Game Programming Using Unity 3D

Lesson 12: Enemy Spawning

Ferdinand Joseph Fernandez

Chief Technological Officer, Dreamlords Digital Inc.

Admin and Co-founder, Unity Philippines Users Group

November 2011

Overview

Again, we'll be continuing where we left off from the previous lesson. If you haven't done Lesson 11, you need to do it before you can continue here.

Right now we only got a handful of zombies. And when you kill them, that's just about it.

How about we let the computer make more enemies as time goes by?

Enemy Spawning

Let's make something simple first. We'll make it that new enemies come out (or “spawn”, as they say) every 10 seconds.

Make a new C# script. Name it “EnemySpawnManager”.

Add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     void Update()
10     {
11         Instantiate(_enemyToSpawn);
12     }
13 }
14
```

Make a new empty game object. Name it “EnemySpawnManager”.

Attach the EnemySpawnManager script to it.

Attach your Enemy prefab to the “Enemy To Spawn” property in the Inspector.

Run the game.

You'll find that the spawning is too fast.

Like how we delay an enemy's attack, we should give a delay to the spawning.

Illustration 1: Oops. That may be too much.

Add this to the code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Instantiate(_enemyToSpawn);
19             _nextSpawnTime = Time.time + _spawnDelay;
20         }
21     }
22 }
23
```

This will give a pause of 1 second before every spawning. You can change the interval by changing the value of _spawnDelay.

Spawning At Edge Of Screen

Right now the zombies spawn at a particular point. This doesn't look very good.

We can make it that the spawning takes place at the edges of the screen.

Add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Vector3 placeToSpawn = new Vector3(0, 0.75f, 0);
```

```
19              Quaternion directionToFace = Quaternion.identity;
20              Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
21              _nextSpawnTime = Time.time + _spawnDelay;
22          }
23      }
24 }
25
```

We added a couple of arguments to Instantiate: the place where the new zombie will appear, and what direction it will face.

Right now we're just telling it to spawn at (0, 0.75, 0) in world space. We just need to change this to the value of where the edge of the screen is.

We're also using Quaternion.identity as the direction where it will face. Quaternion.identity is the equivalent of saying 0 degrees in Euler angles.

Add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Vector3 edgeOfScreen = new Vector3(1.0f, 0.5f, 8.0f);
19             Vector3 placeToSpawn = Camera.main.ViewportToWorldPoint(edgeOfScreen);
20             Quaternion directionToFace = Quaternion.identity;
21             Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
22             _nextSpawnTime = Time.time + _spawnDelay;
23         }
24     }
25 }
26
```

Test the game. Now we're setting the position to the right edge of the screen. No matter where you turn the camera to, the zombies will spawn at the right edge.

How is this working? We'll need a little explanation on viewport space.

Viewport Space

You've heard of local space and world space. The other one is viewport space. And unlike local space and world space, viewport space works largely on 2D.

In viewport space, the X-axis is the left-to-right of the screen, regardless of what the screen is showing. 0.0 means the left edge of the screen, 0.5 means the middle, and 1.0 means the right edge.

The same goes for the Y-axis, Y-axis is the up-to-down of the screen. 0.0 is the bottom edge of the screen, 0.5 is the middle, and 1.0 is the top edge.

The Z-axis on the other hand, is the depth; how far away from the camera.

So with our edgeOfScreen having the values (1.0, 0.5, 8.0), we're saying here: the left edge, raised at the middle, and 8.0 units away from the camera. That's where we want our zombie to spawn.

But Instantiate expects a value in world space, so we need to convert that position from viewport space, into world space. In line 19, ViewportToWorldPoint handles that for us.

Fixing The Spawn Point

You'll notice that with viewport point's the X-axis at 1.0, the zombies still pop-up at the screen. We need to nudge that further away so the player wouldn't see it abruptly appear on the screen. Specifying a value greater than 1.0 will result in a position farther than the right edge of the screen.

Try 1.25 for the value:

```
18                  Vector3 edgeOfScreen = new Vector3(1.25f, 0.5f, 8.0f);
```

You'll find they appear much more subtly now.

Now use this for the Y-axis:

```
18                  Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
```

Remember Random.value returns a number from 0.0 to 1.0. We use that so the spawn point won't be at one position only.

Unfortunately, you may find the zombies dropping down to the ground at times. This is because sometimes, depending on the camera angle, the spawn point would be at the air. We need to make sure we're spawning on the ground.

We need to make a raycast to know if the spot we chose is not empty air:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
```

```
19                Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
20                RaycastHit hit;
21                if (Physics.Raycast(ray, out hit))
22                {
23                    Vector3 placeToSpawn = hit.point;
24                    Quaternion directionToFace = Quaternion.identity;
25                    Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
26                    _nextSpawnTime = Time.time + _spawnDelay;
27                }
28            }
29        }
30 }
31
```

Instead of directly converting the viewport point to a world point, we're now turning it into a raycast. If that raycast hit something (like the ground, for example), the place where that raycast hit will be our spawn point. If it didn't hit anything (if it was in empty air), we don't spawn there.

Now let's randomize between spawning at the left edge or right edge:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     void Update()
15     {
16         if (Time.time >= _nextSpawnTime)
17         {
18             Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
19             if (Random.value > 0.5f)
20             {
21                 edgeOfScreen.x = -0.25f;
22             }
23
24             Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
25             RaycastHit hit;
26             if (Physics.Raycast(ray, out hit))
27             {
28                 Vector3 placeToSpawn = hit.point;
29                 Quaternion directionToFace = Quaternion.identity;
30                 Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
31                 _nextSpawnTime = Time.time + _spawnDelay;
32             }
33         }
34     }
35 }
36
```

Here we give a 50% chance to use the left edge instead of the right edge. We use -0.25 instead of 0.0 so the enemy won't come out abruptly.

Making Sure To Spawn On The Ground Only

Unfortunately, you may find zombies piling on top of one another.

This is because the raycast considers the zombies themselves as a valid spawn point.

We need to make sure we don't spawn on top of other zombies:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     [SerializeField]
07     GameObject _enemyToSpawn;
08
09     [SerializeField]
10     float _spawnDelay = 1.0f;
11
12     float _nextSpawnTime = -1.0f;
13
14     [SerializeField]
15     LayerMask _spawnLayer;
16
17     void Update()
18     {
19         if (Time.time >= _nextSpawnTime)
20         {
```

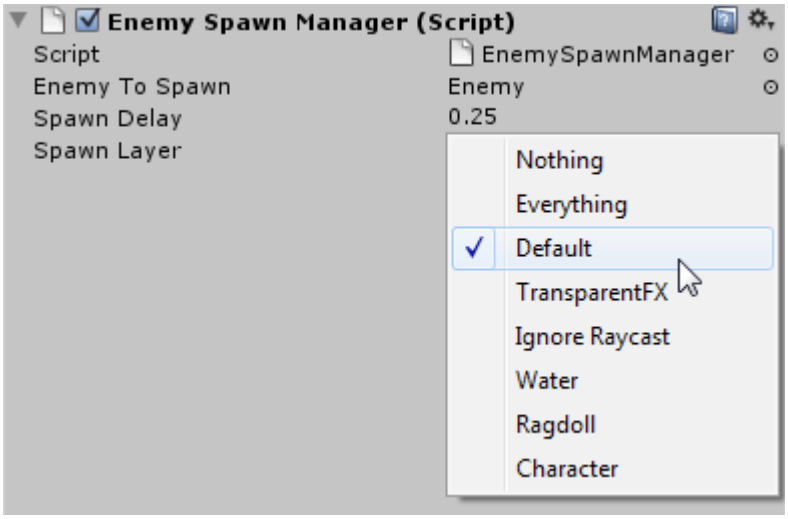
```
21              Vector3 edgeOfScreen = new  Vector3(1.25f, Random.value, 8.0f);
22              if  (Random.value  > 0.5f)
23              {
24                   edgeOfScreen.x = -0.25f;
25              }
26
27              Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
28              RaycastHit hit;
29              if  (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
30              {
31                   Vector3 placeToSpawn = hit.point;
32                   Quaternion directionToFace = Quaternion.identity;
33                   Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
34                   _nextSpawnTime = Time.time + _spawnDelay;
35              }
36         }
37     }
38 }
39
```

We're making use of what is known as a *layer mask* in Unity. We're specifying that we want the raycast to consider certain layers only. The idea is that the ground is on a certain layer, different from the layer that the zombies are using. So if we specify the layer that the ground is using, the raycast will end up considering the ground only.

In line 15 we make a layer mask variable. We'll give it a value in the Inspector later.

In line 29, we make use of that layer mask. Unfortunately with the ordering of arguments for Physics.Raycast, you need to specify a length before you can specify a layer mask. The thing is, we don't want to set a limit to the raycast length. So we just specify here Mathf.Infinity to indicate that.

In your Inspector, you'll see the Spawn Layer property. Right now its set to “Nothing”. Click on that so you can specify which layers you want for the spawn layer.



We'll use “Default” so choose that. If you check, our ground is on the “Default” layer. Meanwhile, our zombies are on the “Character” layer. So choosing “Default” will make the spawning only work on the ground.

Limiting The Spawn

If you notice, after playing a long time, too many zombies are on the screen, and this slows down the framerate very much. We need to put a limit on how many zombies there are.

Illustration 2: Maybe this is too much.

Let's start with a simple rule: There should always be only 30 active zombies. If there are already 30, we wait until the player kills enough zombies before we spawn more.

So with this, we need to keep track of the number of currently active zombies.

Add this code to the EnemySpawnManager script:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06     static int _livingZombies = 0;
07
08     [SerializeField]
09     GameObject _enemyToSpawn;
10
11     [SerializeField]
12     float _spawnDelay = 1.0f;
13
```

```
14      float _nextSpawnTime = -1.0f;
15
16      [SerializeField]
17      LayerMask _spawnLayer;
18
19      void Update()
20      {
21          if (Time.time >= _nextSpawnTime && _livingZombies < 30)
22          {
23              Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
24              if (Random.value > 0.5f)
25              {
26                  edgeOfScreen.x = -0.25f;
27              }
28
29              Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
30              RaycastHit hit;
31              if (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
32              {
33                  Vector3 placeToSpawn = hit.point;
34                  Quaternion directionToFace = Quaternion.identity;
35                  Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
36                  _nextSpawnTime = Time.time + _spawnDelay;
37                  ++_livingZombies;
38              }
39          }
40      }
41 }
42
```

So we're only allowing spawning when our variable hasn't reached 30 yet. But we need to take into account when zombies die.

Add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06      static int _livingZombies = 0;
07      static public void OnEnemyDeath()
08      {
09          --_livingZombies;
10      }
11
12      [SerializeField]
13      GameObject _enemyToSpawn;
14
15      [SerializeField]
16      float _spawnDelay = 1.0f;
17
18      float _nextSpawnTime = -1.0f;
19
20      [SerializeField]
21      LayerMask _spawnLayer;
22
23      void Update()
```

```
24      {
25          if (Time.time >= _nextSpawnTime && _livingZombies < 30)
26          {
27              Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
28              if (Random.value > 0.5f)
29              {
30                  edgeOfScreen.x = -0.25f;
31              }
32
33              Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
34              RaycastHit hit;
35              if (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
36              {
37                  Vector3 placeToSpawn = hit.point;
38                  Quaternion directionToFace = Quaternion.identity;
39                  Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
40                  _nextSpawnTime = Time.time + _spawnDelay;
41                  ++_livingZombies;
42              }
43          }
44      }
45 }
46
```

We'll be calling this OnEnemyDeath whenever an enemy dies. We'll use the Health script since that's where we know when an enemy dies.

Open the Health script and add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06      [SerializeField]
07      int _maximumHealth = 100;
08
09      int _currentHealth = 0;
10
11      override public string ToString()
12      {
13          return _currentHealth + " / " + _maximumHealth;
14      }
15
16      public bool IsDead { get{ return _currentHealth <= 0; } }
17
18      void Start()
19      {
20          _currentHealth = _maximumHealth;
21      }
22
23      public void Damage(int damageValue)
24      {
25          _currentHealth -= damageValue;
26
27          if (_currentHealth < 0)
28          {
29              _currentHealth = 0;
```

```
30          }
31
32          if (_currentHealth == 0)
33          {
34              Animation a = GetComponentInChildren<Animation>();
35              a.Stop();
36
37              if (tag == "Player")
38              {
39                  Destroy(GetComponent<PlayerMovement>());
40                  Destroy(GetComponent<PlayerAnimation>());
41                  Destroy(GetComponent<RifleWeapon>());
42              }
43              else // its an enemy
44              {
45                  EnemySpawnManager.OnEnemyDeath();
46                  Destroy(GetComponent<EnemyMovement>());
47                  Destroy(GetComponentInChildren<EnemyAttack>());
48              }
49
50              Destroy(GetComponent<CharacterController>());
51
52              Ragdoll r = GetComponent<Ragdoll>();
53              if (r != null)
54              {
55                  r.OnDeath();
56              }
57          }
58      }
59 }
60
```

We need to check if its the zombie that's dying here, so we check if the tag is not “Player”, then we know its the enemy.

There we simply call the OnEnemyDeath static function.

While we're at it, let's get rid of the constant 30 value and use a variable. Open the EnemySpawnManager script and add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemySpawnManager : MonoBehaviour
05 {
06      static int _livingZombies = 0;
07      static public void OnEnemyDeath()
08      {
09          --_livingZombies;
10      }
11
12      [SerializeField]
13      GameObject _enemyToSpawn;
14
15      [SerializeField]
16      float _spawnDelay = 1.0f;
17
18      [SerializeField]
```

```
19      int _enemyLimit = 30;
20
21      float _nextSpawnTime = -1.0f;
22
23      [SerializeField]
24      LayerMask _spawnLayer;
25
26      void Update()
27      {
28          if (Time.time >= _nextSpawnTime && _livingZombies < _enemyLimit)
29          {
30              Vector3 edgeOfScreen = new Vector3(1.25f, Random.value, 8.0f);
31              if (Random.value > 0.5f)
32              {
33                  edgeOfScreen.x = -0.25f;
34              }
35
36              Ray ray = Camera.main.ViewportPointToRay(edgeOfScreen);
37              RaycastHit hit;
38              if (Physics.Raycast(ray, out hit, Mathf.Infinity, _spawnLayer.value))
39              {
40                  Vector3 placeToSpawn = hit.point;
41                  Quaternion directionToFace = Quaternion.identity;
42                  Instantiate(_enemyToSpawn, placeToSpawn, directionToFace);
43                  _nextSpawnTime = Time.time + _spawnDelay;
44                  ++_livingZombies;
45              }
46          }
47      }
48 }
49
```

Cleaning Up Corpses

We're limiting the spawning now, but the framerate still suffers when there are lots of 3d models on the scene. We need to remove corpses after they've fallen on the ground.

Add this code to the Health script:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
06     [SerializeField]
07     int _maximumHealth = 100;
08
09     int _currentHealth = 0;
10
11     override public string ToString()
12     {
13         return _currentHealth + " / " + _maximumHealth;
14     }
15
16     public bool IsDead { get{ return _currentHealth <= 0; } }
17
18     void Start()
19     {
20         _currentHealth = _maximumHealth;
21     }
22
23     public void Damage(int damageValue)
24     {
25         _currentHealth -= damageValue;
26
```

```
27          if (_currentHealth < 0)
28          {
29              _currentHealth = 0;
30          }
31
32          if (_currentHealth == 0)
33          {
34              Animation a = GetComponentInChildren<Animation>();
35              a.Stop();
36
37              if (tag == "Player")
38              {
39                  Destroy(GetComponent<PlayerMovement>());
40                  Destroy(GetComponent<PlayerAnimation>());
41                  Destroy(GetComponent<RifleWeapon>());
42              }
43              else // its an enemy
44              {
45                  EnemySpawnManager.OnEnemyDeath();
46                  Destroy(GetComponent<EnemyMovement>());
47                  Destroy(GetComponentInChildren<EnemyAttack>());
48
49                  Destroy(gameObject, 8.0f);
50              }
51
52              Destroy(GetComponent<CharacterController>());
53
54              Ragdoll r = GetComponent<Ragdoll>();
55              if (r != null)
56              {
57                  r.OnDeath();
58              }
59          }
60      }
61 }
62
```

Here we destroy the game object like before, causing the whole game object to dissappear. But we're adding a second argument to Destroy. The second argument acts as a delay in seconds when the Destroy will do its work. So here we delay the deletion of the game object by 8.0 seconds, enough to let ragdoll do its work.

A Different Way To Clean Up Corpses

Simply having the corpses disappear on screen may be too abrupt. We can make it that the deletion only happens when the corpse is not seen in the player's camera.

Make sure to remove the code we previously added, then add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class Health : MonoBehaviour
05 {
```

```
06      [SerializeField]
07      int _maximumHealth = 100;
08
09      int _currentHealth = 0;
10
11      override public string ToString()
12      {
13          return _currentHealth + " / " + _maximumHealth;
14      }
15
16      public bool IsDead { get{ return _currentHealth <= 0; } }
17
18      Renderer _renderer;
19
20      void Start()
21      {
22          _renderer = GetComponentInChildren<Renderer>();
23          _currentHealth = _maximumHealth;
24      }
25
26      public void Damage(int damageValue)
27      {
28          _currentHealth -= damageValue;
29
30          if (_currentHealth < 0)
31          {
32              _currentHealth = 0;
33          }
34
35          if (_currentHealth == 0)
36          {
37              Animation a = GetComponentInChildren<Animation>();
38              a.Stop();
39
40              if (tag == "Player")
41              {
42                  Destroy(GetComponent<PlayerMovement>());
43                  Destroy(GetComponent<PlayerAnimation>());
44                  Destroy(GetComponent<RifleWeapon>());
45              }
46              else // its an enemy
47              {
48                  EnemySpawnManager.OnEnemyDeath();
49                  Destroy(GetComponent<EnemyMovement>());
50                  Destroy(GetComponentInChildren<EnemyAttack>());
51              }
52
53              Destroy(GetComponent<CharacterController>());
54
55              Ragdoll r = GetComponent<Ragdoll>();
56              if (r != null)
57              {
58                  r.OnDeath();
59              }
60          }
61      }
62
```

```
63      void Update()
64      {
65          if (IsDead && !_renderer.isVisible)
66          {
67              Destroy(gameObject);
68          }
69      }
70 }
71
```

Its not so easy to check if an object is currently being seen by the camera or not. The component that knows that is the Renderer component, so in line 18 we make a variable that will hold that.

We need to get a handle to our Renderer component so in line 22 that's what we do. Renderers take care of displaying 3d meshes on screen. If you search through our Enemy game object, there's a game object named “zombie_lowres” that has a Skinned Mesh Renderer component. That's what we're getting here.

Finally, we make an Update function so we can continually check every frame if the renderer is being seen by the camera or not. If not and if its dead, then we delete it.

In Conclusion...

Our zombie game is really taking up shape, we have a zombie horde out to get our player now. You've learned about viewport space, layer masks, and checking whether an object is currently being seen or not.

In the next lesson, we'll introduce the player's goal: the conditions on how he wins the game.