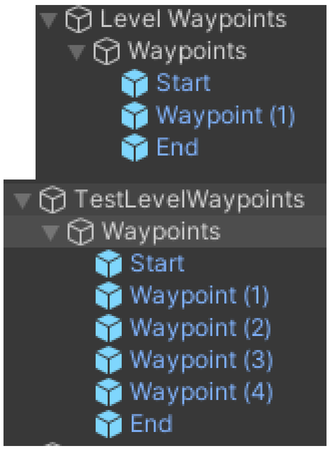
Level Creation –

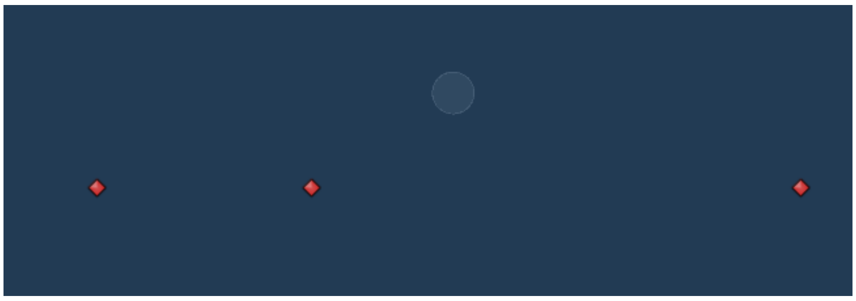
Waypoints –



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.

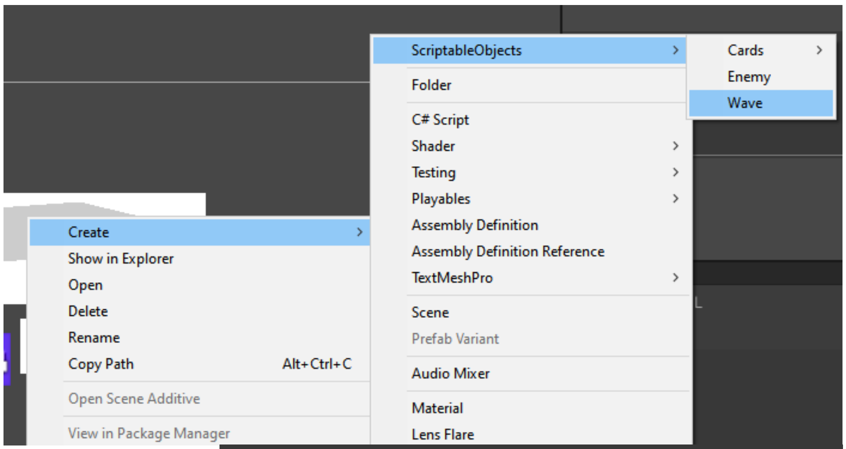
The red diamonds shown below are what represents the Waypoint empties, and do not appear in the actual game, only the editor, allowing for clearer representation of the path being formed.

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



Tower Placement –

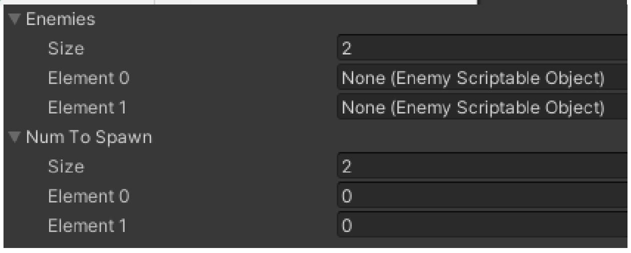
The prefab object “Placement Zones” provides an empty to store the zones and a single placement zone which can be duplicated to cover a map. The zones are tagged as “Placeable Zone” to work in the tower placement coding.



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].

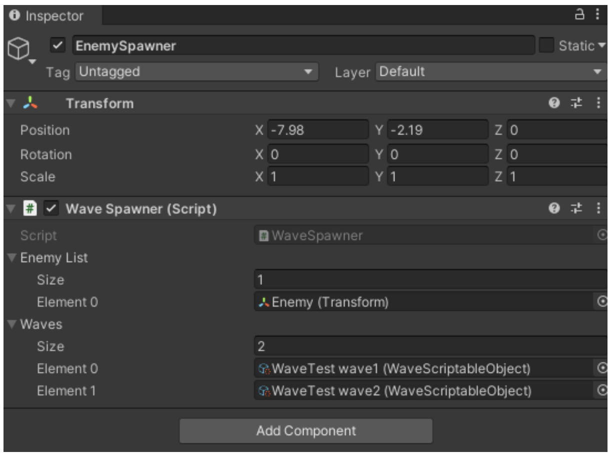


Enemies
- Size: 2
- Element 0: None (Enemy Scriptable Object)
- Element 1: None (Enemy Scriptable Object)

Num To Spawn
- Size: 2
- Element 0: 0
- Element 1: 0

Unity Version – 2019.4.17f

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.

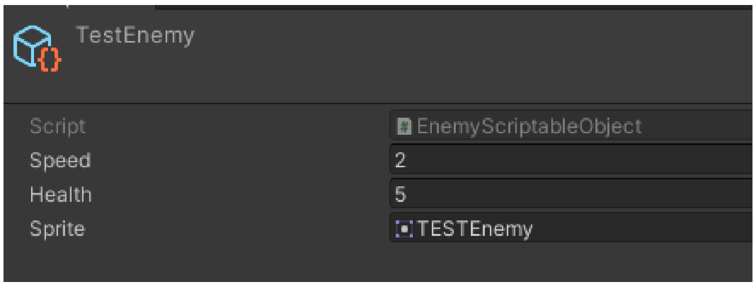

Inspector
EnemySpawner
Static
Tag Untagged
Layer Default
Transform
Position X -7.98 Y -2.19 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Wave Spawner (Script)
Script WaveSpawner
Enemy List
Size 1
Element 0 Enemy (Transform)
Waves
Size 2
Element 0 WaveTest wave1 (WaveScriptableObject)
Element 1 WaveTest wave2 (WaveScriptableObject)
Add Component

Objects

Enemies –

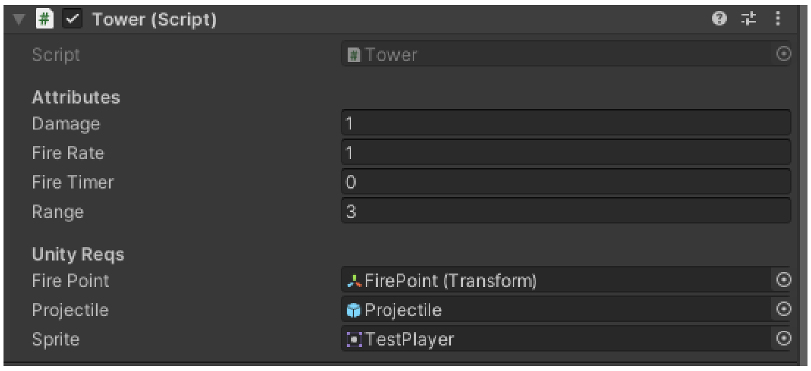
Creating a new enemy is done in the same way as a wave, create a Scriptable object Enemy and fill in the fields as necessary.

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


TestEnemy
Script EnemyScriptableObject
Speed 2
Health 5
Sprite TESTEnemy

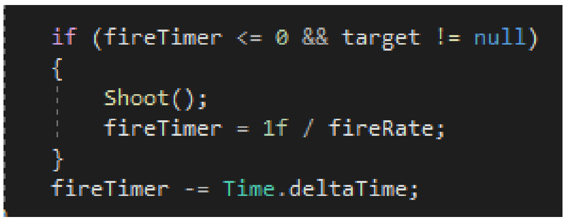
Towers –

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.


Tower (Script)
Script Tower
Attributes
Damage 1
Fire Rate 1
Fire Timer 0
Range 3
Unity Reqs
Fire Point FirePoint (Transform)
Projectile Projectile
Sprite TestPlayer

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.


```
if (fireTimer <= 0 && target != null)
{
    Shoot();
    fireTimer = 1f / fireRate;
}
fireTimer -= Time.deltaTime;
```

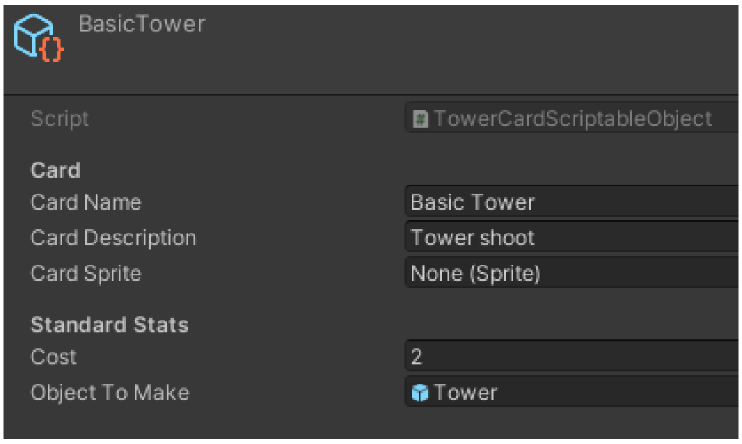
UNITY REQS
Fire point – Where the projectile will fire from on the tower, on the test models it is the centre.
Projectile – The projectile being fired by the tower, currently what applies special effects such as explosion.
Sprite – The sprite being used to represent the tower.

NONE OF THESE FIELDS CAN BE LEFT BLANK.

Unity Version – 2019.4.17f

Cards –

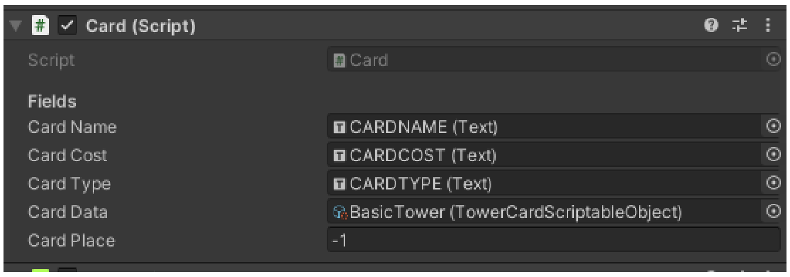
Cards Utilise scriptable objects again since they will be reused so frequently. The “Card” fields can be filled out as necessary to reflect the card, name, what it does and some card art. The standard stats will actually affect gameplay, currently the cost of the tower will effect currency needed to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



BasicTower

Script	TowerCardScriptableObject
Card	
Card Name	Basic Tower
Card Description	Tower shoot
Card Sprite	None (Sprite)
Standard Stats	
Cost	2
Object To Make	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.

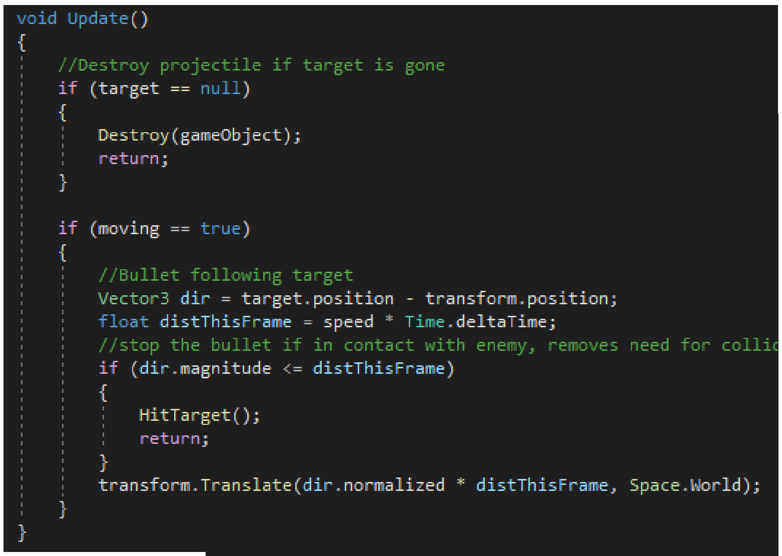


Card (Script)

Script	Card
Fields	
Card Name	CARDNAME (Text)
Card Cost	CARDCOST (Text)
Card Type	CARDTYPE (Text)
Card Data	BasicTower (TowerCardScriptableObject)
Card Place	-1

Unity Version – 2019.4.17f

CODE



```
void Update()
{
    //Destroy projectile if target is gone
    if (target == null)
    {
        Destroy(gameObject);
        return;
    }

    if (moving == true)
    {
        //Bullet following target
        Vector3 dir = target.position - transform.position;
        float distThisFrame = speed * Time.deltaTime;
        //stop the bullet if in contact with enemy, removes need for collic
        if (dir.magnitude <= distThisFrame)
        {
            HitTarget();
            return;
        }
        transform.Translate(dir.normalized * distThisFrame, Space.World);
    }
}
```

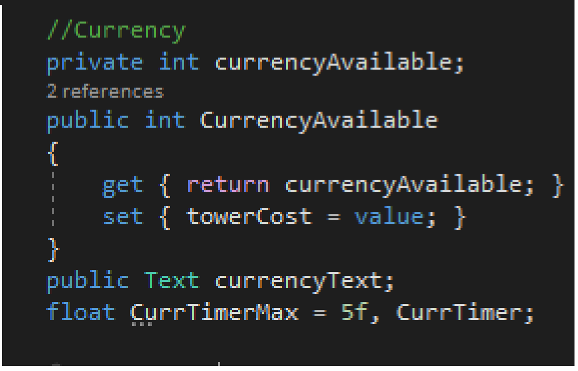
Manager

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.



```
//Currency
private int currencyAvailable;
2 references
public int CurrencyAvailable
{
    get { return currencyAvailable; }
    set { towerCost = value; }
}
public Text currencyText;
float CurrTimerMax = 5f, CurrTimer;
```

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

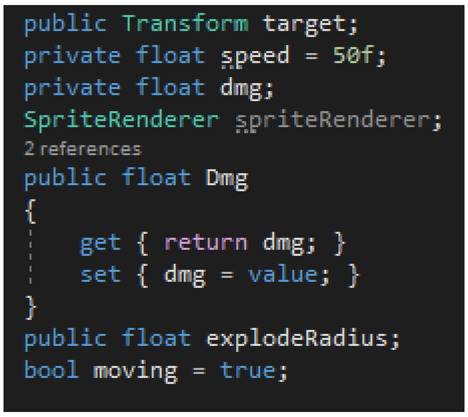
currencyAvailable – Tracks the current currency for playing cards, get and set since cards will check its value to se if they can be played.

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



```
public Transform target;
private float speed = 50f;
private float dmg;
SpriteRenderer spriteRenderer;
2 references
public float Dmg
{
    get { return dmg; }
    set { dmg = value; }
}
public float explodeRadius;
bool moving = true;
```

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.

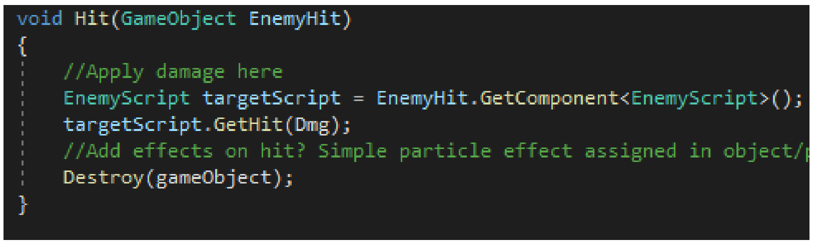
Moving: tracks if the projectile is in a moving state.

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.

Unity Version – 2019.4.17f

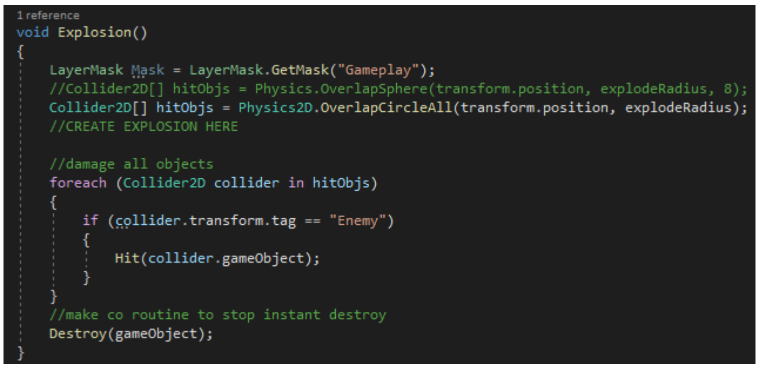
Finally if the magnitude of the direction movement is equal to or less than distThisFrame, it implies that it can no longer move, so the hit target is called.

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.



```
void Hit(GameObject EnemyHit)
{
    //Apply damage here
    EnemyScript targetScript = EnemyHit.GetComponent<EnemyScript>();
    targetScript.GetHit(Dmg);
    //Add effects on hit? Simple particle effect assigned in object/
    Destroy(gameObject);
}
```

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.

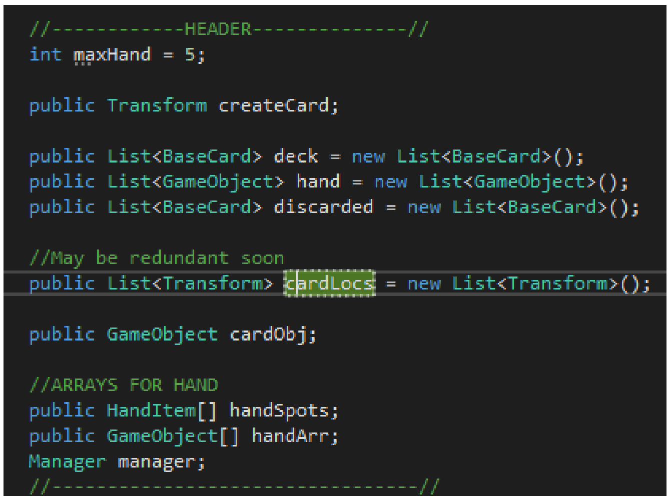


```
1 reference
void Explosion()
{
    LayerMask Mask = LayerMask.GetMask("Gameplay");
    //Collider2D[] hitObjs = Physics.OverlapSphere(transform.position, explodeRadius, 8);
    Collider2D[] hitObjs = Physics2D.OverlapCircleAll(transform.position, explodeRadius);
    //CREATE EXPLOSION HERE

    //damage all objects
    foreach (Collider2D collider in hitObjs)
    {
        if (collider.transform.tag == "Enemy")
        {
            Hit(collider.gameObject);
        }
    }
    //make co routine to stop instant destroy
    Destroy(gameObject);
}
```

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.

CardManager



```
//-----------HEADER-------------//
int maxHand = 5;

public Transform createCard;

public List<BaseCard> deck = new List<BaseCard>();
public List<GameObject> hand = new List<GameObject>();
public List<BaseCard> discarded = new List<BaseCard>();

//May be redundant soon
public List<Transform> cardLocs = new List<Transform>();

public GameObject cardObj;

//ARRAYS FOR HAND
public HandItem[] handSpots;
public GameObject[] handArr;
Manager manager;
//--------------------------------//
```

maxHand:
An int to limit the number of cards which can be held.
createCard:
Stores the transform assigned during drawing a card for card creation.
deck, hand, Discard:
Multiple lists used to store data needed to keep cards in play consistent, the hand actually only hold the card themselves, not the data as the others do. The deck list needs to be public to assign the deck in the debug and eventually a deck building system, though this could be changed to a public function eventually.
cardLocs:
List to store locations where individual cards can be created.

cardObj:
A prefab for a card where the data in the decks is assigned, a single cardobject prefab is used with multiple scriptable object data being assigned from the deck.
handSpots:
used in card creation to create cards based on a transform and whether the spot object is actually open.
handArr:
An array used to ensure that the card objects spawn in the correct spot.

Manager:
A reference to the manager script on the object, necessary for currency manipulation, assigned in the start function by getting the component from the parent gameobject.