NAME: DHAVAL THANKI

COURSE: COMP 4478

PROJECT 2 – AMBUSH IN UNITY

LAKEHEAD UNIVERSITY

STUDENT NUMBER: 0871347

Table of Contents

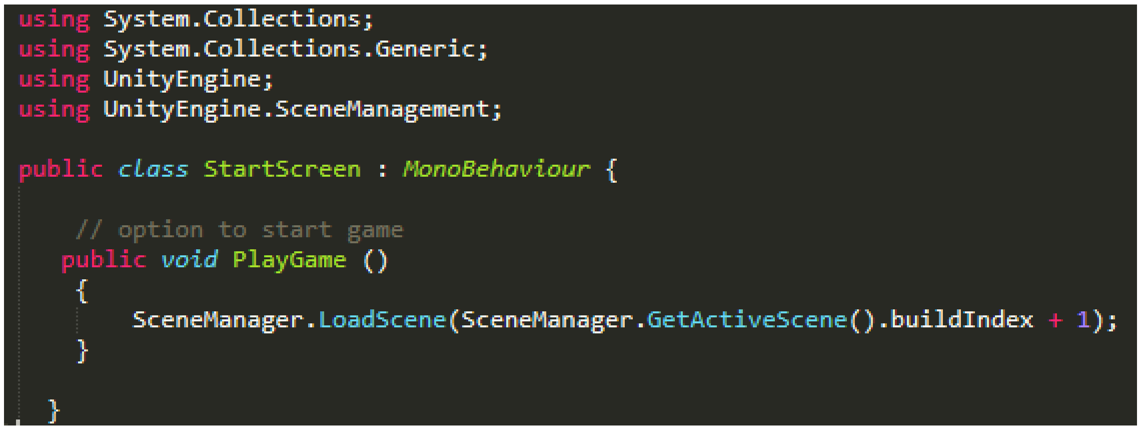
Objective

The objective of this project was to create a 2D game in unity, where the concept and idea was open to individual creativity, to create the character, environment and objectives as desired. For my game, I chose to implement a top-down RPG game titled Ambush. The game comprises of three levels, with the last level being the 'boss level', and the first two involving the main character to walk around pathways, eliminating enemies (such as crabs and dragons) and collecting life potions on the way. The player is granted five lives, represented through four heart icons at the beginning of the game, with the possibility to gain one more through life potions.

Game Script Elements

StartScreen.cs

Using the Scene manager, I load in the first scene from the main start screen/scene using the class "StartScreen".



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class StartScreen : MonoBehaviour {

    // option to start game
   public void PlayGame ()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

  }
```

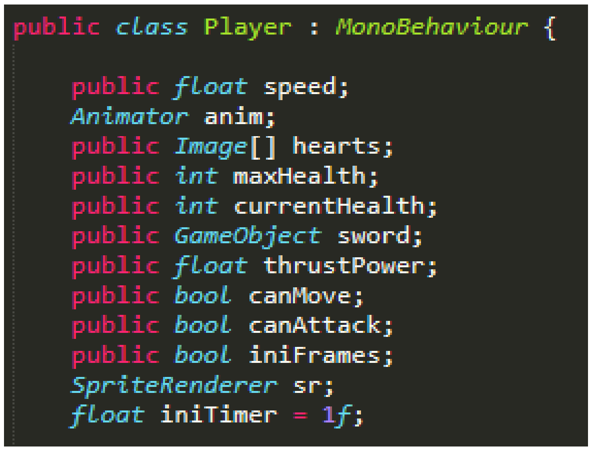
Player.cs

The Player script was developed to comprise of many variables, to allow for the player to carry out multiple features.

speed and *anim* – control the animated movements as well as the speed of the player

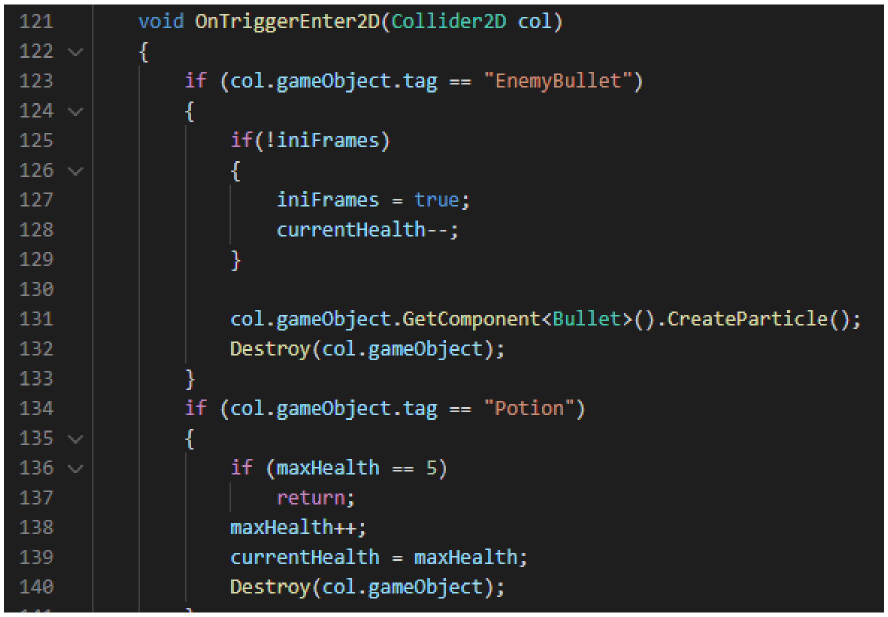
hearts, maxHealth and *currentHealth* – indicate the five lives of the player, including its current health/lives till it's maximum capacity

sword, thrustPower, canMove and *canAttack* - indicates the sword object and it's attributes, such as it's speed (thrust power), ability to move and attack (affect health level of enemies)



```
public class Player : MonoBehaviour {

    public float speed;
    Animator anim;
    public Image[] hearts;
    public int maxHealth;
    public int currentHealth;
    public GameObject sword;
    public float thrustPower;
    public bool canMove;
    public bool canAttack;
    public bool iniFrames;
    SpriteRenderer sr;
    float iniTimer = 1f;
```

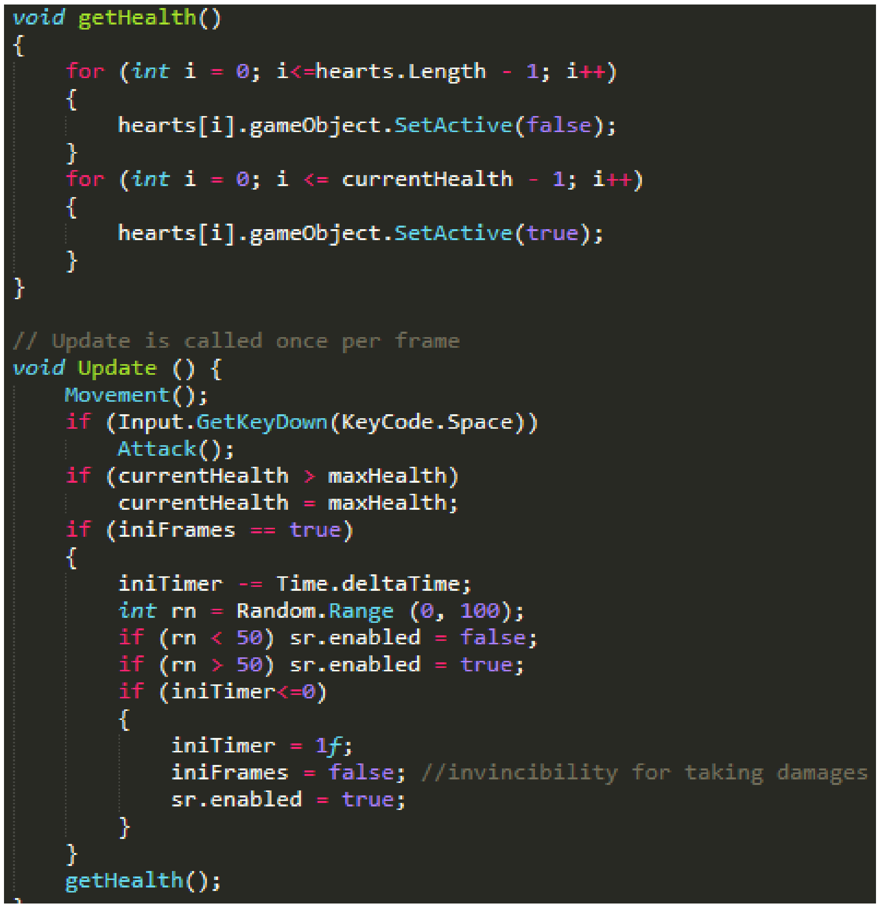


```
void OnTriggerEnter2D(Collider2D col)
{
    if (col.gameObject.tag == "EnemyBullet")
    {
        if(!iniFrames)
        {
            iniFrames = true;
            currentHealth--;
        }

        col.gameObject.GetComponent<Bullet>().CreateParticle();
        Destroy(col.gameObject);
    }
    if (col.gameObject.tag == "Potion")
    {
        if (maxHealth == 5)
            return;
        maxHealth++;
        currentHealth = maxHealth;
        Destroy(col.gameObject);
```

The attached code demonstrates how the player will lose its health through the trigger "EnemyBullet", and how it can regain its health via the "Potion" trigger.

currentHealth--; and *maxHealth++;* respectively indicate the method with which the player's health alters.



```
void getHealth()
{
    for (int i = 0; i<=hearts.Length - 1; i++)
    {
        hearts[i].gameObject.SetActive(false);
    }
    for (int i = 0; i <= currentHealth - 1; i++)
    {
        hearts[i].gameObject.SetActive(true);
    }
}

// Update is called once per frame
void Update () {
    Movement();
    if (Input.GetKeyDown(KeyCode.Space))
        Attack();
    if (currentHealth > maxHealth)
        currentHealth = maxHealth;
    if (iniFrames == true)
    {
        iniTimer -= Time.deltaTime;
        int rn = Random.Range (0, 100);
        if (rn < 50) sr.enabled = false;
        if (rn > 50) sr.enabled = true;
        if (iniTimer<=0)
        {
            iniTimer = 1f;
            iniFrames = false; //invincibility for taking damages
            sr.enabled = true;
        }
    }
    getHealth();
```

The code aside represents how the player will accumulate health and how the number of maximum lives/hearts they may have at any given time is capped.

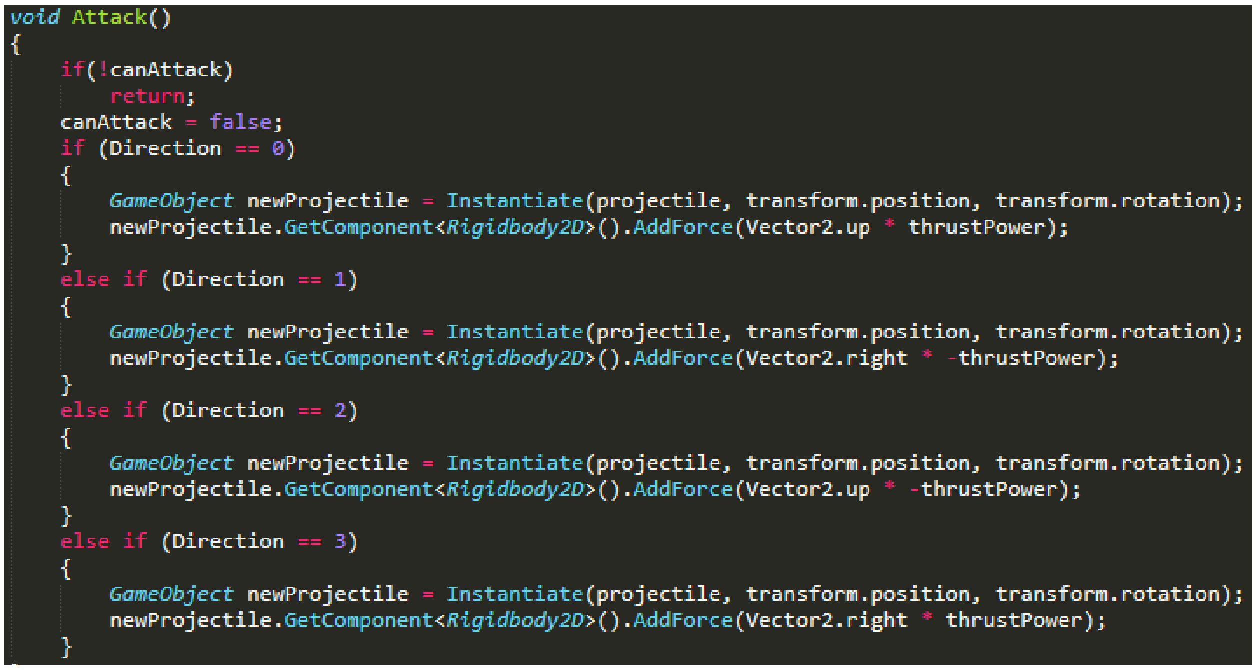
The *Update* function is used to send swords flying to the enemies. This is done through the Space key, using the command *KeyCode.*

The code also keeps the current health value updated (when required), and therefore the ability to throw the sword as the player reaches full health. With lower than five lives, the player is only limited to reaching the sword out to a much shorter distance.

Enemies Attack (Crabs, Dragon, Wizard)

Below is a screenshot that shows the methodology on how the enemy object will attack the player. Just as the player's attack, the enemies attack is mapped according to its direction.

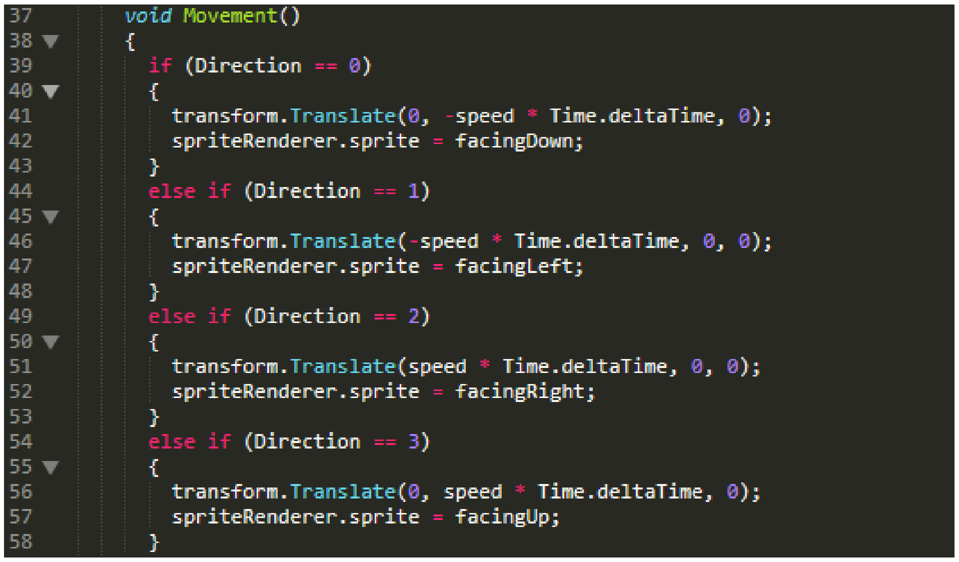
The code also demonstrates instantiation of a projectile that will be destroyed in X amount of time or if the player is hit, whichever comes first.



```
void Attack()
{
    if(!canAttack)
        return;
    canAttack = false;
    if (Direction == 0)
    {
        GameObject newProjectile = Instantiate(projectile, transform.position, transform.rotation);
        newProjectile.GetComponent<Rigidbody2D>().AddForce(Vector2.up * thrustPower);
    }
    else if (Direction == 1)
    {
        GameObject newProjectile = Instantiate(projectile, transform.position, transform.rotation);
        newProjectile.GetComponent<Rigidbody2D>().AddForce(Vector2.right * -thrustPower);
    }
    else if (Direction == 2)
    {
        GameObject newProjectile = Instantiate(projectile, transform.position, transform.rotation);
        newProjectile.GetComponent<Rigidbody2D>().AddForce(Vector2.up * -thrustPower);
    }
    else if (Direction == 3)
    {
        GameObject newProjectile = Instantiate(projectile, transform.position, transform.rotation);
        newProjectile.GetComponent<Rigidbody2D>().AddForce(Vector2.right * thrustPower);
    }
```

Crab.cs

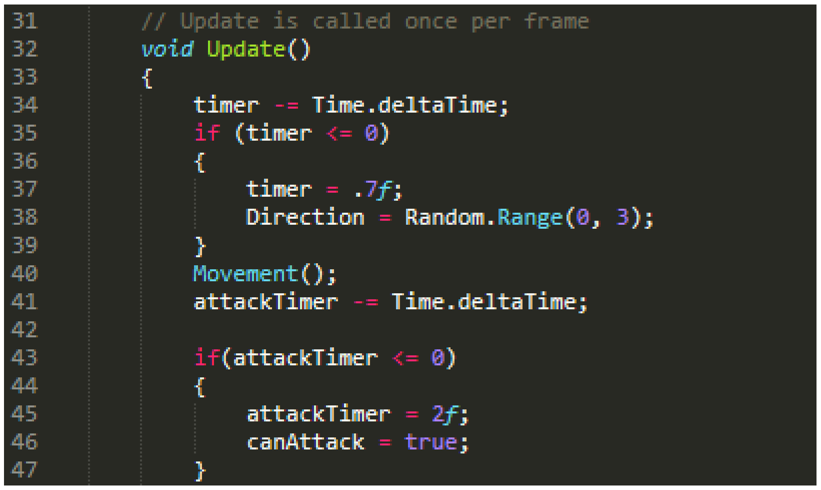
The code below represents methodology for the enemy's movement, with the animated sprites (*facingDown*, *facingLeft, facingRight* and *facingUp*) added in to align with its movement directions.



```
    void Movement()
    {
        if (Direction == 0)
        {
            transform.Translate(0, -speed * Time.deltaTime, 0);
            spriteRenderer.sprite = facingDown;
        }
        else if (Direction == 1)
        {
            transform.Translate(-speed * Time.deltaTime, 0, 0);
            spriteRenderer.sprite = facingLeft;
        }
        else if (Direction == 2)
        {
            transform.Translate(speed * Time.deltaTime, 0, 0);
            spriteRenderer.sprite = facingRight;
        }
        else if (Direction == 3)
        {
            transform.Translate(0, speed * Time.deltaTime, 0);
            spriteRenderer.sprite = facingUp;
        }
```

Dragon.cs

The *Update* function represents the time taken for movement and attack speed.



```
    // Update is called once per frame
    void Update()
    {
        timer -= Time.deltaTime;
        if (timer <= 0)
        {
            timer = .7f;
            Direction = Random.Range(0, 3);
        }
        Movement();
        attackTimer -= Time.deltaTime;

        if(attackTimer <= 0)
        {
            attackTimer = 2f;
            canAttack = true;
        }
```

Wizard.cs

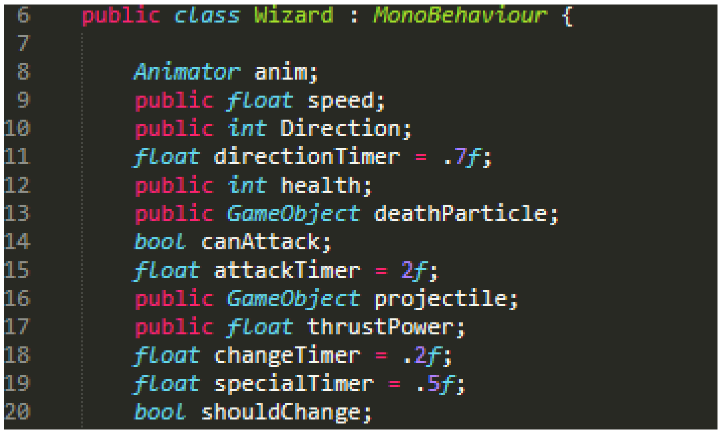
The Wizard script comprises of many variables, to allow for the wizard to carry out multiple features through the level.

Speed, *Direction* and *directionTimer* – controls the speed and direction of the wizard, determined by a timer.

health – indicates the lives of the wizard

canAttack, attackTimer, and *thrustPower* - indicates the bullet object and it's attributes, such as it's speed (thrust power), ability to move and attack (affect health level of Player) which is determined by a set timer.

deathParticle and *projectile* – represents the bullet projectile shot by the wizard and the death particle's cloud-like effect.

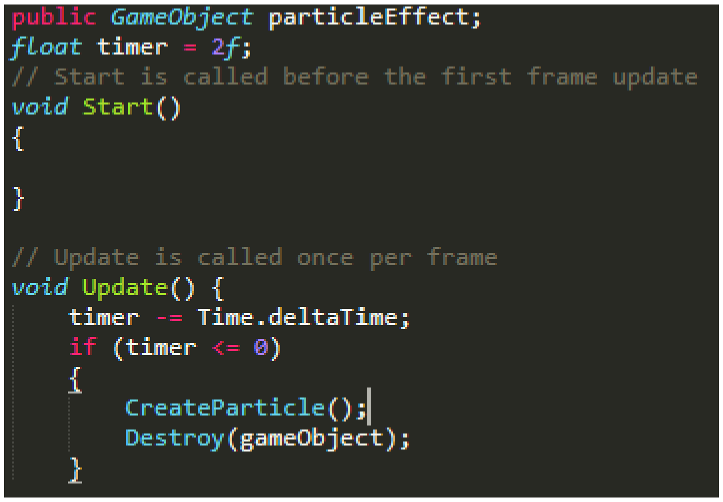


```
public class Wizard : MonoBehaviour {

    Animator anim;
    public float speed;
    public int Direction;
    float directionTimer = .7f;
    public int health;
    public GameObject deathParticle;
    bool canAttack;
    float attackTimer = 2f;
    public GameObject projectile;
    public float thrustPower;
    float changeTimer = .2f;
    float specialTimer = .5f;
    bool shouldChange;
```

Bullet.cs

The *Update* function is used to create particles when the bullet (sword) times out during an attack, or when it hits an enemy. When this happens, a cloud-like effect is created, enhancing the overall animation of the game.



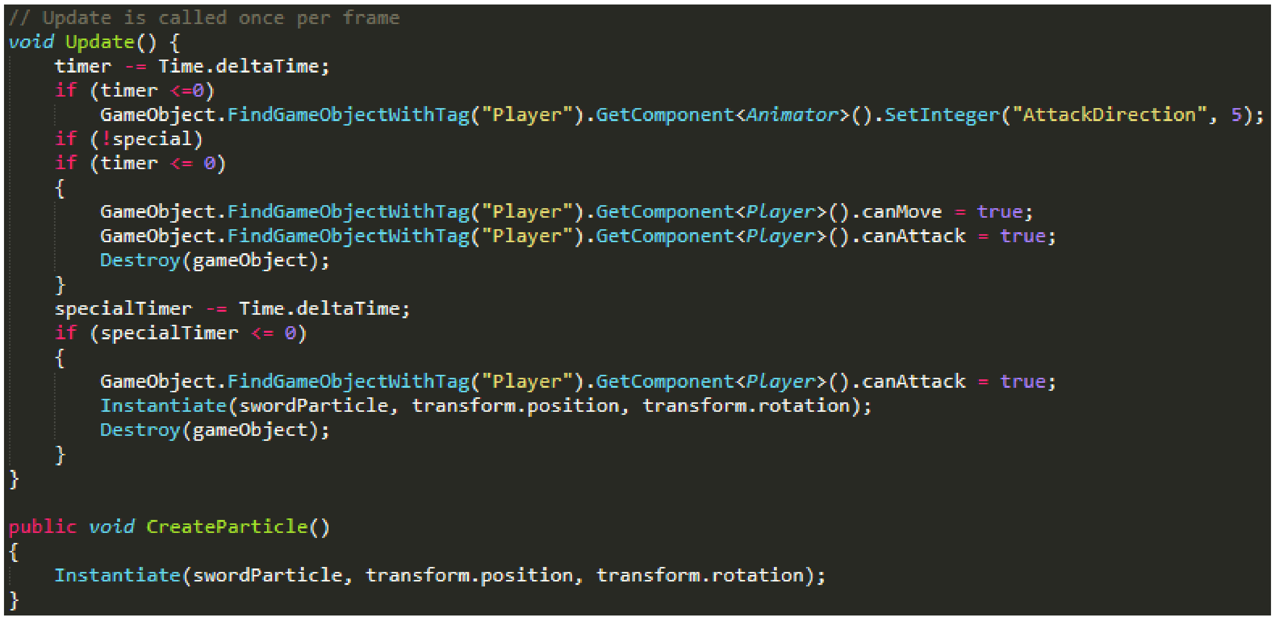
```
public GameObject particleEffect;
float timer = 2f;
// Start is called before the first frame update
void Start()
{

}

// Update is called once per frame
void Update() {
    timer -= Time.deltaTime;
    if (timer <= 0)
    {
        CreateParticle();
        Destroy(gameObject);
    }
```

Sword.cs

Within the sword class, we can determine whether the player has the ability to use the distance-throwing attack instead of the melee attack. This is determined by whether or not the player has maximum health. In the *Update* portion of the script, we will keep refreshing to see if this has changed when the player loses or gains a life.

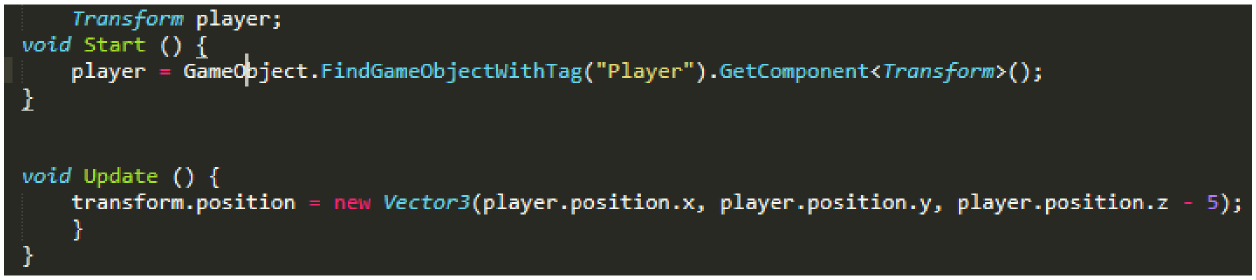


```
// Update is called once per frame
void Update() {
    timer -= Time.deltaTime;
    if (timer <=0)
        GameObject.FindGameObjectWithTag("Player").GetComponent<Animator>().SetInteger("AttackDirection", 5);
    if (!special)
    if (timer <= 0)
    {
        GameObject.FindGameObjectWithTag("Player").GetComponent<Player>().canMove = true;
        GameObject.FindGameObjectWithTag("Player").GetComponent<Player>().canAttack = true;
        Destroy(gameObject);
    }
    specialTimer -= Time.deltaTime;
    if (specialTimer <= 0)
    {
        GameObject.FindGameObjectWithTag("Player").GetComponent<Player>().canAttack = true;
        Instantiate(swordParticle, transform.position, transform.rotation);
        Destroy(gameObject);
    }
}

public void CreateParticle()
{
    Instantiate(swordParticle, transform.position, transform.rotation);
}
```

CamFollow.cs

This is used to let us dynamically follow the player around so that we can have a much more immersive experience.



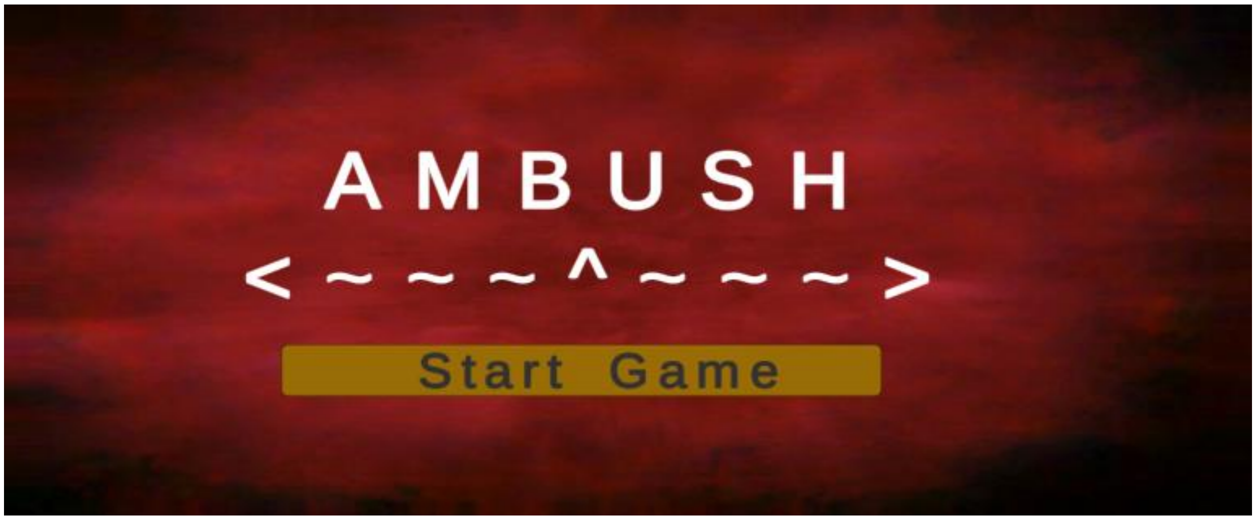
```
    Transform player;
void Start () {
    player = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
}


void Update () {
    transform.position = new Vector3(player.position.x, player.position.y, player.position.z - 5);
    }
}
```

GUI – Graphical User Interface

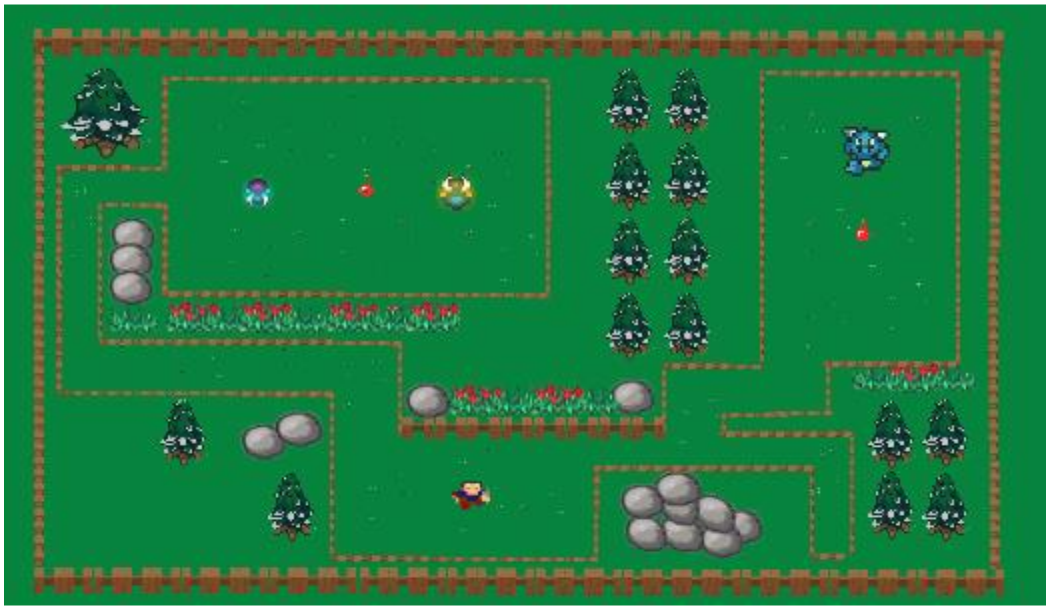
StartScreen.cs

The start button is given an additional attribute of “Highlighted Color”, where when hovered on, changes to a lighter yellow shade.



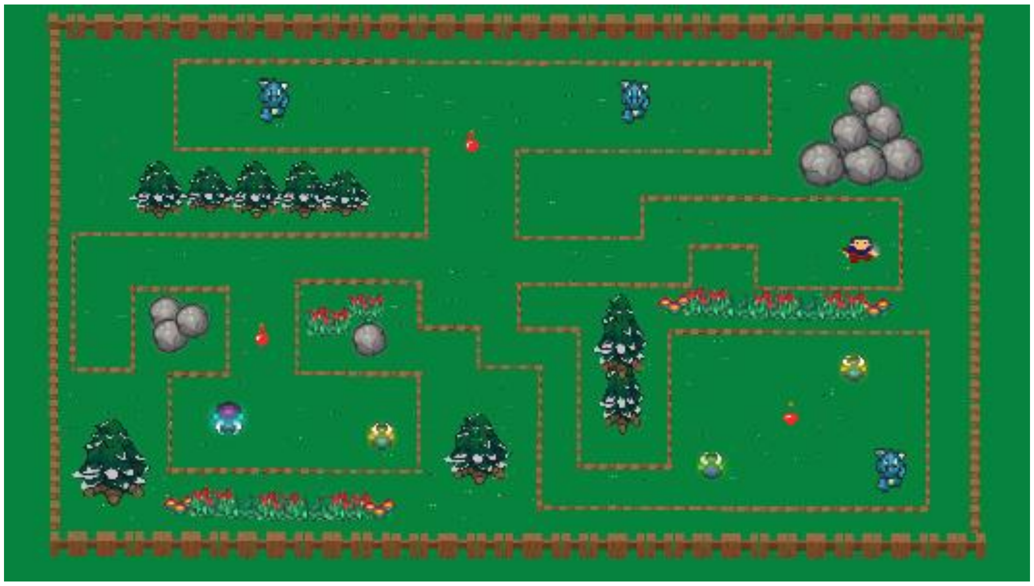
LevelOne.cs – Overall view

Player’s general graphical environment, set to a nature-like theme.



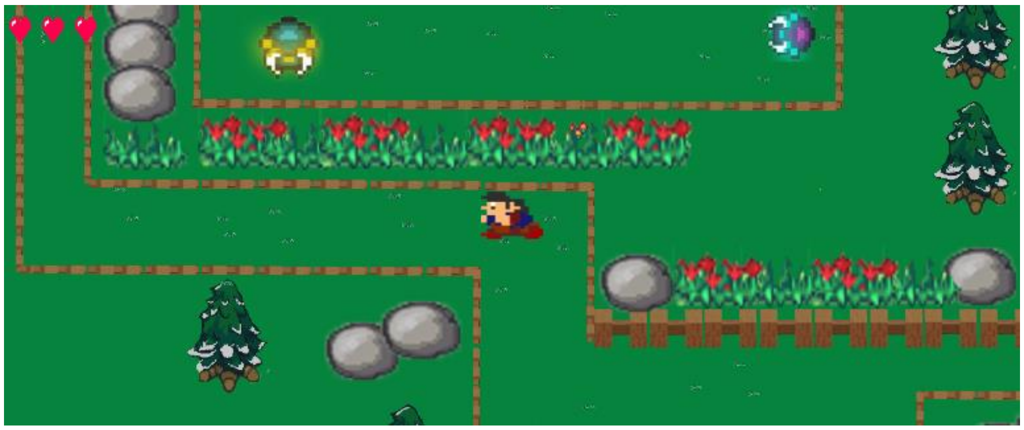
LevelTwo.cs – Overall view

Player’s general graphical environment, set to a nature-like theme.



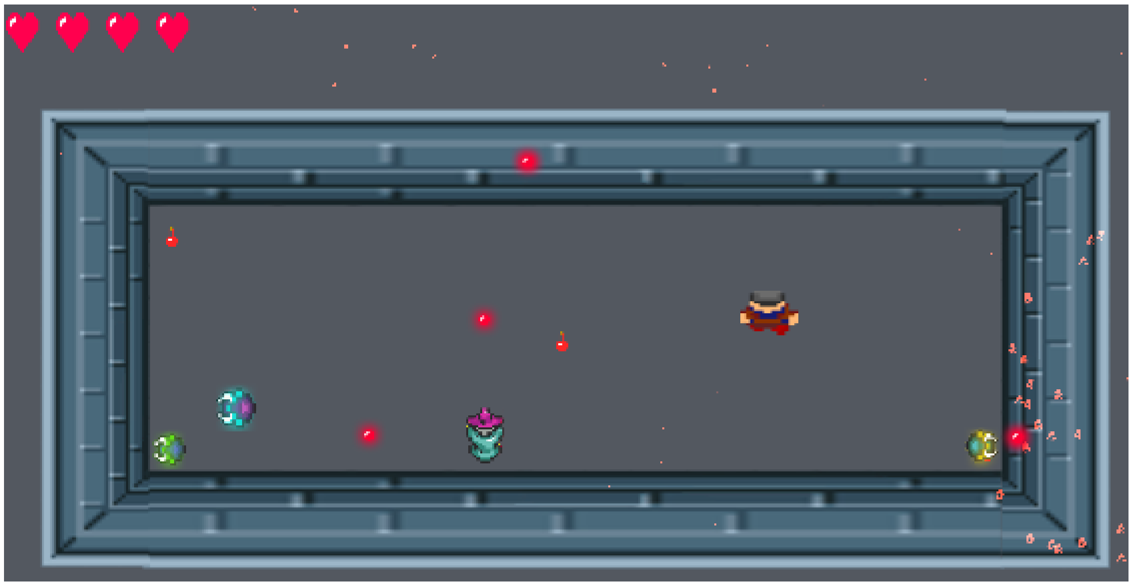
Player POV

Player's perspective during game play, set by a 'Perspective' projection for the Main Camera.

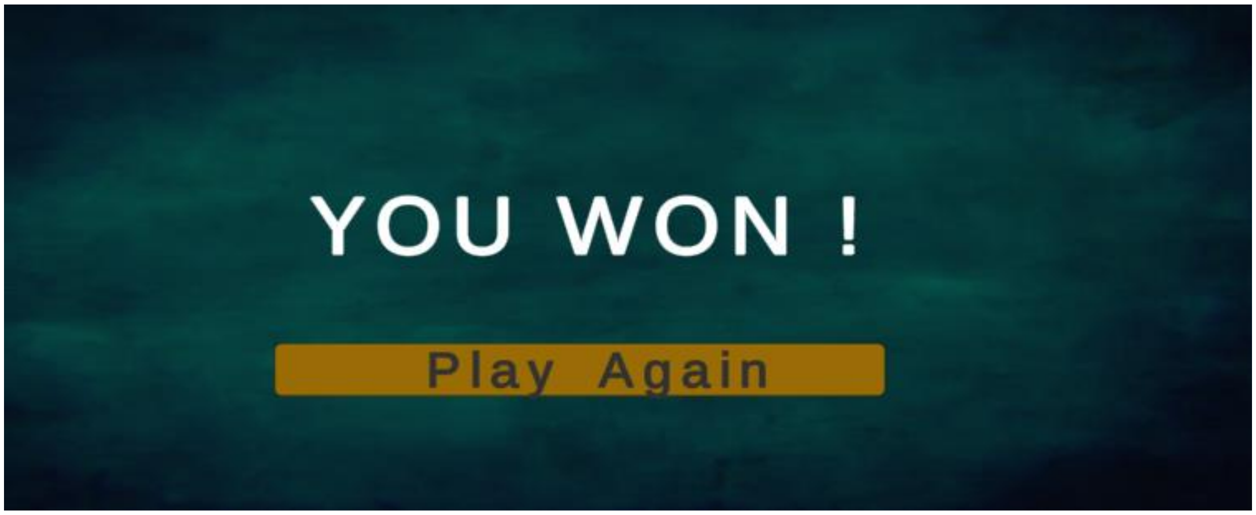


Boss level.cs

The final, boss level is set in a dungeon, with a Wizard enemy with the highest attack/defense capacity. A few previous enemies are added in to increase difficulty for the final level.



EndScreen.cs



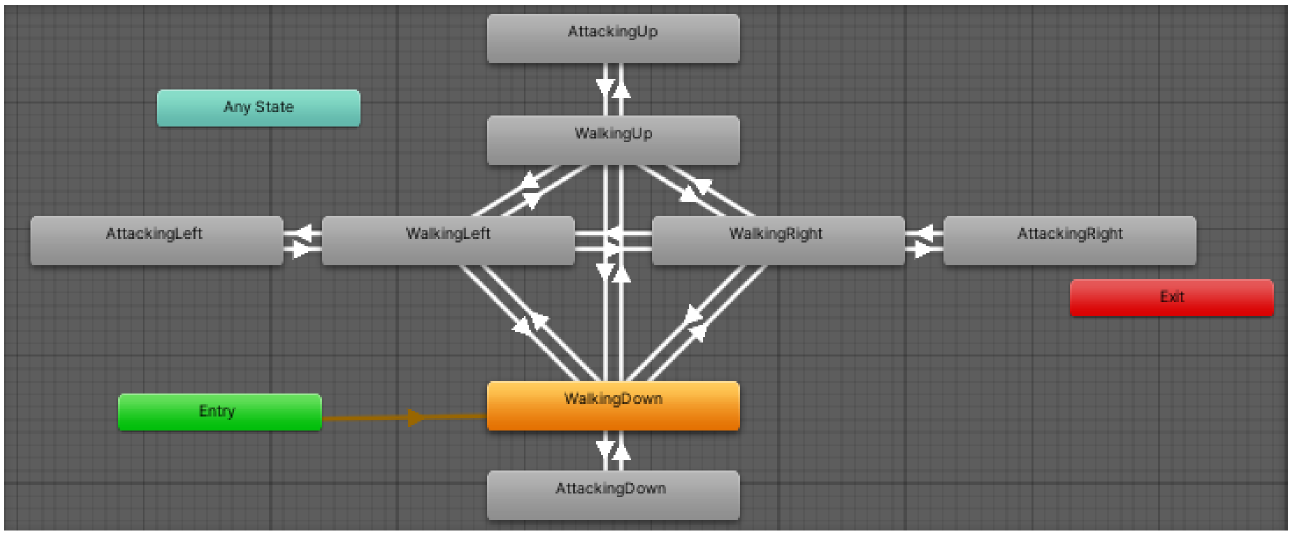
Animation

Player Walking and Attacking Animation

The animation below has two parameters; Direction (set to 0) and AttackDirection (set to 5).

For the Walking transitions, the *Exit Time* and *Fixed Duration* is unselected, with the *Transition Duration* set to 0 and the Conditions set to equal to 0.

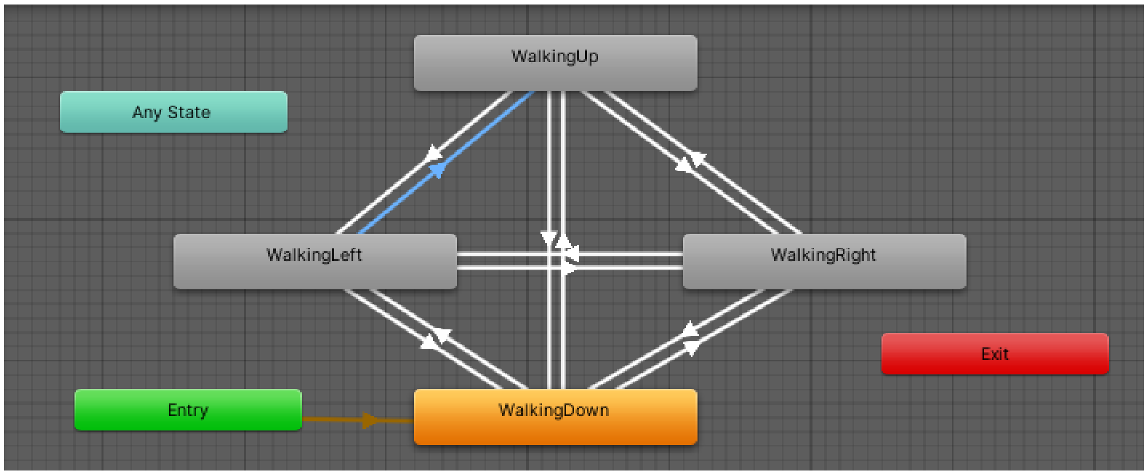
For the Attacking transitions, the *Exit Time* and *Fixed Duration* is selected, with the *Exit Time* set to 0.15 and no Conditions set.



Dragon & Wizard; Walking and Attacking Animation

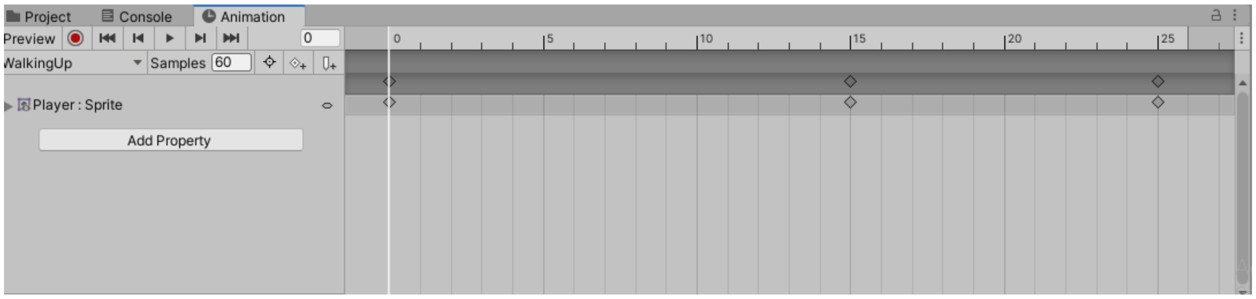
The animation below has one parameter; Direction (set to 0).

For the Walking transitions, the *Exit Time* and *Fixed Duration* is unselected, with the *Transition Duration* and the Conditions ranging from 0-3.



General Animation Frames Layout

The animation below is set to mark the sprites movement of the characters. A difference of 15 and 10 seconds sets the three sprites in motion, to create a semi-realistic character movement. The same is repeated for Walking as well as Attacking motions (in all four directions).



Resources

Start scene - https://www.youtube.com/watch?v=zc8ac_qUXQY&t=584s

Sprites and other resources - Sourced from an online course/Udemy tutorial.