2D Platformer Workshop

By Peter Mitchell

Contents

1 Foreword

This document in combination with the practical demonstration during the workshop and provided materials along with the material for following weeks are intended to introduce the basics of the Unity Game Engine. You can find the full project files on GitHub at:

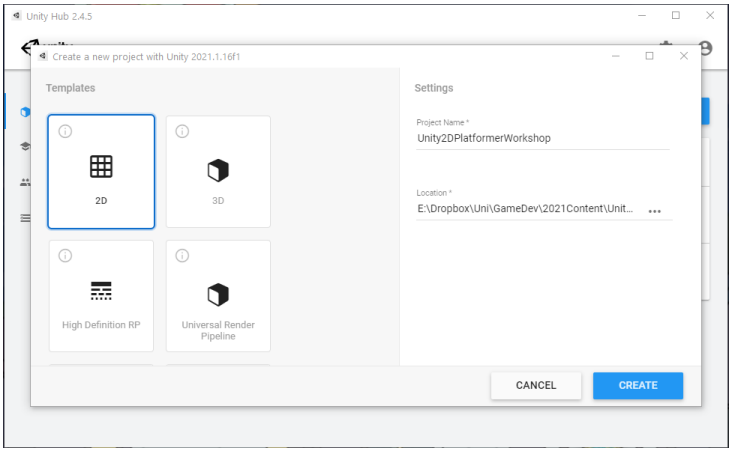
https://github.com/Squirrelbear/Unity2DPlatformerWorkshop

Unity Version Used: 2021.1.16f1

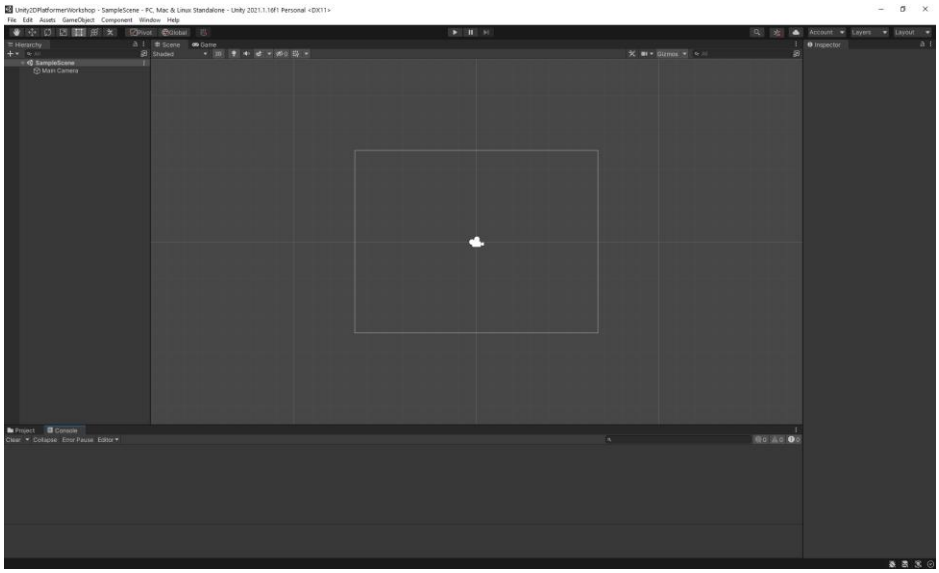
2 Setup and Project Introduction

2.1 Create 2D Project

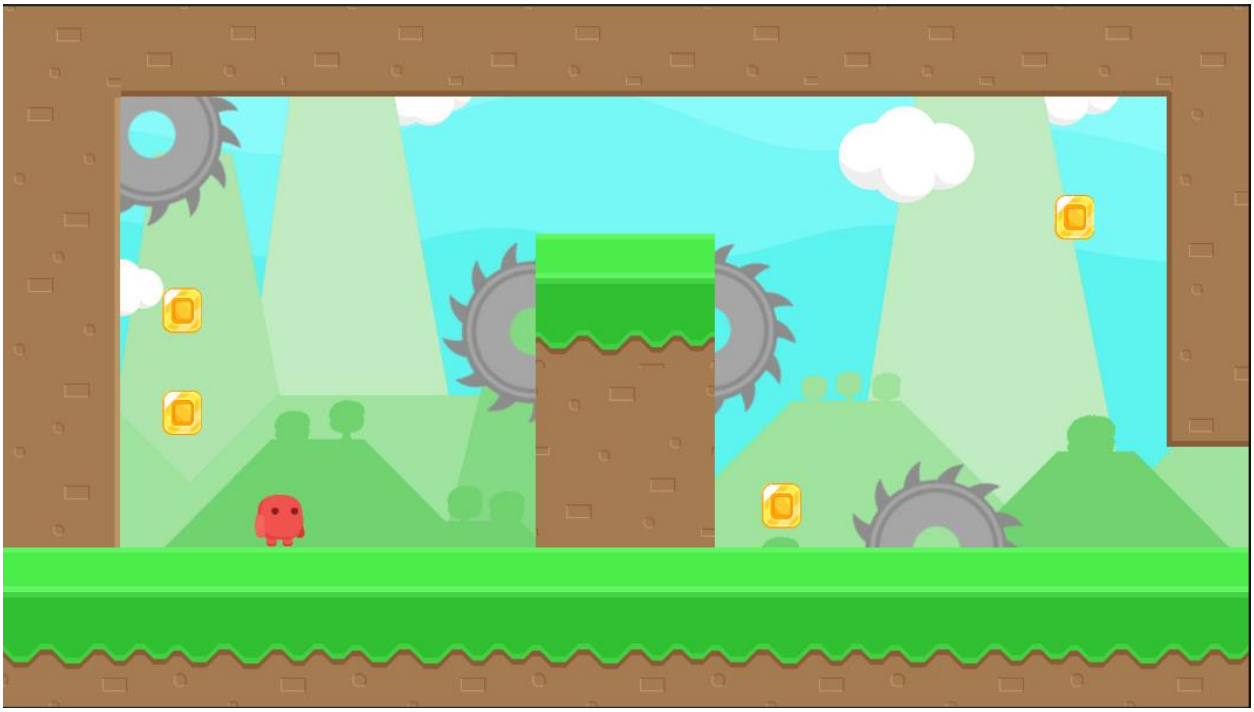
1. Create a new Unity Project and make sure you select 2D as seen below.



2. You will see something like the below once the project is created with a 2D scene containing a single Main Camera in the hierarchy.



2.2 Preview of What is Being Made

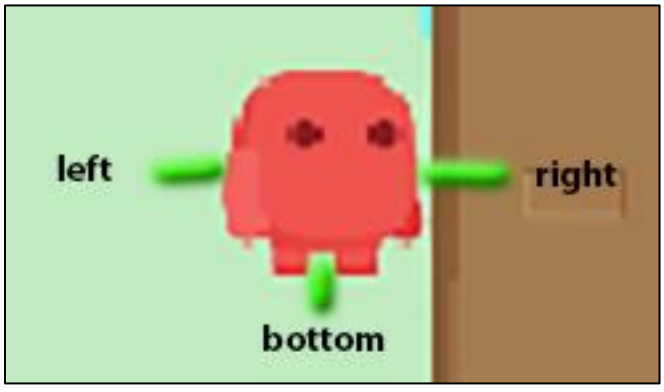


Above you can see a view of the game that is being made as part of this workshop.

2.3 Super Meat Boy Style Character Controller Overview

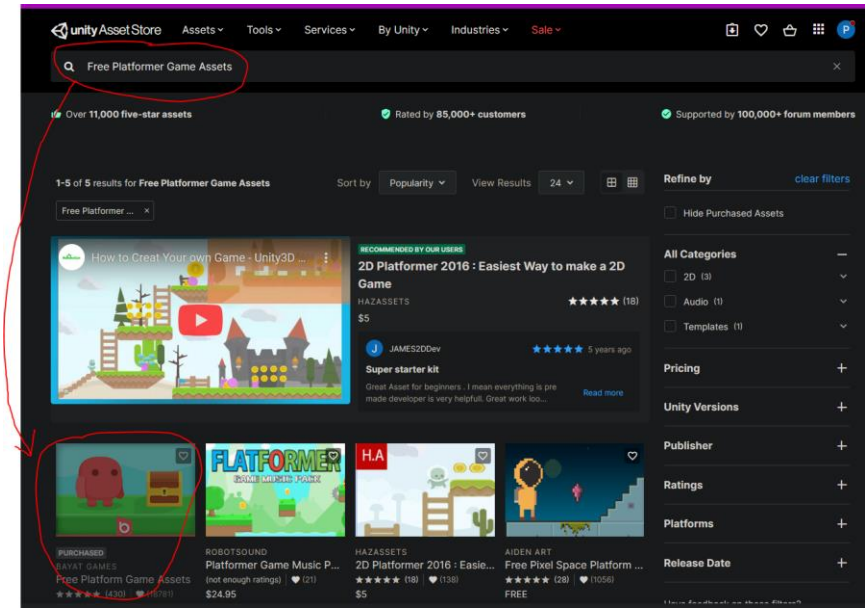
The controller is a very simple one that instantly applies forces when actions occur. The movement is managed with checks of objects around the controller by testing with what are called Raycasts.

Raycasts check for walls/ground using invisible lines checking if there is a collision with anything in that direction. As you can see in the badly drawn images below. When the bottom raycast finds a collision, the object is on the ground. When the left and/or right raycasts are colliding with an object there is a wall there. This allows checking to make the character “glue” onto walls using velocity changes. The controller also applies forces for horizontal acceleration and changes the velocity when a jump occurs.

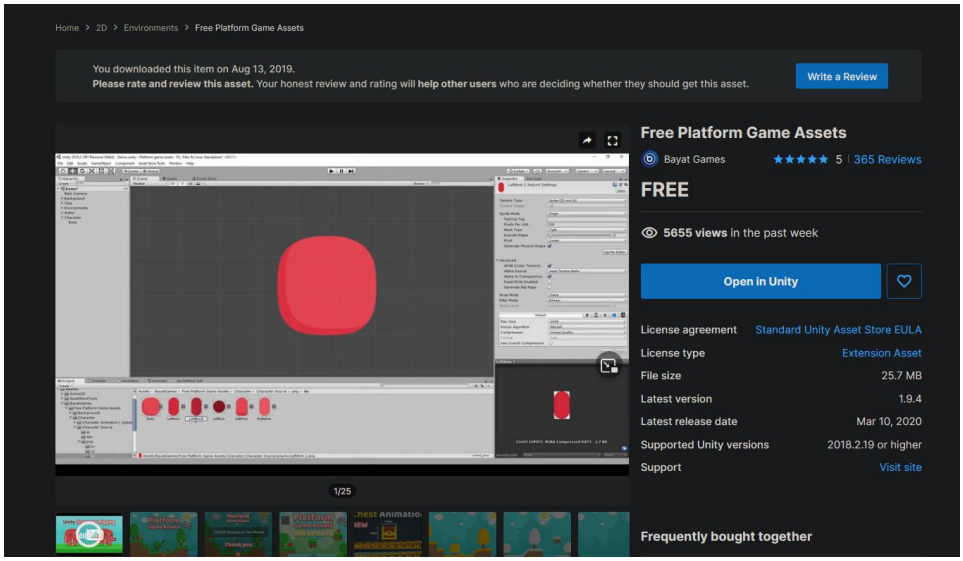


2.4 Adding Content Using the Asset Store

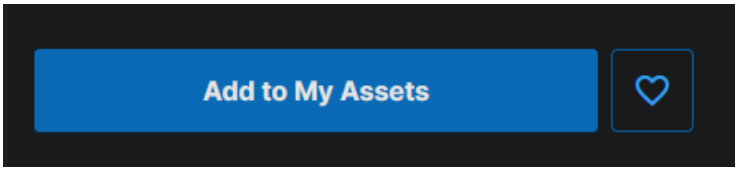
3. Unity recently moved all their asset store viewing from inside the editor to be web only. In older versions of Unity, you can interact with the asset store by going to Window -> Asset Store. Now when you go there you will find a page saying the Asset Store has moved.
4. Go to https://assetstore.unity.com/
5. You should create a Unity account if you haven't already and log in.
6. Search for "Free Platform Game Assets". And you will see the asset appear as circled below.



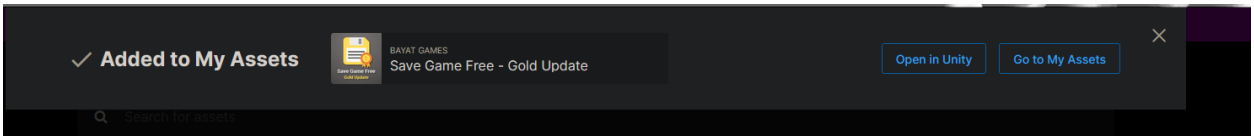
7. Select the image or the name of the asset and you will be taken to the page for that asset. Then you will see something like seen below.



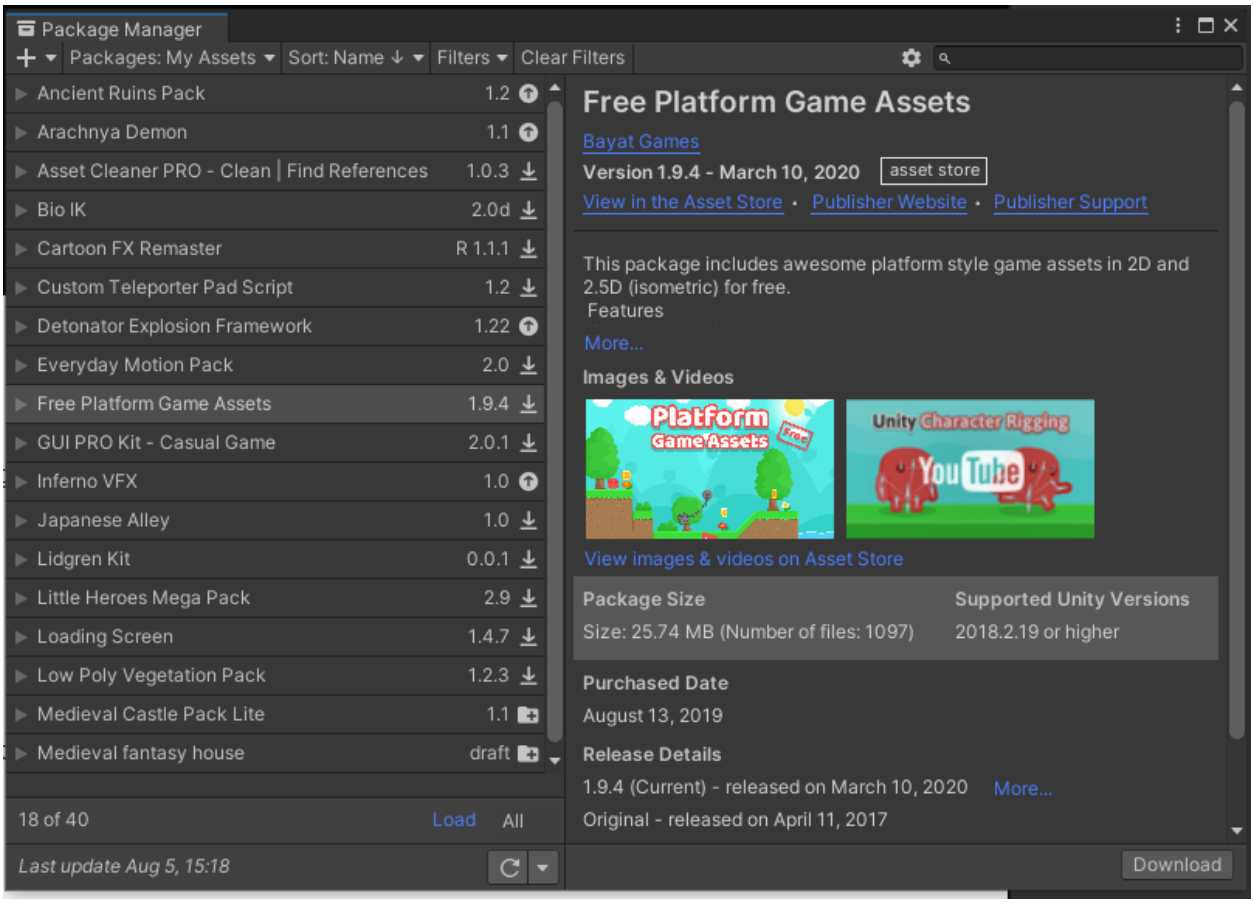
8. Instead of an “Open in Unity” option you will see an “Add to My Assets” button.



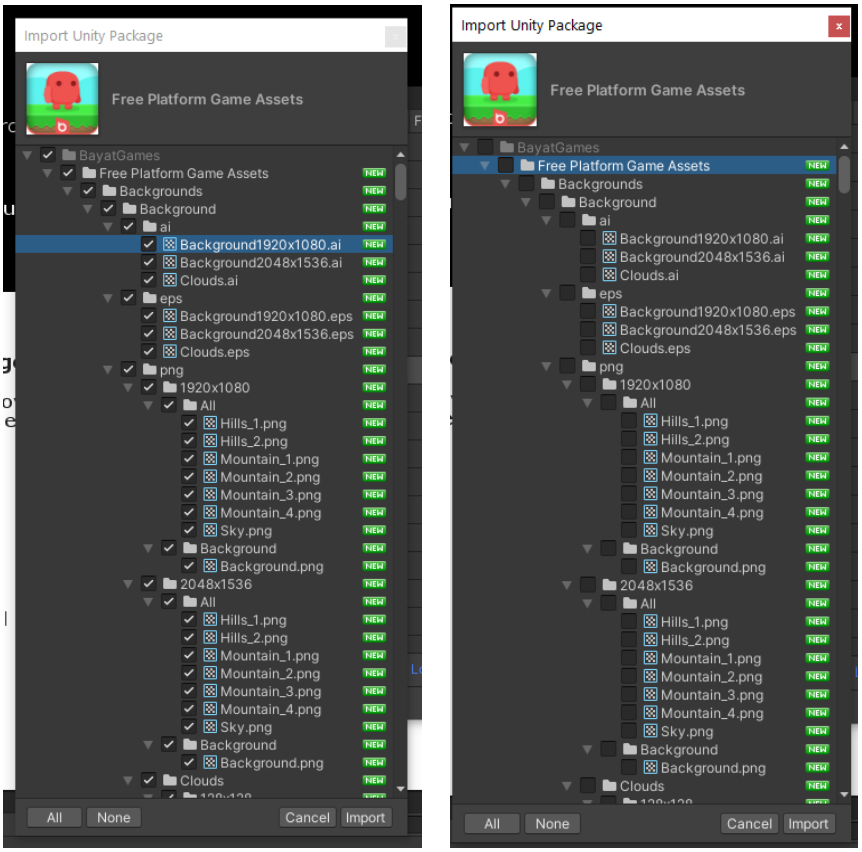
9. Click on that button and accept and terms if you see the question pop up. Then you will see a popup appear at the top of your browser with the following. (Note that the example is using a different asset because I had already claimed the one used in this workshop). “Go to My Assets” will take you to a page with all your assets if you want to view all your have claimed. Or you can immediately click “Open in Unity” to begin importing it.



10. Click the “Open in Unity” button. You may need to select to let your browser open the link with Unity. Once opened you may need to log into Unity as well if you are not already so that it can load all your assets. Once opened successfully you should see that it opens the Package Manager and selects the Free Platform Game Assets package.

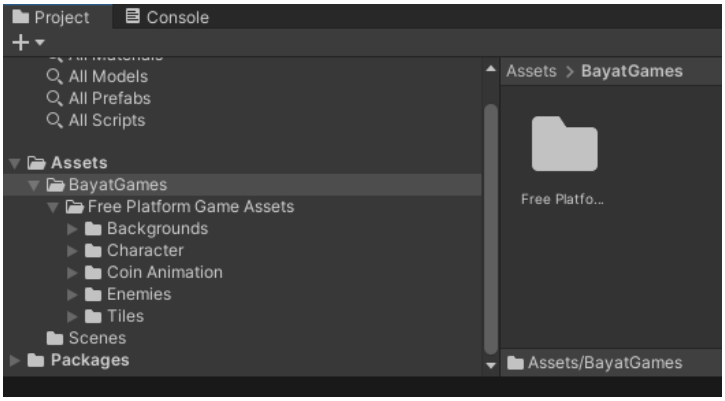


11. Click “Download” in the bottom right. It will download.
12. Then click “Import” that appears in the same place once it has been downloaded. It may take a moment to evaluate the content of the package so let it do its work. Then you will see the below window.



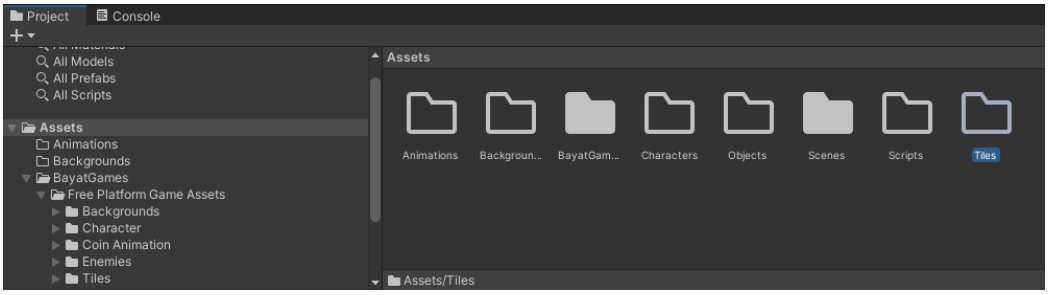
13. We only need a few assets from this collection. Start by unticking the “Free Platform Game Assets” folder. As seen to the right above.
14. Tick the following individual assets:
    - Backgrounds/png/1920x1080/Background/Background.png
    - Character/Character Animation (Update 1.8)/Idle/1x.png
    - Coin Animation/png/1024x128.png
    - Enemies/Enemies/png/Saw.png
    - Tiles/2.5D Tiles/Spring/png/256x256/Dirt.png
    - Tiles/2.5D Tiles/Spring/png/256x256/DirtDown.png
    - Tiles/2.5D Tiles/Spring/png/256x256/DirtLeft.png
    - Tiles/2.5D Tiles/Spring/png/256x256/DirtLeftCorner.png
    - Tiles/2.5D Tiles/Spring/png/256x256/DirtRight.png
    - Tiles/2.5D Tiles/Spring/png/256x256/GrassMid.png

With only these selected click the Import button. You will find it imports everything such that you can find a new folder created with the imported content. Seen over page in the Project tab.



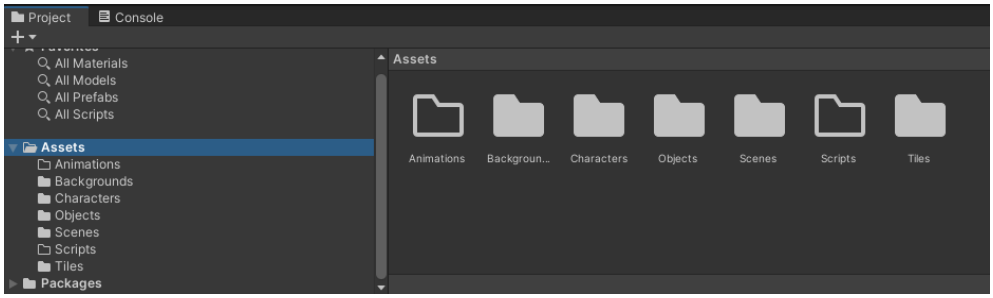
2.5 Folder Setup

15. We are going to move these assets into folders more suitable for managing this project. Create the following folders under Assets.
    - Animations
    - Backgrounds
    - Characters
    - Objects
    - Scripts
    - Tiles



16. Move the files into folders as listed below.
    - Backgrounds
        - Move the Background.png file from the imported Backgrounds folder.
    - Characters
        - Move the Character/Character Animation/Idle/1x.png into Characters folder and rename the file CharacterIdle.png.
    - Objects
        - Move the CoinAnimation/png/1024x128.png to the Objects folder and rename it “Coin.png”.
        - Move the Enemies/Enemies/png/128x128/Saw.png to the Objects folder.
    - Tiles
        - Move the 5 files inside Tiles/2.5D Tiles/Spring/png/256x256/ into the Tiles folder.

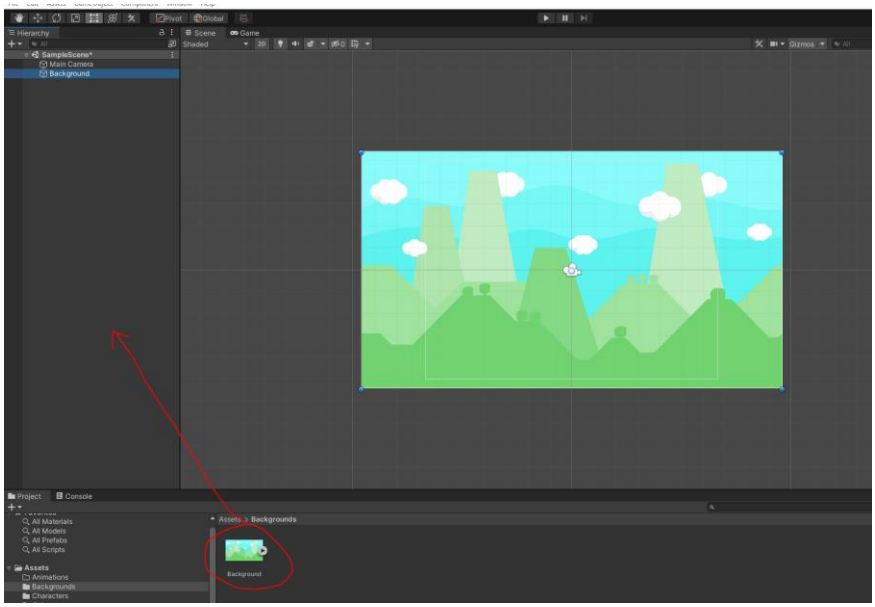
17. You should have moved all the files out of the Free Platform Game Assets now so you can delete the Bayat Games folder. Your folder structure should then appear as below.



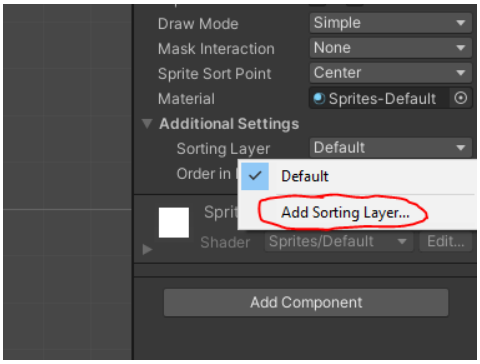
3 Setting up an Environment

3.1 Adding a Background to the Scene

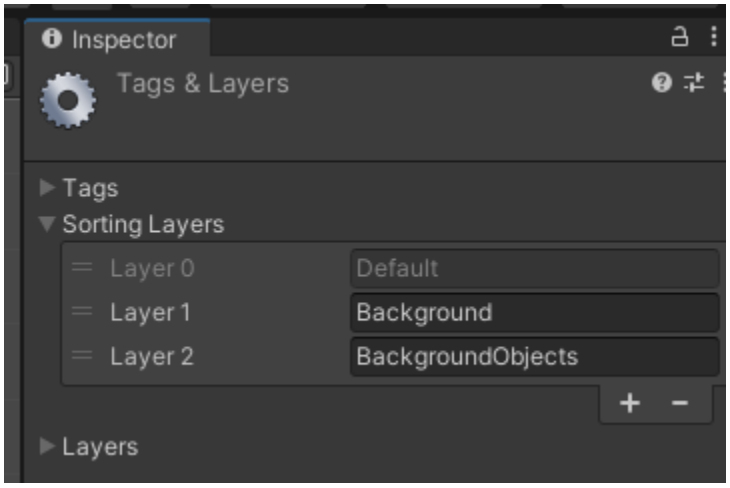
18. Drag the Background into the Scene Hierarchy.



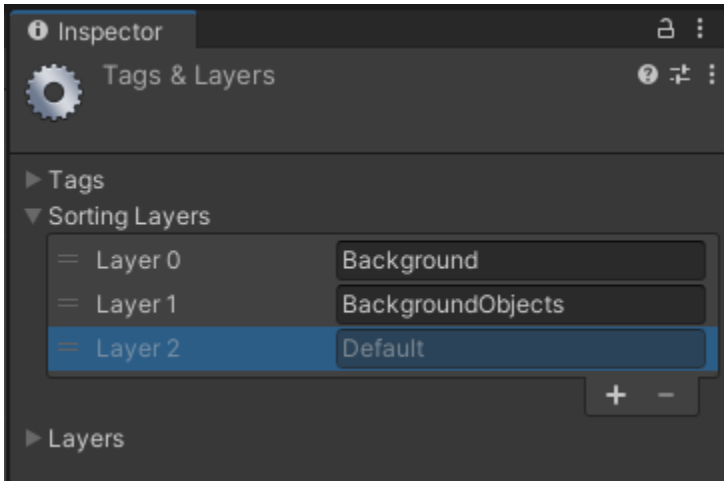
19. Select the Background object. Find the Sorting Layer option and select Add Sorting Layer.



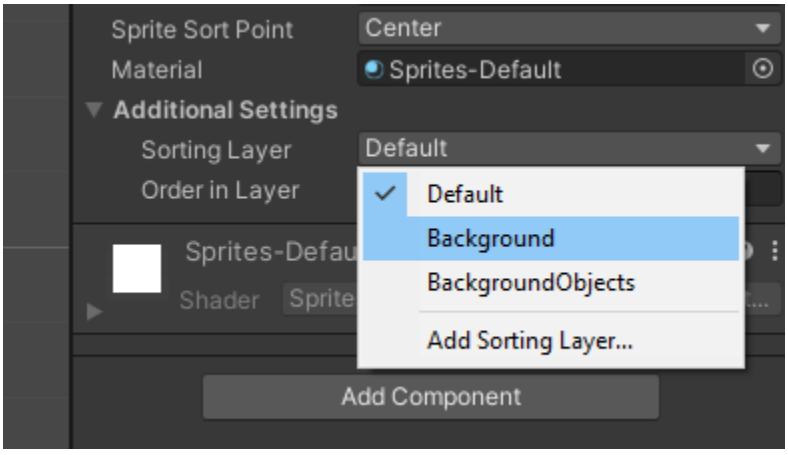
20. Add Two Sorting Layers: One called Background followed by one called BackgroundObjects



21. Drag the “Default” layer to the bottom.

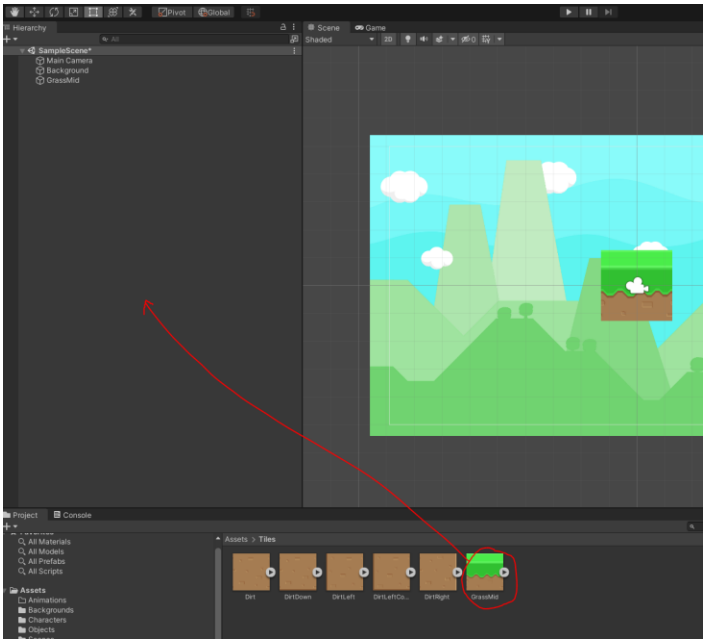


22. Return to the Background and change Sorting Layer to Background

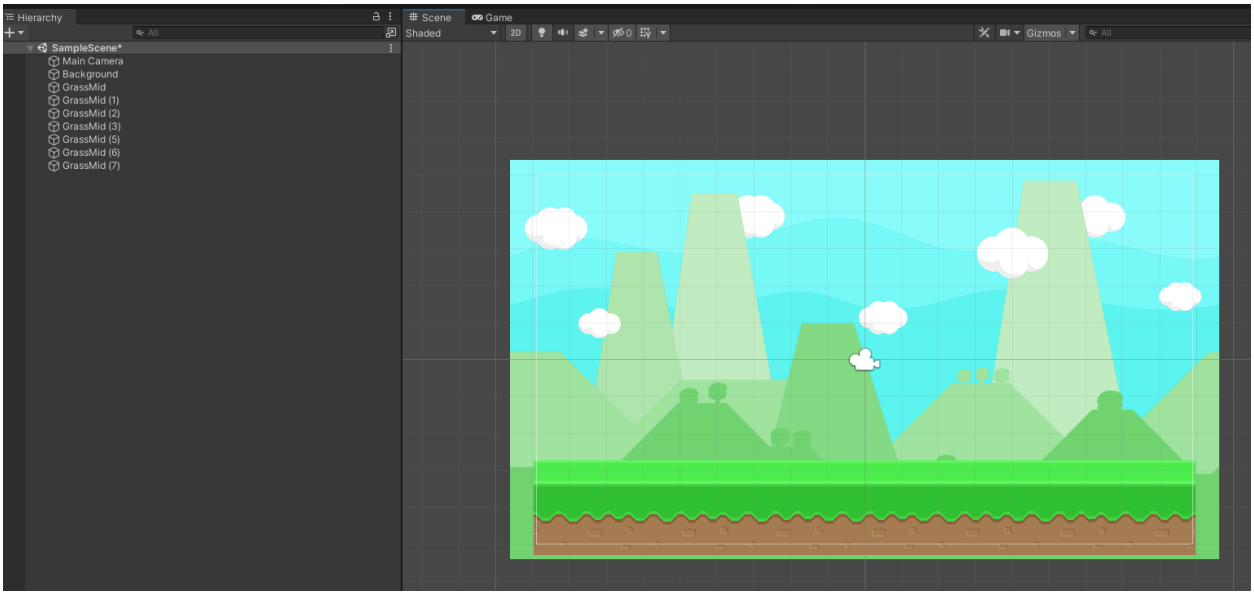


3.2 Add Tiles to the Scene

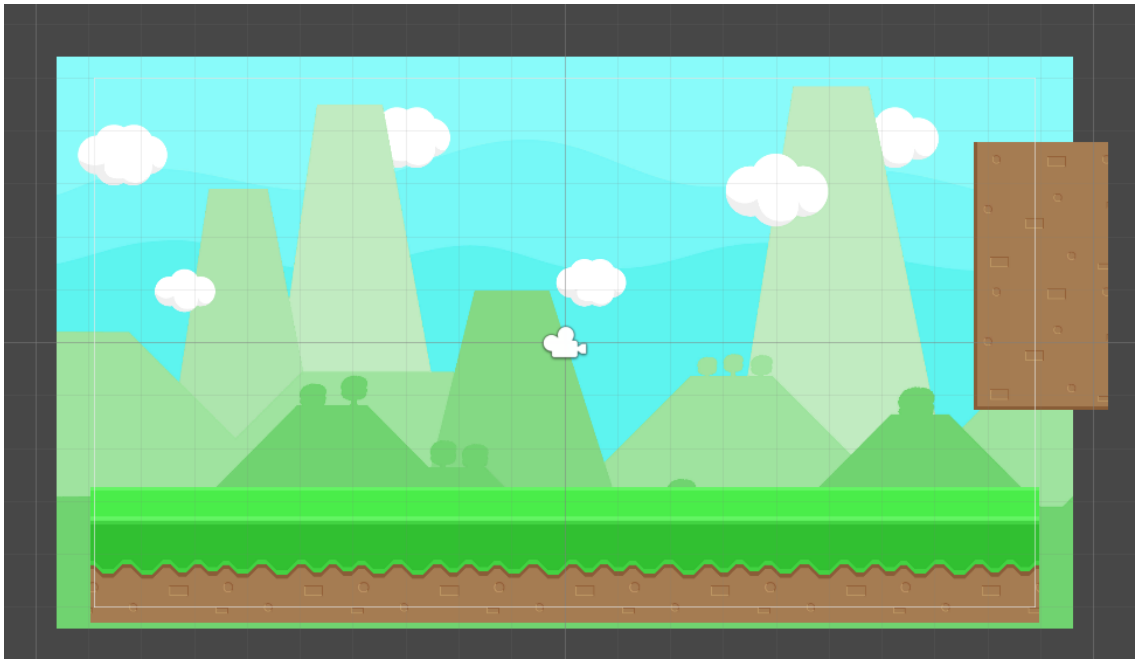
23. Starting by adding some ground. Drag and drop a GrassMid tile into the scene.



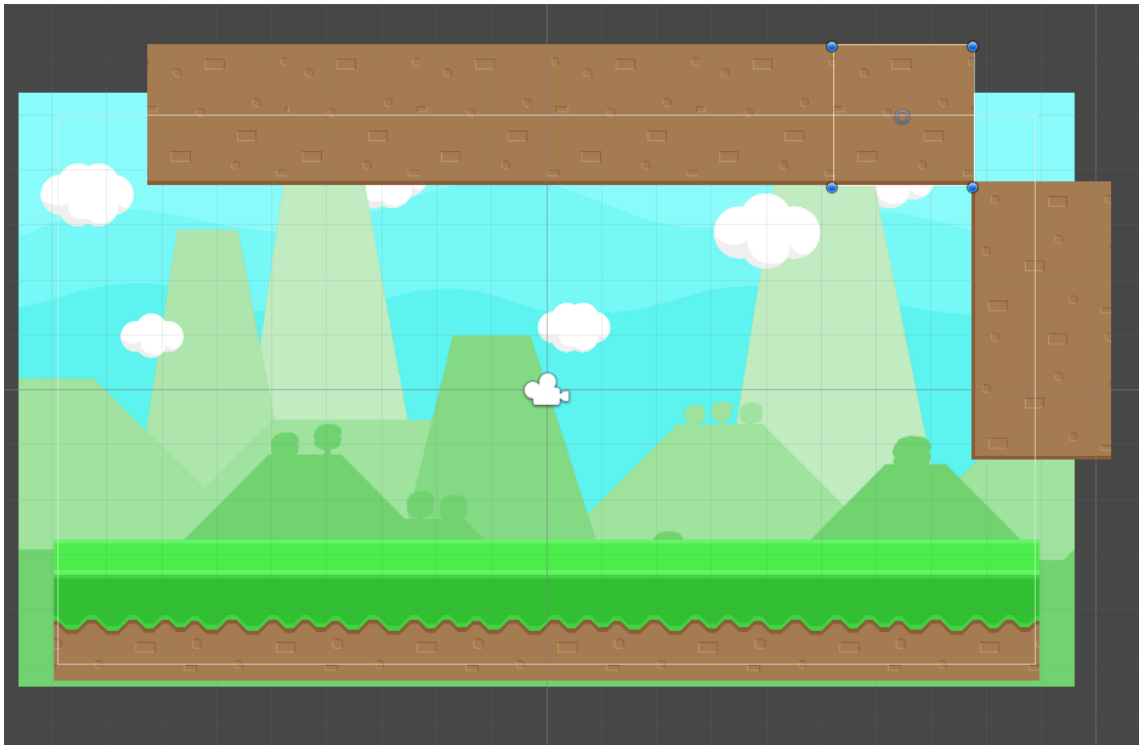
24. Move the tile to the position (0, -4, 0).
25. Add more objects by duplicating the GrassMid object. You can do this with Ctrl+D. Make sure they all have a Y of -4 and move them horizontally to line up with each other. Once you have multiples of them you can duplicate multiples at a time by selecting them and using Ctrl+D again then moving them as a group. You should end up with 8 or 9 objects as shown below.



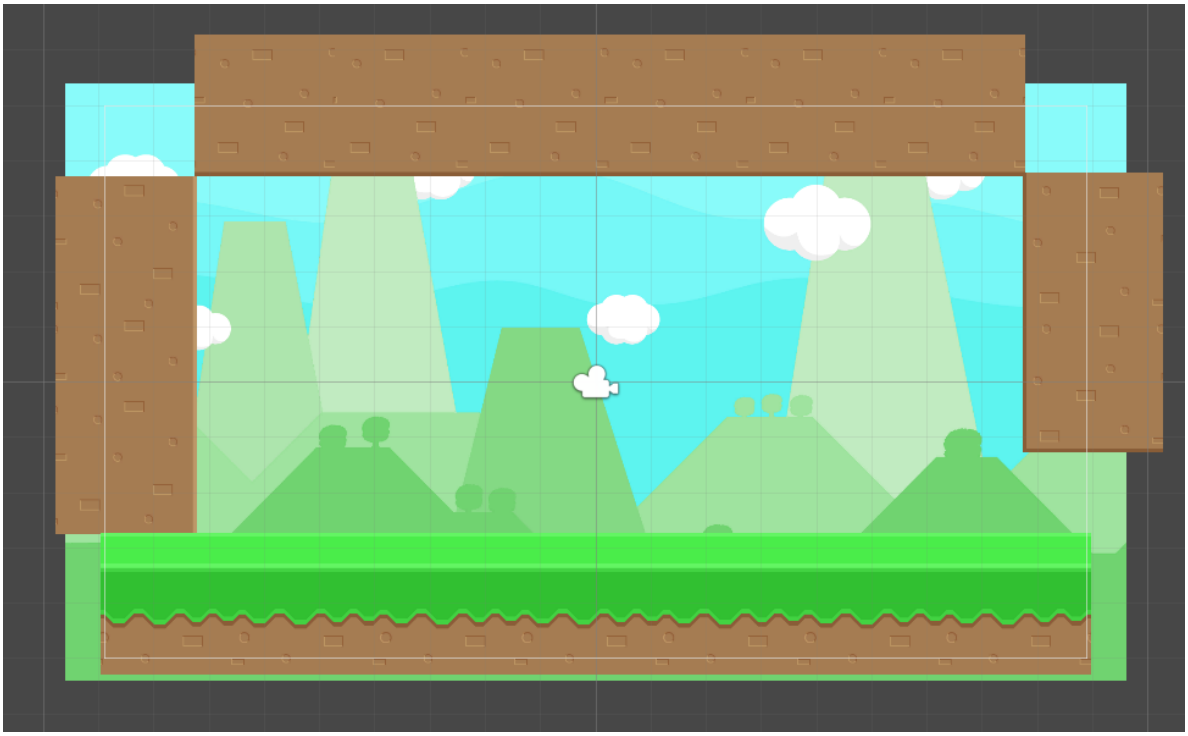
26. Now working on the right wall, drag a DirtLeft into the Hierarchy and set its position to (X = 9, Y =2.5). Then also add a DirtLeftCorner and set its position to (X = 9, Y = 0). It should appear as seen below.



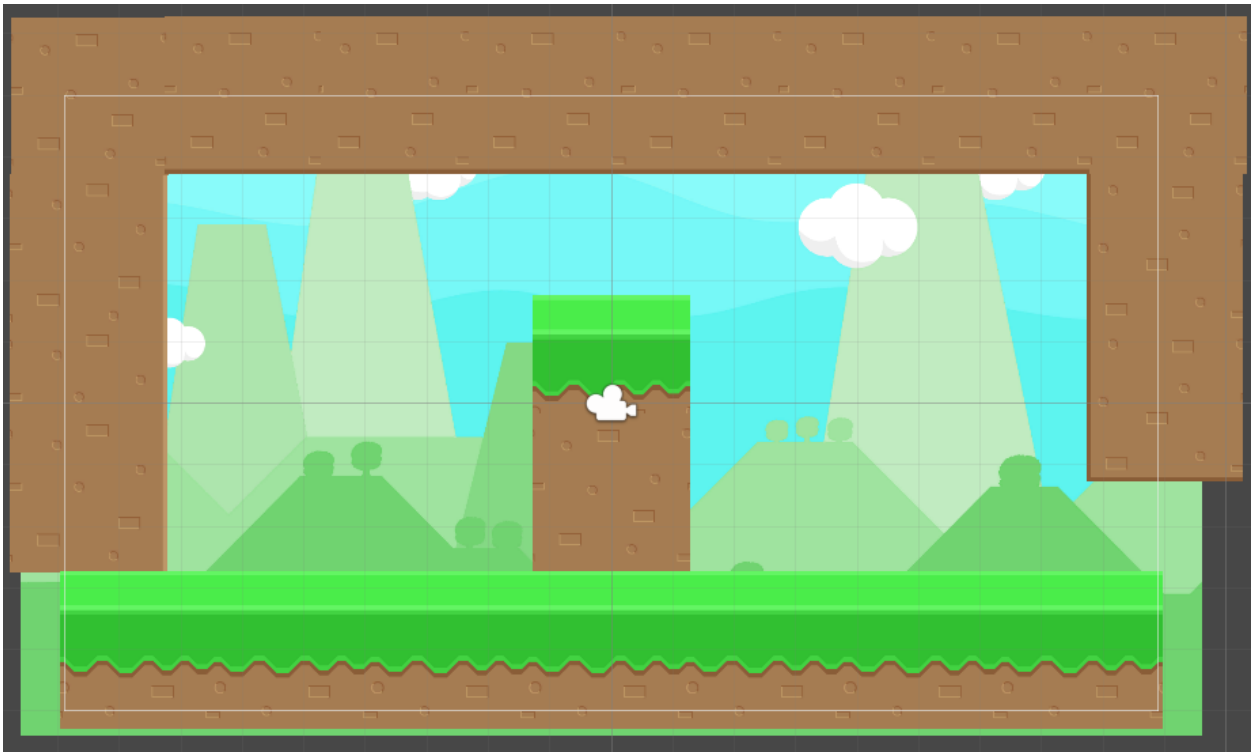
27. Add DirtDown five times to the top. The right most one is set to (X=6.5, Y=5) for reference. It should appear like below.



28. Add three DirtRight that partly overlap to appear as seen below. For reference, the top one is set to be at position X=-8.5, Y=2.44.

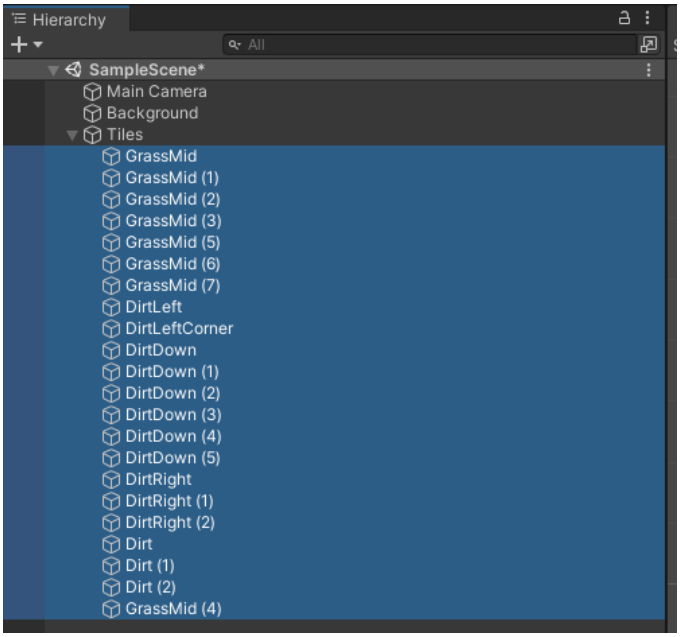


29. Add in three Dirt tiles. One for each of the top corners to fill them in. And one in the middle, with a GrassMid positioned above it as shown below.

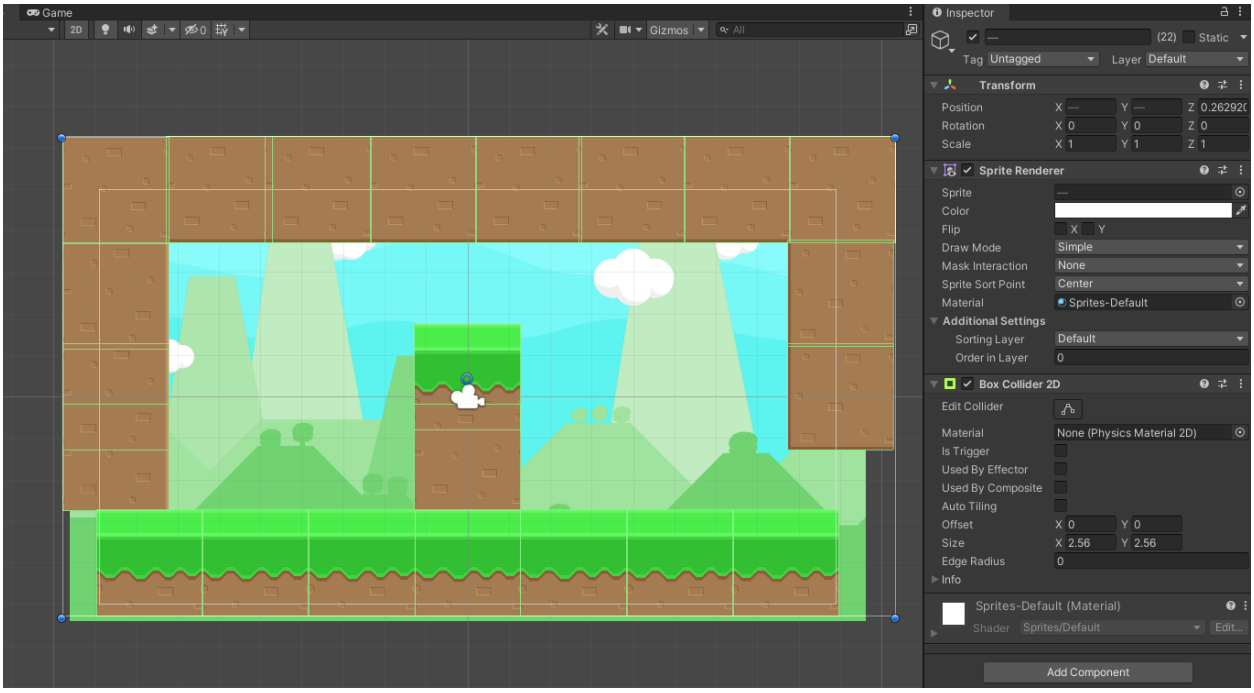


3.3 Organising the Hierarchy

30. Create an Empty Game Object. Name the object Tiles. Select all the Grass/Dirt Blocks and Move them into the Tiles object.



31. Add Colliders to all the dirt/grass objects by selecting all of them then adding the “Box Collider 2D” component. Make sure it says “2D”!!

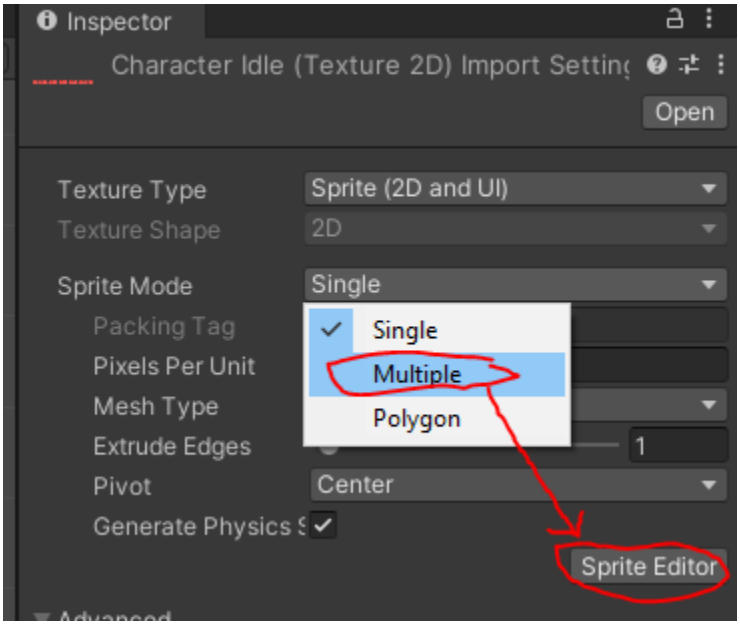


The colliders automatically scale to the sprites. You can see the green outlines showing all the colliders in the scene.

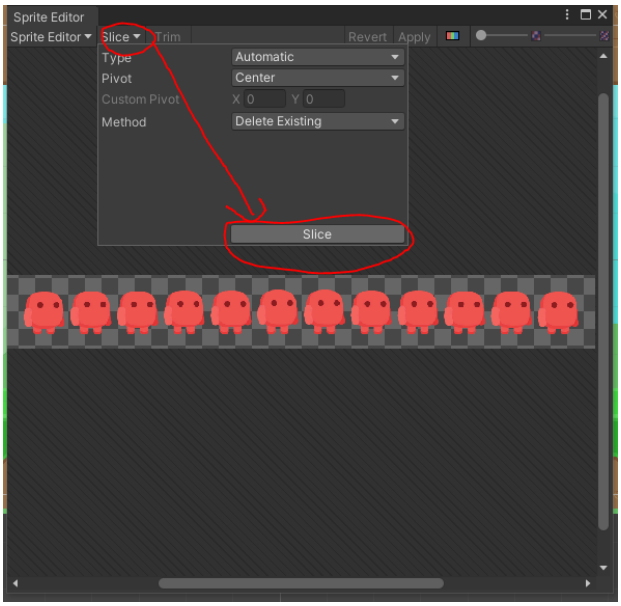
4 Creating a Game Character

4.1 Setting up the Character Sprite

32. Find CharacterIdle in your Characters folder. Change the Sprite mode to Multiple. Then click “Sprite Editor”.



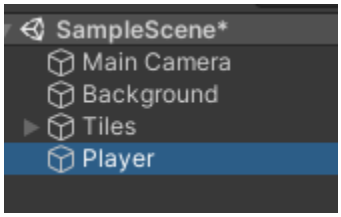
33. Click “Slice”. Then click “Slice” and it will automatically slice the Sprite for you.



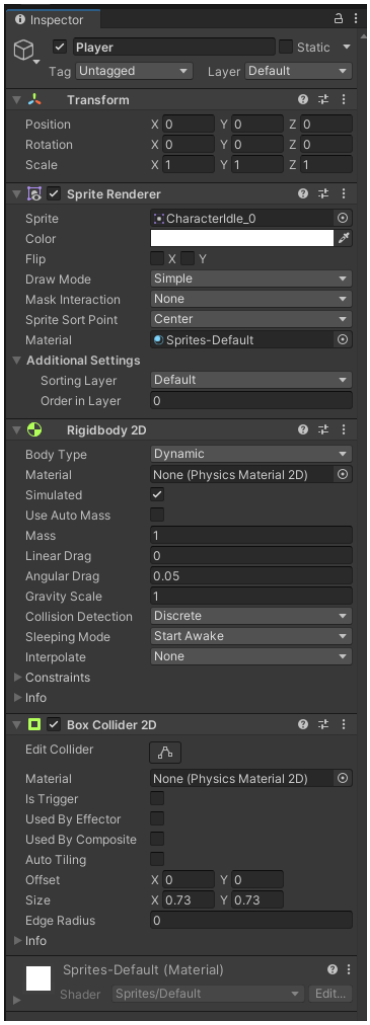
When closing the Sprite Editor click “Apply”. You will now see all the sub-Sprites.

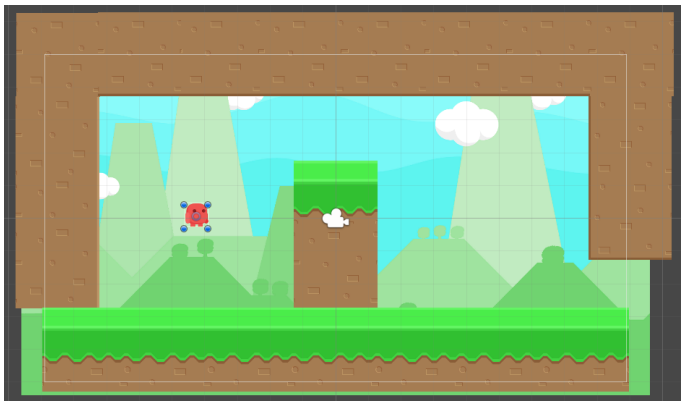
4.2 Adding the Character to the Scene with Collisions

34. Drag CharacterIdle into the scene. It will create an object called CharacterIdle_0 representing the first sprite. Rename the character “Player”.

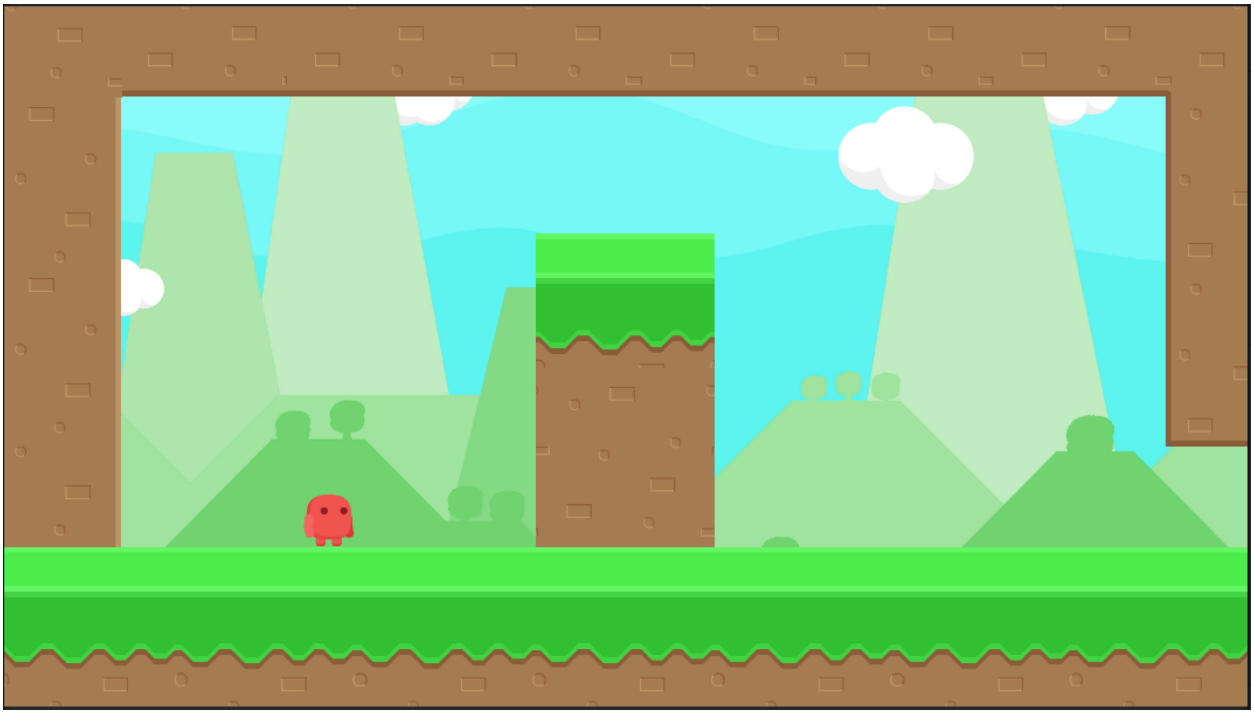


35. Add a RigidBody 2D to the object. Then add a Box Collider 2D to the object. The Player object’s inspector should appear as seen below.





36. Move the character into the left side of the room as seen above.
37. Test that the character falls to the ground when the game runs. The character should stop as shown over the page.



4.3 Character Controller Script

38. Create a C# Script called GroundState. And add the following code to the file.

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

public class GroundState
{
    private GameObject player;
    private float width;
    private float height;
    private float length;

    private Collider2D playerCollider;

    // GroundState constructor.  Sets offsets for raycasting.
    public GroundState(GameObject playerRef)
    {
        player = playerRef;
        playerCollider = player.GetComponent<Collider2D>();
        width = playerCollider.bounds.extents.x + 0.1f;
        height = playerCollider.bounds.extents.y + 0.2f;
        length = 0.05f;
    }

    // Returns whether or not player is touching wall.
    public bool isWall()
    {
        bool left = Physics2D.Raycast(new Vector2(player.transform.position.x - width,
                                                  player.transform.position.y),
                                                  -Vector2.right, length);
```

```
        bool right = Physics2D.Raycast(new Vector2(player.transform.position.x + width,
                                                   player.transform.position.y),
                                                   Vector2.right, length);

        return left || right;
    }

    // Returns whether or not player is touching ground.
    public bool isGround()
    {
        // Directly down
        return Physics2D.Raycast(new Vector2(player.transform.position.x,
                                             player.transform.position.y - height),
                                             -Vector2.up, length);
    }

    // Returns whether or not player is touching wall or ground.
    public bool isTouching()
    {
        return isGround() || isWall();
    }

    //Returns direction of wall.
    public int wallDirection()
    {
        // If there is a wall to the left.
        if (Physics2D.Raycast(new Vector2(player.transform.position.x - width,
                                                   player.transform.position.y),
                                                   -Vector2.right, length))
            return -1;

        // If there is a wall to the right.
        if (Physics2D.Raycast(new Vector2(player.transform.position.x + width,
                                                   player.transform.position.y),
                                                   Vector2.right, length))
            return 1;

        // No wall to either side.
        return 0;
    }
}
```

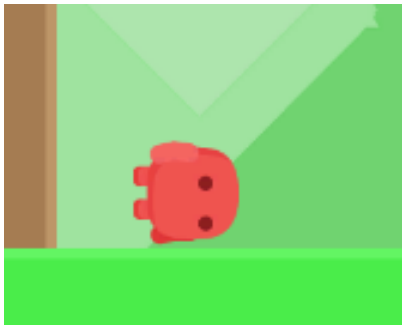
39. Create another C# Script called PlayerController.

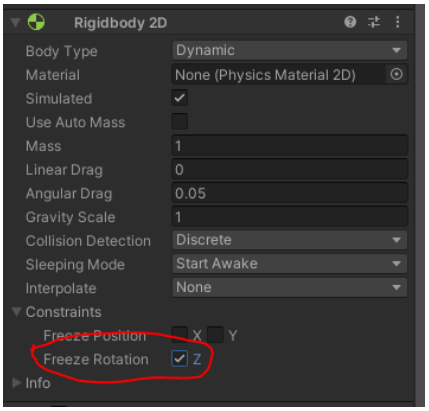
```
using UnityEngine;
using System.Collections;

public class PlayerController : MonoBehaviour
{
    // Character movement speed.
    public float speed = 14f;
    // Horizontal acceleration when not in the air.
    public float accel = 6f;
    // Horizontal acceleration when in the air.
    public float airAccel = 3f;
    // Vertical speed when jumping.
```

```
    public float jump = 14f;

    private GroundState groundState;
    private Rigidbody2D rigidBody;

    void Start()
    {
        //Create an object to check if player is grounded or touching wall
        groundState = new GroundState(transform.gameObject);
        rigidBody = GetComponent<Rigidbody2D>();
    }

    private Vector2 input;

    void Update()
    {
        // Handle input
        if (Input.GetKey(KeyCode.LeftArrow))
            input.x = -1;
        else if (Input.GetKey(KeyCode.RightArrow))
            input.x = 1;
        else
            input.x = 0;

        if (Input.GetKeyDown(KeyCode.Space))
            input.y = 1;
    }

    void FixedUpdate()
    {
        // Move player with force to the left/right
        rigidBody.AddForce(new Vector2(
            ((input.x * speed) - rigidBody.velocity.x)
            * (groundState.isGround() ? accel : airAccel) // different acceleration in
air
            , 0)); // No vertical force

        // Stop player if input.x is 0 (and grounded) and jump if input.y is 1
        rigidBody.velocity = new Vector2(
            (input.x == 0 && groundState.isGround()) ? 0 : rigidBody.velocity.x,
            (input.y == 1 && groundState.isTouching()) ? jump : rigidBody.velocity.y);

        // Add force negative to wall direction (with speed reduction)
        if (groundState.isWall() && !groundState.isGround() && input.y == 1)
            rigidBody.velocity = new Vector2(-groundState.wallDirection() * speed *
0.75f,
                                             rigidBody.velocity.y);

        // Only apply the jumps for single button down presses.
        input.y = 0;
    }
}
```

4.4 Adding the Player Controller

40. Add the PlayerController Script to the Player object and test that it is working
41. You may notice that as you bounce around the character rotates. To prevent this, you will need to select the character and under Rigidbody 2D find “Constraints” and tick “Free Rotation: Z”.

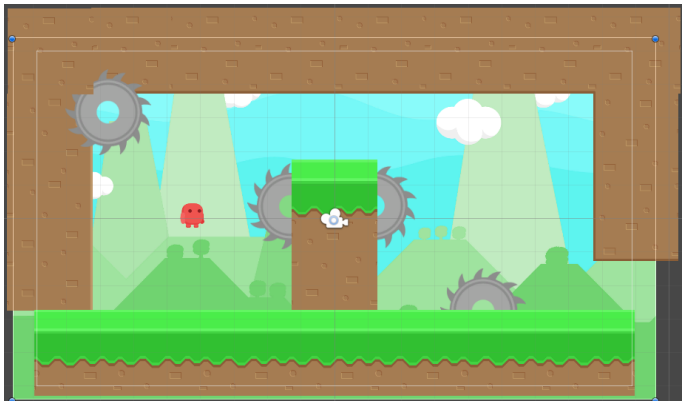




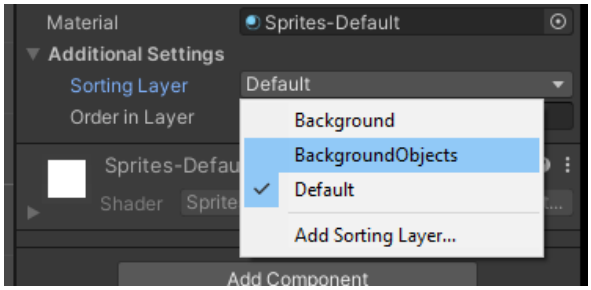
5 Adding Other Objects to the Scene

5.1 Adding a Saw

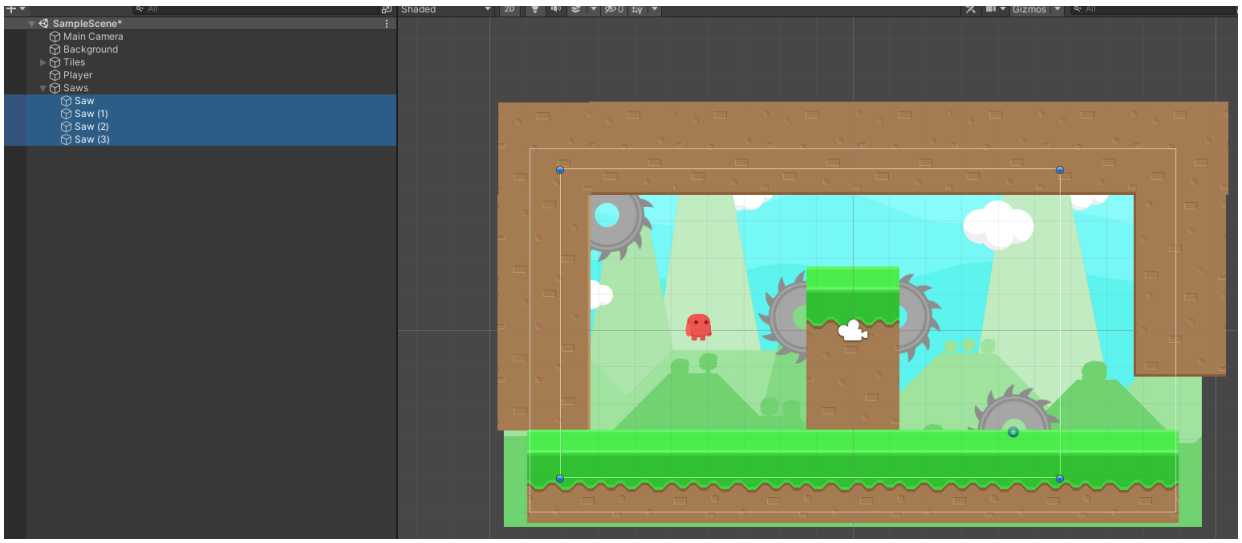
42. Drag four Saw objects as shown. Set their scale to 2 for X,Y, and Z.



43. Select all the Saw objects and change their “Sorting Layer” to “BackgroundObjects” this will force them to appear behind the tiles, but in front of the Background.



44. Create an empty GameObject called “Saws” and move all the Saws into that object to keep them sorted neatly. It should appear as seen below.



5.2 Animating the Saws

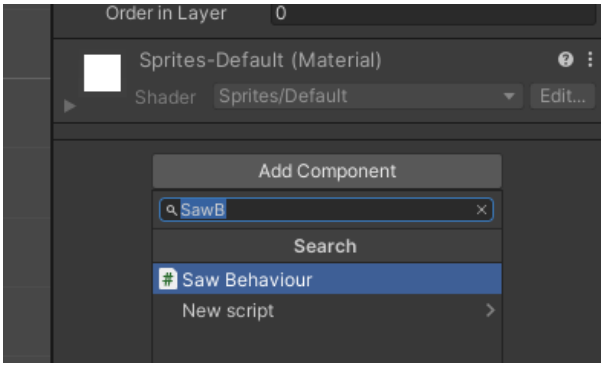
45. Create a C# Script called “SawBehaviour” in the Scripts folder.
46. Declare an instance variable as a public float called speed.
47. In the Update() method write:
    Transform.Rotate(Vector3.forward * speed * Time.deltaTime);

    The script should appear as seen below.

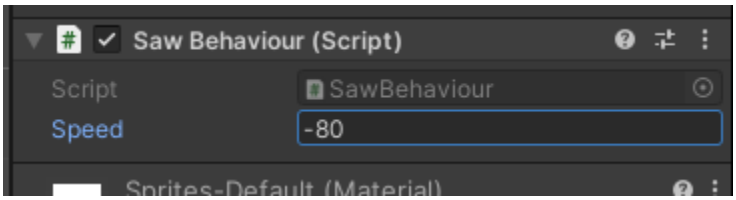
```
Unity Script | 0 references
public class SawBehaviour : MonoBehaviour
{
    // The rotation speed.
    public float speed;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        transform.Rotate(Vector3.forward * speed * Time.deltaTime);
    }
}
```

48. Save the script and return to Unity.

49. Select all the Saw objects and add the SawBehaviour component.



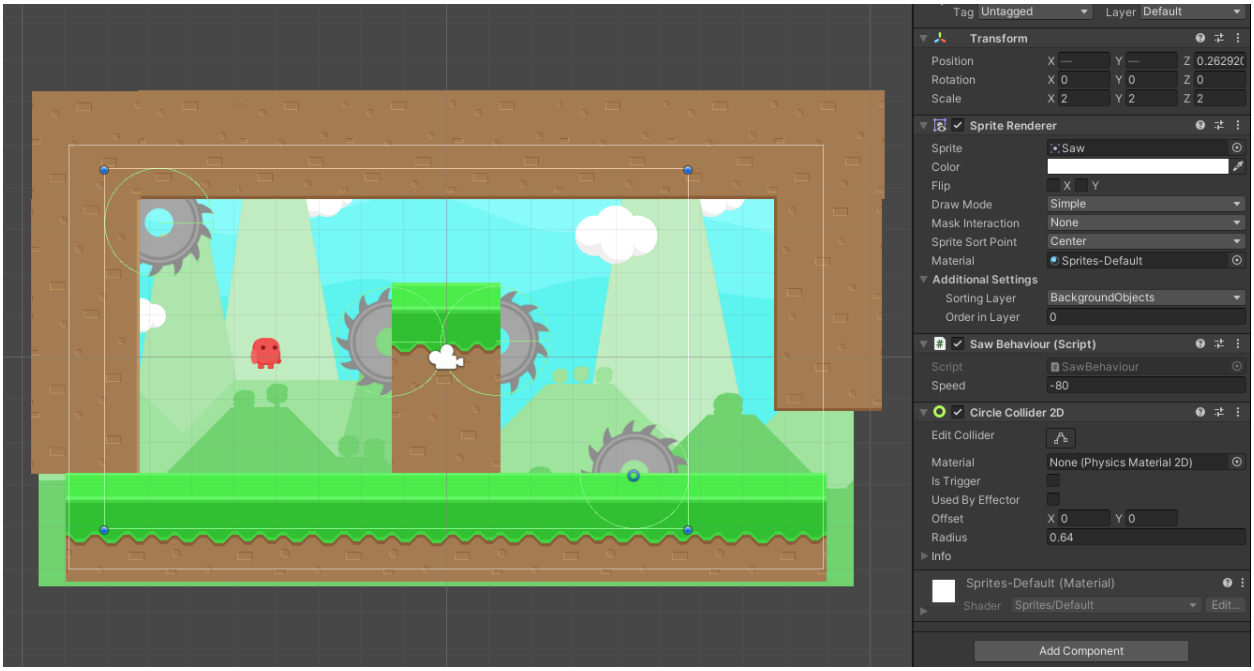
50. Set the speed to -80 in the inspector for the SawBehaviour component.



51. Run the game and test that the saws now rotate clockwise.

5.3 Add a Collider to the Saws

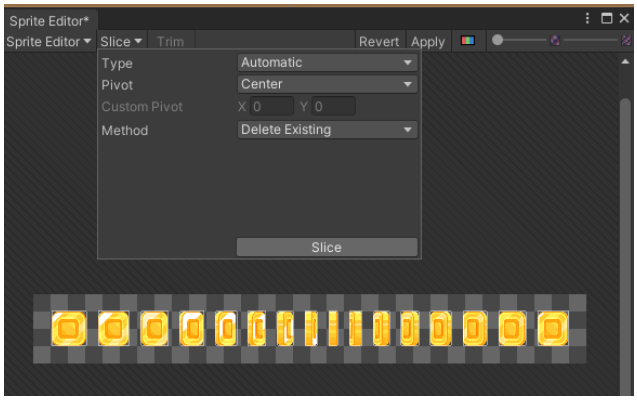
52. Make sure the game is stopped. Then select all the Saw objects and add the Circle Collider 2D. You can see the circle for the collider when they are selected as the green outline.



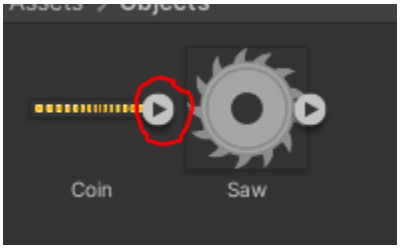
53. Run the game to test they are now rotating.

5.4 Adding Coin Objects with Animation

54. Find the Coin sprite. Select the sprite. Change the Sprite Mode to “Multiple”. Then using the Sprite Editor click Spice and splice the sprite into its individual frames. Then close and apply it.

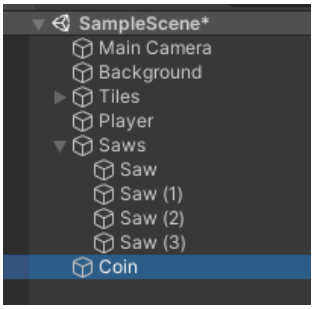


55. Click on the little arrow button on the right of the Coin asset circled below.



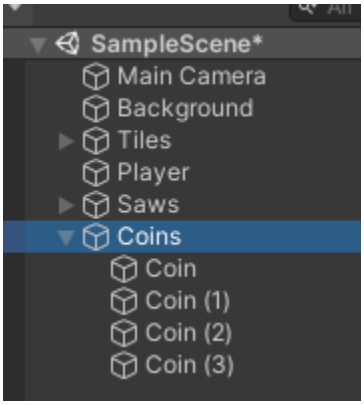
56. Select all the individual sprites that made up the sprite sheet that was split. (Select the first one, then hold shift and click the last one). As shown below.

57. Drag the selection into the Hierarchy. You will be prompted to save it as an Animation. Save it to the Animations folder as “CoinAnimation.anim”.
58. Rename the object “Coin”.

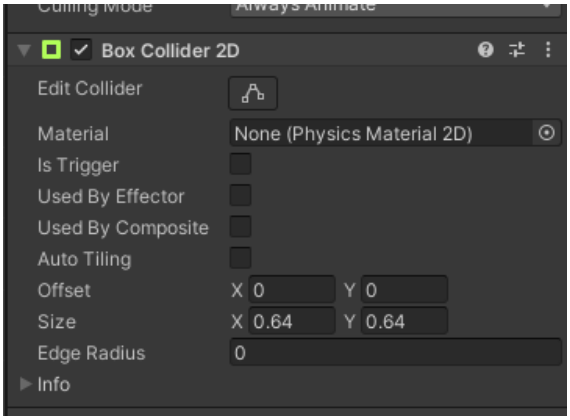


59. Duplicate and place a few coins. You could place them like what is shown below.

60. Run and test that the coins are animating.
61. Select all the “Coin” objects and add them to a new empty GameObject called Coins.

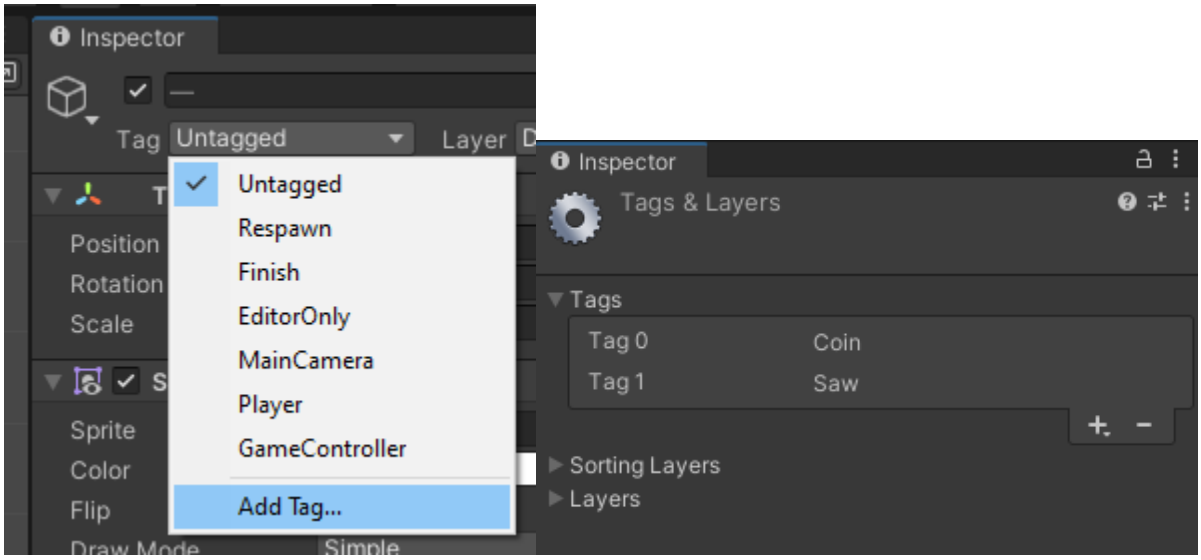


62. With all the coins selected add the Box Collider 2D component to them all.



5.5 Using Tags to Identify Different Objects

63. Select any object and at the top there is a “Tags” field just under the name. Select the dropdown and go to “Add Tag…”. Then add a “Coin” and “Saw” tag to the list. These could be more generic if you had many different objects.



64. Select all the Saw objects and change their Tag to “Saw”.
65. Select all the Coin objects and change their Tag to “Coin”.

6 Handling Interactions using PlayerBehaviour Script

6.1 Collisions with the Coins

66. Create a new C# Script called PlayerBehaviour.
67. Inside the script add a method:

```
void OnCollisionEnter2D(Collision2D collision) {



}
```

This method will trigger any time an object with a collider has collided with another object with a collider.

68. Write an if statement to check if the collision.gameObject.tag is equal to “Coin”. If it is call Destroy and pass collision.gameObject as the argument. This will destroy the object that was collided with.

    I will note that you could have a script on the coins separately that does that so that the coins handle their own destruction.

The code should look like the below.

```
Unity Script | 0 references
public class PlayerBehaviour : MonoBehaviour
{
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }

    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.tag == "Coin")
        {
            Destroy(collision.gameObject);
        }
    }
}
```

69. Save the script. And return to Unity.
70. Add the PlayerBehaviour Script as a component onto the Player object.

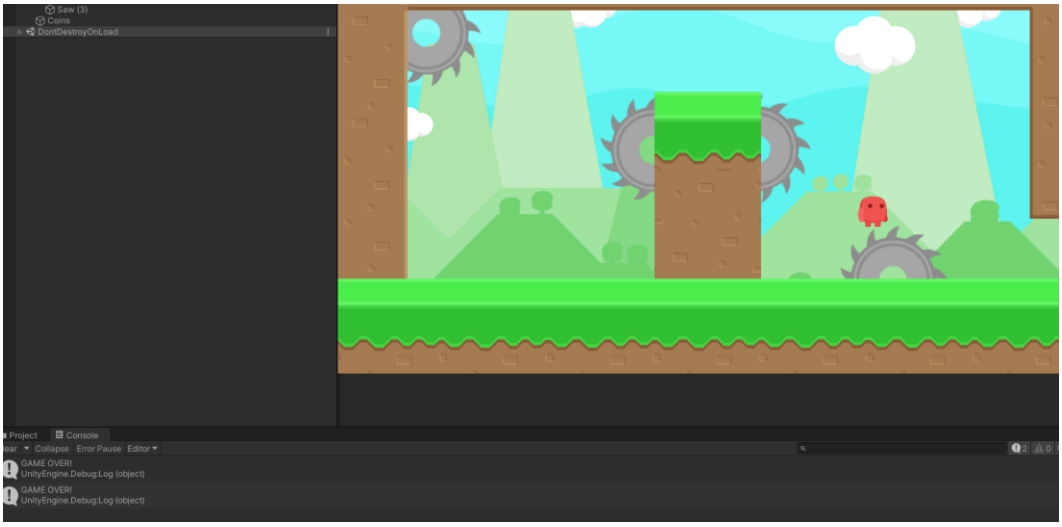
71. Run the program and test that when you jump into the coins they disappear. Do note that because they are acting as solid objects, they will halt momentum. If you wanted the velocity to continue you would need to look at using the Trigger option instead.

6.2 Collisions with the Saws

72. Stop running the program and open the PlayerBehaviour Script.
73. In the OnCollisionEnter2D method add an else if to check if the collision.gameObject.tag is equal to “Saw”. Then if that is true write a Debug.Log method call that outputs “GAME OVER”. As seen below.

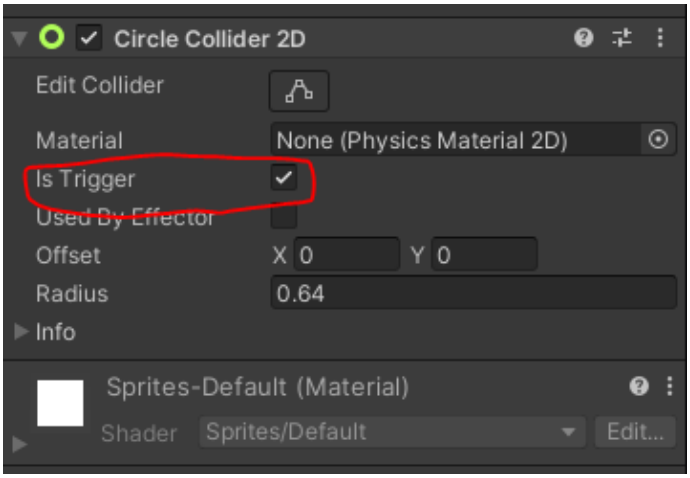
```
Unity Script | 0 references
public class PlayerBehaviour : MonoBehaviour
{
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }

    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.tag == "Coin")
        {
            Destroy(collision.gameObject);
        }
        else if (collision.gameObject.tag == "Saw")
        {
            Debug.Log("GAME OVER!");
        }
    }
}
```

74. Save and return to Unity then test if the Saws trigger the Console message when collided with.

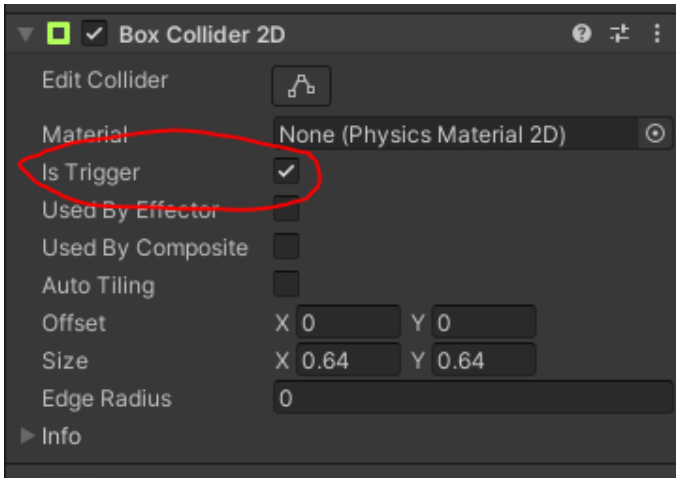


7 Changing the Objects to use Trigger Collisions

75. Select all the Saw objects and tick the “Is Trigger” option in the Circle Collider 2D component.



76. Select all the Coin objects and tick the “Is Trigger” option in the Box Collider 2D component.



77. Open the PlayerBehaviour Script.
78. Change the method name for OnCollisionEnter2D to OnTriggerEnter2D and the variable type to Collider2D.
79. After the Debug.Log method call add the following line.
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);

    This will reload the current scene. This means any time a Saw is collided with it should now restart the game. You will likely need to add a statement to the top to resolve using the SceneManager by adding “using UnityEngine.SceneManagement;”. The code should appear as seen over page.

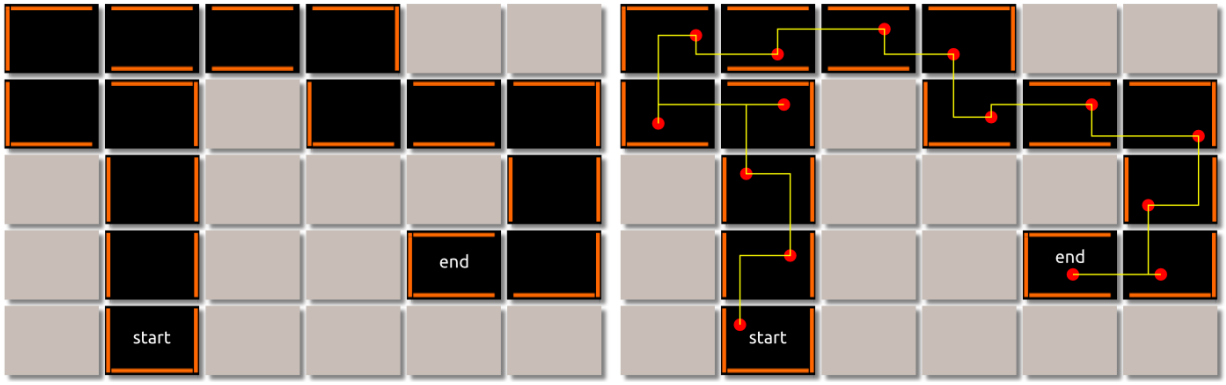
```
Unity Script | 0 references
public class PlayerBehaviour : MonoBehaviour
{
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "Coin")
        {
            Destroy(collision.gameObject);
        }
        else if (collision.gameObject.tag == "Saw")
        {
            Debug.Log("GAME OVER!");
            SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        }
    }
}
```

80. Compile and run to test that collecting coins still works, and that when you collide with the saw blades, they restart the level. You should now also pass through the objects without losing momentum because when Is Trigger is active on the collider, they do not prevent other objects moving into them.

8 Extra Content: Procedural Level

8.1 Algorithm Steps

1. Populate Rooms (2D grid seen in left image below).
2. Create Base Map (Room/Level Boundaries).
3. Create Solution Path (Seen in right image below).
4. Create Extra Platforms (not implemented).
5. Spawn Platforms.
6. Spawn Objects.
7. Determine Player Start Location.



8.2 Controls

- G = Generate New Random Map
- P = Generate alternate random platform sequence (requires G action first!!)
- Left/Right = Apply force left/right
- Space = Jump

8.3 Prefabs Folder Files

The following describes creation of the prefabs, so you understand their construction.

- Coin Prefab
  - Drag "Coin" from the workshop scene into a folder called Prefabs.
  - Rename file CoinPrefab.
- Saw Prefab
  - Drag "Saw" from the workshop scene into a folder called Saw.
  - Rename file SawPrefab.
- PlatformPrefab
  - Drag GrassMid into the scene.

- Add a BoxCollider2D.
- Platform Behaviour.

8.4 Additional Scripts

The following lists the additional scripts added with a brief description of their purpose. View the individual files for expanded detail.

- CameraBehaviour.cs: An unused script that can be used to restrict camera movement within a set of bounds.
- FollowCameraBehaviour.cs: Follows the player with the camera.
- InfiniteBackgroundBehaviour.cs: Tracks a pair of background objects and moves them horizontally to keep the player always seeing a background object at that Y coordinate region. The background does NOT move up, the background colour of the camera is set to match the sky colour to account for this.
- PlatformBehaviour.cs: Allows easy application of state data for generated objects in the procedural system. This is attached to all the tiles that are spawned.
- PlatformGenerator.cs: Generates and manages a collection of Rooms that the PlatformManager can take and add to the scene.
- PlatformManager.cs: Uses the PlatformGenerator to spawn all the procedural objects.
- Singleton: Used for referencing of objects directly.

8.5 Scene Objects

The following describes configuration of the scene objects.

- Main Camera
  - Transform: 8.5, 12.3, -10
  - Camera->Background: 137, 251, 250 (Select colour picker and click on background lightest blue).
  - Add component Follow Camera Behaviour
    - Drag Main Camera into Main Camera
    - Drag Player into Player Transform
- Create Empty Object Background Manager
  - Transform: 0, 10, -1.5
  - Add component Infinite Background Behaviour
    - Once created Drag Background 1 and Background 2 into corresponding variables on this.
    - Drag Main Camera into Cam.
  - Drag in Background into BackgroundManager.
    - Set Transform: 0, 0, 1.5
    - Scale 2, 2, 2
    - Rename Background 1

    - Duplicate Background 1
        - Rename Background 2
        - Set Transform 38, 0, 1.5
- Player
    - Disable Player
- Create Empty Object PlatformSeeder
    - Transform: 0,0,0
    - Create three Empty child objects called:
        - Platforms
        - Objects
        - Enemies
    - Add component Platform Manager
        - Set Platform Textures to size 3.
        - Drag GrassMid into Element 1
        - Drag Dirt into Element 2
        - Drag PlatformPrefab into Platform Prefab
        - Set Width to 15
        - Set Height to 20
        - Set Tile Width to 2.56
        - Set Tile Height to 2.56
        - Set Player by dragging the Player into it.
        - Drag the CoinPrefab and SawPrefab into ObjectPrefabs (make sure that CoinPrefab is 0 and SawPrefab is 1).
    - Add component Platform Generator
        - Set Room Width to 12.
        - Set Room Height to 8.
        - Set Top Layer ID to 1.
        - Set Lower Layer ID to 2.
        - Set Min Room X to -1.
        - Set Max Room X to 4.
        - Set Min Room Y to 0.
        - Set Max Room Y to 4.