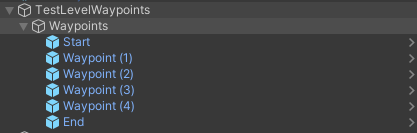
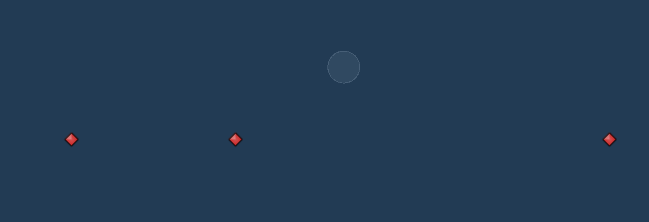
## **Level Creation –**

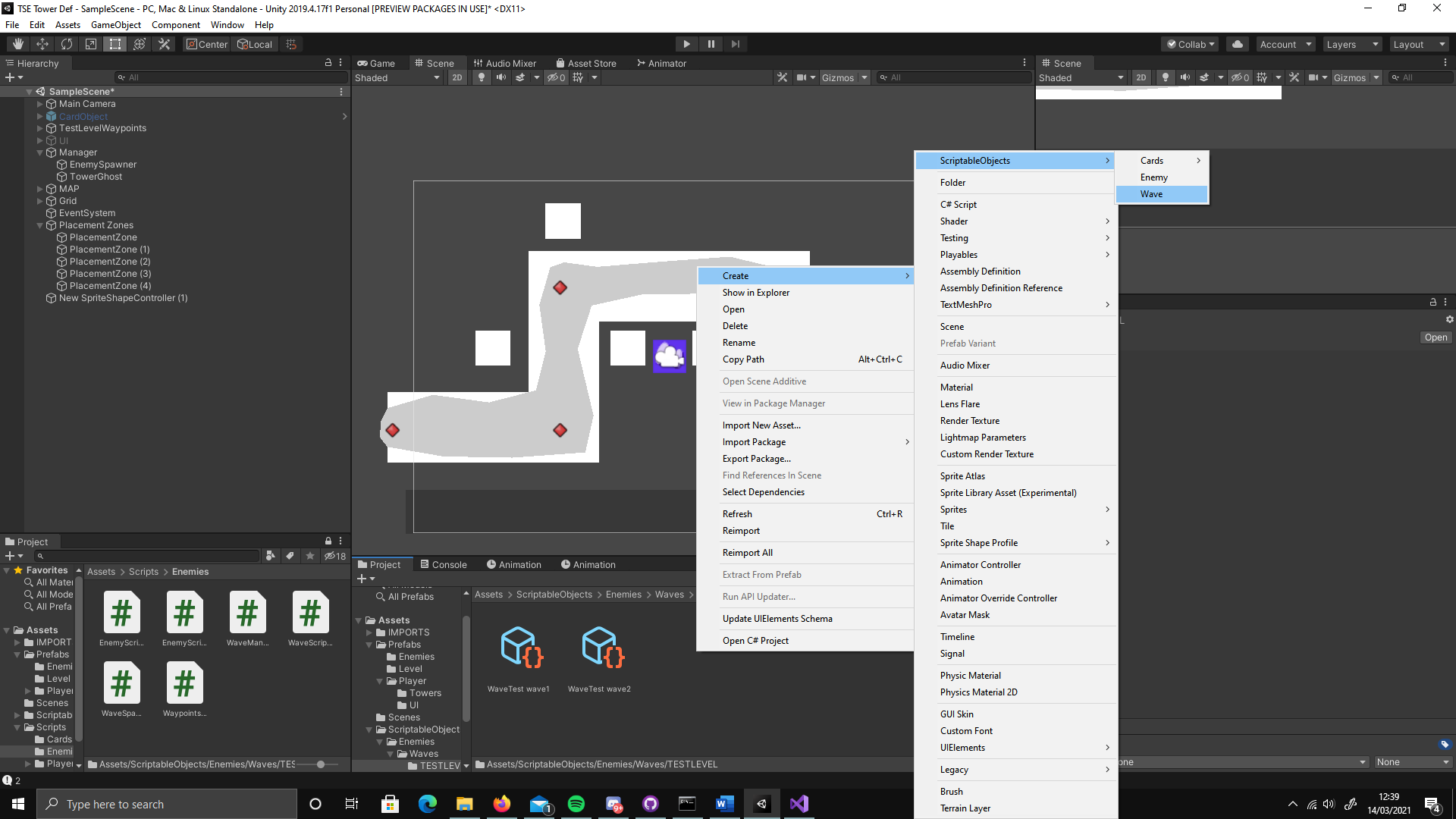
**Waypoints –**

The prefab “Level Waypoints” can be used to create a series of empty objects shown to the right. Adding additional waypoint prefabs between Waypoint(1) and end will add additional points for the enemies to path to, as shown lower right.

The red diamonds shown below are what represents the Waypoint empties, and do not appear in the actual game, only the editor, allowing for clearer representation of the path being formed.

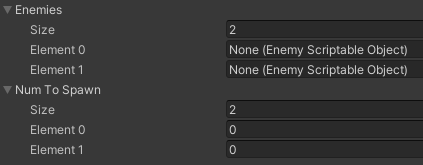
It is advised to create a level map which then has the waypoints following a path set I n the artwork.

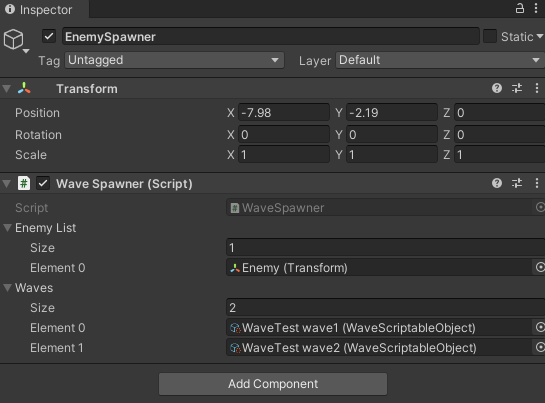
### **Tower Placement –**

 The prefab object “Placement Zones” provides an empty to store the zones and a single placement zone which can be duplicated to cover a map. The zones are tagged as “Placeable Zone” to work in the tower placement coding.

### **Enemy Waves -**

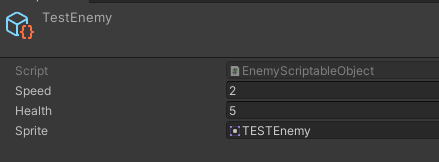
To create a wave, create a scriptable object by right clicking in the project window, Create, Scriptable objects, Wave. (Right)

The scriptable object will have 2 fields, Enemies Size and Num to Spawn size. Both should have the same size. The enemies fields take an enemy scriptable object, this will be what is spawned by the script using this object, the Num To Spawn sets how many of each unity to spawn, applying the position to the matching place in the Enemies, I.E. Enemies[1] spawns Num To Spawn[1].

 Once Waves are created they can be applied to the Enemy Spawner script under manager, the Enemy (Transform) object is essentially an empty base for the enemy stats and sprites to be applied to. Every wave should be added in order here.

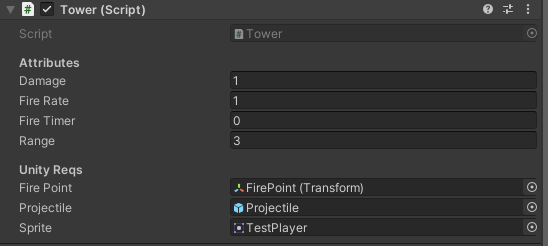
## **Objects**

### **Enemies –**

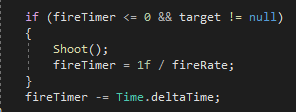
Creating a new enemy is done in the same way as a wave, create a Scriptable object Enemy and fill in the fields as necessary.

(This system may be updated to support more complex enemies.)

### **Towers –**

New towers currently require a new script or additional coding on the “projectile” script to allow for additional On-Hit effects. The 2 current projectiles are BASIC and EXPLOSIVE, with a single variable (explosion radius) deciding whether or not the effect occurs.

The Tower prefab object currently has 7 visible fields.

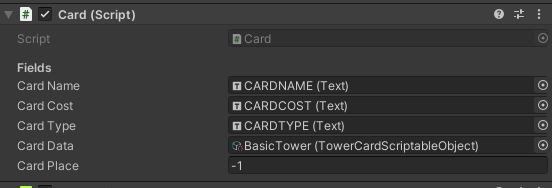
ATTRIBUTES  
Damage – The damage done to enemies hit.  
Fire Rate – The amount of shots per second. 1 second / fire rate.  
Fire Timer - The timer to count how long until the next shot, primarily visible for debugging. If set to a value there will be a delay of that many seconds until the first shot.  
Range – how many distance units the tower can fire over.

UNITY REQS  
Fire point – Where the projectile will fire from on the tower, on the test models it is the centre.  
Projectile – The projectile being fired by the tower, currently what applies special effects such as explosion.  
Sprite – The sprite being used to represent the tower.

NONE OF THESE FIELDS CAN BE LEFT BLANK.

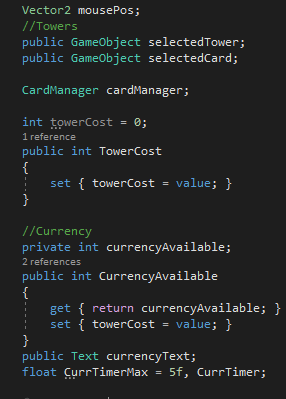
### **Cards –**

Cards Utilise scriptable objects again since they will be reused so frequently. The “Card” fields can be filled out as necessary to reflect the card, name, what it does and some card art. The standard stats will actually affect gameplay, currently the cost of the tower will effect currency needed to play the card and object to Make is a prefab of a created tower.

The card prefab will have all values from the card scriptable object applied to it through the deck script using the Card Data field. The fields shown in the image on the right are the text fields which will be filled, these are prefilled in the prefab so do not require altering.

# **CODE**

## **Manager**

Fields (right):

mousePos – Stores the mouses converted position to a vector for use in functions.

selected Tower/Card – References a gameobject used in the placement of towers.

cardManager – References the script (on the same object) used to control the cards and deck.

Tower Cost – Used in placement of towers, no Get function since it will not be used anywhere else.

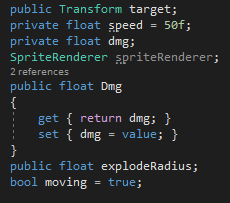
currencyAvailable – Tracks the current currency for playing cards, get and set since cards will check its value to se if they can be played.

currencyText – A Unity text field for displaying the current currency value.

currTimerMax/CurrTimer – both used in the timer to add currency, max is how long to wait and timer is simply a timer value reset to maxtimer at 0.

### **Tower**

### **BasicProjectile**

**Fields(right):**

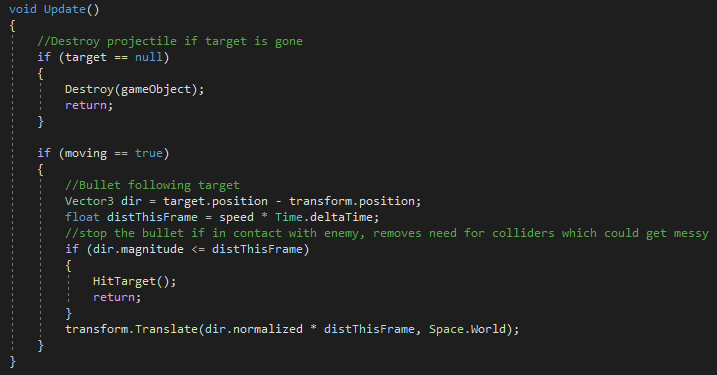
target: The position of the enemy the tower initially shot at, assigned upon this objects creation.

speed: the speed of the projectiles movement, could be altered depending on the projectile to add more variety.

dmg: the damage that will be dealt to the target, assigned upon creation by tower.

explodeRadius: Used to show how large an explosion will occur, if 0 there is no explosion.

Moving: tracks if the projectile is in a moving state.

**Update:**

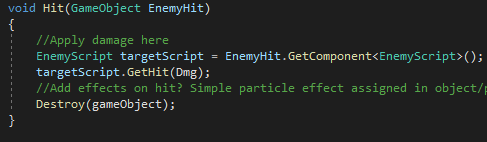
If the target is destroyed (likely by other towers) the projectile will destroy itself to prevent issues or lingering shots.

Direction is calculated by taking the targets current position and taking the projectiles current position away from it, this can then be used in calculations.

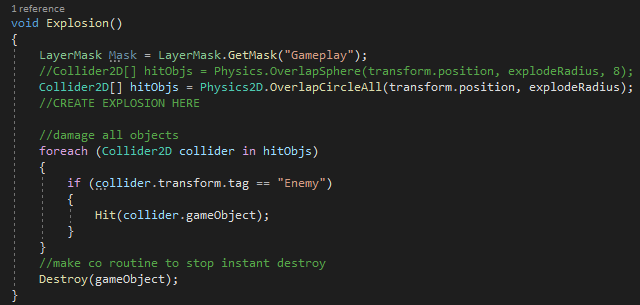
distThisFrame tracks the distance which the projectile has moved by simply taking the speed and multiplying it by the time since the last frame.

Finally if the magnitude of the direction movement is equal to or less than distThisFrame, it implies that it can no longer move, so the hit target is called.

**HitTarget:**

Simple function to call the hit on an enemy, if the explodeRadius is more than 1 (implying the projectile does explode) an explosion effect is called, otherwise a regular hit to the target occurs by getting the gameobject attached to the target transform and passing it to the Hit function.

**Hit:**

First gets the EnemyScript from the target gameobject, which should always be present in an enemy, and then calls a public GetHit function from it with the projectile damage as the input. Finally, the object attached to the script is destroyed.

**Explosion:**

First, a collider array is made based off an overlap circle, this will return **every** collider found in an area set by the radius for use in applying damage.

The actual damage is applied in a foreach which will check the tag for each collider and, if tagged as an enemy, call the Hit function mentioned previously. The use of the if statement is necessary as the hitObjs will return any collider, including non-enemies which wont have the Hit function. After this damage is completed the object is destroyed as normal.

### **EnemyBase**