AGA206 Roll A Ball
Optional Modules

5. Level Select Menu & Multiple Levels

2pts

Pre-requisite: Start and Game Over Screen

Create a new level for the game.
Add buttons for each level in the game (minimum 2)
When the player clicks on a level button, it loads the appropriate level

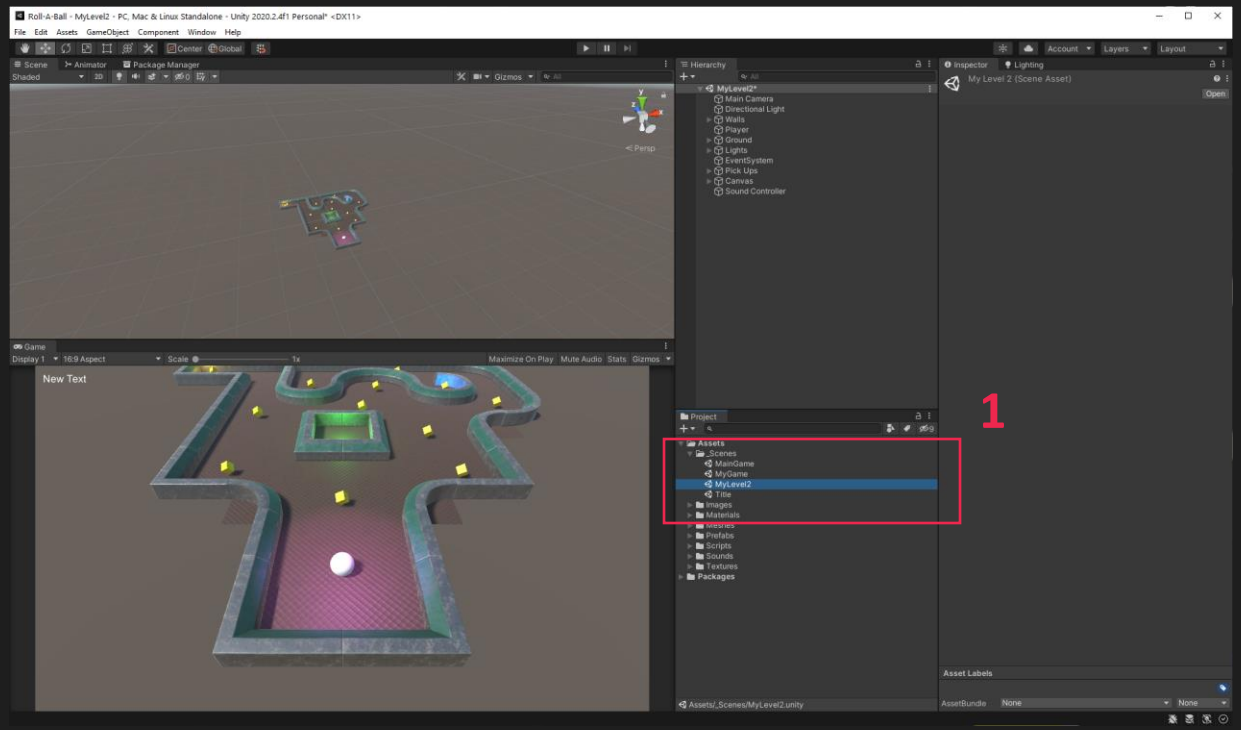
Design: UI Layout, New levels in Unity

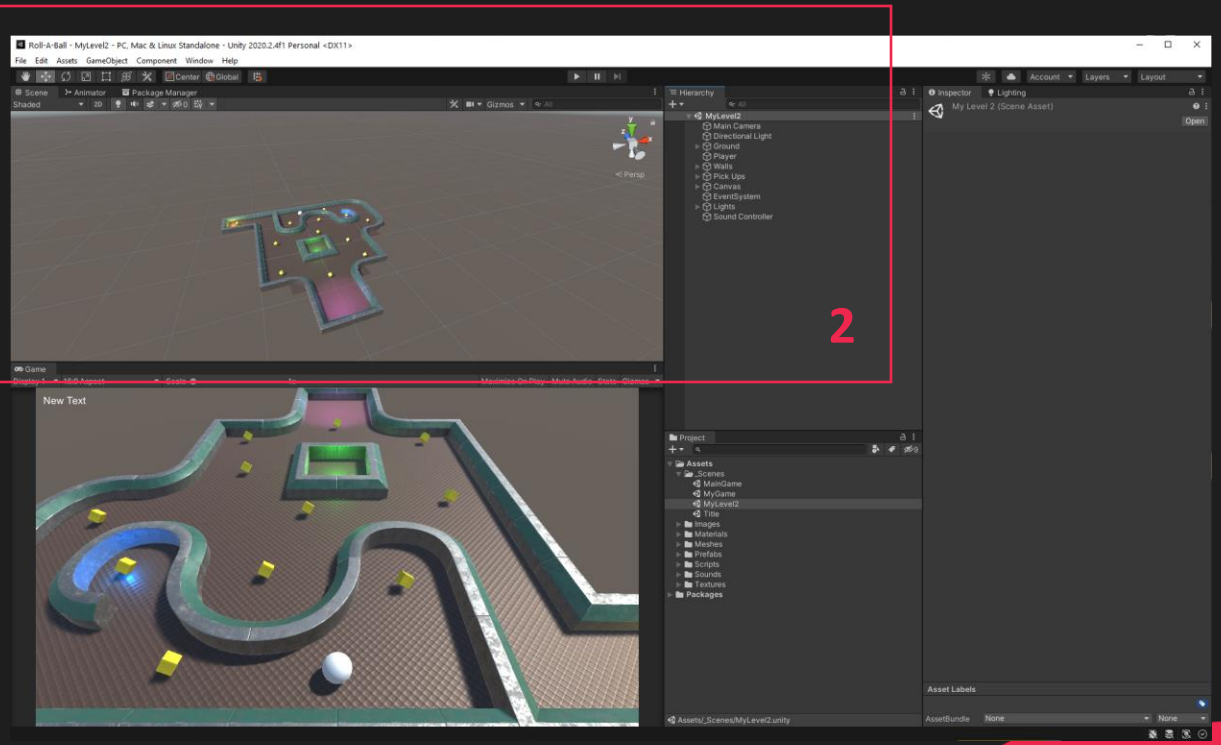
Programming: UI Screens and Buttons, Scene Management

Create the new level

1. In the _Scenes folder, duplicate your first scene and name it something identifiable like MyLevel2
   - We are duplicating so it retains all the functionality we set up so far

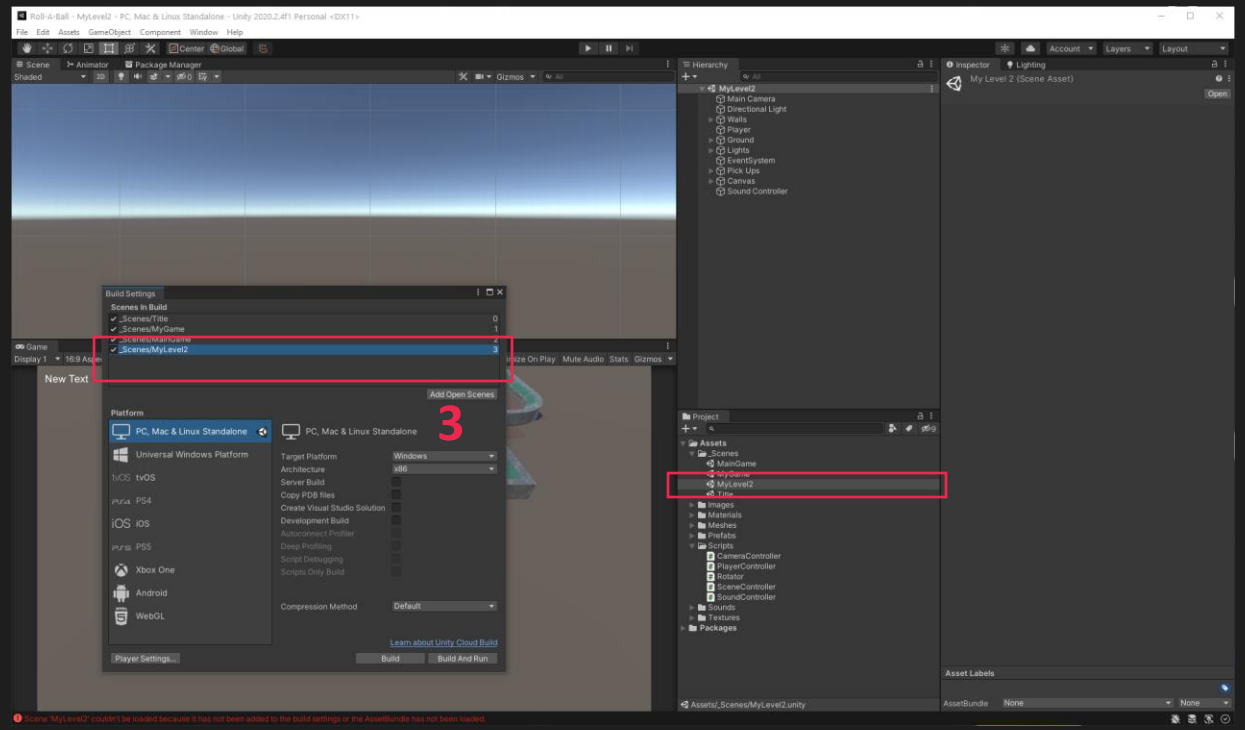
2. Change your level as you wish
   - Replace Textures
   - Create new layout
   - Add extra features from other modules





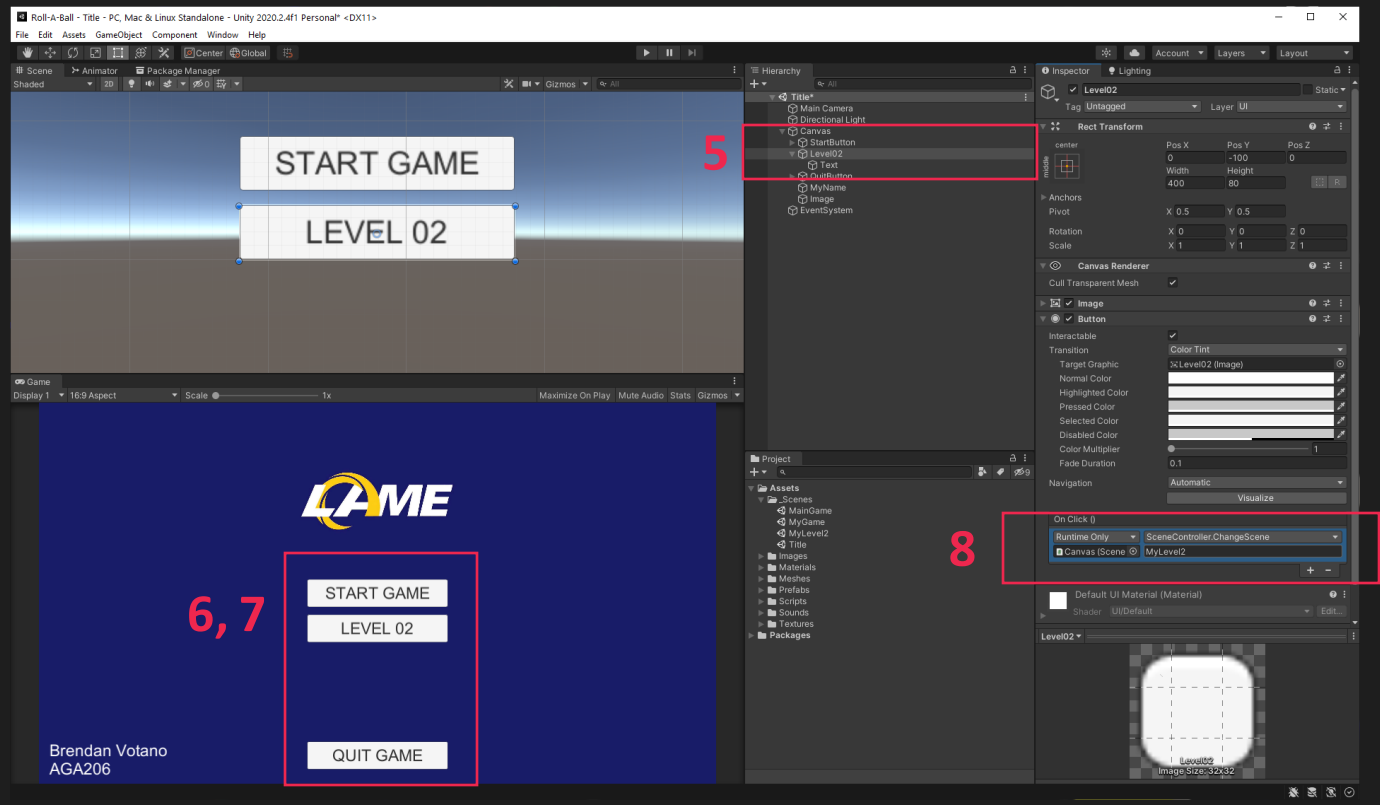
Add the new level to the Build Settings

3. Add the new scene to the Build Settings by dragging it from the project folder or clicking Add Open Scene
   - Build settings are in File Menu > Build Settings



Hook up the new level on the title screen

4. Go to the Title scene we created in Module 1
5. Duplicate the Start Level button and name it Level02
6. Reposition the buttons as you want
7. Change the text on the new button
8. On the new buttons On Click event, change the scene name to match EXACTLY the scene name in your project folder
9. Repeat for any other levels you want to add
10. DONE!!!



6. Speed Run Mode

2pts

Pre-requisite: Start and Game Over Screen

When the level begins, the is a 3 second countdown.
When the countdown has finished, start a timer for the player's progress.
When the player has collected all the pickups, we call the game over condition.
The game over screen displays the current players time, along with the best time for the level

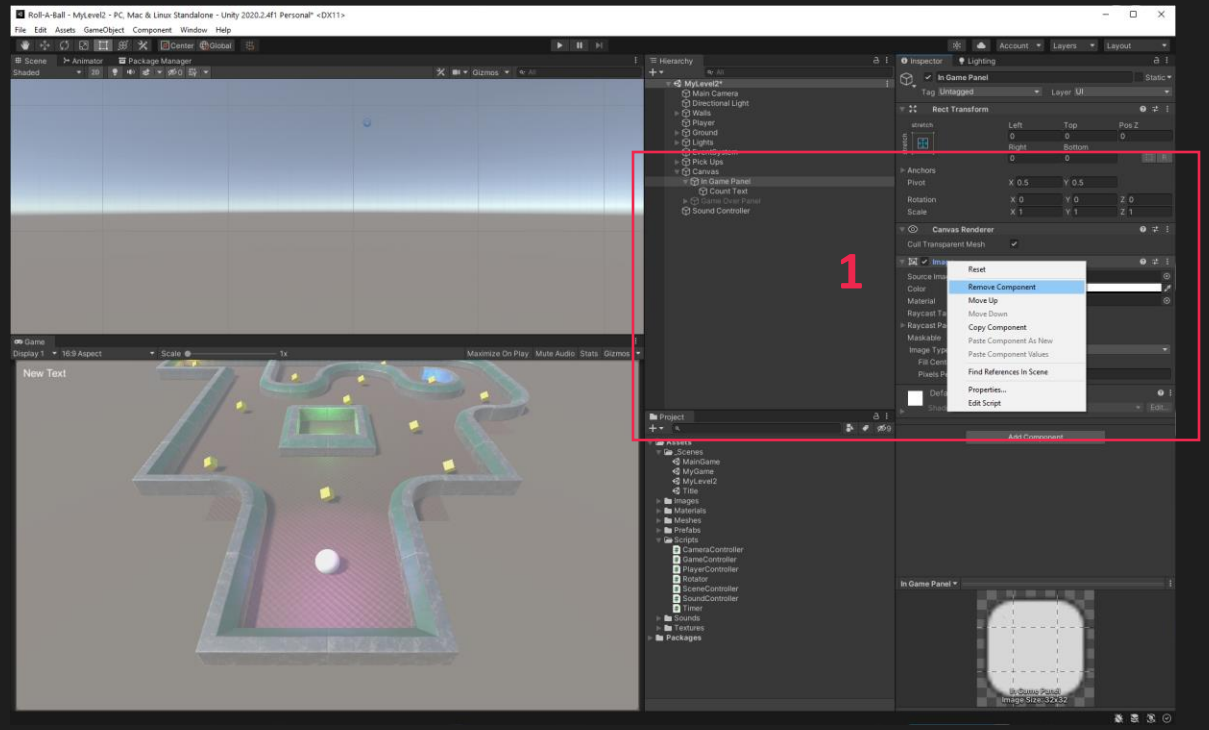
Design: UI Layout

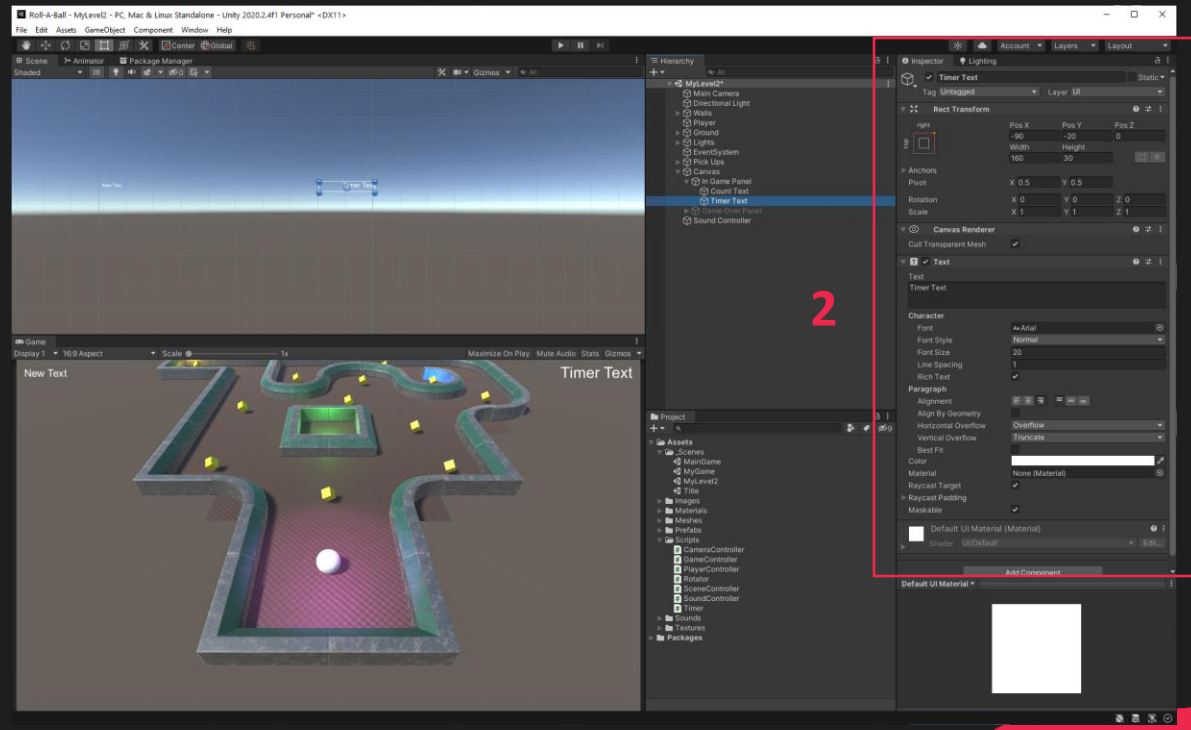
Programming: UI Screens, Timers, Saving player data

Create some new UI

1. In the Canvas, create a new Panel called In Game Panel, remove the image component on it and drag the old Count Text object under it

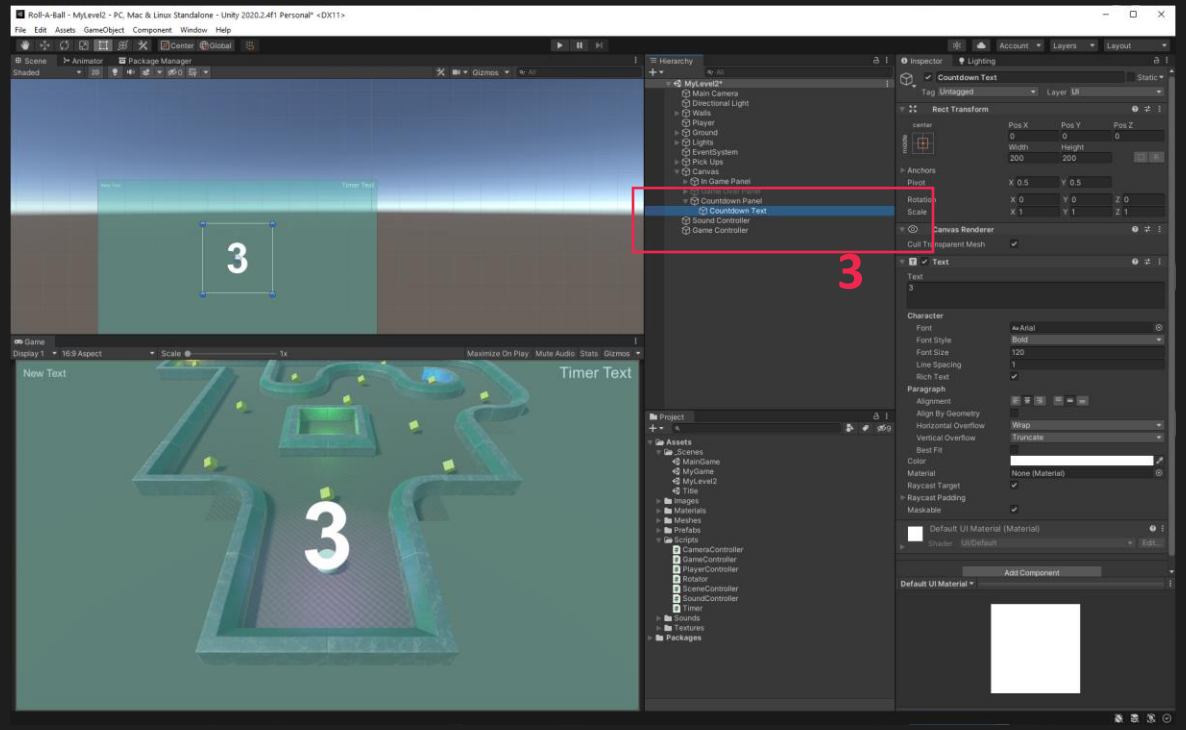
2. In our In Game Panel, create a new UI Text object called Timer Text and position it as in the image
   - Feel free to style it to your liking though



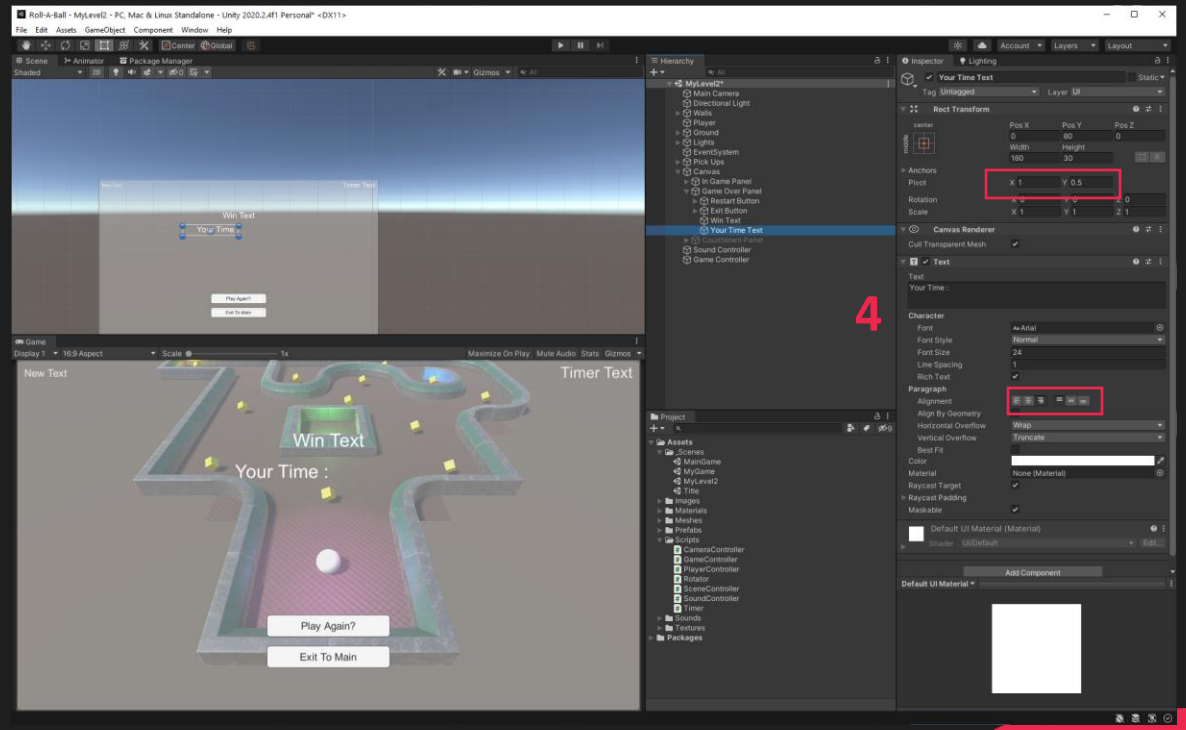


Create some new panels

3. Create another new Panel called Countdown Panel, colour the image as you wish then add a UI Text element as a child called Countdown Text
   - Style as you like or copy the settings in the image

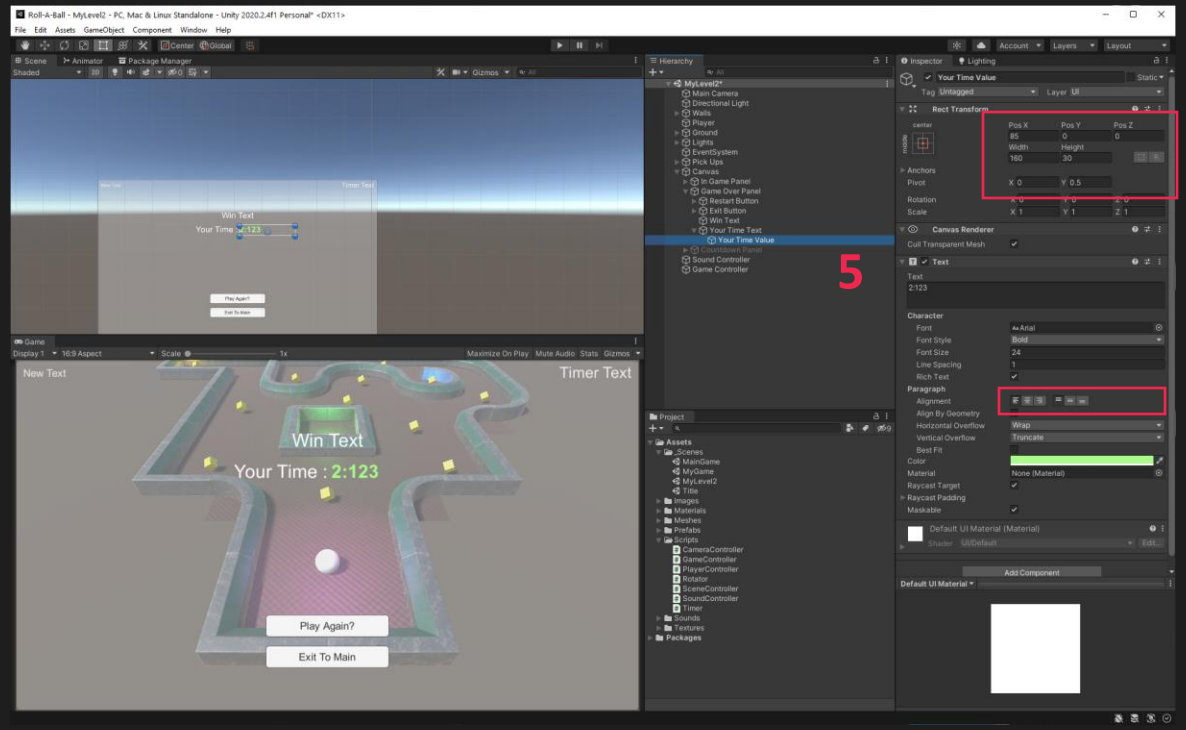


4. Turn Off our Countdown Panel and Turn on our Game Over Panel. Add a UI Text called Your Time Text and set it as in the image.
   - Reposition it, the Win Text and the buttons to fit better

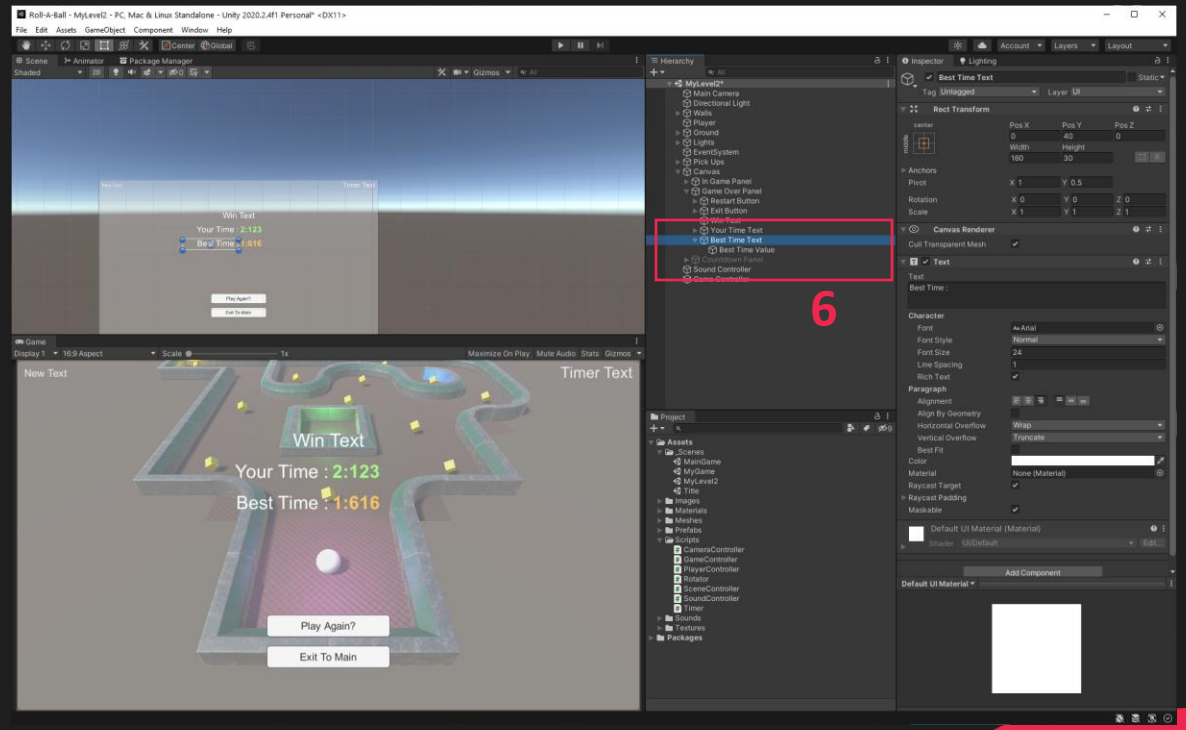


Create some new panels

5. Create a child of the Your Time Text called Your Time Value, put a dummy value in it and position it as in the image
   - Style it how you like with colours, font, etc.



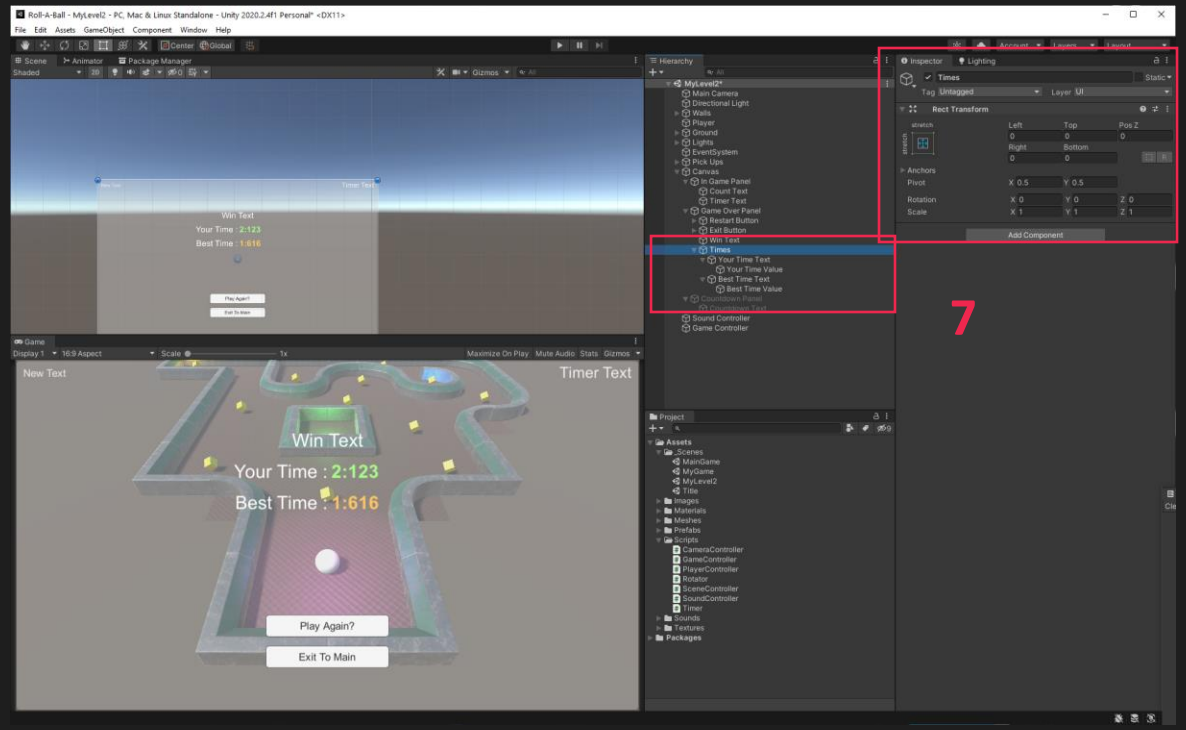
6. Duplicate the Your Time Text object, rename to Best Time Text and the child object Best Time Value. Position it below the Your Time Text

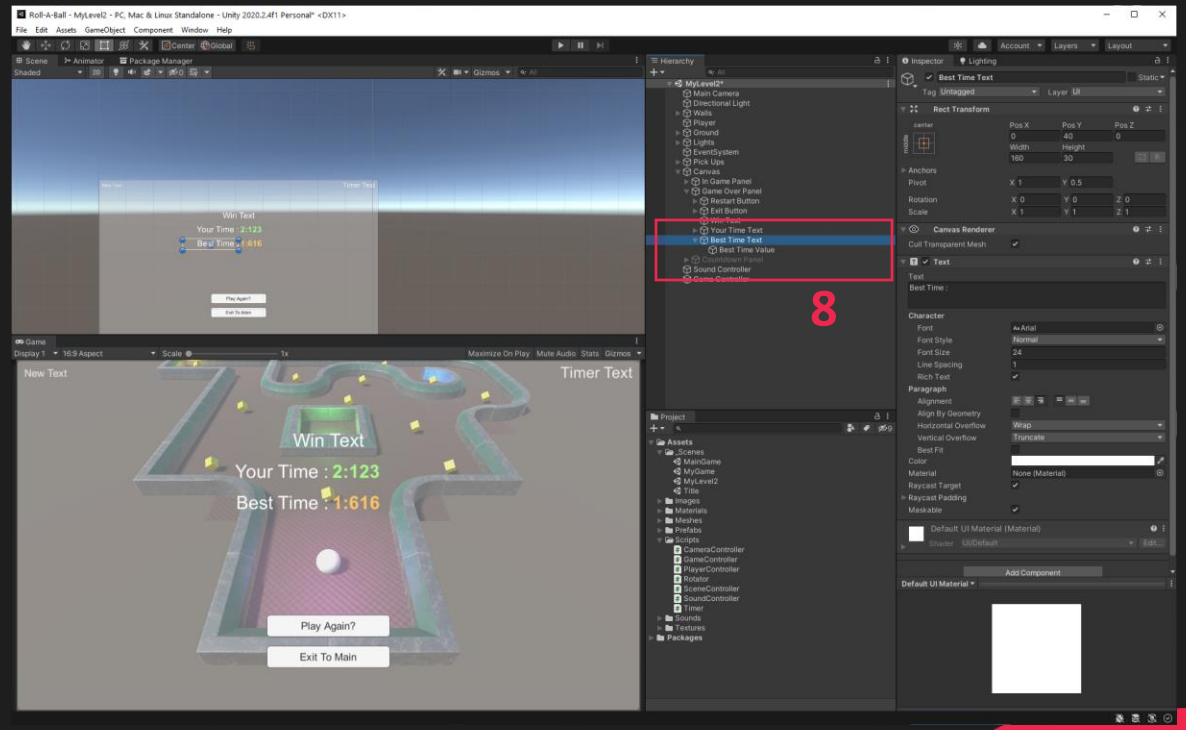


Create some new panels

7. Create an empty gameobject in the Game Over Panel, name it Times, change its settings as in the image and drag the Your Time Text and Best Time Text into it

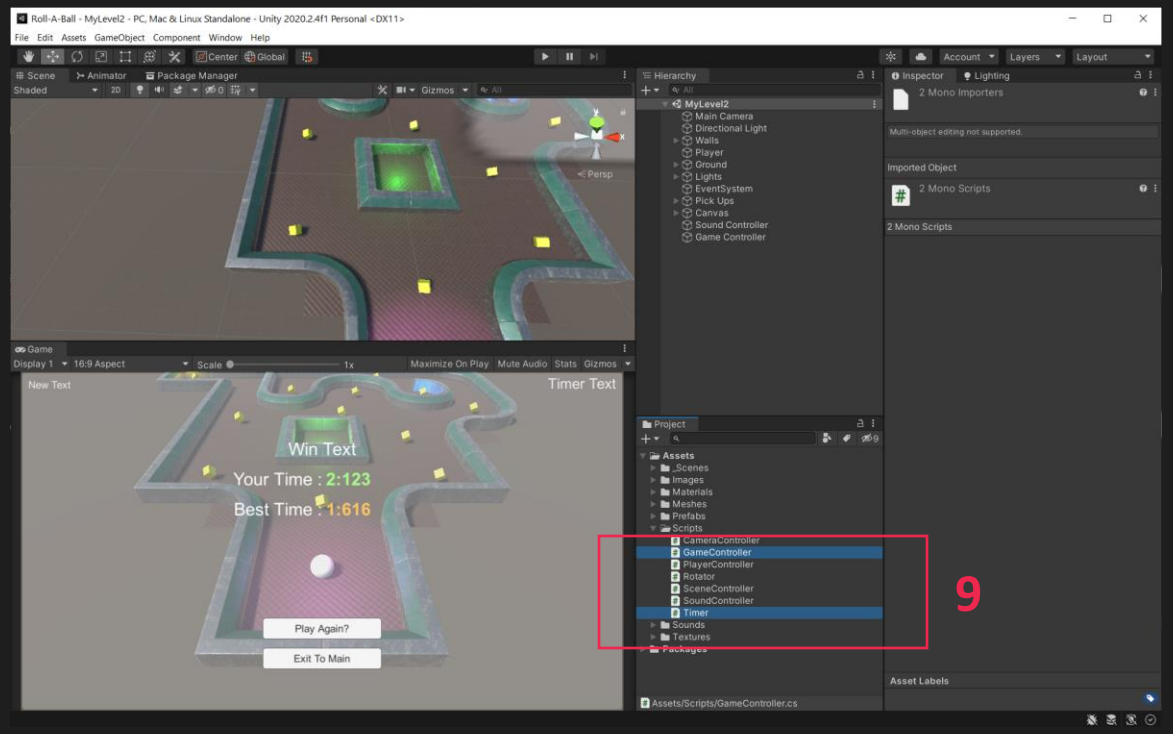
8. Duplicate the Your Time Text object, rename to Best Time Text and the child object Best Time Value. Position it below the Your Time Text



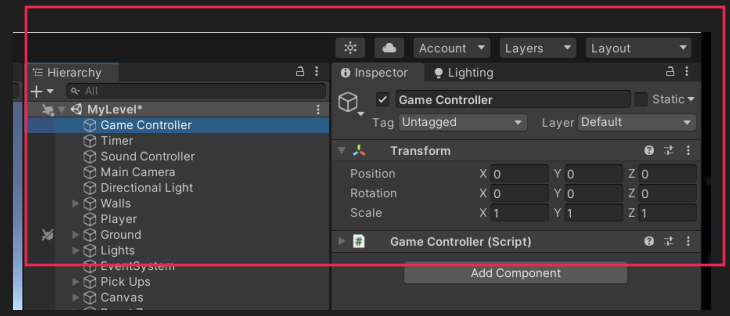


Creating the scripts

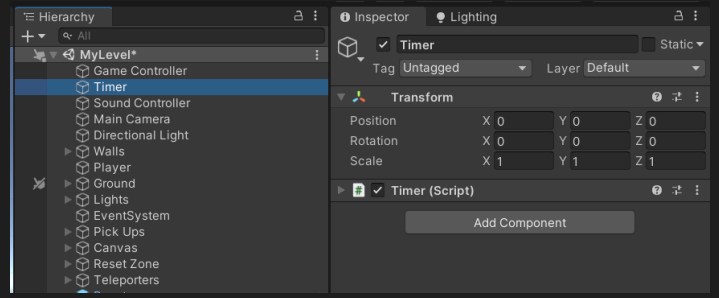
9. Create a script called GameController and a script called Timer



10. A. Create an empty GameObject in the scene and call it Game Controller. Zero out its position Drag the GameController script onto it



10. B. Create an empty GameObject in the scene and call it Timer. Zero out its position Drag the Timer script onto it

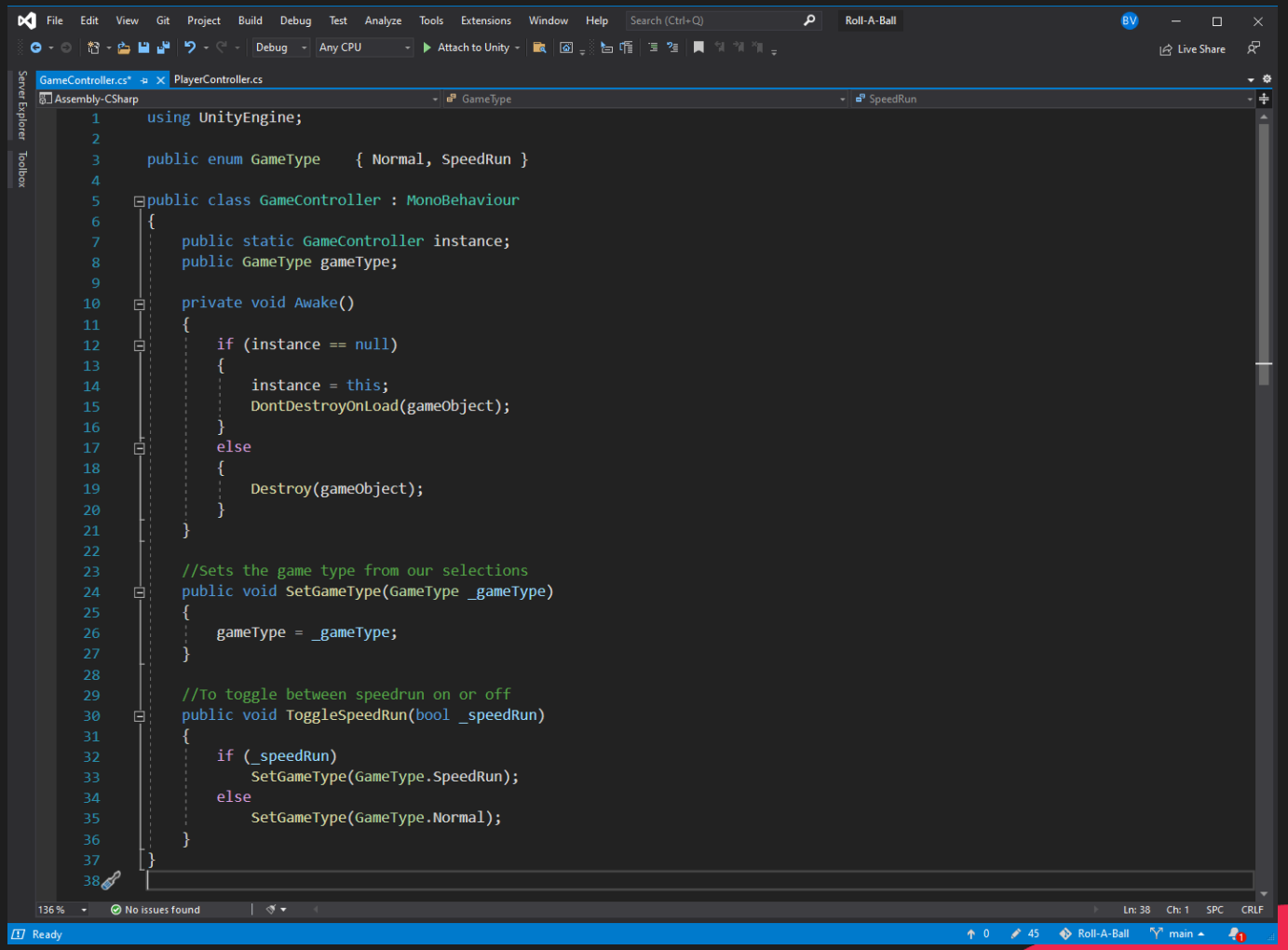


Modify the GameController script

11. Write out the GameController script as follows

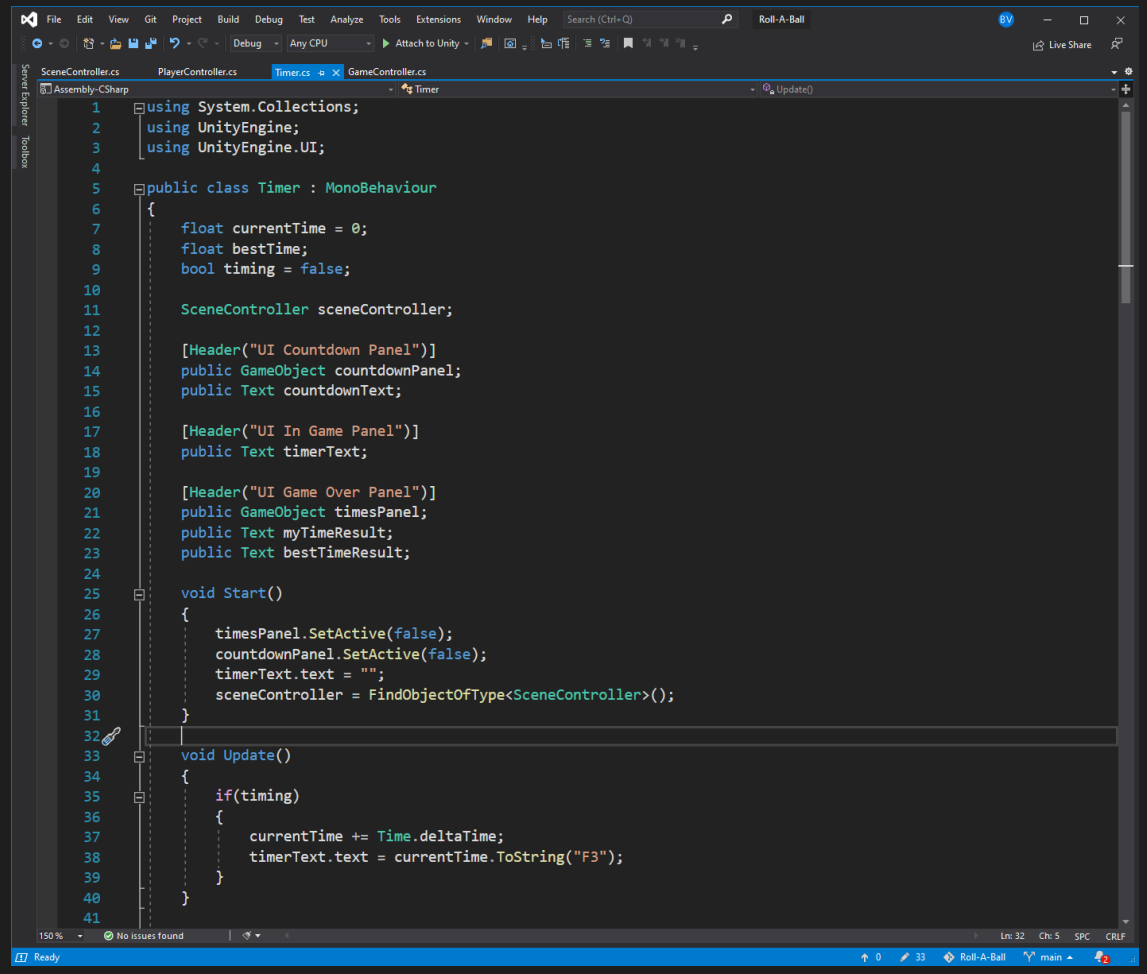
12. There is a lot of new syntax in here but don't worry if you don't fully understand it

NOTE: You may have already created this script in another module. If so, add the new content.



Modify the Timer script

13. Write out the Timer script as follows



```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

public class Timer : MonoBehaviour
{
    float currentTime = 0;
    float bestTime;
    bool timing = false;

    SceneController sceneController;

    [Header("UI Countdown Panel")]
    public GameObject countdownPanel;
    public Text countdownText;

    [Header("UI In Game Panel")]
    public Text timerText;

    [Header("UI Game Over Panel")]
    public GameObject timesPanel;
    public Text myTimeResult;
    public Text bestTimeResult;

    void Start()
    {
        timesPanel.SetActive(false);
        countdownPanel.SetActive(false);
        timerText.text = "";
        sceneController = FindObjectOfType<SceneController>();
    }

    void Update()
    {
        if(timing)
        {
            currentTime += Time.deltaTime;
            timerText.text = currentTime.ToString("F3");
        }
    }

    1 reference
    public IEnumerator StartCountdown()
    {
        yield return new WaitForEndOfFrame();
        if (PlayerPrefs.HasKey("BestTime"))
            bestTime = PlayerPrefs.GetFloat("BestTime" + sceneController.GetSceneName());
        else
            bestTime = 1000f;

        countdownPanel.SetActive(true);
        countdownText.text = "3";
        yield return new WaitForSeconds(1);
        countdownText.text = "2";
        yield return new WaitForSeconds(1);
        countdownText.text = "1";
        yield return new WaitForSeconds(1);
        countdownText.text = "Go!";
        yield return new WaitForSeconds(1);
        StartTimer();
        countdownPanel.SetActive(false);
    }

    1 reference
    public void StartTimer()
    {
        currentTime = 0;
        timing = true;
    }

    1 reference
    public void StopTimer()
    {
        timing = false;
        timesPanel.SetActive(true);
        myTimeResult.text = currentTime.ToString("F3");
        bestTimeResult.text = bestTime.ToString("F3");

        if (currentTime <= bestTime)
        {
            bestTime = currentTime;
            PlayerPrefs.SetFloat("BestTime" + sceneController.GetSceneName(), bestTime);
            bestTimeResult.text = bestTime.ToString("F3") + " !! NEW BEST !!";
        }
    }

    1 reference
    public bool IsTiming()
    {
        return timing;
    }
}
```

Modify the PlayerController script

14. Add the highlighted variable at the top

15. Add the following to the Start function

16. Add to the FixedUpdate function

17. Add to the WinGame function

NOTE: You may not have some other parts of this script (eg SoundController) if you have not done that module

```
private int pickupCount;

//Controllers
SoundController soundController;
GameController gameController;
Timer timer;
```

```
void Start()
{
    rb = GetComponent<Rigidbody>();
    count = 0;
    pickupCount = GameObject.FindGameObjectsWithTag("Pick Up").Length;
    gameOverScreen.SetActive(false);
    SetCountText();
    winText.text = "";
    soundController = FindObjectOfType<SoundController>();

    gameController = FindObjectOfType<GameController>();
    timer = FindObjectOfType<Timer>();
    if (gameController.gameType == GameType.SpeedRun)
        StartCoroutine(timer.StartCountdown());
}
```

```
void WinGame()
{
    gameOverScreen.SetActive(true);
    winText.text = "You Win!";
    soundController.PlayWinSound();

    if(gameController.gameType == GameType.SpeedRun)
        timer.StopTimer();

}
```

```
void FixedUpdate()
{
    if (gameController.gameType == GameType.SpeedRun && !timer.IsTiming())
        return;

    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");
    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
    rb.AddForce(movement * speed);
}
```

Making the connections

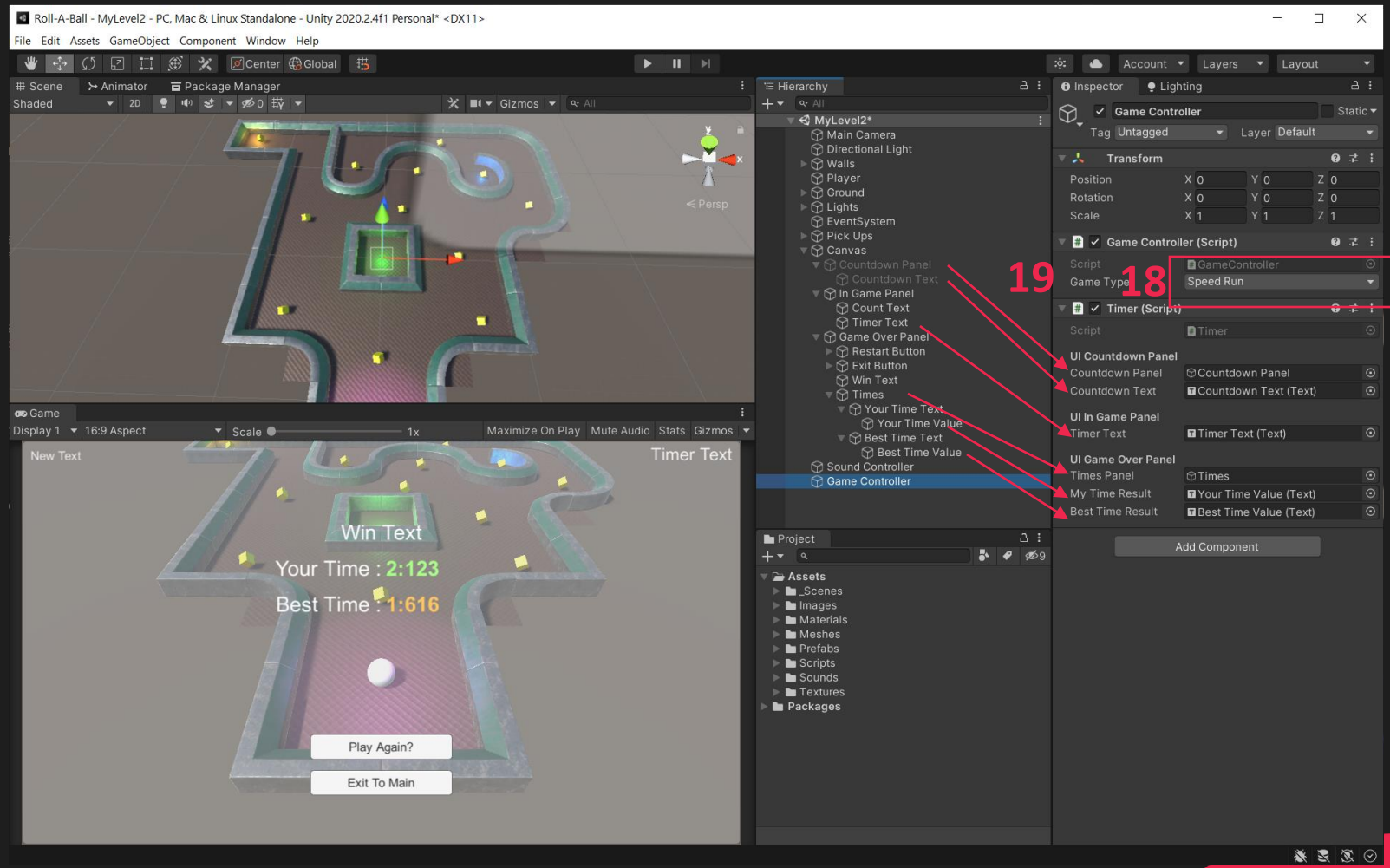
NOTE: The image shows both the Timer and GameController on the same object when they should be separate as in steps 10A & 10B

18. Change the Game Type on the GameController to Speed Run

19. Drag over all the GameObjects to the appropriate slots on the Timer object

RUN THE GAME AND YOU SHOULD NOW HAVE SPEED RUN WORKING

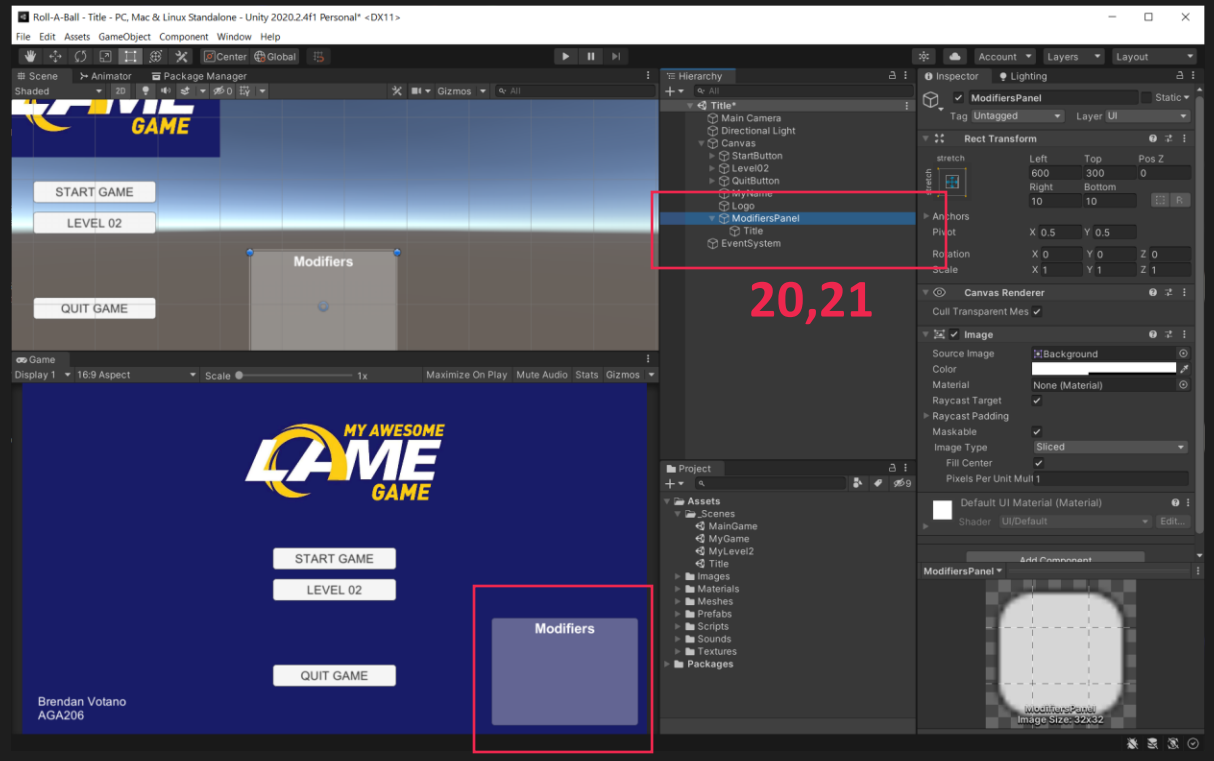
We now need to add the option to toggle it from our Title scene

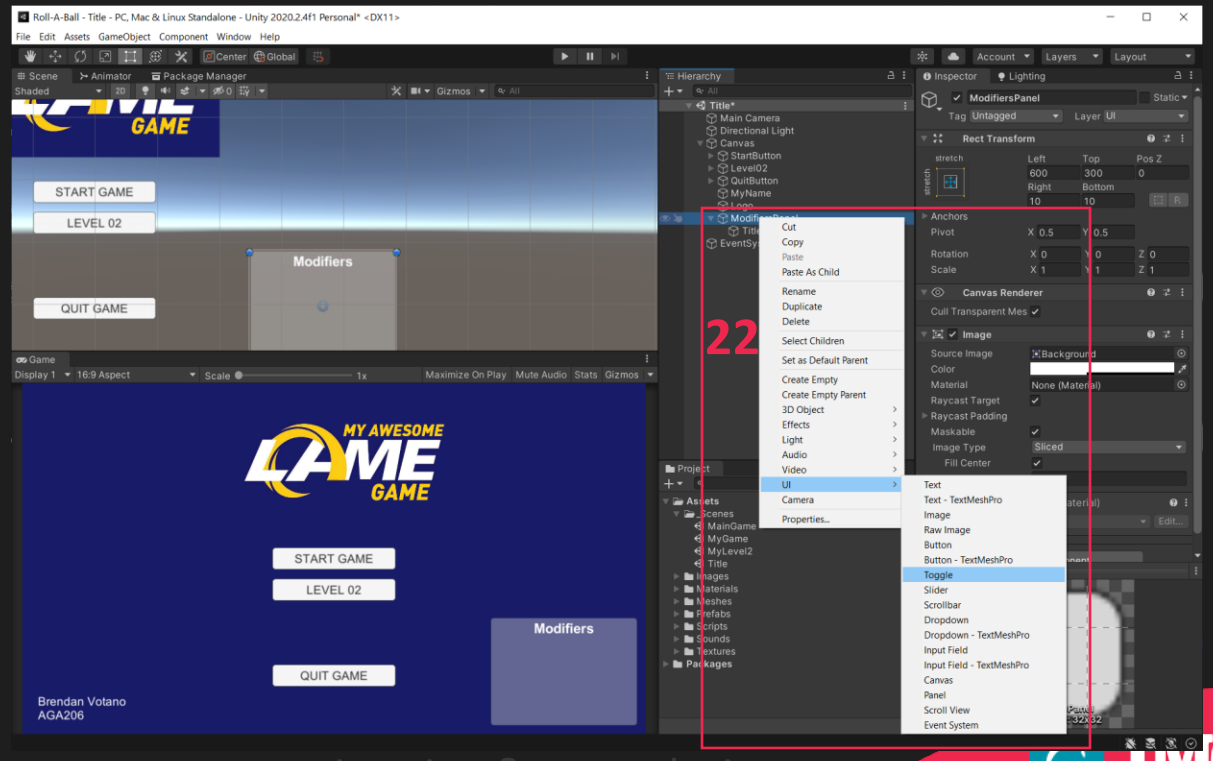


NOTE: The order of objects in your canvas may be different to this image

Adding to the Title Scene

20. Back in our Title scene, in our Canvas create a Panel called ModifiersPanel and position it like so.
21. Add a UI text to this panel called Title and change the text to Modifiers
22. Highlight the ModifiersPanel and then right click to create a new UI > Toggle



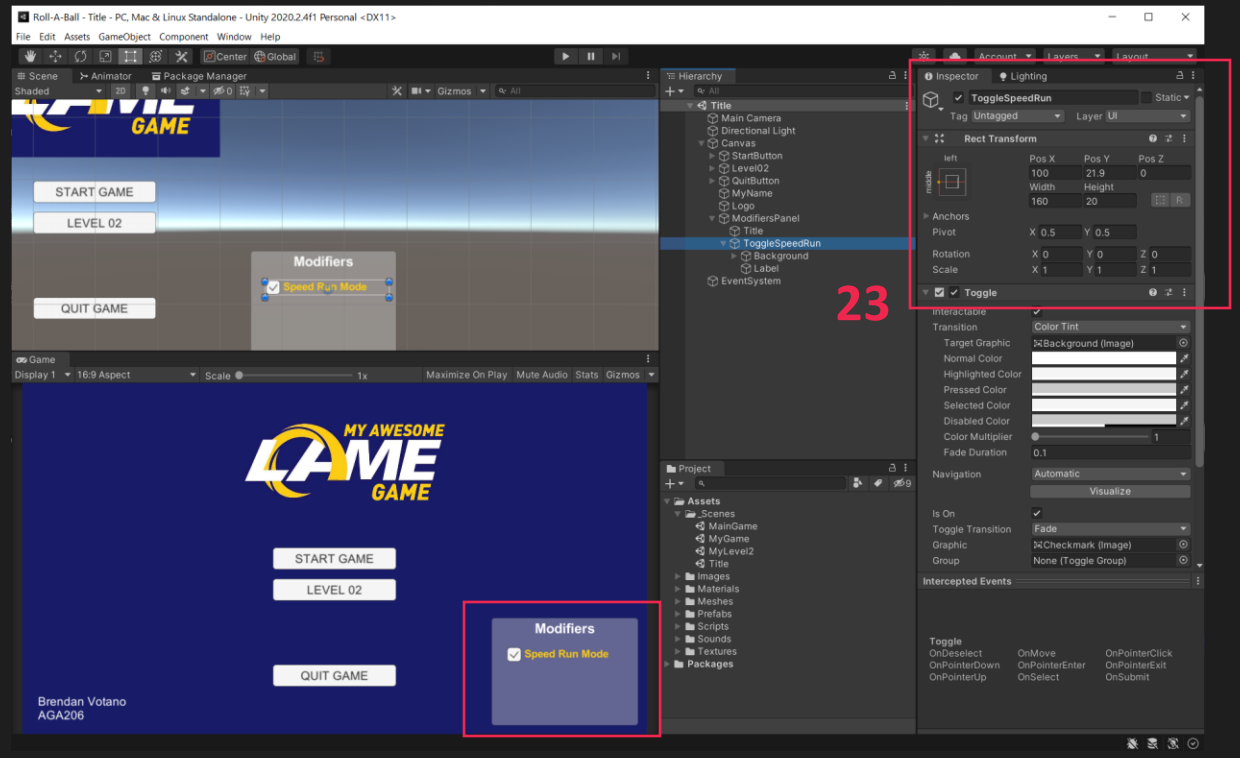


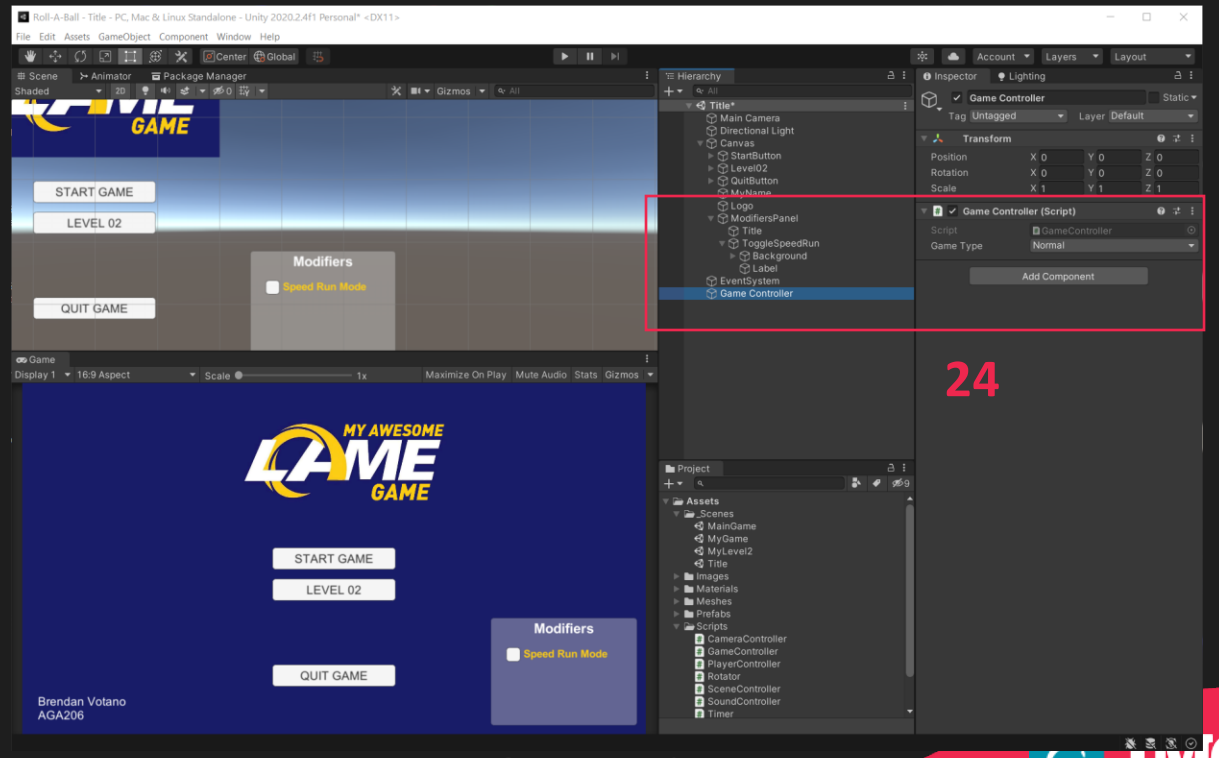
NOTE: You may or may not have some of these elements already depending on other modules

Adding to the Title Scene

23. Position and change colours, font, etc to something like the following

24. Create an empty GameObject at the root called Game Controller and drag on the GameController script

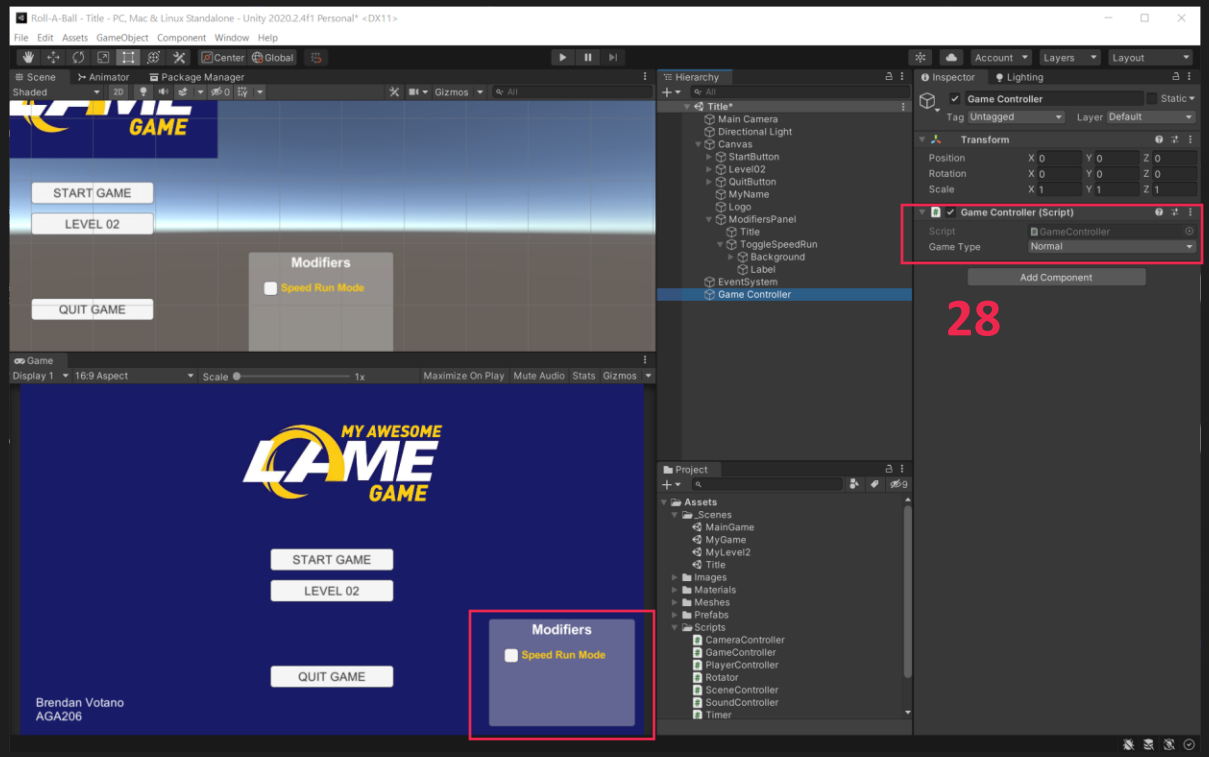




NOTE: You may or may not have some of these elements already depending on other modules

Getting the Toggle to work

25. N/A
26. N/A
27. N/A
28. When you play the scene and change the toggle, you should see it update on the GameController
    - However, there is an issue we must fix. The toggle defaults to on every time we run the game.
    - We need to get it to set according to the saved GameController settings



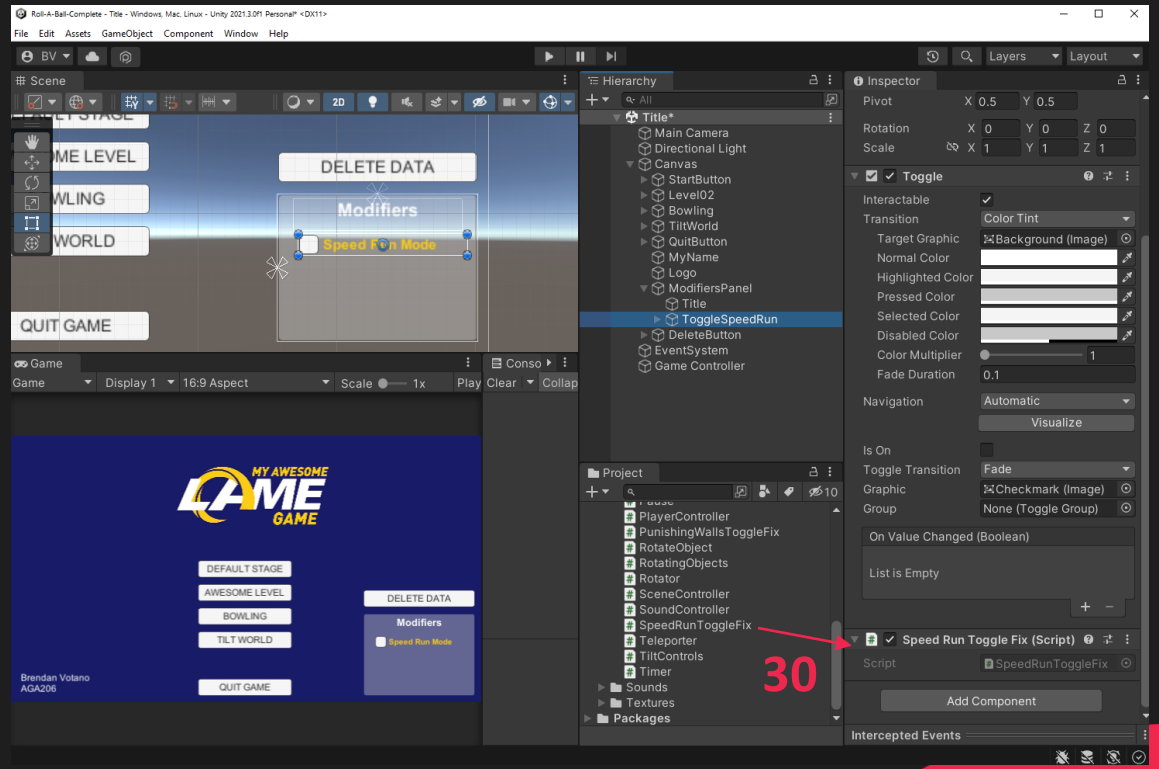
NOTE: You may or may not have some of these elements already depending on other modules

Hacky Toggle Fix

29. Create a script called SpeedRunToggleFix and fill it in EXACTLY as follows

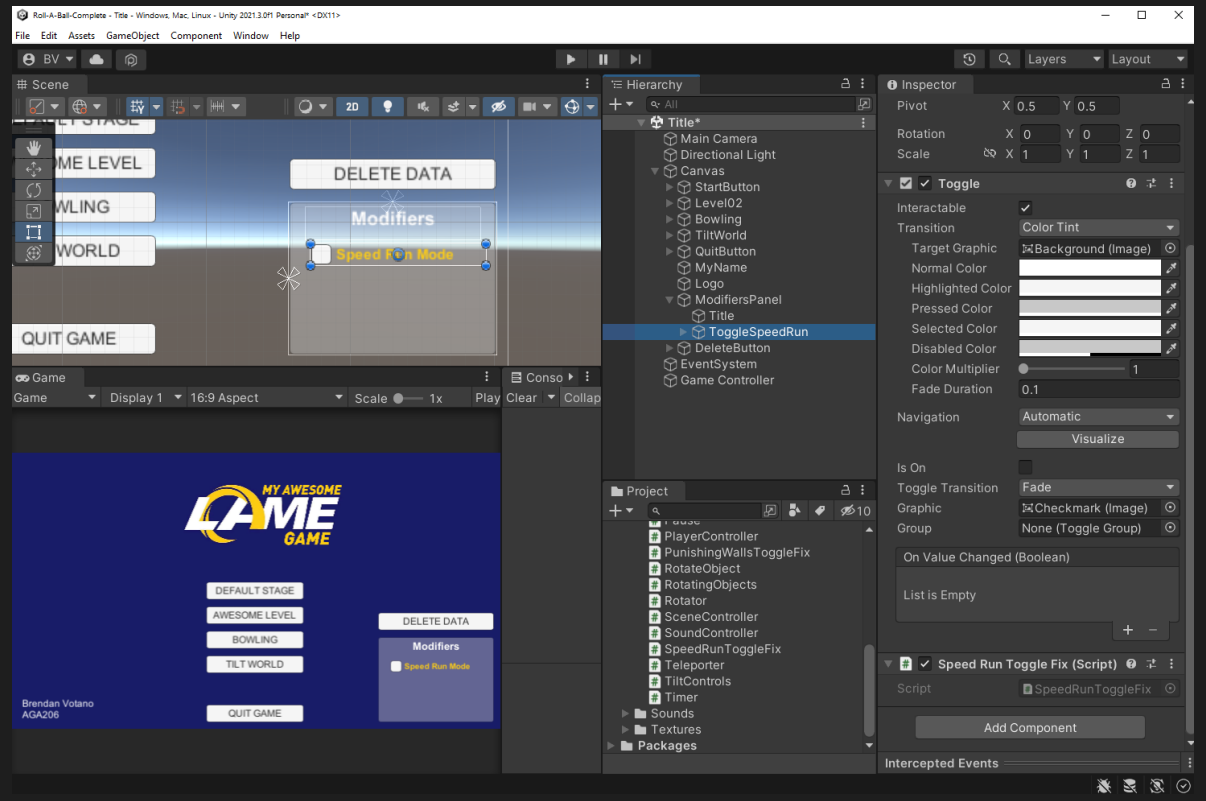
```
SpeedRunToggleFix.cs | SceneController.cs | CameraController.cs | Timer.cs* | GameController.cs | PlayerController.cs
Assembly-CSharp                                  SpeedRunToggleFix

1   using System.Collections;
2   using UnityEngine;
3   using UnityEngine.UI;
4
    0 references
5   public class SpeedRunToggleFix : MonoBehaviour
6   {
7       GameController gameController;
8       Toggle toggle;
9
        0 references
10      void Start()
11      {
12          gameController = FindObjectOfType<GameController>();
13          toggle = GetComponent<Toggle>();
14          StartCoroutine(FixSpeedRunToggle());
15      }
16
        1 reference
17      IEnumerator FixSpeedRunToggle()
18      {
19          yield return new WaitForEndOfFrame();
20          if (gameController.gameType == GameType.SpeedRun)
21              toggle.isOn = true;
22          else
23              toggle.isOn = false;
24
25          toggle.onValueChanged.AddListener((value) => gameController.ToggleSpeedRun(toggle.isOn));
26      }
27  }
```

30. Drag the SpeedRunToggleFix script to the ToggleSpeedRun object



DONE!!!

- The speed mode toggle now should apply to every level in your game
- You still need to ensure that the UI canvas is setup in each level though as in Steps 1 – 19 though you do not need to redo any scripts
- Alternatively, once you have this scene setup, duplicate it as the base for your other new scenes so that all the connections are already there



7. New Look and Feel

2pts

Pre-requisite: None

Create a set of textures/materials that create new look and feel to the game.
Some examples:

- Neon World (Tron)
- Elemental World (ice, fire, etc)
- Steampunk
- Mini Golf

You should also change the look of the ball and/or collectibles accordingly
Example: Ball > Golf ball, Collectibles > Little golf flags

Design: Modelling, Texturing, Emission maps

Programming: None

Tasks

1. Create a new model
2. Create new textures

8. World Tilt Mode

2pts

Pre-requisite: Start and Game Over Screen

Instead of controlling the ball, players control the world, which is titled on the origin point when the player uses the input axis. Think of those physical games where you are trying to get the marble into a hole.

This game world should be a new button option on the title screen.

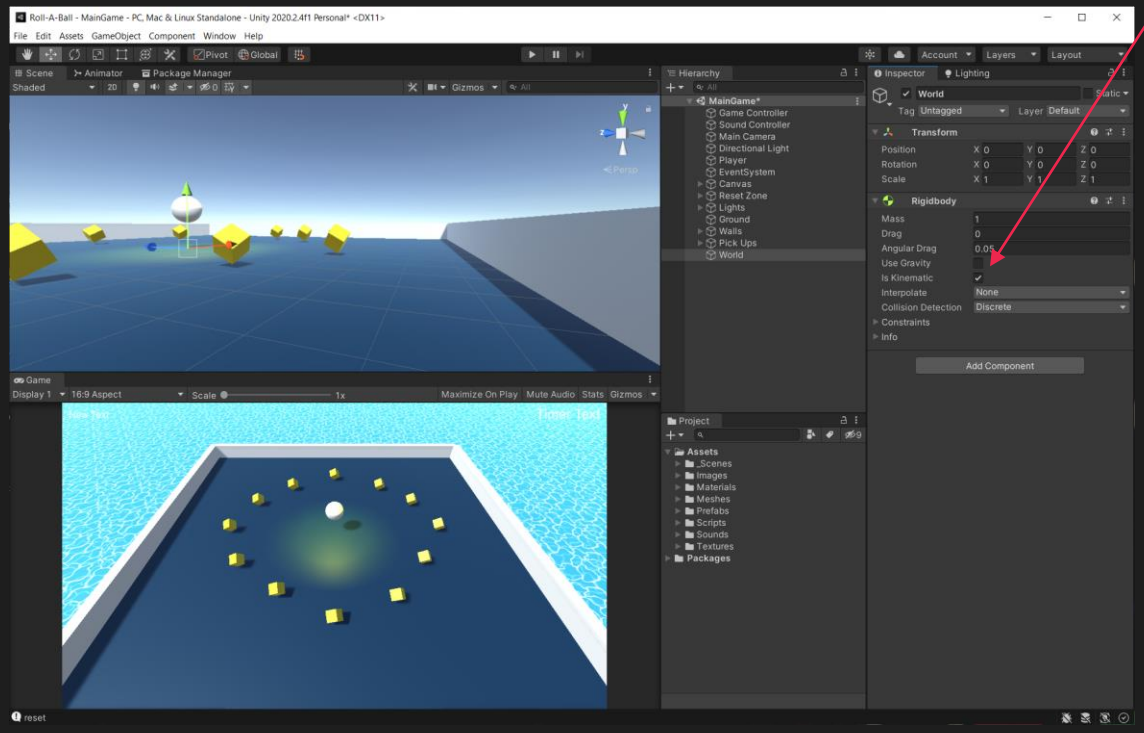
Design: Level to take advantage of the new control

Programming: Player input, reconfiguration of certain objects

Setup the world in Unity

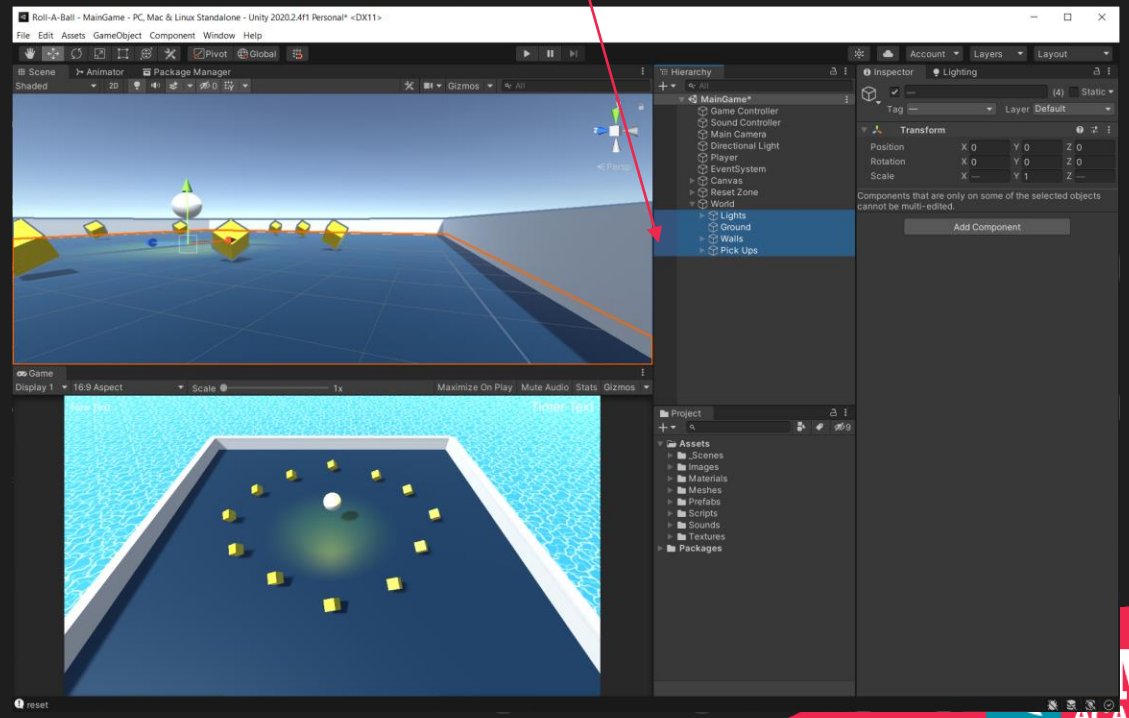
1. Duplicate a scene and call it TiltWorld. Modify this stage to have a cool design appropriate for the play type.

2. Create an empty GameObject called World and zero it out.



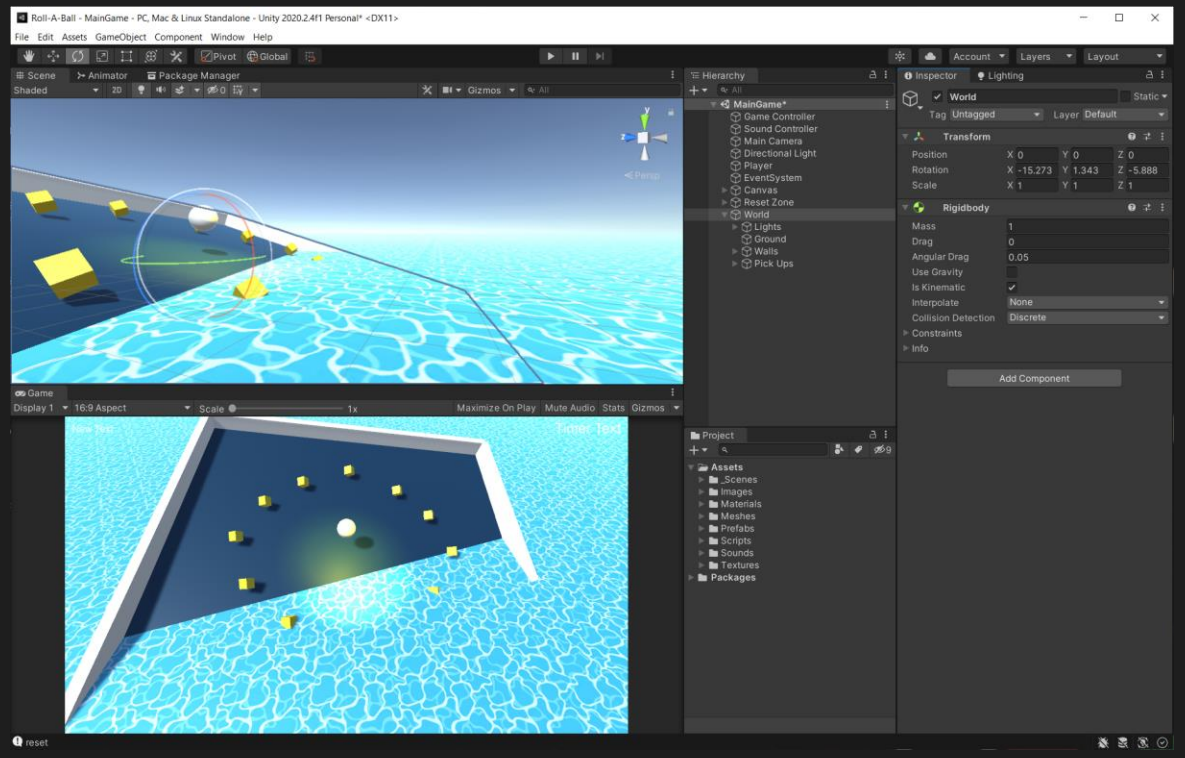
3. Add a Rigidbody component, uncheck Use Gravity and check Is Kinematic

4. Drag all the objects from our scene that need to tilt into the new World object

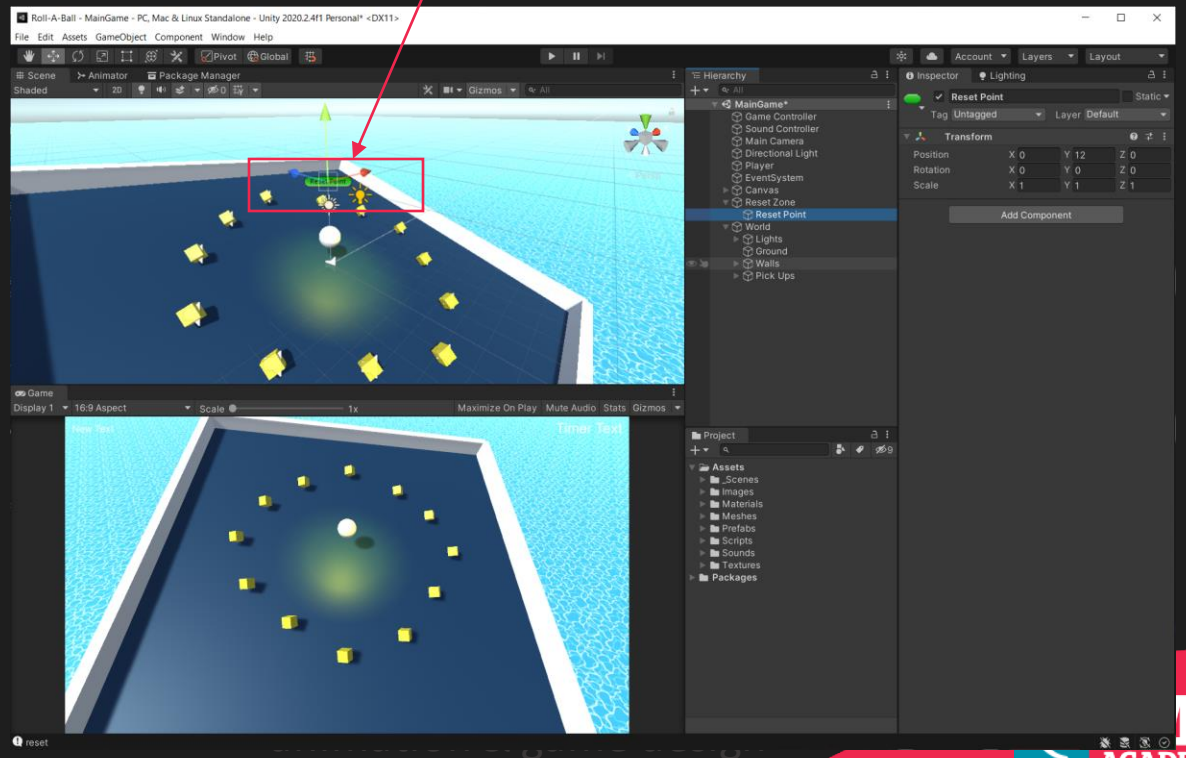


Setup the world in Unity

5. Rotate the World object and you will most likely clip into the Reset Zone mesh and collider



6. To fix this, move the Reset Zone a lower than the World Objects lowest point (rotate to 90 degrees to see)
7. Ensure the Reset Point is moved to be just above the World zero point

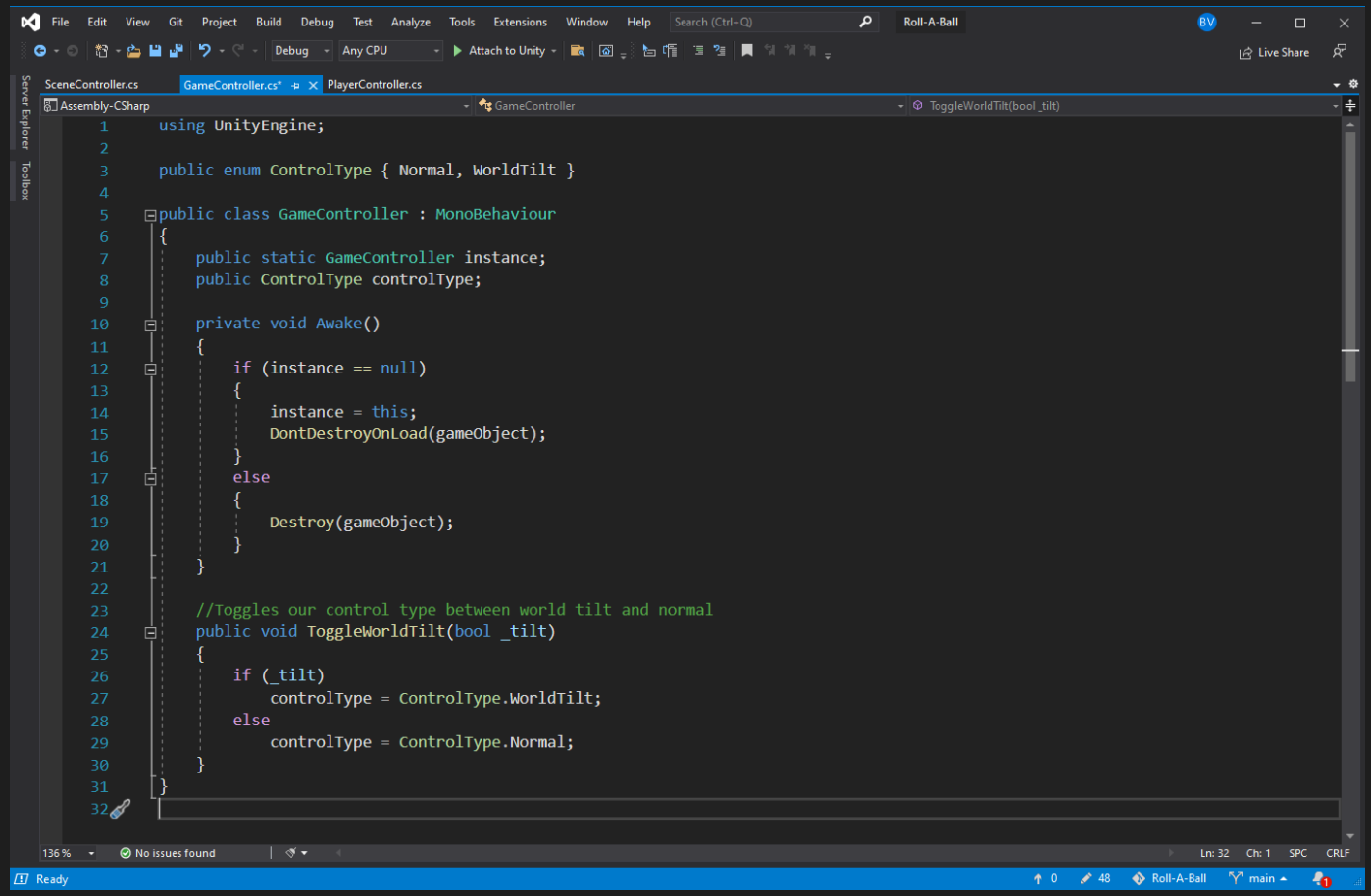


Create some new scripts

8. Create a script called GameController.
   - If you have done the Speed Run module you will already have this setup.

9. Add the following to the script.
   - If you have done Speed Run, add in the extra lines where they belong

10. This setting is an option for us in the game to choose a control type to be regular play or with a tilt world



Create some new scripts

11. Create a new script called TiltControls and copy it EXACTLY as follow

12. Since our Player moves with physics, we need to rotate our world with physics too.

13. In the SceneController script set up in one of the mandatory modules, add the following line to the ToTitleScreen function

GameController.cs | TiltControls.cs | SoundController.cs | PlayerController.cs

Assembly-CSharp | TiltControls | Start()

```
using UnityEngine;

public class TiltControls : MonoBehaviour
{
    public float speed = 20f;
    Rigidbody rb;

    private void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    private void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 moveVector = new Vector3(moveVertical * speed, 0, -moveHorizontal * speed);
        Quaternion deltaRotation = Quaternion.Euler(moveVector * Time.fixedDeltaTime);
        rb.MoveRotation(rb.rotation * deltaRotation);
    }
}
```

```
//Loads out Title scene. Must be called Title exactly
public void ToTitleScene()
{
    GameController.instance.controlType = ControlType.Normal;
    SceneManager.LoadScene("Title");
}
```

Add to our PlayerController script

14. Add a reference to our GameController in the variable declaration section at the top
    - NOTE: You may not have some of these such as SoundCountroller or timer if you have not done certain modules

15. Add the line to find the GameController in the start function

16. Add the lines to the FixedUpdate function that will exit the function should our control type be WorldTilt

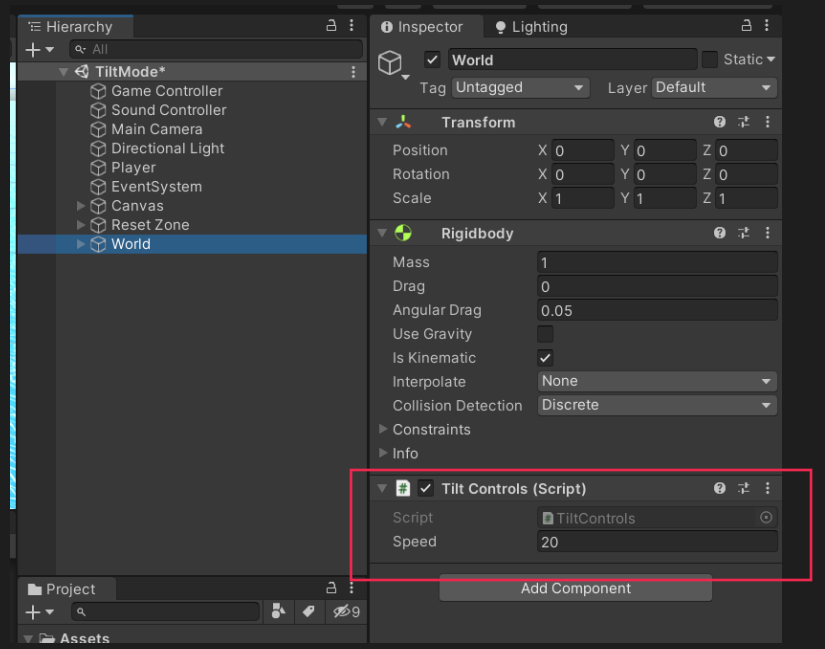
```
    bool grounded = true;

    //Controllers
    SoundController soundController;
    GameController gameController;
    Timer timer;

void Start()
{
    rb = GetComponent<Rigidbody>();
    count = 0;
    pickupCount = GameObject.FindGameObjectsWithTag("Pick Up").Length
                + GameObject.FindGameObjectsWithTag("Bowling Pin").Length;
    gameOverScreen.SetActive(false);
    SetCountText();
    winText.text = "";
    resetPoint = GameObject.Find("Reset Point");
    gameController = FindObjectOfType<GameController>();
}
```

```
void FixedUpdate()
{
    if (resetting)
        return;

    if (gameController.controlType == ControlType.WorldTilt)
        return;
```

Add to our PlayerController script

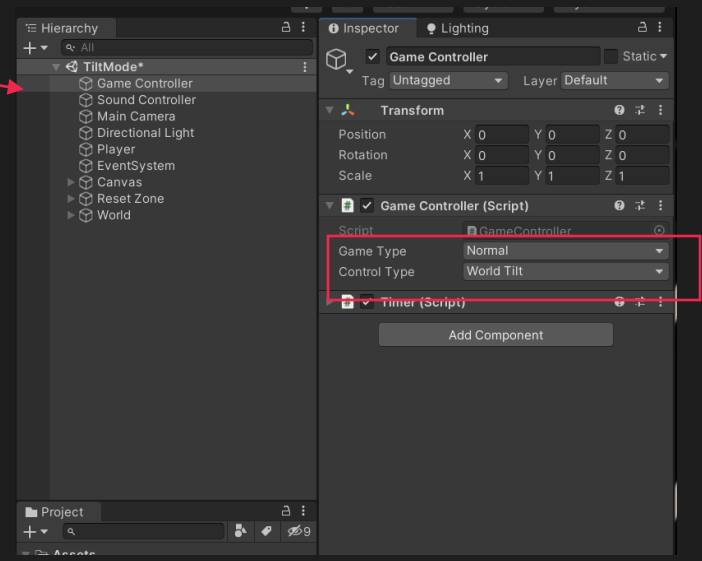
17. Add the TiltControls script to the World object

18. Create an empty Game Object, called Game Controller and zero it out
    - If you have done the Speed Run module you will already have this setup.



17

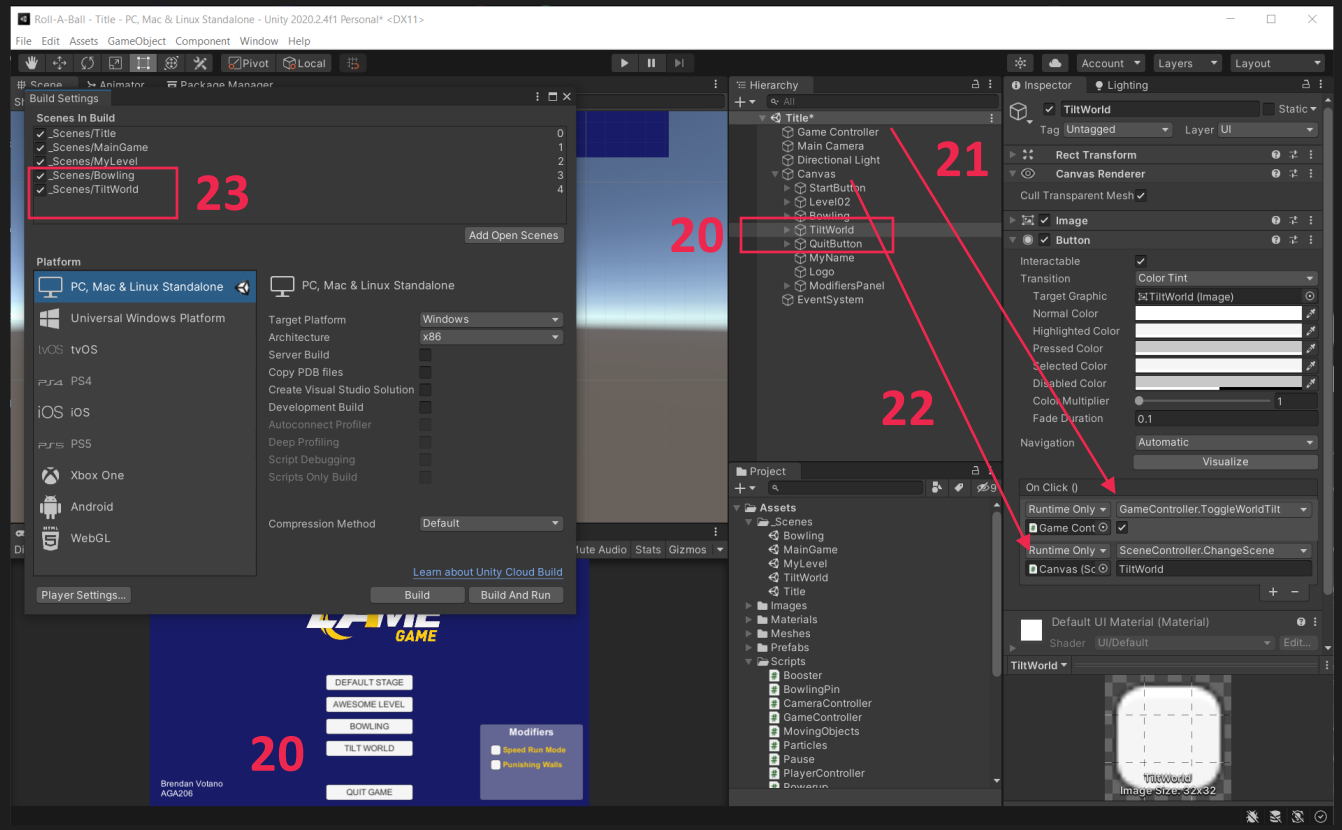
19. Change the Control Type to World Tilt in levels you wish to control this way.
    - NOTE: If you have not done the Speed Run module, you will not see the Timer and Game Type



19

- NOTE: You will need to make sure you have the GameController script in every scene now. For the non-tilt scenes, keep the Control Type to Normal

Add to the title scene



20. Duplicate one of the level buttons and call it Tilt World. Position it appropriately

21. In the Tilt World buttons OnClick event, drag the GameController object to the slot and select GameController > ToggleWorldTilt from the dropdown. Check the box

22. Create a second OnClick event, drag the Canvas object to the slot and select SceneController > ChangeScene. Put in text to the EXACTLY the same as the Tilt Worlds scene name

NOTE: The order of the OnClick events must be as in the steps and image or it wont work

23. Make sure you add the new scene to the Build Settings!

24. Congratulations. You are now DONE!!!

NOTE: You may or may not have some of these elements already depending on other modules

9. Sound

1pt

Pre-requisite: None

Play sounds on triggered events (eg. hitting a wall, collecting an item, etc).
As you develop more modules, add sounds where appropriate.
You may source these audio files online.

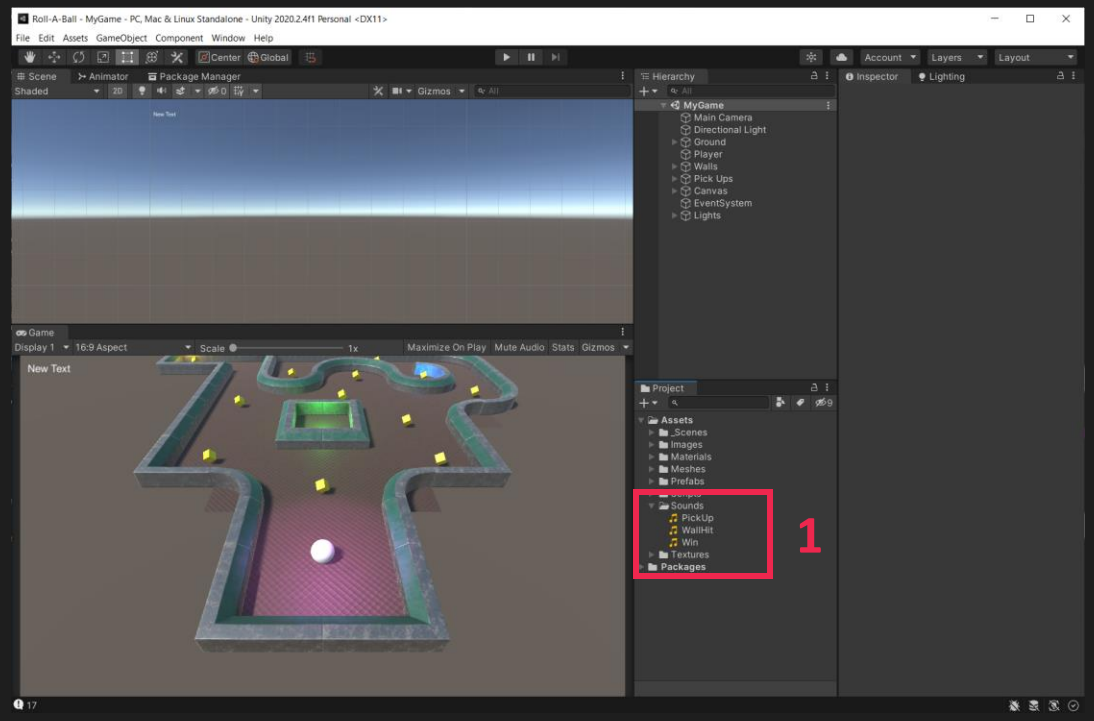
Design: None

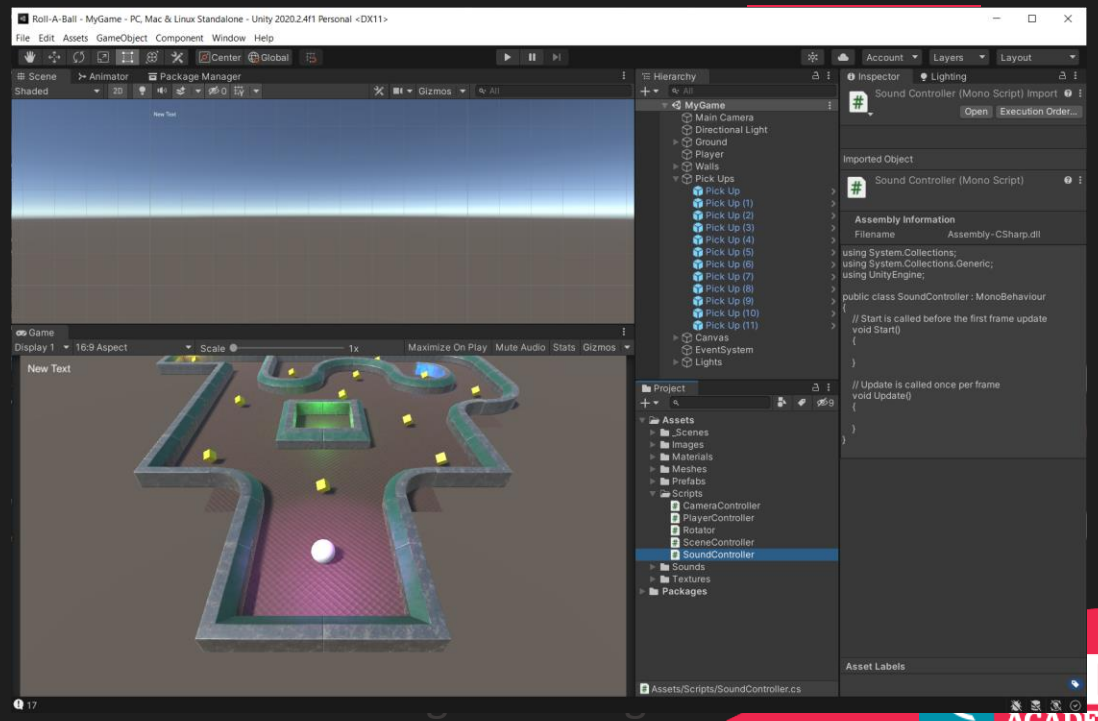
Programming: Trigger Events

Get Some Sound Effects

1. Create a folder called Sounds
   Find sounds for the following (wav, mp3, ogg)
   - Hitting a wall (Call it WallHit)
   - Collecting a Pick Up (call it PickUp)
   - Wining the game (Call it Win)

2. In the Scripts folder, create a script called SoundController





Write our SoundController script

3. Copy out the following into our SoundController script (This is all one script split over two images)
4. We are creating a generic PlaySound() function to handle our pickup and other sounds
5. Our PlayCollisionSound() function will play objects we collide with that were passed into the function

```
using UnityEngine;

public class SoundController : MonoBehaviour
{
    public AudioClip pickupSound;
    public AudioClip winSound;

    AudioSource audioSource;

    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    public void PlayPickupSound()
    {
        PlaySound(pickupSound);
    }

    public void PlayWinSound()
    {
        PlaySound(winSound);
    }
```

```
    void PlaySound(AudioClip _newSound)
    {
        //Set the audiosources audioclip to be the passed in sound
        audioSource.clip = _newSound;
        //Play the audiosource
        audioSource.Play();
    }

    public void PlayCollisionSound(GameObject _go)
    {
        //Check to see if the collided object has an Audiosource.
        //This is a failsafe in case we forgot to attach one to our wall
        if (_go.GetComponent<AudioSource>() != null)
        {
            //Play the audio on the wall object
            _go.GetComponent<AudioSource>().Play();
        }
    }
}
```

4

5

Modify our PlayerController script

6. Add a reference to our SoundController in the variables section

7. Assign the soundController variable in the Start function using the inbuilt FindObjectOfType<> function

8. In our WinGame function, we call the soundControllers PlayWinSound function to play the appropriate sound
   - The WinGame function was implemented in the mandatory title screen module

6

```
private int count;
private int pickupCount;    //The num

//Controllers
SoundController soundController;
```

7

```
void Start()
{
    rb = GetComponent<Rigidbody>();
    count = 0;
    pickupCount = GameObject.FindGameObjectsWithTag("Pick Up").Length;
    gameOverScreen.SetActive(false);
    SetCountText();
    winText.text = "";
    soundController = FindObjectOfType<SoundController>();
}
```

8

```
void WinGame()
{
    gameOverScreen.SetActive(true);
    winText.text = "You Win!";
    soundController.PlayWinSound();
}
```

Modify our PlayerController script

9. In Our OnTriggerEnter function, we add a call to the soundControllers PlayPickupSound function

10. We create a new OnCollisionEnter function.
This behaves similar to OnTriggerEnter but is listening for a collision.
If we collide with something tagged “Wall”, we play the soundControllers PlayCollisionSound function passing in the object we collided with.

```
void OnTriggerEnter(Collider other)
{
    if(other.gameObject.CompareTag("Pick Up"))
    {
        other.gameObject.SetActive(false);
        count = count + 1;
        SetCountText();
9       soundController.PlayPickupSound();
    }
}
```

```
10  private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Wall"))
        {
            soundController.PlayCollisionSound(collision.gameObject);
        }
    }
```

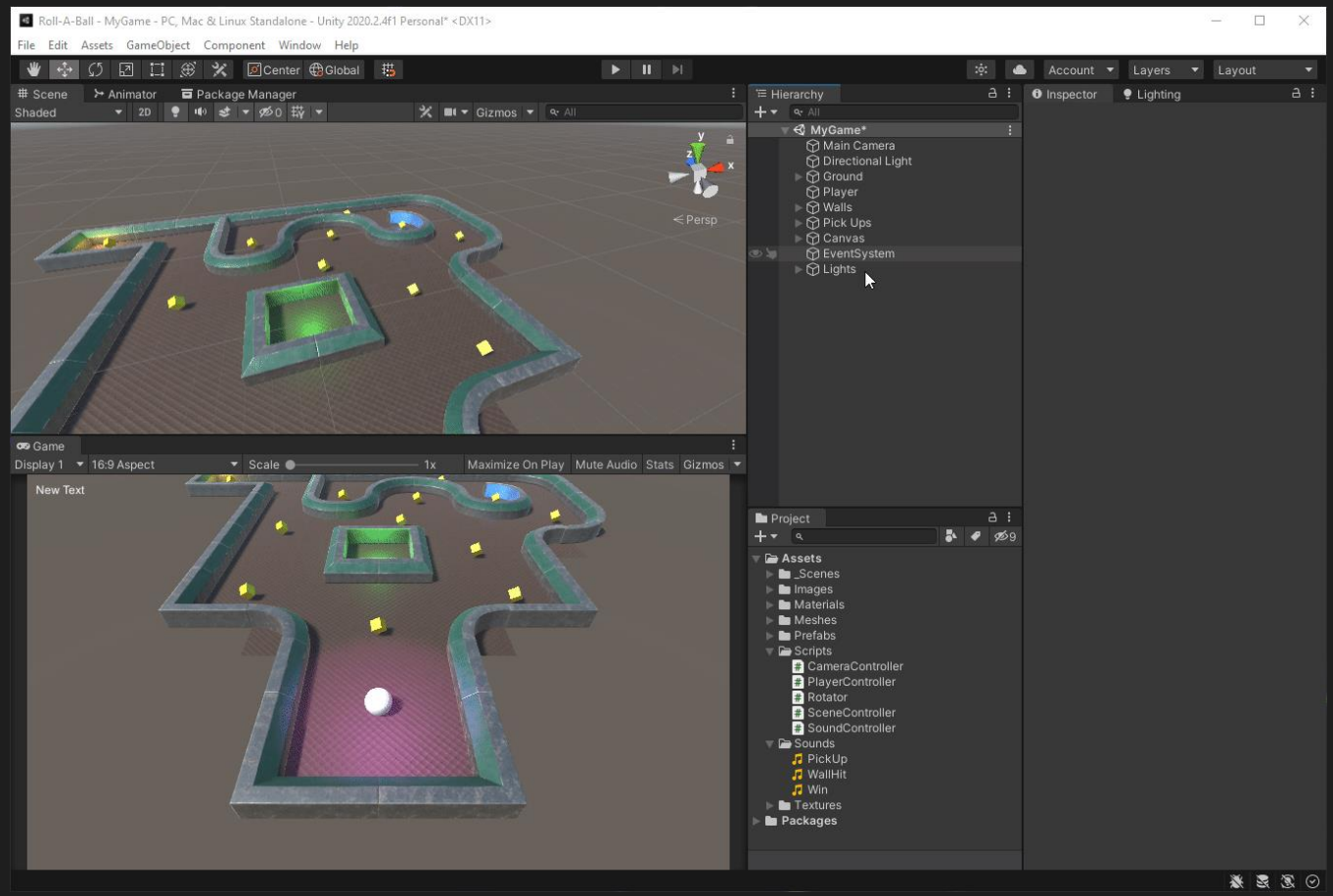
Get Some Sound Effects

11. Create an empty GameObject in the hierarchy called Sound Controller, zero it out and drag on the SoundController script.

12. Drag the Pick Up and Win sounds to the appropriate slots

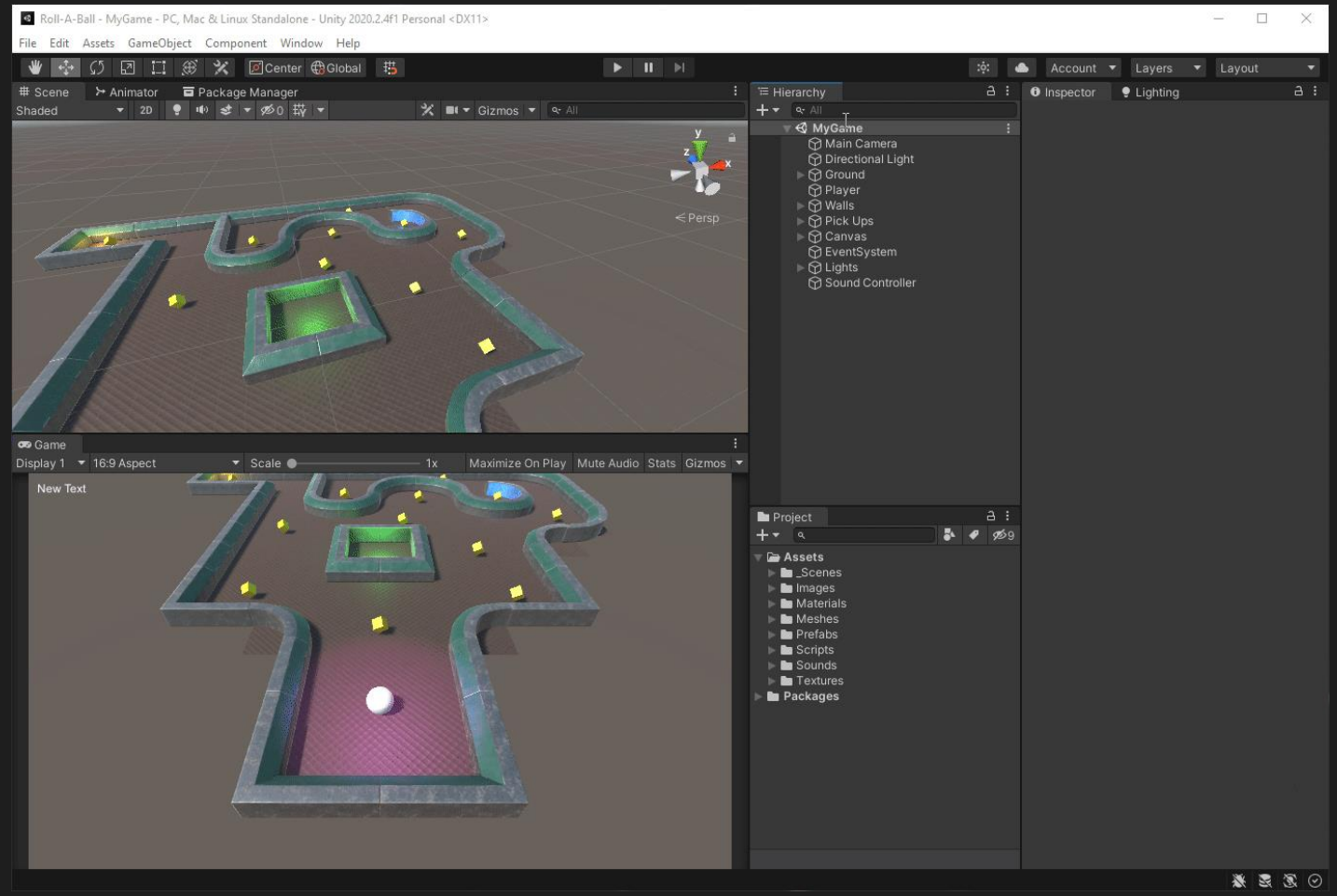
13. Add an AudioSource component to the Sound Controller through the Add Component dropdown

14. Turn off Play On Awake



Setup the Walls

15. Open a wall prefab from the Project folder.
16. Add an AudioSource component
17. Turn off PlayOnAwake
18. Drag the WallHit sound to the AudioClip slot on the AudioSource
19. Apply the tag “Wall” to the prefab
    - The first time you will need to set this tag up but afterwards you can just select it from the dropdown
20. Save the prefab
    - If you have autosave on, you will skip this step
21. Repeat on all the Prefabs used as walls



DONE!!!

- You should now have basic sound in your game.

- Expand upon the sounds as you add further modules. They should all follow a similar method to implement as these.

10. Particles

1pt

Pre-requisite: Start and Game Over Screen

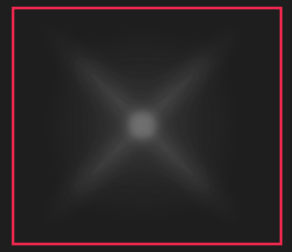
Add some particle effects to the game world to create a better atmosphere.
A new particle effect will play when key conditions are met such as collecting a pickup.

Design: Particle effects and textures

Programming: Instantiating

Setup the particle material

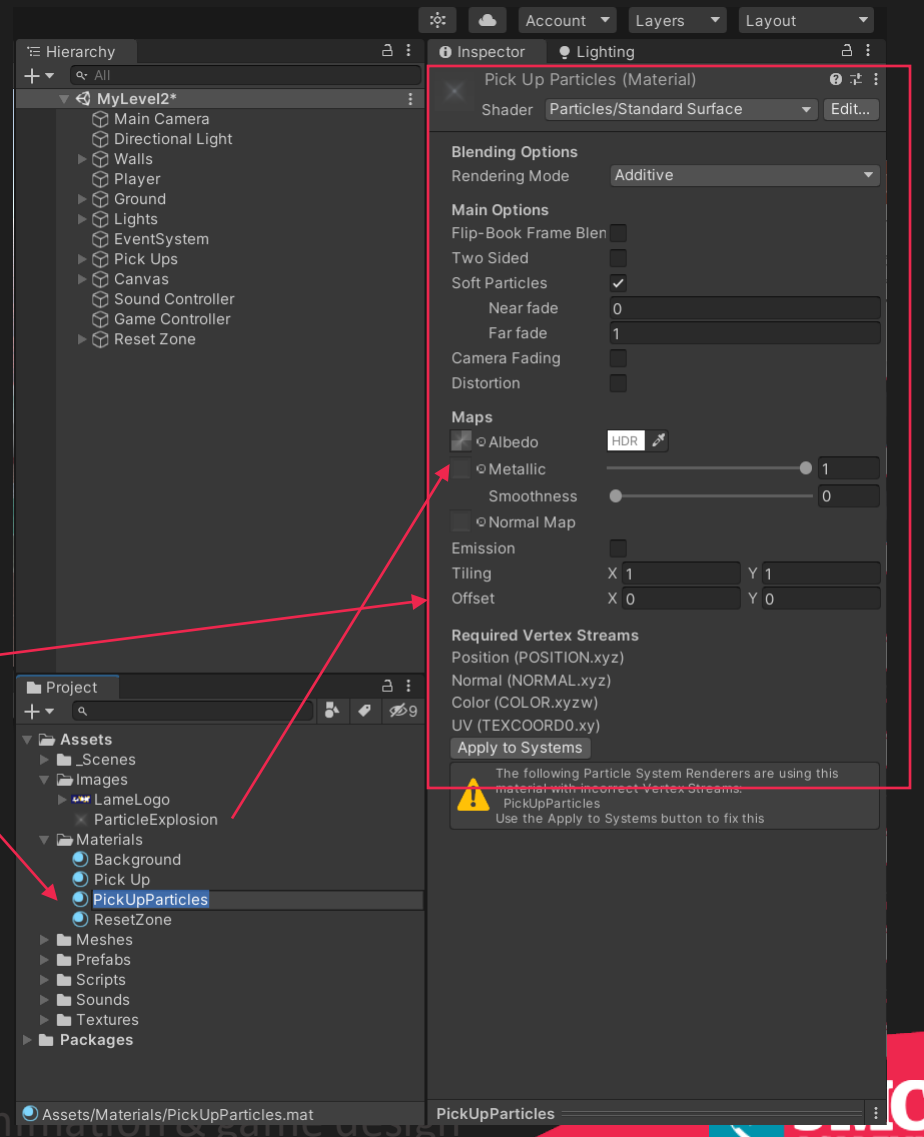
1. Find/Create a texture to use as the particle and save to Images folder. Ideally it should be white on a transparent background. The image is an example of one



2. Create a new Material called PickUpParticles in the Materials folder

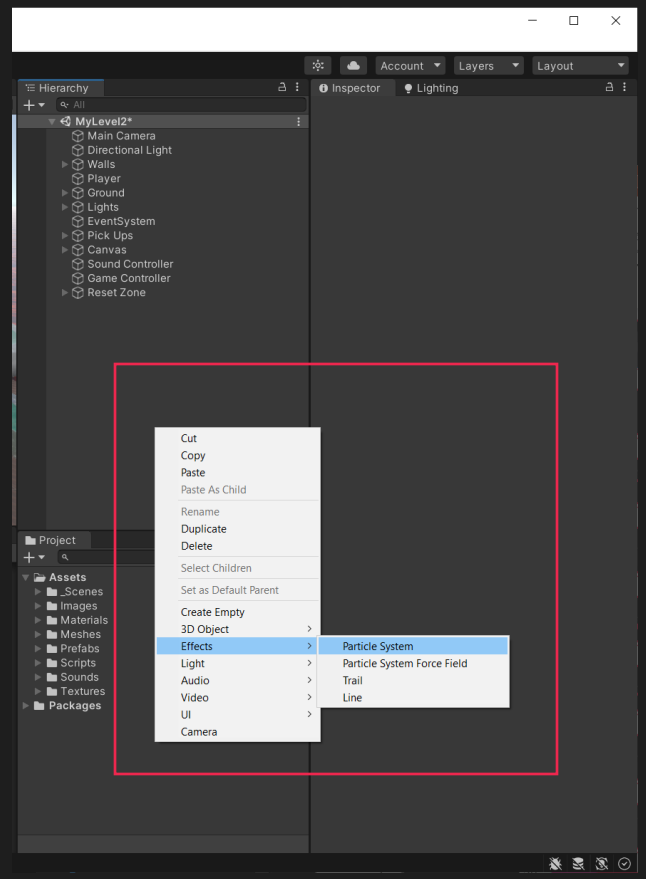
3. Change the Materials settings to the same as the side

4. Drag the Texture to the Albedo slot on the Material

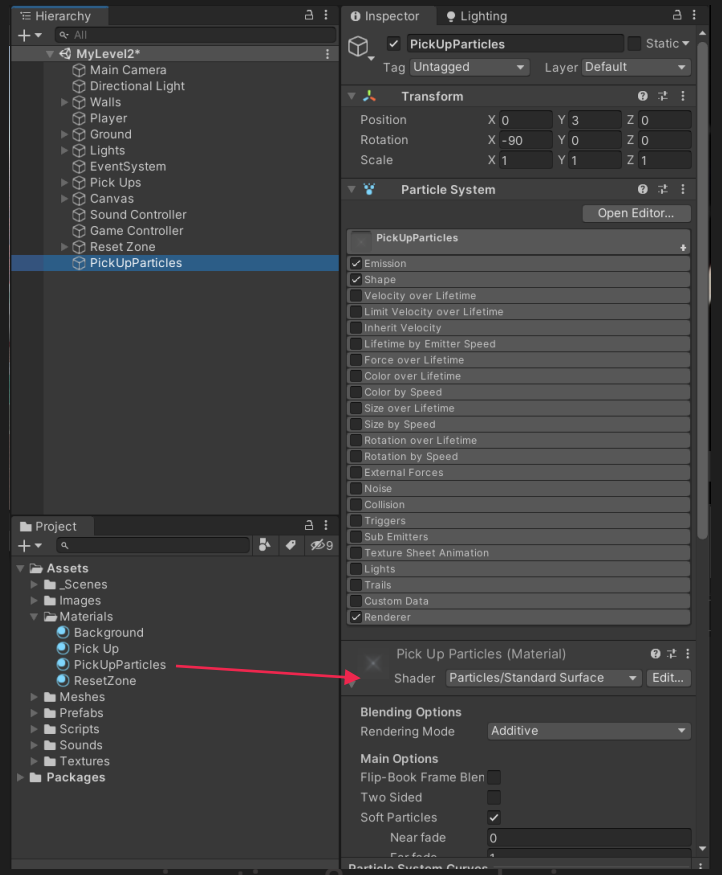


Setup the particle object

5. Create a new Particle System by right clicking in the hierarchy, then Effects > Particle System.
Name this PickUpParticles

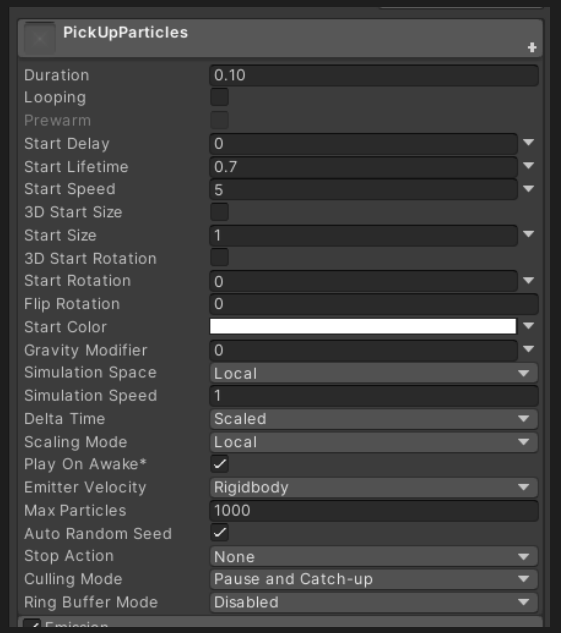


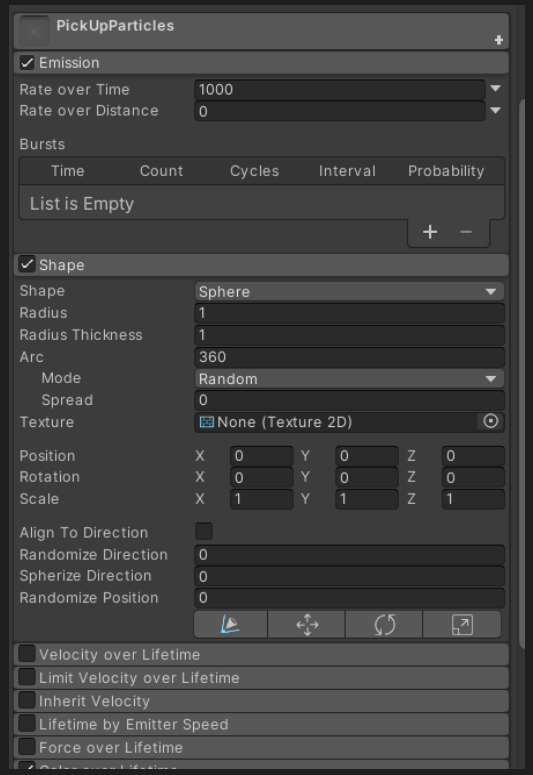
6. Drag the PickUpParticles Material onto the bottom of the new Particle System

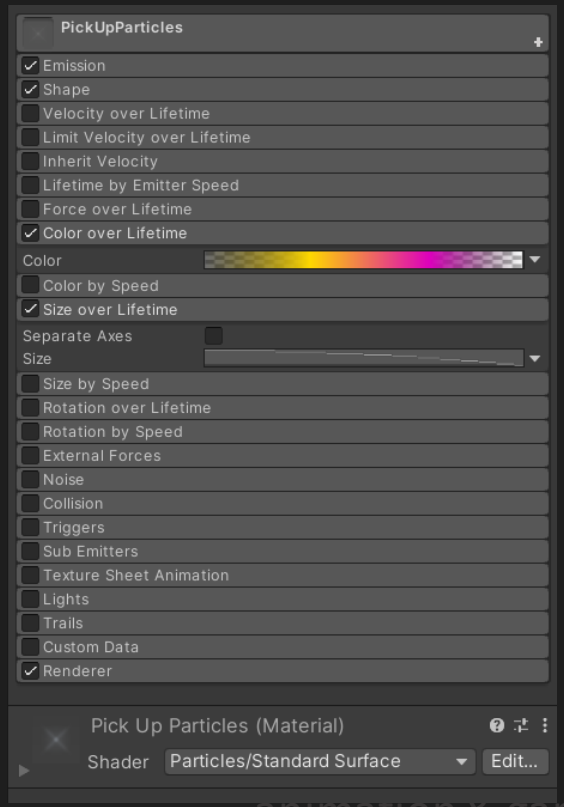


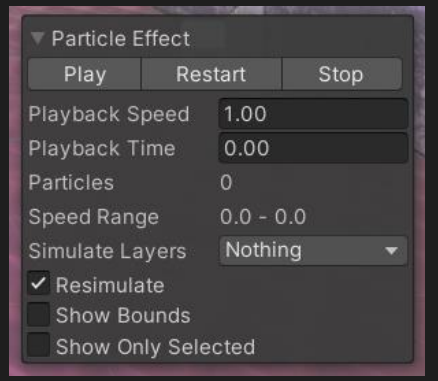
Refine the particle object

7. Particles have an almost infinite amount of options. One can get lost in them for a long time. Below are some settings to get a simple firework burst effect
8. You can preview the Particle Effect in the scene by clicking on the Play button in the scenes Particle Effect Box



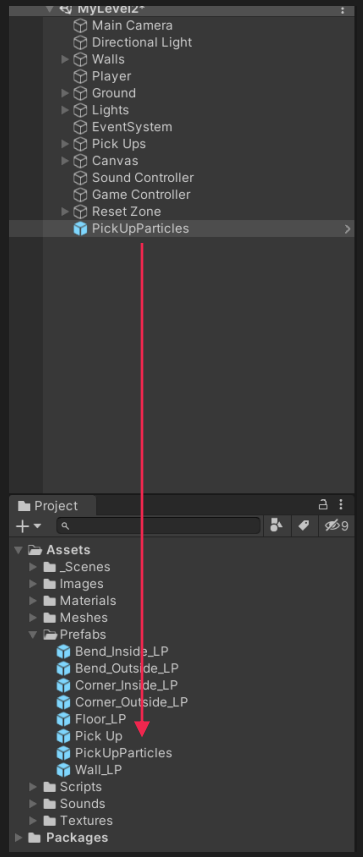






Create a script to fire our particle

9. Turn the particle into a Prefab by dragging it from our scene to our Prefabs folder.
You can now delete it from the scene



10. Create a script called Particles and fill it out as in the example

Particles.cs* | SceneController.cs | PlayerController.cs | Timer.cs | GameController.cs

Assembly-CSharp | Particles

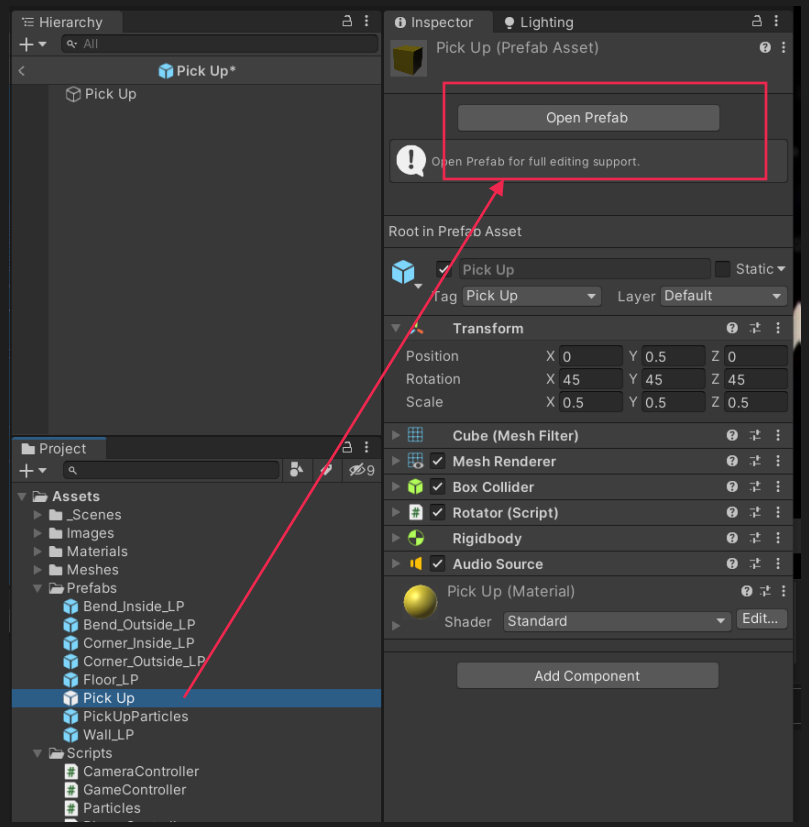
```
using UnityEngine;

public class Particles : MonoBehaviour
{
    public GameObject particlePrefab;

    public void CreateParticles()
    {
        Instantiate(particlePrefab, transform.position, transform.rotation);
    }
}
```

11. Add the following line to the PlayerController script in the OnTriggerEnter function

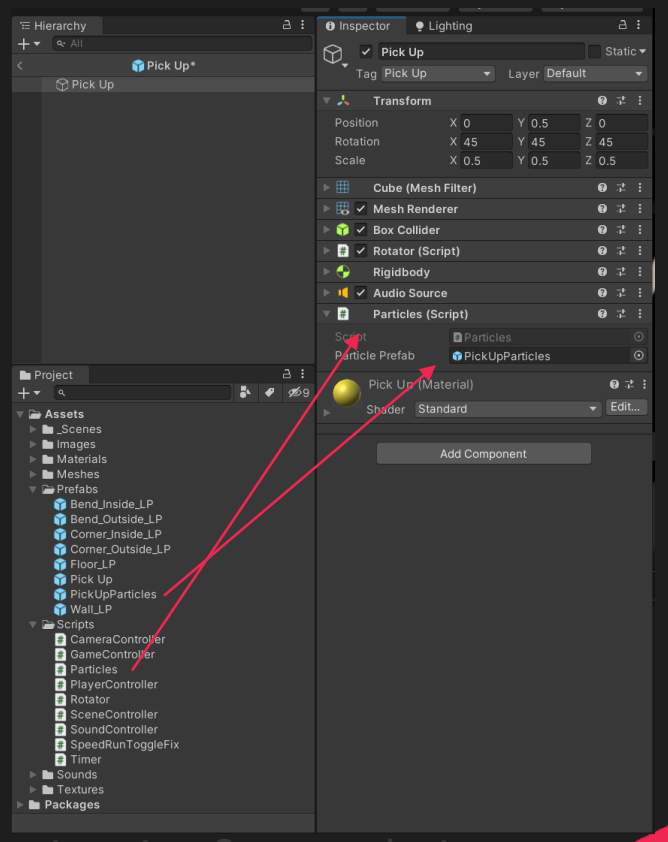
```
void OnTriggerEnter(Collider other)
{
    if(other.gameObject.CompareTag("Pick Up"))
    {
        other.GetComponent<Particles>().CreateParticles();
        other.gameObject.SetActive(false);
        count = count + 1;
        SetCountText();
        soundController.PlayPickupSound();
    }
}
```

Modify the Pick Up prefab

12. Open the Pick Up prefab by selecting it in the Project folder, then Open Prefab



13. Drag over the Particles script onto the prefab and drag the PickUpParticles prefab onto the Particle Prefab slot and you are DONE!!!



11. Rotating Wall or Door

1pt

Pre-requisite: None

Create a door or wall that rotates its position throughout the game.

Design: Model some elements such as doors, spikes, etc.

Programming: Coroutines, Lerping

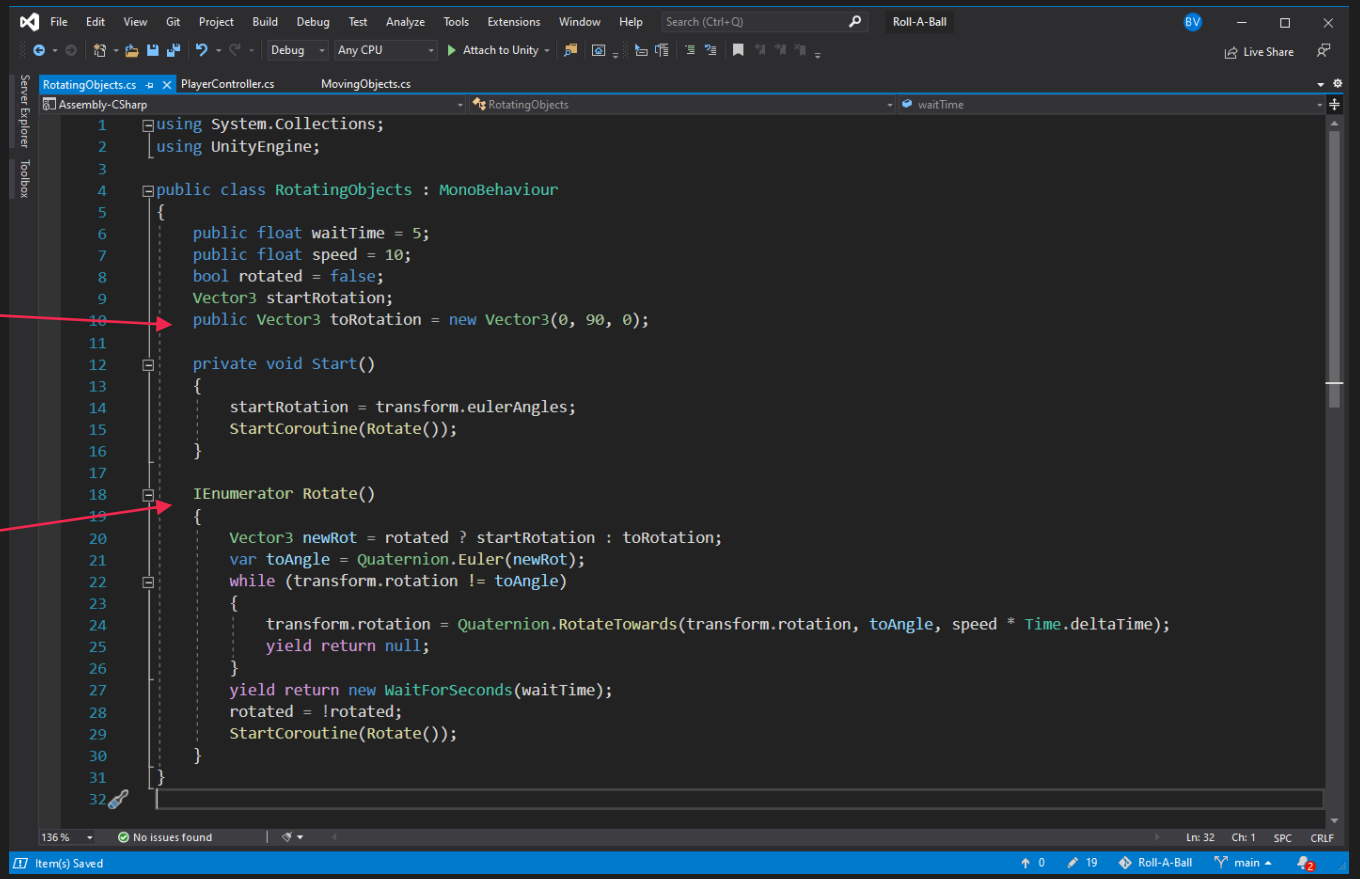
Create the RotatingObjects script

8. Create a script called RotatingObjects

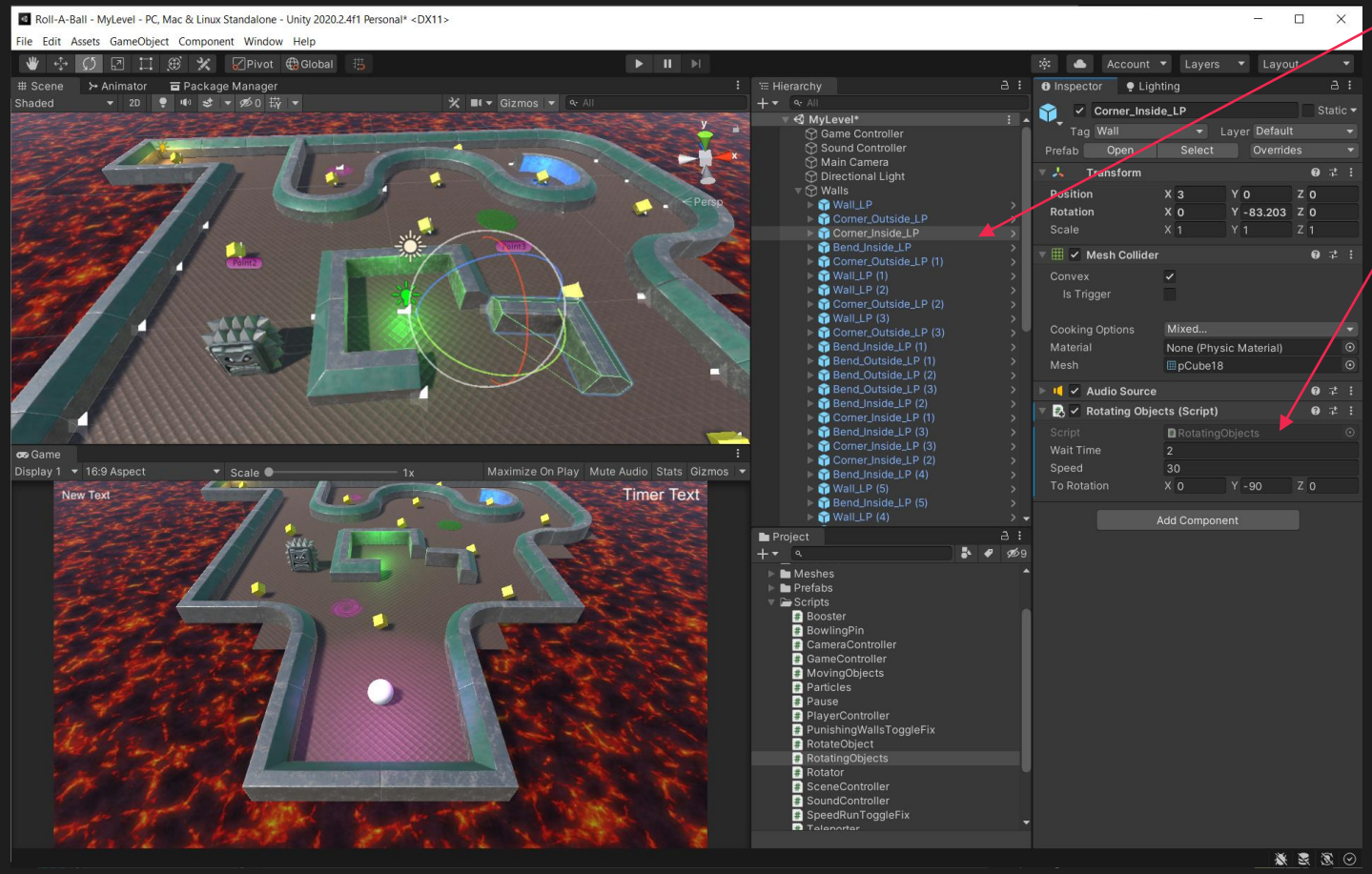
9. Copy it as in the image

10. The toRotation variable will be where we want to object to rotate to.
    - We can change this in the inspector later

11. Using a coroutine, we rotate to either the toRotation or the startRotation depending on our current rotation, wait some time then do it again



Setup in the scene



8. Attach the RotatingObjects script to anything in the scene we wish to rotate.

9. Adjust the variables to suit your needs

10. DONE!!!

12. Ramps and Other Environments

1pt

Pre-requisite: None

Create some ramps, jumps and other exciting models for your game world.

Design: Model some elements such as ramps, jumps, etc.

Programming: None

Create some new objects

1. Create some ramps and/or other environment models and texture them

13. Powerups - Speed Related

1pt

Pre-requisite: None

Create a powerup that can speed up or slow down the player

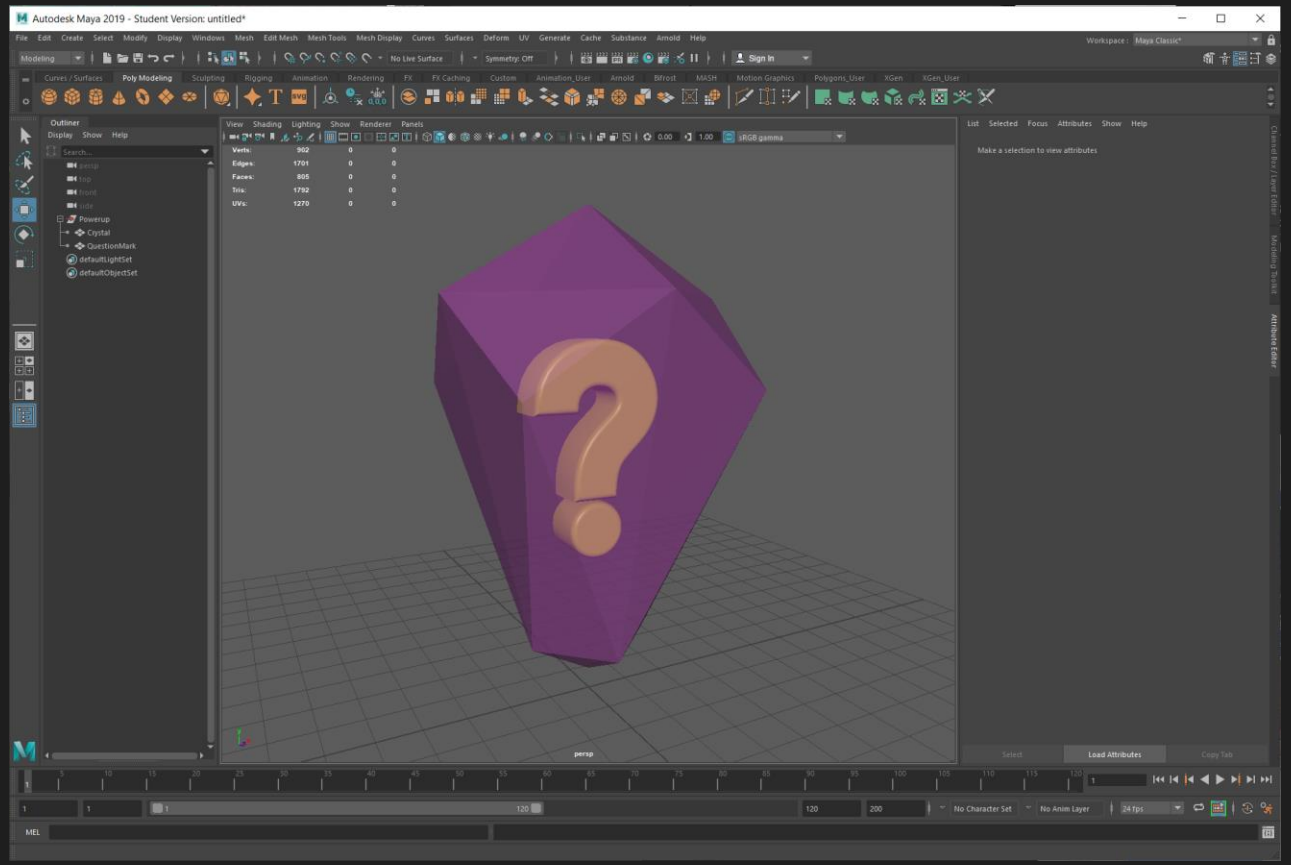
Design: Modelling and texturing a powerup

Programming: Enums, Trigger Events, Coroutines

Create the Powerup Model

1. Model a powerup in Maya

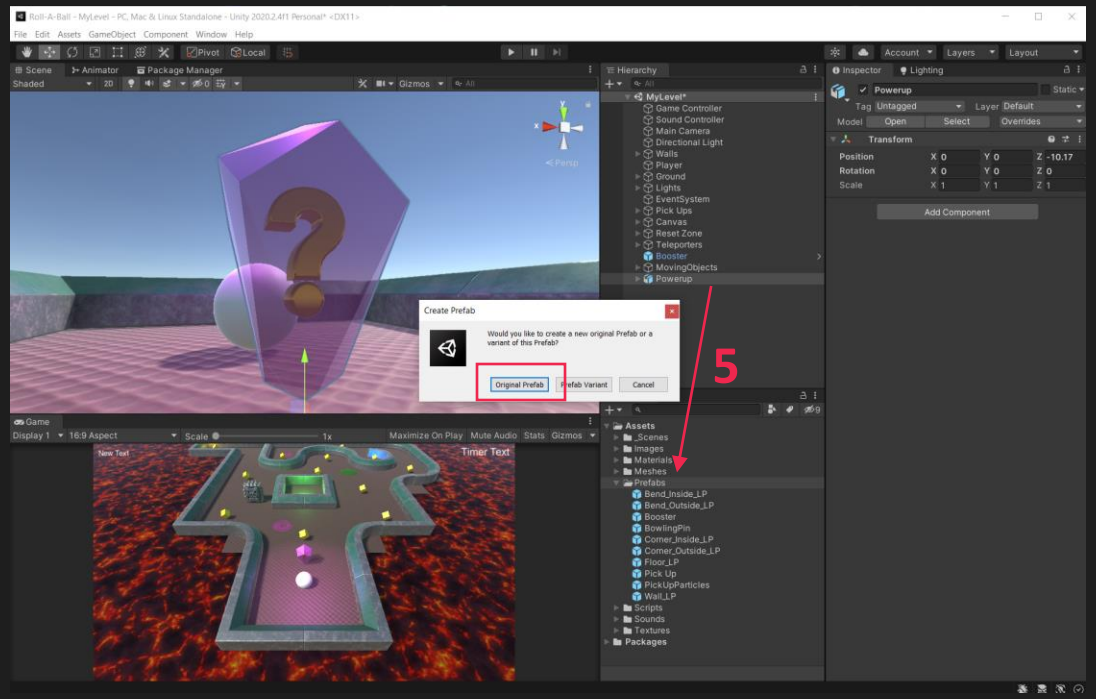
2. When happy with it, Export the model as an .fbx file into your unity project in the Meshes folder

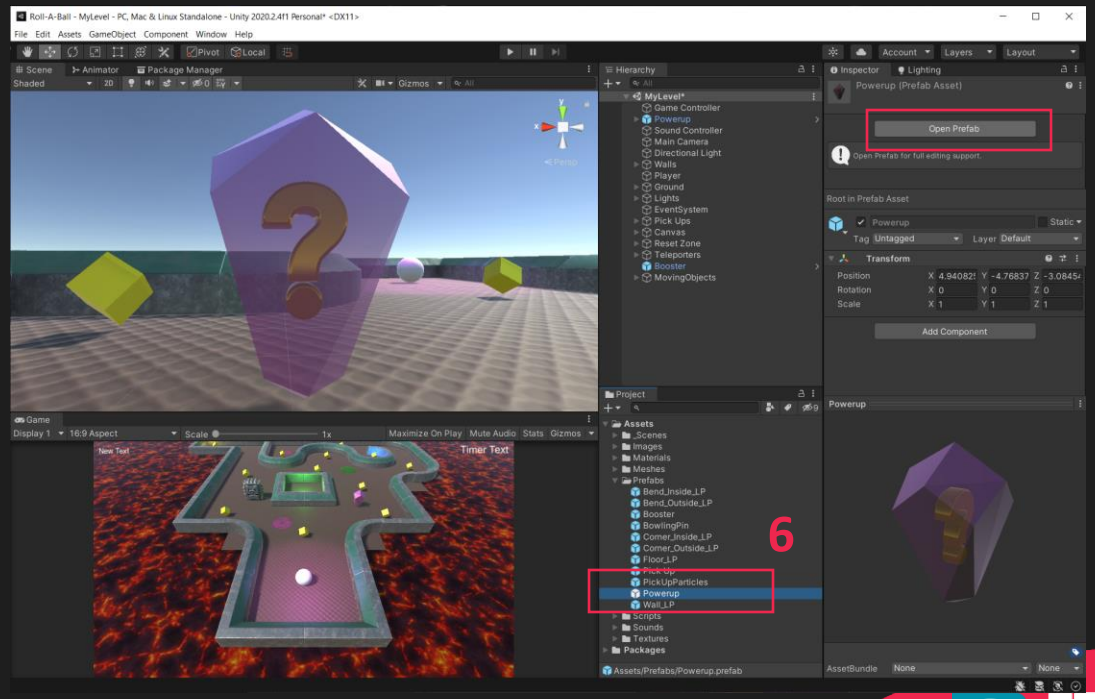


NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Setup the object in Unity

3. Drag the object from the project folder into the scene
4. Setup the Materials to look how you want in Unity
5. Click drag the object down into the Prefabs folder and select Original Prefab
6. Click on the prefab object in the project folder
7. In the Inspector, click on the OpenPrefab button

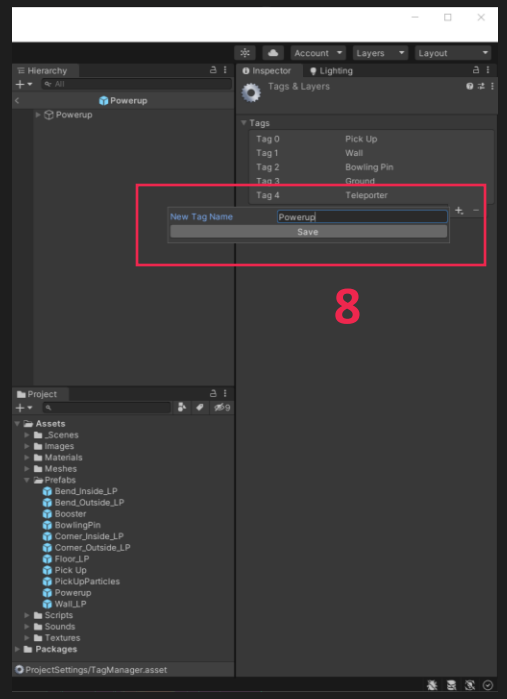


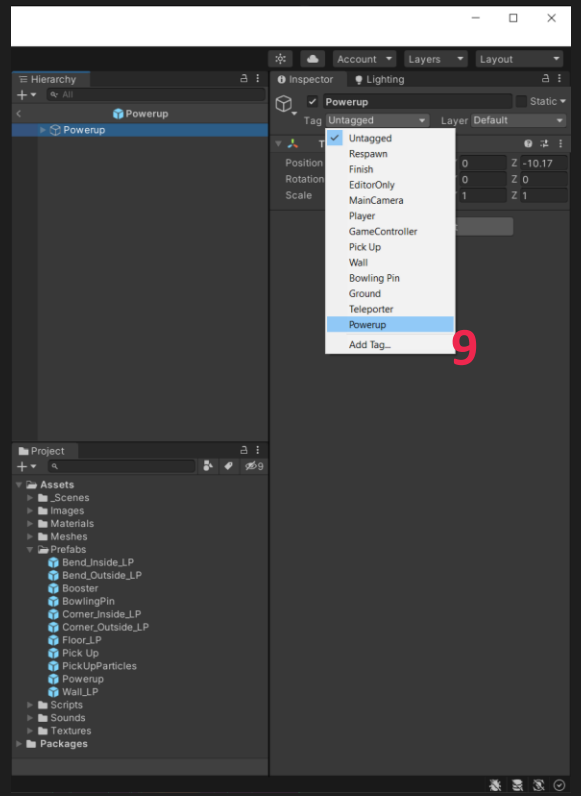


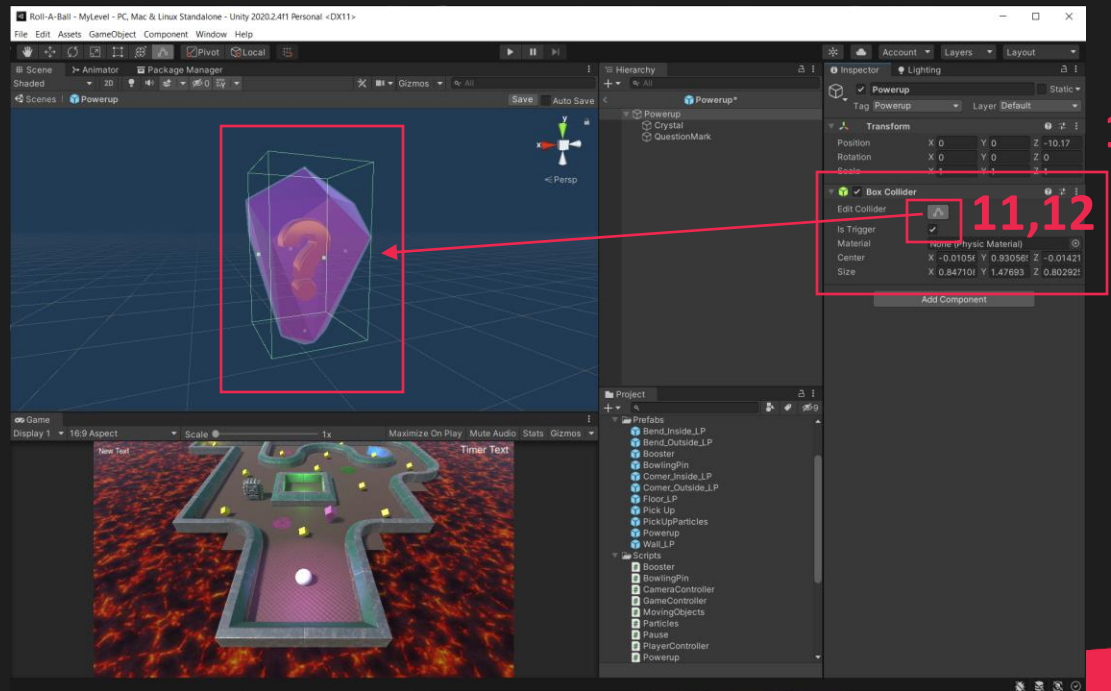
NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Setup the object in Unity

8. Add a tag “Powerup” to the tags section
9. Apply the tag to the Powerup prefab root object
   - The topmost object if you have multiple objects
10. Attach a box collider to the prefab root object
11. Edit the collider to cover the powerup object by dragging the little squares at the edge of the box
12. Check IsTrigger box







NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Modify our PlayerController script

13. Add a public float baseSpeed variable just under our speed variable
    - [HideInInspector] means that even though the variable is public, it will not show in the inspector.

```
public float speed = 10.0f;
[HideInInspector]
public float baseSpeed;
```

14. In our Start function, we will set the baseSpeed to be that of the speed by adding the following line

```
void Start()
{
    baseSpeed = speed;
    rb = GetComponent<Rigidbody>();
    count = 0;
```

15. Add the following chunk to the OnTriggerEnter function. We are moving the object position way out of view rather than disabling it as a hacky way to ensure that the script on the powerup still runs after collected.
    - If you are adding sound, this is where you should make the call to the SoundController

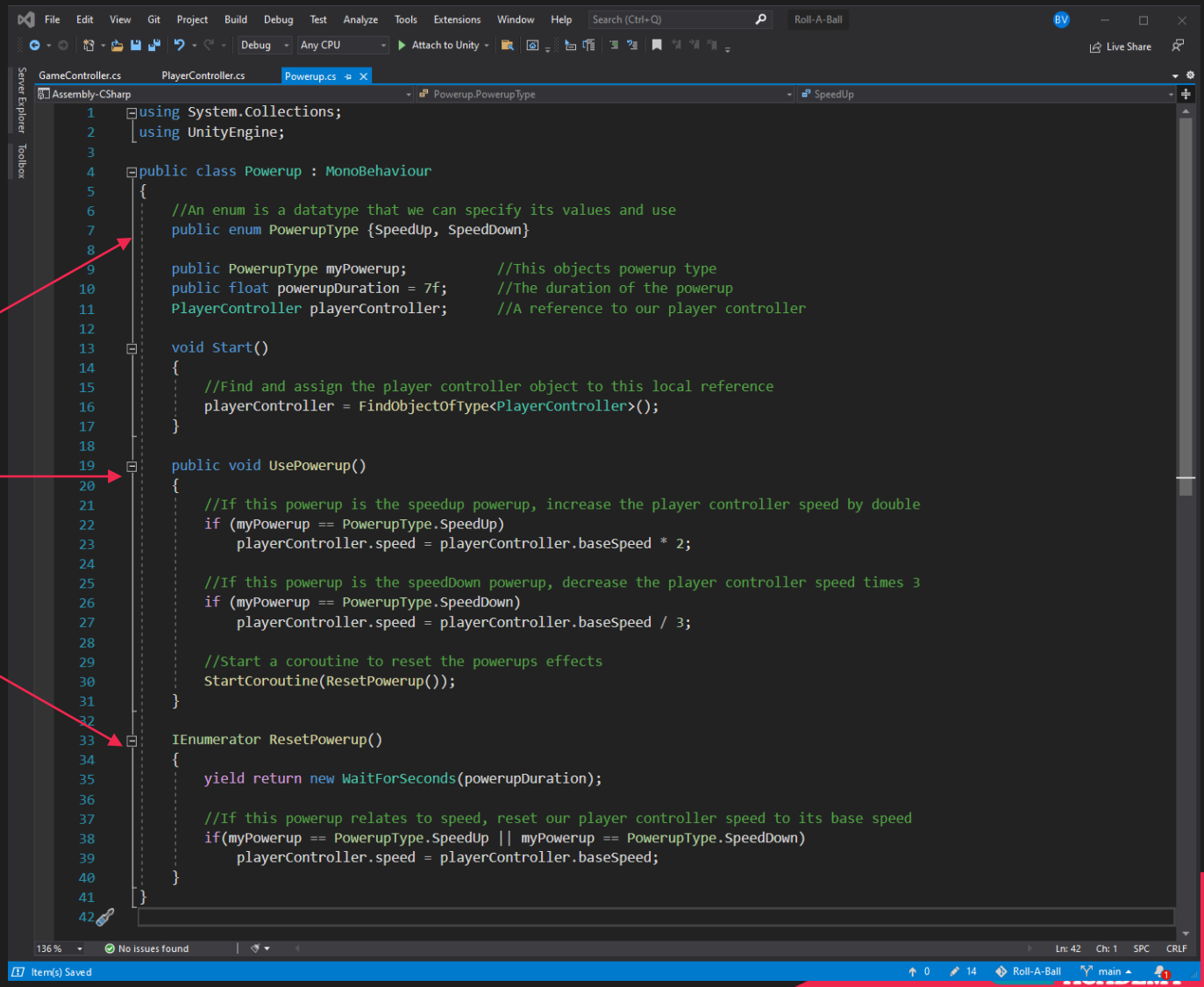
```
void OnTriggerEnter(Collider other)
{
    if(other.gameObject.CompareTag("Pick Up"))
    {
        other.GetComponent<Particles>().CreateParticles();
        other.gameObject.SetActive(false);
        count = count + 1;
        SetCountText();
        soundController.PlayPickupSound();
    }

    if(other.gameObject.CompareTag("Powerup"))
    {
        other.GetComponent<Powerup>().UsePowerup();
        other.gameObject.transform.position = Vector3.down * 1000;
    }
}
```

Create/Modify our Powerup script

16. Create a powerup script and fill it out as in the image (or modify your powerup script if already created)

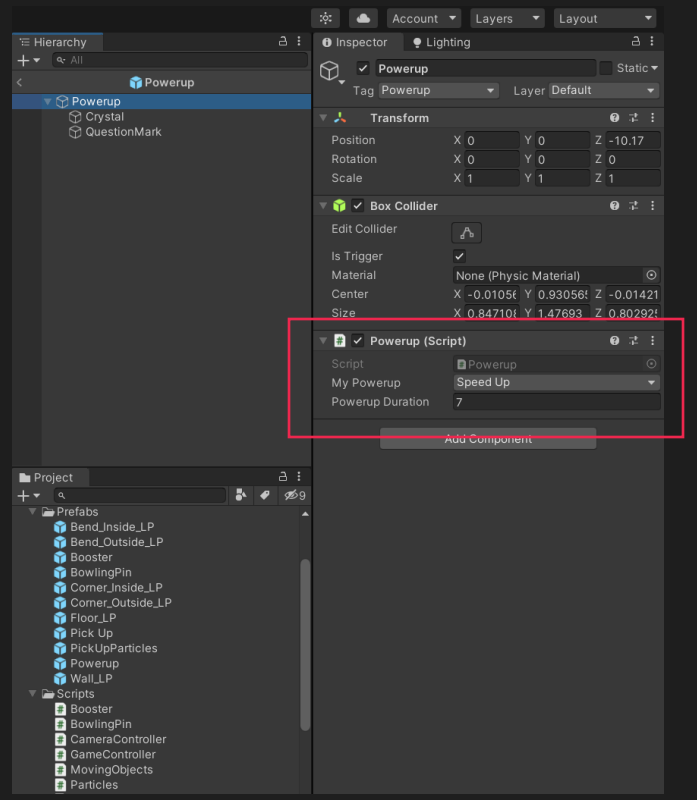
17. NOTES If you are doing multiple powerup modules :
    1. You will be adding into the PowerupType enum each module rather than creating a new enum each time.
    2. You will be adding to the UsePowerup function each time
    3. You will be adding to the ResetPowerup coroutine each time

18. This script has comments in to explain what everything is doing



Setup the object in Unity

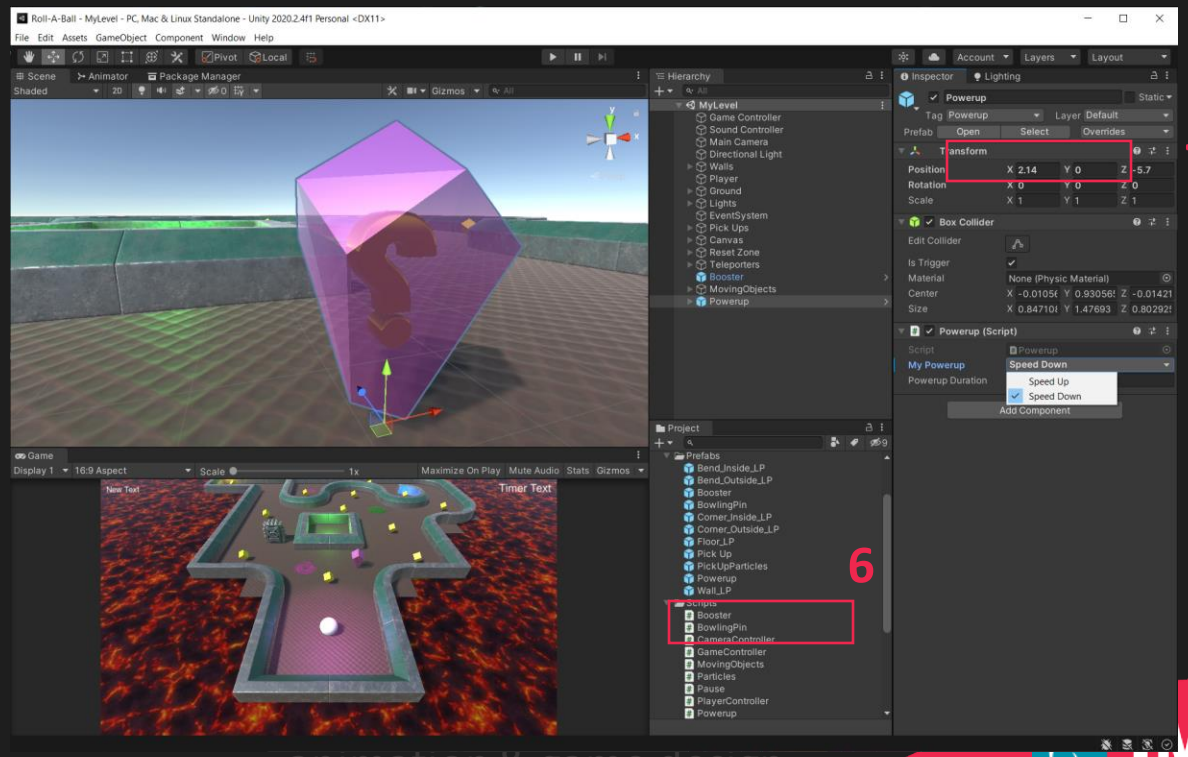
19. Attach the Powerup script to our Prefab, save it then go back into our Unity scene
    - You may have already done this in a previous powerup module



19

20. Place the powerups where you want and set their dropdown to the appropriate powerup type

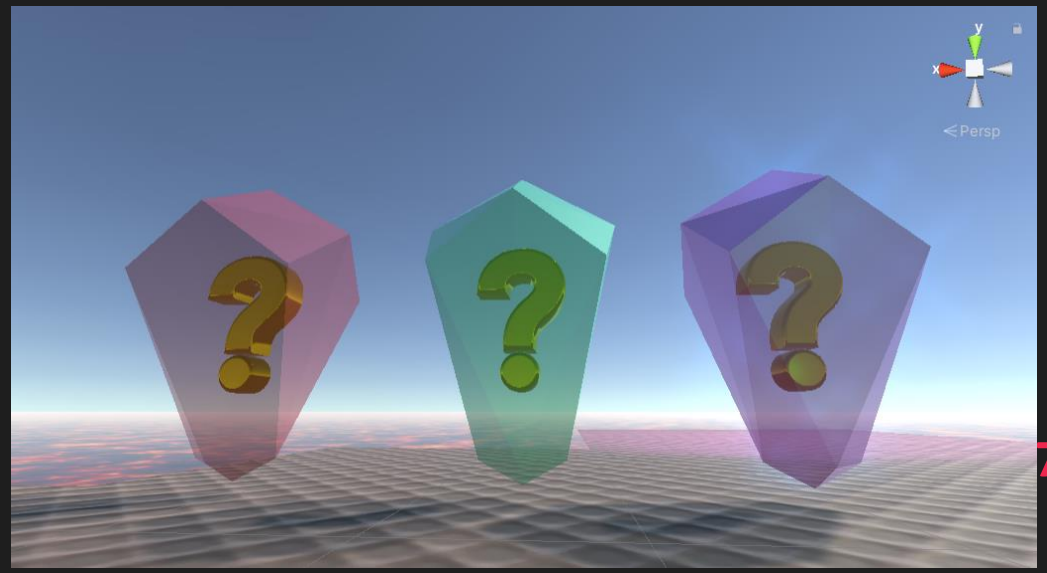
DONE!!!



Powerup Enhancements

- The powerup can be enhanced through other modules.

1. Sound when collected
   - Different sound depending on the powerup type
2. Particle Effects
3. Moving Powerups
   - Powerups move around the world



- Colour the powerups differently based on their type, or keep them all the same so the player doesn't know what's in it!
- Randomize the powerups on game start by adding the following to the Start Function

```
void Start()
{
    //Find and assign the player controller object to this local reference
    playerController = FindObjectOfType<PlayerController>();

    //This line will change the myPowerup typoe to be random each time
    myPowerup = (PowerupType)Random.Range(0, System.Enum.GetValues(typeof(PowerupType)).Length);
}
```

14. Powerups - Size Related

1pt

Pre-requisite: None

Create a powerup that can grow or shrink the player

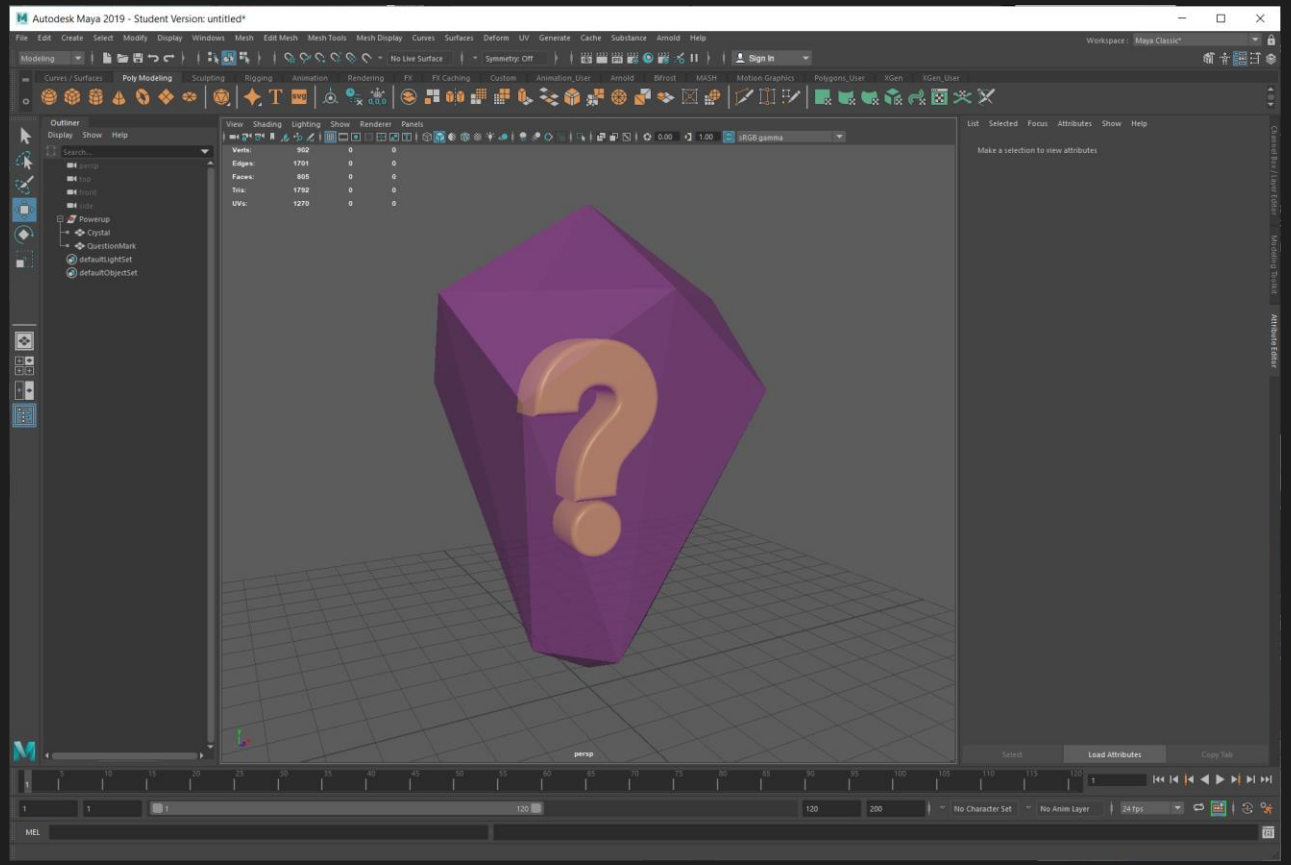
Design: Modelling and texturing a powerup

Programming: Enums, Trigger Events, Coroutines

Create the Powerup Model

1. Model a powerup in Maya

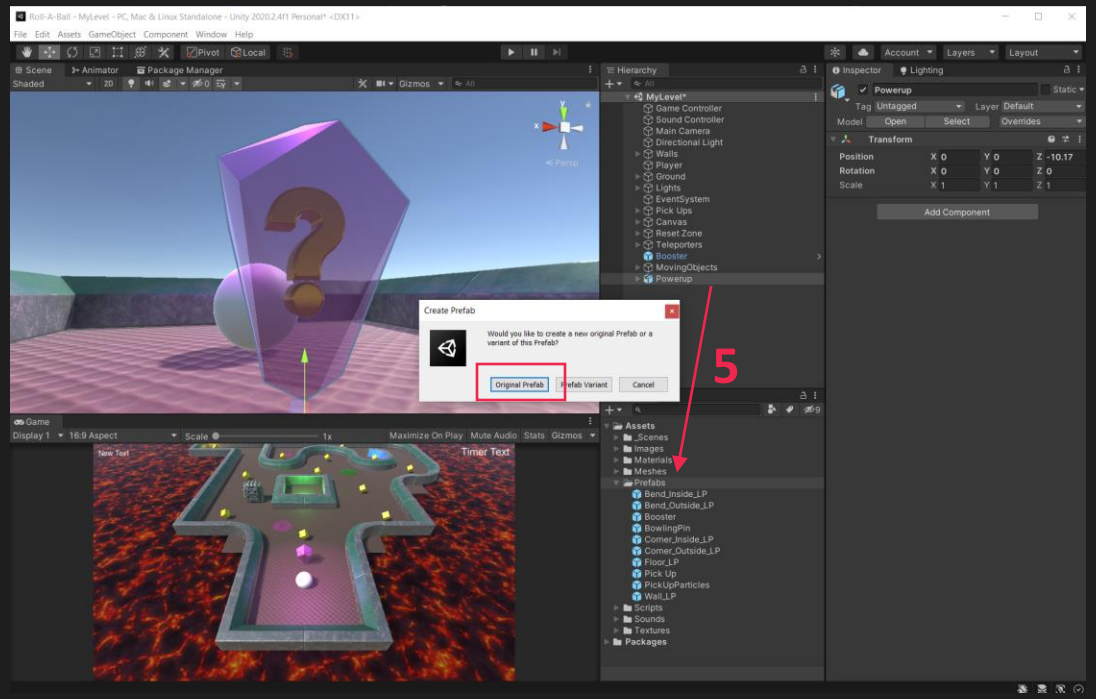
2. When happy with it, Export the model as an .fbx file into your unity project in the Meshes folder

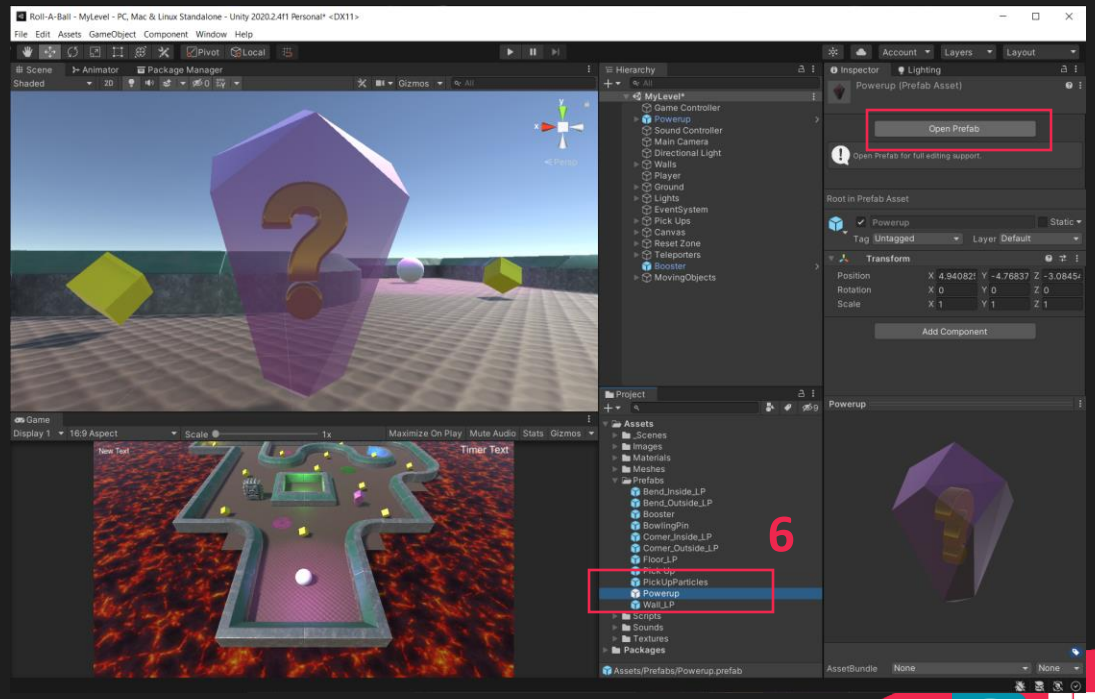


NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Setup the object in Unity

3. Drag the object from the project folder into the scene
4. Setup the Materials to look how you want in Unity
5. Click drag the object down into the Prefabs folder and select Original Prefab
6. Click on the prefab object in the project folder
7. In the Inspector, click on the OpenPrefab button

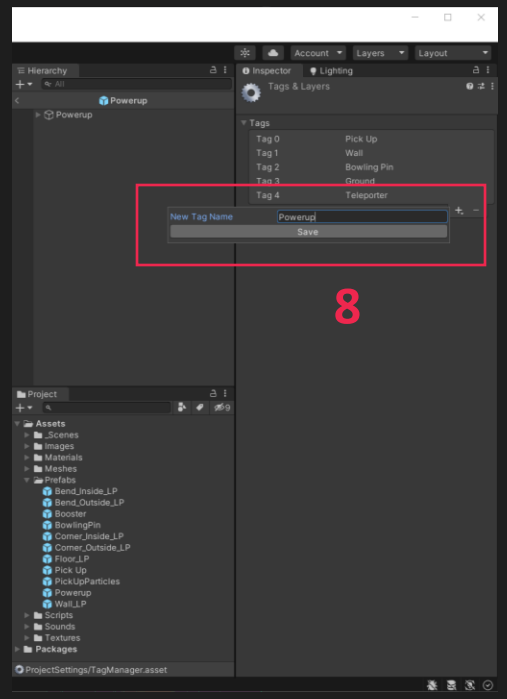


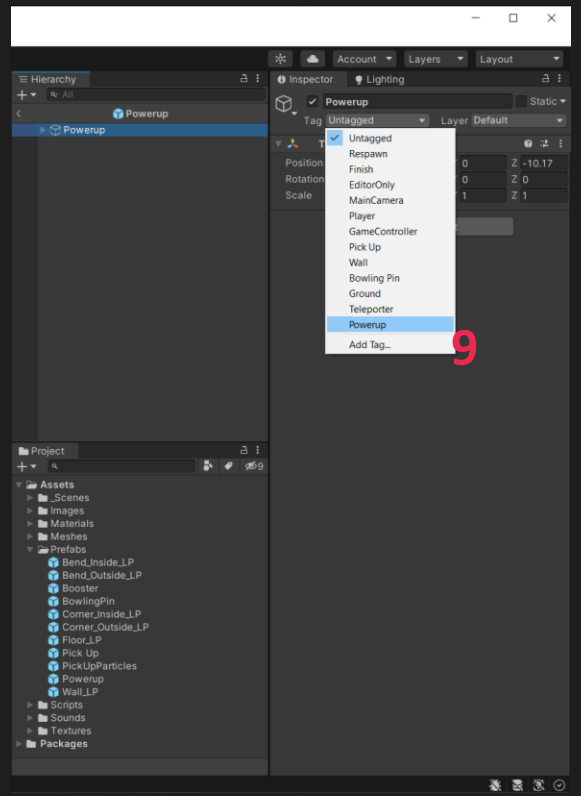


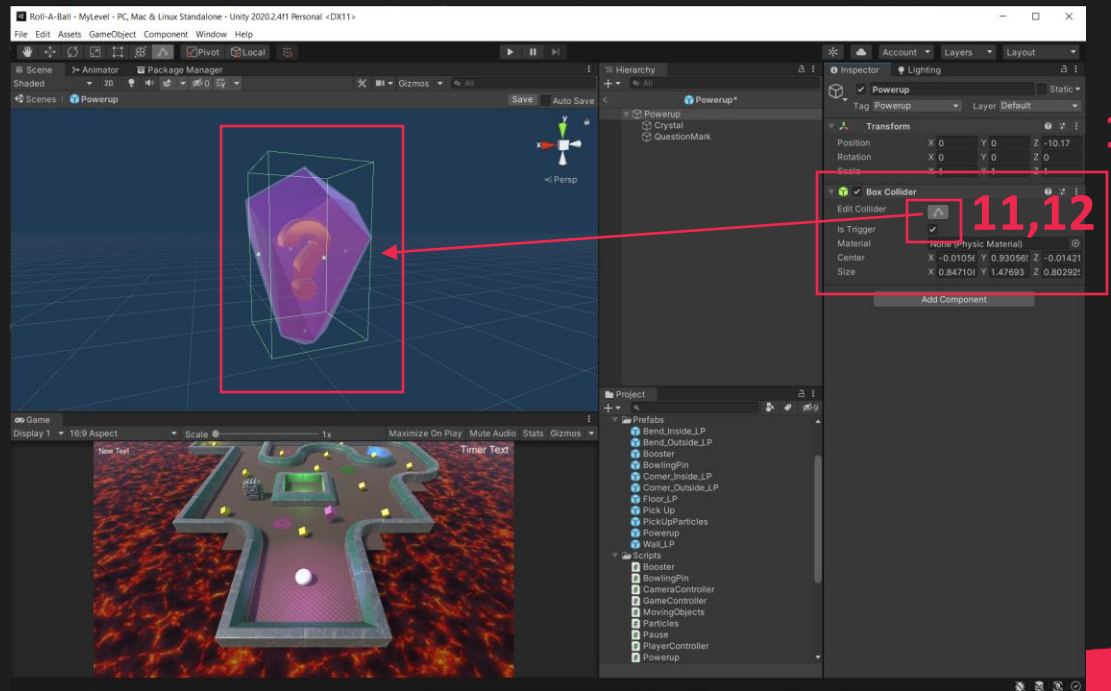
NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Setup the object in Unity

8. Add a tag “Powerup” to the tags section
9. Apply the tag to the Powerup prefab root object
   - The topmost object if you have multiple objects
10. Attach a box collider to the prefab root object
11. Edit the collider to cover the powerup object by dragging the little squares at the edge of the box
12. Check IsTrigger box







NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Modify our PlayerController script

13. Add the following chunk to the OnTriggerEnter function of our GameController.
We are moving the object position way out of view rather than disabling it as a hacky way to ensure that the script on the powerup still runs after collected.
    - If you are adding sound, this is where you should make the call to the SoundController

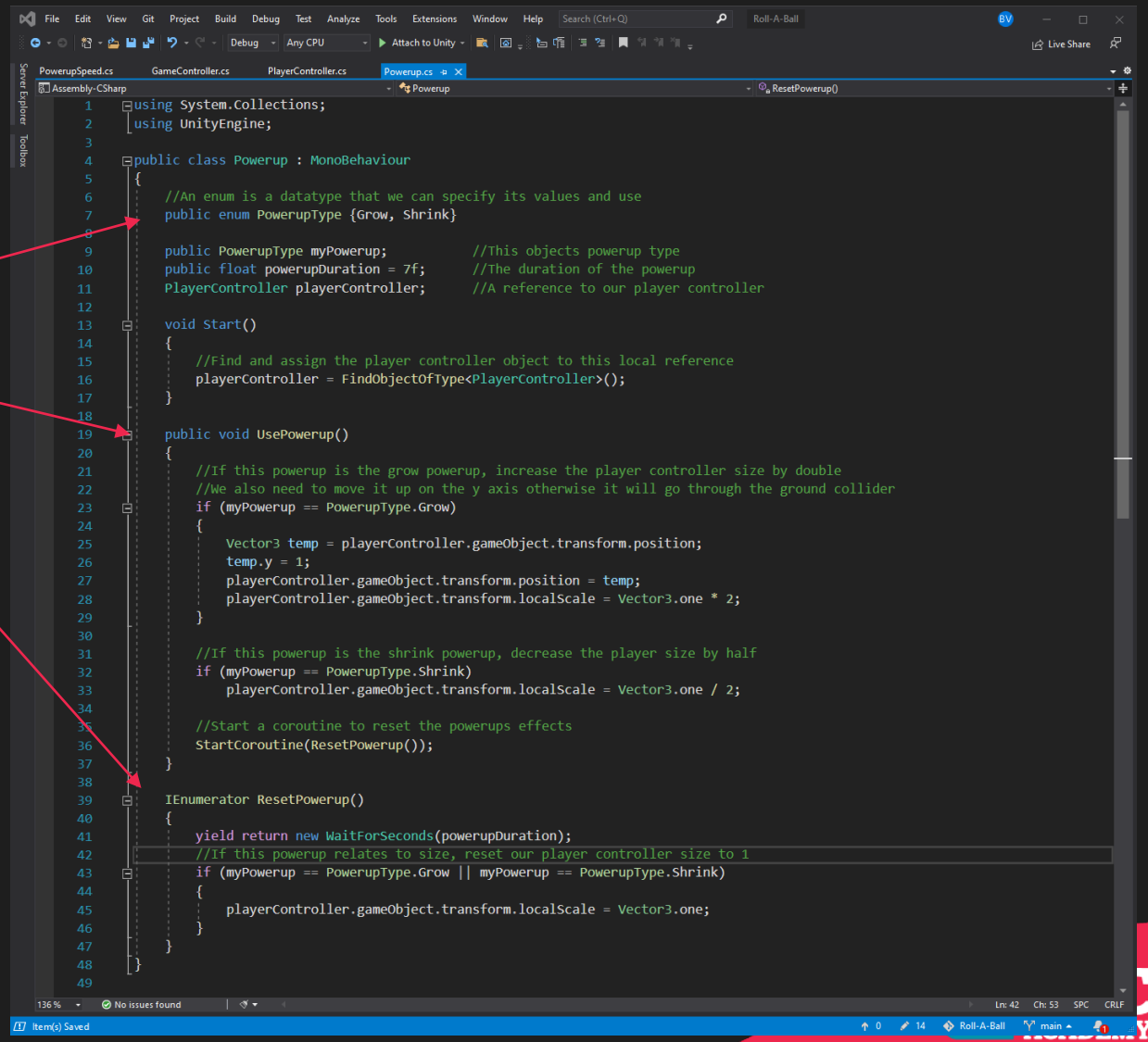
```
void OnTriggerEnter(Collider other)
{
    if(other.gameObject.CompareTag("Pick Up"))
    {
        other.GetComponent<Particles>().CreateParticles();
        other.gameObject.SetActive(false);
        count = count + 1;
        SetCountText();
        soundController.PlayPickupSound();
    }

    if(other.gameObject.CompareTag("Powerup"))
    {
        other.GetComponent<Powerup>().UsePowerup();
        other.gameObject.transform.position = Vector3.down * 1000;
    }
}
```

Create/Modify our Powerup script

16. Create a powerup script and fill it out as in the image (or modify your powerup script if already created)

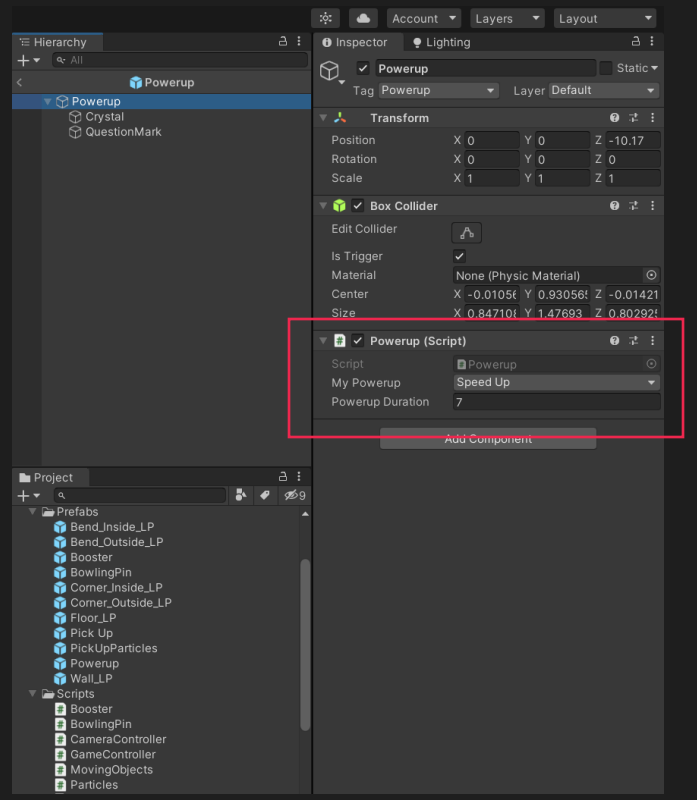
17. NOTES If you are doing multiple powerup modules :
    1. You will be adding into the PowerupType enum each module rather than creating a new enum each time.
    2. You will be adding to the UsePowerup function each time
    3. You will be adding to the ResetPowerup coroutine each time

18. This script has comments in to explain what everything is doing



Setup the object in Unity

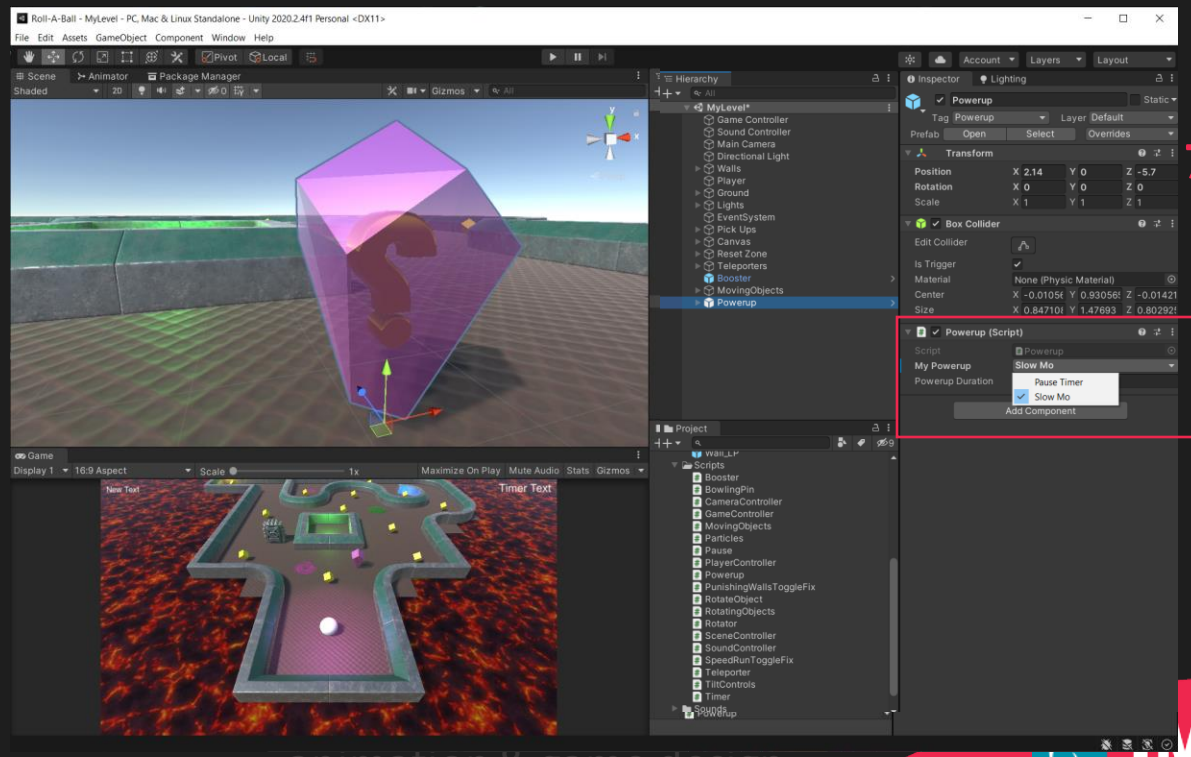
19. Attach the Powerup script to our Prefab, save it then go back into our Unity scene
    - You may have already done this in a previous powerup module



19

20. Place the powerups where you want and set their dropdown to the appropriate powerup type

DONE!!!

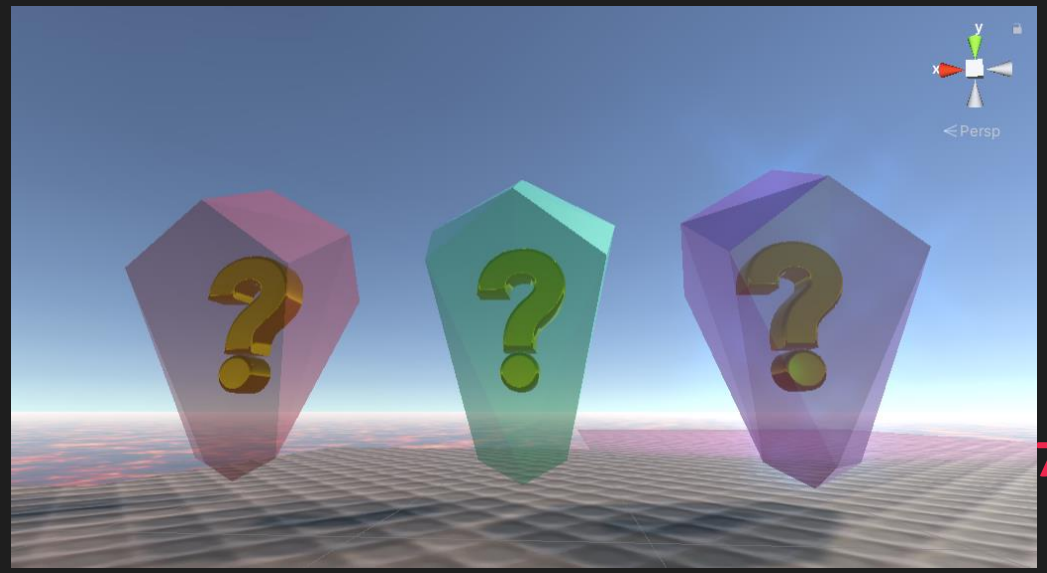


7

Powerup Enhancements

- The powerup can be enhanced through other modules.

1. Sound when collected
   - Different sound depending on the powerup type
2. Particle Effects
3. Moving Powerups
   - Powerups move around the world



- Colour the powerups differently based on their type, or keep them all the same so the player doesn't know what's in it!
- Randomize the powerups on game start by adding the following to the Start Function

```
void Start()
{
    //Find and assign the player controller object to this local reference
    playerController = FindObjectOfType<PlayerController>();

    //This line will change the myPowerup typoe to be random each time
    myPowerup = (PowerupType)Random.Range(0, System.Enum.GetValues(typeof(PowerupType)).Length);
}
```

15. Powerups - Time Related

1pt

Pre-requisite: Timer

Create a powerup that can pause the timer and do a slow-mo effect

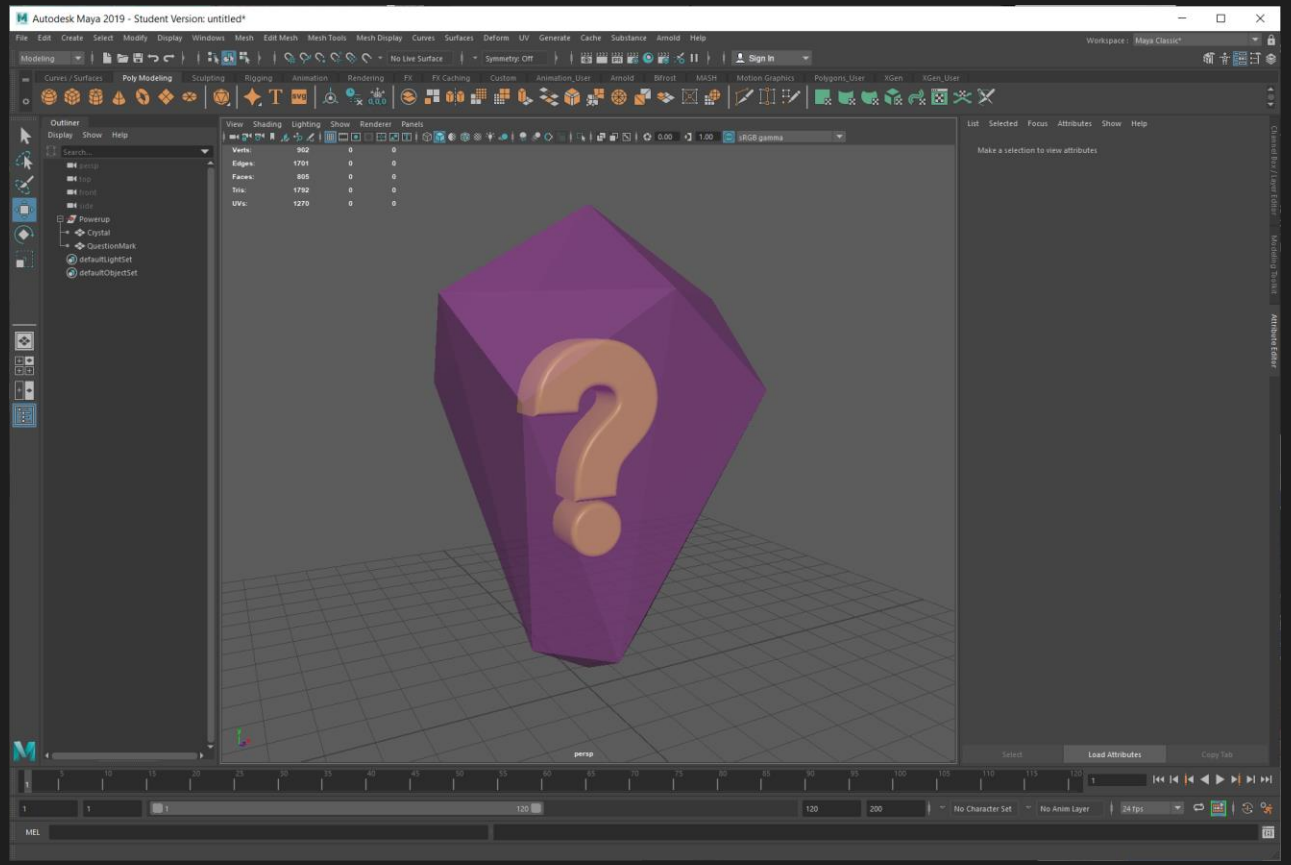
Design: Modelling and texturing a powerup

Programming: Enums, Trigger Events, Coroutines

Create the Powerup Model

1. Model a powerup in Maya

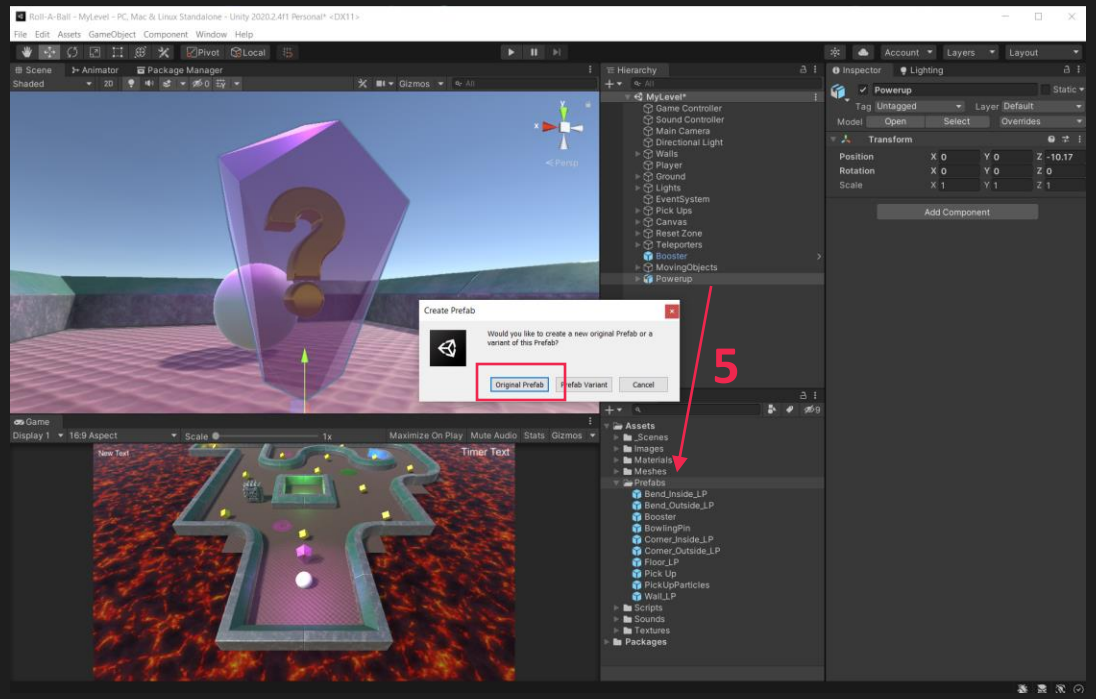
2. When happy with it, Export the model as an .fbx file into your unity project in the Meshes folder

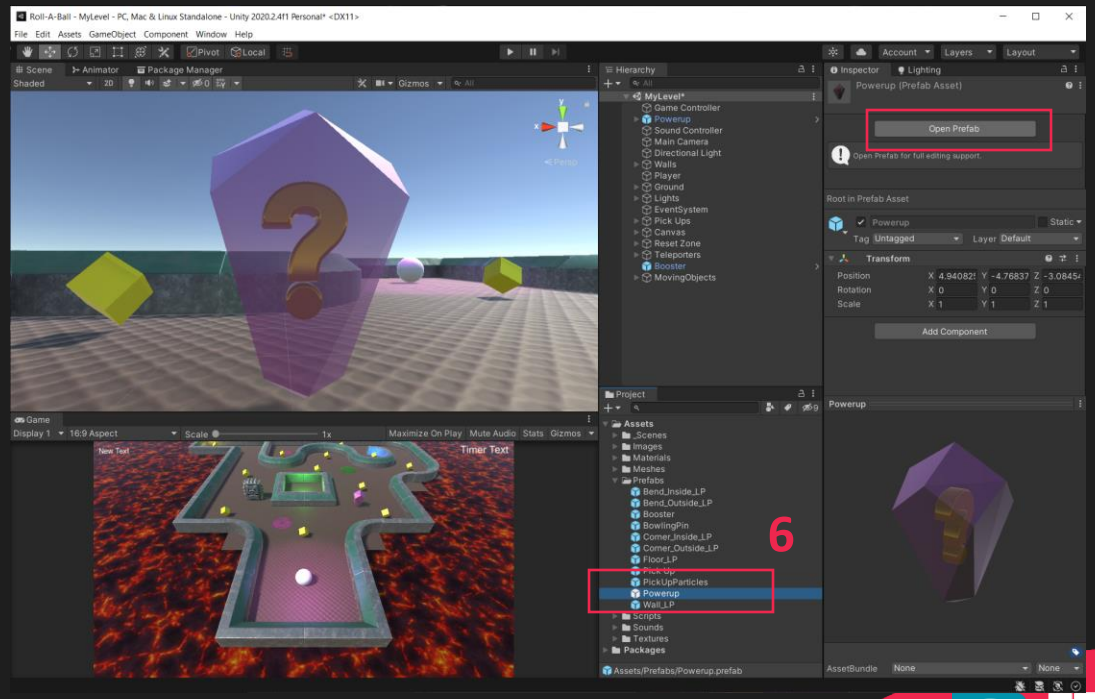


NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Setup the object in Unity

3. Drag the object from the project folder into the scene
4. Setup the Materials to look how you want in Unity
5. Click drag the object down into the Prefabs folder and select Original Prefab
6. Click on the prefab object in the project folder
7. In the Inspector, click on the OpenPrefab button

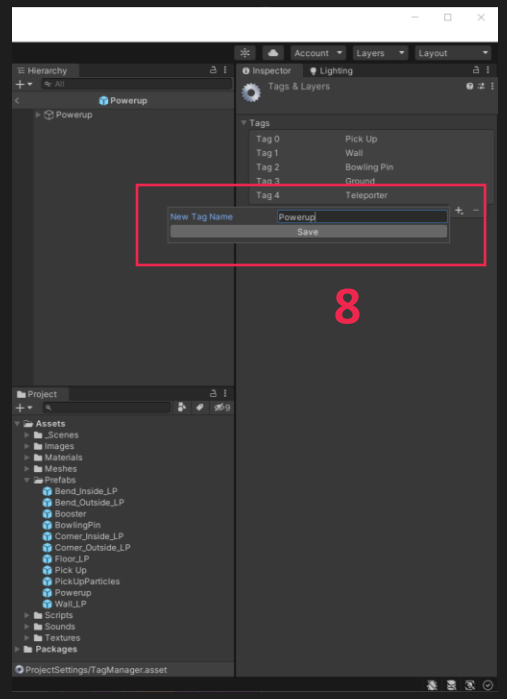


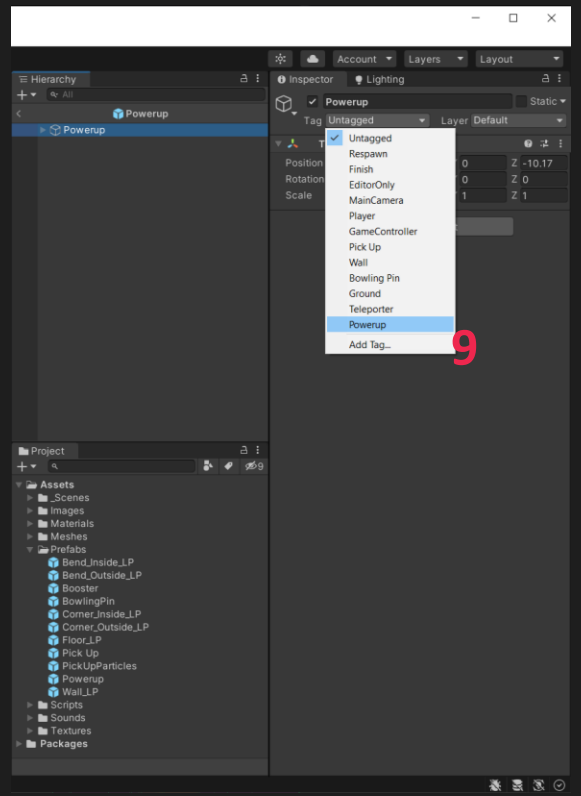


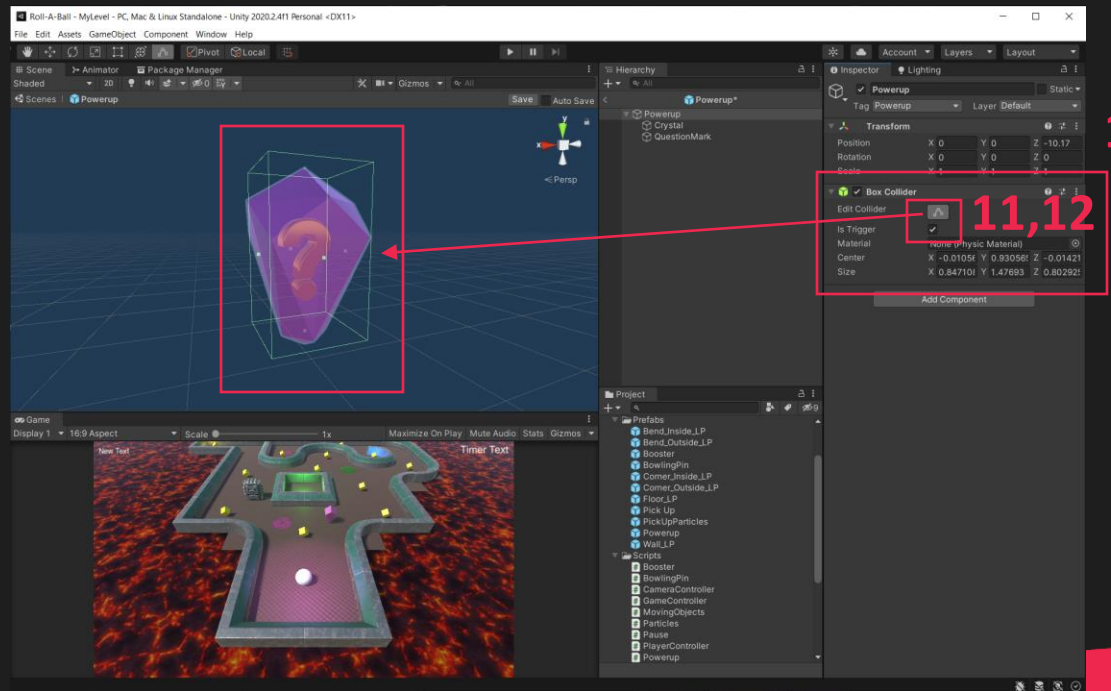
NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Setup the object in Unity

8. Add a tag “Powerup” to the tags section
9. Apply the tag to the Powerup prefab root object
   - The topmost object if you have multiple objects
10. Attach a box collider to the prefab root object
11. Edit the collider to cover the powerup object by dragging the little squares at the edge of the box
12. Check IsTrigger box







NOTE: You may have already done this step in another Powerup Module. If so skip ahead.

Modify our Timer script

13. Add a paused Boolean variable at the top of the script

14. Modify our if statement in the Update function to check if we aren't paused to use the timer

15. Add the following lines to the StopTimer function

16. Create a new PauseTimer function at the bottom, that passes in a bool and changes the paused value to it

17. Create a new ChangeTimeScale function at the bottom that passes in a float and changes Unity's time scale to the passed in value

```
float currentTime = 0;
float bestTime;
bool timing = false;
bool paused = false;
```

```
void Update()
{
    if(timing && !paused)
    {
        currentTime += Time.deltaTime;
        timerText.text = currentTime.ToString("F3");
    }
}
```

```
public void StopTimer()
{
    ChangeTimeScale(1);
    paused = false;
    timing = false;
    timesPanel.SetActive(true);
```

```
public void PauseTimer(bool _paused)
{
    paused = _paused;
}

public void ChangeTimeScale(float _timeScale)
{
    Time.timeScale = _timeScale;
}
```

Create/Modify our Powerup script

16. Create a powerup script and fill it out as in the image (or modify your powerup script if already created)

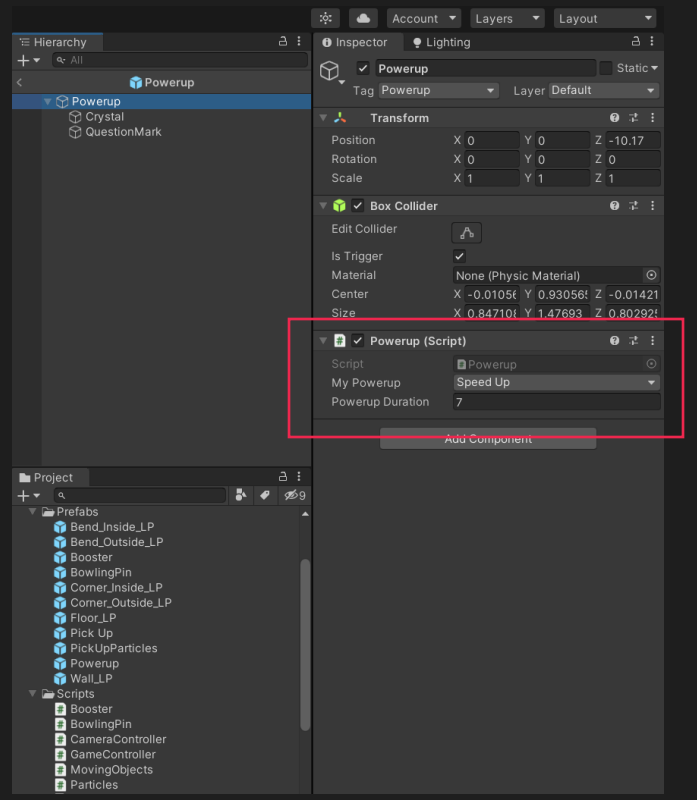
17. NOTES If you are doing multiple powerup modules :
    1. You will be adding into the PowerupType enum each module rather than creating a new enum each time.
    2. You will be adding to the UsePowerup function each time
    3. You will be adding to the ResetPowerup coroutine each time

18. This script has comments in to explain what everything is doing

```
embly-CSharp                    Powerup.PowerupType                    SlowMo
1   using System.Collections;
2   using UnityEngine;
3
4   public class Powerup : MonoBehaviour
5   {
6       //An enum is a datatype that we can specify its values and use
7       public enum PowerupType {PauseTimer, SlowMo}
8
9       public PowerupType myPowerup;           //This objects powerup type
10      public float powerupDuration = 7f;      //The duration of the powerup
11      Timer timer;                            //A reference to our timer
12
13      void Start()
14      {
15          //Find and assign the timer object to this local reference
16          timer = FindObjectOfType<Timer>();
17      }
18
19      public void UsePowerup()
20      {
21          //If this powerup is the pause powerup, pause the timer in the timer script
22          if (myPowerup == PowerupType.PauseTimer)
23              timer.PauseTimer(true);
24
25          //If this powerup is the slowMo powerup, change the timers timescale to slower (0.4)
26          if (myPowerup == PowerupType.SlowMo)
27              timer.ChangeTimeScale(0.4f);
28
29          //Start a coroutine to reset the powerups effects
30          StartCoroutine(ResetPowerup());
31      }
32
33      IEnumerator ResetPowerup()
34      {
35          yield return new WaitForSeconds(powerupDuration);
36          //If this powerup is the pause, unpase the timer in the timer script
37          if (myPowerup == PowerupType.PauseTimer)
38              timer.PauseTimer(false);
39
40          //If this powerup is the slowmo powerup, set thye timescale back to 1
41          if (myPowerup == PowerupType.SlowMo)
42              timer.ChangeTimeScale(1);
43      }
44  }
```

Setup the object in Unity

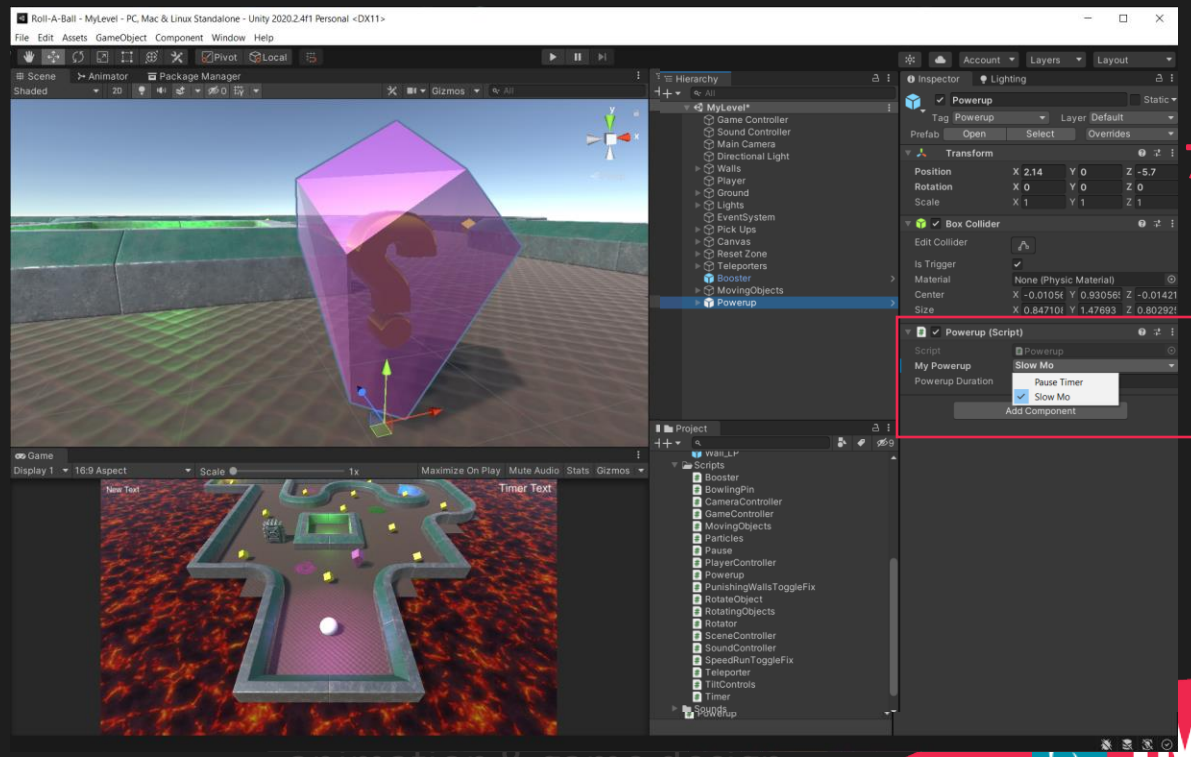
19. Attach the Powerup script to our Prefab, save it then go back into our Unity scene
    - You may have already done this in a previous powerup module



19

20. Place the powerups where you want and set their dropdown to the appropriate powerup type

DONE!!!

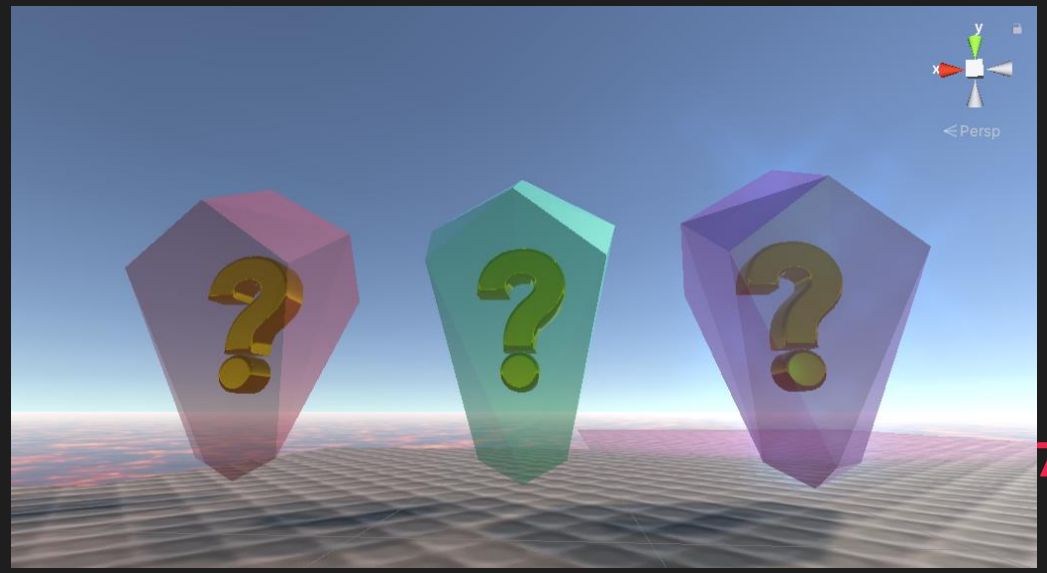


7

Powerup Enhancements

- The powerup can be enhanced through other modules.

1. Sound when collected
   - Different sound depending on the powerup type
2. Particle Effects
3. Moving Powerups
   - Powerups move around the world



- Colour the powerups differently based on their type, or keep them all the same so the player doesn't know what's in it!
- Randomize the powerups on game start by adding the following to the Start Function

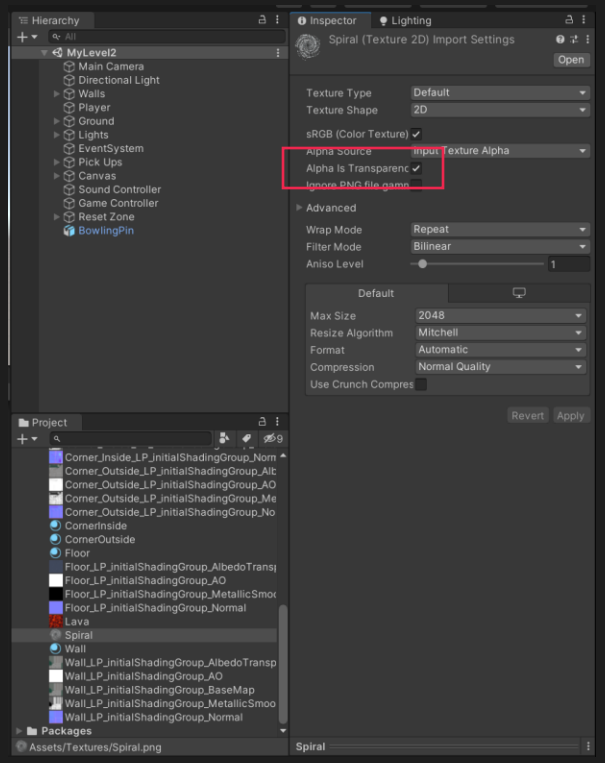
```
void Start()
{
    //Find and assign the player controller object to this local reference
    playerController = FindObjectOfType<PlayerController>();

    //This line will change the myPowerup typoe to be random each time
    myPowerup = (PowerupType)Random.Range(0, System.Enum.GetValues(typeof(PowerupType)).Length);
}
```

16.Teleporter	1pt
Pre-requisite: None	
Physical zone that transports player to another area when touched.	
Design: Texturing, Particles	
Programming: Trigger Events	

Texture and Material

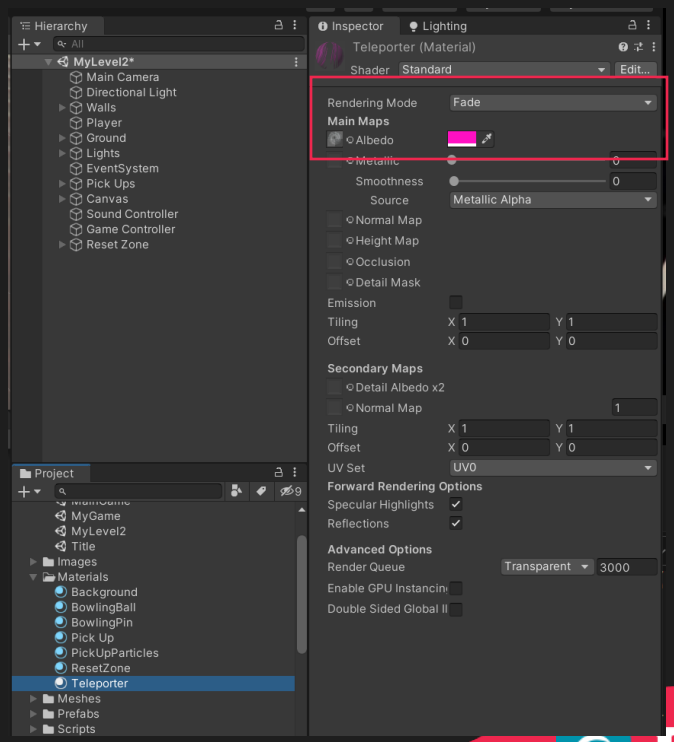
1. Create a spiral design in photoshop with a transparent background. Keep it to shades of white as we'll colour it in unity. Save as a .png

2. After importing the Texture, click on its settings and check Alpha is Transparent

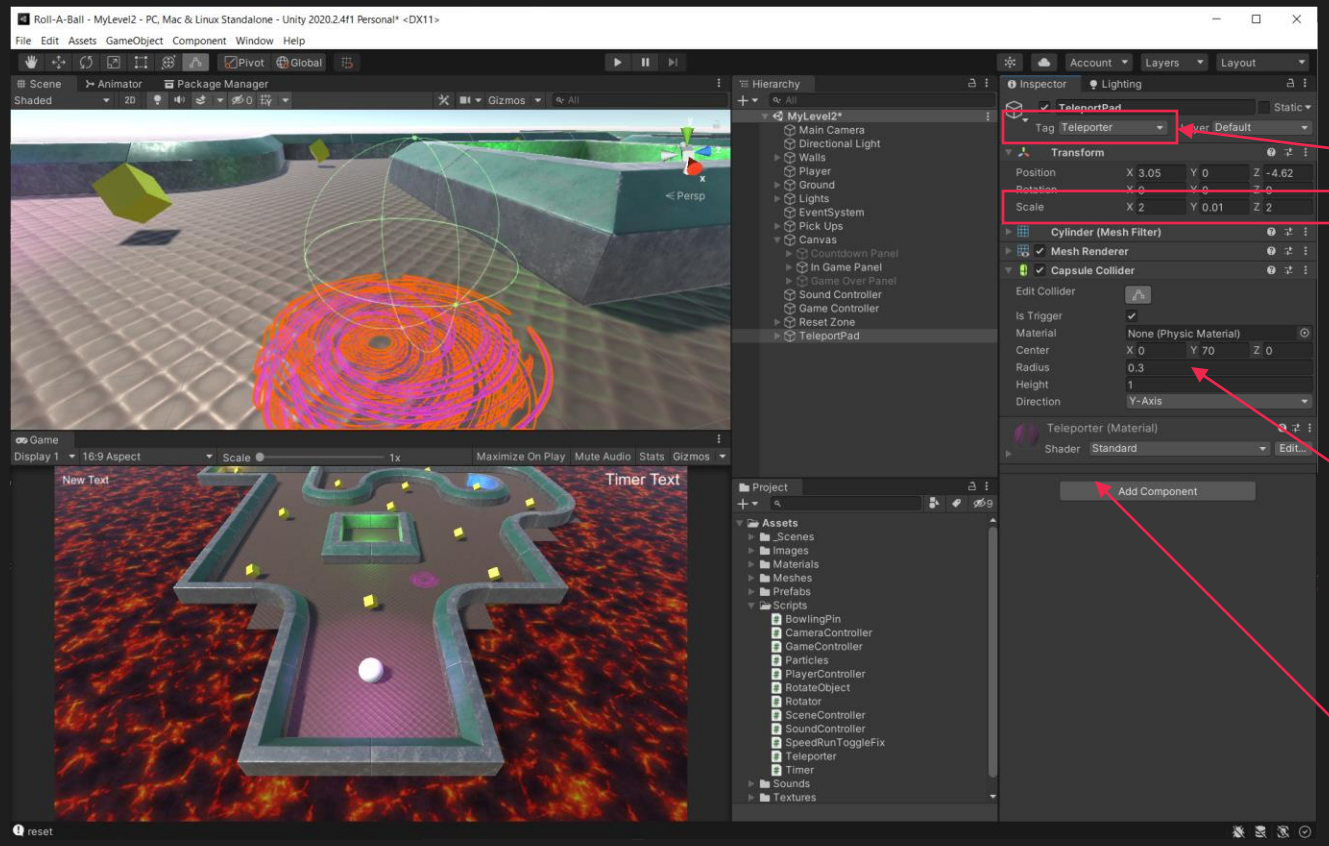


3. Create a Teleporter Material and drag the texture to the Albedo slot

4. Change the Rendering Mode to Fade and colour as you wish



Setup the object in Unity



5. Create a new 3D Object > Cylinder and name it TeleportPad

6. Add a tag Teleporter to the object

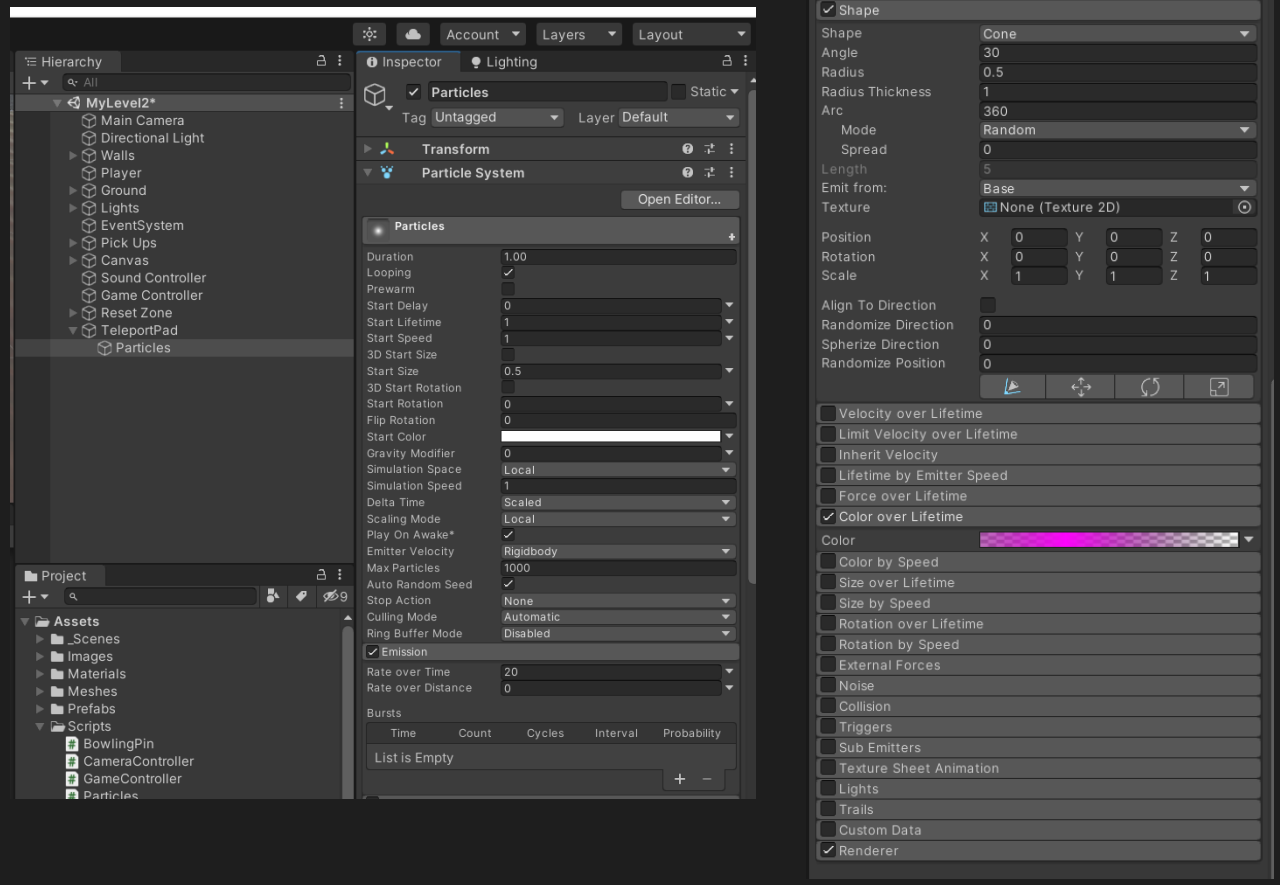
7. Scale it wafer thin (0.01) on the y axis and to 2 on the x and z

8. Check on Is Trigger on the collider. We must also edit the collider so it sits above the ground completely. Adjust the Center y value. Click on the  to see the colliders bounds

9. Drag on our Teleporter Material

Add some flair

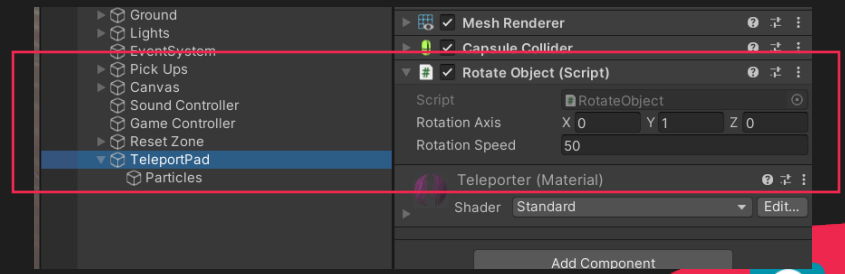
10. Right click on the TeleportPad > Effects > Particle System to create some particles. The settings below are a good start on your particle system but change as you like



11. Create a new script called RotateObject EXACTLY as below

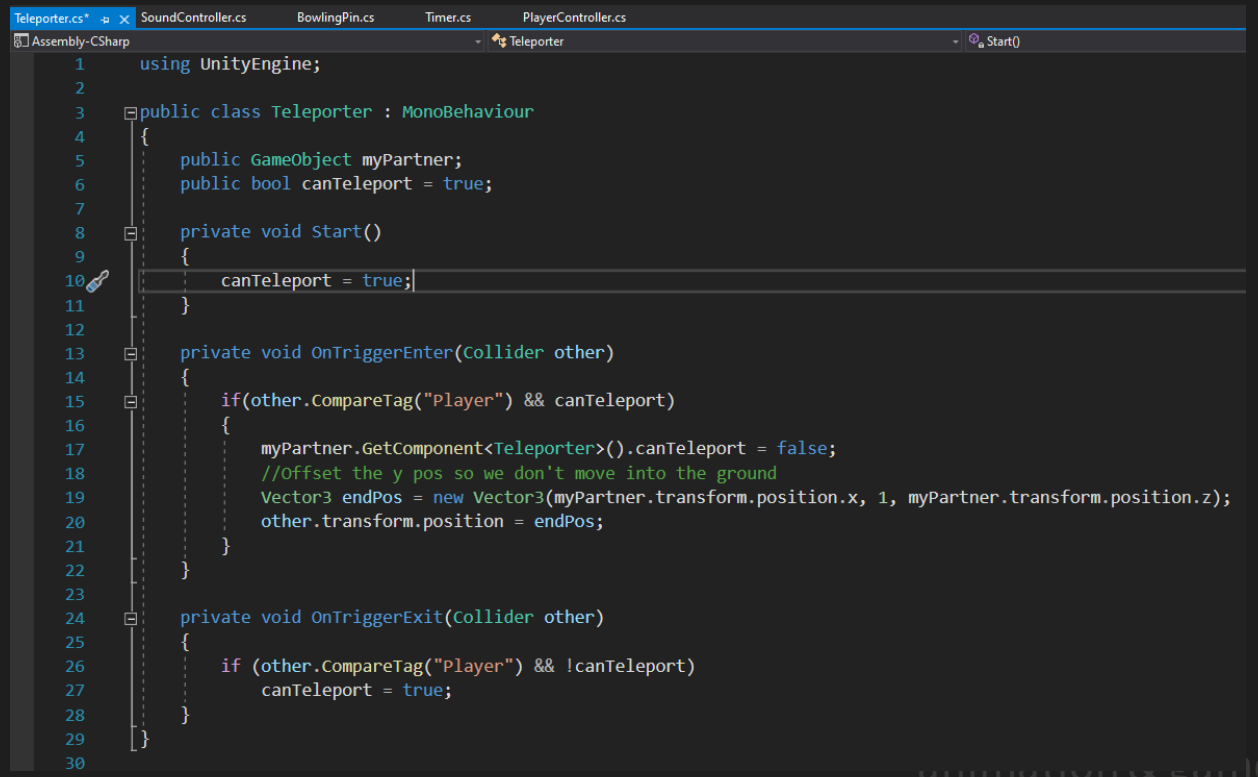
```
using UnityEngine;

public class RotateObject : MonoBehaviour
{
    public Vector3 rotationAxis;
    public float rotationSpeed;


    void Update()
    {
        transform.Rotate(rotationAxis * rotationSpeed * Time.deltaTime);
    }
}
```

12. Attach the script to the TeleportPad object and set the settings as below

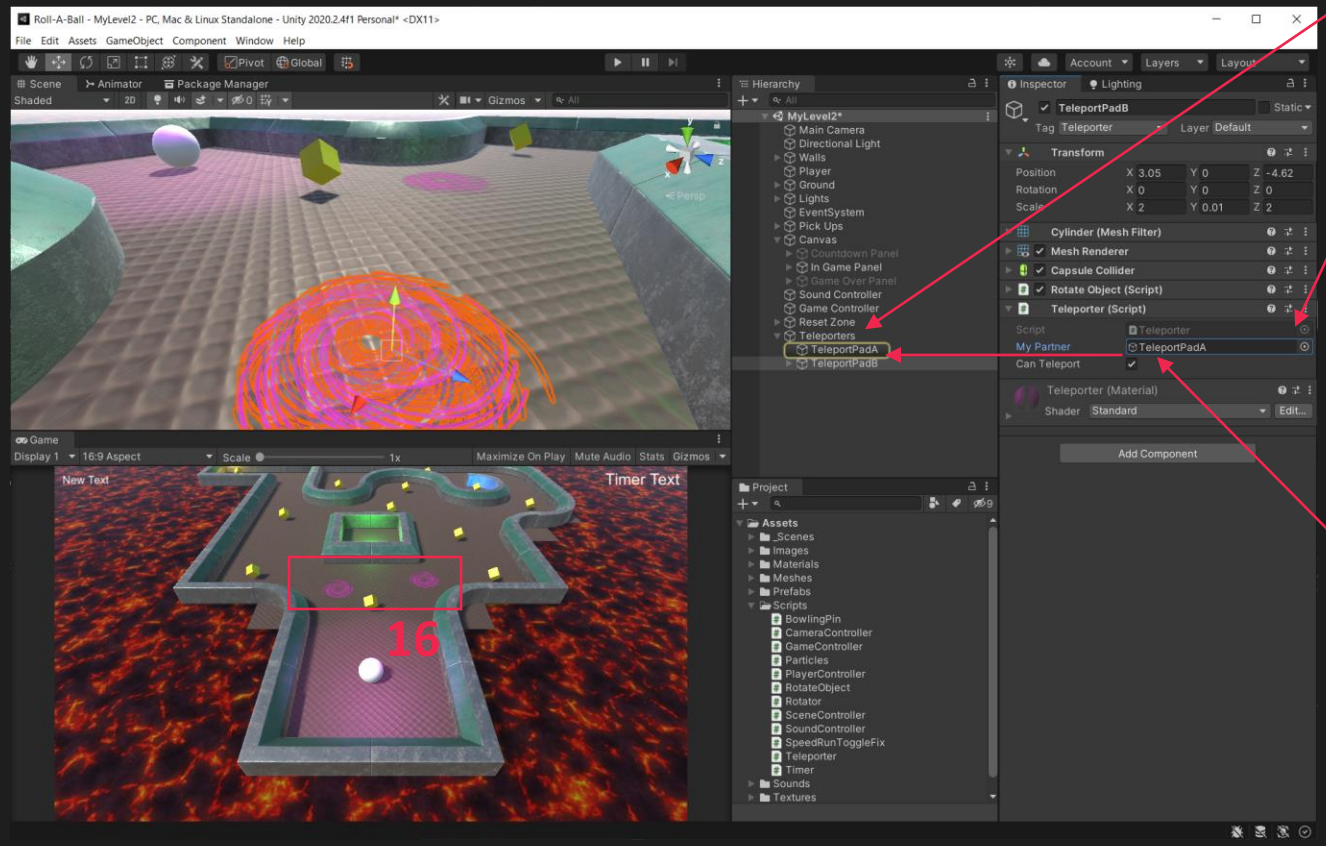


Create the Teleporter script

13. Create a script called Teleporter and copy as below.
What we are doing is checking for a trigger on an object tagged player. If true we set false our partners canTeleport.
Once the player has left our partners trigger zone, they can teleport again.
This is needed to prevent OnTriggerEnter happening again as soon as you teleport to the new trigger zone causing an infinite teleporting loop.



Setup all in Unity



14. Create an empty GameObject named Teleporters and zero it out

15. Add the Teleporter script to the TeleportPad and drag into the new Teleporters object

16. Duplicate the TeleportPad, position the new TeleportPad elsewhere in the scene and rename them TeleportPadA & TeleportPadB

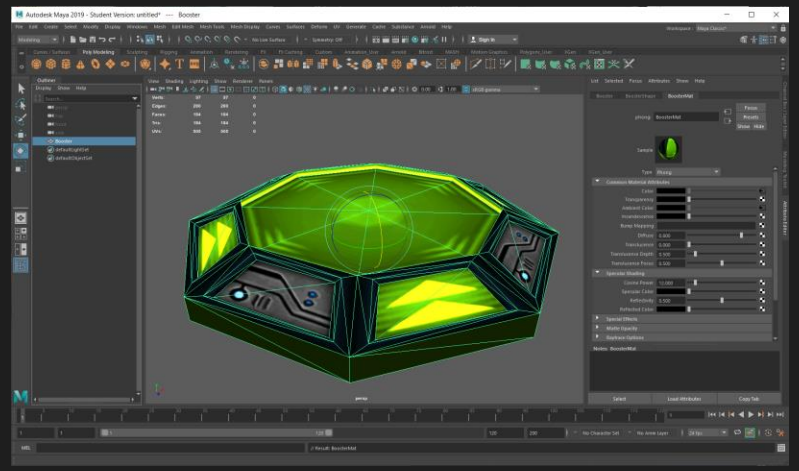
17. On each TeleportPad, drag the OTHER TeleportPad to the My Partner slot
    - TeleportPadA on TeleportPadB, etc

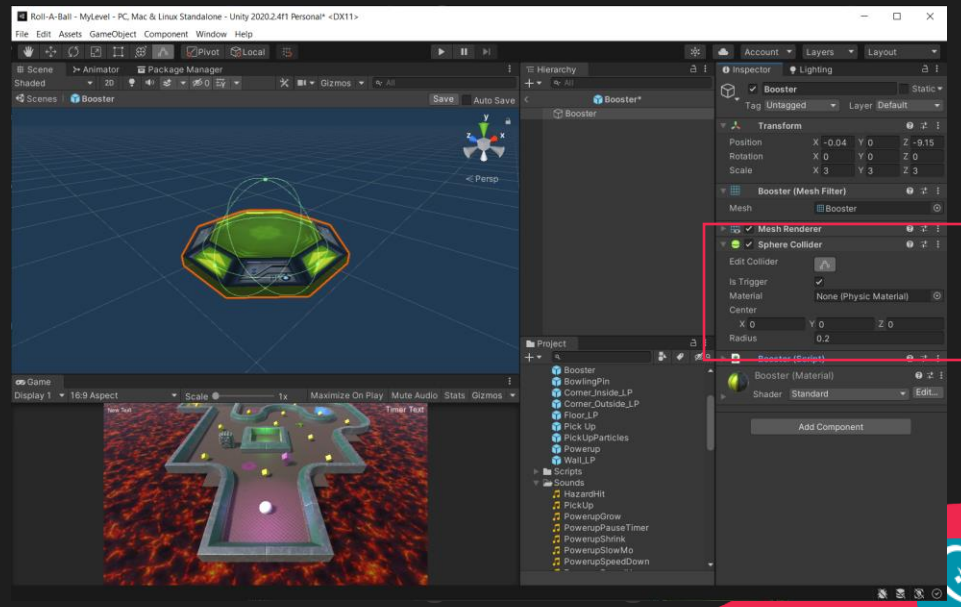
18. DONE!!!

17.Jump/Booster Pad	1pt
Pre-requisite: None	
Physical zone that transports player to another area when touched.	
Design: Texturing, Particles	
Programming: Trigger Events	

Create the Booster Model and Texture

1. Create a Model and Texture for your booster object
2. Bring the model into Unity and into your scene
3. Setup the Materials and Textures on the model
4. Name the model Booster
5. Remove the collider (if it has one) and change collider to a Sphere Collider
6. Resize the Sphere Collider to best fit your model





5, 6

Create the Booster script

7. Create a script called Booster EXACTLY as below
8. The Tooltip allows us to show a message in the inspector when we highlight a variable
9. If you are adding sound, it should go into this OnTriggerEnter function just after the line: other.attachedRigidbidy….

Booster.cs | SoundController.cs | Timer.cs | PlayerController.cs

Assembly-CSharp | Booster | OnTriggerEnter(Collider other)

```
1   using UnityEngine;
2
3   public class Booster : MonoBehaviour
4   {
5       [Tooltip("To change the boost direction, use a 1 or 0 in each x,y,z. So for forwards, use (0,0,1) ")]
6       //Default our boostDirection to up
7       public Vector3 boostDirection = new Vector3(0,1,0);
8       public float boostPower = 250;
9
10      private void OnTriggerEnter(Collider other)
11      {
12          if(other.CompareTag("Player"))
13          {
14              other.attachedRigidbody.AddForce(boostDirection * boostPower);
15          }
16      }
17  }
18
```

Modify the PlayerController script

10. Modify the PlayerController script to let us only move if we are grounded. First add a grounded bool at the top.

```
    private int pickupCount;
    GameObject resetPoint;
    bool resetting = false;
    bool grounded = true;
```

11. Update our FixedUpdate so that we can only move when grounded is true.

```
void FixedUpdate()
{
    if (resetting)
        return;

    if (grounded)
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
        rb.AddForce(movement * speed);
    }
}
```

12. We are using our OnCollisionStay to check if we are on a collider tagged “Ground”

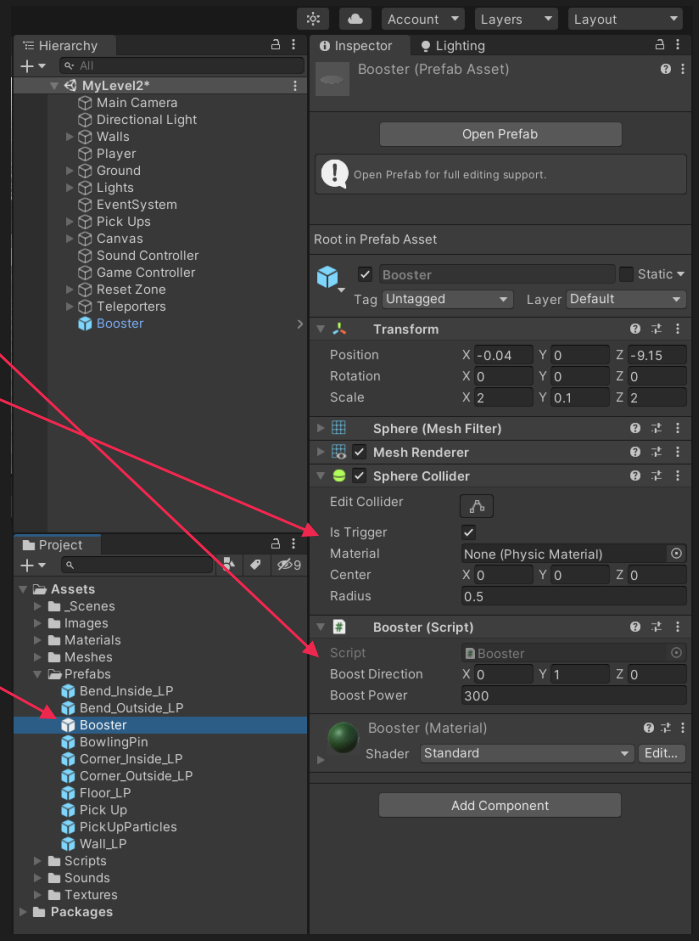
13. If we exit the Ground collider (eg in the air) then we cannot move the ball.

```
private void OnCollisionStay(Collision collision)
{
    if(collision.collider.CompareTag("Ground"))
        grounded = true;
}

private void OnCollisionExit(Collision collision)
{
    if (collision.collider.CompareTag("Ground"))
        grounded = false;
}
```

14. Make sure to tag anything else you want to control on (eg Ramps) with the tag “Ground”

Modify the PlayerController script

15. Add the Booster script to your Booster object
16. Ensure the collider is set to Is Trigger
17. Turn the Booster into a Prefab
18. Position in your game scene as desired
19. DONE!!!!



18.Bowling Pins

1pt

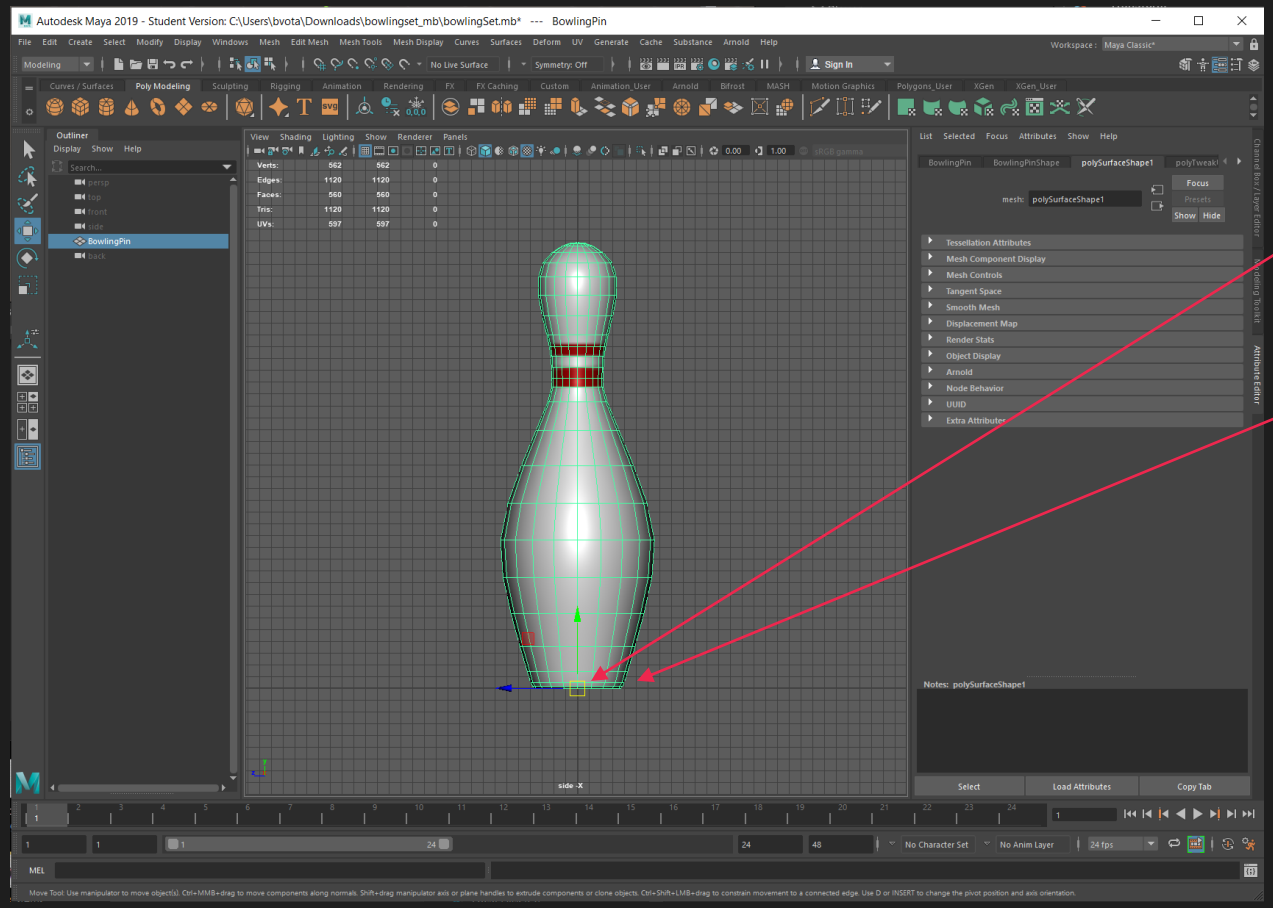
Pre-requisite: None

Collectibles with physics that do not disappear when touched, rather just turn off and scatter based on impact velocity

Design: Bowling Pin Model, Bowling Pin Texture, Bowling Ball Texture

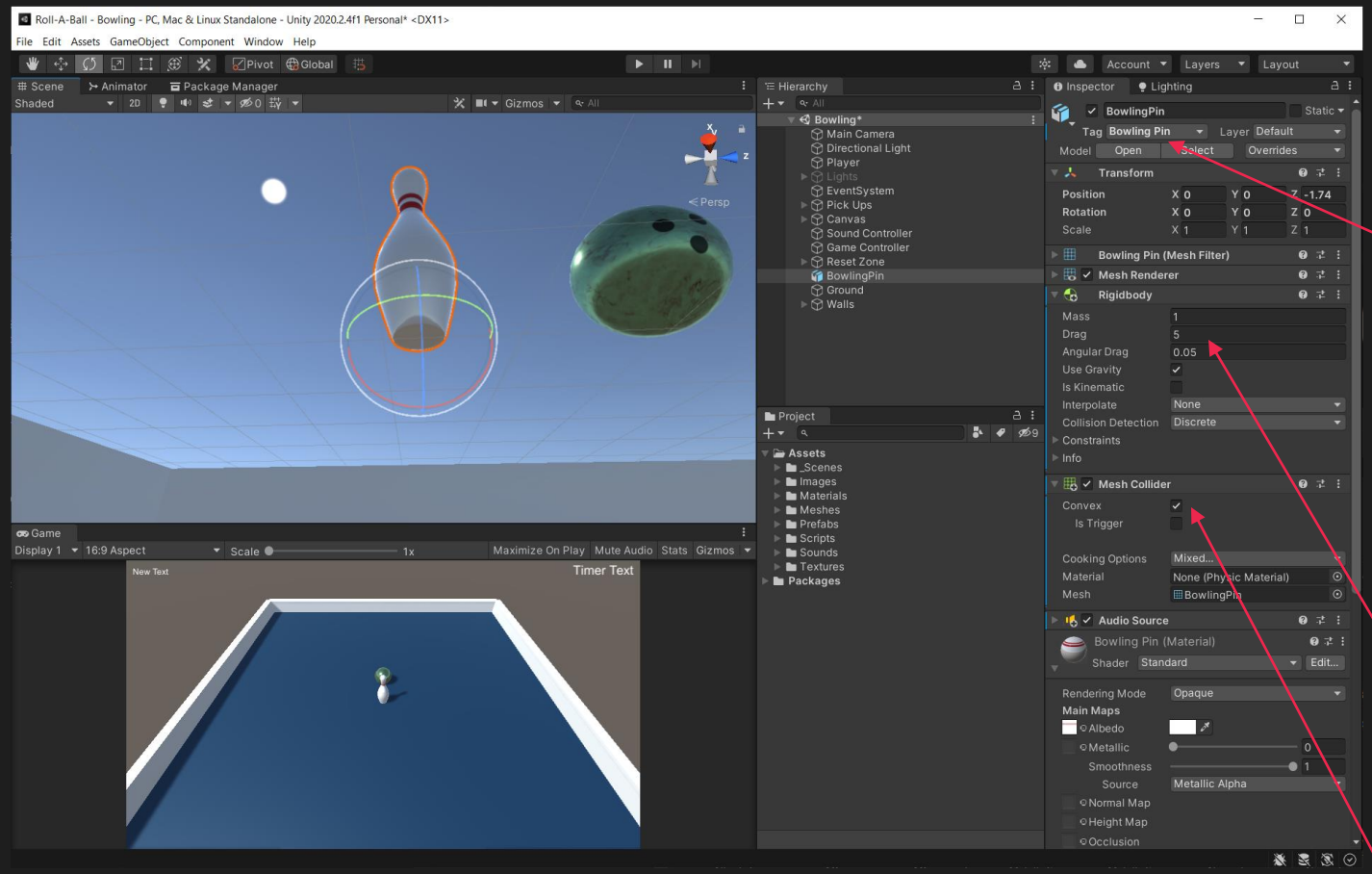
Programming: Collision Events

Model the Bowling Pin



1. Model and texture a bowling pin in Maya
2. Endure that the Pivot is at the bottom centre of the model
3. Ensure the bottom of the model has a completely flat surface

Setup the Bowling Pin



4. Setup your bowling pin model with it's material and create another new one for the Player as a bowling ball

5. Add a tag Bowling Pin and assign it to the Bowling Pin

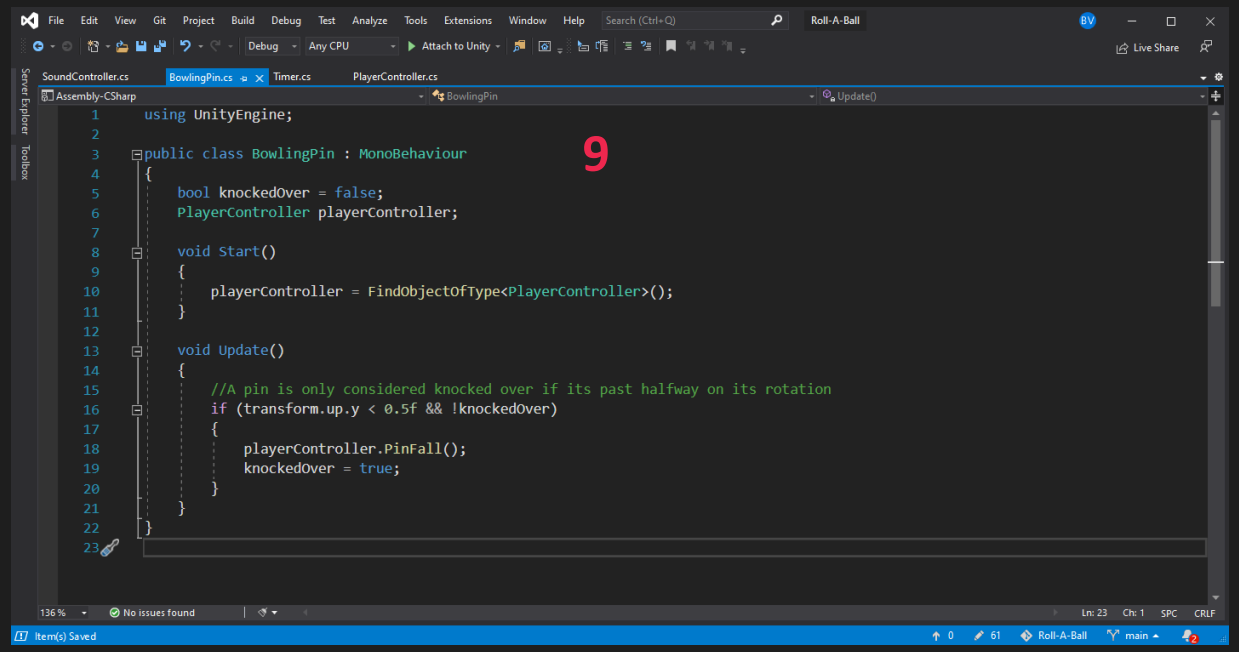
6. Add a Rigidbody and Mesh Collider component via the Add Component menu

7. Change the settings on the Rigidbody so the model doesn't fall over when you press play
   - You will need to tweak these to your model

8. Turn on Convex in the Mesh Collider

Setup the scripts

9. Create the BowlingPin script as below




```
using UnityEngine;

public class BowlingPin : MonoBehaviour
{
    bool knockedOver = false;
    PlayerController playerController;

    void Start()
    {
        playerController = FindObjectOfType<PlayerController>();
    }

    void Update()
    {
        //A pin is only considered knocked over if its past halfway on its rotation
        if (transform.up.y < 0.5f && !knockedOver)
        {
            playerController.PinFall();
            knockedOver = true;
        }
    }
}
```

10. Modify the PlayerController scripts Start function
    - This can only be done after doing the mandatory module Dynamic number of Pickups

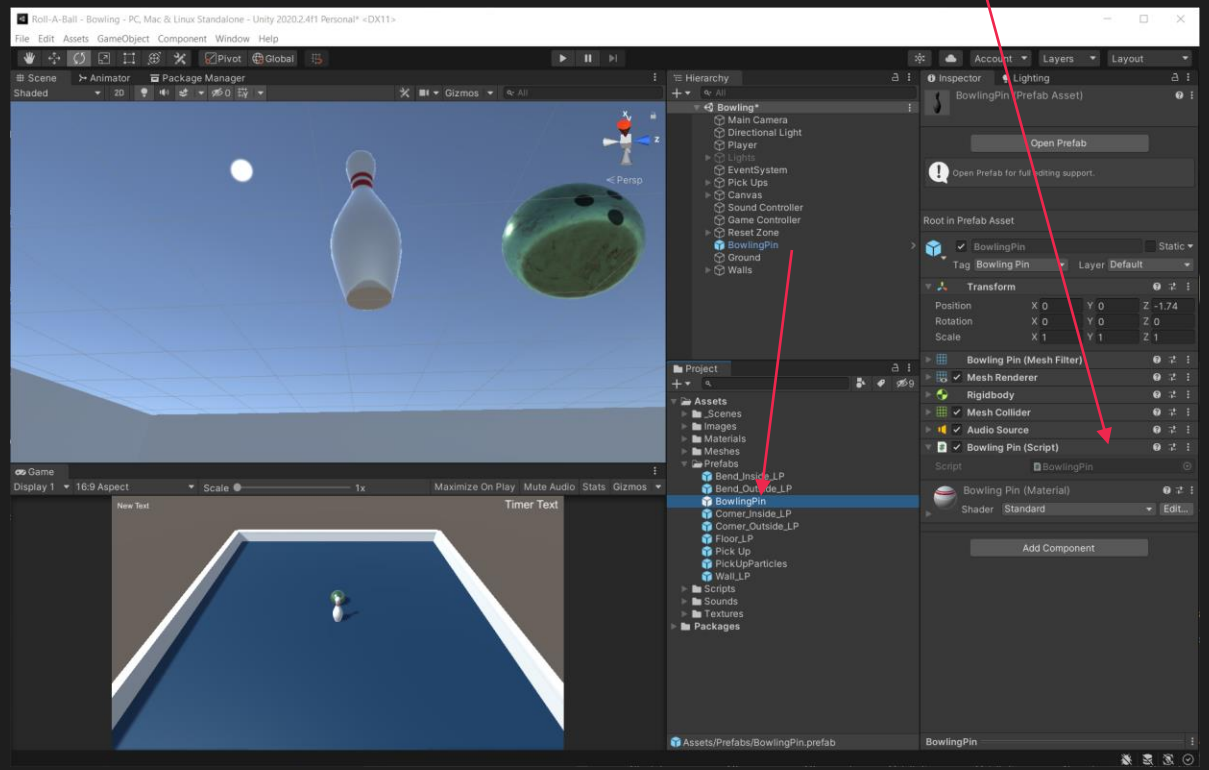
```
void Start()
{
    rb = GetComponent<Rigidbody>();
    count = 0;
    pickupCount = GameObject.FindGameObjectsWithTag("Pick Up").Length
                + GameObject.FindGameObjectsWithTag("Bowling Pin").Length;
    gameOverScreen.SetActive(false);
    SetCountText();
```

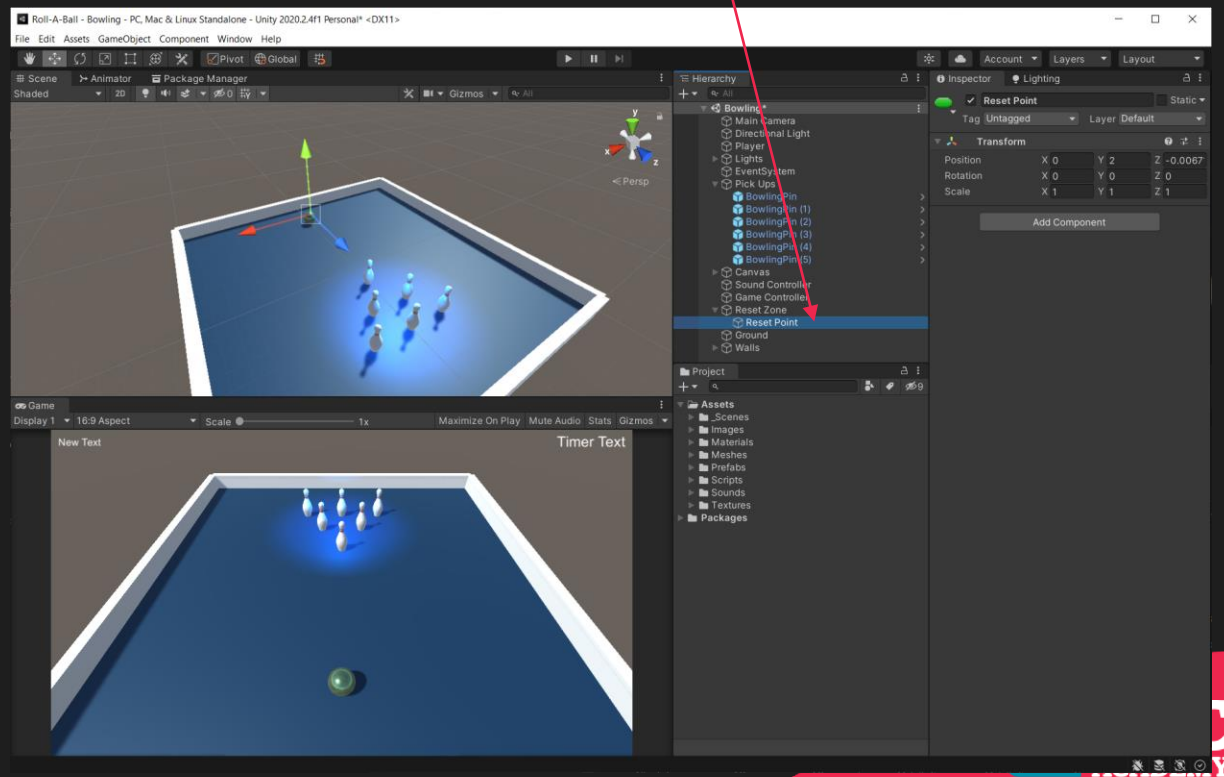
11. Add a new function to the bottom of the PlayerController called PinFall
    - In here is where you can also add sound if you do that module

```
public void PinFall()
{
    count += 1;
    SetCountText();
}
```

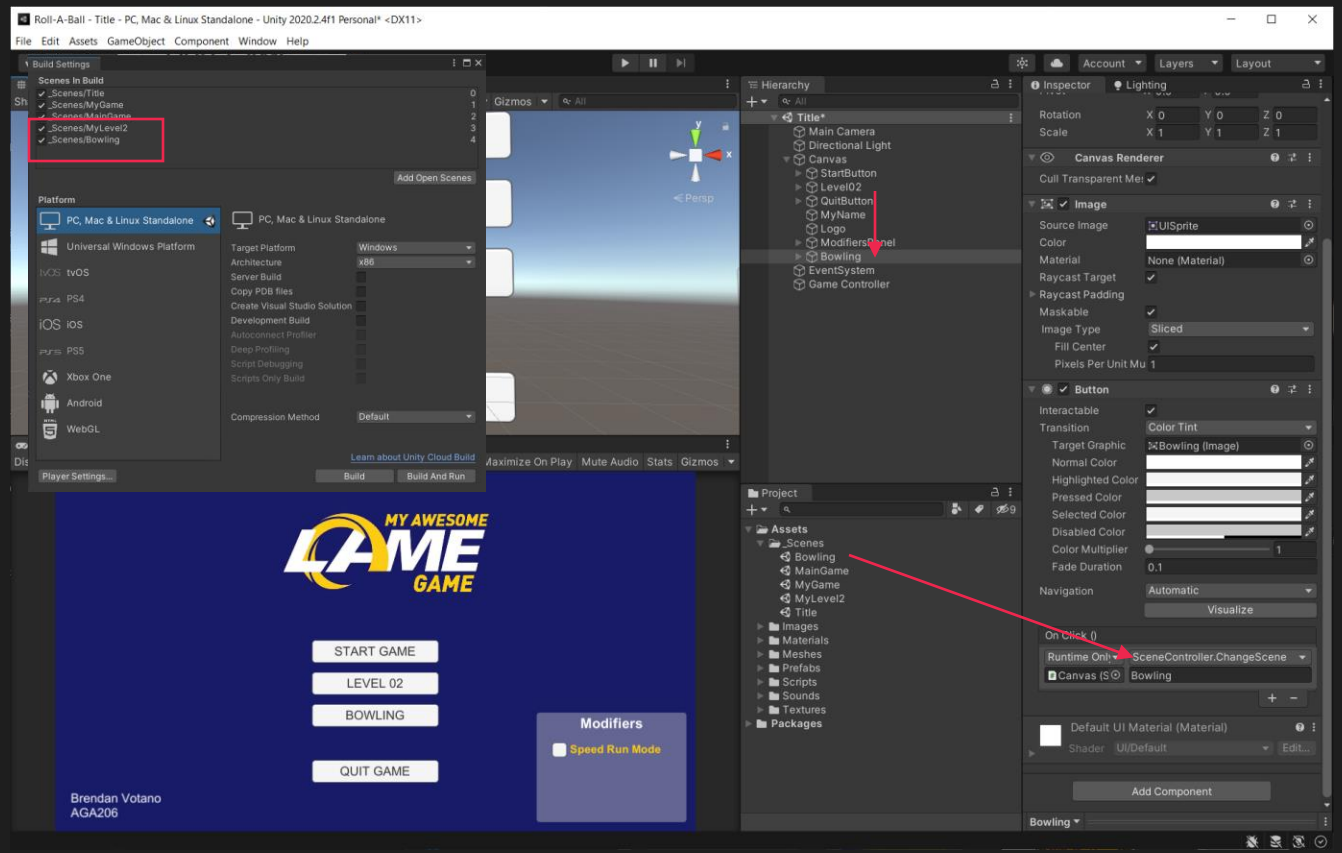
Setup the scene

12. Attach the BowlingPin script to the BowlingPin
13. Turn the BowlingPin into a prefab
14. Add more pins to the stage
15. Don't forget to move your Reset Point to somewhere in the new stages play area





Add to the title scene



16. Go to the Title scene
17. Duplicate one of the level buttons and call it Bowling. Position it appropriately
18. In the Bowling buttons On Click event, change the text to the EXACTLY the same as the Bowling levels name
19. Make sure you add the new scene to the Build Settings!
20. Congratulations. You are now DONE!!!

NOTE: You may or may not have some of these elements already depending on other modules

19. Moving Hazard

1pt

Pre-requisite: Reset Zone

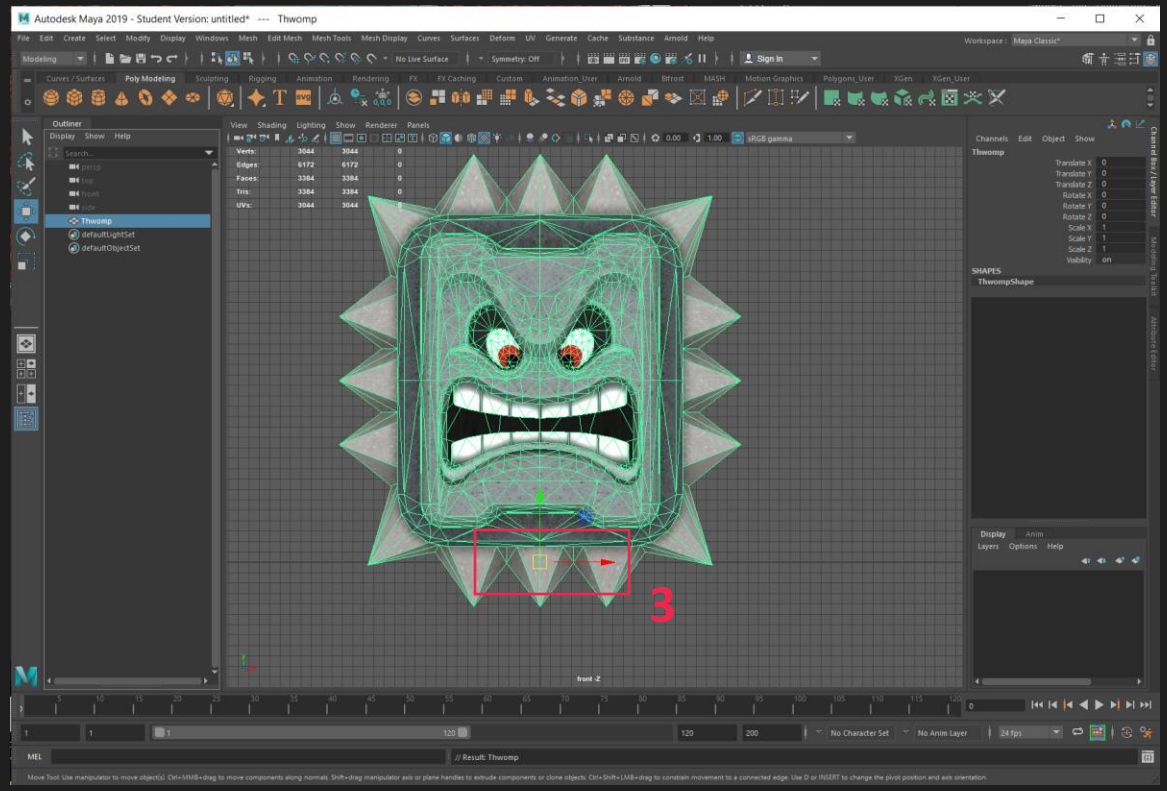
A block that moves between a series of points, and if you touch it's collider you reset

Design: Model and Texture a hazard

Programming: Collision Events, Coroutines

Create the Hazard model

1. Model a hazard object in Maya
2. Texture the object appropriately
3. Ensure the pivot is at the centre bottom of the object.
4. Export the object as an .fbx called Hazard and import into Unity

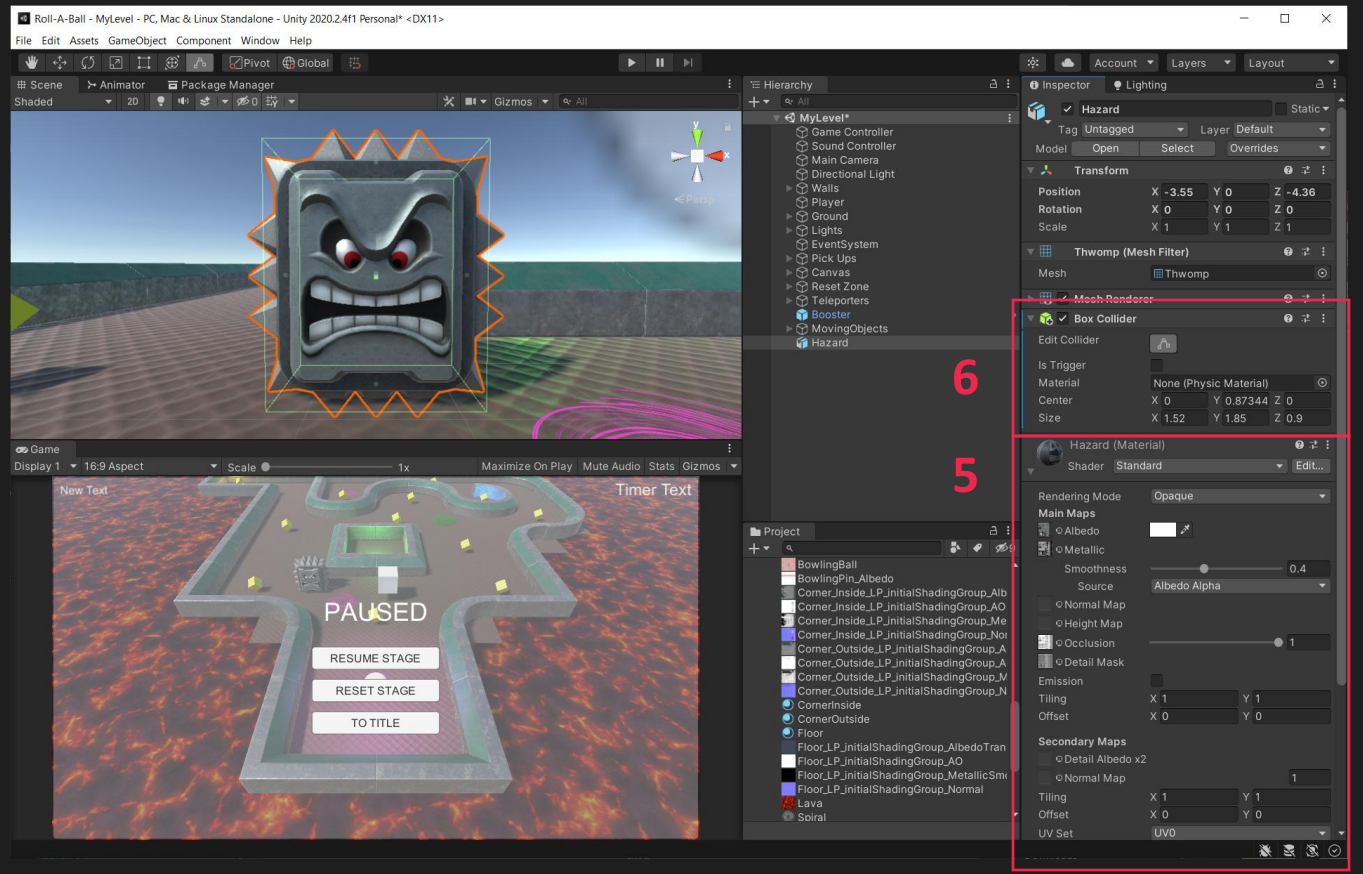


Create/Modify the GameController script

5. Setup the object in your scene with the materials and scale as needed

6. Call the object Hazard if it isn't already

7. Add a Box Collider (or the appropriately shaped one to your model) and size it to fit



Create the MovingObjects script

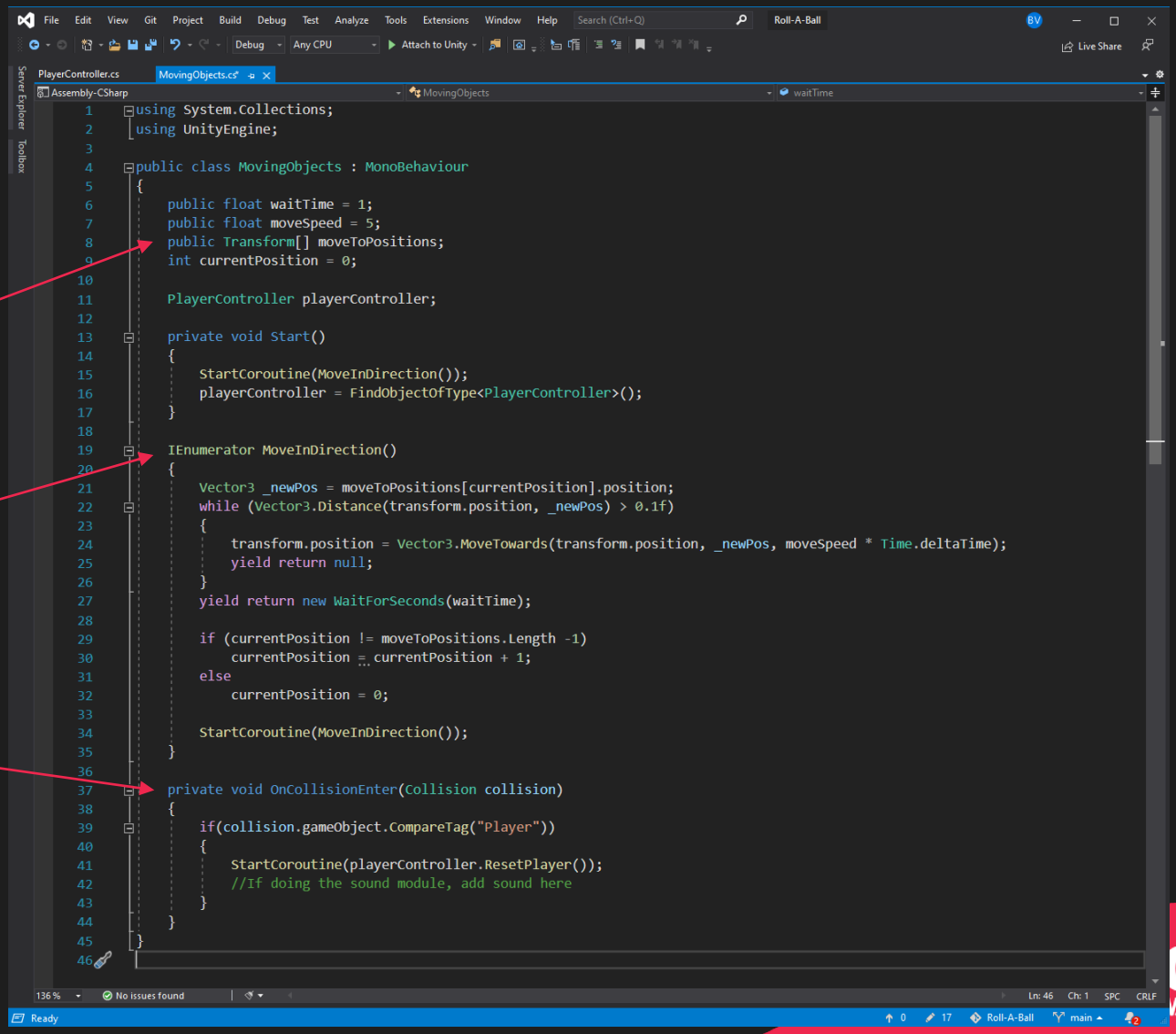
8. Create a script called MovingObjects

9. Copy it as in the image

10. This script takes an array of positions for us to move through.

11. Using a coroutine, we move to the current position in our array, each time incrementing the destination to the next position before running the coroutine again.

12. If we collide with an object tagged “Player”, we then Reset the Player
    - This function was completed in the Mandatory module Reset Zone and must be completed before this module.

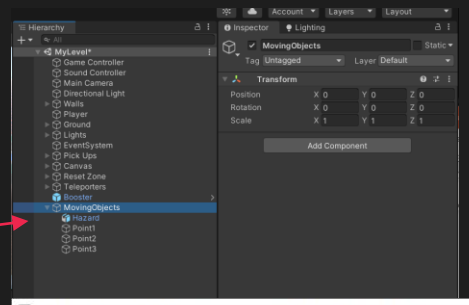


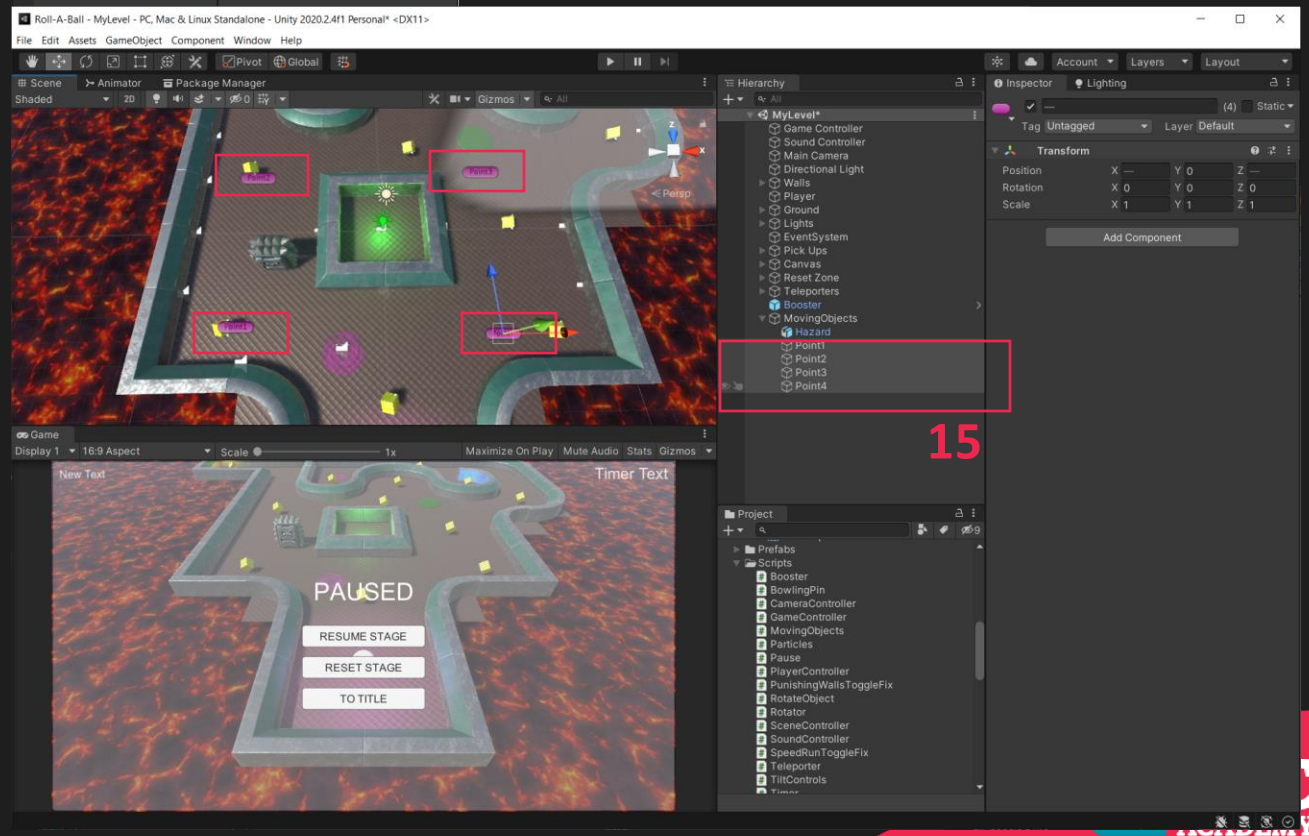
Setting it up in the scene

13. Create an empty Game Object called MovingObjects and zero it out (x = 0, y = 0, z = 0)

14. Move the hazard object into it as a child

15. Create a number of points for the object to move between (minimum 2) and ensure their y positions are at 0 (This will keep the object moving along the ground provided the objects pivot was set at the bottom of the mesh)





Making the connections

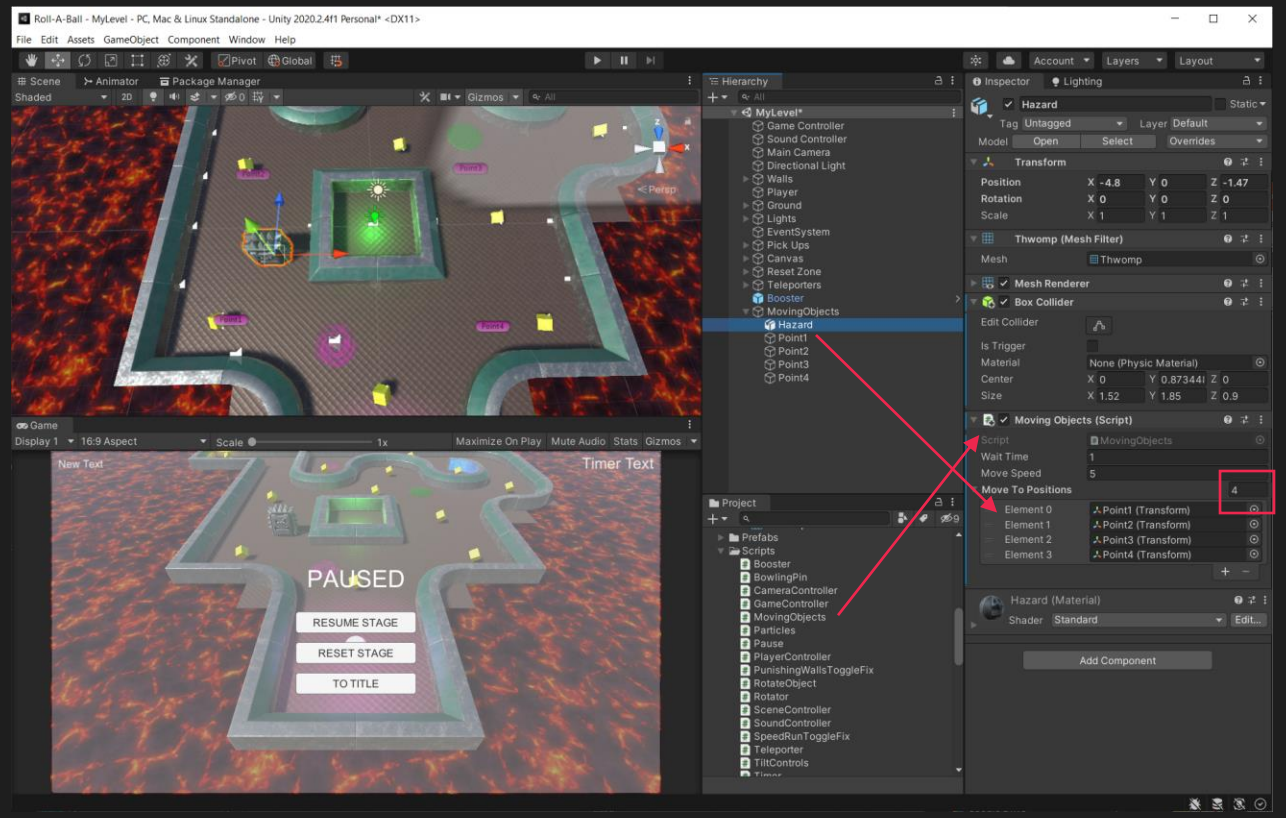
16. Drag the Moving Objects script onto the Hazard Game Object

17. Make the Move To Positions array the size of your move to points.

18. Drag each Move To Point game object from the hierarchy to each slot on the script

19. Adjust the Move Speed and Wait Time to your liking

20. DONE!!!



20. Punishing Walls

1pt

Pre-requisite: None

Add an option to the title screen that turns all walls into punishing walls that reset the player if touched.

Design: None

Programming: Collision Events, Game Controllers

Create/Modify the GameController script

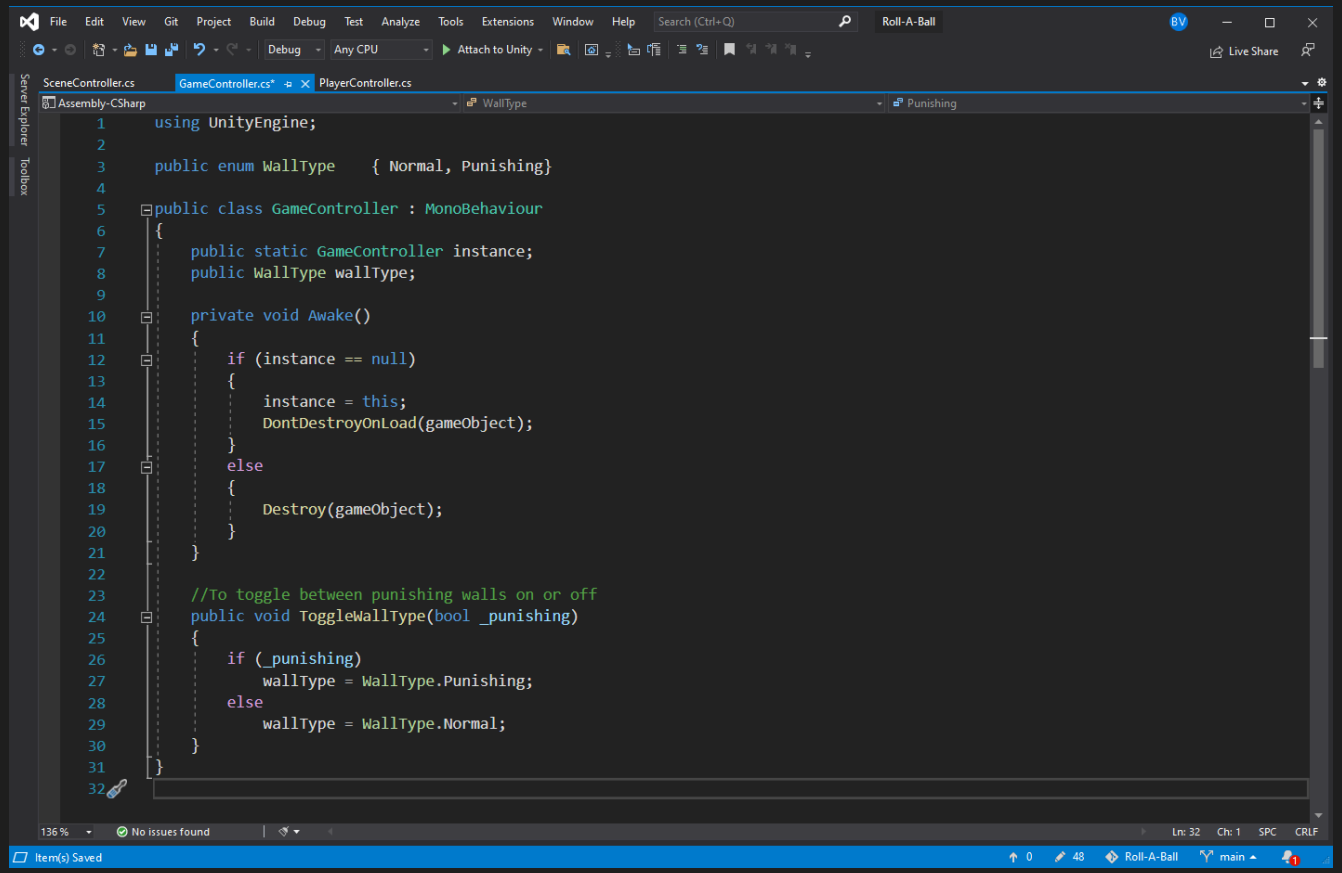
1. Create a script called GameController

You may already have the GameController script from a previous module.

If so, fill in the parts from the script to the side in the appropriate places

2. Write out the GameController script as follows

3. There is a lot of new syntax in here but don't worry if you don't fully understand it



Add to our PlayerController script

4. Add a reference to our GameController in the variable declaration section at the top
   - NOTE: You may not have some of these such as SoundCountroller or timer if you have not done certain modules

5. Add the line to find the GameController in the start function

6. Add the lines to the FixedUpdate function that will exit the function should our control type be WorldTilt

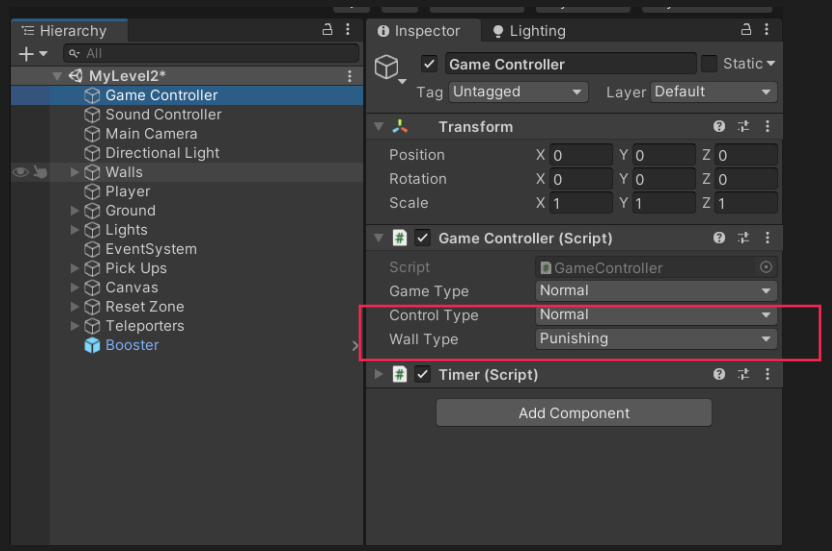
```
    bool grounded = true;

    //Controllers
    SoundController soundController;
    GameController gameController;
    Timer timer;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        count = 0;
        pickupCount = GameObject.FindGameObjectsWithTag("Pick Up").Length
                    + GameObject.FindGameObjectsWithTag("Bowling Pin").Length;
        gameOverScreen.SetActive(false);
        SetCountText();
        winText.text = "";
        resetPoint = GameObject.Find("Reset Point");
        gameController = FindObjectOfType<GameController>();
    }
```

```
    private void OnCollisionEnter(Collision collision)
    {
        if(collision.gameObject.CompareTag("Respawn"))
        {
            StartCoroutine(ResetPlayer());
        }

        if (collision.gameObject.CompareTag("Wall"))
        {
            if (gameController.wallType == WallType.Punishing)
                StartCoroutine(ResetPlayer());
        }
    }
```

Unity scene setup

7. Create an empty Game Object, call it Game Controller and zero it out
   - You may already have this set up from another module

8. Change the Wall Type to Punishing to test



8 NOTE: You may not have some of these options depending on other modules you have done

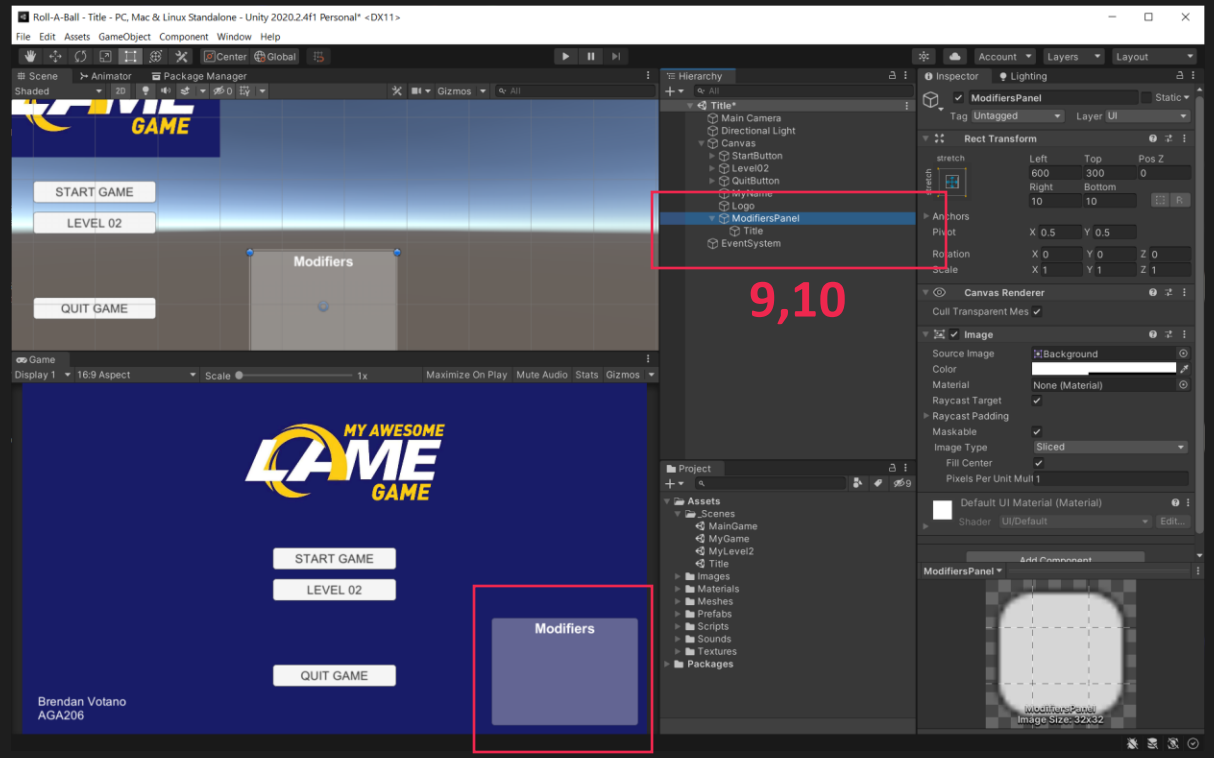
- NOTE: You will need to make sure you have the GameController script in every scene now.

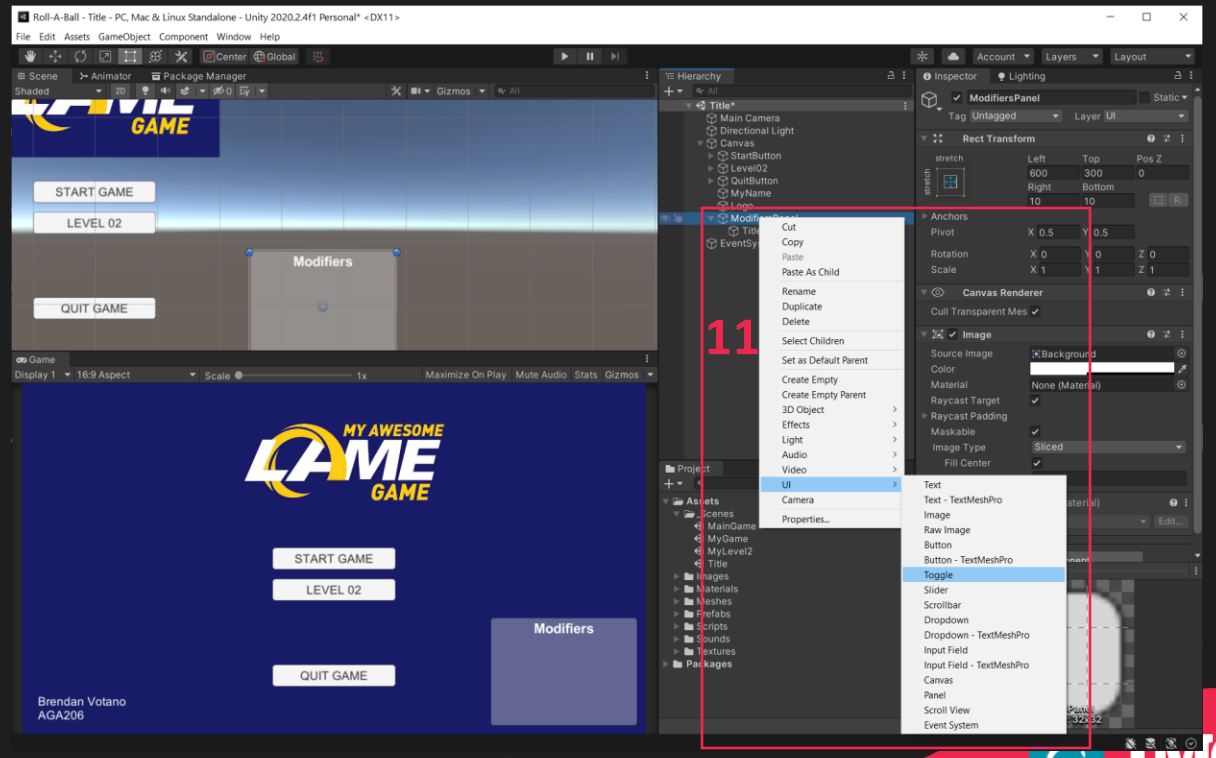
Adding to the Title Scene

9. Back in our Title scene, in our Canvas create a Panel called ModifiersPanel and position it like so.
10. Add a UI text to this panel called Title and change the text to Modifiers

NOTE: You may or may not have this already

11. Highlight the ModifiersPanel and then right click to create a new UI > Toggle



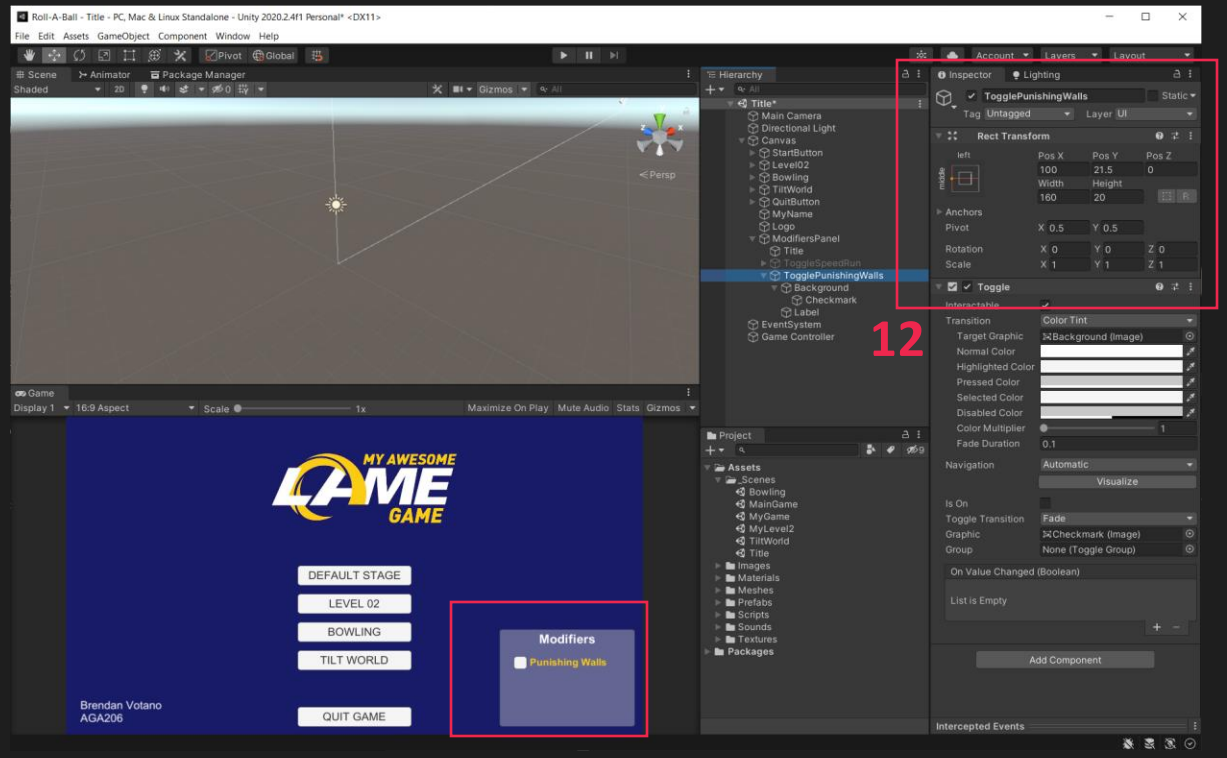


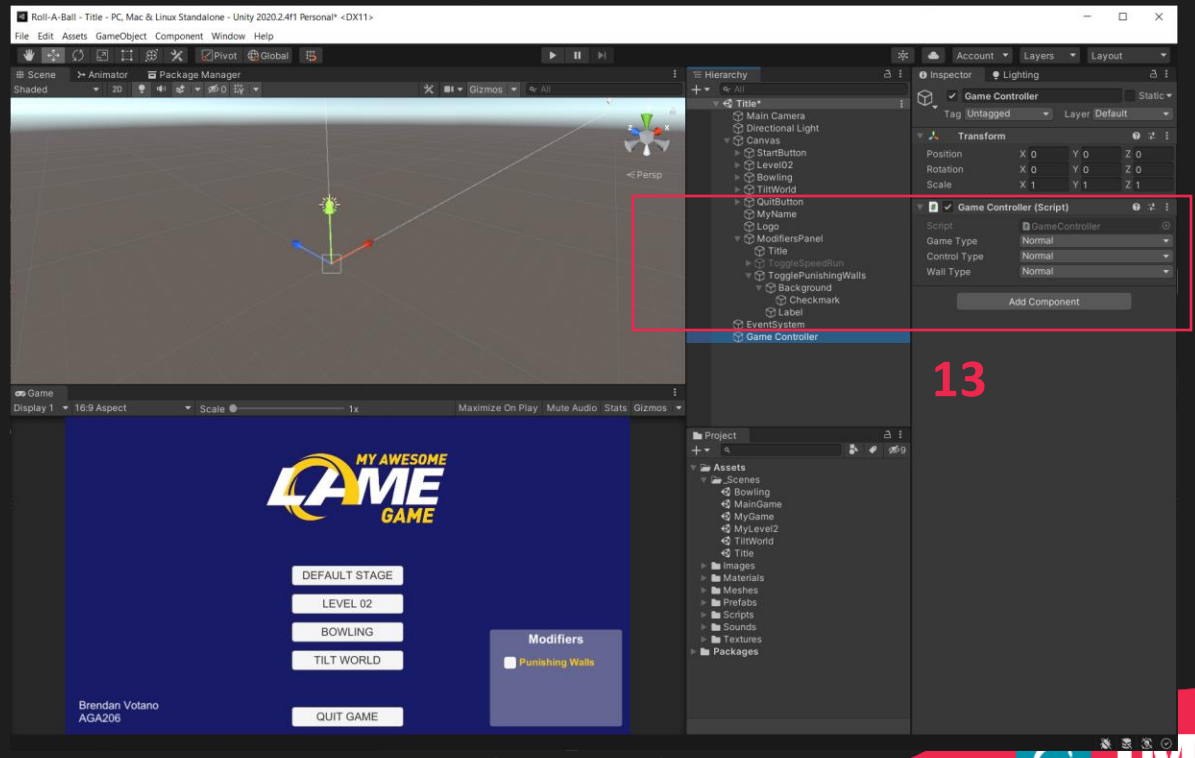
Adding to the Title Scene

12. Position and change colours, font, etc to something like the following

13. Create an empty GameObject at the root called Game Controller and drag on the GameController script

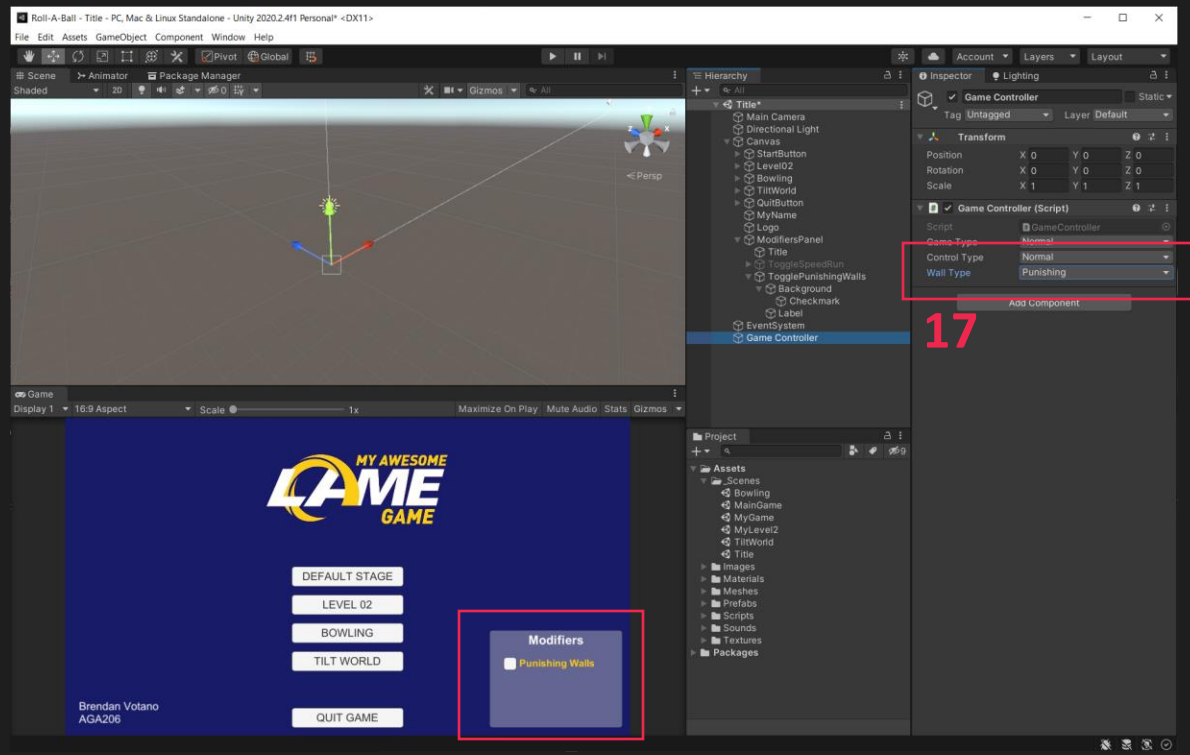
NOTE: You may or may not have some of these elements already depending on other modules





Getting the Toggle to work

14. Redundant
15. Redundant
16. Redundant
17. When you play the scene and change the toggle, you should see it update on the GameController
    - However, there is an issue we must fix. The toggle defaults to on every time we run the game.
    - We need to get it to set according to the saved GameController settings



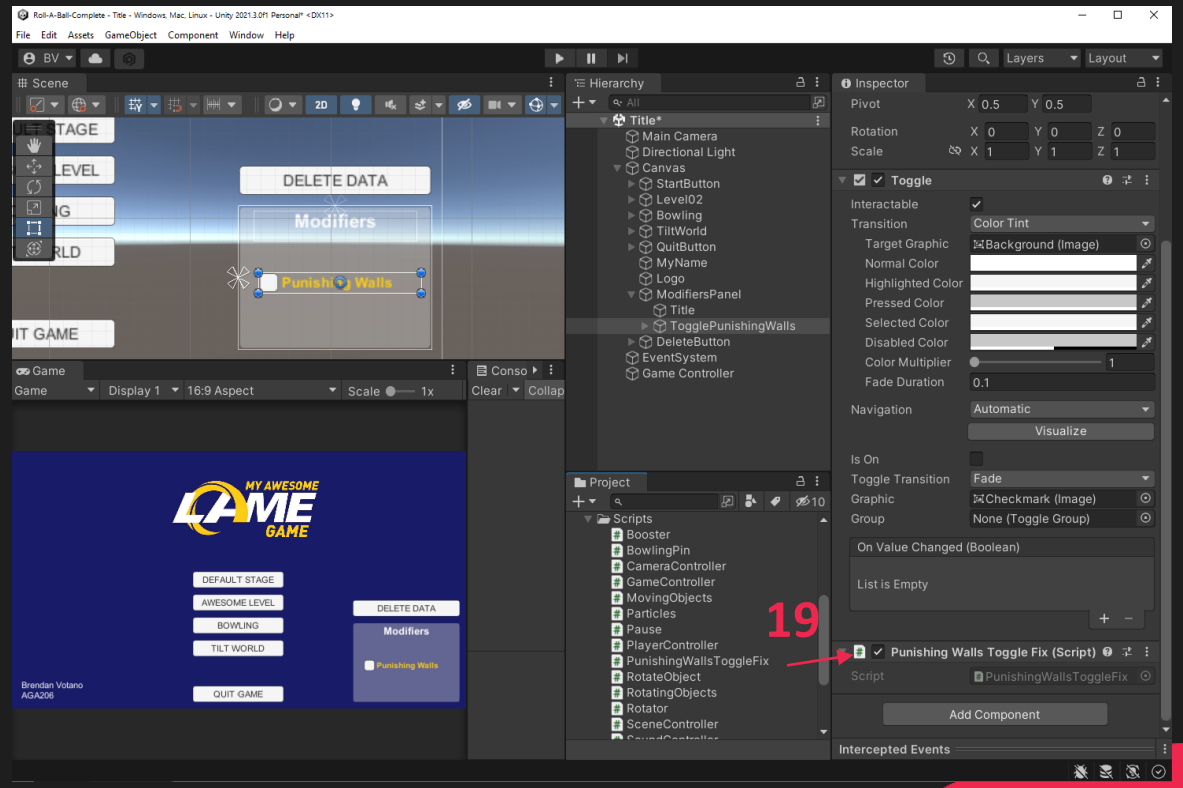
NOTE: You may or may not have some of these elements already depending on other modules

Hacky toggle fix

18. Create a script called PunishingWallsToggleFix and fill it in EXACTLY as follows

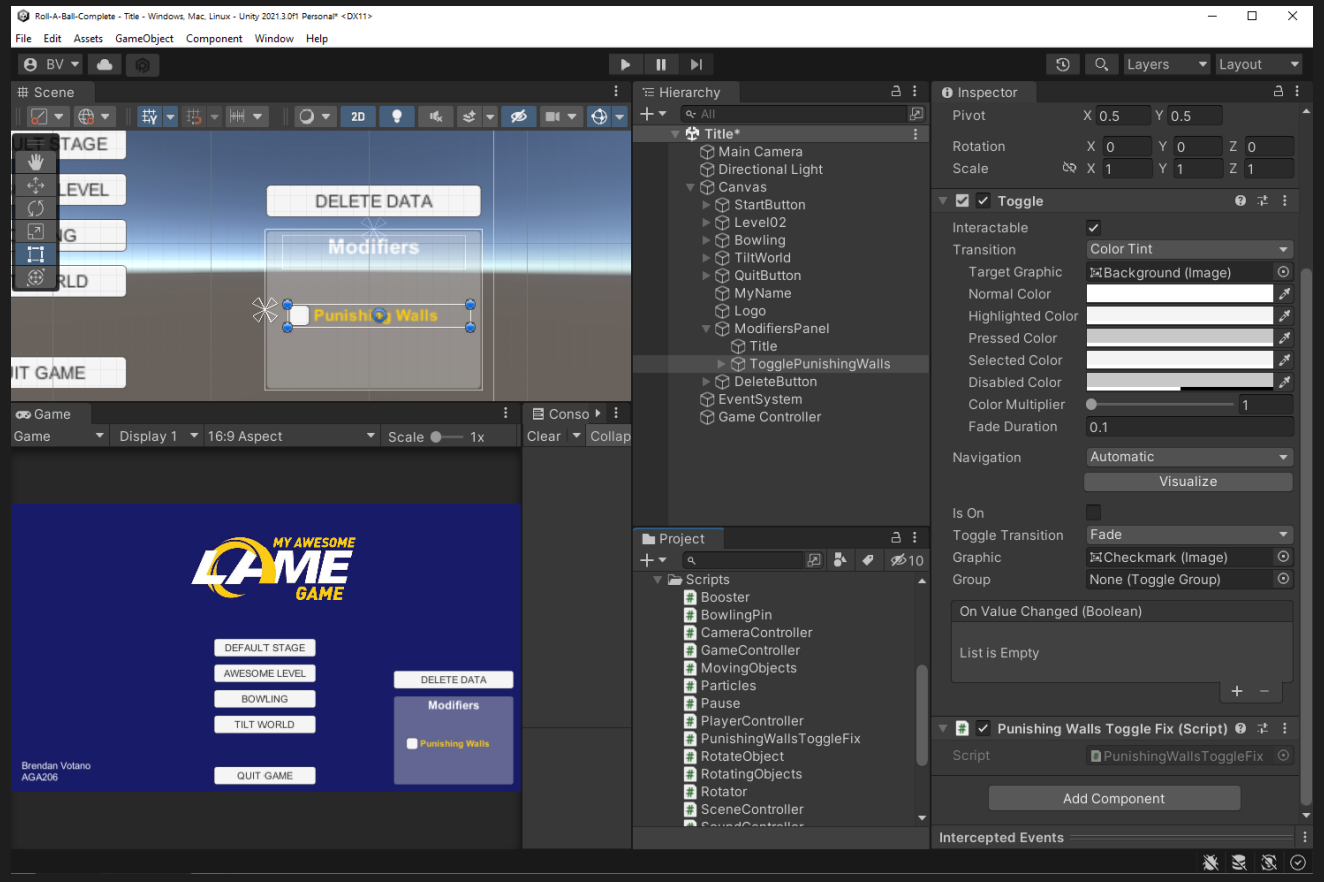
```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

0 references
public class PunishingWallsToggleFix : MonoBehaviour
{
    GameController gameController;
    Toggle toggle;

    0 references
    void Start()
    {
        gameController = FindObjectOfType<GameController>();
        toggle = GetComponent<Toggle>();
        StartCoroutine(FixWallToggle());
    }

    1 reference
    IEnumerator FixWallToggle()
    {
        yield return new WaitForEndOfFrame();
        if (gameController.wallType == WallType.Punishing)
            toggle.isOn = true;
        else
            toggle.isOn = false;

        toggle.onValueChanged.AddListener((value) => gameController.ToggleWallType(toggle.isOn));
    }
}
```

19. Drag the PunishingWallsToggleFix script to the TogglePunishingWalls object



DONE!!!

- The punishing walls mode toggle now should apply to every level in your game



21. Free Camera Mode

2pts

Pre-requisite: None

Have the ability to choose between fixed or free camera mode

Design: None

Programming: Camera Controllers, Player Controllers

Modify the CameraController script

1. Add the following new sections to the Camera Controller script
2. Ensure that the enum is at the top just under the using UnityEngine section

```
4
                4 references
5       public enum CameraStyle {Fixed, Free}
                2 references
6       public class CameraController : MonoBehaviour
7       {
8           public GameObject player;
9           public CameraStyle cameraStyle;
10          public Transform pivot;
11          public float rotationSpeed = 1f;
12
13          private Vector3 offset;
14          private Vector3 pivotOffset;
15
```

Modify the CameraController script

1. Add the following new lines to the Start() function

```
void Start()
{
    //The offset of the pivot from the player
    pivotOffset = pivot.position - player.transform.position;
    //The offset from the player
    offset = transform.position - player.transform.position;
}
```

2. Rewrite the LateUpdate() function to what is on the right

```
void LateUpdate()
{
    //If we are using the fixed camera mode
    if (cameraStyle == CameraStyle.Fixed)
    {
        //Set the cameras position to be the players position plus the offset
        transform.position = player.transform.position + offset;
    }

    //If we are using the free camera mode
    if (cameraStyle == CameraStyle.Free)
    {
        //Make the pivot position follow the player
        pivot.transform.position = player.transform.position + pivotOffset;
        //Work out the angle from the mouse input as a quaternion
        Quaternion turnAngle = Quaternion.AngleAxis(Input.GetAxis("Mouse X") * rotationSpeed, Vector3.up);
        //Modify the offset by the turn angle
        offset = turnAngle * offset;
        //Set the camera position to that of the pivot plus the offset
        transform.position = pivot.transform.position + offset;
        //make the camera look at the pivot
        transform.LookAt(pivot);
    }
}
```

Modify the PlayerController script

1. Add the following new variable at the top
   - NOTE: you may not have all these other variables in your script based on what modules you have done.

```
Color originalColour;
bool grounded = true;

//Controllers
SoundController soundController;
GameController gameController;
CameraController cameraController;
Timer timer;
```

2. Add the following line to the Start() function
   - NOTE: you may not have all these other lines in your script based on what modules you have done.

```
void Start()
{
    rb = GetComponent<Rigidbody>();
    count = 0;
    pickupCount = GameObject.FindGameObjectsWithTag("Pick Up").Length
                + GameObject.FindGameObjectsWithTag("Bowling Pin").Length;
    gameOverScreen.SetActive(false);
    SetCountText();
    winText.text = "";
    resetPoint = GameObject.Find("Reset Point");
    originalColour = GetComponent<Renderer>().material.color;

    soundController = FindObjectOfType<SoundController>();
    gameController = FindObjectOfType<GameController>();
    cameraController = FindObjectOfType<CameraController>();

    timer = FindObjectOfType<Timer>();
    if (gameController.gameType == GameType.SpeedRun)
        StartCoroutine(timer.StartCountdown());
}
```

Modify the PlayerController script

1. Add the following section to the FixedUpdate() function
   - NOTE: you may not have all these other lines in your script based on what modules you have done.

```
void FixedUpdate()
{
    if (resetting)
        return;

    if (gameController.gameType == GameType.SpeedRun && !timer.IsTiming())
        return;

    if (gameController.controlType == ControlType.WorldTilt)
        return;

    if (grounded)
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

        if(cameraController.cameraStyle == CameraStyle.Free)
        {
            //rotates the player to the direction of the camera
            transform.eulerAngles = Camera.main.transform.eulerAngles;
            //translates the input vectors into coordinates
            movement = transform.TransformDirection(movement);
        }

        rb.AddForce(movement * speed);
    }
}
```

Unity Setup

1. Create an empty game object, call it Pivot, then position the new Pivot gameObject at the same position as the Player
2. Drag the Pivot to the Pivot slot on the CameraController, and change the Camera Style to Free