Technical Design Document

Overview

Game Summary

Tower's Up is a strategic fast turn by turn game in which player will have to control pieces to build a tower. The objective is to push the other player to make the tower fall. To do this he will have several powers to help him in his task.

In Crazy Tower, you will choose the location of the piece that will fall and build a tower. This is a versus multiplayer turned based game where your goal is cause the fall of the tower by the other player. The players starts with (X) HP and each piece that will of the tower will cost 1 HP.

System Requirement

- Computer Running Window 7+
- A keyboard or a Controller
- Graphic Card 1080+

Technical Risks / Challenge

The main challenge of the project is to have a fully functional turn by turn game with two distinct controllers.

Tool

Engine Used

Unity 2018.3.2f1

3rd Party Tool / Assets

- InControl
- DoTween
- EZCameraShake
- Aura 2

Versionning

Using Unity Collaborate Service.

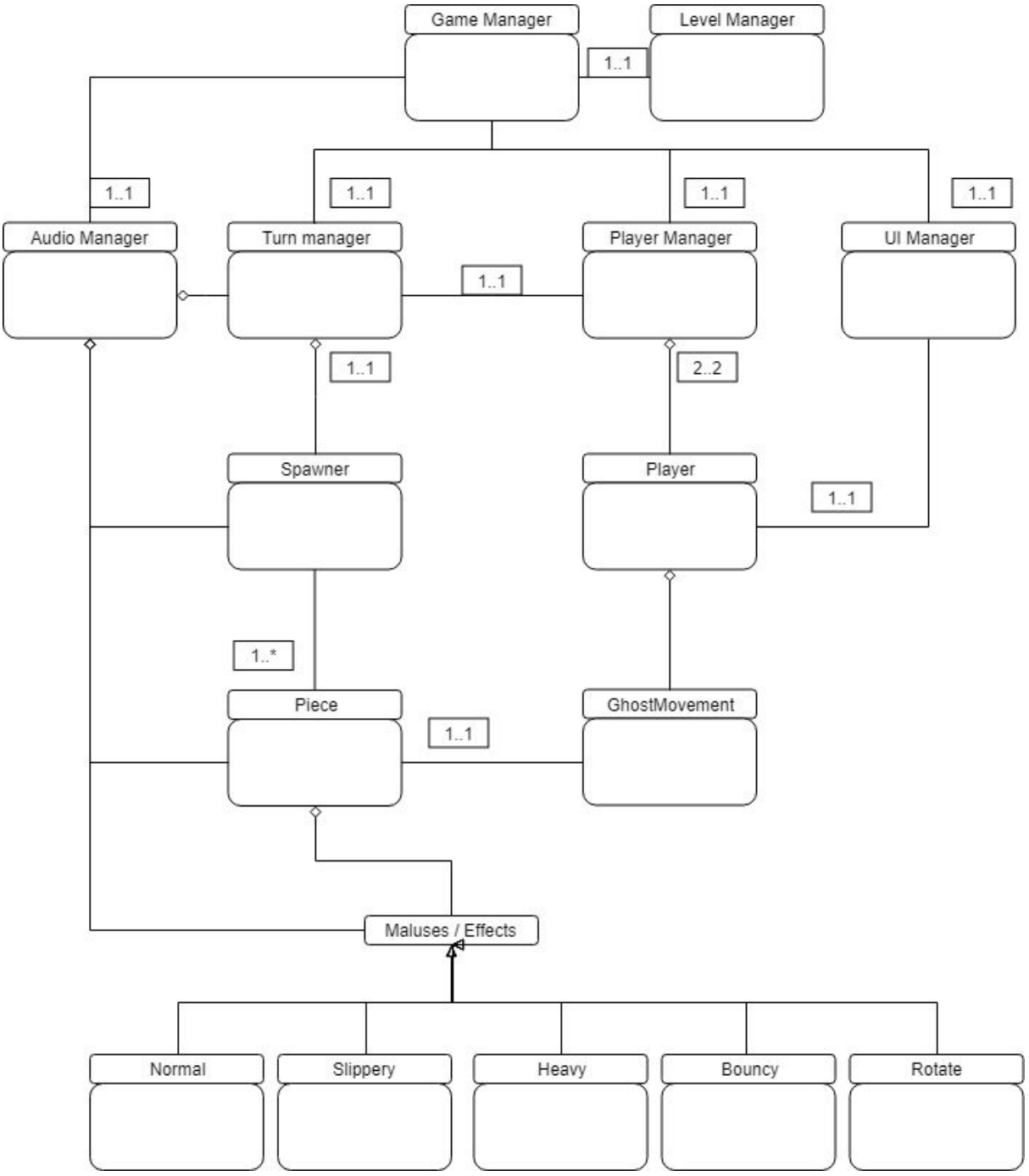
Code Overview

General Architecture

The game takes place in only one scene that contains various elements (explained in the entity diagram and the technical features)

We created several manager that handles all the big parts of the codes. They are called in all the different scripts. We created a Game Manager, an Audio Manager, a Player Manager, a Turn Manager and a UI Manager.

Entity Diagram



Technical Features

Input System

InControl

We used the asset InControl to ease the use of the controller. *"InControl is an input manager for Unity3D that standardizes input mappings across platforms for common controllers. It is written in C# and strives to make it easy to add cross-platform controller support to your game."*

http://www.gallantgames.com/pages/incontrol-introduction

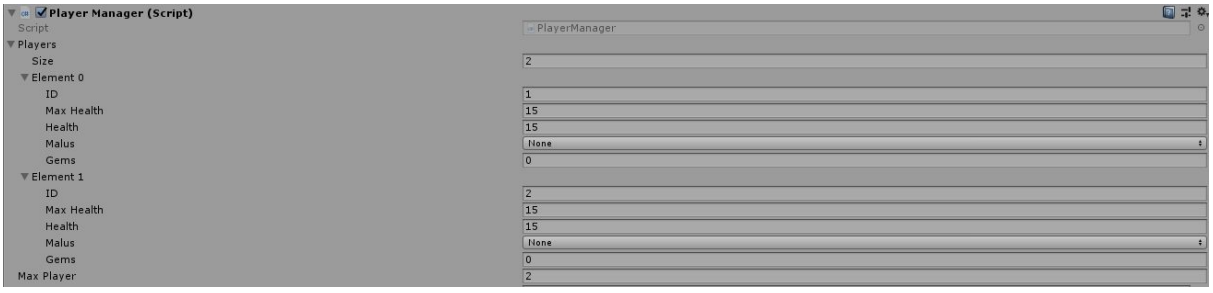
Since we want to develop the game on different platform we thought it will be useful to start using InControl from the beginning of the project. It is in the Script ***Player*** that we initialized InControl. The script manage also all the player's stats (Max Health, Health, Current Malus, Gems, etc.).

Player Manager

The Player Manager handle everything to the number of player in the game. When a player join it will add him in a list, and when the player leaves it will remove him. For the game right now the only playable condition is when two players joined and does not disconnect their controller.

The Player Manager has a "Don't Destroy On Load" features that allow the players to navigate through different scenes.

This Script also Initialize the UI for the first time you play. If you replay a game the initialization will be handled by the Game Manager. It is also where you can see the player's Stats (HP, number of gems, current Malus, ...)



Physics

Piece's Physics

Even though the game is in 3D, all the physic of the game is in 2D. We decided to use 2D physic first because we didn't need the Z axis and because the 2D physic of Unity is a little more reactive to this kind of game. We used the Rigidbody2D component on each piece. We also added an AddForce script on each piece so that the piece can take speed with time over its fall. It makes the overall game more organic.

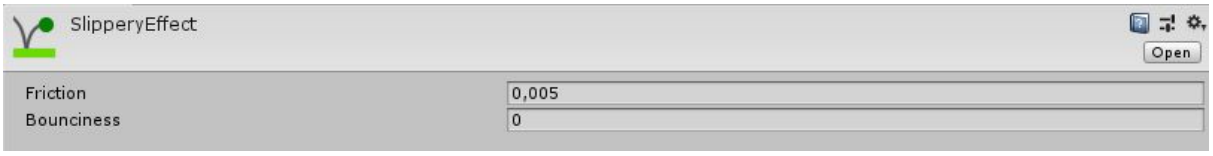
We used a polygon collider to make the hitbox of the piece as precise as possible so that when it is in contact with other piece it reacts as clean as we could.

Powers

Once a player collect 3 gems, he will obtain a random Power. There are currently 4 powers implemented in the game, 3 of them touch the physic of the pieces.

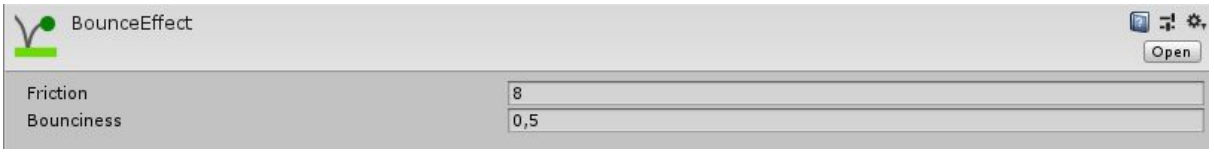
Ice

The Ice Power makes the piece slippery and slide on every other things that collides with it. We used a ***PhysicsMaterial2D*** for this power. We reduced the friction so that it is less slowed by it.



Jelly

The Jelly Power makes the piece bounce on others. We also used a ***PhysicsMaterial2D*** for this power. We did not touch the friction but we just rise the Bouciness as you can see below:



Heavy

The Heavy power make the piece fall faster and so have a bigger impact on touch. Since we used an AddForce in the physic (that is multiplied by a mass) we increased the mass of the piece.

Rotate

The rotate power lock the rotation of the piece but only in the controller. The player can no longer rotate his piece with the trigger of his controller.

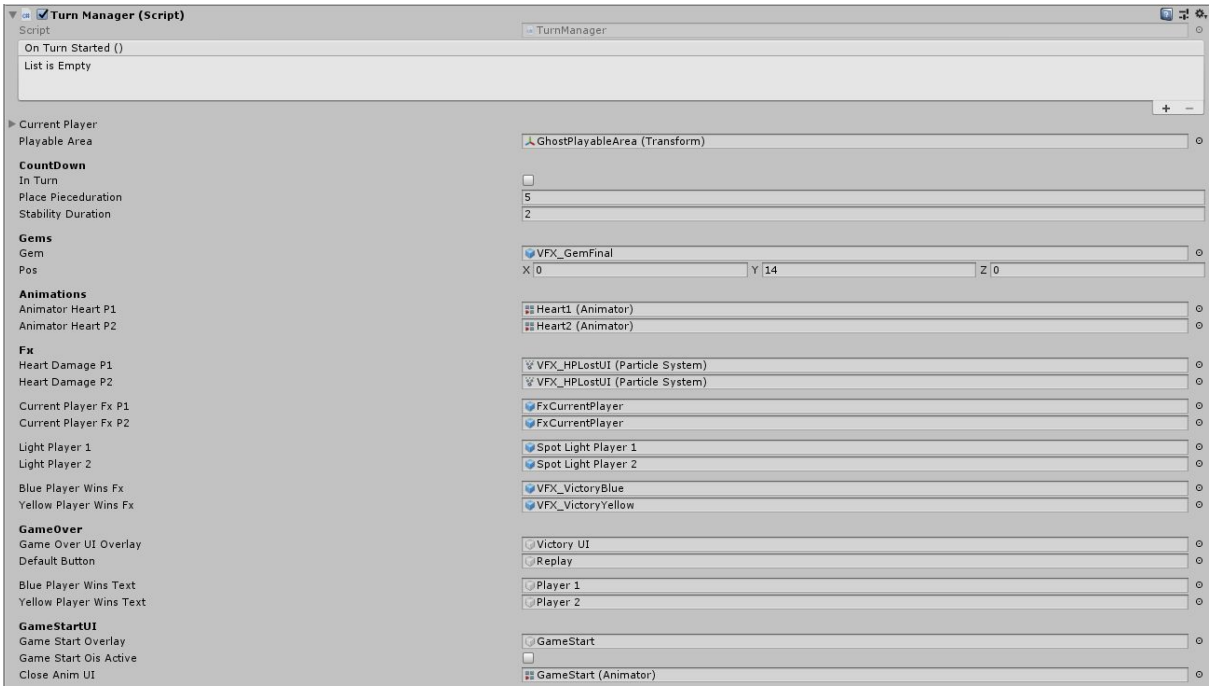
To emphasize the use of a power we scripted a function that on use, the vignette of the screen is more visible so that the player can focus easily on the action and we froze the time during the animation of the Fx. That way the player can really know that his opponent used a power on him. It also make the power more impressive.

Multiplayer

Turn Manager

The turn manager handle a lot of things. First it handles of course the turn by turn system. At the beginning of each turn it will call the function “Set Turn ()” which will change the current player, spawn a new piece, etc. Then in the “Update ()” it will follow its course : let the player place his piece for 5 seconds, then checks if all pieces are sleeping (if they stop moving), then will use the function “Switch Turn()” which switch the current player to the other one and then re initialize the “Set Turn ()” function.

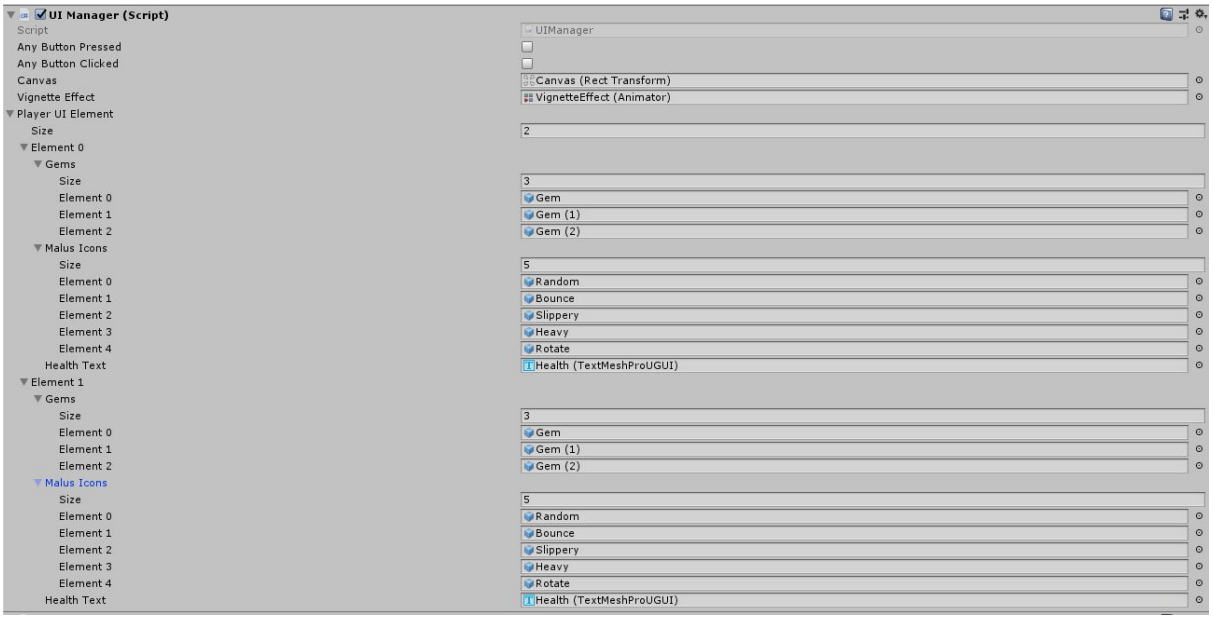
It also handle different things like the color of the background / different animation according to the current player, etc. This is also where you will find the “Game Over ()” function. which checks if the current player have more than 0 HP.



GUI / HUD

UI Manager

The UI Manager handles the UI of the Game (no surprise there). The "***PlayerUIElement***"



Timer

When the player place his piece, he has 5 seconds to do so. We needed a timer that is organic and easy to see. We were inspired by the Stamina bar in Zelda Breath of the Wild. The Timer is a HUD following the current piece.

Audio

Audio Manager

The Audio Manager is pretty simple, it contains a ***list of AudioClips*** that receives all the audio clips, and an ***array of AudioSources*** (so that it can play multiple sound at the same time).

It has been se to an instance so we just have to call it the following way when we need it:

```
AudioManager.instance.PlaySFX(AudioManager.instance.audioclip_List["index of sound"]);
```