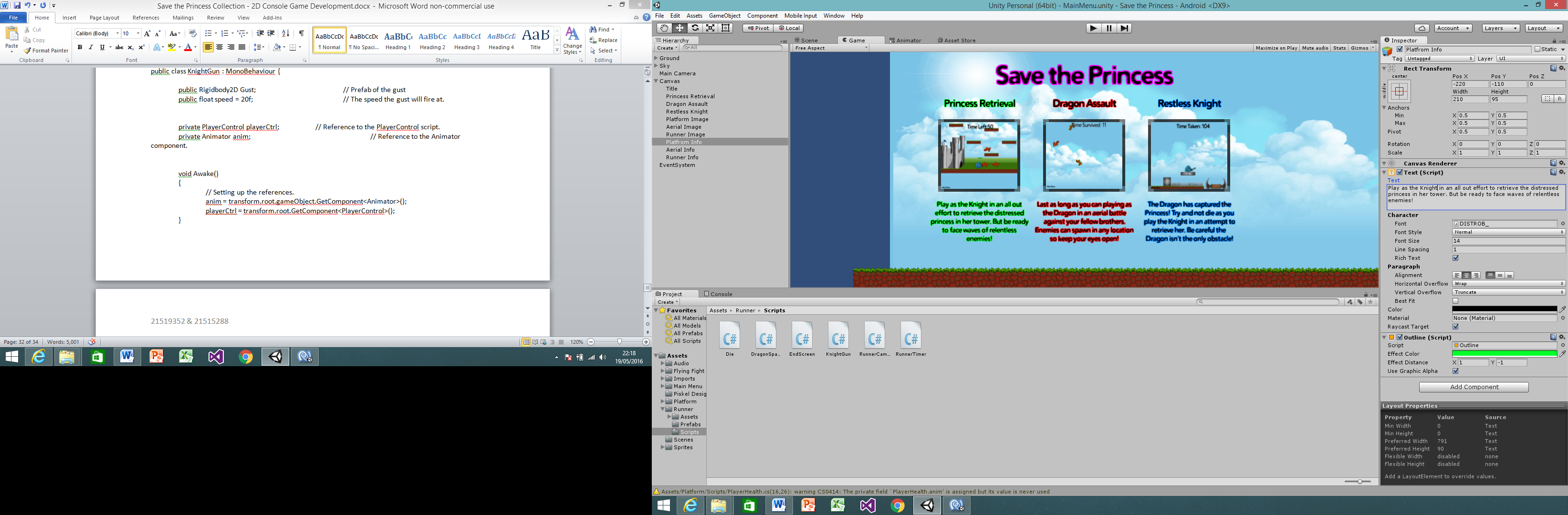
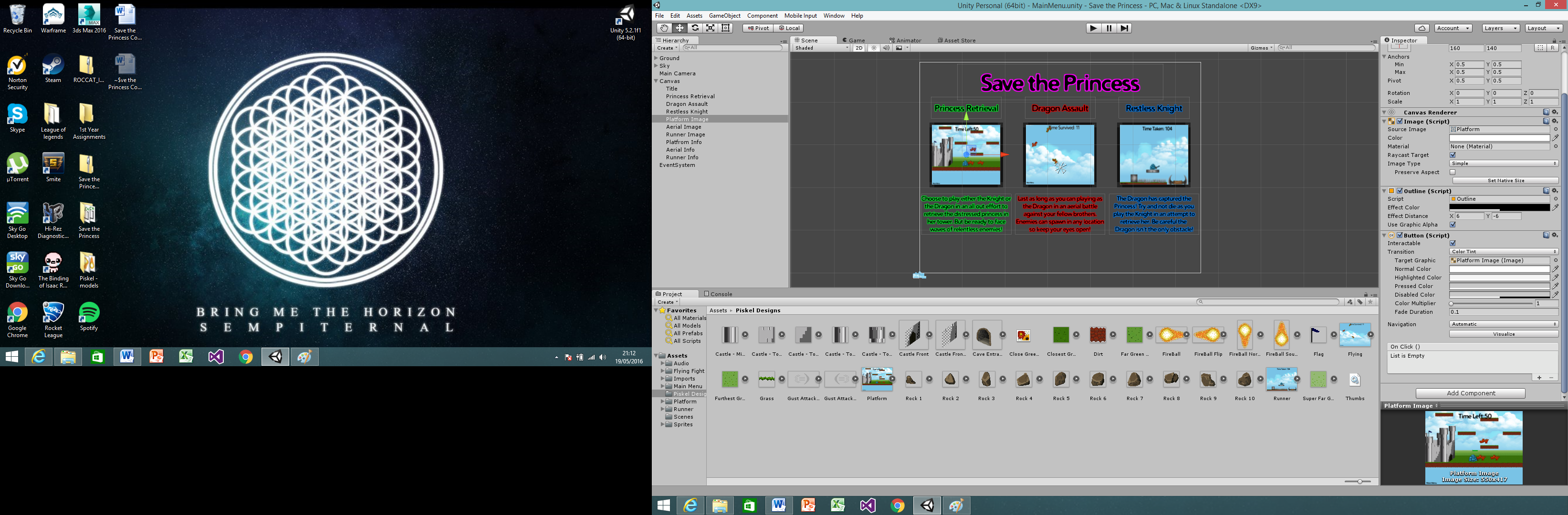
**Save the Princess Collection – 2D Console Game Development**

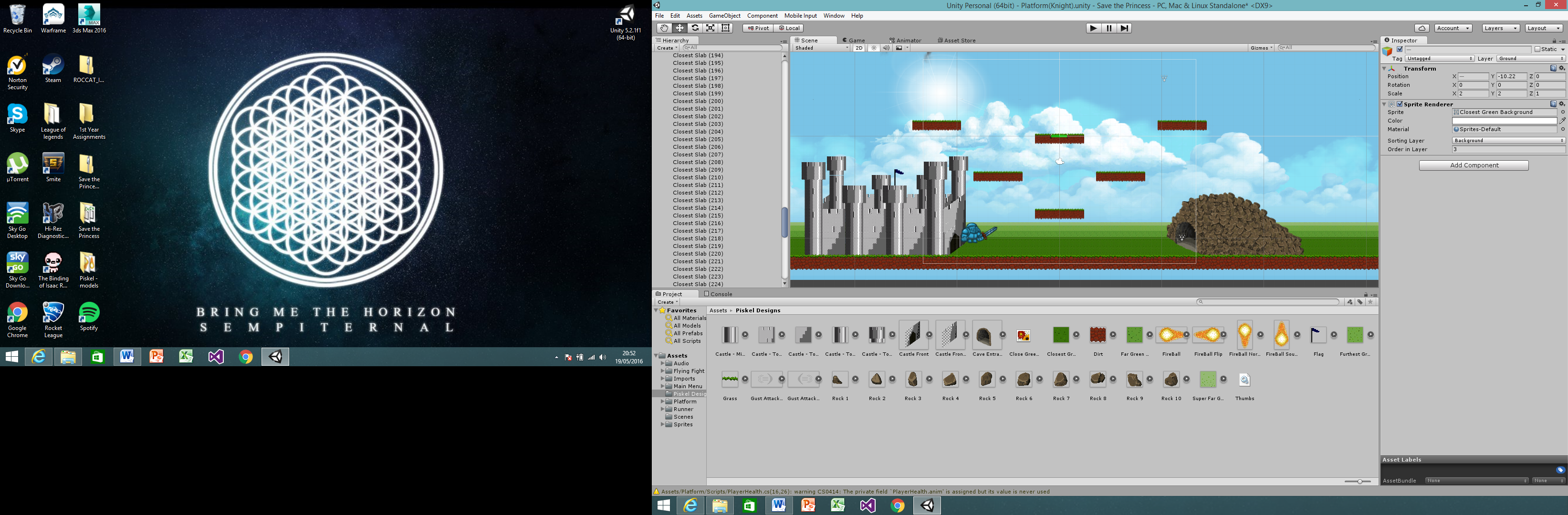
# Screenshots

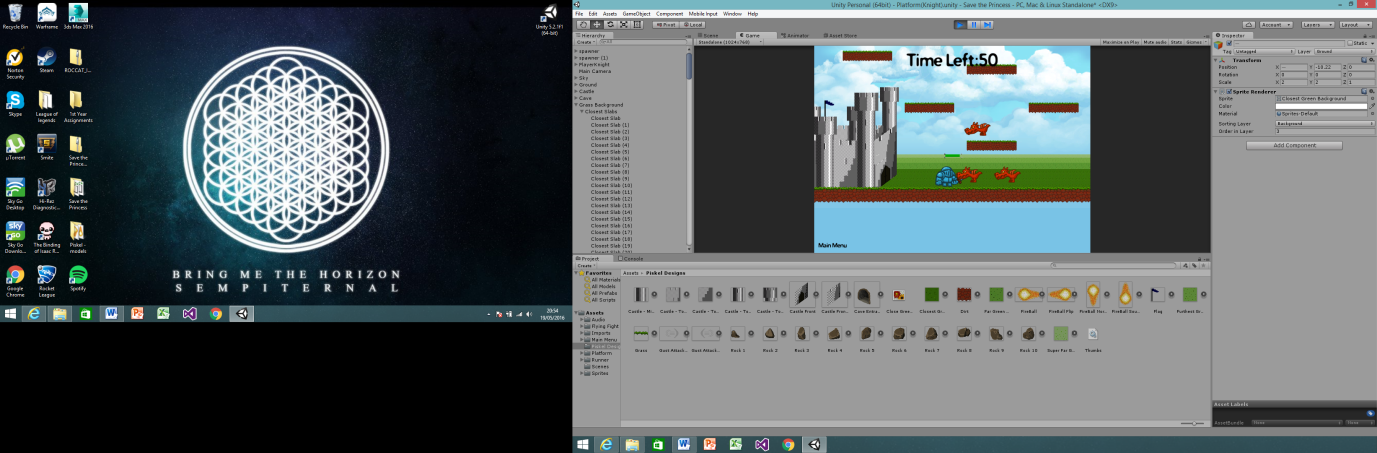
## Main Menu



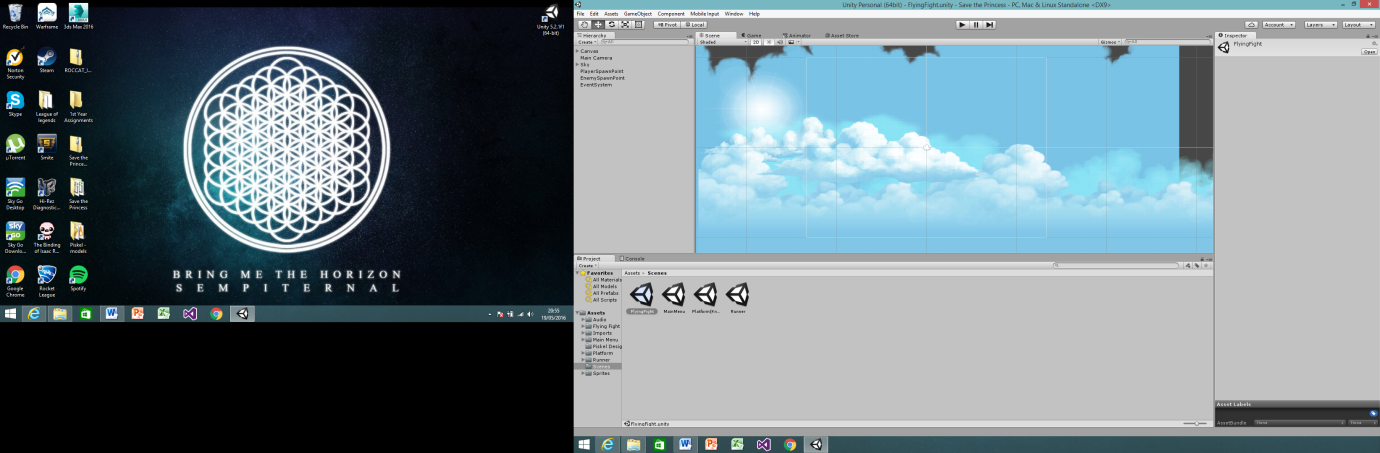
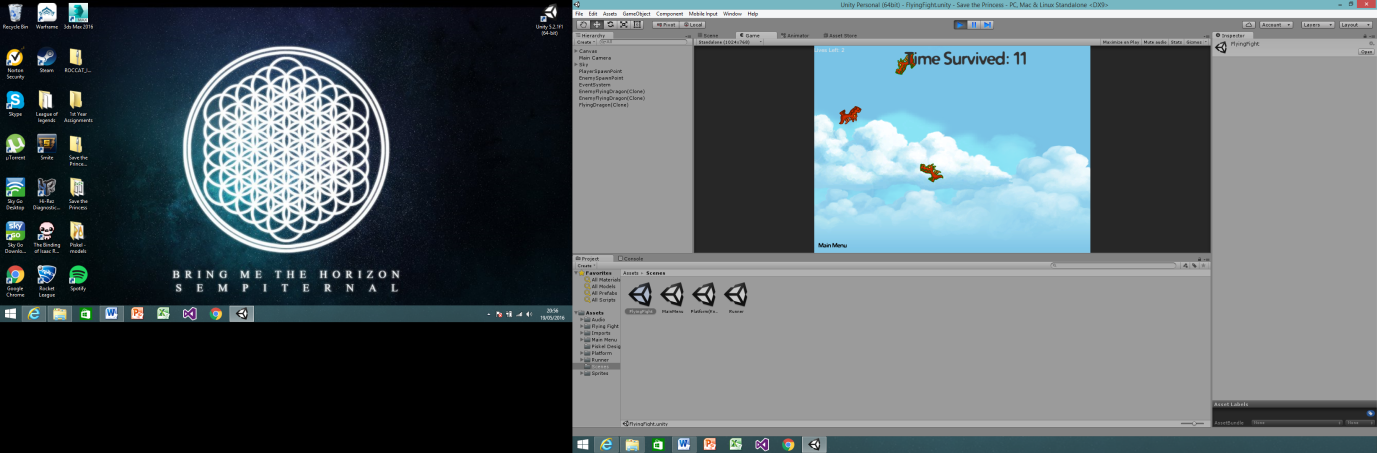


## Princess Retrieval

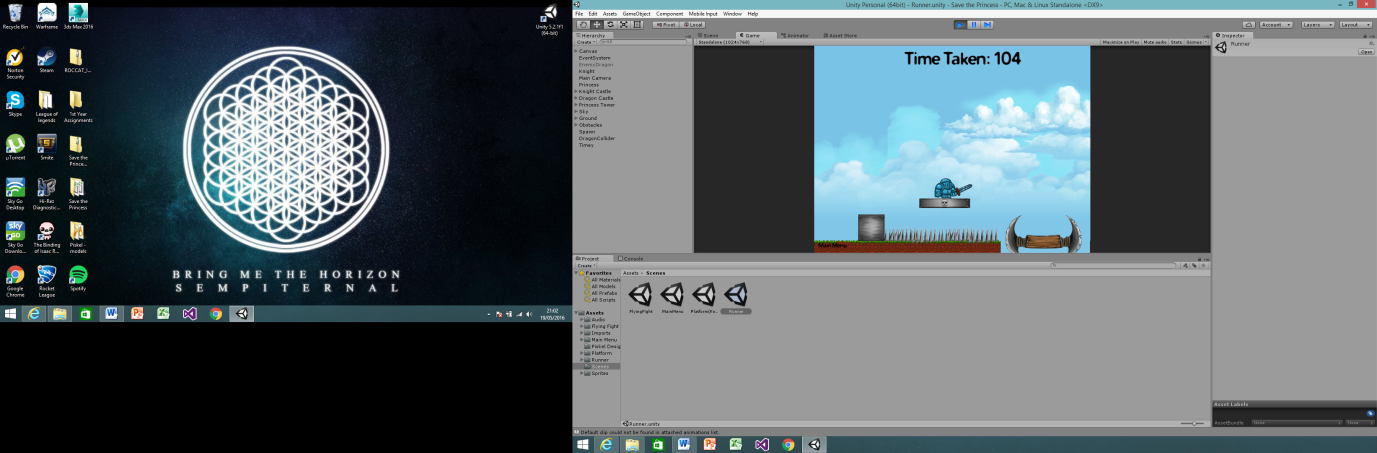




## Dragon Assault



## Restless Knight



# Scripts

## Main Menu

### LoadLevel

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class LevelLoad : MonoBehaviour {

public Button Image;

public GameObject ImageVisibility;

public int LevelNum = 0;

// Use this for initialization

void Start () {

Image = Image.GetComponent<Button> ();

}

public void LoadLevel()

{

Application.LoadLevel (LevelNum); // Send user back to start menu

}

}

## Princess Retrieval

### Bye

using UnityEngine;

using System.Collections;

public class Bye : MonoBehaviour {

void OnTriggerEnter2D(Collider2D col)

{

Destroy (col.gameObject);

}

}

### CameraFollow

using UnityEngine;

using System.Collections;

public class CameraFollow : MonoBehaviour

{

public float xMargin = 1f; // Distance in the x axis the player can move before the camera follows.

public float yMargin = 1f; // Distance in the y axis the player can move before the camera follows.

public float xSmooth = 8f; // How smoothly the camera catches up with it's target movement in the x axis.

public float ySmooth = 8f; // How smoothly the camera catches up with it's target movement in the y axis.

public Vector2 maxXAndY; // The maximum x and y coordinates the camera can have.

public Vector2 minXAndY; // The minimum x and y coordinates the camera can have.

private Transform player; // Reference to the player's transform.

void Awake ()

{

// Setting up the reference.

player = GameObject.FindGameObjectWithTag("Player").transform;

}

bool CheckXMargin()

{

// Returns true if the distance between the camera and the player in the x axis is greater than the x margin.

return Mathf.Abs(transform.position.x - player.position.x) > xMargin;

}

bool CheckYMargin()

{

// Returns true if the distance between the camera and the player in the y axis is greater than the y margin.

return Mathf.Abs(transform.position.y - player.position.y) > yMargin;

}

void FixedUpdate ()

{

TrackPlayer();

}

void TrackPlayer ()

{

// By default the target x and y coordinates of the camera are it's current x and y coordinates.

float targetX = transform.position.x;

float targetY = transform.position.y;

// If the player has moved beyond the x margin...

if(CheckXMargin())

// ... the target x coordinate should be a Lerp between the camera's current x position and the player's current x position.

targetX = Mathf.Lerp(transform.position.x, player.position.x, xSmooth \* Time.deltaTime);

// If the player has moved beyond the y margin...

if(CheckYMargin())

// ... the target y coordinate should be a Lerp between the camera's current y position and the player's current y position.

targetY = Mathf.Lerp(transform.position.y, player.position.y, ySmooth \* Time.deltaTime);

// The target x and y coordinates should not be larger than the maximum or smaller than the minimum.

targetX = Mathf.Clamp(targetX, minXAndY.x, maxXAndY.x);

targetY = Mathf.Clamp(targetY, minXAndY.y, maxXAndY.y);

// Set the camera's position to the target position with the same z component.

transform.position = new Vector3(targetX, targetY, transform.position.z);

}

}

### Countdown

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class Countdown : MonoBehaviour {

public float timeLeft = 300.0f;

public GameObject Princess;

public GameObject PrincessText;

public Text text;

void Update()

{

timeLeft -= Time.deltaTime; // lower time left based on time

text.text = "Time Left:" + Mathf.Round(timeLeft); // round the time to display whole numbers only

if(timeLeft <= 1)

{

Application.LoadLevel(0);

}

else if (timeLeft <= 10)

{

// remove the princess text

PrincessText.SetActive (false);

}

else if (timeLeft <= 15) {

// display the princesstower & Text to inform the user

Princess.SetActive (true);

PrincessText.SetActive (true);

}

}

}

### Destroyer

using UnityEngine;

using System.Collections;

public class Destroyer : MonoBehaviour

{

public bool destroyOnAwake; // Whether or not this gameobject should destroyed after a delay, on Awake.

public float awakeDestroyDelay; // The delay for destroying it on Awake.

public bool findChild = false; // Find a child game object and delete it

public string namedChild; // Name the child object in Inspector

void Awake ()

{

// If the gameobject should be destroyed on awake,

if(destroyOnAwake)

{

if(findChild)

{

Destroy (transform.Find(namedChild).gameObject);

}

else

{

// ... destroy the gameobject after the delay.

Destroy(gameObject, awakeDestroyDelay);

}

}

}

void DestroyChildGameObject ()

{

// Destroy this child gameobject, this can be called from an Animation Event.

if(transform.Find(namedChild).gameObject != null)

Destroy (transform.Find(namedChild).gameObject);

}

void DisableChildGameObject ()

{

// Destroy this child gameobject, this can be called from an Animation Event.

if(transform.Find(namedChild).gameObject.activeSelf == true)

transform.Find(namedChild).gameObject.SetActive(false);

}

void DestroyGameObject ()

{

// Destroy this gameobject, this can be called from an Animation Event.

Destroy (gameObject);

}

}

### EndGame

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

public class EndGame : MonoBehaviour {

public GameObject Image;

public GameObject Title;

public Button Retry;

public Button Quit;

public GameObject RetryVisibility;

public GameObject QuitVisibility;

public GameObject Timer;

public GameObject EndClear;

public GameObject Character;

public GameObject EndDestroyer;

public GameObject Spawner1;

public GameObject Spawner2;

public GameObject Fake;

public GameObject MainCamera;

public GameObject Centre;

void Start ()

{

Retry = Retry.GetComponent<Button> ();

Quit = Quit.GetComponent<Button> ();

}

void OnTriggerEnter2D(Collider2D col)

{

if (col.gameObject.tag == "Player")

{

// if the player is in the collider then gameobjects are set to become active or hide and the main camera is moved to its central position

RetryVisibility.SetActive (true);

QuitVisibility.SetActive (true);

Image.SetActive (true);

Title.SetActive (true);

Timer.SetActive (false);

EndClear.SetActive (true);

Character.SetActive (false);

EndDestroyer.SetActive (true);

Spawner1.SetActive (false);

Spawner2.SetActive (false);

Fake.SetActive (true);

MainCamera.transform.position = Centre.transform.position;

}

}

public void Restart()

{

Application.LoadLevel (1); // restarts level by loading the same scene again

}

public void MainMenu()

{

Application.LoadLevel (0); // Send user back to start menu

}

}

### Enemy

using UnityEngine;

using System.Collections;

public class Enemy : MonoBehaviour

{

public float moveSpeed = 2f; // The speed the enemy moves at.

public int HP = 1; // How many times the enemy can be hit before it dies.

public AudioClip[] deathClips; // An array of audioclips that can play when the enemy dies.

public GameObject hundredPointsUI; // A prefab of 100 that appears when the enemy dies.

public float deathSpinMin = -100f; // A value to give the minimum amount of Torque when dying

public float deathSpinMax = 100f; // A value to give the maximum amount of Torque when dying

public float attackCooldown = 0f;

Transform target;

private Transform frontCheck; // Reference to the position of the gameobject used for checking if something is in front.

private bool dead = false; // Whether or not the enemy is dead.

private Animator anim;

void Awake()

{

// Setting up the references.

//score = GameObject.Find("Score").GetComponent<Score>();

anim = GetComponent<Animator>();

}

void Update ()

{

if (transform.rotation.z != 0) {

transform.rotation = Quaternion.Euler(0,0,0);

}

attackCooldown -= Time.deltaTime;;

}

void FixedUpdate ()

{

target = GameObject.FindWithTag ("Player").transform; // find the player and set as target

Vector3 forwardAxis = new Vector3 (0, 0, -1);

//look at the target and move towards the target constantly

transform.LookAt (target.position, forwardAxis);

Debug.DrawLine (transform.position, target.position);

transform.eulerAngles = new Vector3 (0, 0, -transform.eulerAngles.z);

transform.position -= transform.TransformDirection (Vector2.up) \* moveSpeed ;

Vector3 myself = this.transform.position;

float dist = Vector3.Distance(target.position, myself);

if( dist < 5f && attackCooldown < 1f) {

anim.SetTrigger("Shoot");

attackCooldown = 2f;

}

// If the enemy has zero or fewer hit points and isn't dead yet...

if(HP <= 0 && !dead)

// ... call the death function.

Death ();

}

public void Hurt()

{

// Reduce the number of hit points by one.

HP--;

}

void Death()

{

// Set dead to true.

dead = true;

// Allow the enemy to rotate and spin it by adding a torque.

GetComponent<Rigidbody2D>().fixedAngle = false;

GetComponent<Rigidbody2D>().AddTorque(Random.Range(deathSpinMin,deathSpinMax));

// Find all of the colliders on the gameobject and set them all to be triggers.

Collider2D[] cols = GetComponents<Collider2D>();

foreach(Collider2D c in cols)

{

c.isTrigger = true;

}

}

}

### FolowPlayer

using UnityEngine;

using System.Collections;

public class FollowPlayer : MonoBehaviour

{

public Vector3 offset; // The offset at which the Health Bar follows the player.

private Transform player; // Reference to the player.

void Awake ()

{

// Setting up the reference.

player = GameObject.FindGameObjectWithTag("Player").transform;

}

void Update ()

{

// Set the position to the player's position with the offset.

transform.position = player.position + offset;

}

}

### Gun

using UnityEngine;

using System.Collections;

public class Gun : MonoBehaviour

{

public Rigidbody2D fireball; // Prefab of the rocket.

public float speed = 20f; // The speed the rocket will fire at.

private PlayerControl playerCtrl; // Reference to the PlayerControl script.

private Animator anim; // Reference to the Animator component.

void Awake()

{

// Setting up the references.

anim = transform.root.gameObject.GetComponent<Animator>();

playerCtrl = transform.root.GetComponent<PlayerControl>();

}

void Update ()

{

// If the fire button is pressed...

if(Input.GetButtonDown("Fire1"))

{

// ... set the animator Shoot trigger parameter and play the audioclip.

anim.SetTrigger("Shoot");

//GetComponent<AudioSource>().Play();

// If the player is facing right...

if(playerCtrl.facingRight)

{

// ... instantiate the rocket facing right and set it's velocity to the right.

Rigidbody2D bulletInstance = Instantiate(fireball, transform.position, Quaternion.Euler(new Vector3(0,0,0))) as Rigidbody2D;

bulletInstance.velocity = new Vector2(speed, 0);

}

else

{

// Otherwise instantiate the rocket facing left and set it's velocity to the left.

Rigidbody2D bulletInstance = Instantiate(fireball, transform.position, Quaternion.Euler(new Vector3(0,0,180f))) as Rigidbody2D;

bulletInstance.velocity = new Vector2(-speed, 0);

}

}

}

}

### MainMenuQuit

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class MainMenuQuit : MonoBehaviour {

public Button Quit;

void Start ()

{

Quit = Quit.GetComponent<Button> ();

}

public void MainMenu()

{

Application.LoadLevel (0); // Send user back to start menu

}

}

### Pauser

using UnityEngine;

using System.Collections;

public class Pauser : MonoBehaviour {

private bool paused = false;

// Update is called once per frame

void Update () {

if(Input.GetKeyUp(KeyCode.P))

{

paused = !paused;

}

if(paused)

Time.timeScale = 0;

else

Time.timeScale = 1;

}

}

### PlayerControl

using UnityEngine;

using System.Collections;

public class PlayerControl : MonoBehaviour

{

[HideInInspector]

public bool facingRight = true; // For determining which way the player is currently facing.

[HideInInspector]

public bool jump = false; // Condition for whether the player should jump.

public bool Grounded;

public LayerMask GroundDetection;

private Collider2D Coll;

public float moveForce = 365f; // Amount of force added to move the player left and right.

public float maxSpeed = 5f; // The fastest the player can travel in the x axis.

public float jumpForce = 1000f; // Amount of force added when the player jumps.

private Animator anim; // Reference to the player's animator component.

void Start ()

{

Coll = GetComponent<Collider2D> ();

}

void Awake()

{

// Setting up references.

anim = GetComponent<Animator>();

}

void Update()

{

Grounded = Physics2D.IsTouchingLayers (Coll, GroundDetection);

float v = 0.0f;

#if UNITY\_STANDALONE\_WIN

// If the jump button is pressed and the player is grounded then the player should jump.

if(Input.GetButtonDown("Jump"))

if(Grounded)

{

jump = true;

}

if (transform.rotation.z != 0) {

transform.rotation = Quaternion.Euler(0,0,0);

}

#else

Vector2 touchOrigin = -Vector2.one;

if (Input.touchCount > 0) {

// if there was a touch then add it to the index and review if it was meant to be a jump

Touch touch = Input.touches[0];

if (touch.phase == TouchPhase.Began) {

touchOrigin = touch.position;

}

else if (touch.phase == TouchPhase.Ended) {

Vector2 touchEnd = touch.position;

float y = touchEnd.y - touchOrigin.y;

v = (y > 500)? 1.0f : -1.0f;

}

}

#endif

}

void FixedUpdate ()

{

float h = 0.0f;

#if UNITY\_STANDALONE\_WIN

// Cache the horizontal input.

h = Input.GetAxis("Horizontal");

// The Speed animator parameter is set to the absolute value of the horizontal input.

anim.SetFloat("Speed", Mathf.Abs(h));

// If the player is changing direction (h has a different sign to velocity.x) or hasn't reached maxSpeed yet...

if(h \* GetComponent<Rigidbody2D>().velocity.x < maxSpeed)

// ... add a force to the player.

GetComponent<Rigidbody2D>().AddForce(Vector2.right \* h \* moveForce);

// If the player's horizontal velocity is greater than the maxSpeed...

if(Mathf.Abs(GetComponent<Rigidbody2D>().velocity.x) > maxSpeed)

// ... set the player's velocity to the maxSpeed in the x axis.

GetComponent<Rigidbody2D>().velocity = new Vector2(Mathf.Sign(GetComponent<Rigidbody2D>().velocity.x) \* maxSpeed, GetComponent<Rigidbody2D>().velocity.y);

// If the input is moving the player right and the player is facing left...

if(h > 0 && !facingRight)

// ... flip the player.

Flip();

// Otherwise if the input is moving the player left and the player is facing right...

else if(h < 0 && facingRight)

// ... flip the player.

Flip();

// If the player should jump...

if(jump)

{

// Add a vertical force to the player.

GetComponent<Rigidbody2D>().AddForce(new Vector2(0f, jumpForce));

// Make sure the player can't jump again until the jump conditions from Update are satisfied.

jump = false;

}

#else

Vector2 touchOrigin = -Vector2.one;

if (Input.touchCount > 0) {

// if there is a touch then add it to the index and reviews the origin point to calculate the horizontal movement

Touch touch = Input.touches[0];

if (touch.phase == TouchPhase.Began) {

touchOrigin = touch.position;

}

else if (touch.phase == TouchPhase.Ended) {

Vector2 touchEnd = touch.position;

float x = touchEnd.x - touchOrigin.x;

h = (x > 0)? 1.0f : -1.0f;

}

}

#endif

}

void Flip ()

{

// Switch the way the player is labelled as facing.

facingRight = !facingRight;

// Multiply the player's x local scale by -1.

Vector3 theScale = transform.localScale;

theScale.x \*= -1;

transform.localScale = theScale;

}

}

### PlayerHealth

using UnityEngine;

using System.Collections;

public class PlayerHealth : MonoBehaviour

{

public float health = 100f; // The player's health.

public float repeatDamagePeriod = 2f; // How frequently the player can be damaged.

public AudioClip[] ouchClips; // Array of clips to play when the player is damaged.

public float hurtForce = 10f; // The force with which the player is pushed when hurt.

public float damageAmount = 10f; // The amount of damage to take when enemies touch the player

private SpriteRenderer healthBar; // Reference to the sprite renderer of the health bar.

private float lastHitTime; // The time at which the player was last hit.

private Vector3 healthScale; // The local scale of the health bar initially (with full health).

private PlayerControl playerControl; // Reference to the PlayerControl script.

private Animator anim; // Reference to the Animator on the player

void Awake ()

{

// Setting up references.

playerControl = GetComponent<PlayerControl>();

healthBar = GameObject.Find("Health").GetComponent<SpriteRenderer>();

anim = GetComponent<Animator>();

// Getting the intial scale of the healthbar (whilst the player has full health).

healthScale = healthBar.transform.localScale;

}

void OnCollisionEnter2D (Collision2D col)

{

// If the colliding gameobject is an Enemy...

if(col.gameObject.tag == "Enemy")

{

// ... and if the time exceeds the time of the last hit plus the time between hits...

if (Time.time > lastHitTime + repeatDamagePeriod)

{

// ... and if the player still has health...

if(health > 0f)

{

// ... take damage and reset the lastHitTime.

TakeDamage(col.transform);

lastHitTime = Time.time;

}

// If the player doesn't have health, do some stuff, let him fall into the river to reload the level.

else

{

// Find all of the colliders on the gameobject and set them all to be triggers.

Collider2D[] cols = GetComponents<Collider2D>();

foreach(Collider2D c in cols)

{

c.isTrigger = true;

}

// Move all sprite parts of the player to the front

SpriteRenderer[] spr = GetComponentsInChildren<SpriteRenderer>();

foreach(SpriteRenderer s in spr)

{

s.sortingLayerName = "UI";

}

// ... disable user Player Control script

GetComponent<PlayerControl>().enabled = false;

// ... disable the Gun script to stop a dead guy shooting a nonexistant bazooka

GetComponentInChildren<Gun>().enabled = false;

}

}

}

}

void TakeDamage (Transform enemy)

{

// Make sure the player can't jump.

playerControl.jump = false;

// Create a vector that's from the enemy to the player with an upwards boost.

Vector3 hurtVector = transform.position - enemy.position + Vector3.up \* 5f;

// Add a force to the player in the direction of the vector and multiply by the hurtForce.

GetComponent<Rigidbody2D>().AddForce(hurtVector \* hurtForce);

// Reduce the player's health by 10.

health -= damageAmount;

// Update what the health bar looks like.

UpdateHealthBar();

// Play a random clip of the player getting hurt.

/\* int i = Random.Range (0, ouchClips.Length);

AudioSource.PlayClipAtPoint(ouchClips[i], transform.position);

\*/

}

public void UpdateHealthBar ()

{

// Set the health bar's colour to proportion of the way between green and red based on the player's health.

healthBar.material.color = Color.Lerp(Color.green, Color.red, 1 - health \* 0.01f);

// Set the scale of the health bar to be proportional to the player's health.

healthBar.transform.localScale = new Vector3(healthScale.x \* health \* 0.01f, 1, 1);

}

}

### Princess

using UnityEngine;

using System.Collections;

public class Princess : MonoBehaviour {

public GameObject princess;

public GameObject stalker;

public GameObject win;

public GameObject Bye;

void OnTriggerEnter2D()

{

// Sets the stalker princess to active and removes the one from the tower as well as activated the win condition collider

princess.SetActive (false);

stalker.SetActive (true);

win.SetActive (true);

Bye.SetActive (true);

}

}

### Remover

using UnityEngine;

using System.Collections;

public class Remover : MonoBehaviour

{

void OnTriggerEnter2D(Collider2D col)

{

// If the player hits the trigger...

if(col.gameObject.tag == "Player")

{

// .. stop the camera tracking the player

GameObject.FindGameObjectWithTag("MainCamera").GetComponent<CameraFollow>().enabled = false;

// .. stop the Health Bar following the player

if(GameObject.FindGameObjectWithTag("HealthBar").activeSelf)

{

GameObject.FindGameObjectWithTag("HealthBar").SetActive(false);

}

// ... destroy the player.

Destroy (col.gameObject);

// ... reload the level.

StartCoroutine("ReloadGame");

}

else

{

// Destroy the enemy.

Destroy (col.gameObject);

}

}

IEnumerator ReloadGame()

{

// ... pause briefly

yield return new WaitForSeconds(2);

// ... and then reload the level.

Application.LoadLevel(Application.loadedLevel);

}

}

### Rocket

using UnityEngine;

using System.Collections;

public class Rocket : MonoBehaviour

{

public GameObject explosion; // Prefab of explosion effect.

void Start ()

{

// Destroy the rocket after 2 seconds if it doesn't get destroyed before then.

Destroy(gameObject, 2);

}

void OnExplode()

{

// Create a quaternion with a random rotation in the z-axis.

Quaternion randomRotation = Quaternion.Euler(0f, 0f, Random.Range(0f, 360f));

// Instantiate the explosion where the rocket is with the random rotation.

//Instantiate(explosion, transform.position, randomRotation);

}

void OnTriggerEnter2D (Collider2D col)

{

// If it hits an enemy...

if(col.tag == "Enemy")

{

// ... find the Enemy script and call the Hurt function.

col.gameObject.GetComponent<Enemy>().Hurt();

// Call the explosion instantiation.

OnExplode();

// Destroy the rocket.

Destroy (gameObject);

}

// Otherwise if the player manages to shoot himself...

else if(col.gameObject.tag != "Player")

{

// Instantiate the explosion and destroy the rocket.

OnExplode();

Destroy (gameObject);

}

}

}

### Spawner

using UnityEngine;

using System.Collections;

public class Spawner : MonoBehaviour

{

public float spawnTime = 5f; // The amount of time between each spawn.

public float spawnDelay = 3f; // The amount of time before spawning starts.

public GameObject[] enemies; // Array of enemy prefabs.

void Start ()

{

// Start calling the Spawn function repeatedly after a delay .

InvokeRepeating("Spawn", spawnDelay, spawnTime);

}

void Spawn ()

{

// Instantiate a random enemy.

int enemyIndex = Random.Range(0, enemies.Length);

Instantiate(enemies[enemyIndex], transform.position, transform.rotation);

// Play the spawning effect from all of the particle systems.

foreach(ParticleSystem p in GetComponentsInChildren<ParticleSystem>())

{

p.Play();

}

}

}

### SetParticleSortingLayer

using UnityEngine;

using System.Collections;

public class SetParticleSortingLayer : MonoBehaviour

{

public string sortingLayerName; // The name of the sorting layer the particles should be set to.

void Start ()

{

// Set the sorting layer of the particle system.

GetComponent<ParticleSystem>().GetComponent<Renderer>().sortingLayerName = sortingLayerName;

}

}

### Zoom

using UnityEngine;

using System.Collections;

public class Zoom : MonoBehaviour {

public float zoomSpeed = 0.05f;

private float minSize = 6.0f;

private float maxSize = 20.0f;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

if (Input.touchCount == 2) {

// If using two fingers and pulling towards or away on a mobile device it will activate a camera zooming feature

Touch touchZero = Input.GetTouch(0);

Touch touchOne = Input.GetTouch(1);

Vector2 touchZeroPrevPos = touchZero.position - touchZero.deltaPosition;

Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;

float prevTouchDeltaMag = (touchZeroPrevPos - touchOnePrevPos).magnitude;

float touchDeltaMag = (touchZero.position - touchOne.position).magnitude;

float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;

Camera.main.orthographicSize += deltaMagnitudeDiff \* zoomSpeed;

Camera.main.orthographicSize = Mathf.Clamp(GetComponent<Camera>().orthographicSize, minSize, maxSize);

}

}

}

## Dragon Assault

### EnemyShooting

using UnityEngine;

using System.Collections;

public class EnemyShooting : MonoBehaviour {

public Vector3 bulletOffset = new Vector3(0, 0.5f, 0);

public GameObject bulletPrefab;

int bulletLayer;

public float fireDelay = 0.50f;

float cooldownTimer = 0;

Transform player;

void Start() {

bulletLayer = gameObject.layer;

}

// Update is called once per frame

void Update () {

if(player == null) {

// Find the player's ship!

GameObject go = GameObject.FindWithTag ("Player"); //track the player

if(go != null) {

player = go.transform; // shoot at player

}

}

cooldownTimer -= Time.deltaTime;

if( cooldownTimer <= 0 && player != null && Vector3.Distance(transform.position, player.position) < 4) {

// SHOOT!

//Debug.Log ("Enemy Pew!");

cooldownTimer = fireDelay;

Vector3 offset = transform.rotation \* bulletOffset;

GameObject bulletGO = (GameObject)Instantiate(bulletPrefab, transform.position + offset, transform.rotation); // spawn the bullet

bulletGO.layer = bulletLayer;

}

}

}

### EnemySpawner

using UnityEngine;

using System.Collections;

public class EnemySpawner : MonoBehaviour {

public GameObject enemyPrefab;

float spawnDistance = 12f;

float enemyRate = 6;

float nextEnemy = 1;

// Update is called once per frame

void Update () {

nextEnemy -= Time.deltaTime; // use time to determine when to spawn the new enemy

if(nextEnemy <= 0) {

nextEnemy = enemyRate; // reset spawn time

enemyRate \*= 0.9f; // lower enemy rate for increasing diffculty

if(enemyRate < 2)

enemyRate = 2; // set every 2 seconds to lowest possible, for performance purposes mainly

Vector3 offset = Random.onUnitSphere;

offset.z = 0;

offset = offset.normalized \* spawnDistance;

Instantiate(enemyPrefab, transform.position + offset, Quaternion.identity);

// spawn the enemy at a random position over 12f from the player spawn

}

}

}

### PlayerSpawner

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class PlayerSpawner : MonoBehaviour {

public GameObject playerPrefab; //prefab for spawning the player

GameObject playerInstance; // tracked version of player

public GameObject Image;

public GameObject Title;

public Button Retry;

public Button Quit;

public GameObject RetryVisibility;

public GameObject QuitVisibility;

public int numLives = 4; // number of lives

float respawnTimer; // how long it takes to respawn

// Use this for initialization

void Start ()

{

Retry = Retry.GetComponent<Button> ();

Quit = Quit.GetComponent<Button> ();

SpawnPlayer(); // run the spawnplayer function

}

void SpawnPlayer()

{

numLives--; // remove a life

respawnTimer = 2; // set respawn time

playerInstance = (GameObject)Instantiate(playerPrefab, transform.position, Quaternion.identity); // spawn and track object

}

// Update is called once per frame

void Update ()

{

if(playerInstance == null && numLives > 0) { // check for instance and lives

respawnTimer -= Time.deltaTime; // decrease the respawnTimer over time

if(respawnTimer <= 0) { // if the respawn timer is 0 spawn player

SpawnPlayer();

}

}

}

void OnGUI()

{

if(numLives > 0 || playerInstance!= null) {

GUI.Label( new Rect(0, 0, 100, 50), "Lives Left: " + numLives); // create a GUI element that displays the amount of lives you have left

}

else

{

RetryVisibility.SetActive (true);

QuitVisibility.SetActive (true);

Image.SetActive (true);

Title.SetActive (true);

}

}

public void Restart()

{

Application.LoadLevel (2); // restarts level by loading the same scene again

}

public void MainMenu()

{

Application.LoadLevel (0); // Send user back to start menu

}

}

### DamageHandler

using UnityEngine;

using System.Collections;

public class DamageHandler : MonoBehaviour {

public int health = 1;

public float invulnPeriod = 0;

float invulnTimer = 0;

int correctLayer;

SpriteRenderer spriteRend;

void Start() {

correctLayer = gameObject.layer;

// NOTE! This only get the renderer on the parent object.

// In other words, it doesn't work for children. I.E. "enemy01"

spriteRend = GetComponent<SpriteRenderer>();

if(spriteRend == null) {

spriteRend = transform.GetComponentInChildren<SpriteRenderer>(); // find sprite renderer

if(spriteRend==null) {

Debug.LogError("Object '"+gameObject.name+"' has no sprite renderer.");

}

}

}

void OnTriggerEnter2D() { // if it hits lower health

health--;

if(invulnPeriod > 0) {

invulnTimer = invulnPeriod;

gameObject.layer = 10;

}

}

void Update() {

if(invulnTimer > 0) {

invulnTimer -= Time.deltaTime; // lower invu time

if(invulnTimer <= 0) {

gameObject.layer = correctLayer;

if(spriteRend != null) {

spriteRend.enabled = true;

}

}

else {

if(spriteRend != null) {

spriteRend.enabled = !spriteRend.enabled;

}

}

}

if(health <= 0) {

Die(); // run die

}

}

void Die() {

Destroy(gameObject); // destroy the shot object

}

}

### FacesPlayer

using UnityEngine;

using System.Collections;

public class FacesPlayer : MonoBehaviour {

public float rotSpeed = 90f;

Transform player;

// Update is called once per frame

void Update () {

if(player == null) {

// Find the player's ship!

GameObject go = GameObject.FindWithTag ("Player"); // find player

if(go != null) {

player = go.transform; // player transform

}

}

// At this point, we've either found the player,

// or he/she doesn't exist right now.

if(player == null)

return; // Try again next frame!

// HERE -- we know for sure we have a player. Turn to face it!

Vector3 dir = player.position - transform.position;

dir.Normalize();

float zAngle = Mathf.Atan2(dir.y, dir.x) \* Mathf.Rad2Deg - 90;

Quaternion desiredRot = Quaternion.Euler( 0, 0,zAngle );

transform.rotation = Quaternion.RotateTowards( transform.rotation, desiredRot, rotSpeed \* Time.deltaTime);

}

}

### MoveForward

using UnityEngine;

using System.Collections;

public class MoveForward : MonoBehaviour {

public float maxSpeed = 5f;

// Update is called once per frame

void Update () {

// calculate current position and move foward from that position constantly

Vector3 pos = transform.position;

Vector3 velocity = new Vector3(0, maxSpeed \* Time.deltaTime, 0);

pos += transform.rotation \* velocity;

transform.position = pos;

}

}

### PlayerMovement

using UnityEngine;

using System.Collections;

public class PlayerMovement : MonoBehaviour {

public float maxSpeed = 5f;

public float rotSpeed = 180f;

float shipBoundaryRadius = 0.5f;

void Start () {

}

void Update () {

// ROTATE the ship.

// Grab our rotation quaternion

Quaternion rot = transform.rotation;

// Grab the Z euler angle

float z = rot.eulerAngles.z;

// Change the Z angle based on input

z -= Input.GetAxis("Horizontal") \* rotSpeed \* Time.deltaTime;

// Recreate the quaternion

rot = Quaternion.Euler( 0, 0, z );

// Feed the quaternion into our rotation

transform.rotation = rot;

// MOVE the ship.

Vector3 pos = transform.position;

Vector3 velocity = new Vector3(0, Input.GetAxis("Vertical") \* maxSpeed \* Time.deltaTime, 0);

pos += rot \* velocity;

// RESTRICT the player to the camera's boundaries!

// First to vertical, because it's simpler

if(pos.y+shipBoundaryRadius > Camera.main.orthographicSize) {

pos.y = Camera.main.orthographicSize - shipBoundaryRadius;

}

if(pos.y-shipBoundaryRadius < -Camera.main.orthographicSize) {

pos.y = -Camera.main.orthographicSize + shipBoundaryRadius;

}

// Now calculate the orthographic width based on the screen ratio

float screenRatio = (float)Screen.width / (float)Screen.height;

float widthOrtho = Camera.main.orthographicSize \* screenRatio;

// Now do horizontal bounds

if(pos.x+shipBoundaryRadius > widthOrtho) {

pos.x = widthOrtho - shipBoundaryRadius;

}

if(pos.x-shipBoundaryRadius < -widthOrtho) {

pos.x = -widthOrtho + shipBoundaryRadius;

}

// Finally, update our position!!

transform.position = pos;

}

}

### PlayerShooting

using UnityEngine;

using System.Collections;

public class PlayerShooting : MonoBehaviour {

public Vector3 bulletOffset = new Vector3(0, 0.5f, 0);

public GameObject bulletPrefab;

int bulletLayer;

public float fireDelay = 0.25f;

float cooldownTimer = 0;

void Start() {

bulletLayer = gameObject.layer;

}

// Update is called once per frame

void Update () {

cooldownTimer -= Time.deltaTime;

if( Input.GetButton("Fire1") && cooldownTimer <= 0 ) {

// Do the SHOOT!

cooldownTimer = fireDelay;

Vector3 offset = transform.rotation \* bulletOffset;

GameObject bulletGO = (GameObject)Instantiate(bulletPrefab, transform.position + offset, transform.rotation);

bulletGO.layer = bulletLayer;

}

}

}

### Self Destruct

using UnityEngine;

using System.Collections;

public class SelfDestruct : MonoBehaviour {

public float timer = 1f;

void Update () {

// cleanup for objects that don't need to be around forever, no forever flying fireballs

timer -= Time.deltaTime;

if(timer <= 0) {

Destroy(gameObject);

}

}

}

### Timer

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class Timer : MonoBehaviour {

public Text text;

public float timeTotal = 0f;

public PlayerSpawner spawnerscript; // refers to the player spawner script

// Update is called once per frame

void Update () {

if (spawnerscript.numLives >= 1) // references the numLives variable from the player spawner script

{

timeTotal += Time.deltaTime; // increases total time based on time that has passed

text.text = "Time Survived: " + Mathf.Round (timeTotal); // round the time to display whole numbers only

}

}

}

## Restless Knight

### Die

using UnityEngine;

using System.Collections;

public class Die : MonoBehaviour {

public GameObject Character;

public GameObject Spawn;

void OnTriggerEnter2D(Collider2D col)

{

if (col.gameObject.tag == "Player") {

Character.transform.position = Spawn.transform.position; // send the player back to spawn if they fall off the map

}

else

{

Destroy(col.gameObject); // destroy anything that isn't the player that falls off the map

}

}

}

### DragonSpawner

using UnityEngine;

using System.Collections;

public class DragonSpawn : MonoBehaviour {

public GameObject Dragon;

void OnTriggerEnter2D()

{

Dragon.SetActive (true); // when you go through the end collider make the dragon visible

}

}

### EndScreen

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class EndScreen : MonoBehaviour {

public GameObject Image;

public GameObject Title;

public Button Retry;

public Button Quit;

public GameObject RetryVisibility;

public GameObject QuitVisibility;

public GameObject Timey;

void Start ()

{

Retry = Retry.GetComponent<Button> ();

Quit = Quit.GetComponent<Button> ();

}

void OnTriggerEnter2D (Collider2D col)

{

if (col.gameObject.tag == "Player")

{

RetryVisibility.SetActive (true);

QuitVisibility.SetActive (true);

Image.SetActive (true);

Title.SetActive (true);

Timey.SetActive (false);

}

}

public void Restart()

{

Application.LoadLevel (3); // restarts level by loading the same scene again

}

public void MainMenu()

{

Application.LoadLevel (0); // Send user back to start menu

}

}

### KnightGun

using UnityEngine;

using System.Collections;

public class KnightGun : MonoBehaviour {

public Rigidbody2D Gust; // Prefab of the gust

public float speed = 20f; // The speed the gust will fire at.

private PlayerControl playerCtrl; // Reference to the PlayerControl script.

private Animator anim; // Reference to the Animator component.

void Awake()

{

// Setting up the references.

anim = transform.root.gameObject.GetComponent<Animator>();

playerCtrl = transform.root.GetComponent<PlayerControl>();

}

void Update ()

{

// If the fire button is pressed...

if(Input.GetButtonDown("Fire1"))

{

// ... set the animator Shoot trigger parameter and play the audioclip.

anim.SetTrigger("Shoot");

// If the player is facing right...

if(playerCtrl.facingRight)

{

StartCoroutine(ShootRight());

}

else

{

StartCoroutine(ShootLeft());

}

}

}

IEnumerator ShootRight() {

// ... instantiate the gust facing right and set it's velocity to the right.

yield return new WaitForSeconds(0.5f);

Rigidbody2D bulletInstance = Instantiate(Gust, transform.position, Quaternion.Euler(new Vector3(0,-2f,0))) as Rigidbody2D;

bulletInstance.velocity = new Vector2(speed, 0);

}

IEnumerator ShootLeft() {

// Otherwise instantiate the gust facing left and set it's velocity to the left.

yield return new WaitForSeconds(0.5f);

Rigidbody2D bulletInstance = Instantiate(Gust, transform.position, Quaternion.Euler(new Vector3(0,-2f,180f))) as Rigidbody2D;

bulletInstance.velocity = new Vector2(-speed, 0);

}

}

### RunnerCamera

using UnityEngine;

using System.Collections;

public class RunnerCamera : MonoBehaviour {

public PlayerControl ThePlayer;

private Vector3 PlayerPosition;

private float DistanceToMove;

void Start ()

{

ThePlayer = FindObjectOfType<PlayerControl>(); // find the playercontrol script

PlayerPosition = ThePlayer.transform.position;

}

void Update ()

{

DistanceToMove = ThePlayer.transform.position.x - PlayerPosition.x; // calculate movement

transform.position = new Vector3 (transform.position.x + DistanceToMove, transform.position.y, transform.position.z); // set new position

PlayerPosition = ThePlayer.transform.position;

}

}

### RunnerTimer

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

public class RunnerTimer : MonoBehaviour {

public Text text;

public float timeTotal = 0f;

void Update () {

timeTotal += Time.deltaTime; // increases total time based on time that has passed

text.text = "Time Taken: " + Mathf.Round (timeTotal); // round the time to display whole numbers only

}

}

# Contribution Breakdown

Luke = 21519352

Ryan = 21515288

Main Menu

Art = Luke

UI = Luke

Scripts = Ryan

Princess Retrieval

Art = Luke

UI = Ryan

Scripts = Joint Effort on majority of scripts used for this game

Dragon Assault =Ryan

Restless Knight = Luke