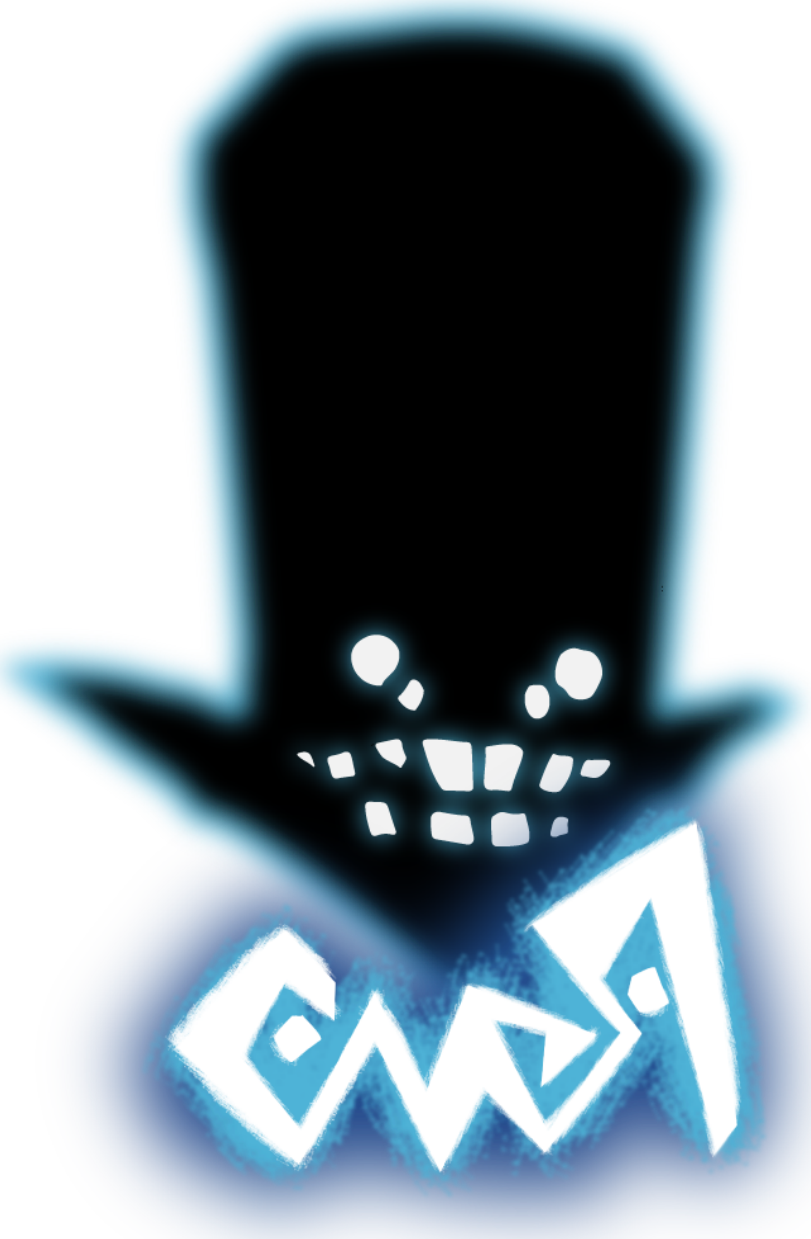
eveR
Technical Design Document



Index

Overview

Game Summary

Ever is a **3D Bossfight** Shoot'em up game in top down view where the player embodies a human child, accompanied by a magical orb that gives him **Telekinesis powers**, With this power, he will have to **upgrade his skills** in order to defeat powerful enemies.

System Requirements

- **Computer** with at least **Windows 7**
- A **keyboard and a mouse** or a **controller**

Technical risks/ challenge

The main risk of this project is the fact that at the beginning, we didn't know when we may get the 3d assets so, even if we knew that we had to save some time to **integrate** them, we didn't know when we may be able to start the integration and had to find ways to not being dependant of too much assets.

Furthermore, having **3 bosses**, **cutscenes** and **upgrade system** is quite a **large scope**, which means that we had to make **flexible and reusable code**.

Tools

Engine Used

Unity 2018.3.2f1

3rd Party Tools/ Assets

- InControl
- DoTween
- Amplify Shader
- Cinemachine

Versionning

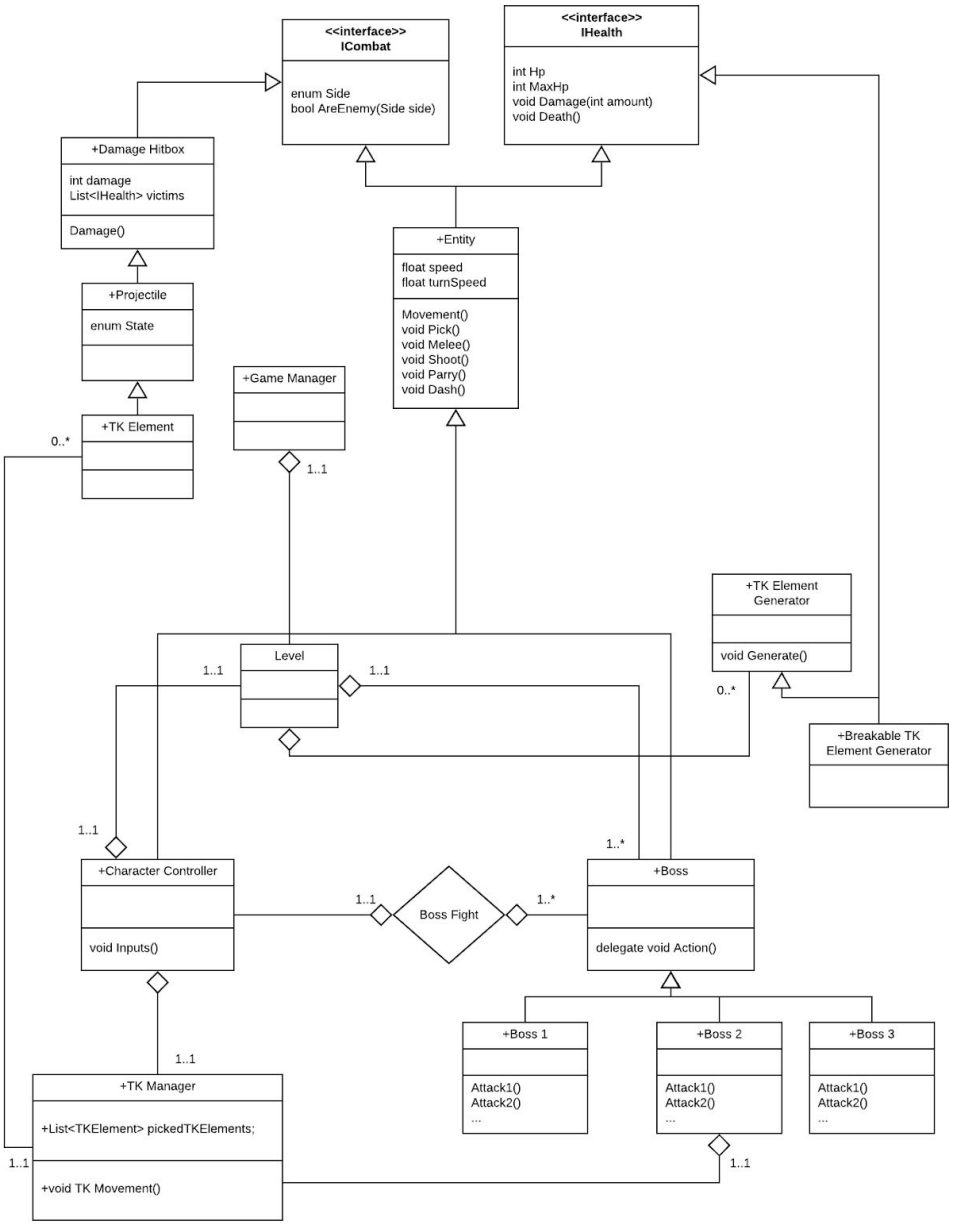
Gitlab + Sourcetree

Code Overview

General architecture

The game is jointed around the **boss fight**. There is **skill upgrades**, **cutscenes**, **dialogues** but everything is in the game to give more importance to the boss fights. The **LD elements** provides advantages or disadvantages to the player, the music support the fight climax and obviously the bosses give challenges to the player.

Entity Diagram



Entities Description

IHealth : interface that is used for everything that is damageable in the game.

ICombat : interface that store the side of each combat element.

Entity : Parent to either the character controller and the bosses, with all the common element between them.

CharacterController : Class that detect all the inputs of the player.

Boss : Script that describe the behaviour common to all bosses (each one of them has its own children script that describe his specific behaviour)

Hitbox : Script that deal damage when an enemy entity is in its detection area.

Projectile : Specific hitbox that is shoot by entities, it can have several states (like “isExplosive” or “isBouncing”). Every enemy projectiles can be parry by the player

TkElement: Specific projectile that can be picked by the player telekinesis power. Every player attack use them.

TkManager: Script that manage the TkElement picked by the player

TkElement Generator: script that generate tkElement on the map. There is two variations of that script, one generate tkElements after a constant time and another is breakable and generate them when it’s destroyed.

Technical features

Input System

The game is playable with a **controller** or with **mouse and keyboard**. We chose to use **InControl** for the **input system**, so we can use the same action to detect an input regardless of the controller used. **InControl** also allow us to let the player **rebind** his control as he wants.

AI

At the beginning of the project, we made a **mistake**, we didn't define a **boss template** before designing them. For this reason, the three bosses have very few in common, so the parent boss script only contain the things relative to the **phases switches**.

The first boss' AI is pretty simple, the boss **move slowly in the direction of the player** and each 2 seconds, use a **random attack pattern** in the pool he can do this phase.

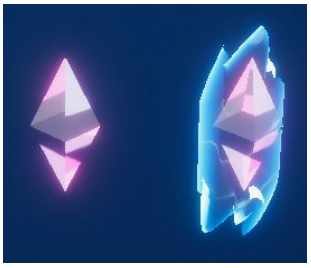
The second boss' AI is a bit more complex, the boss always **try to get crystals** (like the player) so he always **move pretty quickly** from pack to pack and don't have the same attack pattern when he have crystals and when he doesn't.

The third boss' AI is particular because sometimes the **boss and his head are pulling apart** and they have **different patterns but synchronized**.

Graphics

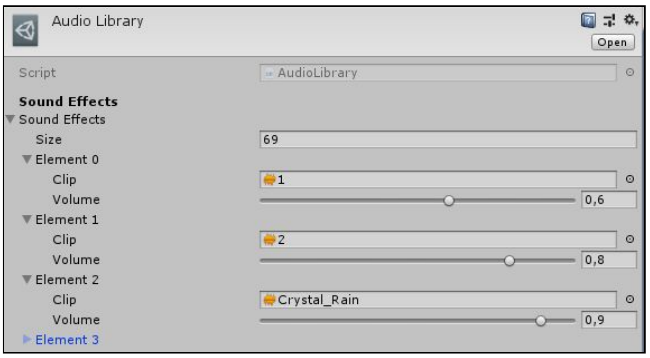
We made the choice to **not use HDRP** from Unity mainly for **optimization and possible risks**. The artistic direction being cartoonish, we applied a **toon shader** and keep things as simple as possible in terms of rendering.

Some shaders were still made using **amplify shader** for some specifics vfx like the character or UI.



Sounds

The system for the SFX of the game is pretty simple, there is a **singleton** named "**Audio Manager**" which contain the function that play the SFX and also a **library** (a scriptable object) with a big array that store all the audio clips of the game with a **volume associated** with each one of them. This system allow us to get the simplicity to play all the SFX at the same place but also to tweak their volume separately.



For the music, there is a system which activate **additionals instruments' layer** at some point of the fights, so the music get more and more intense as the player get further into the fight.

Save System

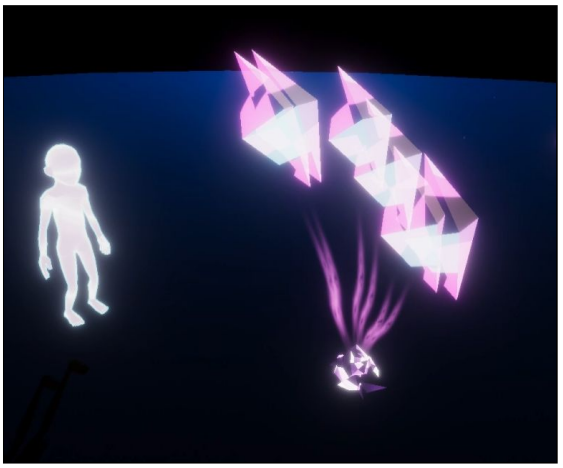
To save the player progression and settings, a **Game Handler** class manage all the data like the current level, the upgrades, the bindings etc … This class convert a storage class into json file on save and read it on load. In the case the file doesn't exist, it will create it and set all progression and settings value to default.

Character Controller

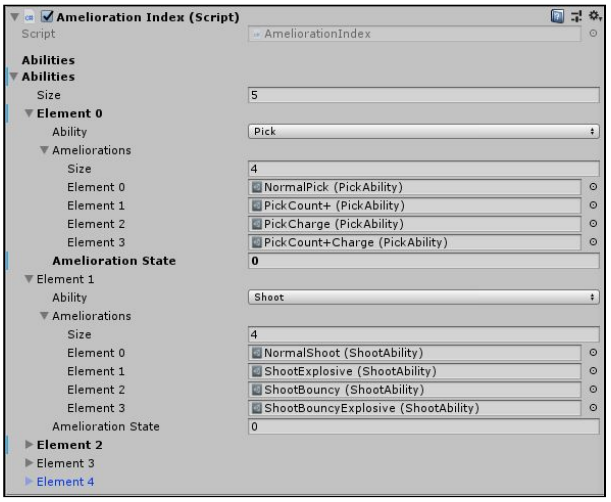
The character always **move forward**, and he is always turning to **face the inputs direction**. That allow us to use **only one animation** for all the character's movements.

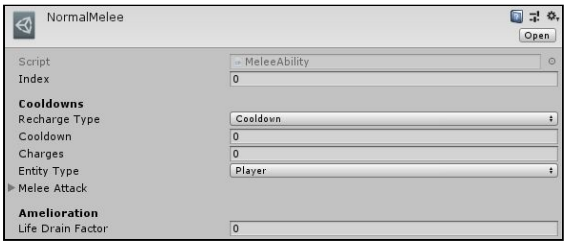
To increase the game feel, a slight **smooth time** is applied on the **character rotation** without being too intrusive in terms of responsivity. Same goes with an **acceleration** to give a more enjoyable movement.

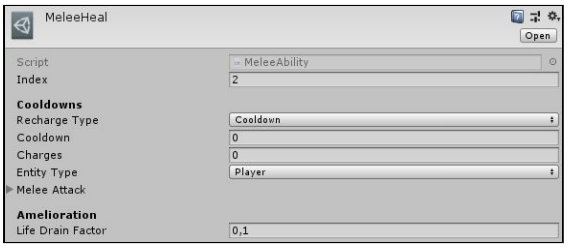
The character abilities being related to crystals, it was really important to make the behaviour of these as **smooth and understandable** as possible. In this idea we made **modular animations** using some spring function and tweens.



Upgrade System







Because the abilities are all made with **scriptable objects** it was easy to create **variation**, only by creating new instance of them. All the instances are stored in a **singleton** class that contain, for each abilities, 4 scriptable objects (the base skill, the first upgrade, the second and both) and an integer that is the current amelioration state between 0 and 3. At the beginning of each fight, the ability of the character is changed by the one corresponding to the current amelioration state. Finally, all the amelioration states are stored in an array that is saved and load by the save system.

Camera

In the cutscenes of the game, we used **cinemachine** to make smooth camera shots and transitions.

In the boss fight phases, the camera is a top down camera that **focus on the point between the player and the boss**, the **position and field of view adapt** itself so the characters never leave the screen and there isn't a large unoccupied space on the screen when the characters are close.

Pool System

We used **pool system** to re-use the same projectiles again and again rather to instantiate them every time we need them. That made us save some performance especially when there is a lot of projectiles in the screen at the same time.