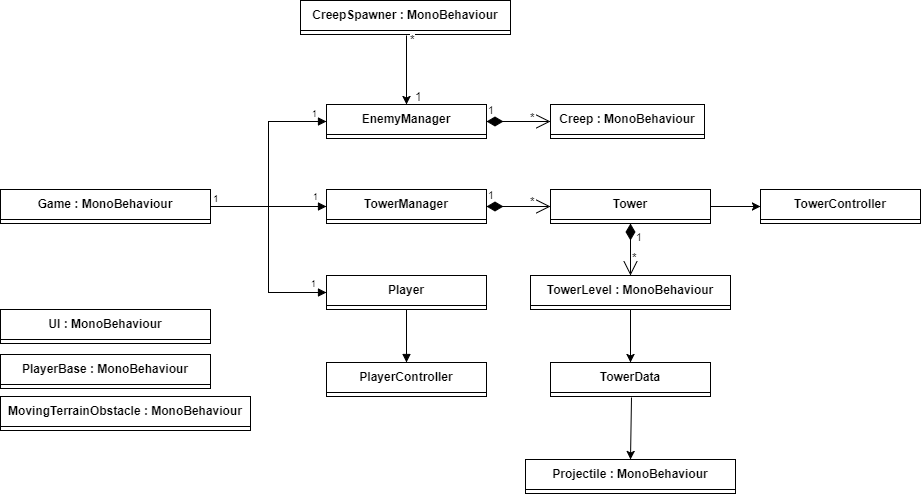
Tower Defense Game Documentation

by Gabriel Cirera (January 2018)



# Class explanation

***Game***: This is the starting point class and contains all the basic elements to run the game.

***EnemyManager***: Stores the enemies being spawned and also the enemy templates loaded from resources. It allows to spawn a random enemy loaded from resources.

***Creep***: This is a monster and it moves the instance through the navigation path on the terrain. It represents the model and the renderer at the same time to simplify code.

***CreepSpawner***: Controls the sequence of spawn of a creep. It uses the Fibonacci sequence to simulate in a fast and easy way the waves of enemies and the increase of difficulty.

***TowerManager***: Stores the tower instances and also the tower templates loaded from resources.

***Tower***: This is a container of the upgrade levels of a tower and manages the current level of upgrade and it has a controller.

***TowerController***: This manages the shooting and rotation of a tower.

***TowerLevel***: This is the level upgrade of a tower and it comes with its *TowerData* with tweak variables and its renderable.

***TowerData***: it has tweak data for the towers as shooting rate or radius.

***Player***: Contains all the data used for the player as health, coins, score and its controller.

***PlayerController***: Controls the player state and input and gets information from the UI when a button is pressed. Has also the ability to create and upgrade towers.

***Projectile***: it represents the model and renderer of a projectile.

***UI***: it contains the whole ui to manage the game: buttons and counters and the end level popup.

***MovingTerrainObstacle***: it represents an obstacle that dynamically modifies the path of the creeps in the terrain.

# Assets Distribution

I usually like to distribute the assets on a self-explanatory folder.

Materials: all the materials being used on the game

Prefabs: the prefabs being used on the game

Resources:

* Enemies: All kind of prefabs enemies we would have on the game.
* Towers: it contains prefabs of all the tower types with its levels of upgrade.

Scenes: the scenes we use on the game. In our case we have only one.

Scripts: all the scripts being used on the game.

NOTE: The structure should be improved while adding more assets, more detailed folder should be added while adding more enemies, tower types and new elements to the game. As this is a very simple demo I did not complicate it.

# Scene

### MonoBehaviours

In the Scene I added some manual *MonoBehaviours* into some instances to make it simpler. The *Game* class has its own *GameObject* so it has the whole logic on it. On the other hand the *CreepSpawners* are also added manually into the level in the right place, in our case inside the paths.

The *UI* is another element added into the scene as it is not going to be dynamic or modifiable.

The Obstacles are also setup on the scene. In this case I only added one that moves to block the path to the enemies called *MovingTerrainObstacle*.

### Navigation Paths

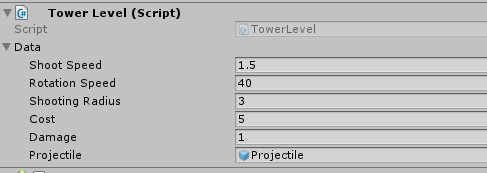
I created paths for the enemies because I found it more fun and more challenging for the player. My first version I did not use paths and the setup of the scene had to be more complex to make it a bit fun so I decided to use paths.

# Prefabs

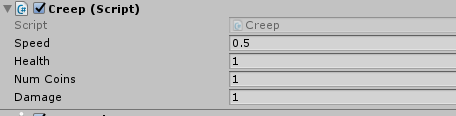
I created some prefabs to ease tweaking.

In the folder Resources you can find the Towers and tweak the following.

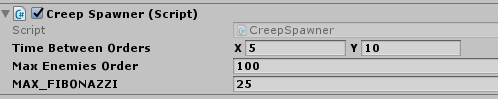
TowerLevel:



Creep:



CreepSpawner:



NOTE: The Player class is not holding from a prefab because we do not want the designer to modify any of its values, but if we would like to do be able to modify its values such initial coins available or initial health we would need to create a Prefab.

# Interaction between logic and assets

To communicate between assets I mainly use delegates. This is because it is a functionality that already comes with C# and it is easy to use. The problem is that this generates dependency, and I do not like such thing. I would have preferred to have an Event System to make it more generic.

In one case with the projectile when it gets its target and has to apply damage I send a message to its target with the function name “*ApplyDamage*” to make it more generic.

### Player delegates

* CoinsChanged: called when earned or used coins.
* PlayerLost: called when player health is 0 or less.
* ScoreChanged: called when score earned.

### PlayerController delegates

* PlacingTower: called when the player has created a tower and needs to place it on the terrain.
* TowerConstructed: called when the player finally clicks to build the tower.
* TowerSelected: called when the player selects an existent tower.
* TowerCanUpgrade: called when a selected tower can be upgraded.

### PlayerBase delegates

* PlayerBaseTouched: called when a creep enters in the base this is called.

### Creep delegates

* CreepDied: called when enemy died.

### UI delegates

* CreateTowerButtonPressed: called when the create tower button is pressed.
* CancelTowerButtonPressed: called when the cancel tower button is pressed.
* UpgradeTowerButtonPRessed: called when the upgrade tower button is pressed.
* OnReplayGameRequested: called when the replay game button is pressed.

# Resources folder

I preferred to use the resources folder to avoid having to assign manually stuff on the scene. I know this has given me problems many times while modifying assets, classes and scenes. The scene has a minimal setup and then all the Game class manages the instantiation of the main systems.

The EnemyManager will load from resources all its available enemies, the more they are created on the folder the more variety we will get on the game.

The TowerManager does the same with the tower prefabs, it loads them all and prepare them to be instantiated by the user. In this demo I only created a Basic tower with 3 level of upgrade.