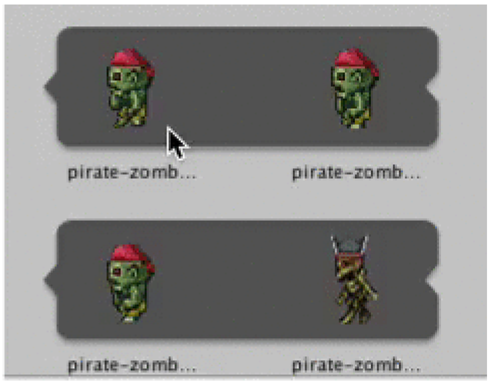
Unity of the Dead part 2

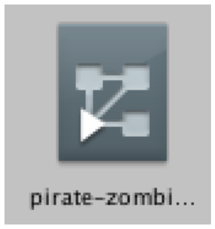
1. Enemy animations

In the Assets > Sprites folder, select the Enemies image and whilst selecting 3 frames at a time, drag the left, up and right frames of the pirate zombie into the scene to create the animations for that direction.

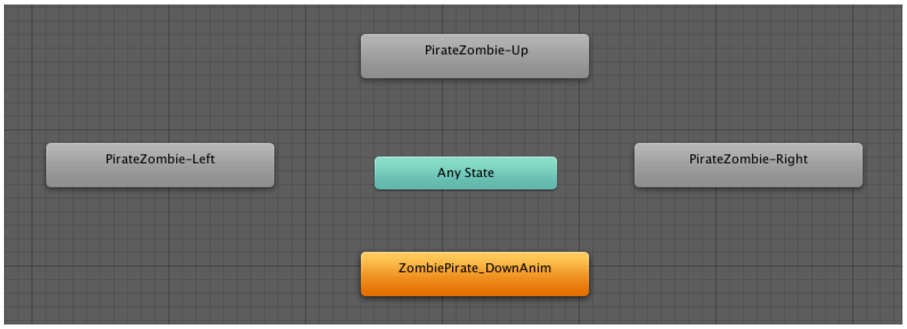


You can delete the animations from the Hierarchy as we will be adding them to the animator we are currently using.

In the Assets > Animations folder double click the Pirate-Zombie animator



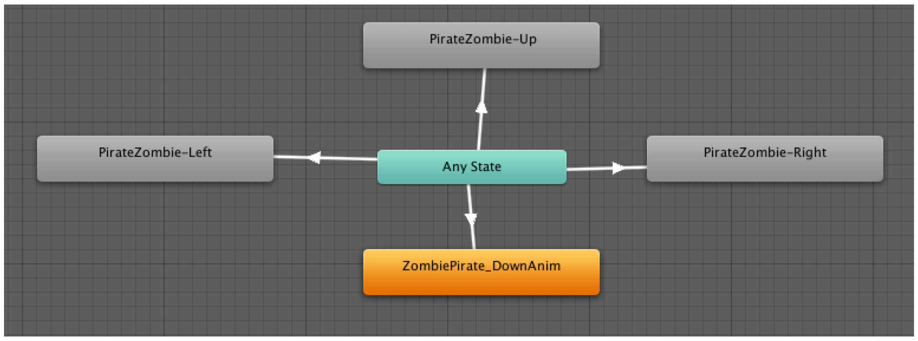
Then in the animator drag the 3 animations you just created into the view and arrange them as follows:



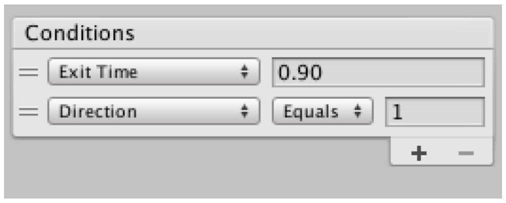
In the bottom left of the window click the + button next to Parameters and add a new Int called

‘Direction’ with default value 0.

In the animator right click ‘Any State’ and choose to ‘Make Transition’ and then click on any of the other boxes to create an arrow ‘Transition’ between the two. Do this four times so that all of the animations are linked to any state.



Now select each of these arrows in turn and in the Inspector add a new condition by clicking the + button below the Exit Time condition. For the parameter choose Direction and keep the condition set to ‘Equals’ but set the value to be 0 for the Up transition, 1 for Right, 2 for Down and 3 for Left.



Conditions

Exit Time	0.90	
Direction	Equals	1

The animator states are set up now but we need to control them from a script. The best place to do this is the Follow Target Script. Firstly lets create some constant variables to represent each of the values we just put in the transitions. So at the top of the script add the following:

```
private const int UP = 0;

private const int RIGHT = 1;

private const int DOWN = 2;

private const int LEFT = 3;
```

This way we can refer to the animation we want to show by name instead of number, which will make the code easier to read.

Next up let’s add a function that sets which animation should be played. So at the bottom of the script add the following:

```
void SetDirection(int direction)

{

        GetComponent<Animator>().SetInteger("Direction",direction);

}
```

This will change the enemys animation to whichever gets passed to it.

Finally we need to determine which animation needs to be played based on the location of the enemy and the player. As the animation needs to be updated constantly it's best to perform this check in the Update function, so firstly append the following to the Update function after the MoveTowards code:

```
Vector2 distance = transform.position - target.transform.position;

if(Mathf.Abs(distance.x) > Mathf.Abs(distance.y))

{

    SetDirection((distance.x > 0) ? LEFT : RIGHT);

}

else

{

    SetDirection((distance.y > 0) ? DOWN : UP);

}
```

The first thing this code does is work out the distance between both characters positions. It then checks to see which is greater the y distance or the x distance. We don't have animations for diagonals so we need to pick the closest vertically or horizontally which is why we compare these values. Once that is known, we check to see which direction of that axis the enemy should face. The code has been shortened for readability but the line of code:

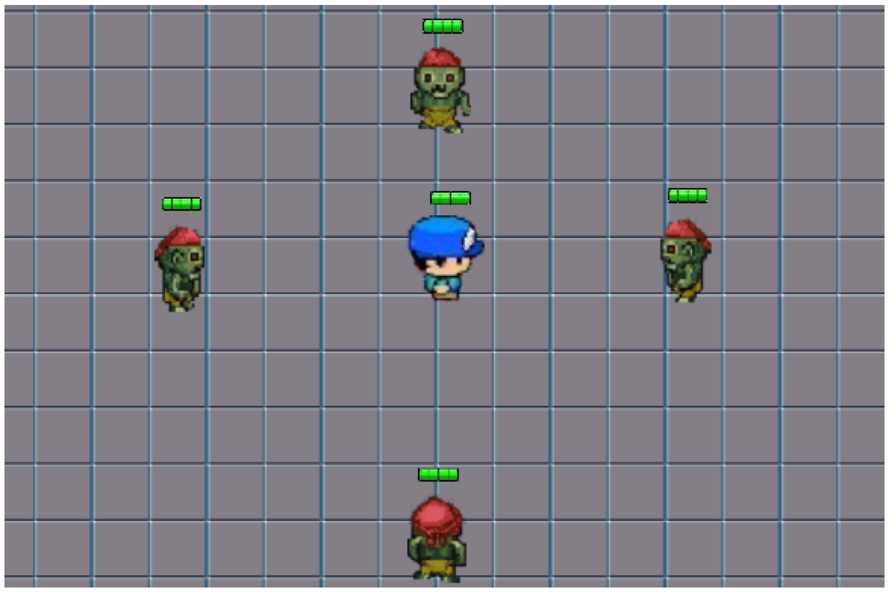
```
SetDirection((distance.x > 0) ? LEFT : RIGHT);
```

Is exactly the same as the longer block of code:

```
if(distance.x > 0)
```

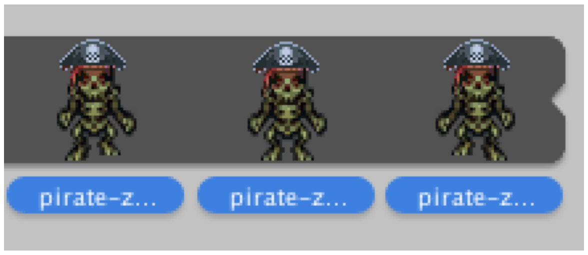
```
            {
                    SetDirection(LEFT);
            }
            else
            {
                    SetDirection(RIGHT);
            }
```

Now if you test the scene you will see that all of the enemies will face the player as they pursue him.



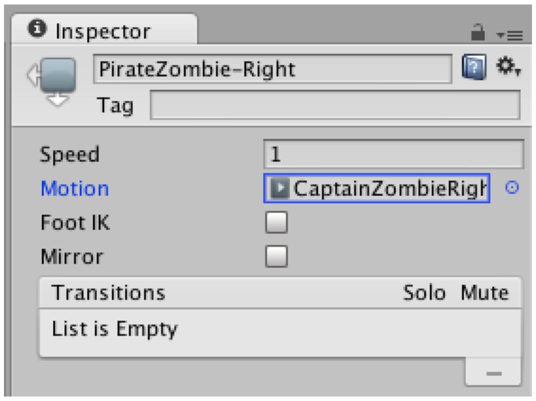
2. Multiple enemies

Ok it’s time to create another enemy so that we have some variety in the game. Start by creating 4 more directional animations for the Zombie Pirate Captain in the Assets > Sprites > Enemies image.



Next up duplicate the first enemies Animator by selecting it under Assets > Animations and hitting Cmd + D to create a copy. Rename the copy so that it's something like ZombieCaptainAnimator and then double click to open it up.

In the Animator select each of the 4 boxes and in the Inspector change the motion value to the corresponding animation for the new enemy e.g. switch 'PirateZombieUp' for 'CaptainZombieUp'.



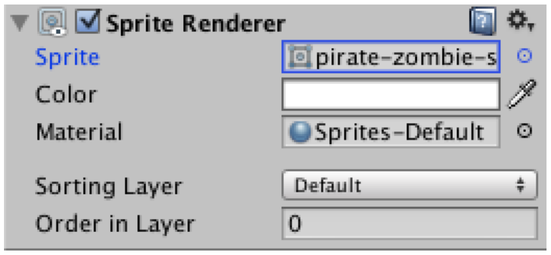
PirateZombie-Right

Speed 1
Motion CaptainZombieRigh
Foot IK
Mirror

List is Empty

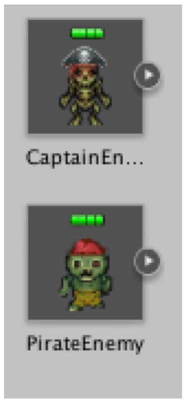
The animations for the new enemy are ready to go but we need to create a new prefab that refers to these animations. Select the Enemy prefab from the Assets > Prefabs folder and use Cmd + D to duplicate it and rename it to something like CaptainEnemy prefab.

With the prefab selected, look at the Inspector and change the Controller value of the Animator to point to the CaptainEnemyAnimator. You will notice that the icon of the prefab still shows the old enemy. This is because the icon image is actually controlled by the Sprite Renderer component. When a game object has both a sprite renderer and an animation component attached to it the animation component takes precedence at runtime and we won't see the default image attached to the Sprite Renderer. However to make it easier to distinguish whilst the game isn't running we should change the sprite value of the sprite renderer to point to one of the captain sprites so it's easier to recognise.



Sprite Renderer

Sprite	pirate-zombie-s
Color	
Material	Sprites-Default
Sorting Layer	Default
Order in Layer	0



To test the new enemy, select the EnemyController object in the Hierarchy and change the Enemy Prefab value of the spawn script to point to the new enemy.

If you test the scene you will see that the new enemy is running around however it behaves exactly like the other enemy which is a bit dull so let’s tweak it’s components so that it behaves differently.

Let’s make the new enemy more powerful than the first one to give more challenge.

With the Captain Prefab selected change the following components values:

Set the Destroyable components toughness to ‘2’ so it takes more shots to kill.

Set the Damaging Colliders damage to ‘0.03’ so the enemy can kill the player faster.

Set the Follow Targets Follow speed to ‘0.01’ so it moves slower and gives the player a fighting chance.

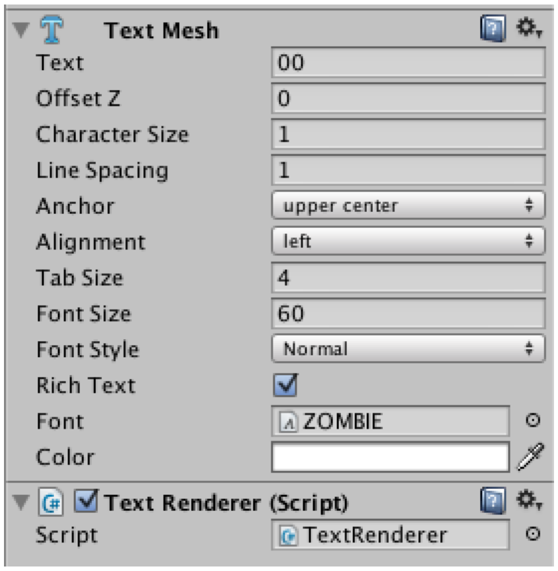
When you are happy with your new enemy change it back to the original one as we will add the harder one in later on.

3. Level system

Create a new Scene and call it Start Screen. To this scene create a new Empty Game Object and rename it to “LevelController”. To this object add the script ‘’LevelController’ to it. This script has a public array that takes a list of level names which the game will play through in order. For now set the length to 2 and set the values to “Level1” and “Level2”.

The LevelController script calls a special Unity function ‘DontDestroyOnLoad(gameObject)’, this ensures that when a different scene gets loaded the gameobject and any components attached to it do not get destroyed. The script also has a function to load the next available level which we need to call from the level itself.

Open the original scene and rename it to “Level1”. In this scene create another Empty Game Object and name it ‘Level Timer’. Now create a 3D Text Mesh (Menu Bar > Game Object > Create Other > 3D Text) and nest the text under the Level Timer object by dragging it onto it. This text mesh is going to be used to display the remaining time for the level. Add the ‘Text Renderer’ script to the text object so that the text will render in front of the sprites, then setup the values of the Text Mesh as follows:



Text Mesh

Text	00
Offset Z	0
Character Size	1
Line Spacing	1
Anchor	upper center
Alignment	left
Tab Size	4
Font Size	60
Font Style	Normal
Rich Text	✓
Font	ZOMBIE
Color	⬜

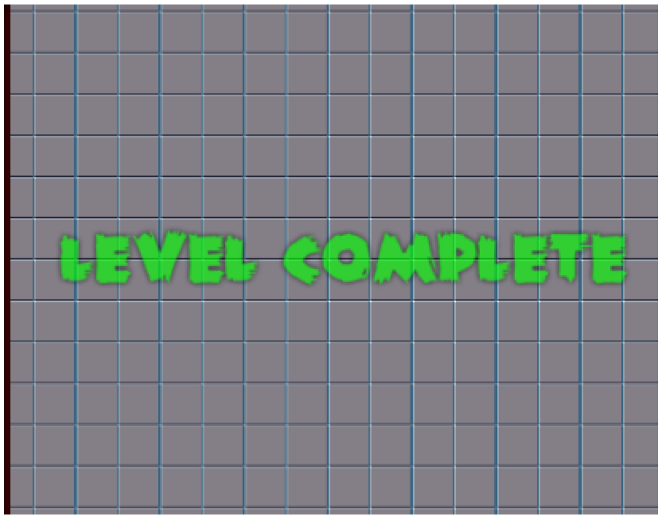
Text Renderer (Script)

Script	TextRenderer

Now select the Level Timer object and add to it the script that is also named ‘Level Timer’. The script takes 2 values: ‘Time Limit’ is how long the level will last in seconds. The Text Mesh value needs to have the new text object assigned to it.

If you look at the script you will see that it tries to load a scene called ‘CompleteScene’ that does not exist yet so we need to create this first. I just copied the original level and deleted everything except for the camera and background from it.

If you look in the Assets > Sprites folder you will find a ‘LevelCompleteText’ image, drag this into the scene, scale and position it so it’s quite central and add the CompletedLevelScript to it. This script waits a set number of frames and then asks the LevelController to load the next level. This allows for a little pause to catch your breath between levels.



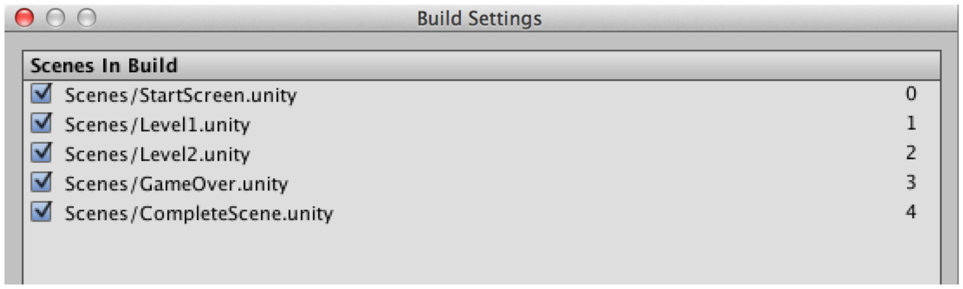
One issue we have when the level changes is that when a new scene is loaded the old scene and all of it’s components get destroyed. Currently when the player gets destroyed the GameOverOnDestroyed script tries to load the GameOver scene. So when we complete the level the game will jump straight to the GameOver screen which is not what we want. To fix this, modify the OnDestroy function of that script so that it looks like this:

```
void OnDestroy(){

        if(!_isShuttingDown && !Application.isLoadingLevel)

        {

            Application.LoadLevel("GameOver");

        }

    }
```

Re-open the StartScene and you will see that the counter now counts down towards zero. When the counter hits zero the level complete scene is loaded and then a few seconds later the game will try to load the next level.

In the Level Controller we added a ‘Level2’ which does not exist yet so Duplicate the Level1 scene, rename it and add it to the build settings. You may want to change some of the features of this level so it behaves differently, such as changing the time limit and the values of the EnemySpawnScript attached to the EnemyController object.

Make sure to add all of the scenes to the Build Settings (File > Build Settings) and add each scene by dragging it in.


Build Settings

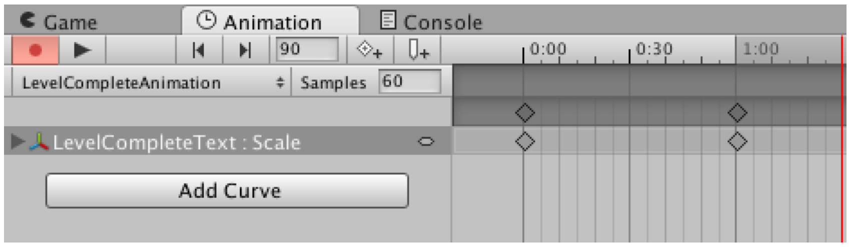
Scenes In Build

☑ Scenes/StartScreen.unity	0
☑ Scenes/Level1.unity	1
☑ Scenes/Level2.unity	2
☑ Scenes/GameOver.unity	3
☑ Scenes/CompleteScene.unity	4

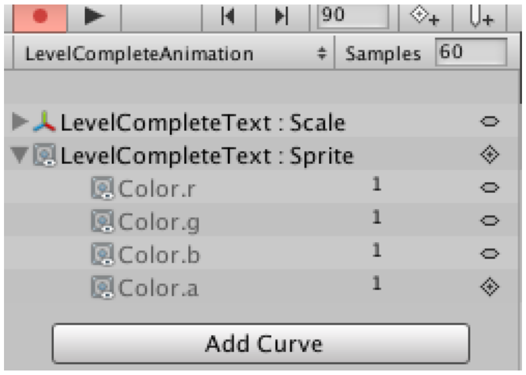
Now when you play the game from the StartScreen you should be able to play through Level 1 and then Level 2 in order!

4. In built animation

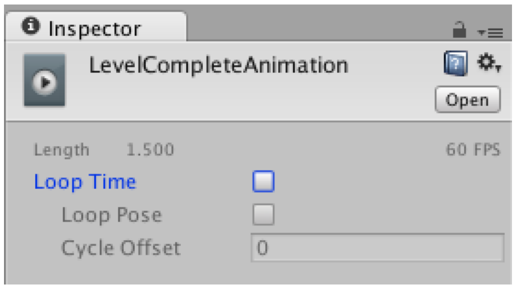
Let's do some animation inside Unity itself. Go back to the CompleteScene and select the text animation in the Hierarchy, then open the Animation view (Window > Animation). Firstly click the record button and you will be prompted to save the animation, so just call it LevelCompleteAnimation. Then click the 'Add Curve' button and choose Transform > Scale (you will need to expand the transform options by clicking the arrow to the left of it). You will see that 2 sets of frames appear on the timeline at 0 and 1.



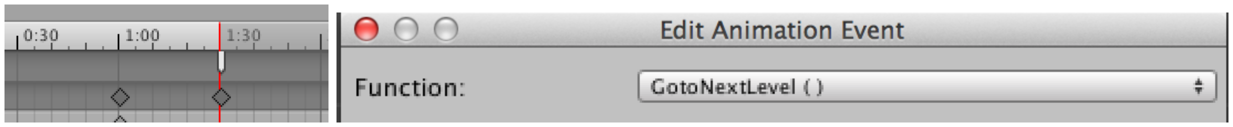
Select the frame at 1 and change the scale of the object so that its twice as big then reselect the first frame and click the Play button to see a preview of the text scaling smoothly between the frames. To give the animation a bit of bounce, right click on the timeline at 1.30 and choose 'Add Key', at this point scale the object back to it's original size. Now click the Add Curve button again and choose Sprite Renderer > Color. If you then expand the LevelCompleteText : Sprite entry in the Animation window you will see that it has 4 sub properties for colour r, g, b, a.



We only want to change the ‘a’ or alpha value to make the text fade into the scene. so select the keyframe for a at 0 and change the value from 1 to 0. Now add a new keyframe at 1 and set the alpha value to be 1 here. Now if you test the animation you will see that the sprite fades in and bounces at the end. You will notice that the animation loops continuously, to prevent this select the animation in the Assets folder and in the Inspector uncheck the ‘Loop Time’ value.

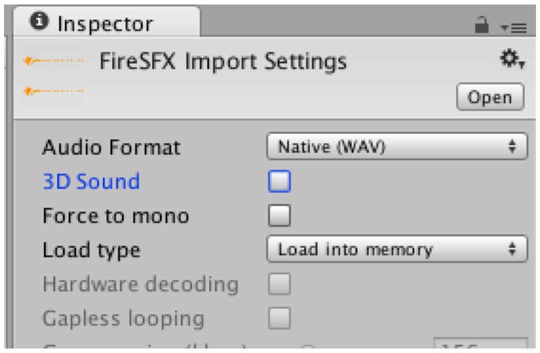


Currently the next level gets loaded after the countdown completes but it would be nicer to do this when the animation completes instead. One way to do this is to add an animation event to the animation. With the Text sprite selected and the animation view open right click in the dark grey section just under the 1.30 value on the time bar. You should see an option to ‘Add Animation Event, Choose this and you will be presented with a dropdown of any public function that is also attached to this game object. Choose GotoNextLevel from the dropdown. Now when the animation finishes the level will load in sync.



5. Sfx

Let’s add some sounds to the game. First up select each of the sounds in the Assets > Sounds folder and in the Inspector, make sure that 3D Sound is unchecked and click Apply. this will ensure that the sounds plays at the correct volume for a 2D game.


Inspector

FireSFX Import Settings

Open

Audio Format: Native (WAV)
3D Sound: ☐
Force to mono: ☐
Load type: Load into memory
Hardware decoding: ☐
Gapless looping: ☐

Let's start with the main music of the game the BGM. With the StartScreen open, drag the GameBGM sound onto the LevelController object.

To the player object add an Audio Source component and modify the FireScript so that it has the following variables:

```
public AudioClip fireSFX;

private AudioSource _audioSource;
```

Modify the Start() function so that it stores a reference to the AudioSource component;

```
_audioSource =  GetComponent<AudioSource>();
```

And Modify the FireBullet function so that it calls the following line:

```
_audioSource.PlayOneShot(fireSFX);
```

Now whenever you fire a bullet the 'fire' sound will play.

Next lets add some zombie sfx. To each of the enemy prefabs add the GroanSFX and set the Loop value to true.

Acknowledgements:

All assets are used under the Creative Commons licence.

Fire SFX   http://soundbible.com/1780-Bow-Fire-Arrow.html

Groan SFX   http://soundbible.com/1059-Mummy-Zombie.html

BGM http://www.nosoapradio.us/   DST-CyberOps.mp3

Zombie sprites: http://opengameart.org/content/pirate-zombie-and-skeleton-32x48

Background tiles : http://opengameart.org/content/sci-fi-platformer-tiles-32x32