Game Programming Using Unity 3D

Lesson 11: Enemy Damaging

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 10, you need to do it before you can continue here.

We can damage the zombies now, but the zombies aren't fighting back. So we'll add code to let the zombies damage the player when close enough.

Displaying Health

We need to be able to tell whether we're getting damaged. So we'll be displaying the player's health on screen.

First we need the Health script to be able to output its values.

Add this 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     void Start()
17     {
18         _currentHealth = _maximumHealth;
19     }
20
21     public void Damage(int damageValue)
22     {
23         _currentHealth -= damageValue;
24
25         if (_currentHealth <= 0)
26         {
27             Animation a = GetComponentInChildren<Animation>();
28             a.Stop();
29
30             Destroy(GetComponent<PlayerMovement>());
31             Destroy(GetComponent<PlayerAnimation>());
32
33             Destroy(GetComponent<EnemyMovement>());
34             Destroy(GetComponent<CharacterController>());
35
36             Ragdoll r = GetComponent<Ragdoll>();
```

```
37                if (r != null)
38                {
39                    r.OnDeath();
40                }
41            }
42        }
43 }
44
```

Here we provide an implementation of the ToString function. We just return the current and maximum health.

We'll be displaying that string on the screen, so we'll be editing the Player GUI. Add this code to the PlayerGui script:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _crosshair;
08
09     Health _playerHealth;
10
11     void Start()
12     {
13         _playerHealth = GetComponent<Health>();
14     }
15
16     void OnGUI()
17     {
18         float x = (Screen.width - _crosshair.width) / 2;
19         float y = (Screen.height - _crosshair.height) / 2;
20         GUI.DrawTexture(new Rect(x, y, _crosshair.width, _crosshair.height), _crosshair);
21     }
22 }
23
```

Then we make use of that ToString function:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class PlayerGui : MonoBehaviour
05 {
06     [SerializeField]
07     Texture2D _crosshair;
08
09     Health _playerHealth;
10
11     void Start()
12     {
13         _playerHealth = GetComponent<Health>();
14     }
15
16     void OnGUI()
17     {
```

```
18          GUI.Label(new  Rect(5,5,100,100), "Health: "  + _playerHealth.ToString());
19
20          float  x = (Screen.width - _crosshair.width) / 2;
21          float  y = (Screen.height - _crosshair.height) / 2;
22          GUI.DrawTexture(new  Rect(x, y, _crosshair.width, _crosshair.height), _crosshair);
23      }
24 }
25
```

In this instance, its understood we want the ToString function, so we can omit that:

```
18          GUI.Label(new  Rect(5,5,100,100), "Health: "  + _playerHealth);
```

Run the game now and you should see the player health at the upper left corner.

Making The Enemies Attack

We'll make it that coming close to any zombie will damage the player.

Here's how we'll do it: We'll attach a sphere to the enemy. Every time the player is inside that sphere, he gets damaged.

Go to GameObject > Create Empty.

Rename that new game object as “EnemyAttack”.

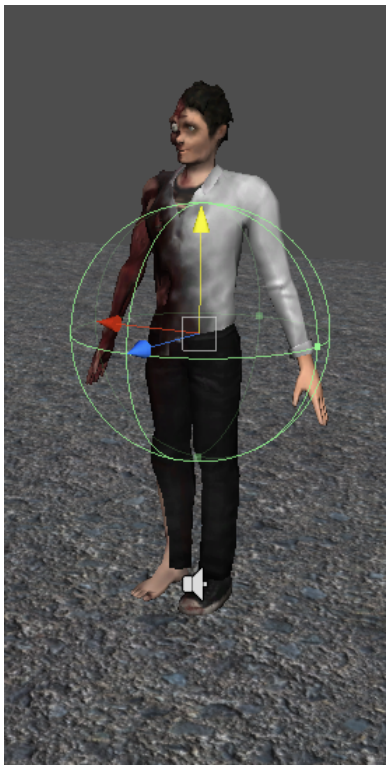
Make sure our EnemyAttack is selected. Go to Component > Physics > Sphere Collider. Our EnemyAttack should have a sphere now.

Parent the EnemyAttack to one of your Enemy game objects.

It will complain about breaking the prefab connection. Go ahead and click “Continue”. Don't worry, we'll reconnect it later.

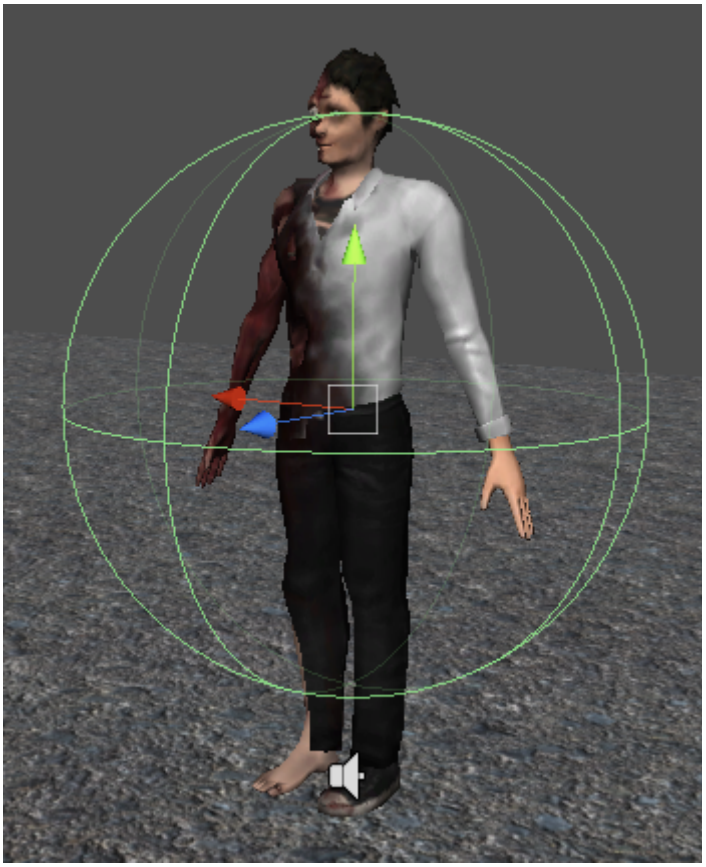
Find the EnemyAttack inside the Enemy game object. Select it. Set its position to (0, 0, 0). Make sure the rotation is (0, 0, 0) and the scale is (1, 1, 1).

Now move the sphere upwards to his pelvis. It doesn't have to be exact.

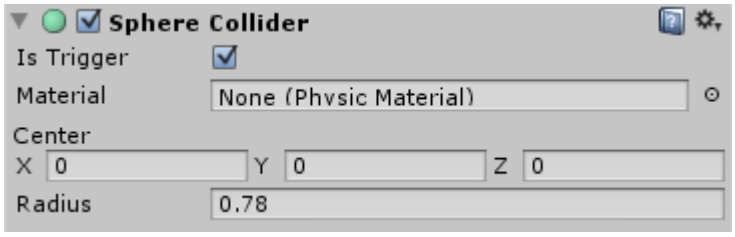


This sphere will act as the zombie's attack range, so it should be large enough to how far the zombie should be able to attack. We'll enlarge the sphere.

In the Inspector, the Sphere Collider should be there. Enter 0.78 in the Radius property.



Also in the Inspector, you'll see a property called “Is Trigger”. Set it as checked. Making the sphere a trigger will allow the player to pass through the sphere, instead of bumping it.



With the EnemyAttack still selected, change the layer to “Ignore Raycast”. We need this so the player shooting code will still work. Player shooting is done with raycasts, and the sphere we have would be shielding the zombie if we didn't set it to “Ignore Raycast”.

Now we'll make code to do the actual damaging.

Create a new C# script. Name it “EnemyAttack”.

Add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     void OnTriggerStay(Collider other)
07     {
08         if (other.tag == "Player")
09         {
10             Debug.Log("we're colliding with player!");
11         }
12     }
13 }
14
```

OnTriggerStay is a function recognized by Unity. It automatically gets called when something is inside our collider (our sphere, in this instance).

So when something is inside the sphere, in line 8, we check if that something is the player. We use the tag property to check it. If you remember, our player has the “Player” tag. This is another handy use for the tag.

For now, we're just testing this and displaying a log message in line 10.

Now attach this EnemyAttack script to the EnemyAttack game object.

Then select your Enemy game object and click “Apply” on the Inspector so all our zombies will have the attack functionality.

Now let's test this. Go ahead and run the game. When you run to a zombie, our log message indicating so should display.

Damaging The Player

Now let's add code to actually damage the player. Add this to the EnemyAttack script:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     void OnTriggerStay(Collider other)
07     {
08         if (other.tag == "Player")
09         {
10             Health playerHealth = other.GetComponent<Health>();
11             playerHealth.Damage(1);
12         }
13     }
14 }
15
```

Test the game now. Let yourself get hurt.

You'll find that the player gets damaged too fast. We'll need to fix that.

Fixing The Fast Damaging

OnTriggerStay gets called continually whenever there is something inside the sphere. The problem is it gets called roughly every frame. So if your framerate is 60 frames per second, the damaging gets called 60 times per second!

Furthermore every PC will have different framerates, so its not good to depend on that.

Our solution is to rely on time instead. In Unity, you can measure the time that has elapsed since the game started. This is what we'll be using.

So let's say we want the zombie to damage by 5 every 1 second. Open the EnemyAttack script:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     float _nextTimeAttackIsAllowed = -1.0f;
07
08     void OnTriggerStay(Collider other)
09     {
10         if (other.tag == "Player" && Time.time >= _nextTimeAttackIsAllowed)
11         {
12             Health playerHealth = other.GetComponent<Health>();
13             playerHealth.Damage(5);
14             _nextTimeAttackIsAllowed = Time.time + 1.0f;
15         }
16     }
17 }
18
```

In line 10, Time.time returns the number of seconds that has elapsed since the game started. We check that with our variable “_nextTimeAttackIsAllowed”, to see if we can attack now.

In line 14, we update the “_nextTimeAttackIsAllowed” variable with the current time, plus 1 second.

This means we're allowing the zombie to attack 1 second from now.

Now let's make use of variables:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class EnemyAttack : MonoBehaviour
05 {
06     float _nextTimeAttackIsAllowed = -1.0f;
07
08     [SerializeField]
09     float _attackDelay = 1.0f;
10
11     [SerializeField]
12     int _damageDealt = 5;
13
14     void OnTriggerStay(Collider other)
15     {
16         if (other.tag == "Player" && Time.time >= _nextTimeAttackIsAllowed)
17         {
18             Health playerHealth = other.GetComponent<Health>();
19             playerHealth.Damage(_damageDealt);
20             _nextTimeAttackIsAllowed = Time.time + _attackDelay;
21         }
22     }
23 }
24
```

Player Death

We've disabled player movement and animation in the previous lesson, but we can actually still shoot. We'll need to fix that.

Open 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     void Start()
17     {
18         _currentHealth = _maximumHealth;
19     }
```

```
20
21      public void Damage(int damageValue)
22      {
23          _currentHealth -= damageValue;
24
25          if (_currentHealth <= 0)
26          {
27              Animation a = GetComponentInChildren<Animation>();
28              a.Stop();
29
30              Destroy(GetComponent<PlayerMovement>());
31              Destroy(GetComponent<PlayerAnimation>());
32              Destroy(GetComponent<RifleWeapon>());
33
34              Destroy(GetComponent<EnemyMovement>());
35
36              Destroy(GetComponent<CharacterController>());
37
38              Ragdoll r = GetComponent<Ragdoll>();
39              if (r != null)
40              {
41                  r.OnDeath();
42              }
43          }
44      }
45 }
46
```

Here we delete the RifleWeapon to disable shooting.

Preventing Negative Health

If you don't want the health display to show negative values, the fix for that is easy:

```
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      void Start()
17      {
18          _currentHealth = _maximumHealth;
19      }
20
21      public void Damage(int damageValue)
```

```
22      {
23          _currentHealth -= damageValue;
24
25          if (_currentHealth < 0)
26          {
27              _currentHealth = 0;
28          }
29
30          if (_currentHealth == 0)
31          {
32              Animation a = GetComponentInChildren<Animation>();
33              a.Stop();
34
35              Destroy(GetComponent<PlayerMovement>());
36              Destroy(GetComponent<PlayerAnimation>());
37              Destroy(GetComponent<RifleWeapon>());
38
39              Destroy(GetComponent<EnemyMovement>());
40
41              Destroy(GetComponent<CharacterController>());
42
43              Ragdoll r = GetComponent<Ragdoll>();
44              if (r != null)
45              {
46                  r.OnDeath();
47              }
48          }
49      }
50 }
51
```

Fixing Enemy Attack When Dead

Enemies shouldn't be able to attack when they've been killed. We need to delete the EnemyAttack 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      void Start()
17      {
18          _currentHealth = _maximumHealth;
```

```
19      }
20
21      public void Damage(int damageValue)
22      {
23          _currentHealth -= damageValue;
24
25          if (_currentHealth < 0)
26          {
27              _currentHealth = 0;
28          }
29
30          if (_currentHealth == 0)
31          {
32              Animation a = GetComponentInChildren<Animation>();
33              a.Stop();
34
35              Destroy(GetComponent<PlayerMovement>());
36              Destroy(GetComponent<PlayerAnimation>());
37              Destroy(GetComponent<RifleWeapon>());
38
39              Destroy(GetComponent<EnemyMovement>());
40              Destroy(GetComponentInChildren<EnemyAttack>());
41
42              Destroy(GetComponent<CharacterController>());
43
44              Ragdoll r = GetComponent<Ragdoll>();
45              if (r != null)
46              {
47                  r.OnDeath();
48              }
49          }
50      }
51  }
52
```

Remember our EnemyAttack is in a child game object, so we use GetComponentInChildren instead of just GetComponent here.

Displaying Game Over

Well the death code works nicely now. We'll need a display message for indication as well.

Our assets folder has a subfolder named Gui. Inside it is a file GameOverScreen.png. We'll be displaying that when the player dies.

In your Project View, select the “TheGame” folder. Create a new folder. Name it “Gui”. Copy the GameOverScreen.png there.

Select the GameOverScreen that was just imported. In your Inspector, it should show the Texture Type set to “Texture”. Change this to “GUI”. Click “Apply”.

Basically what we need to do is check if the player is dead, and if so, display the picture.

We need to create code first that checks if the player is dead.

Go to your Health script and add this:

```
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                Destroy(GetComponent<PlayerMovement>());
38                Destroy(GetComponent<PlayerAnimation>());
39                Destroy(GetComponent<RifleWeapon>());
40
41                Destroy(GetComponent<EnemyMovement>());
42                Destroy(GetComponentInChildren<EnemyAttack>());
43
44                Destroy(GetComponent<CharacterController>());
45
46                Ragdoll r = GetComponent<Ragdoll>();
47                if  (r != null)
48                {
49                     r.OnDeath();
50                }
51           }
52      }
53 }
54
```

IsDead is a C# property, a kind of a cross between a function and a variable. To outside code it looks and acts like a variable, but inside the class definition, its a function.

Now we'll add code that makes use of this.

Create a new C# script. Name it “CombatGui”.

Add this code:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class CombatGui : MonoBehaviour
05 {
06     Health _playerHealth;
07
08     void Start()
09     {
10         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
11         _playerHealth = playerGameObject.GetComponent<Health>();
12     }
13
14     void OnGUI()
15     {
16         if (_playerHealth.IsDead)
17         {
18             // code to display game over image here
19         }
20     }
21 }
22
```

Here we get a handle to the player's health. We do so by getting a handle to the player first, using his tag. Then out of the player game object, we get his health using GetComponent. Then we continually check if the player is dead with the IsDead property.

Now to finally add code to display the game over image:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class CombatGui : MonoBehaviour
05 {
06     Health _playerHealth;
07
08     [SerializeField]
09     Texture2D _gameOverImage;
10
11     void Start()
12     {
13         GameObject playerGameObject = GameObject.FindGameObjectWithTag("Player");
14         _playerHealth = playerGameObject.GetComponent<Health>();
15     }
16
17     void OnGUI()
18     {
19         if (_playerHealth.IsDead)
20         {
21             GUI.DrawTexture(new Rect(0, 0, _gameOverImage.width, _gameOverImage.height),
22                             _gameOverImage);
23         }
24     }
```

```
25 }
26
```

You've seen code like this before in the PlayerGui when we draw a crosshair on the screen.

Let's test this code. Create a new empty game object. Name it “CombatGui”. Attach the CombatGui script to it. Then assign the GameOverScreen image to the “Game Over Image” property in the Inspector.

Right now, we're displaying the image at X, Y coordinates (0, 0) which is the top left corner of the screen. But just like the crosshair, we can center this image on the screen:

```
                                                    .
                                                    .
                                                    .
17      void  OnGUI()
18      {
19          if  (_playerHealth.IsDead)
20          {
21              float  x = (Screen.width - _gameOverImage.width) / 2;
22              float  y = (Screen.height - _gameOverImage.height) / 2;
23              GUI.DrawTexture(new  Rect(x, y, _gameOverImage.width, _gameOverImage.height),
24                                      _gameOverImage);
25          }
26      }
27 }
28
```

This is what you should end up with:

In Conclusion...

You've learned about quite a few things just to be able to make enemies attack and damage the player. You've learned about trigger spheres and how to use Time.time.

Next we'll make code that will automatically create more enemies on the screen.