Report Software Architecture

For the course Software Architecture a report is required to inform the reader of design choices.

First off, lets inform the reader of some of the unchanging ground rules of this code base.

## Naming conventions

Class names: PascalCase (I.E. public class PlayerController)

Public variables: Camelcase (I.E. public GameObject cameraObject)

Private variables: camelCase plus m\_ (i.e. private Material m\_tileMaterial)

Enums : all caps (i.e. Enum ENEMYSTATUS)

Public functions: PascalCase (i.e. public void Update())

Private functions: PascalCase (i.e. public bool CheckForValidPath())

## Scriptable objects (SO)

Scriptable objects are used in different cases.

* As replacement of certain enum’s.
* As a singleton for global variables
* As a configuration on how towers upgrade.
* As a configuration on how a wave composition looks like

Condition for replacing an enum is the need of multiple statuses an object can be with checks which overlap, example: according to a tile status, it can be built upon, if it’s a spawn point, end point or already built upon it cannot create another tower on top of it. However if it’s an open spot or a previous pathing node it can be built upon.

Singleton for global variables, this is useful for multiple controllers that use the same variables. For example the amount of resources you have is required for the ResourceController to see whether you can build or not, it is also used by the ResourceTextController to set the correct text value on the UI.

Configuration on how tower upgrades, this is a requirement in the manual and gives flexibility on how the upgrade system works in order to balance the level. A tower can be upgraded per level and the level increments are described in the SO.

Configurating a wave composition is an easy way to create new waves and the strength of the waves.

## Interfaces

I made use of some interfaces as well, as this was recommended and is good practice in general.

However I noticed that a lot of entities did not have a lot in common to justify setting up more interfaces. I did it for Enemy, however due to every type of enemy inheriting from the Enemy class there were no reasons to edit the interface or re-use the interface in another component.

## Singleton

As described before, I’ve made use of singletons for global variables. This uses the Interface ISerializationCallbackReceiver to control serializing and deserializing the variable.

An example of this is the ResourceValue SO, there is a text controller for controlling UI parts. A resourceController to see if towers can be bought with the current value if towers are being bought or upgraded.

## Observable

For seeing if an enemy died, I made a GameEventEnemyDied, which EnemySpawner listens to. On an enemy dying or disappearing the EnemySpawner it gets an event which Enemy died. This has the advantage that 1 EnemySpawner class can see if there are any Enemies left from the wave before spawning new ones.