the watchable monster and its type.

Stage 1 Evolutions
Stage 1 evolutions are the monsters form after they are hatched. These monsters can be referred to as babies and are rather weak. When the monster reaches level 16 it can be evolved into its next stage.

Stage 2 Evolutions
Stage 2 evolutions can be referred to as the juveniles of an evolution line, making them stronger than Stage 1 evolutions, but have not yet unleashed their full potential. When a monster reaches level 36, it can be evolved into the next and final stage.

Stage 3 Evolutions
Stage 3 evolutions are the strongest evolutions of an evolution line. They can be referred to as adults and have already a high potential which can be increased by leveling them up further.

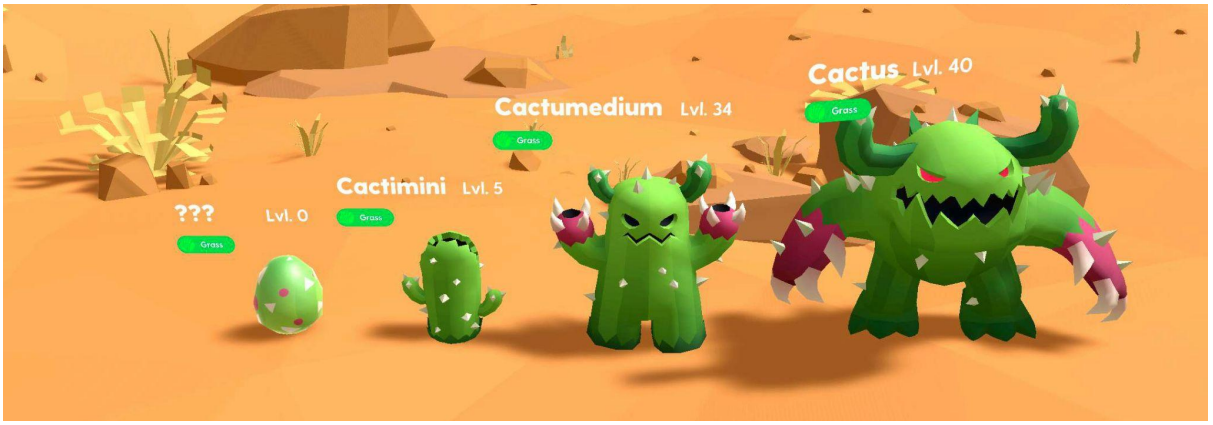
1.4.2 Monster Types

Inspired from the elements as well as the types of the Pokemon games we defined several types a monster - but also an attack - can have, such as Water, Grass, Fire, Bug, Poison, Fight, Ground, Psychic, Ghost, or Dark. As seen in the figure below, for each monster type, we designed unique icons as well as colors which helps to differentiate them better and quicker and strengthen associations to the monsters. Monster types operate on the scissors and paper principle, in which some types are more effective against a certain type but less effective against another. To give an example, water is effective against fire, fire effective against grass, and grass effective against water.

1.4.3 Overview of Monsters

This section gives a small overview of the used evolution lines and monsters used in our application as well as some of their characteristics.

Cactus Evolution Line



The Cactus Evolution Line belongs to the grass type, attacking physically with sharp claws, bites, and shooting thorns. Monsters of this evolution line are rather bulky, being able to take a lot of damage while at the same time having strong attacks.

Shadow Evolution Line

The Shadow Evolution Line belongs to the fire type, attacking physically with sharp claws but also using flames to damage the opponent from the distance. These monsters are solid all-rounders, functional in all scenarios.

Phantom Evolution Line

The Phantom Evolution Line belongs to the ghost type, attacking from the distance by frightening the opponent. Monsters of this evolution line have very strong attacks, however to the expense of their defense.

Draculo Evolution Line

The Draculo Evolution Line belongs to the dark type, using quick and sneaky attacks to confuse the opponent. Monsters of the Draculo evolution line are quick and deal a lot of damage.

1.4.4 Animations

As mentioned before, all monster prefabs of the asset pack came with default animations. Using the Animator of each prefab these animations can be triggered either by script or by UI events. We used the following animations for bringing more life into the application, making the monsters more dynamic and alive, and making the attacks more transparent through feedback in the form of animations.

Idle
The Idle animations only display a minimal movement of the monster itself - mostly the monster is standing or flying. This animation loops until a new animation has been triggered.

Walking
As the name suggests, walking animations let monsters walk in place. This animation loops until a new animation has been triggered. Combining the walking animation with a path, a monster can walk along this path.

Attacks
Each monster has several attack animations which differ in style. Mostly the animations are in the form of a physical hit through its extremities, however, some are displaying projectile attacks for far-ranging attacks. In the latter, however, only the monster is animated and no projectile is given. After the animation has been played the Animator will switch to the idle animation automatically.

Taking Damage
This animation is triggered once the monster takes damage. After the animation has been played, the Animator will switch to the idle animation automatically.

Faint
The faint animation represents the death of a monster. We use the faint animation to indicate if a monster has been defeated.

1.4.5 Attacks

All of our monsters have two attacks - one weaker and a stronger attack. Each of the monster's attacks look different in style due to the monster's attack animation. Thereby every attack has a type - such as the monster types - and a damage value which it will deal towards other monsters.

Particle Systems & Animations
As already described in the section above, the monsters already have attack animations. However, we gave every evolution line a signature attack for which a particle system and an additional animation have been designed. These animations have been synced with the attack animations of the monsters so that they appear as one animation seamlessly.

Thorns

Monsters of the Cactus line shoot thorns as a grass attack. The higher the evolution, the more thorns are shot and thus the more damage dealt. We animated the thorns shooting out of the monster's mouth or extremities along the z-axis towards an opponent.

Flame

Monsters of the Shadow line shoot flames as a fire attack. The higher the evolution, the stronger and bigger the effect. The flame effect is built from three different particle systems, which is the flame itself, an additive fire at the beginning, and glowing particles. We set the duration of the particle systems to one second which resulted in a nice effect of shooting a flame forwards.

Ghost Orb

Monsters of the Phantom line shoot a ghost orb. The Ghost Orb is built from three particle systems: the base of the Orb, its fill, and sparkling particles surrounding it. The higher the evolution, the stronger and bigger the orb. In addition, we animated the orb along the z-axis of the monster to be shot out of the monster's hands during the attack animation.

Tornado

The Draculo evolution line is able to create a tornado and cover themselves inside it while attacking. The tornado is designed from two-particle systems which have the shape of a cone turned upside down. In order to create more diversity in colors and gradients as well as add more turbulence, we overlapped these two particle systems.

2. Development

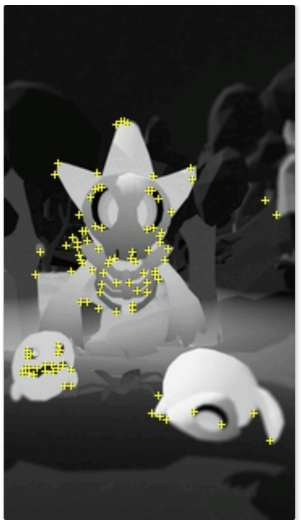
2.1 Plattform

For the development of our application, we have chosen the Gameengine Unity (version 2020.3.24f1). Since all of our team members have an iPhone, we implemented the application for iOS using Apple's ARKit plugin for Unity. However a build for Android using the ARCore plugin should be possible as well - however, it wasn't tested. The first steps towards implementing the application were initially made with Unity's ARFoundation plugin. However, initial image tracking tests with this plugin resulted in some problems. Multiple image targets could not be tracked simultaneously, which would make it impossible to visualize a battle between two cards. Although this was worked around by a script in which targets could be tracked alternating per frame, this resulted in poor performance. Another problem that arises was that only one GameObject or 3D model of a monster can be linked per image target - due to the evolution mechanics of the monsters, in which several monster models of an evolution series "share" one card, this was indispensable. Those problems led us to switch to Vuforia's AR Plugin for Unity, which was able to solve all the problems described above. This way, multiple images could be tracked simultaneously, and multiple 3D models could be assigned to one image target. For image tracking, a library of targets based on the monster cards was used.

2.2 Image Tracking

Since our game has multiple cards and associated monsters, we created an image library in Vuforia's Developer Portal and connected it to our Unity project. This has enabled us to extend, delete, replace and adjust images and test them immediately within the game seamlessly.
Since some of the initial maps did not have enough features for good tracking, they were adapted and extended with more features within several iterations to enable smooth tracking. Furthermore, the width of an image has been fixed to 137mm. Using the fixed size of images all monster sizes can be scaled correctly to the physical card sizes even if they have been moved.
As already described in the section above, each image target had to be assigned to multiple monsters and an egg of one evolution line. We achieved that by making the monsters and egg prefabs to children of the corresponding image. However at this stage, whenever an image has been

tracked, all monster prefabs of that target have been displayed at the same time. To circumvent this problem, we included a „TrackingHandler“ as well as an „EvolutionHandler“ which communicate and act together in order to display only one and the right monster. A detailed functionality of these two components is described in Section 2.6 Inspector Scene.



2.3 Plane Tracking

Plane tracking scans the area for flat surfaces and indicates through a crosshair at which position an object can be placed. By tabbing on the screen the object will be placed on the position of the crosshair. We have decided on the use of plane tracking in order to give the user the option to define the spawn position of those game components by themselves. Thereby, we make use of it in two instances. First is the placement of the portal in the Portal Scene. Second, since it is not clear for the application which card/monster belongs to the player and which to the opponent, a game field can be placed using plane tracking which contains augmented placeholders for the player's card and the opponent's card. The exact function of the placeholders can be read in section 2.8.1 Player Trigger & Enemy Trigger . Natively, Vuforia enables the option to place - and thus duplicate - the children of the PlaneStage as often as the user specifies a new spawn position. However, since in both cases we needed only one instance of the placed object, we disabled that option. What nevertheless turned out to be a problem was that when a user touched the screen

another time after an already placed object, this object was moved to the new location defined by the crosshairs. This also conflicted with the UI elements of the overlay UI whereby touching the elements by the user simultaneously moved the object. To solve this problem, we created a script that sets the plane tracker inactive as soon as an object is placed.

2.4 Game Cards

The game card monster models were put into custom-created environments. The design of all cards is minimal as the level and UI, as well as the stats, are displayed in the game itself. On every card, all 3 Evolutions are depicted. As we were facing tracking issues at the start of the project we also designed the cards in a way they have good to excellent tracking features. Coming to this, all our cards have at least 4 of 5 Stars in the Vuforia tracking evaluation. More about the features and the tracking itself can be found in 2.2 Image tracking.




2.5 User Interface

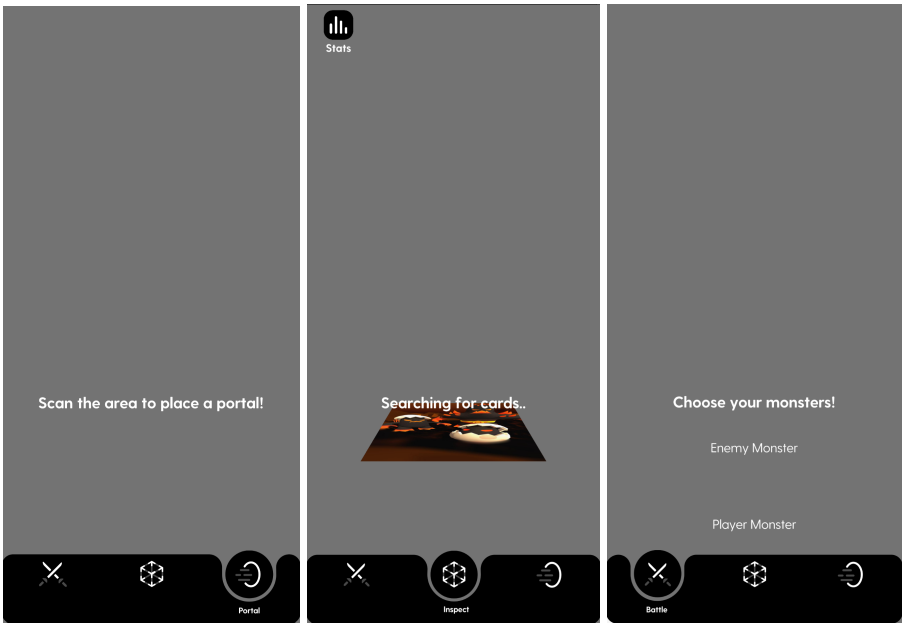
In this scene, the UI is rather minimal. However, there are some UI elements since each monster has unique information such as name, stats, and attacks that have to be shown. Some of the elements are rendered by the camera in an overlay UI, some are augmented next to the monsters. The decision of having two types of UIs leads back to user testing. In the first stages of the application, all of the GUI was augmented next to the monsters and hit detection of interactive elements was performed by raycasting (a touched pixel shoots ray along the z-axis and checks whether

an UI element has been hit). However, as in the user testing, it became apparent that those targets are more difficult to hit by the user since monsters far away have smaller UI elements, so we decided to provide two types of UIs.

2.5.1 Overlay UI

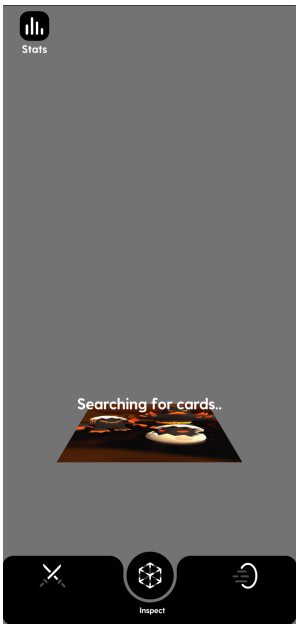
These UI elements are rendered by the camera in a canvas which is rendered to screen-space overlay in the inspector. Thus it has the resolution of the camera (smartphone screen). The idea behind the overlay UI is to display all interactable UI elements, thus making the interaction easier.

Navbar
Within the navbar, there are three buttons each for switching between the Inspector Scene, Battle Scene, and Portal Scene. Clicking on the button will switch the current scene with the respective scene relative to the button. This is achieved by a script „SwitchScenes.cs“ which uses the SceneManager of Unity to load the scene.



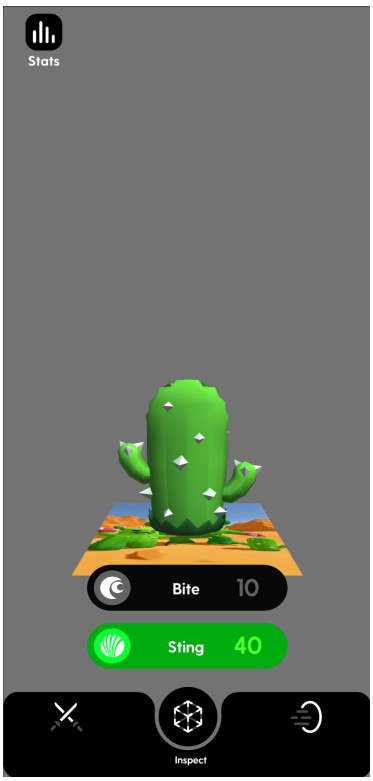
Stats Button
Every monster has an individual name, level, attacks, and progress bar which is used for leveling up, this information has to be shown. Since first user feedback from early testing revealed that the screen is rather overloaded, we decided to provide an option to the user to show this information on their demand - which is the stats button. Clicking on this button will reveal more detailed information about the monster's progress

for leveling up as well as its attack buttons which will trigger the respective attack.



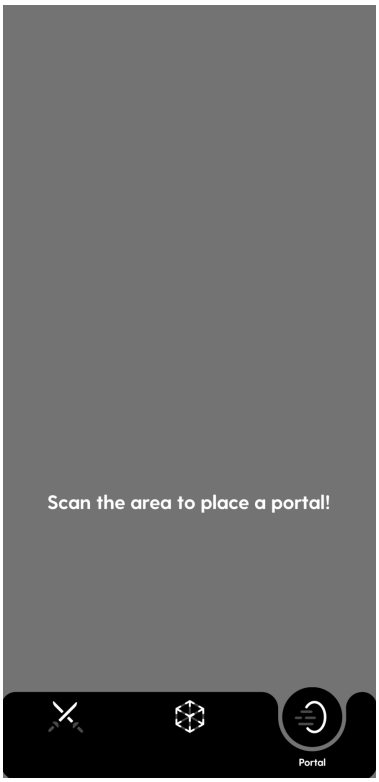
Attack Buttons

Each monster has two attacks - one weaker and a stronger attack. Clicking on an attack button, a unique attacking animation of the rigged monster model will be played. For some attacks, in addition to the model animation, a particle system is shown and animated. We used the OnClick( )-method of each button to trigger the animations using each model and each particle systems animator.



Labels

To make the tracking and game state of the application more transparent to the user, we decided to include some tips and status feedback incorporated into a label. The text of the label will be updated based on the state of the game.



2.5.2 Augmented UI

These UI elements are rendered by the camera in a canvas which is set to world space in the inspector. The canvases are each children of respective GameObjects making them inherit the transform components of location and rotation of their parents. This results in an effect that these UI elements are augmented next to - or related to - the GameObjects they belong to. The idea behind the augmented UI is to display all information - mostly about a monster - that doesn't need to be interactive but gives information about a monster. They show a visual dependency as well as strengthen the augmented approach of the app itself. Since Vuforia specifies the coordinate system for the application, which is at a very small scale of 0.01, some adjustments in terms of scaling had to be made regarding the applications GameObjects but also to the augmented UI. Due to the small scale, fonts of labels appeared extremely pixelated next to the monsters, making them unreadable. This problem could be fixed by increasing the font size of the labels by a multiple (> 200) and scaling the labels down drastically at the same time (< 0.000001). These values are only a rough estimation if a sharper resolution is desired the threshold of those two values can be increased.

Since it may occur in the combat scene that monsters and their associated canvases are viewed from a different perspective - such as from the side or from behind - study participants in our usability test noticed negatively that in these cases the augmented canvases are displayed from a very sharp angle or even mirrored from behind, resulting in poor readability. This led us to include a functionality of the canvas which always faces the camera independent of the angle of view towards a monster. We achieved that with a script called „LookAtCamera“, which updates and adapts the rotation of the augmented UI canvas each frame.

Monster Information Canvas

In this canvas, all information about a monster is shown including the name and level as text fields as well as its type in the form of an image.

Progress bar

The progress bar indicates how much training has to be done by the user to level up a monster. It must be noted that this progress bar is only an image and non-functional since we haven't implemented the interface towards a health app. After user testing, we decided to place the process bar horizontally next to the monster card instead of vertically under the monster information to enhance the visibility of the monster itself. The progress bar is only included in the inspector scene.

Health bar
For displaying the monster's health points correctly and making it visually transparent, we implemented a health bar for each monster. Thereby it changes appearance based on the monster's current health points. We implemented the bar using several components, however, the most important ones are a background image for the background of the bar and an additional image as a child of the background image which references the health points. Thereby the child image coordinate origin is the far left side of the background image. This has the effect that if the child image is scaled down it will decrease its width from right to left. In addition the color of the child image changes depending on the current health points of the monster from green (health >= 50%) to yellow (health < 50% & health > 10%) and red (health <= 10%). The health bar is only included in the Battle Scene.

2.6 Inspector Scene

This section provides an overview of the scene's most important functionality and components.

2.6.1 Tracking Handler

The tracking handler inherits a script „TrackingSwitcher.cs“ which has a variable called „monsterID“ as well as methods to change the ID depending on the tracked image. Whenever an image is being tracked a method of this script is called to set the monsterID to the representative monster (e.g. the when the image of the cactus line has been tracked the method „ChangeMonsterIdToCactus( )“ is being called which sets the monster ID to 1 as 1 is the ID for the cactus evolution line). This information is very important for handling evolutions, changing the GUI, and the Battle System.

2.6.2 Evolution Handler

The evolution handler inherits a script „EvolutionSwitcher.cs“ which has control over all displayable monster models as well as the GUI. By default, the eggs of the monster evolution lines will be displayed when tracking the representative card the first time. For each evolution line, the script contains a private variable of an integer between 0 and 3. Thereby 0 translates to the egg, 1 translates to the first evolution, 2 translates to the second evolution and 3 translates to the third evolution (e.g. cactiEvolution = 0 translates to the egg of the cactus evolution line being shown). The user is able to see the monster stats as well as its attacks by clicking on the „Stats“-button, which triggers the method showStats( ). Normally when the user has trained enough to hatch, level up, or evolve a monster, within the Stats canvas a button will appear to hatch, level up, or evolve the monster (since we don’t have the backend functionality of accessing Health-App stats, we trigger the event by a UI button named „Evolve“ to accesses the method evolveMonster( ) which hatches or evolves the monster directly).

public void evolveMonster( )
This method hatches or evolves a monster. This method first checks which card and therefore evolution line is currently being tracked by accessing the public variable „monsterID '' from the Tracking Handler. Afterward, it checks what evolution-ID the evolution line has. Depending on this ID the current monster model will be disabled (becomes invisible) and the model of its next evolution line enabled (becomes visible). At the same time, the augmented GUI of the monster will be updated to its name, level, and type, and the evolution ID is increased by one. To smoothen the switch between the evolutions out, an animated particle system that represents an evolution effect will be triggered. When the evolution ID of an evolution line is 3 it will do nothing.

public void showStats( )
This method shows the GUI stats of a monster or egg as well as the buttons for its attacks. It first checks which card and therefore evolution line is currently being tracked by accessing the public variable „monsterID '' from the Tracking Handler. Afterward, it checks what evolution ID the evolution line has. Depending on this ID the stats canvas of the respective monster will be augmented next to it on the ground and the buttons of its attacks will be displayed on the overlay canvas.

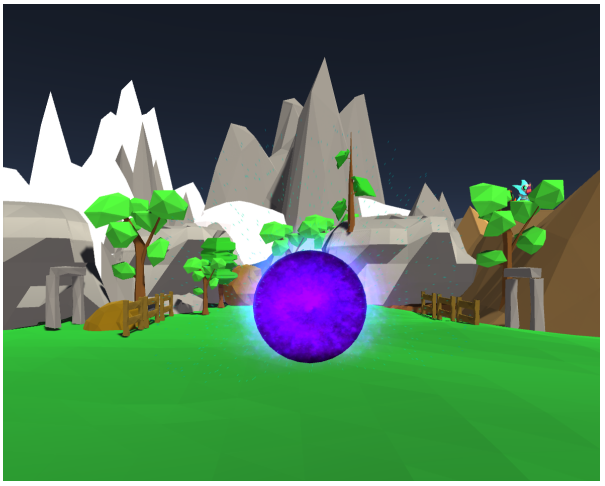
2.7 Portal Scene

The development of the Portal can be divided into three parts. In the following, we will go through the main issues of implementing and coding this portal starting with the environment going further on with the stencil shader script, particle system, and animation trigger.

2.7.1 Environment

The Environment is based on a free Asset in the unity store. (https://assetstore.unity.com/packages/3d/environments/low-poly-pack-94605)

All prefabs in the portal environment are based on this pack and were modified to create a unique portal environment. The following figures show the Portal scene.



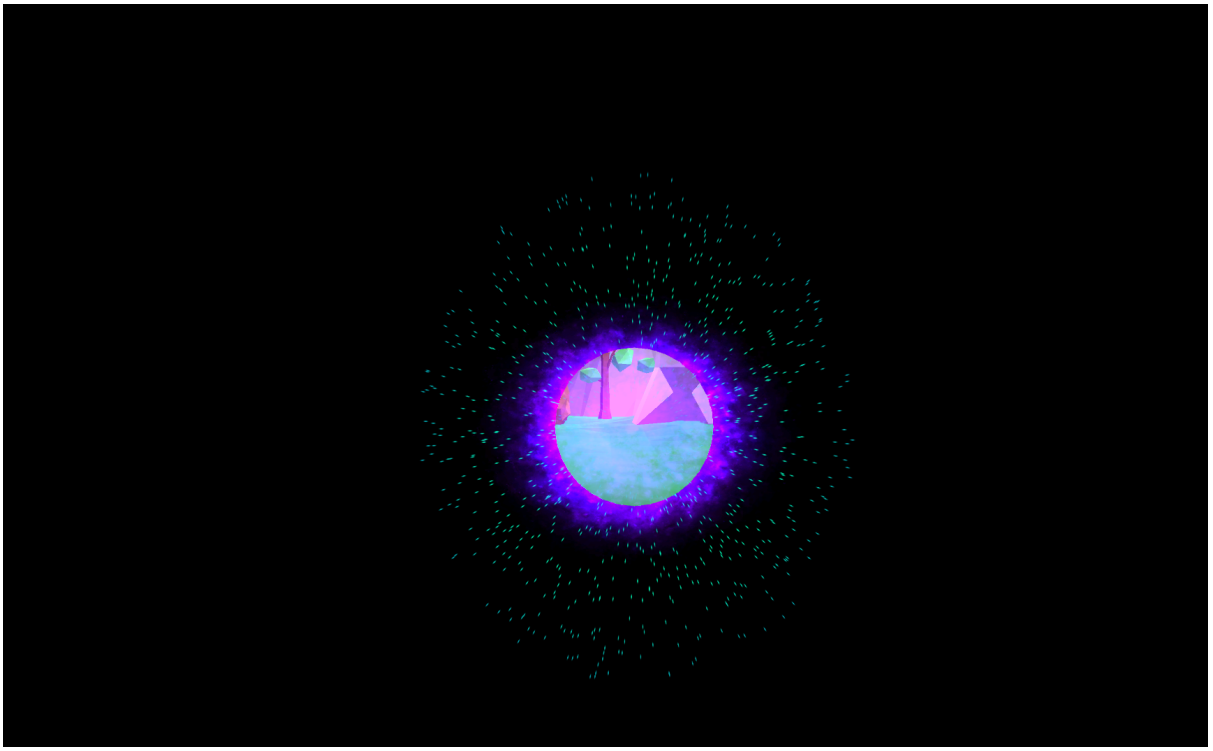


Fig. 2: Renderings of the actual MonstAR Portal & Portal Environment

2.7.2 Stencil Shader

As described, we wanted to see what's hiding behind the portal. It was vital to find a solution to how to render the environment just behind the portal window. The main idea now was to write a script attached to the portal which is just displaying the materials visible behind this portal window. The approach now was to modify the shaders for every material & the skybox so it is making a stencil test.

First step was to create a simple shader for the portal with just the stencil property “Assets/Shader/PortalWindow.shader”. Therefore we follow accordingly to the documentation (link: https://docs.unity3d.com/Manual/SL-Stencil.html) and modify the code like:

```
Shader "Custom/PortalWindow"
{

  SubShader  {

  ZWrite off
  ColorMask 0

  Stencil{

    Ref 1
    Pass replace
  }
```

```
        Pass
        {

        }
    }
}
```

The next step was to find the legacy shaders within unity to have the same shaders we need to modify. (downloaded from: https://unity3d.com/de/get-unity/download/archive)
The “Standard.shader” and the “Skybox procedural.shader” were also modified with the stencil test in the subshader region and a stencil test enum to debug and the shader´s function.

Now the last step was to create a script that triggers the disabling or enabling of the stencil test based on your position. The following “Assets/interdimensional.cs” script is creating an array of materials (the materials of which the environment consists of) to switch the stencil for all respective shaders linked to the material on or off. At the start, we set the stencil test on Equal (if we want to change the number in the shader it's more consistent like this) for each material in the chosen public material array. The rest of the script is just if-casing whether the “AR camera” is currently outside or inside the portal with colliders within the portal and the “AR Camera”.
The following code was written into “interdimensional.cs”:

```
...
public class interdimensional : MonoBehaviour
{
    public Material[] materials;
    // Start is called before the first frame update
    void Start()
    {
        foreach (var mat in materials)
        {
            mat.SetInt("_StencilTest", (int)CompareFunction.Equal);
        }
    }

    void OnTriggerStay (Collider other){
        if(other.name != "ARCamera")
        return;

        // Außerhalb des Portals
        if(transform.position.z > other.transform.position.z)
```

```
    {
        Debug.Log("Outside");
        foreach (var mat in materials)
        {
            mat.SetInt("_StencilTest", (int)CompareFunction.Equal);
        }

        //Innerhalb des Portals

    } else {
        Debug.Log("Inside");
        foreach (var mat in materials)
        {
            mat.SetInt("_StencilTest", (int)CompareFunction.NotEqual);
        }
    }
}

void OnDestroy()
{
    foreach (var mat in materials)
        {
            mat.SetInt("_StencilTest", (int)CompareFunction.NotEqual);
        }

}
....
```

After coding it was just necessary to add respective triggers and colliders to portal and AR Camera and of course add all the materials of the environment. With this approach we can basically use every render pipeline (ref fig. 1)and of course have less trouble to build it.

Render pipeline compatibility

Feature name	Built-in Render Pipeline	Universal Render Pipeline (URP)	High Definition Render Pipeline (HDRP)	Custom SRP
Stencil	Yes	Yes	Yes	Yes

fig. 1 ...: Stencil Filter - Render pipeline compatibility (https://docs.unity3d.com/Manual/SL-Stencil.html)

2.7.3 Portal Particle System

The particle system of the portal consists of three custom-built Particle Systems. The first one “OuterRing” represents the light blue outer Circle. It is just a hazy-looking ring with some moving particles (see fig. 3). The second Outer circle “OuterDiffuse” particle system consists of hazy deep blue to purple particles moving around the portal shape creating a slight illusion of diffusion of the portal (see fig. 3). The last one “ParticlesBlackhole” consists of little particles moving in the direction of

the portal creating an illusion of a blackhole sucking-in effect(see fig. 3).

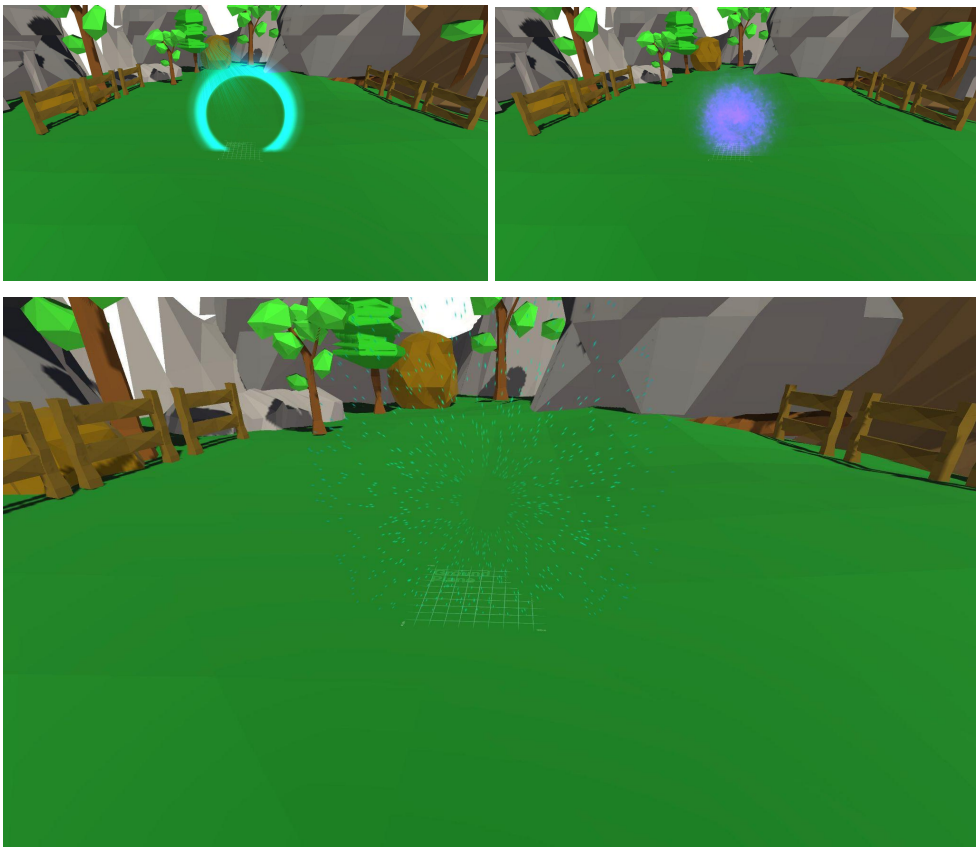


Fig. 3: Particle Systems in order of announcing

2.7.4 Animation Trigger

As the “AR Camera” moves into the portal the animation and pathing of the monster should be triggered. The script “TriggerMonster.cs” was coded to implement this behavior. It is just a simple trigger script triggering the walking and pathing animation coded in “Follow Path.cs”. For the path, a free-to-download “path asset” was used. It is creating lines between several Game objects and giving animation presets for moving and steering over several path points.
The Monster animations are offset within the “Trigger Monster.cs” script with a Coroutine and IEnumerator “selectMonster()” Method so the monster animations start with a slight offset of 2 sec.

2.8 Battle Scene

2.8.1 Player Trigger & Enemy Trigger (Script)

Since it was problematic to find out which of the two tracked cards belongs to the player and which to the opponent, we decided to use a solution that augments two placeholders for the cards with the help of plane tracking. These placeholders have a rigid body and box collider component and - depending on the placeholder - a script called "PlayerTrigger.cs" with a public variable playerMonsterID or „EnemyTrigger.cs" with a public variable enemyMonsterID. As soon as the collider of a monster collides with one of the two placeholders' colliders, the method "OnCollisionEnter(Collision col)" is executed, whereby the respective card is assigned either to the player or the opponent.

OnCollisionEnter(Collision col)
This method sets either the playerMonsterID or the enemyMonsterID to the assigned monster card. This is achieved by calling the parent object (monster) of the collided collider and then matching it to the monster ID (e.g. when the collider of the cactus monster collides with the collider of the player placeholder collider, the playerMonsterID is set to 1 as this is the ID for the cactus evolution line). At the same time, the Battle UI is updated to indicate that the respective monster has been registered as the player's monster.

OnCollisionExit(Collision col)
This method is called when a monster collider exits a placeholder collider, setting the playerMonsterID or enemyMonsterID to null. This indicates that no monster has been assigned and used to switch monsters for a rematch.

2.8.2 Unit (Script)

The "Unit.cs" script is attached to every monster and functions as a class that every monster inherits. This script defines and updates the name, level, maximum health points, and current health points for each monster. In addition, it serves as an interface for the HUD which reads the data of the monster from this script.

public bool TakeDamage (int damage)
This method updates the current HP based on the damage taken by an enemy attack. This subtracts the damage of the attack from the current health points. When the health drops to zero, the animator of the monster plays the "Faint" animation, if not, it plays the "Take damage" animation.

2.8.3 Battle HUD (Script)

The BattleHUD.cs script is attached to every Augmented UI Canvas of a monster. It allows initializing the maximum HP of a monster in the UI and updates the UI based on the current HP by changing the HP value in a text field and updating the health bars.

public void SetupHUD(Unit unit)
This method initializes the Augmented Battle UI of each monster at the beginning of a battle by accessing the "Unit.cs" component of the monster.

public void SetHP(int newHP)
This method updates the UI based on the new HP of the monster. First, it parses the current HPs of a monster as a percentage of its maximum HP, which defines the scaling of the health bar image. Then, depending on the percentage value, the color of the image is changed to green, yellow, or red.

2.8.4 Battle System

The battle system is designed as a Turn-based battle System as known from the Pokemon games. For this purpose, five battle states have been defined:

2.8.4.1 IDLE

The IDLE state is executed at the beginning of the match and serves to initialize the fight in which the monsters are assigned to the player and the opponent. During this State void CheckTracking( ) is executed per frame.

void CheckTracking( )

This method accesses the Player Trigger and Enemy trigger to check if a playerMonsterID and an enemyMonsterID have been registered (playerMonsterID != 0 && enemyMonsterID != 0). If that is the case a UI button to start the battle becomes visible. This way the players have the option to start the battle by themself. If one of the monsterIDs equals null, this method loops.

2.8.4.2 START

The START state is executed at the beginning of the fight and defines the first move of the fight. This may be the player's turn or the opponent's. The Battle System switches to the START state when the UI button to start the battle has been pressed. This also calls the method void SetupBattle( ).

public void SetupBattle( )

This method accesses the Unit of the player’s and the enemy’s monster depending on the playerMonsterID and enemyMonsterID and assigns it to a playerUnit and enemyUnit class. In addition, the playerHUD and enemyHUD are updated based on the playerUnit and enemyUnit through the SetupHUD( )-method, and the attack buttons of the player’s monster are set active. After that a coroutine for the method startBattle( ) starts.

IEnumerator StartBattle( )

This method updates a label of the overlay UI to indicate that the battle is starting. After waiting for one second the battle state switches to PLAYERTURN on ENEMYTURN.

2.8.4.3 PLAYERTURN

The PLAYERTURN state is executed on the player's turn. This enables the monster's attack button, allowing the player to choose and use one of the monster's attacks.

public void OnAttackButton(int damage)

This method is called when one of the two attack buttons of the monster has been pressed by the player. It stores the damage of the attack and starts a coroutine for the method PlayerAttack(int damage) and passes the damage value. When the button is pressed, the attack animation of the monster and the effects of the respective attack are also executed.

IEnumerator PlayerAttack(int damage)

This method uses the passed damage value and calculates/updates the enemy Unit's damage by calling the TakeDamage( ) method of the enemyUnit class after waiting for half a second and updating the dialog label to the enemy. At the same time, the enemyHUD will be updated based on the new values of the enemyUnit class. If the enemy monster is defeated by the attack, the Battle state is set to WON and the method EndBattle( ) is called. If it survives, the Battle state is set to ENEMYTURN and a coroutine for the method EnemyTurn( ) is started.

2.8.4.4 ENEMYTURN

The ENEMYTURN state is set on the opponent's turn. This disables the attack button of the player monster, which does not allow the player to take any action.

IEnumerator EnemyTurn( )

This method simulates the attack of the opponent. After waiting two seconds and updating the dialog label, a randomized number between one and two is generated, which decides which of the two attacks the opponent monster uses. Depending on which monster is the opponent and which attack was chosen, the corresponding attack animation is played and the damage value of the attack is used. The playerUnit's damage is calculated/updated with the damage of the attack by calling the TakeDamage( ) method of the playerUnit class after waiting for half a second and updating the dialog label to the enemy. At the same time, the playerHUD will be updated based on the new values of the playerUnit class. If the player monster is defeated by the attack, the Battle state is set to LOST and the method EndBattle( ) is called. If it survives, the Battle state is set to PLAYERTURN and a coroutine for the method PlayerTurn( ) is started.

2.8.4.5. WON & LOST

The Won state is set if the opponent's monster has been defeated by the player.

void EndBattle( )

This method ends the battle. Based on the battle state of LOST or WON the dialog text will be changed and all attack buttons of the player's monster

disabled. In addition, a Button for calling a rematch will be set active which executes the method Rematch( ).

public void Rematch( )

This method resets all values of the playerUnit and enemyUnit to their initialized values as well as setting the playerMonsterID and enemyMonsterID to null, which allows the players to switch monsters for a new battle or use the same again.

3. Usability Test

On the 21st of December 2021, we finished the three scenes for the respective main components. At this point in our project, we decided to test our prototype with two colleagues of our UXD-Master course.
We set up a Usability test with a brief questionnaire and a System Usability Scale (Affairs, 2013).

3.1 Study Design

To make it more feasible we created a google form with 3 scenarios testing each function of our prototype and a System Usability Scale after all scenarios.
We divided the scenarios into main tasks and asked the same 5 questions after each scenario:

1. Does it work as expected? If not tell us what was bothering you
2. What would you change?
3. What was the coolest feature?
4. How interactable felt that scene? (Likert scale 0-7)

3.2 Results & Findings

As we just tested two people we cannot say much about the actual data we conducted, but with this testing, we had at least a vague idea of what to change.

3.2.1 SUS

Our Evaluation of the System Usability Scale scored 63,75, so according to measuringu.com it is scoring in the Ok segment (ref fig. 4).

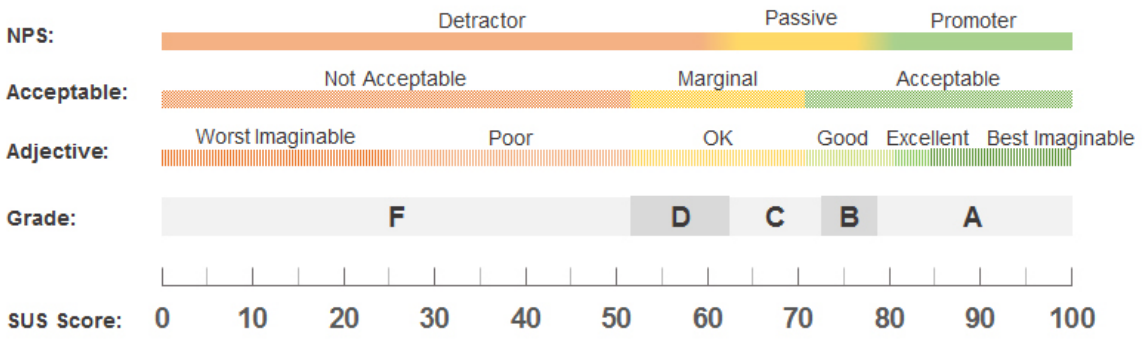


fig. 4: SUS Evaluation Score Table (Jeff Sauro, *5 ways to interpret a sus score*)

That means our app was OK but there is a need for redesign to score better in the “good” or “excellent” segment.

3.2.2 Observations and Insights

Most of the participants were able to finish all tasks, except the portal scene. All of the participants were mentioning the “cool” animations and liked the design style we used. The following paragraphs go more specifically into the Quotes and Observations of each Scenario.

Scenario 1: Inspector Scene
The most critical issue every Participant was mentioning was the missing labels and instructions for this Scene. In addition to that, the Participants were mentioning that the stats for each monster were distracting them. The stats at this time were displayed next to the Monster and were bouncing and wiggling around. They also found a bug by switching the monsters. Monsters did not disappear after the app lost track, so they could not switch it. To increase the usability we chose to add labels and instruction, as well as change the positioning of stats and fixed the bug.

The scene also felt quite interactable with 5,5 from 7 (ref fig. 5). So at this scene, we decided to not put more features in it as it already feels quite intractable.

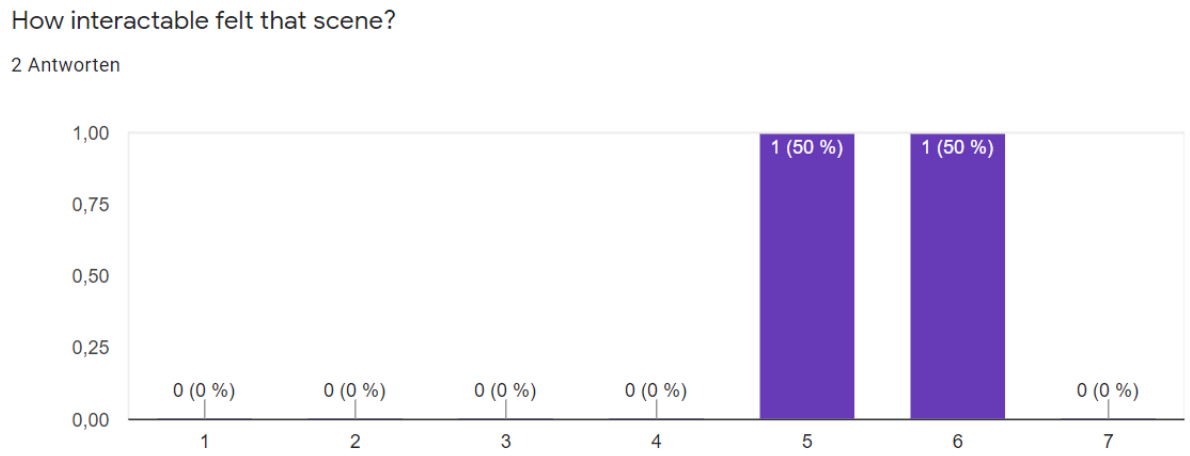


Fig. 5: Interactivity of Scenario 1: Inspector Scene

Scenario 2: Portal Scene

The portal scene itself could not fully be tested as it was spawning somewhere in the AR-Scene. Participants were reporting that they also would like instructions. When they were finding the portal they felt that it looked nice but as they couldn't reach it due to the not adjusted distance we can not say more about the actual environment.

As the scene did not work out properly and it also scored quite bad in the interactivity with 2,5 (ref fig. 6) we decided to add a plane tracking and some instructions to guide the user.

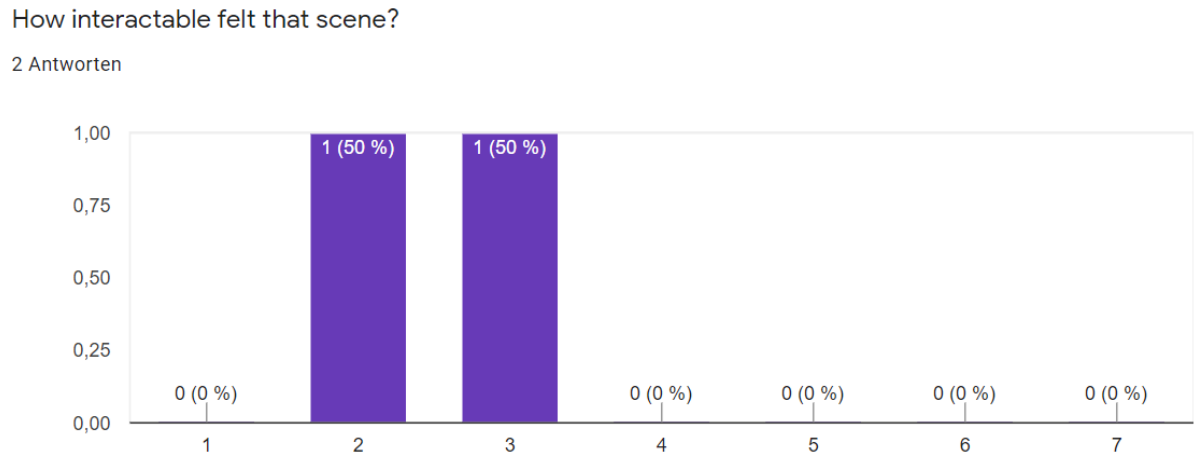


Fig. 6 : Interactivity of Scenario 2: Portal scene

Scenario 3: Battle Scene

The battle scene was reported well by the Participants. Every participant could finish the Scenario. The only things that bothered them were the missing instructions on where to put the card and the stats, which were

not facing the camera while battling. Out of the Observations and the testing, we decided to add instructions and placeholders, as well as a function so that the stats will always face the camera.
As the scene was reported to be quite interactive with a score of 6 out of 7 (ref fig. 7), we decided to not add more features.

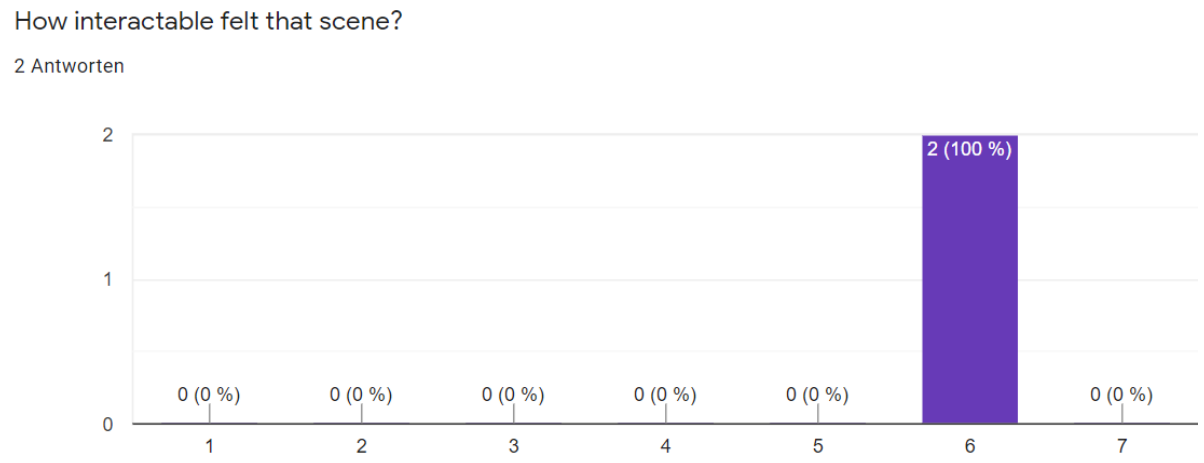


Fig. 7: Interactivity of Scenario 3: Battle Scene

4. Future Work

Since all major flaws of the usability test have already been resolved by an additional development iteration, we would like to evaluate the application again and see the effect on potential users. In addition, we would like to add more monsters to the application and incorporate more monster types and attacks subsequently. Since the battle system itself is rather simple right now, we would like to make the battle system more complex to enhance competitiveness. This would include high fidelity monster stats such as an attack, defense, and speed values as well as supplementing high fidelity damage calculations but also healing. For the latter, we think about implementing the rock-paper-scissors mechanic of the monster types at which some attack damage values would be boosted due to an effective type.

5. Literature

Affairs, A. S. for P. (2013, September 6). *System usability scale (SUS)*. Usability.gov. Retrieved January 11, 2022, from https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Jeff Sauro, P. D. (n.d.). *5 ways to interpret a sus score*. MeasuringU. Retrieved January 13, 2022, from https://measuringu.com/interpret-sus-score/