JUMP INTO 2D PLATFORM GAME CREATION

POLITEKNIK MALAYSIA
METRO KUALA LUMPUR

N.H. Samad, S.K Deraman, A. H. Mohamad

JUMP INTO 2D PLATFORM GAME CREATION

JABATAN TEKNOLOGI MAKLUMAT DAN KOMUNIKASI

POLITEKNIK METrO KUALA LUMPUR
NO. 2 – 14, JALAN SETIAWANGSA 10,
TAMAN SETIAWANGSA
54200 KUALA LUMPUR

TEL : 03-4251 8000 FAX: 03-4251 7979

Published by :

POLITEKNIK METrO KUALA LUMPUR
No. 2 – 14, Jalan Setiawangsa 10, Taman Setiawangsa
54200 Kuala Lumpur

Jump into 2D Platform Game Creation

First Published 2024
@ Politeknik METrO Kuala Lumpur

e ISBN 978-967-0074-20-7

Perpustakaan Negara Malaysia

Edition 1 : 2024 : Jump into 2D Platform Game Creation

Cataloguing-in-Publication Data

Perpustakaan Negara Malaysia

A catalogue record for this book is available from the National Library of Malaysia

eISBN 978-967-0074-20-7

Preface

This eBook introduces you to a series of lab sheet to develop a 2D game (Platform game). It mainly focuses on the steps by steps guidance on how to develop a 2D platform game using Unity software.

JUMP INTO 2D PLATFROM GAME CREATION introduces the techniques to develop computer games by using game development tool. The readers will be able to understand the concepts of sprites and object, game interactivity, level design and scripting in the development of game of different genres.

Since this eBook provides an outlook of the overall guidance to develop 2D platform game, after going through this material, you will find yourself at a moderate level. It will heaps your knowledge from basics to the next levels

Acknowledgement

The highest gratitude to Allah SWT because with His permission, this Introduction to Jump into 2D Platform Game Creation was successfully published. This eBook is published as a guide or reference for students who take the 2D Game Development course at Malaysia Polytechnic. In preparing this eBook, various challenges and obstacles need to be faced before being able to produce an eBook. We would like to express our deepest gratitude to our family, the Polytechnic e-Learning Coordinator, and colleagues for their guidance and support in the production of this eBook.

We would also like to thank the following for permission to reproduce copyright photos:

- Unity
- Canva
- freepik.com

We hope that this ebook can be put to good use by all who use it.
Thank you.

N.H. Samad, N.H. Samad, S.K Deraman, A. H. Mohamad
Nov 2024

Table of Contents

Table of Contents

Topic 1.0 : Introduction to 2D Game Assets

A) Installation 2D Game Development software

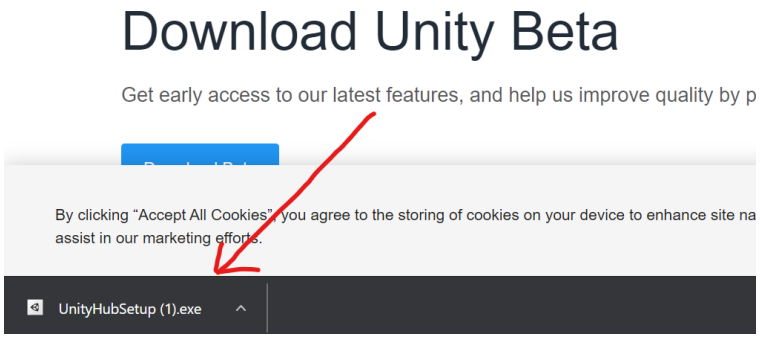
1. Installing Unity Hub

a. The Unity Hub is a management tool that you can use to manage all of your Unity Projects and installations, To install the Unity Hub for Windows, macOS, and Linux visit :

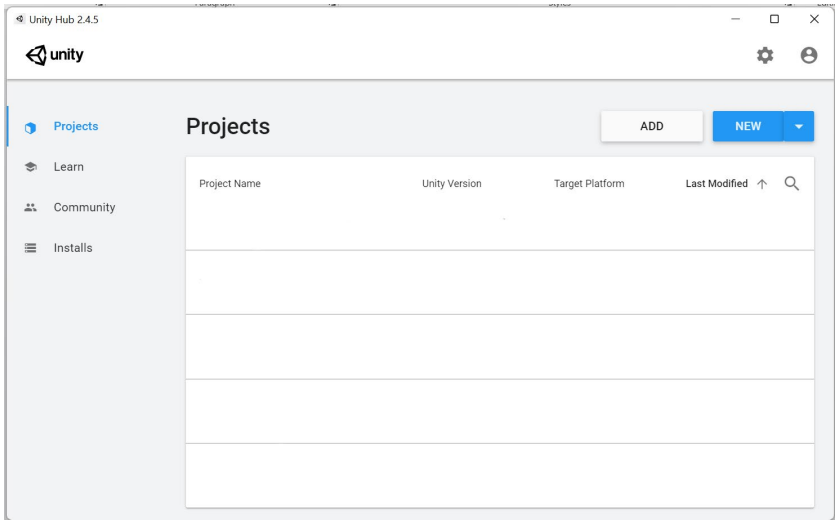
https://unity3d.com/get-unity/download

b. Choose UNITY HUB to begin download installation file

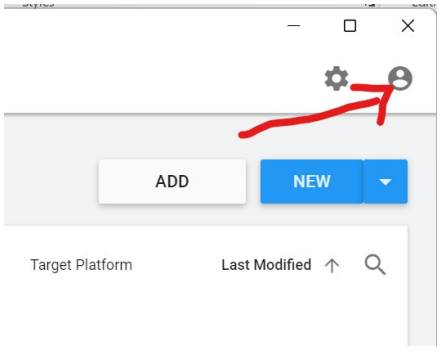
c. Download file and select the file to begin installation



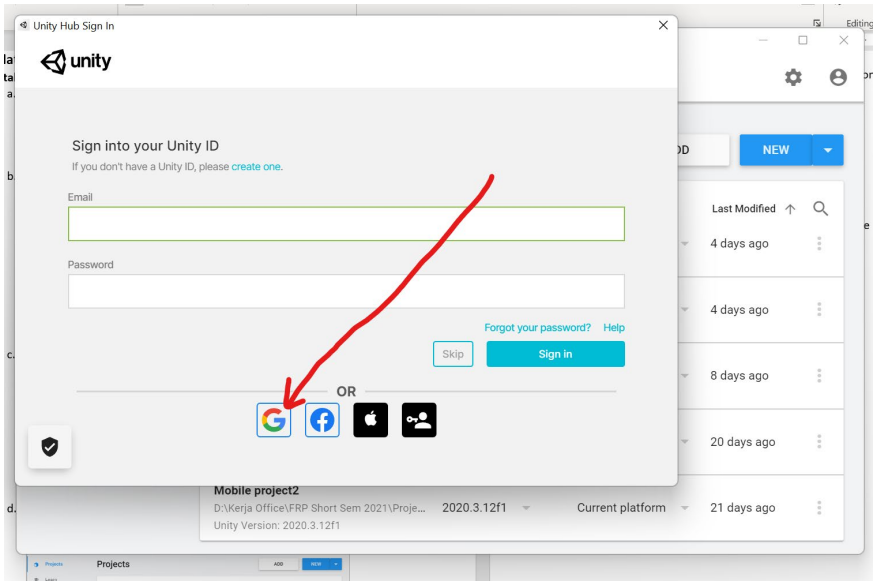
d. Complete installation and run UNITYHUB



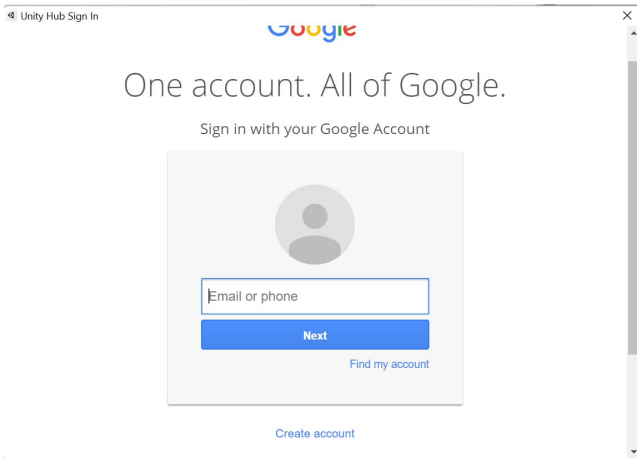
e. Create new UNITY Account by click on icon below :



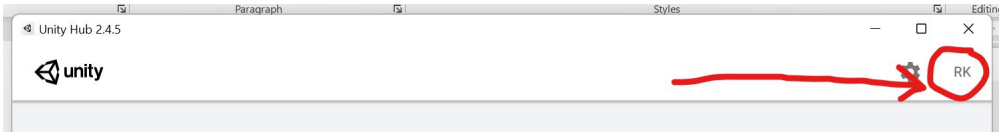
f. Login using your GMAIL account to create new user account



g. Complete sing-in step to complete UNITY account registration

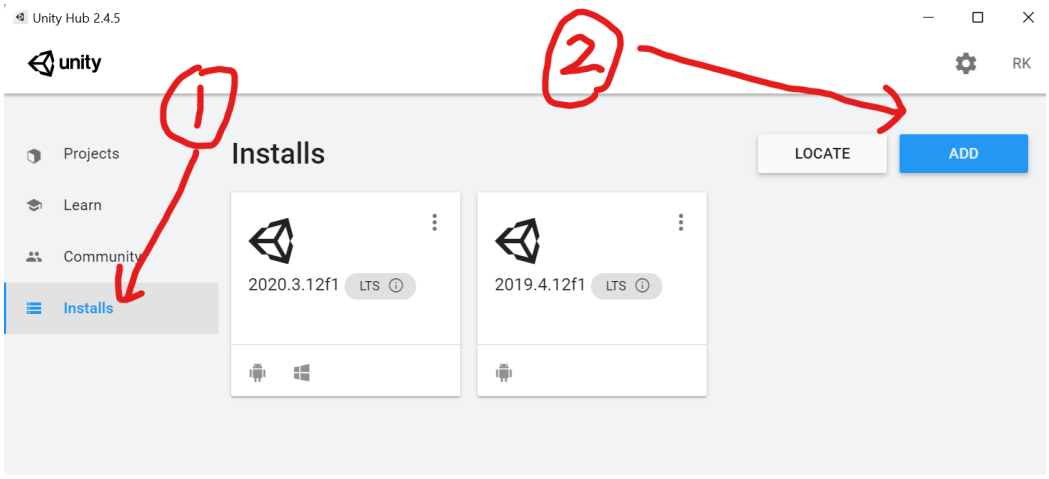


h. Your Avatar at top right connect will show your name:

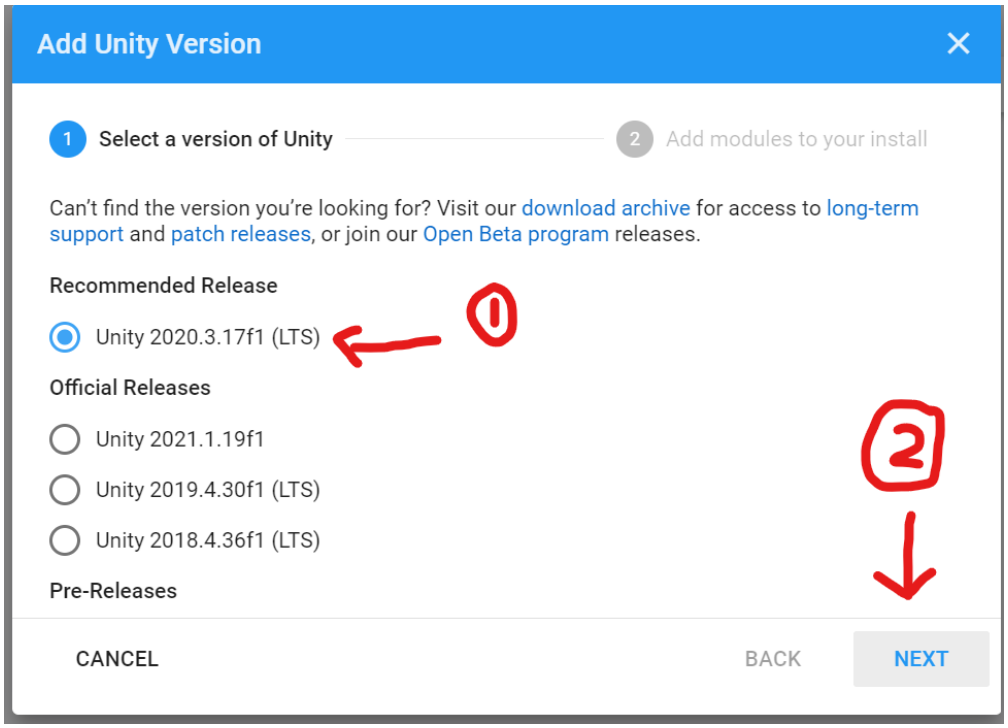


2. Installing UNITY ENGINE

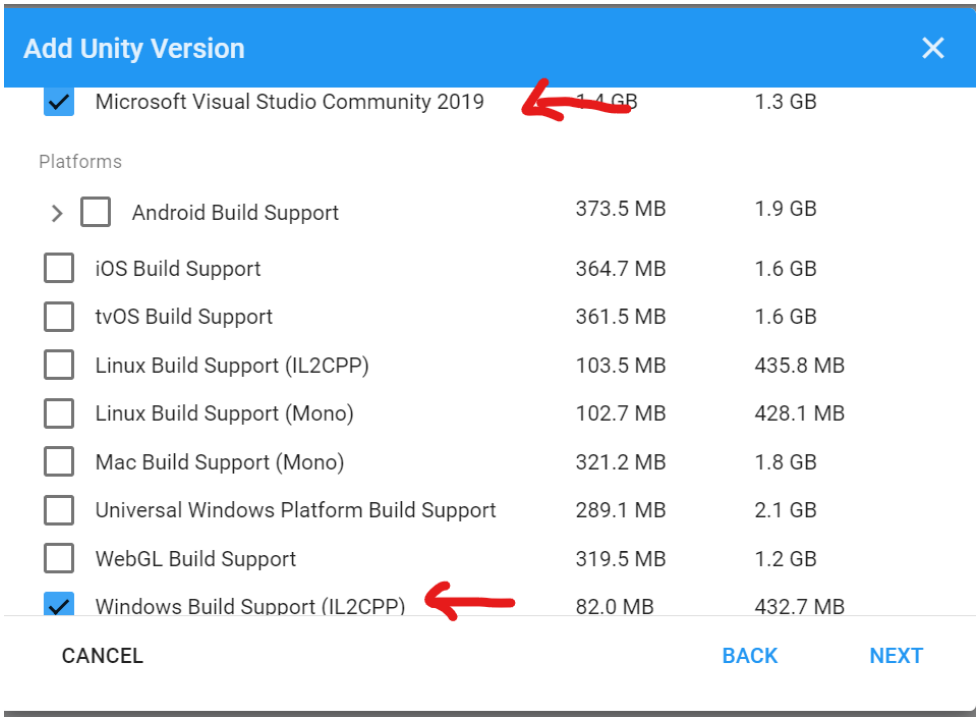
a. Run UNITY HUB and select INSTALL; then Select ADD button to add installation for new **UNITY ENGINE VERSION**



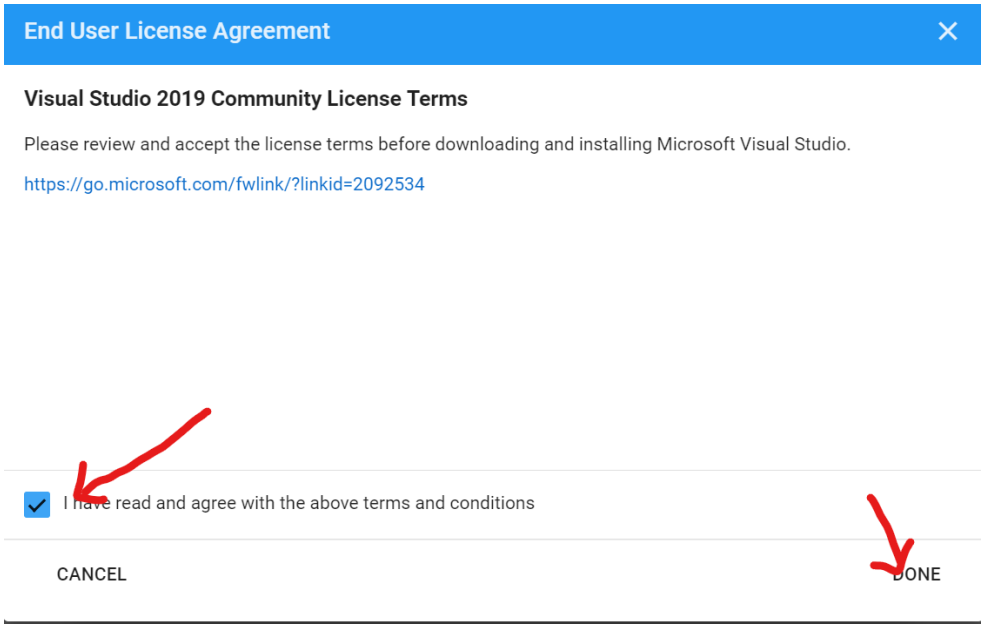
b. Select 2020.3.17f1(LTS) and Select NEXT button



c. Select Microsoft Visual Studio Community 2019 and Windows Build Support; then select NEXT button to proceed installation process.

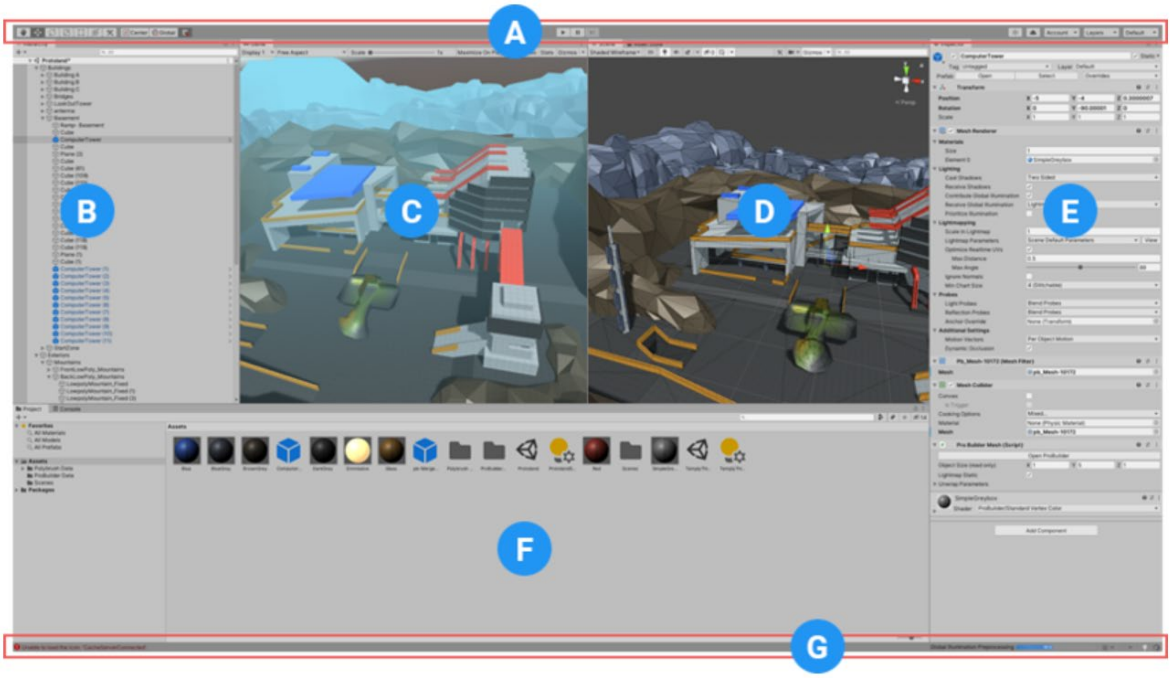


d. Agree condition and select DONE button



e. Wait until installation complete

B) GAME ENGINE'S INTERFACE



(A) The Toolbar : provides access to the most essential working features. On the left it contains the basic tools for manipulating the Scene view and the GameObjects within it. In the centre are the play, pause and step controls. The buttons to the right give you access to Unity Collaborate, Unity Cloud Services and your Unity Account, followed by a layer visibility menu, and finally the Editor layout menu (which provides some alternate layouts for the Editor windows, and allows you to save your own custom layouts).

(B) The Hierarchy window : is a hierarchical text representation of every GameObject in the Scene. Each item in the Scene has an entry in the hierarchy, so the two windows are inherently linked. The hierarchy reveals the structure of how GameObjects attach to each another.

(C) The Game view : simulates what your final rendered game will look like through your Scene Cameras . When you click the Play button, the simulation begins.

(D) The Scene view : allows you to visually navigate and edit your Scene. The Scene view can show a 3D or 2D perspective, depending on the type of Project you are working on.

(E) The Inspector Window : allows you to view and edit all the properties of the currently selected GameObject. Because different types of GameObjects have different sets of properties, the layout and contents of the Inspector window change each time you select a different GameObject.

(F) The Project window : displays your library of Assets that are available to use in your Project. When you import Assets into your Project, they appear here.

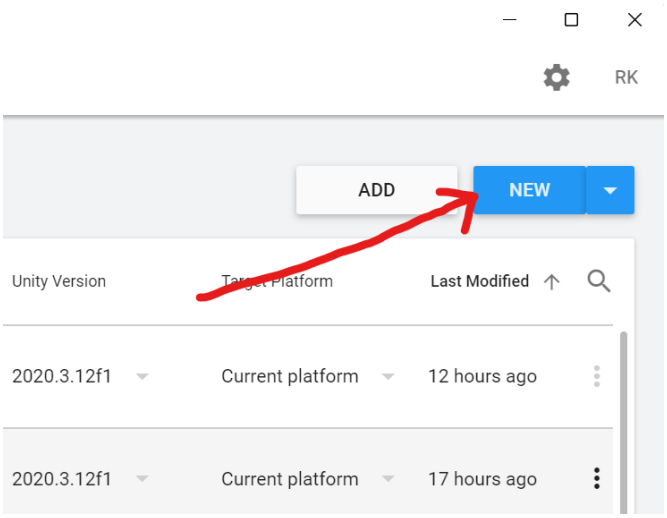
(G) The status bar : provides notifications about various Unity processes, and quick access to related tools and settings.

C) RESOURCES IN GAME DEVELOPMENT

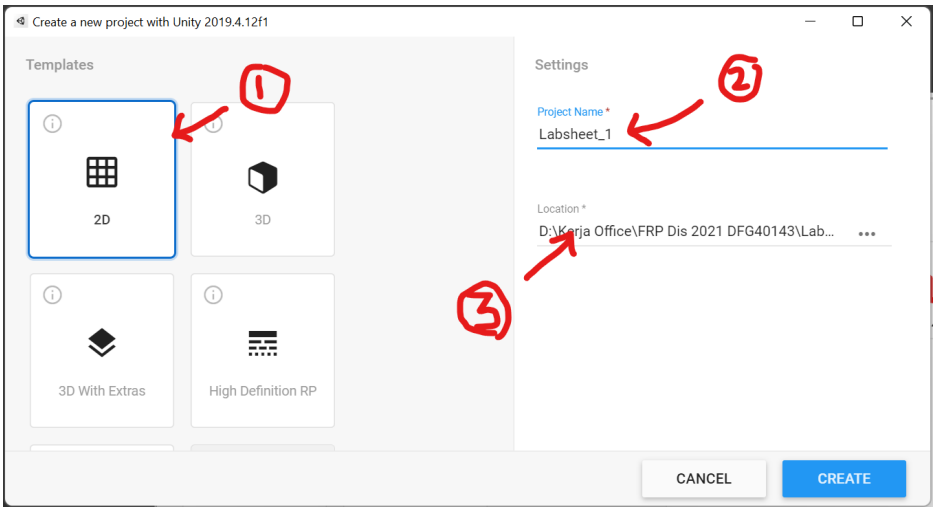
1. Creating New Unity Project File

a. Login to your CIDOS portal account, and download **"labsheet1_resources.unitypackage"** from download section.

b. Open UNITY HUB to create new project; select NEW button at the top right corner of UNITY HUB window



c. Next, choose **2D** from template, then name or project as "labsheet1" and locate your project file into New Folder called "Labsheet".

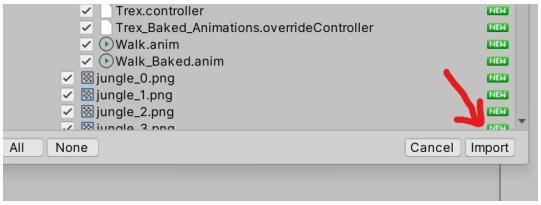


d. Select CREATE button to create the project

2. Importing unitypackage file into project

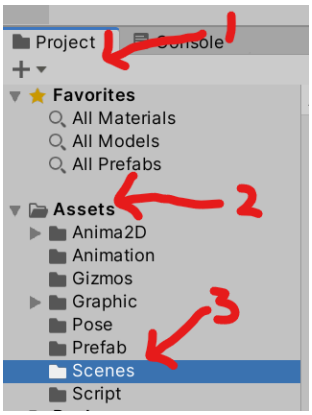
a. Select and double click **(2X)** at "labsheet1_resources.unitypackage" to perform installation new unitypackage file.

b. Select IMPORT button to begin installation

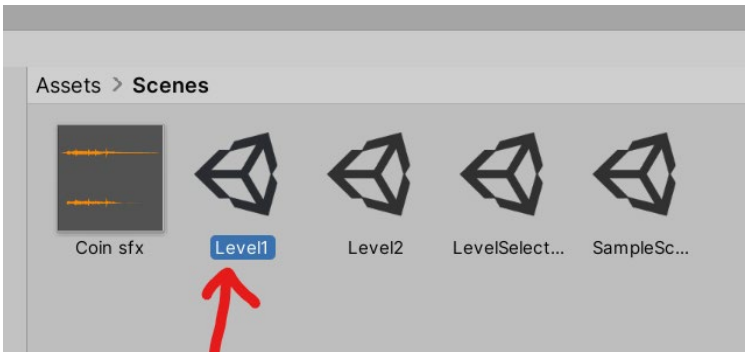


3. Opening Game Scene

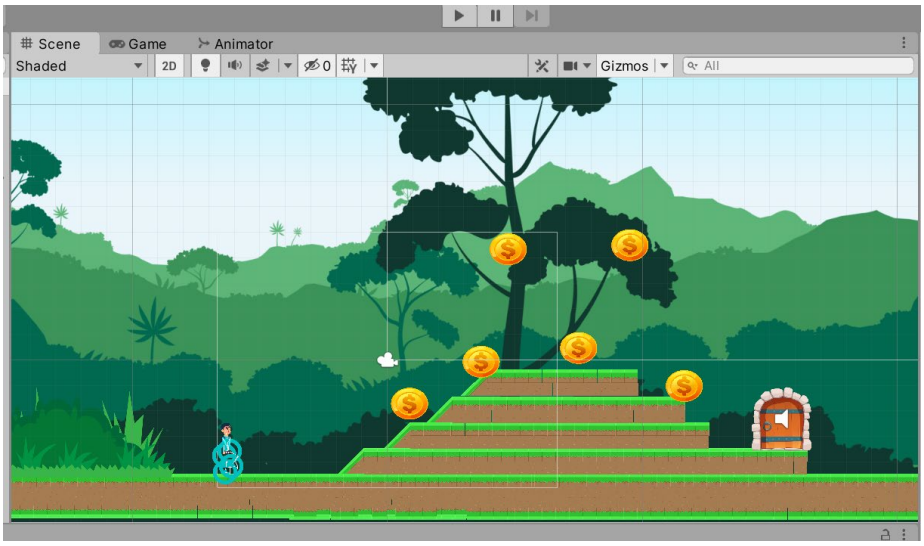
a. Select **“Scene”** folder in **Project Panel** inside **“Assets”** folder



b. Select and Double Click **(2X)** on scene file called **“level 1”** inside **Scene folder**



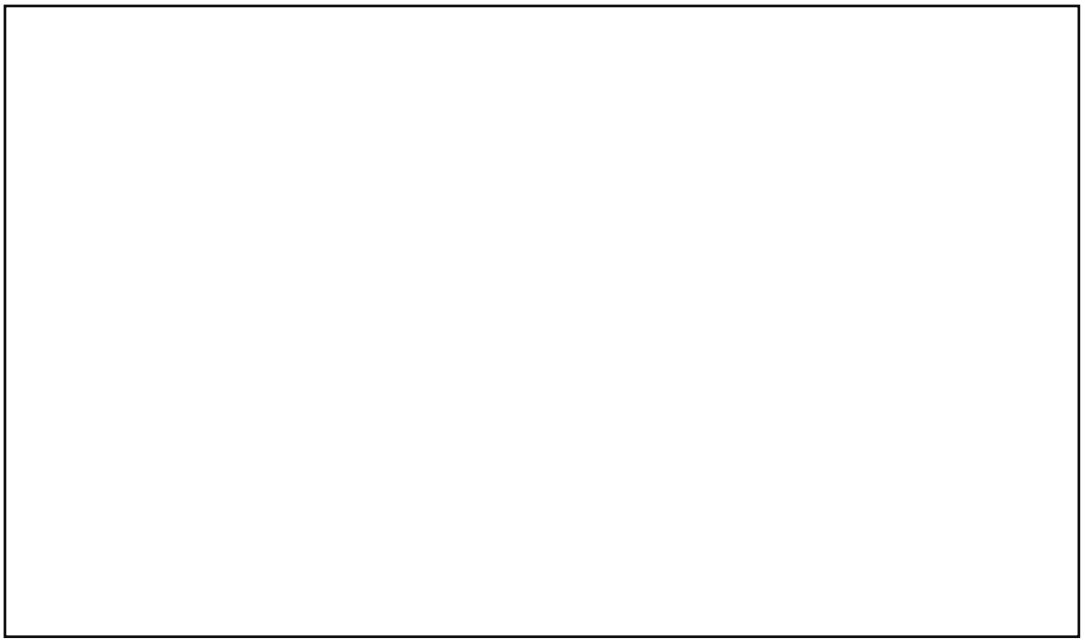
c. Your Scene Panel will show as below :



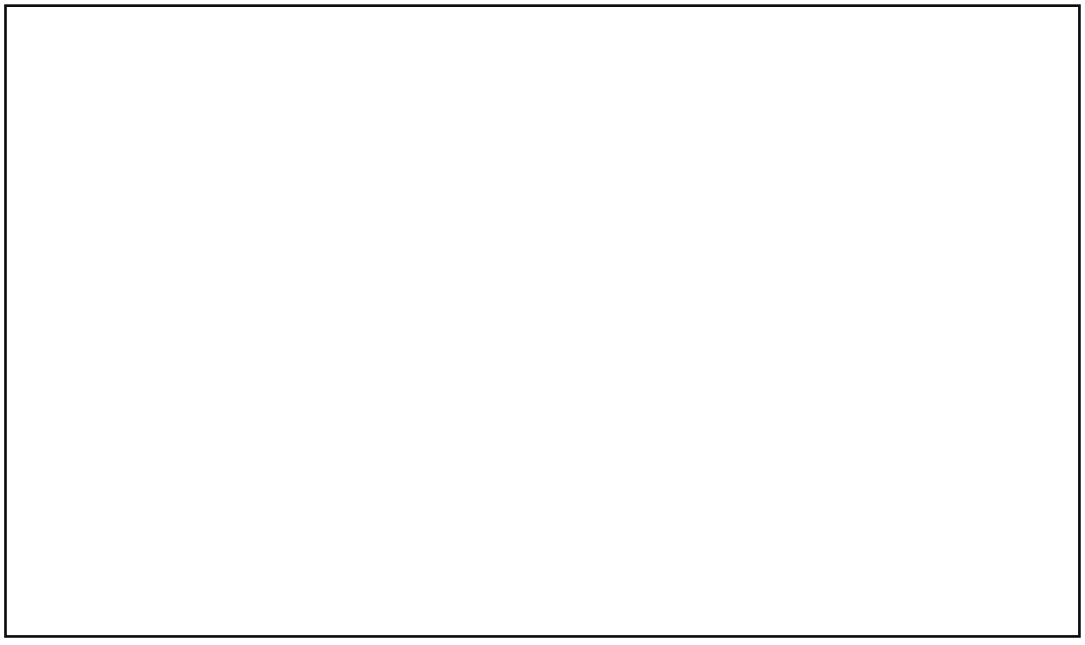
EXERCISE :

Fill the blank with CORRECT image from unity project (labsheet_1) : (use snapping tool)

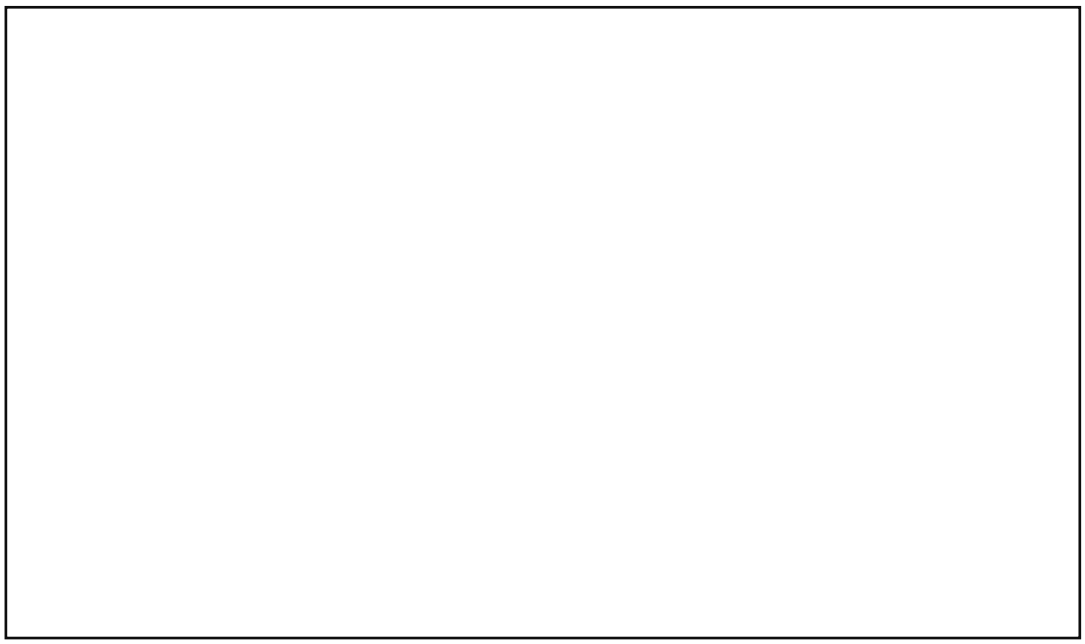
1. **GAMEOBJECT** is the most important concept in the Unity Editor. Every object in your game is a GameObject, from characters and collectible items to lights, cameras and special effects.



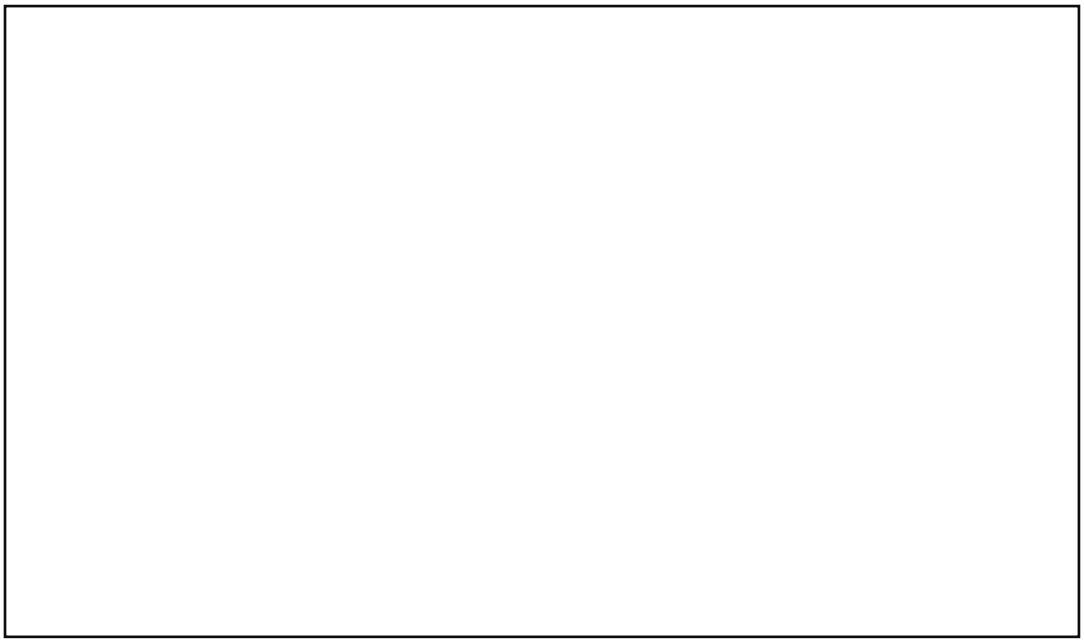
2. **SCENE** are where you work with content in Unity. They are assets that contain all or part of a game or application.



3. **Audio** can be it background music or sound effects. It can import most standard audio file formats and has sophisticated features for playing sounds in 3D space, optionally with effects like echo and filtering applied.



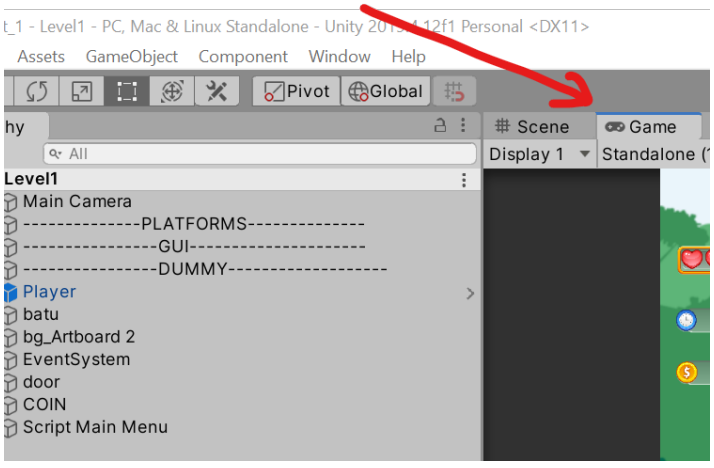
4. **Font** used to display the text. Text can be used to provide captions or labels for other GUI controls or to display instructions or other text.



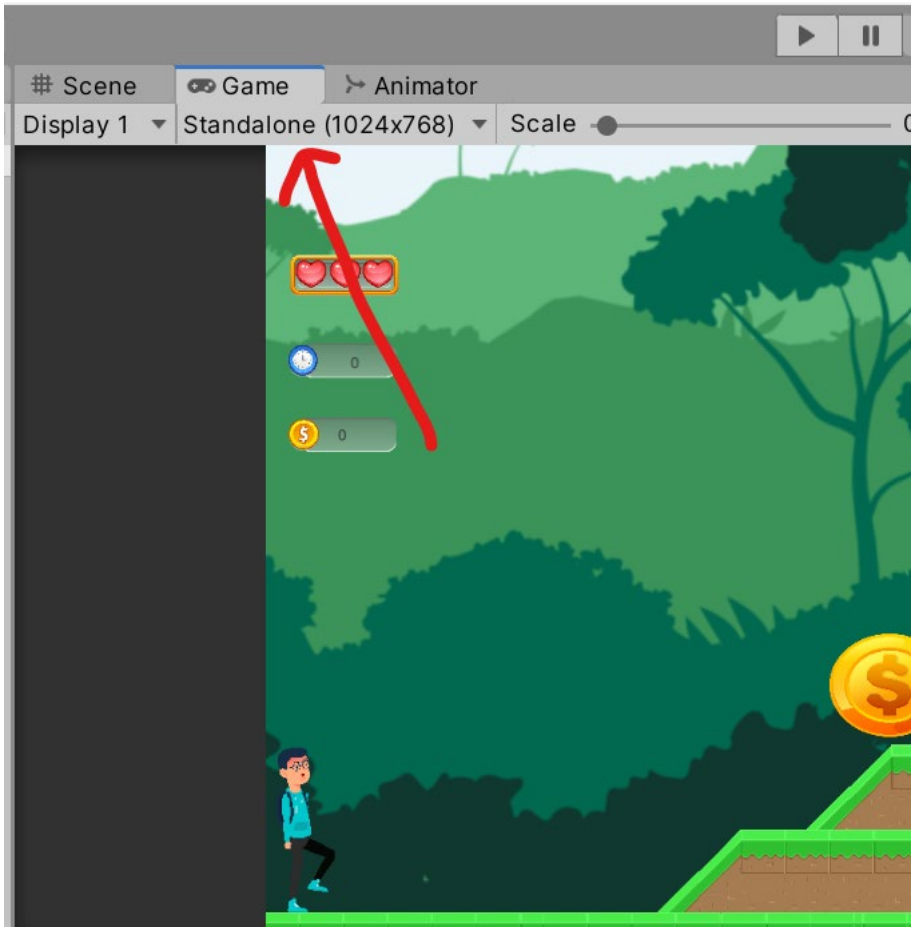
5. **Script** is an essential ingredient in all applications you make in Unity. Most applications need scripts to respond to input from the player and to arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behaviour of objects or even implement a custom AI system for characters in the game.

D) GAME RESOLUTION SETTING

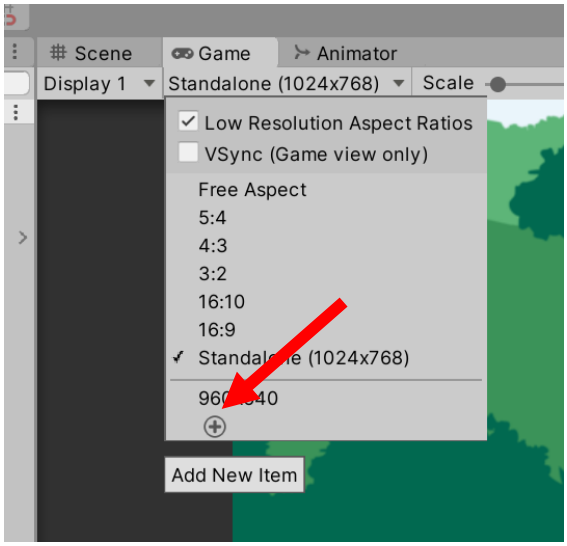
1. Open previous game project (C.1)
2. Select Game View



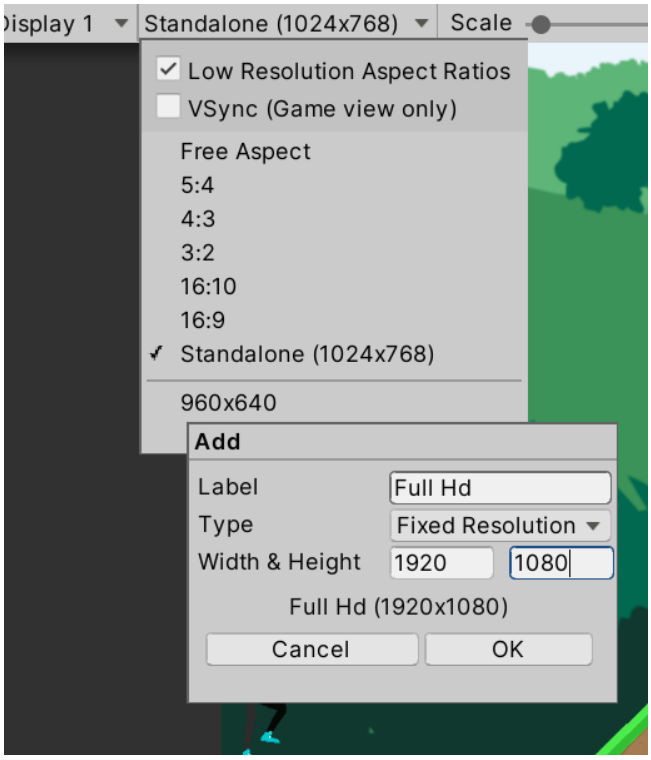
3. From DISPLAY 1 select STANDALONE (1024X768)



4. Select (+) button to add new Resolution



5. Insert New value as below :
    a. LABEL = Full HD
    b. Width & Height = 1920 ; 1080



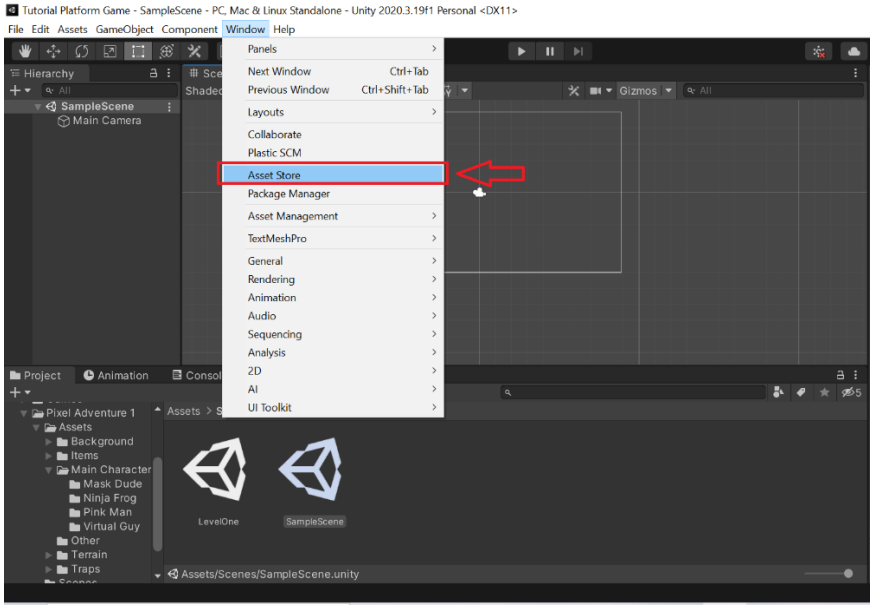
6. Select OK button

Topic 2.0 : Tilemap & Player Movement

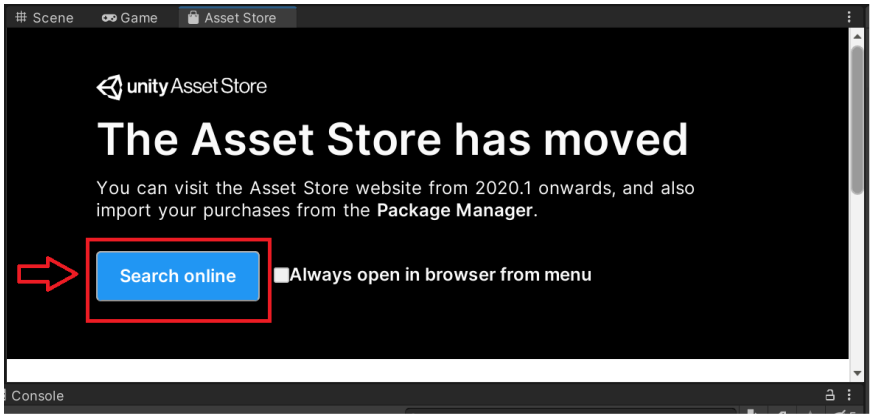
A) Downloading Pixel Adventure 1 from Unity Asset Store

1. Open Asset Store

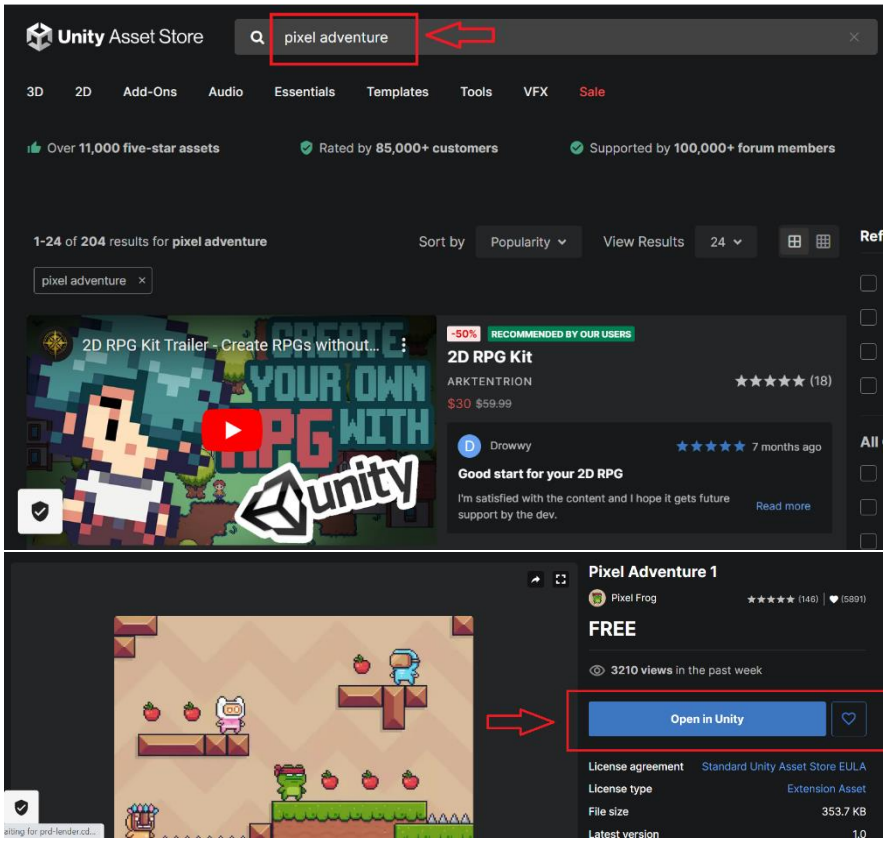
a. Click on Window tab in unity then click on Asset Store



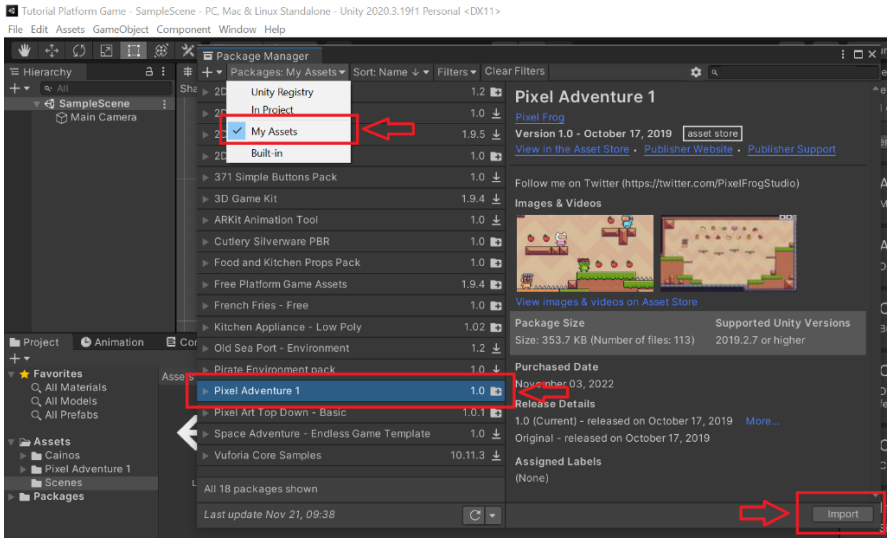
b. A popup will open and click on Search Online



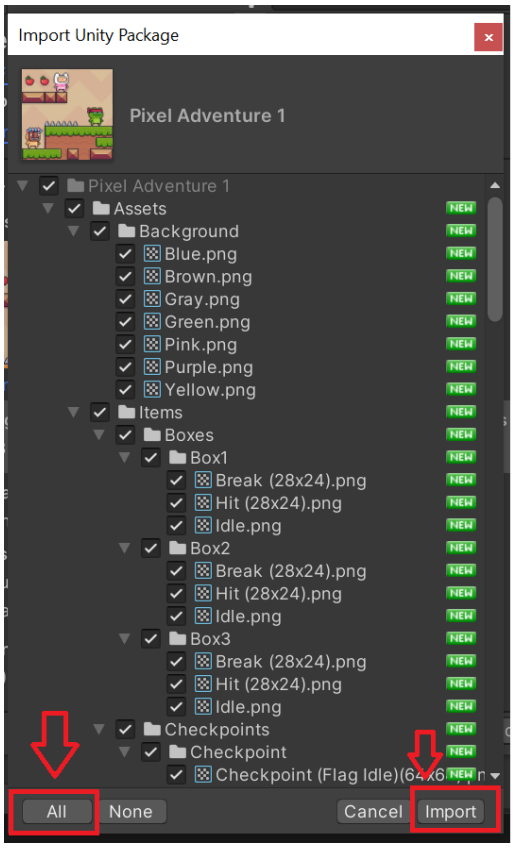
c. Type Pixel Adventure in asset store (chrome). Click Pixel Adventure 1 to download and click Open in Unity

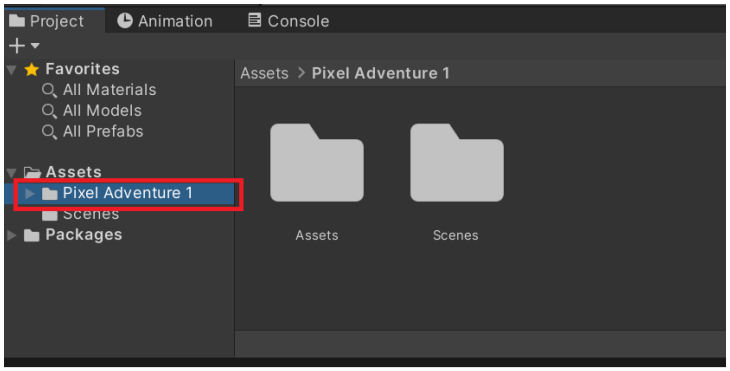


d. Click Window > Package Manager > Packages: My Assets > Pixel Adventure 1. Click download and then import to your project



e. A popup of Import Unity Package will be open, click All and then click Import

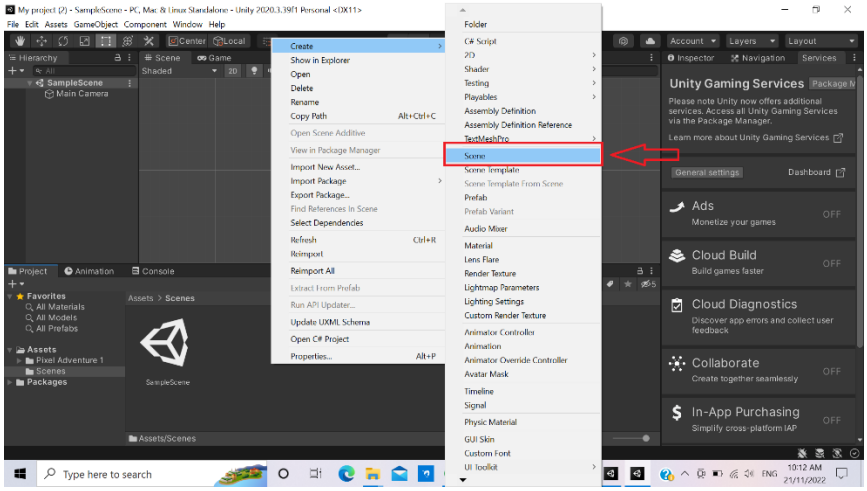


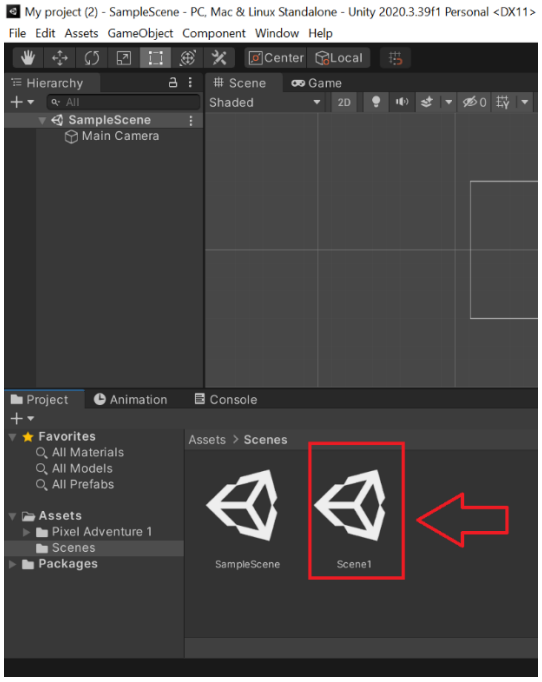


f. Tips: you must login to your unity id before downloading and importing the asset from Unity Asset Store

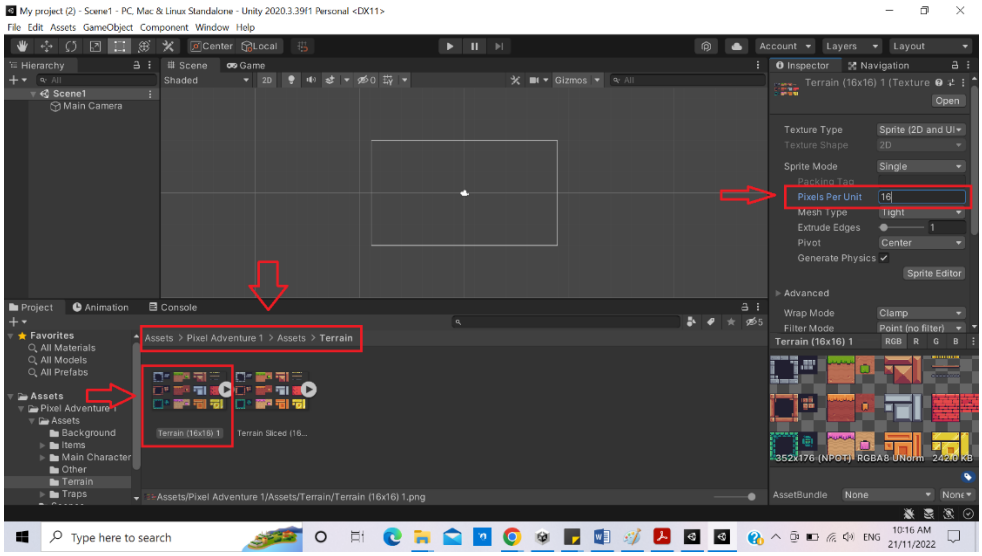
2. Create a Tilemap & Tile Palette

a. Create a new scene name “Scene1” in Scene folder

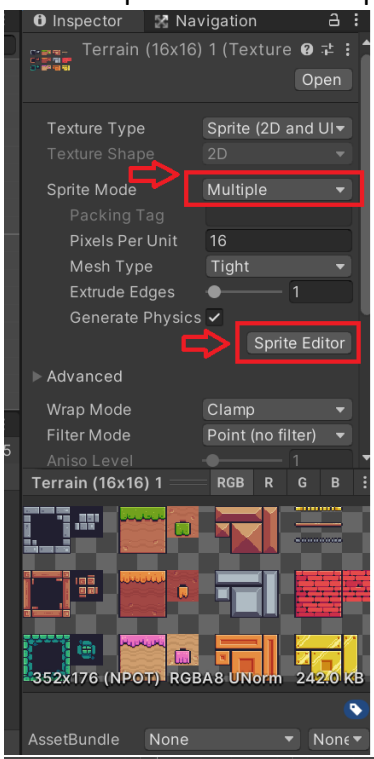


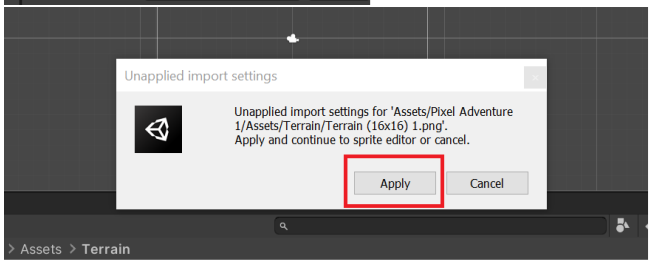


b. Navigate to Assets > Pixel Adventure 1 > Assets > Terrain. There will be two sprites, click on the first sprite and change the size (Pixels Per Unit) in the inspector panel to 16 and click Apply.

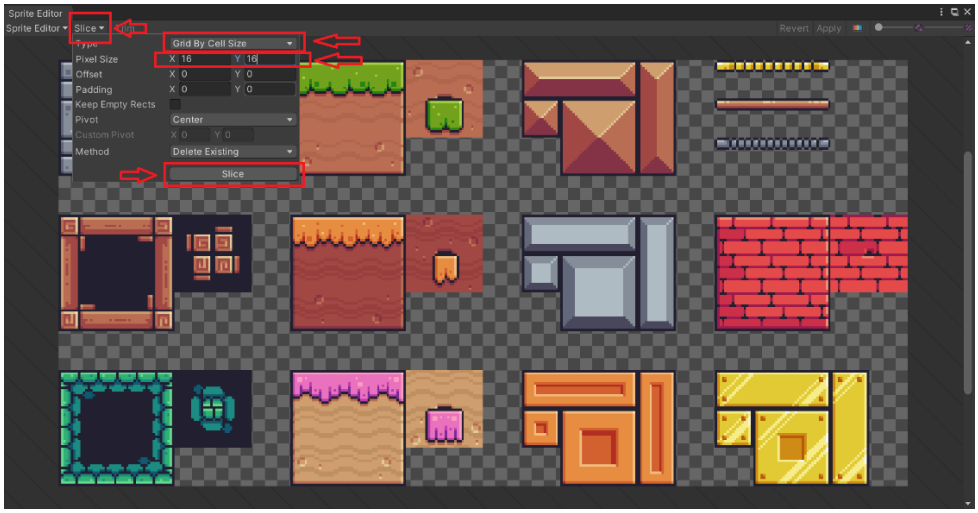


c. Then, change the sprite mode to multiple and then click on Sprite Editor in the Inspector of Terrain sprite and click Apply if a popup appears

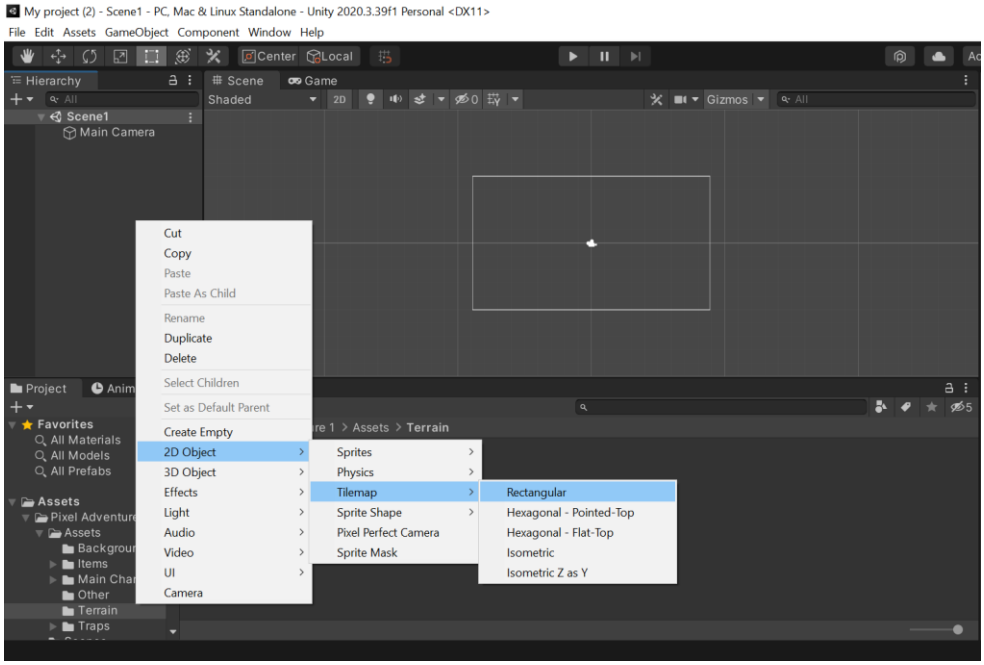




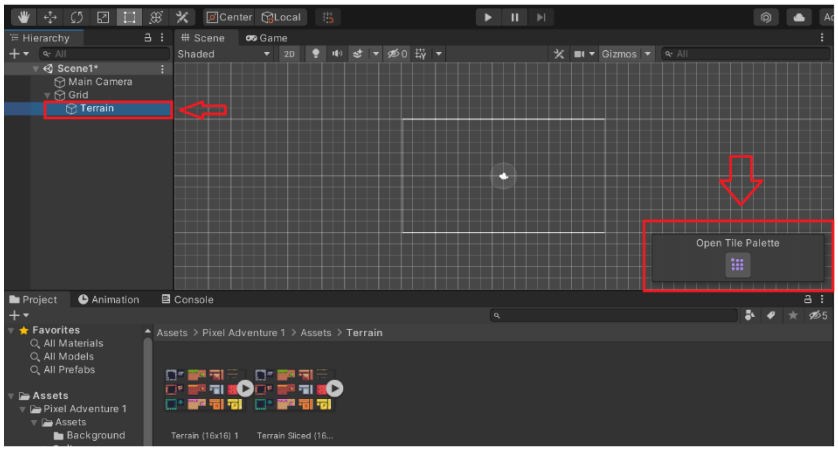
d. Sprite editor popup will be appearing, click on Slice and click Type: Grid By Cell Size. Change the Pixel Size 16 x 16 and then click Slice and then click Apply once the sprite slices successfully

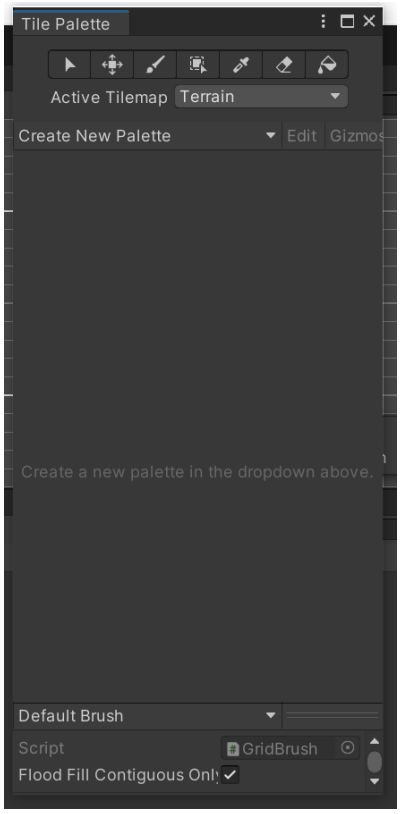


e. Right click on the hierarchy panel to create a tilemap. Click on 2D Object > Tilemap and click Rectangular

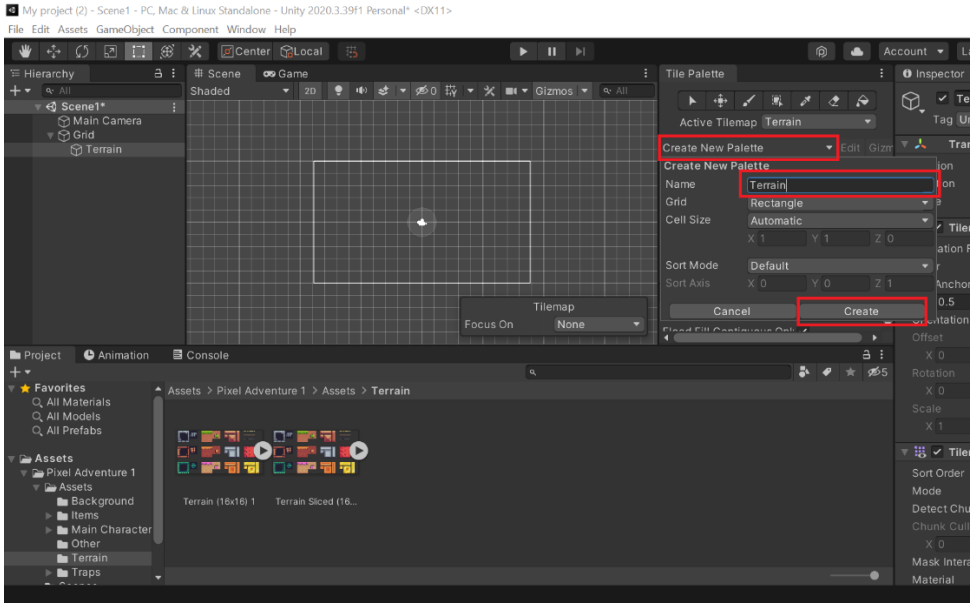


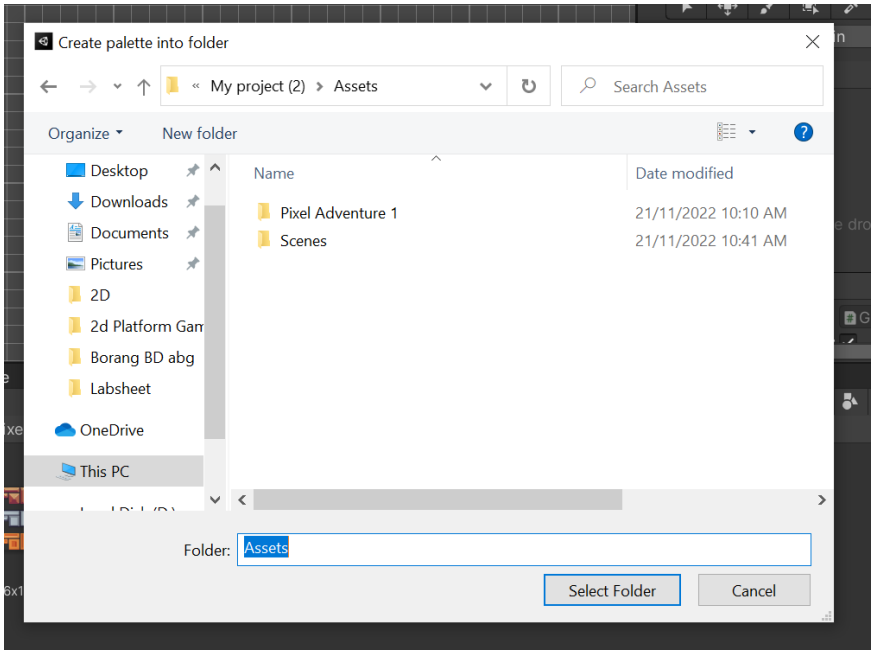
f. Rename the Tilemap to Terrain and click Open Tile Map. A tile palette popup will appear



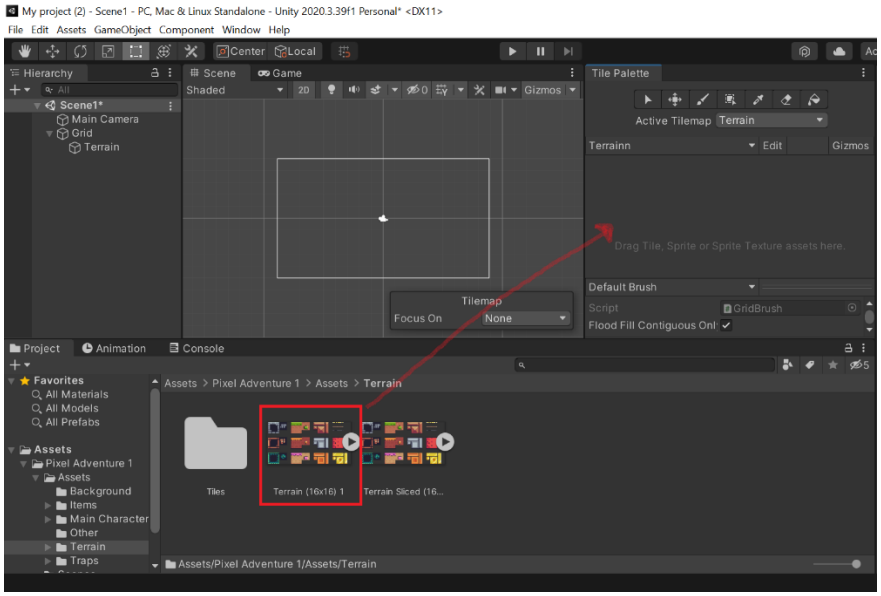


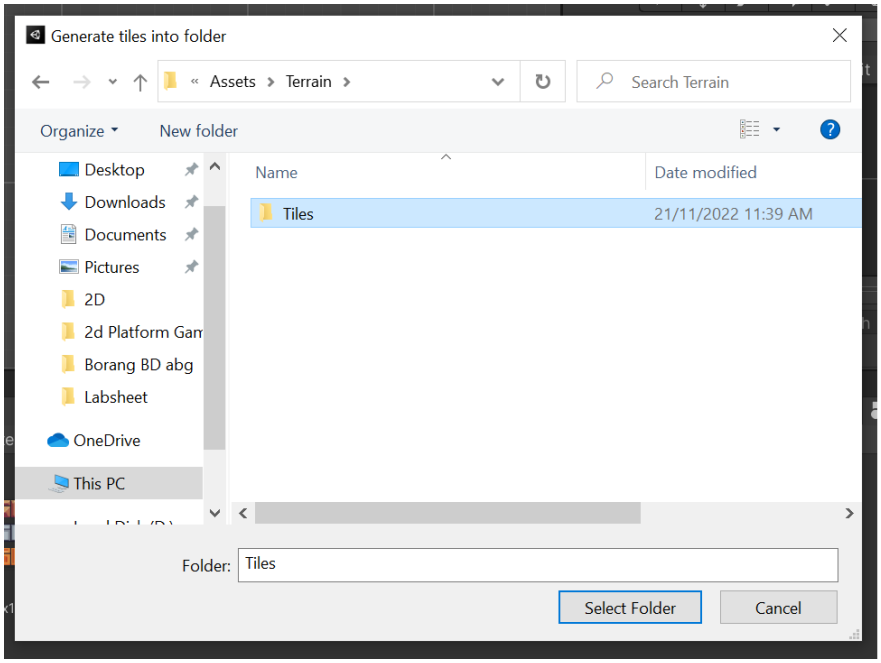
g. Click “Create New Palette” then name it as Terrain and then click “Create” and save the palette in Assets folder



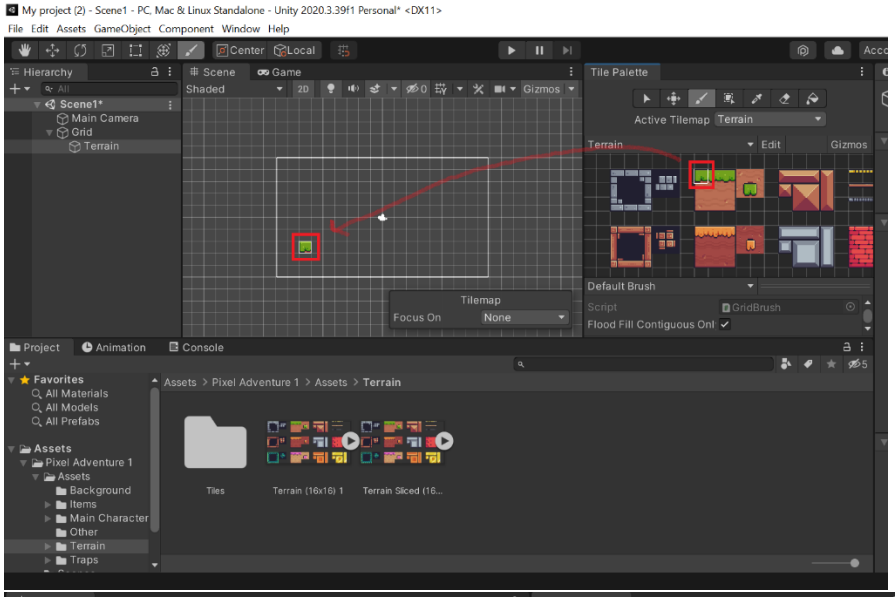


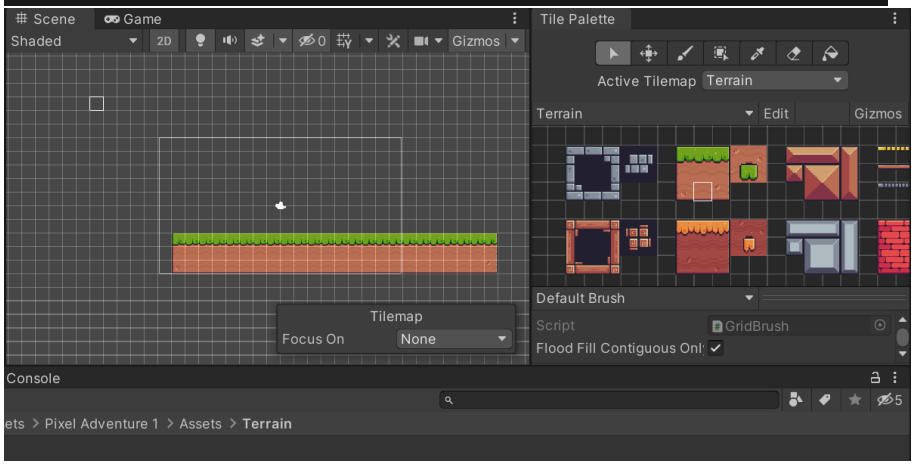
h. Then, drag the sprites that been slices (16x16) into the tile palette and save the tiles to a new folder name Tiles in the Terrain folder



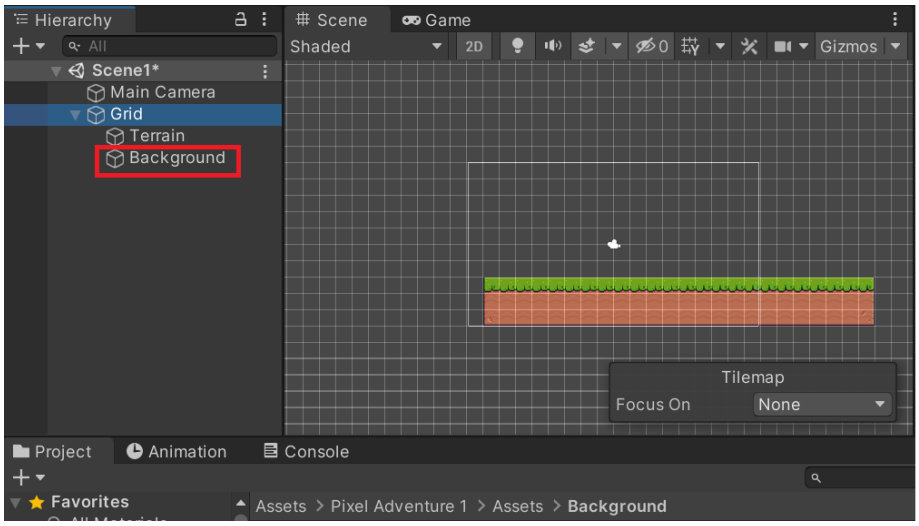


i. Then click on the tile in the tile palette and click on the Terrain scene to start creating the paring. The result shown as below

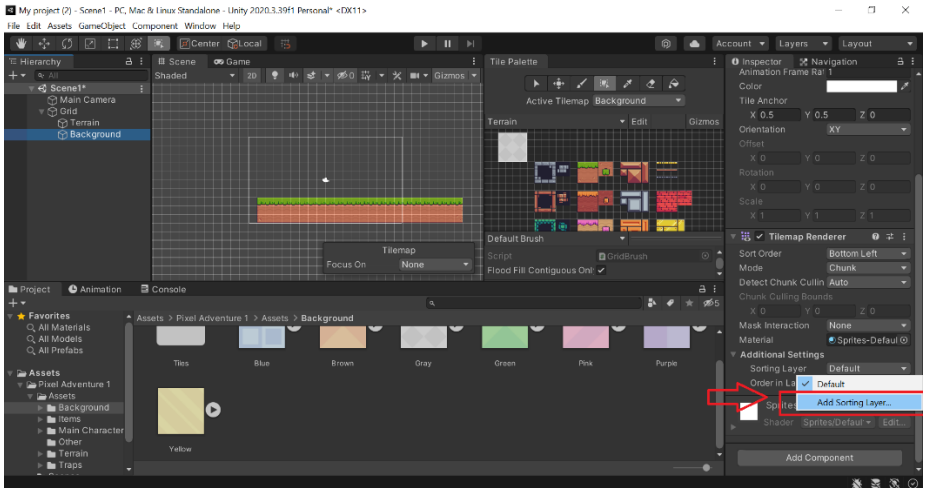




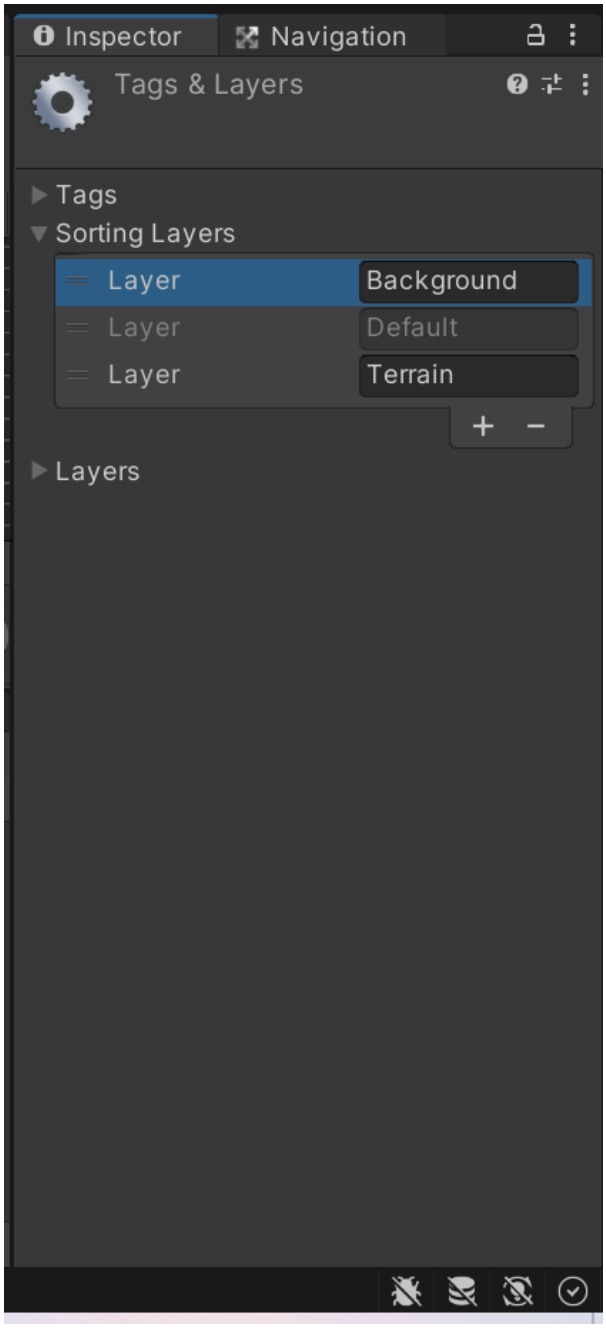
j. Create empty rectangular tilemap for “Background”



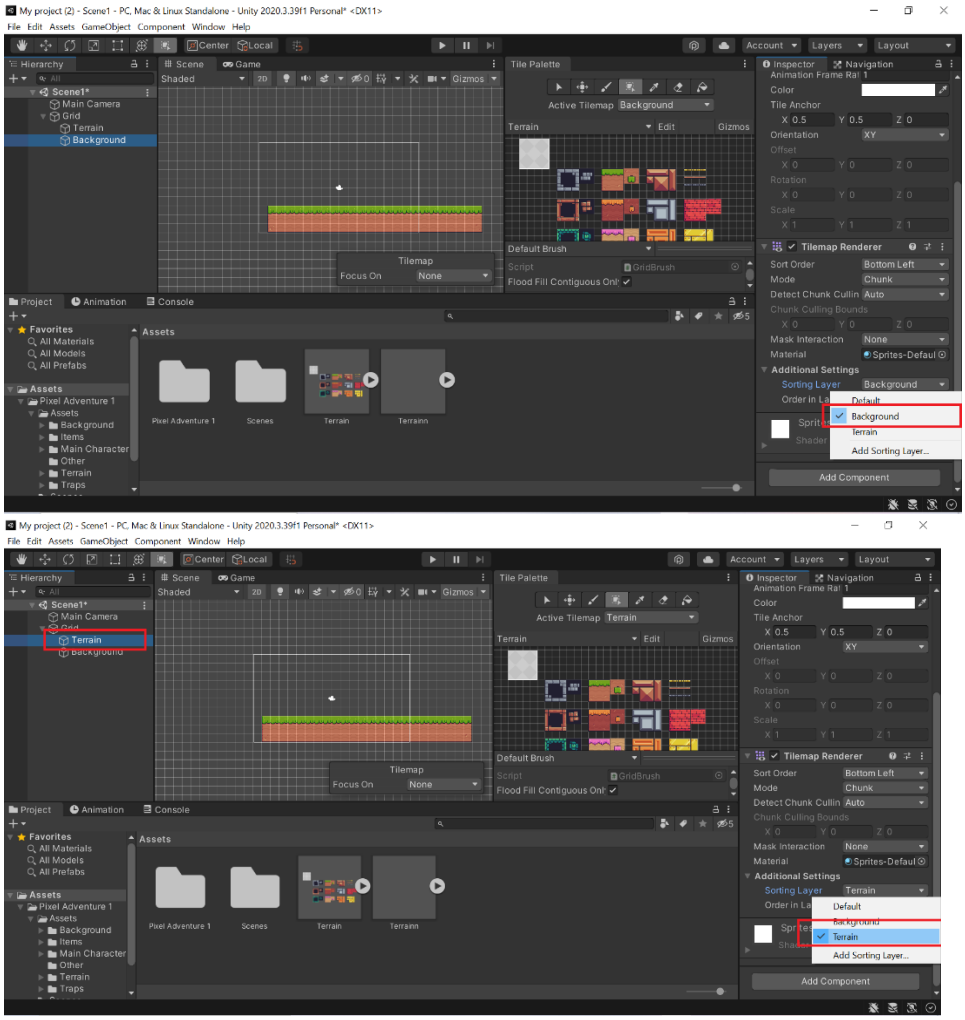
k. Click on Background in the hierarchy and click on Add Sorting Layer in the inspector



l. Click + to add sorting layer. Then add 2 more sorting layer and rename to “Background” and “Terrain”. Sort the position of the sorting layer as below

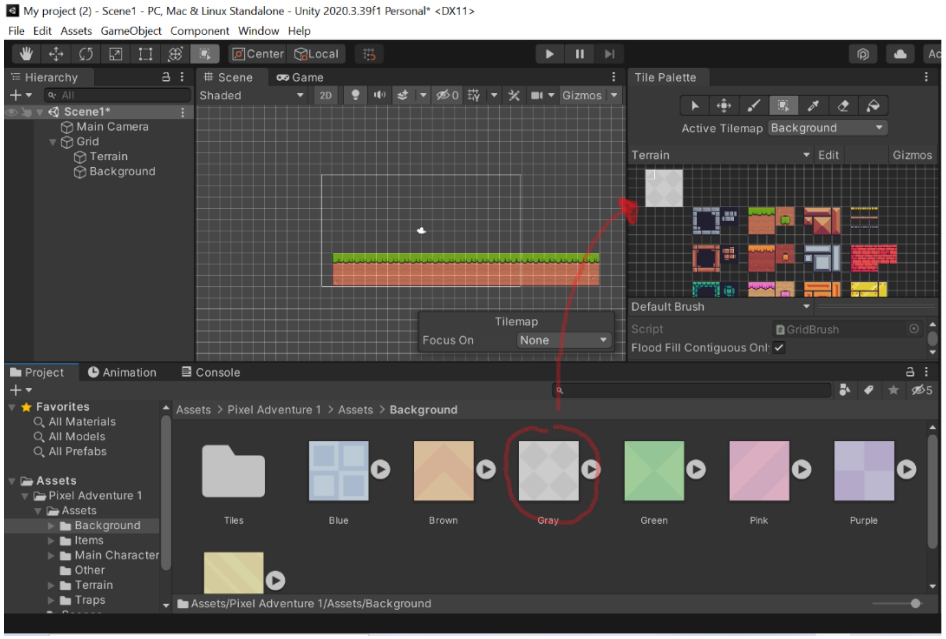


m. Then click on Background in the hierarchy and change the sorting layer to Background. Do the same to Terrain game object in the hierarchy

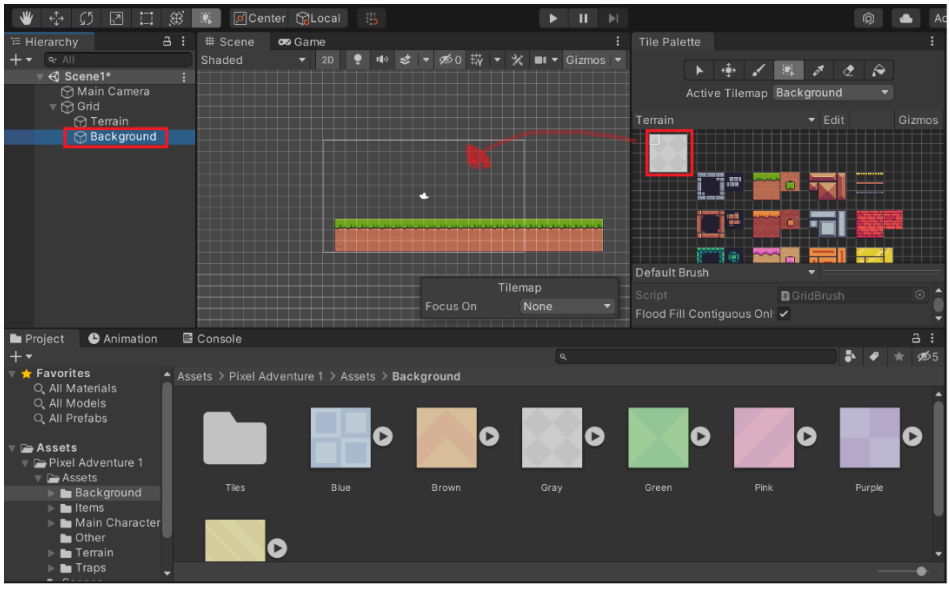


n. Repeat step 2 (b) until 2 (2) for the Background game object. Make sure to use the sprite from Background Folder

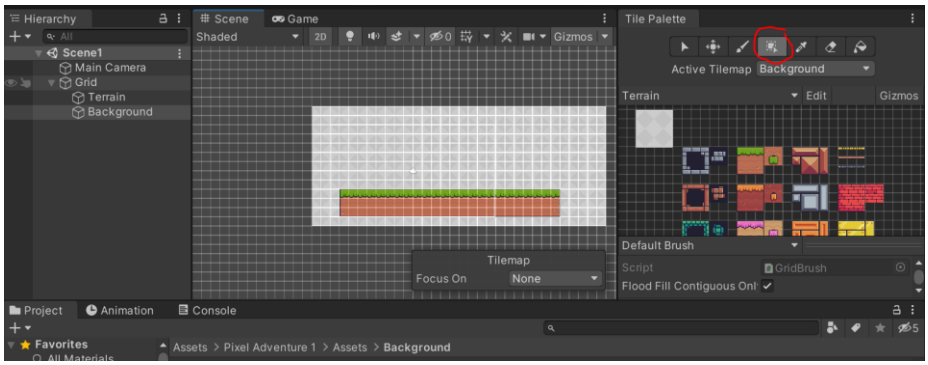
o. After slicing and creating sprite for the desire background, drag the sprite into the same Terrain Tiles used earlier. Make sure to save the background tile palette in new folder.



p. Then choose all the tiles for background and paste to the Background tilemap



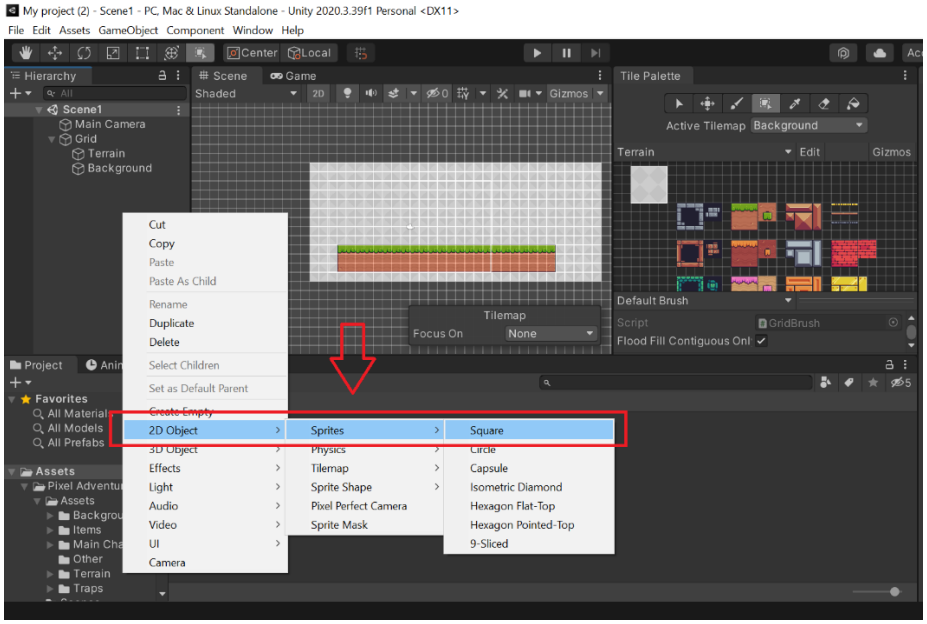
q. The results are as shown bellows. Make sure the background is at the back of the terrain

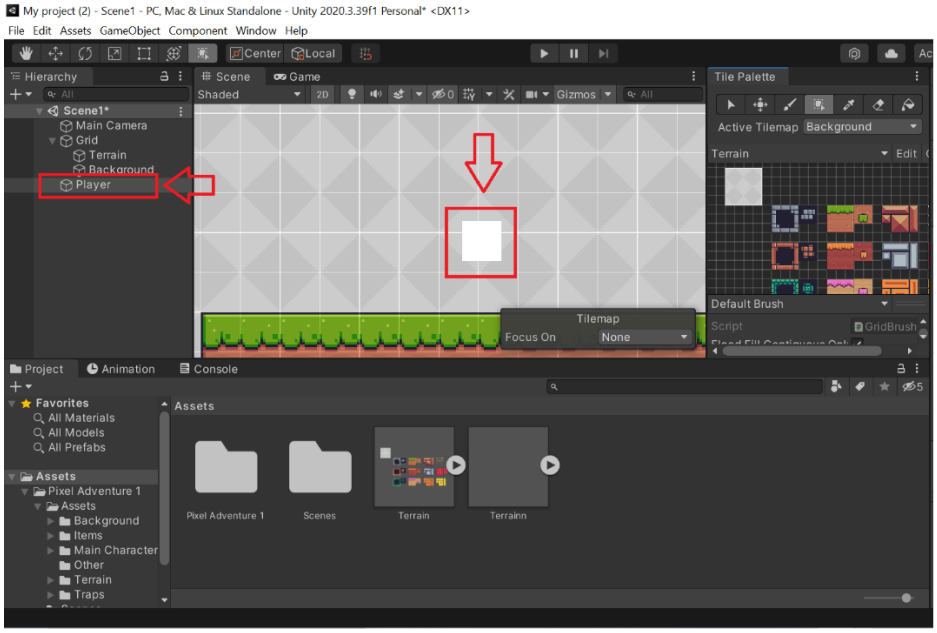


r. Done creating tilemap and tile

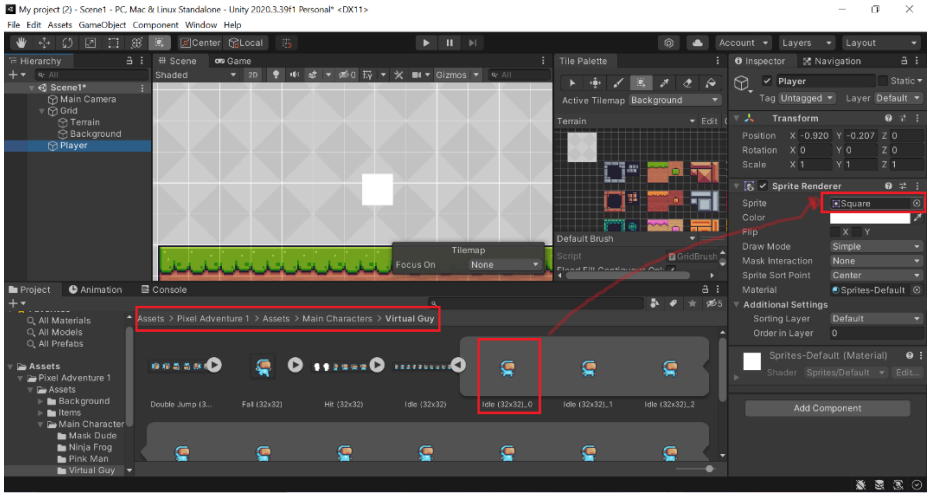
3. Create main player

a. Right click on the hierarchy and click on 2D Object > Sprites > Square. Then, rename the square to player

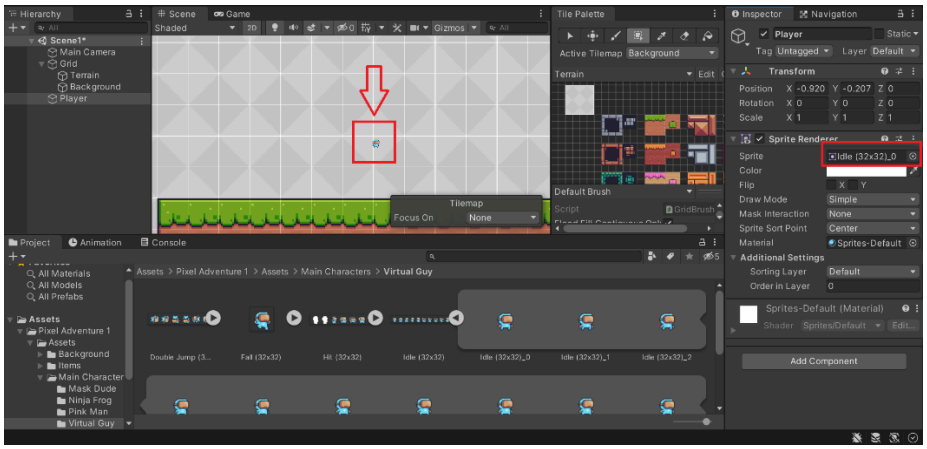




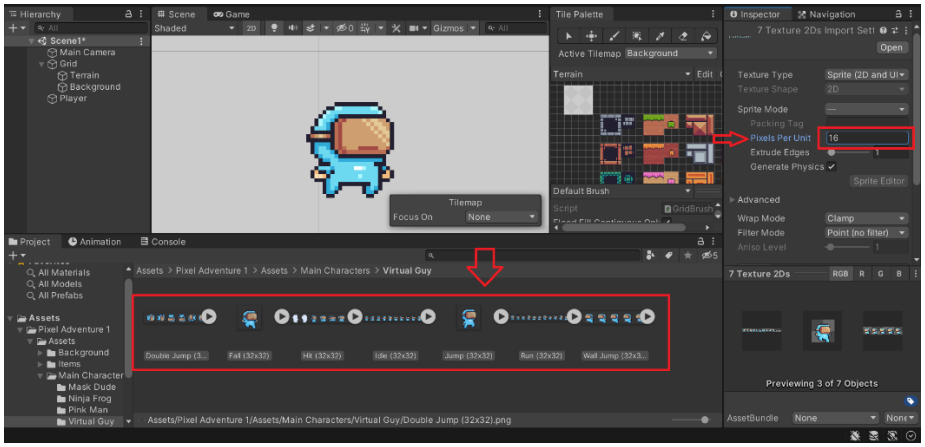
b. Click on the Player in the hierarchy then navigate to Assets > Pixel Adventure 1 > Assets > Main Characters > Virtual Guy. Drag the first sprite of Idle (32x32) into the Sprite at Sprite Renderer in the Player's inspector panel



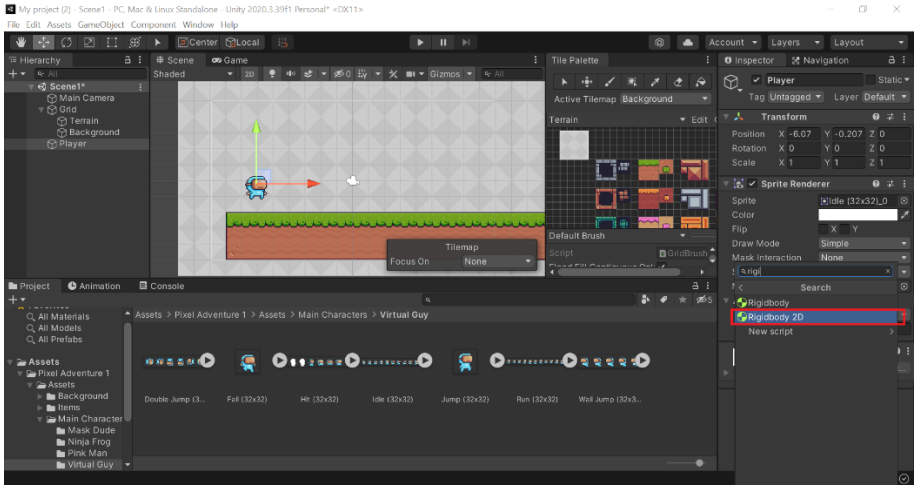
c. The square sprite will be changes to the Virtual Guy sprite as shown below



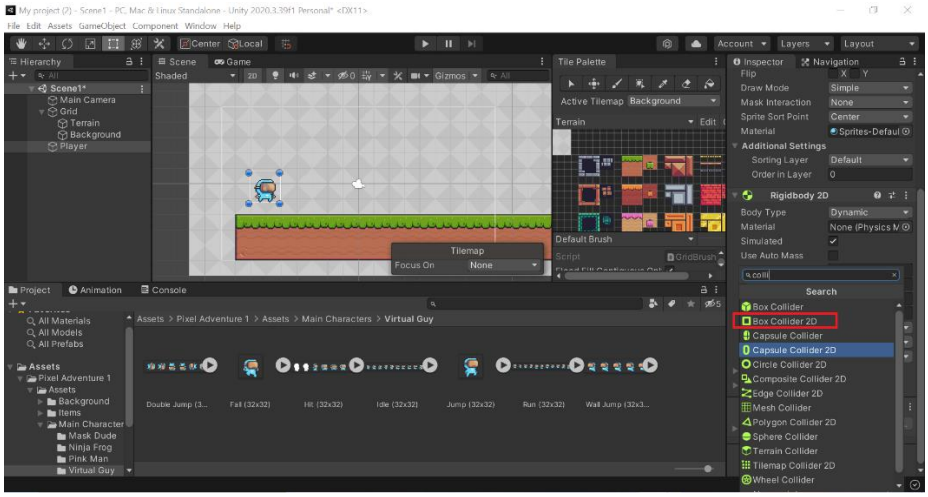
d. Change the size of all sprites available in Virtual Guy folder to 16 Pixels Per Unit and click on Apply at the bottom. Now the size of the player is proportionate with the terrain size



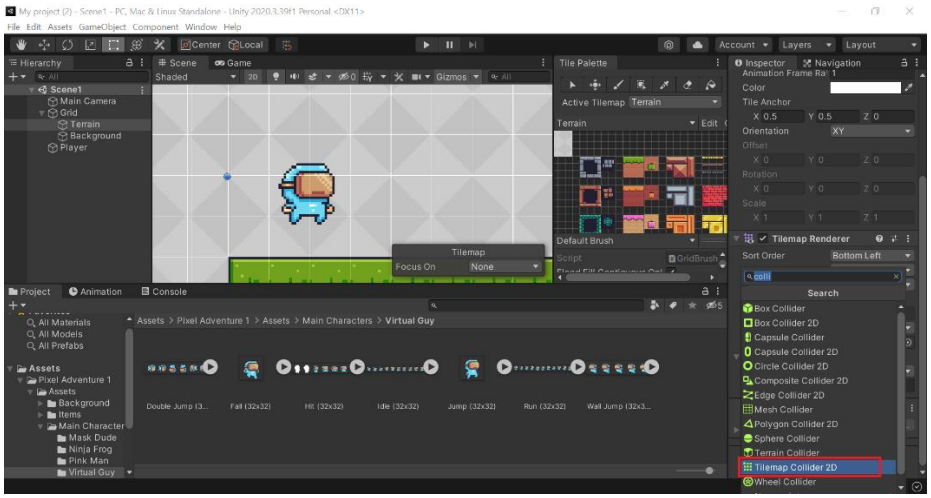
e. Next add Rigidbody 2D to the player in order to put a mass to the player by click on Add Component in the player inspector panel

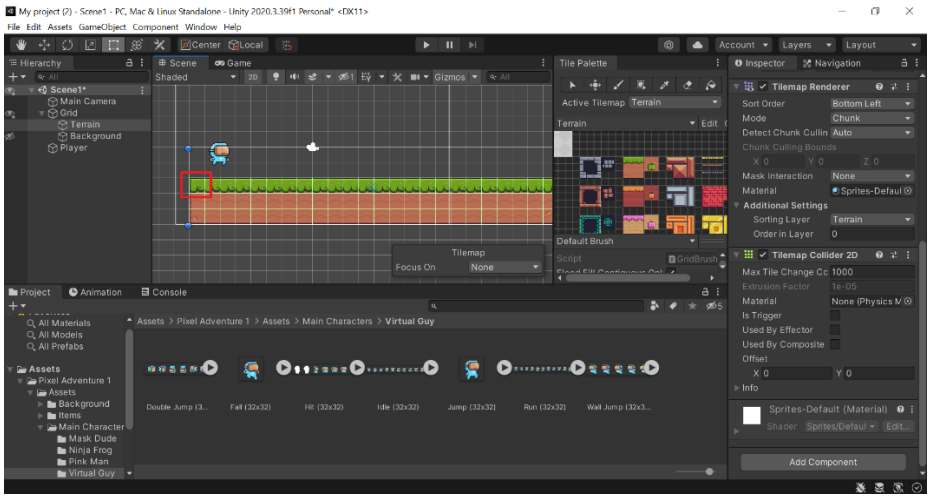


f. Next add Box Collider 2D to the player in order to put a mass to the player by click on Add Component in the player inspector panel
Tips: Collider is used to make sure the player able to collide with the terrain

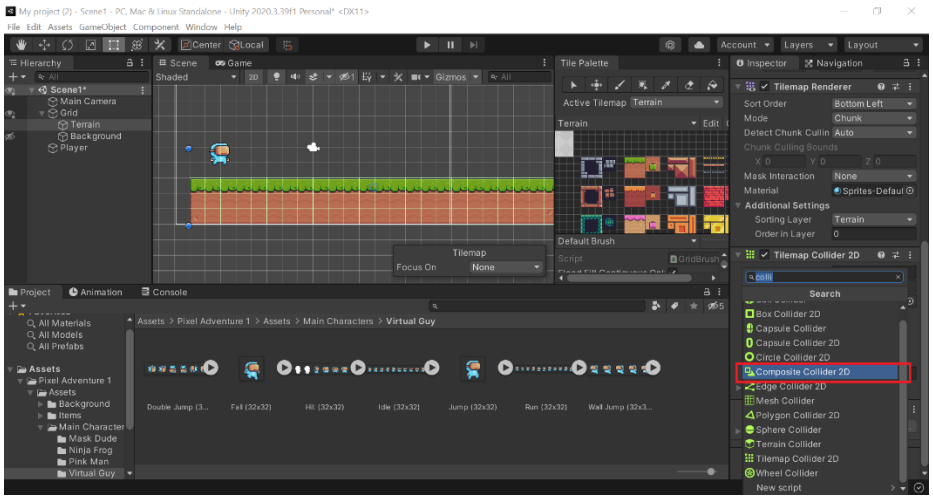


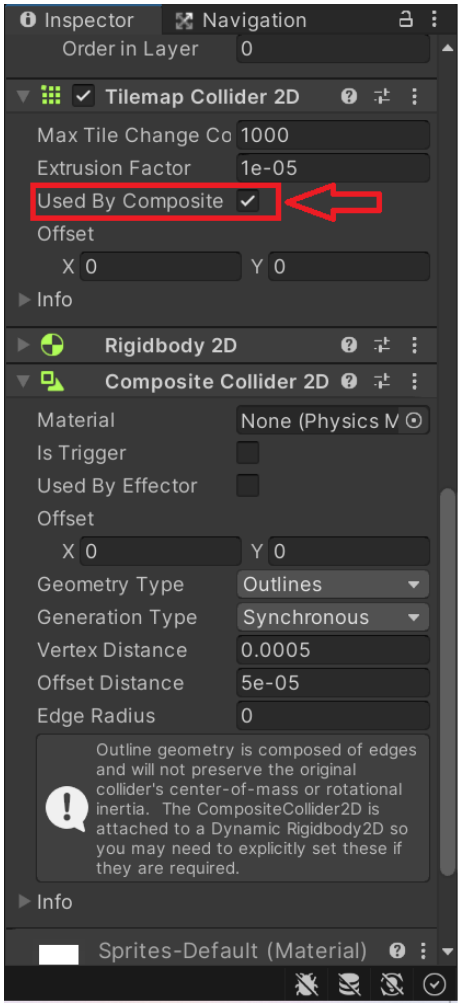
g. Next add Tilemap Collider to the terrain in order to make sure the player able to stop when collide with the terrain by click Add Component in the Terrain Inspector and click on Tilemap Collider 2D
Tips: Tilemap Collider 2D will put collider on each of the tile palette for the terrain tilemap as shown below



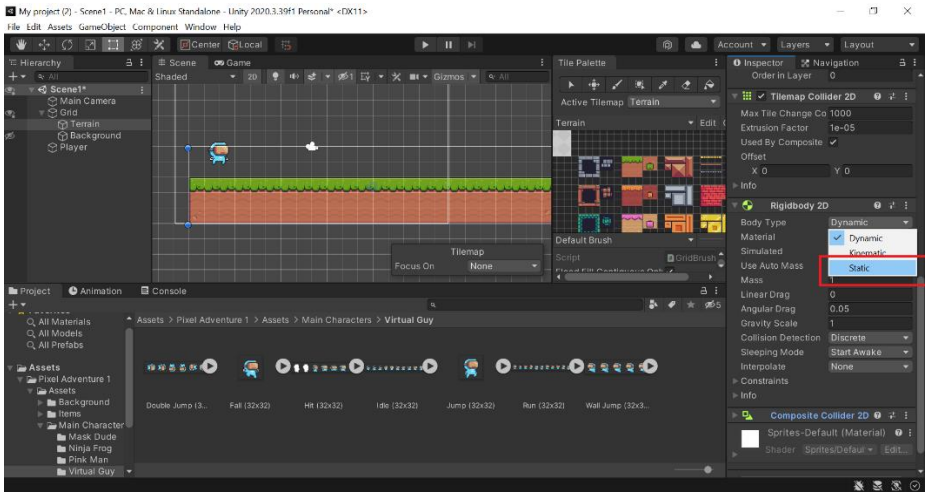


h. Next add Composite Collider to the Terrain in order to merge all the tilemap collider into one big collider and then Check on Used By Composite at the Tilemap Collider 2D as shown below





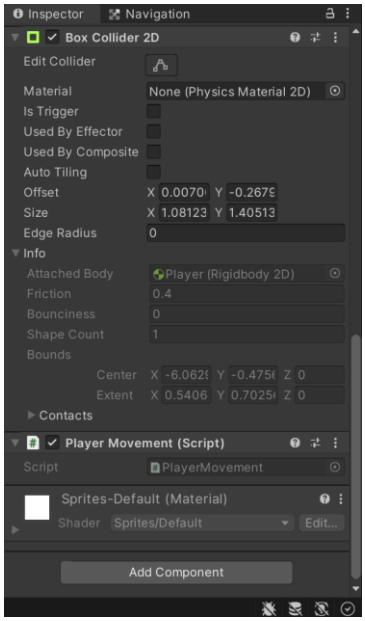
i. Next change the Body Type of Rigidbody 2D in Terrain inspector to static in order to remove gravity concept from the terrain



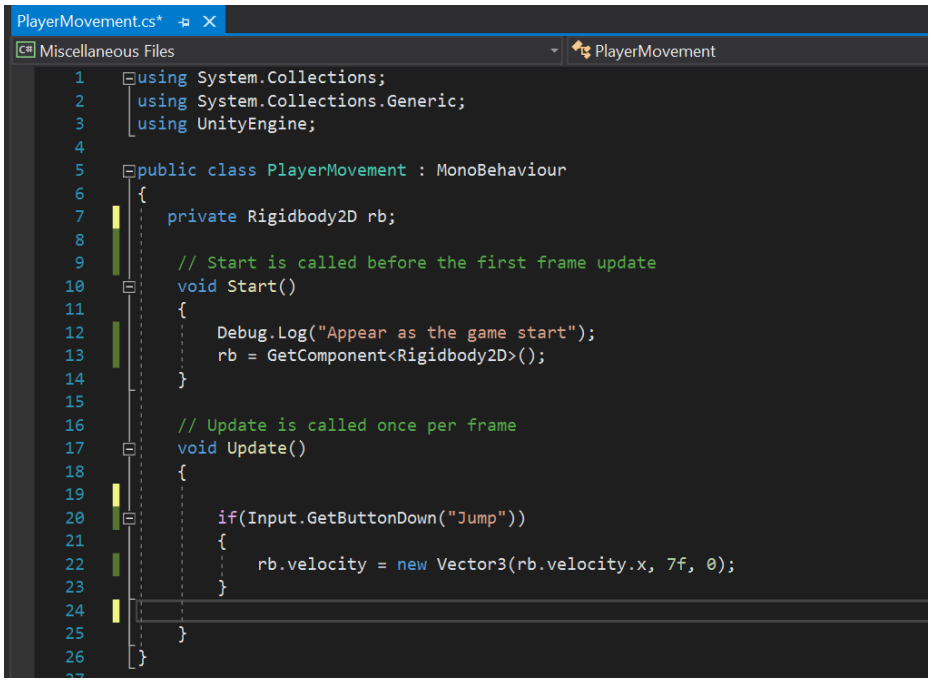
j. Play the game and observe the result

4. Create Player Movement

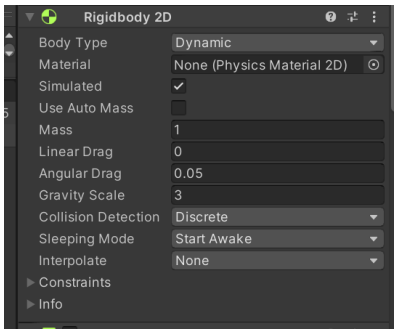
a. Create a new C# script name “PlayerMovement” and attach the script to Player game object



b. Next open the script using any IDE and add the following script to make the player jump up (7) when space button is push down (for PC)



c. Change the gravity scale of the player to 3 in order to make the player jump lower when space is push



d. Play the game and observe

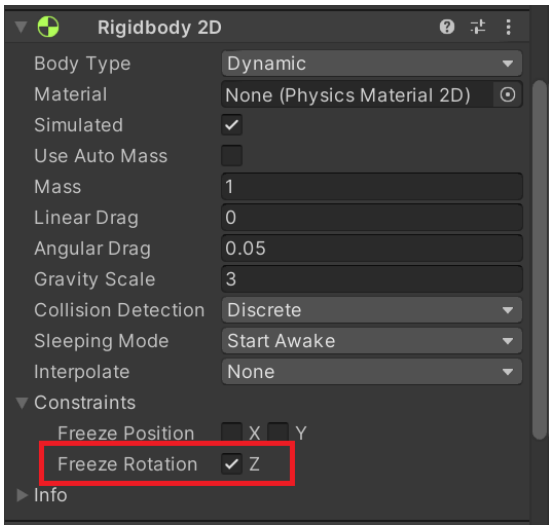
e. Next, modify the coding to add left and right movement to the player by adding the following code in the Update method in PlayerMovement script

Tips: Do not forget to declare the dirX variable before start method

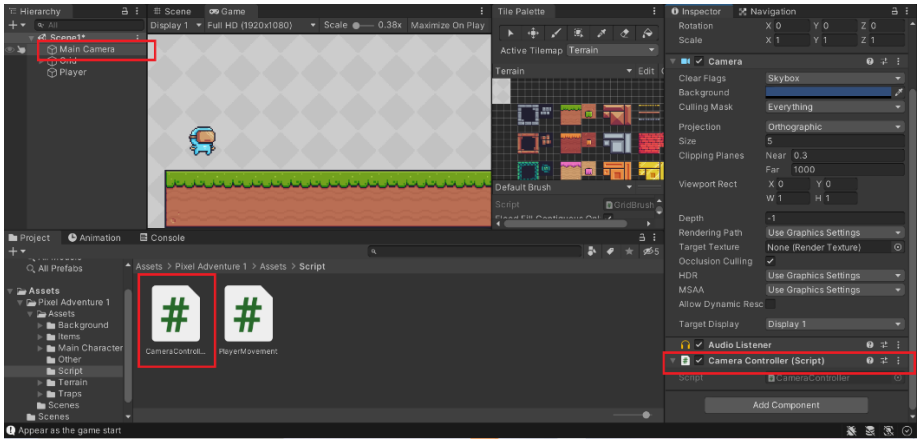
```
private float dirX = 0f;
```

```
void Update()
{
    dirX = Input.GetAxisRaw("Horizontal");
    rb.velocity = new Vector2(dirX * 7f, rb.velocity.y);

    if(Input.GetButtonDown("Jump"))
    {
        rb.velocity = new Vector3(rb.velocity.x, 7f, 0);
    }

}
```

f. Next freeze rotation Z of the player to disable the player from rotating in Z position by navigate to Player game object > Rigidbody 2D component in the inspector

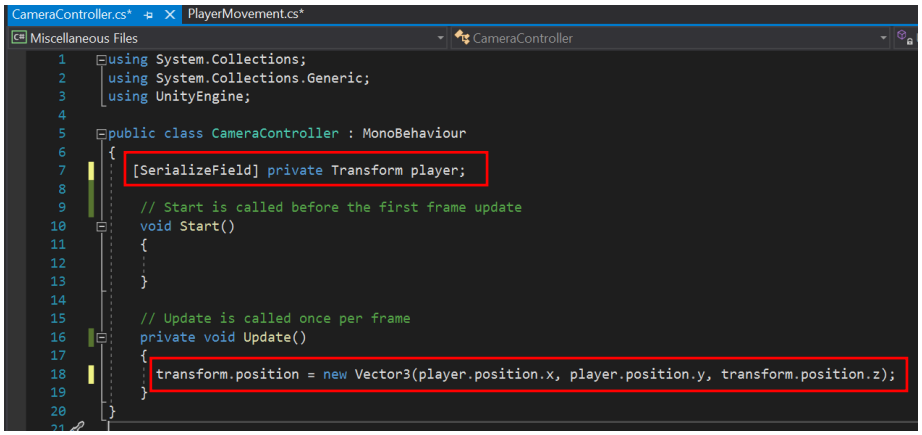


g. In order to make the camera follows the player create a new script name CameraController and attach to the Main Camera game object in the hierarchy



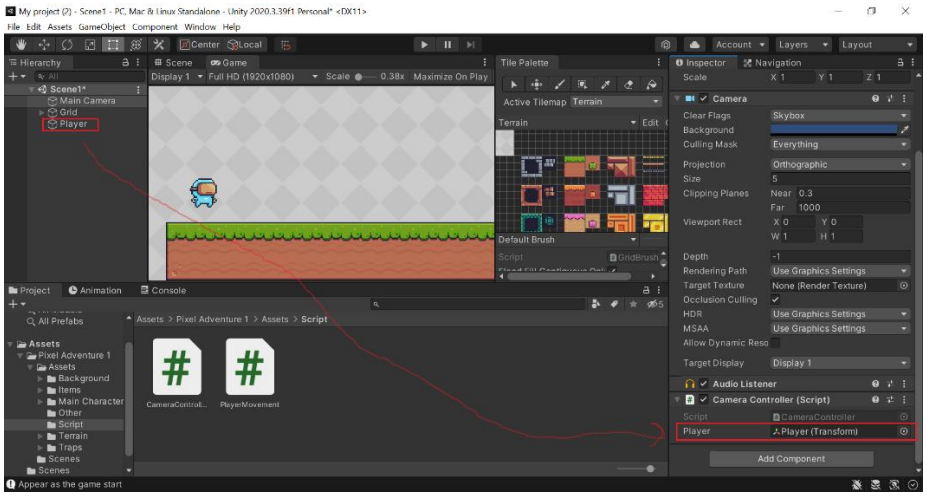
h. Open the CameraController script in Visual Studio and add the following code

Tips: [SerializeField] is use to make the value of available to be access in unity. We can achieve the same result by using public



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

public class CameraController : MonoBehaviour
{
    [SerializeField] private Transform player;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    private void Update()
    {
        transform.position = new Vector3(player.position.x, player.position.y, transform.position.z);
    }
}
```

i. Next, drag the Player game object to variable Player in the Main Camera inspector panel



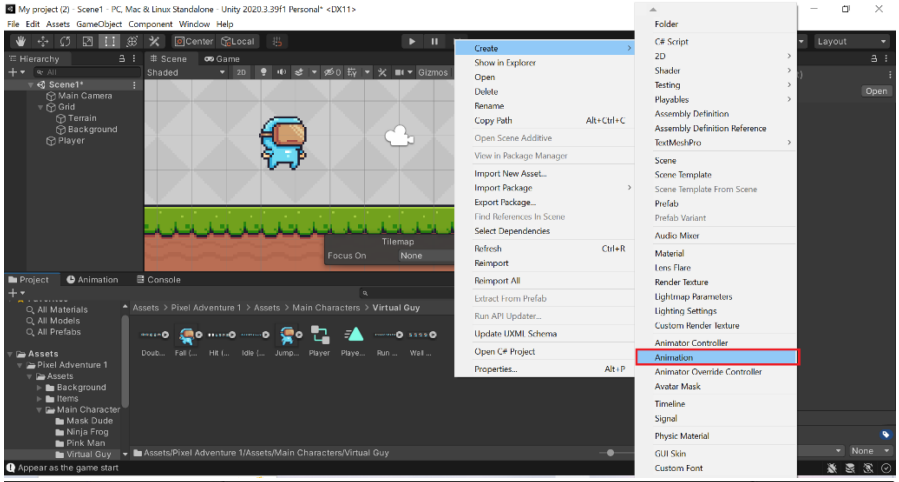
j. Click play and observe the result

Topic 3.0 : Player Animation

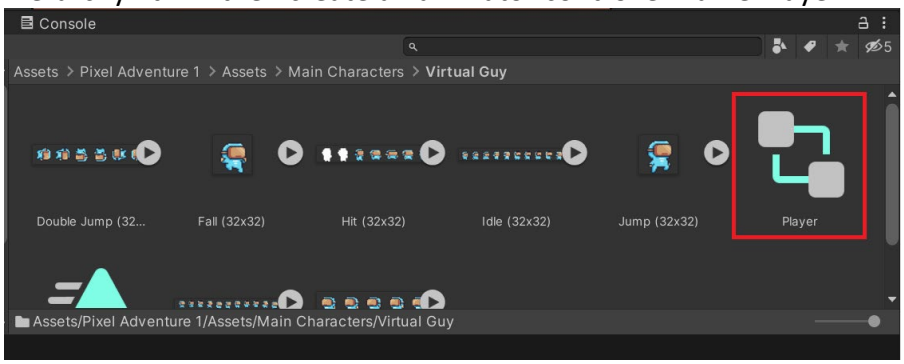
A) Create animation for the player

1. Create Animation for the main player

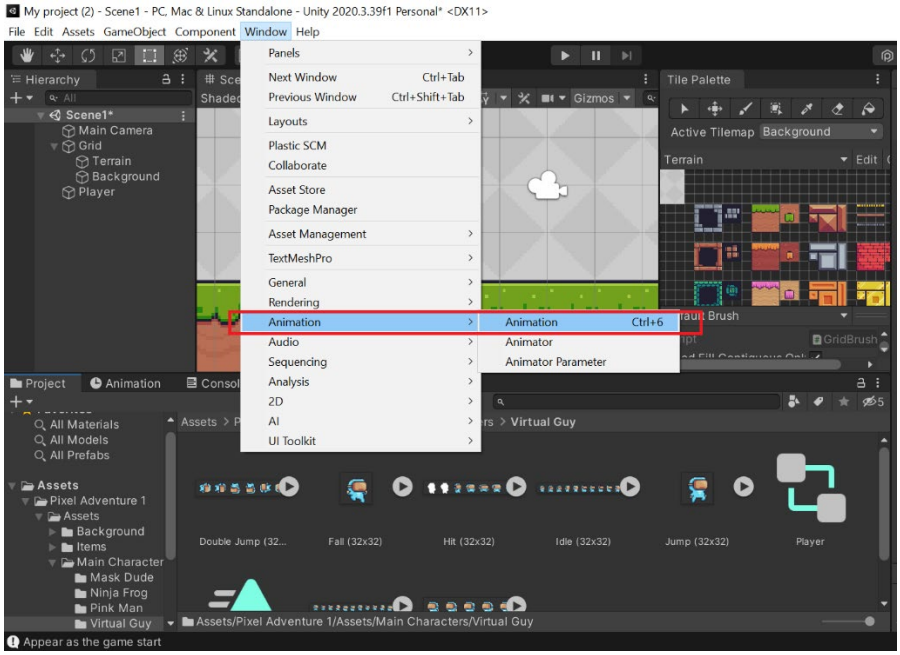
a. Right click on the project panel and click on Create > Animation to create an animation for the player. Rename the animation to Player_Idle

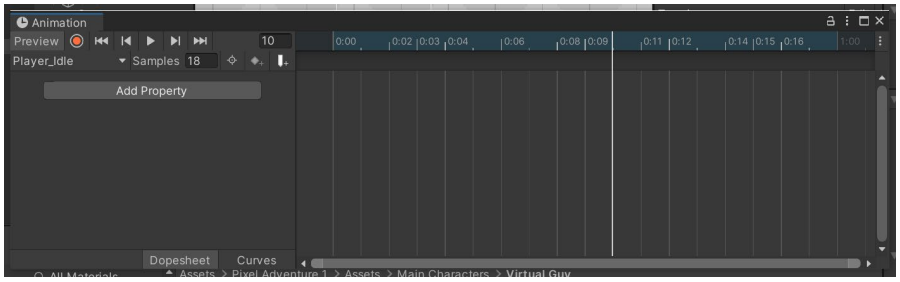


b. Next, drag the Player_Idle animation file to Player game object in the hierarchy. It will then create an animator controller name Player

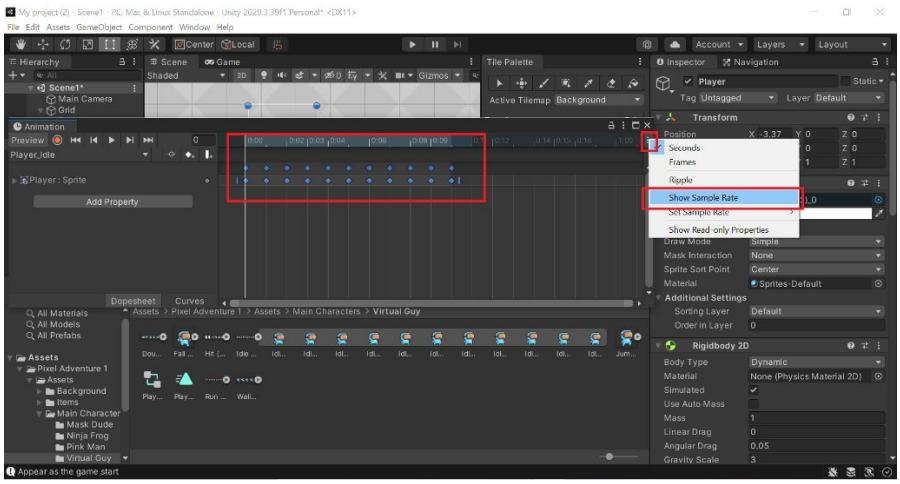


c. Then go to Window > Animation > Animation to open the animation state panel. An animation panel will be popup as shown

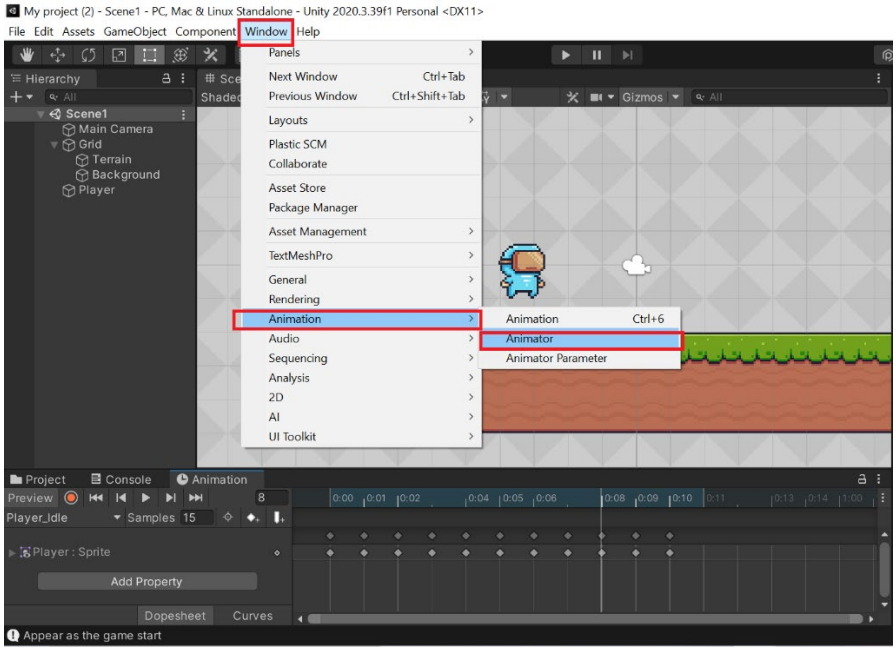




d. Next, drag the Idle sprites collection in the animation panel. Change the sample rate to 18

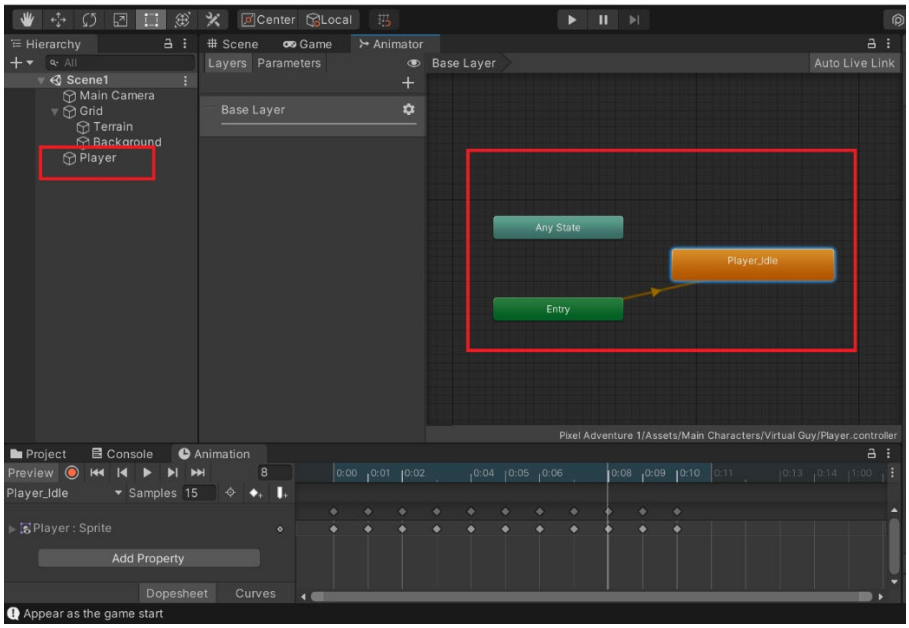


e. Next, open the animator controller panel by click on Window > Animation > Animator



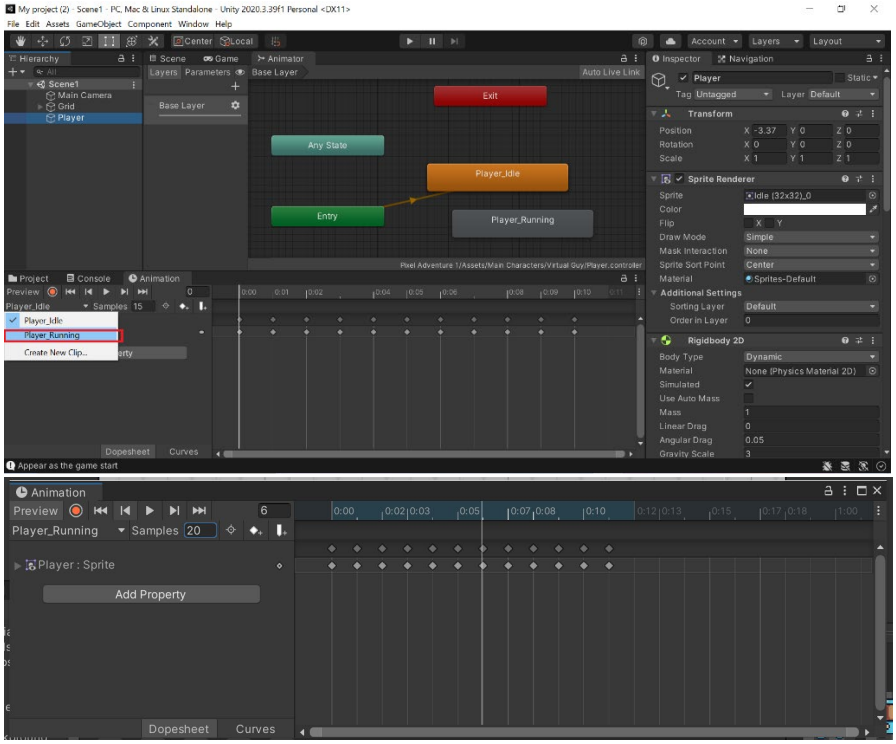
f. An animator controller panel will be open and there will be 4 state available

Tips : Click on Player game object to make sure the animation state are for player animation

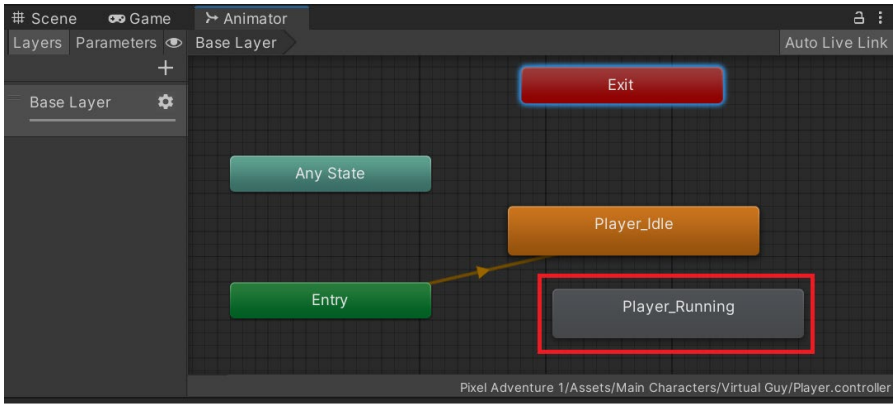


g. Next, create an animation for running and drag the animation to Player game object. Then open animation tab and adding running sprite to the Player_Running

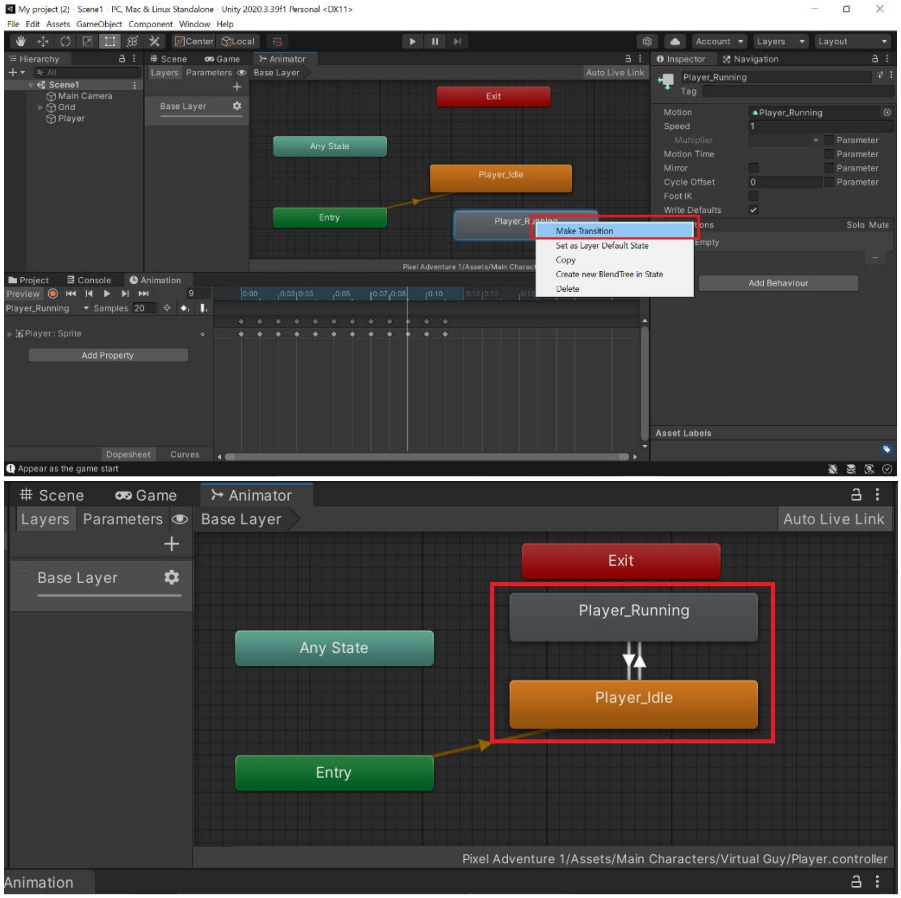
Note: Modify the sample rate of Player_Running to 20



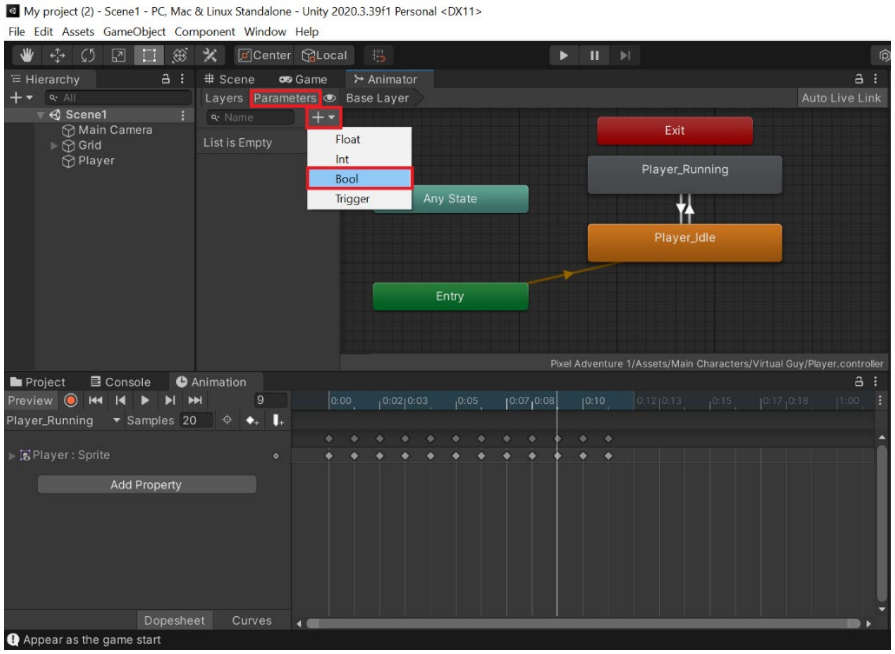
h. Open Animator controller and now there's a new state created name Player_Running as shown below



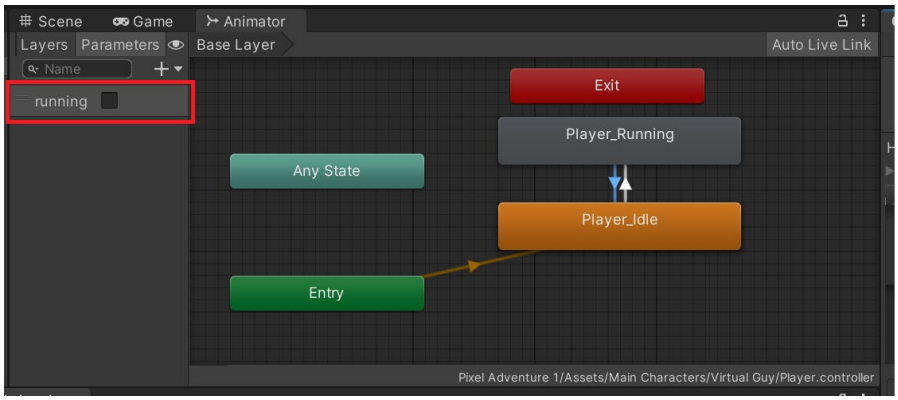
i. Next, in animator controller right click on Player_Running and click on Make Transition then drag to the Player_Idle state. Repeat the ssame process to Player_Idle state in order to switch the transition from running to idle and from idle to running



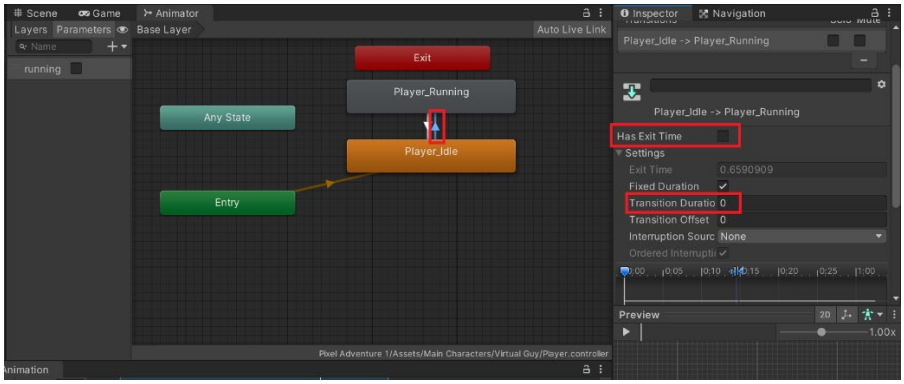
j. Next add parameter to the state in order to create the condition for the state by click on Parameters tab. Then click + sign and choose Bool



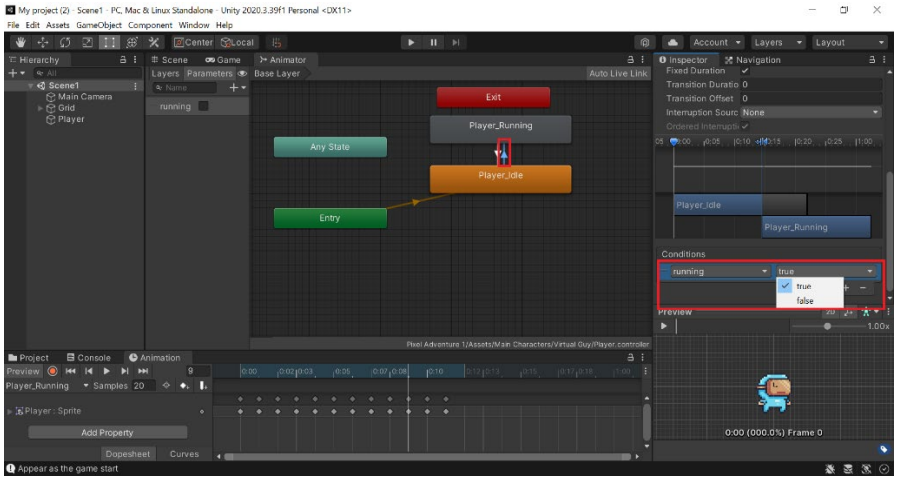
k. Name the parameter created as running



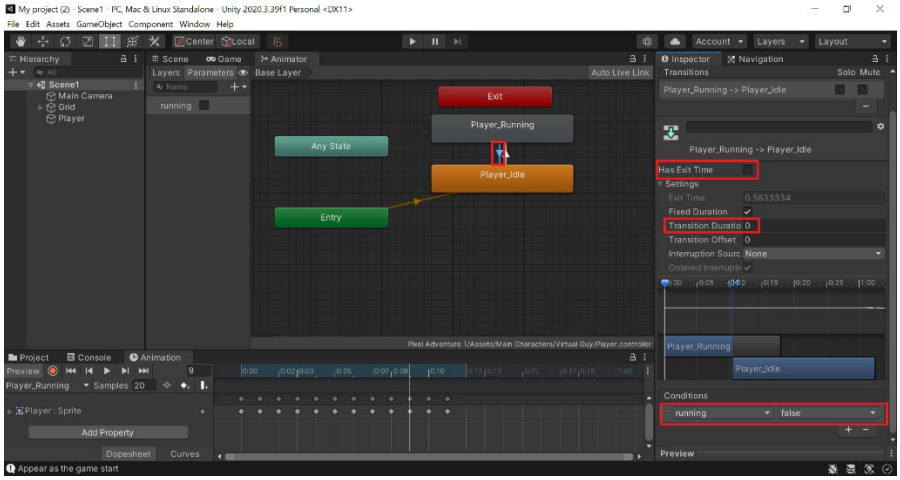
l. Click on one of the arrow from Player_Idle state to Player_Running state then untick has exit time and set the Transition Duration to 0 in the inspector tab



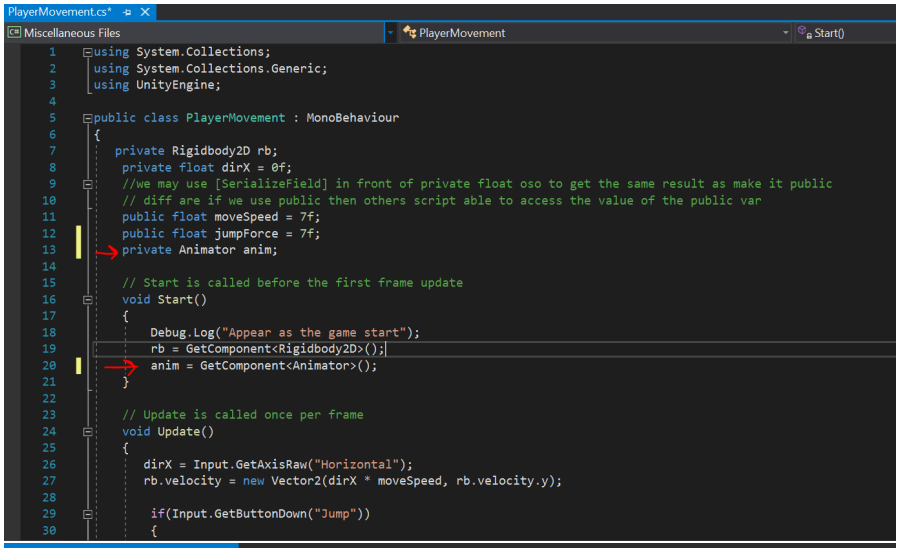
m. Next, go to Condition at the bottom of the inspector and click + sign and set the condition of the parameter to true as shown below
Note: Make sure the condition for the parameters is for Idle to Running

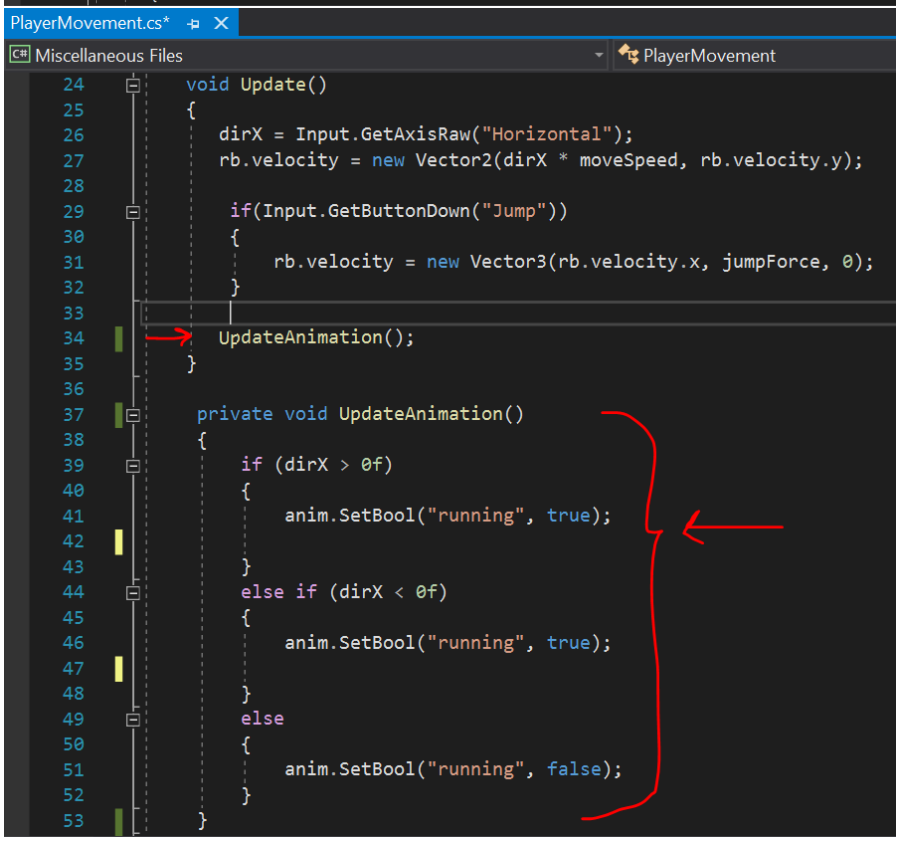


n. Repeat step (m) for Player_Running state to Player_Idle state arrow and set the running condition to false



o. Next add the following code to PlayerMovement script in order to make the Player_Idle state to Player_Running state transitioning automatically from one to another





p. Add the following code to the PlayerMovement script in order to make the Player able to turn right or turn left

```
private Rigidbody2D rb;
private float dirX = 0f;
//we may use [SerializeField] in front of private float oso to get the same result as make it public
// diff are if we use public then others script able to access the value of the public var
public float moveSpeed = 7f;
public float jumpForce = 7f;

private Animator anim;
private SpriteRenderer sprite;
```

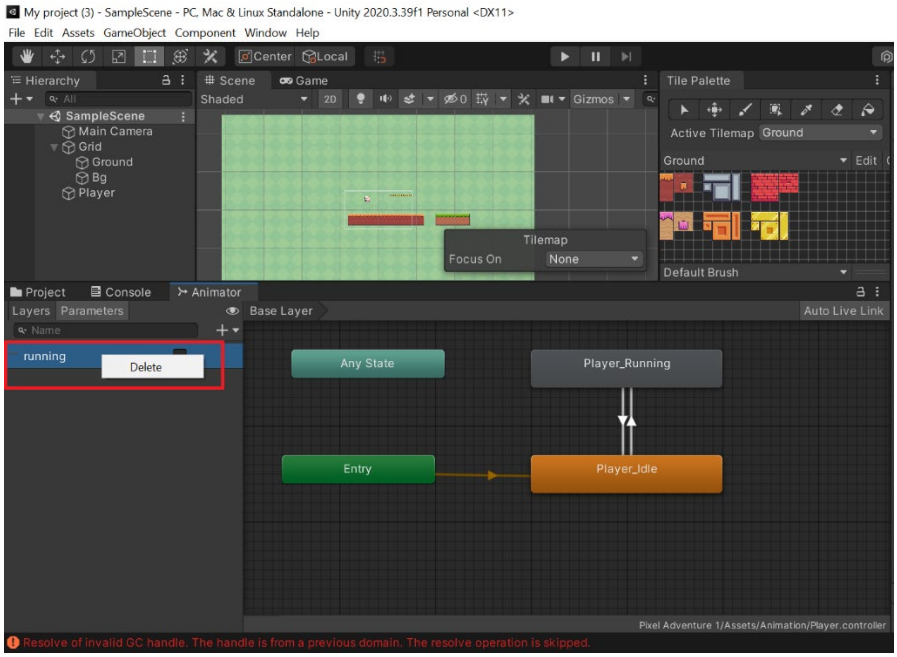
```
void Start()
{
    Debug.Log("Appear as the game start");
    rb = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    sprite = GetComponent<SpriteRenderer>();
}
```

```
private void UpdateAnimation()
{
    if (dirX > 0f)
    {
        anim.SetBool("running", true);
        sprite.flipX = false;
    }
    else if (dirX < 0f)
    {
        anim.SetBool("running", true);
        sprite.flipX = true;
    }
    else
    {
        anim.SetBool("running", false);
    }
}
```

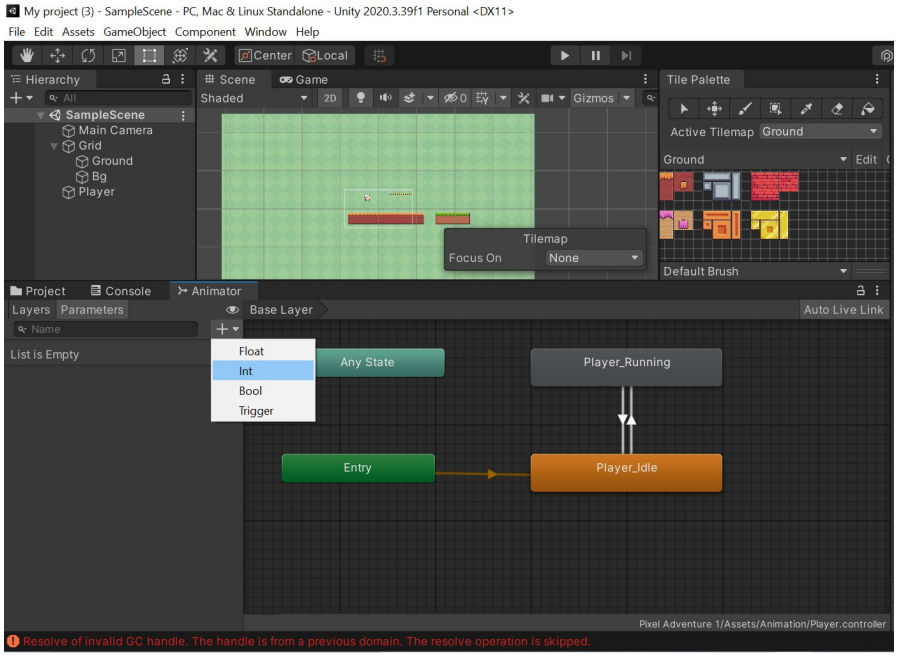
q. Play and observe the result

r. In order to maximize the use of transition state, create a new variable name state that contain all the transition state available

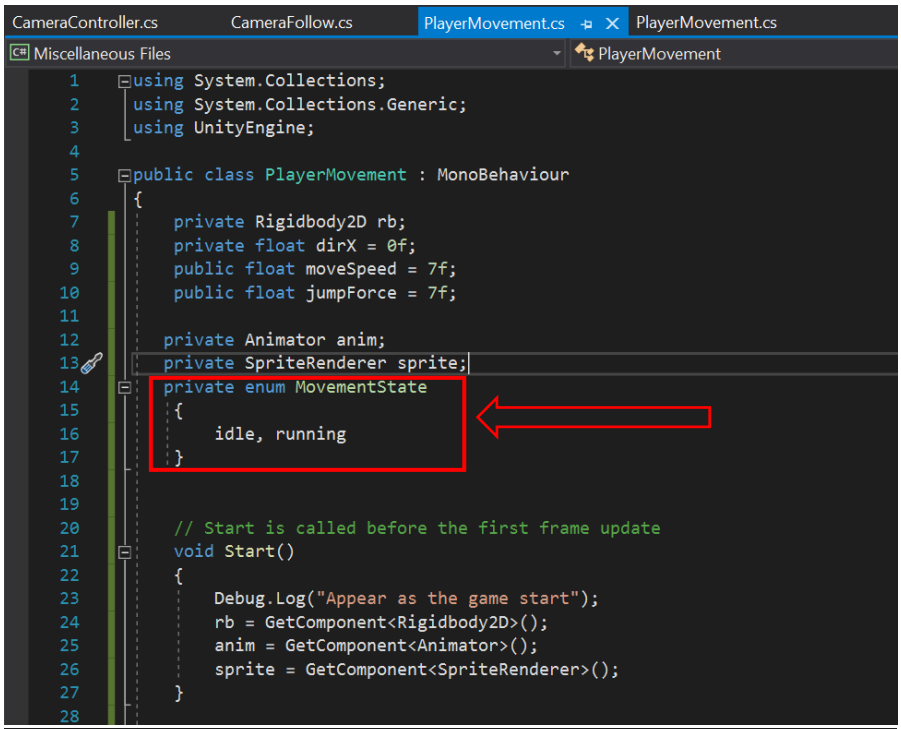
s. Delete the running parameters from the animator controller tab

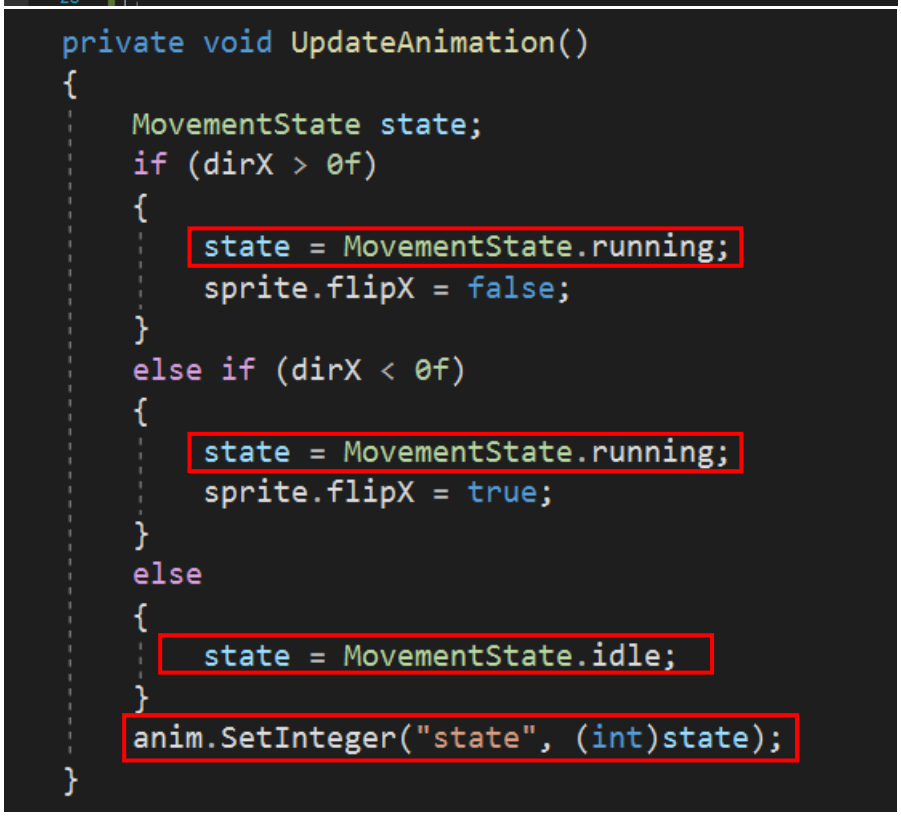


t. Click on + sign at the Parameters tab to add a new parameter name state



u. Next, modify the animation code to add multiple transition state by creating collection of related constants (idle and running state)





v. Play and observe the result

Topic 4.0 : Practical Work (Multiple Animation)

PRACTICAL WORK

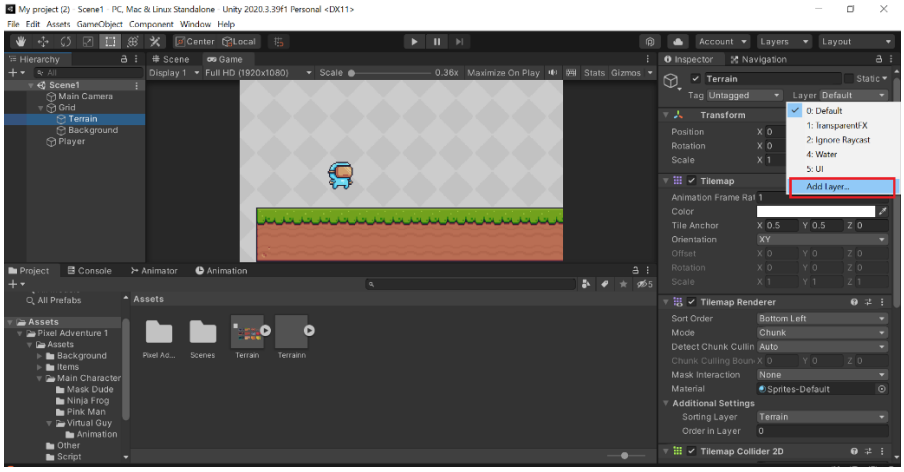
1. Using the unity package provided, create animation for Player jumping and falling state
   ……
2. Modify the PlayerMovement script to active the jumping transition state for the Player
   ……
3. Modify the PlayerMovement script to active the falling transition state for the Player
   ……
4. Paste the the script and the outcome below

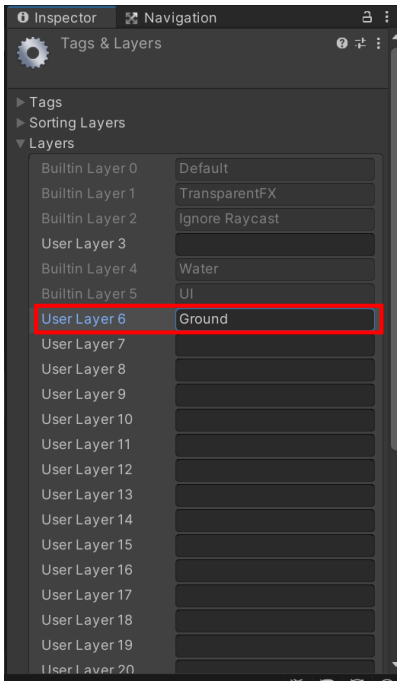
Topic 5.0 : Ground Checking, Enemy AI & Death

A) Set ground checking for the player

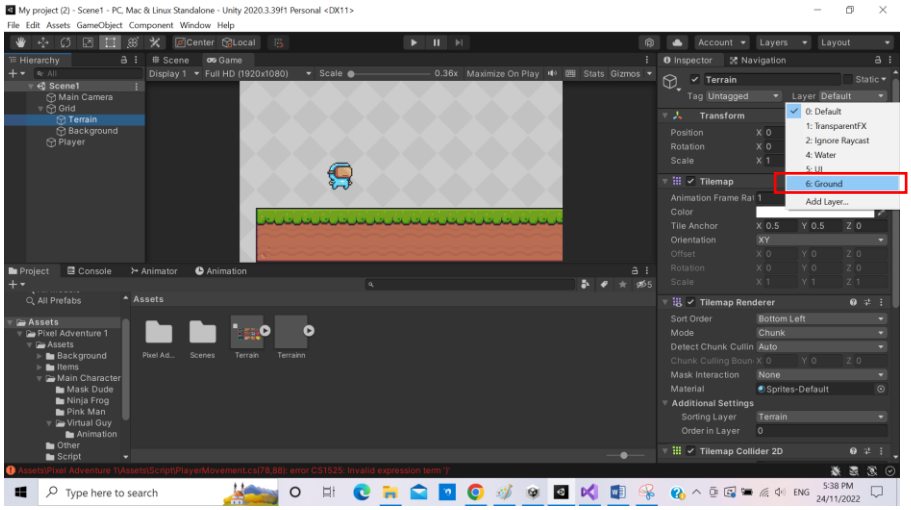
1. Create IsGround method to check if the player collides with the ground or not

a. Click on Terrain Tilemap and declare a new layer name Ground





b. Then go back to Terrain and click on Ground as Layer in the inspector tab



c. Modify the PlayerMovement script by adding the following code

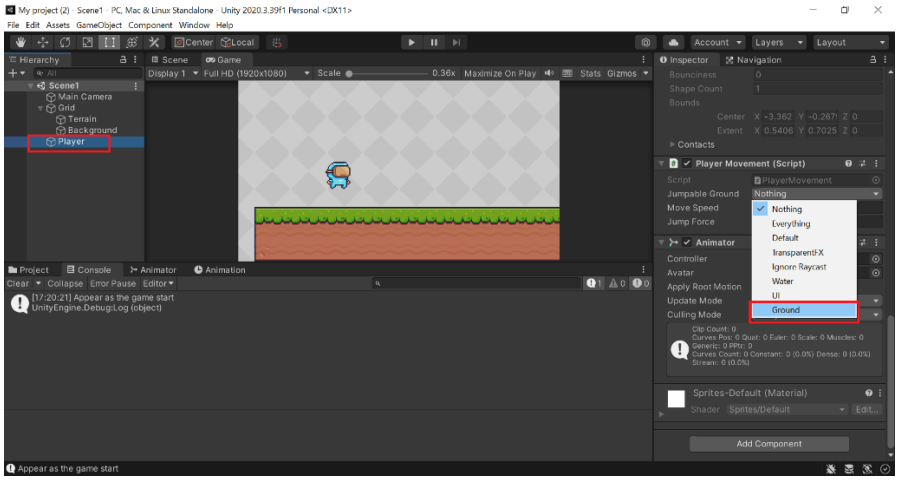
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    private BoxCollider2D coll;
    [SerializeField] private LayerMask jumpableGround;

    private Rigidbody2D rb;
    private float dirX = 0f;
    //we may use [SerializeField] in front of private float oso to get the same result as make it public
    // diff are if we use public then others script able to access the value of the public var
    public float moveSpeed = 7f;
    public float jumpForce = 7f;

    private Animator anim;
    private SpriteRenderer sprite;
    private enum MovementState
    {
        idle, running, jumping, falling
    }
```

```
// Update is called once per frame
void Update()
{
    dirX = Input.GetAxisRaw("Horizontal");
    rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);

    if(Input.GetButtonDown("Jump") && IsGrounded())
    {
        rb.velocity = new Vector3(rb.velocity.x, jumpForce, 0);
    }

    UpdateAnimation();
}
```

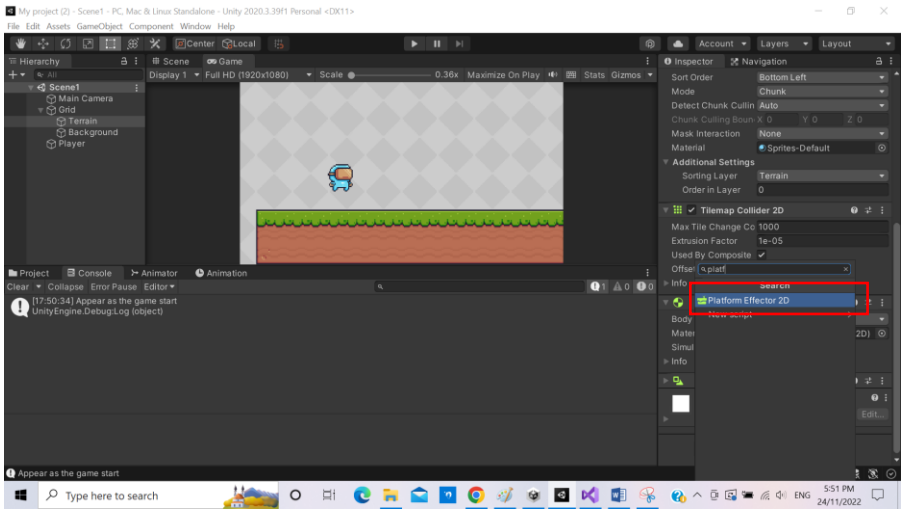
d. Next, add a new method to add a BoxCast2D to the player by adding the following code after UpdateAnimation method

```
private bool IsGrounded()
{
    return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f, Vector2.down, .1f, jumpableGround);
}
```

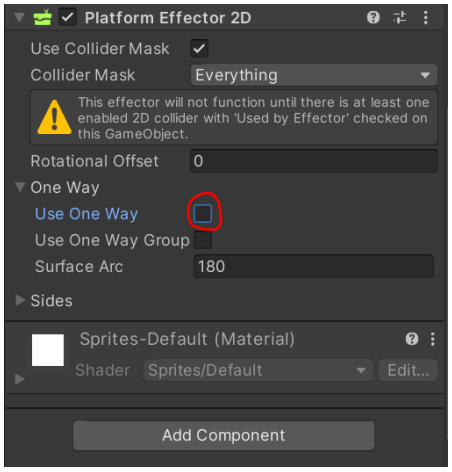
e. Then click on Player game object and click on Ground at the Jumpable Ground in the inspector tab under Player Movement script



f. Next, click on Terrain and add Platform Effector 2D by click on add component in the inspector



g. Next, untick Use One Way at the Platform Effector 2D component



B) Collect & Count Items

1. Create a game object for the player to collect and gain point

a. Create a 2D game object and rename as Cherry

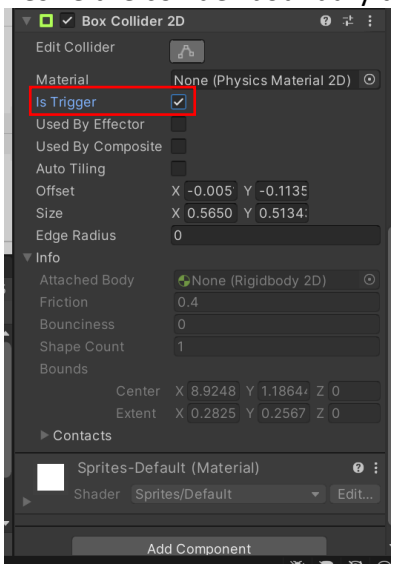
b. Change the sprite of the Cherry game object to cherry image from Fruits folder
Notes: use the first image of the sprite collector

c. Create animation for the cherry and rename as Cherry_Anim

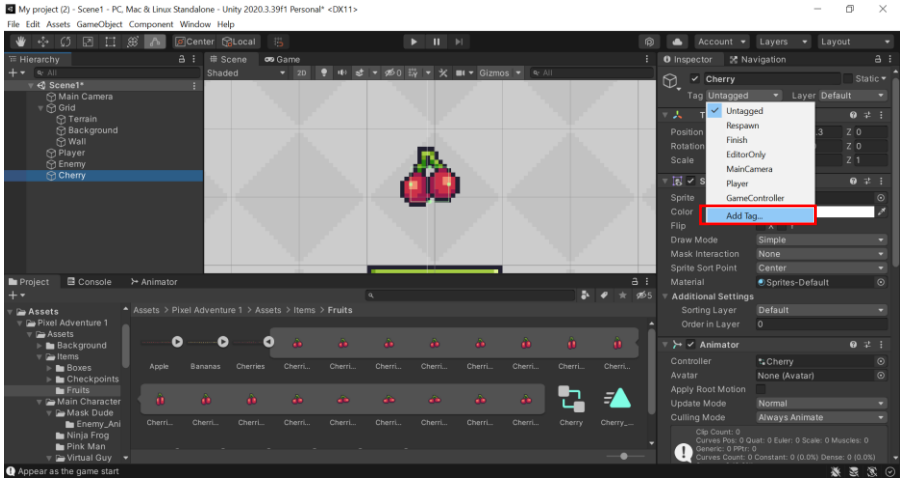
d. Drag all the images of the cherry in the animation tab and set the sample rate to 30

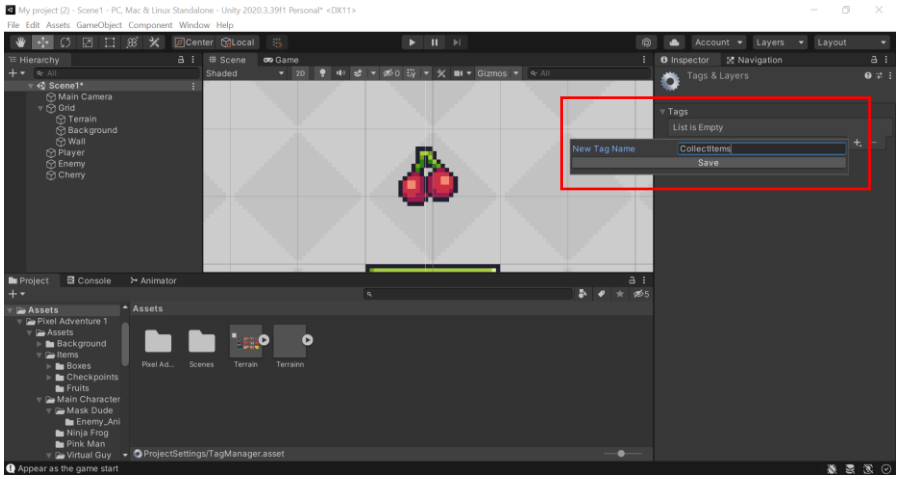
e. Next add Box Collider 2D to the Cherry game object

f. Resize the collider boundary to fit the cherry sprite and tick Is Trigger

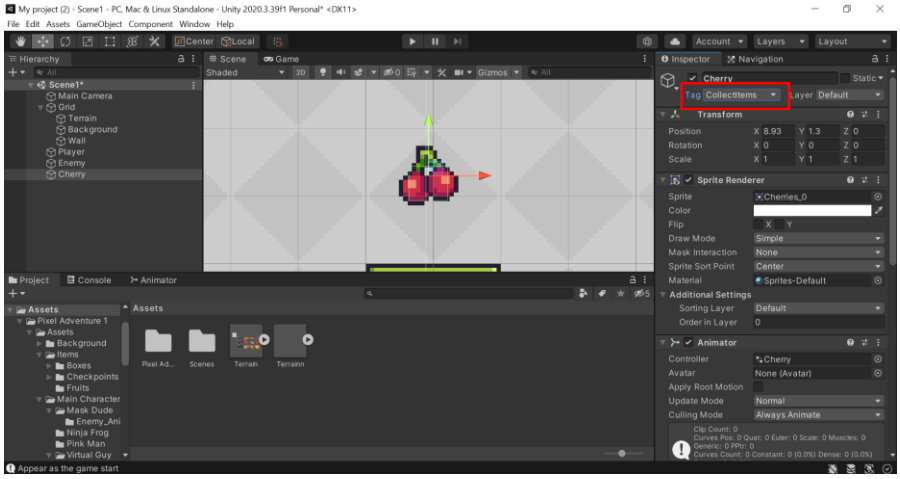


g. Next create a tag name CollectItems in the Cherry Inspector tab





h. Then change the tag of the Cherry game object to the created tag earlier



i. Next create a C# script called ItemCollection and attach to player

j. Add the following lines of codes in the ItemCollection script
Tips: Make sure the tag name is correct

PlayerMovement.cs | ItemCollection.cs

Miscellaneous Files | ItemCollection

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

public class ItemCollection : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
    //may use OnCollision if use other than box collider
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("CollectItems"))
        {
            Destroy(collision.gameObject);
            Debug.Log("Dah makan cherry");

        }
    }
}
```

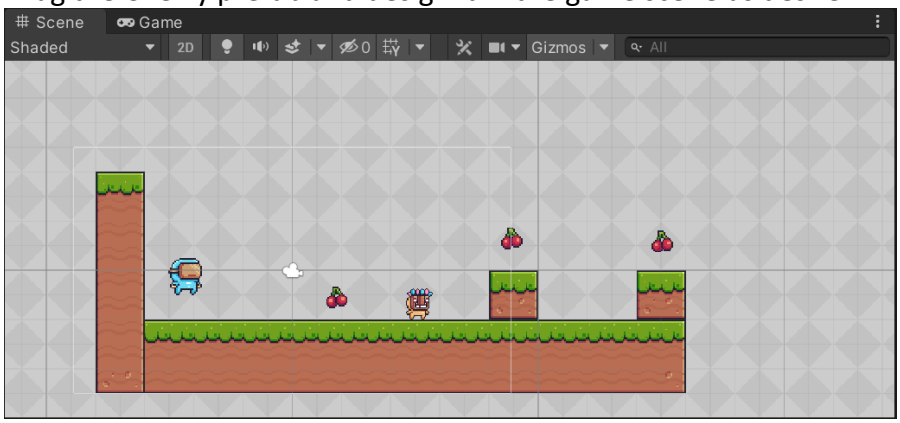
k. Play the game and observe the cherry will disappear when the player collides with the cherry

l. Next to make sure the cherry adds a points for the player add the following codes in the ItemCollection script

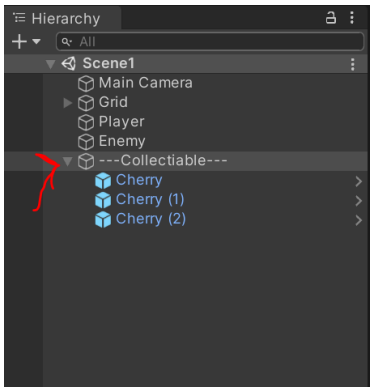
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ItemCollection : MonoBehaviour
{
    private int cherries = 0;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
    //may use OnCollision if use other than box collider
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("CollectItems"))
        {
            Destroy(collision.gameObject);
            cherries++;
            Debug.Log("Dah makan cherry = " + cherries);

        }
    }
}
```

m. Next create a Prefab folder and drag Cherry game object in the prefab folder
Tips: This step is used to create a prefab

n. Drag the Cherry prefab and design it in the game scene as desire



o. Organize all the prefab in the hierarchy by creating an empty game object and rename as Collectable



p. Drag all the cherry prefab under Collectable

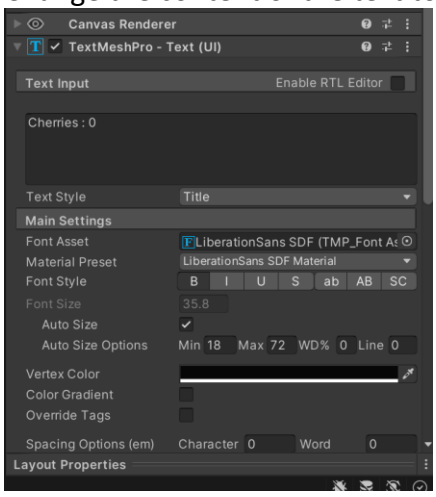
q. Play the game and observe the result

2. Create counter placeholder in the game scene

a. Right click on the hierarchy and click UI > Text - TextMeshPro. Rename to Collect_Item
Notes: You may use regular text instead of TextMeshPro

b. Positioning the text on the top left of the game

c. Change the content of the text to Cherries : 0



d. Open the ItemCollection script and add the following codes

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

```
private int cherries = 0;
[SerializeField] private TMP_Text collect;
// Start is called before the first frame update
void Start()
{

}

// Update is called once per frame
void Update()
{

}
//may use OnCollision if use other than box collider
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("CollectItems"))
    {
        Destroy(collision.gameObject);
        cherries++;
        Debug.Log("Dah makan cherry = " + cherries);
        collect.text = "Cherries : " + cherries;

    }
}
```

e. Done and play the game

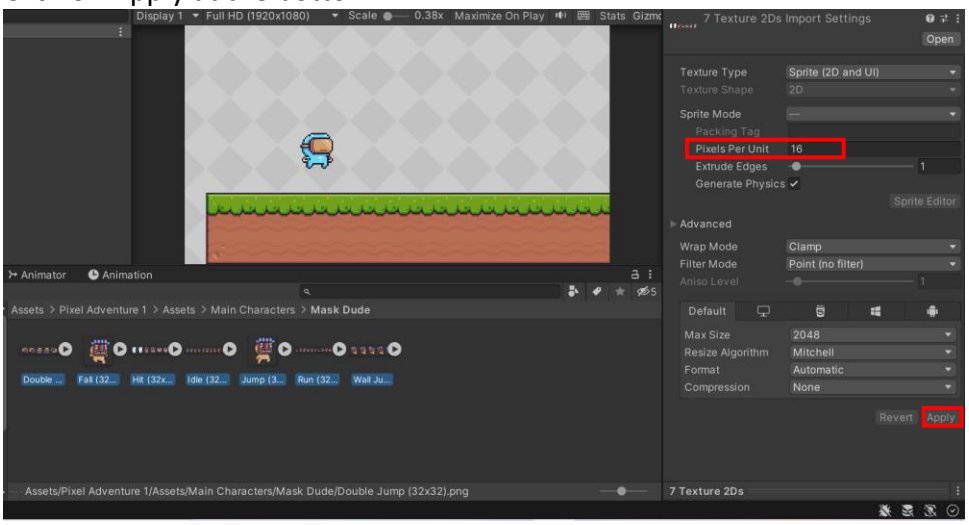
C) Enemy AI

1. Create a game object name enemy in the hierarchy

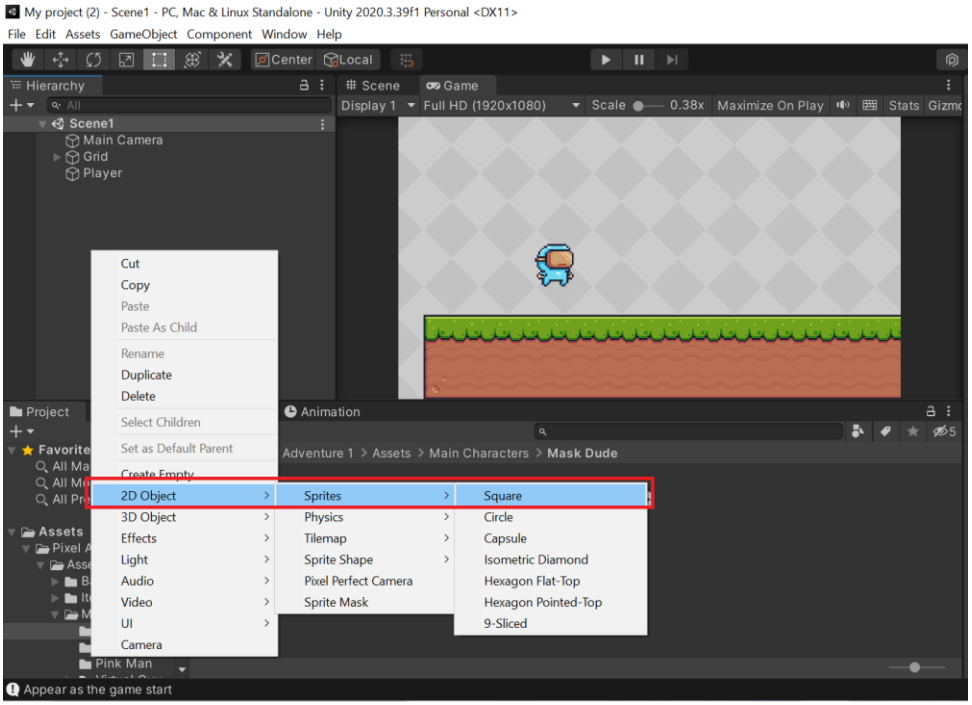
a. Navigate to Assets > Pixel Adventure 1 > Assets > Main Characters > Mask Dude

b. Shift + A all the sprite and change the pixels per unit to 22 in the inspector

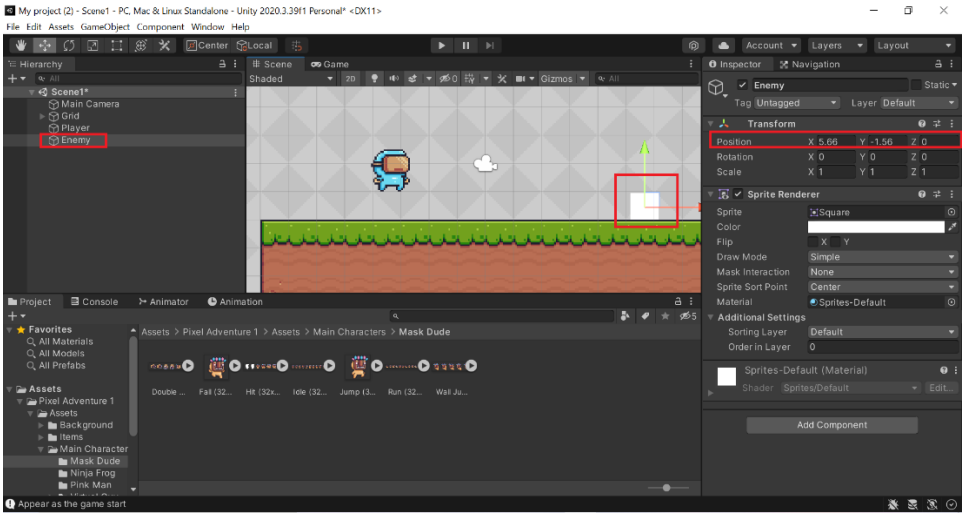
c. Click on Apply at the bottom



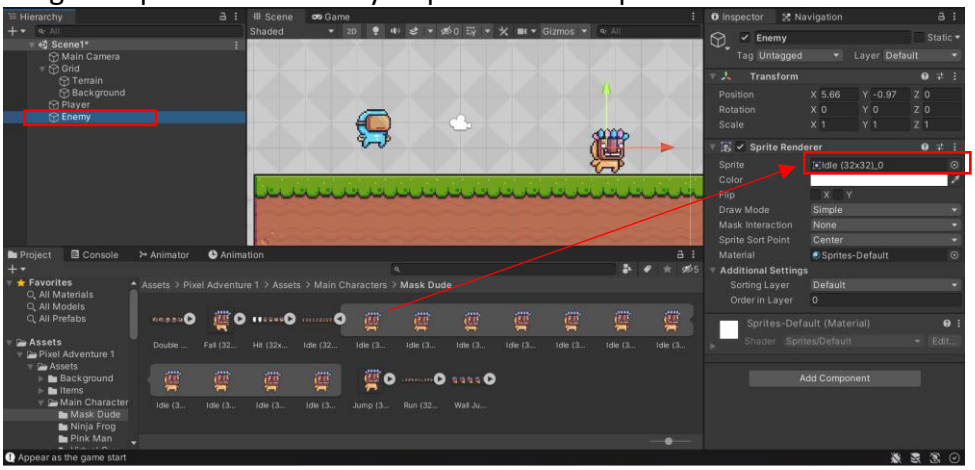
d. Right click in hierarchy and create 2D object name enemy as shown below



e. Move the enemy to any desire position



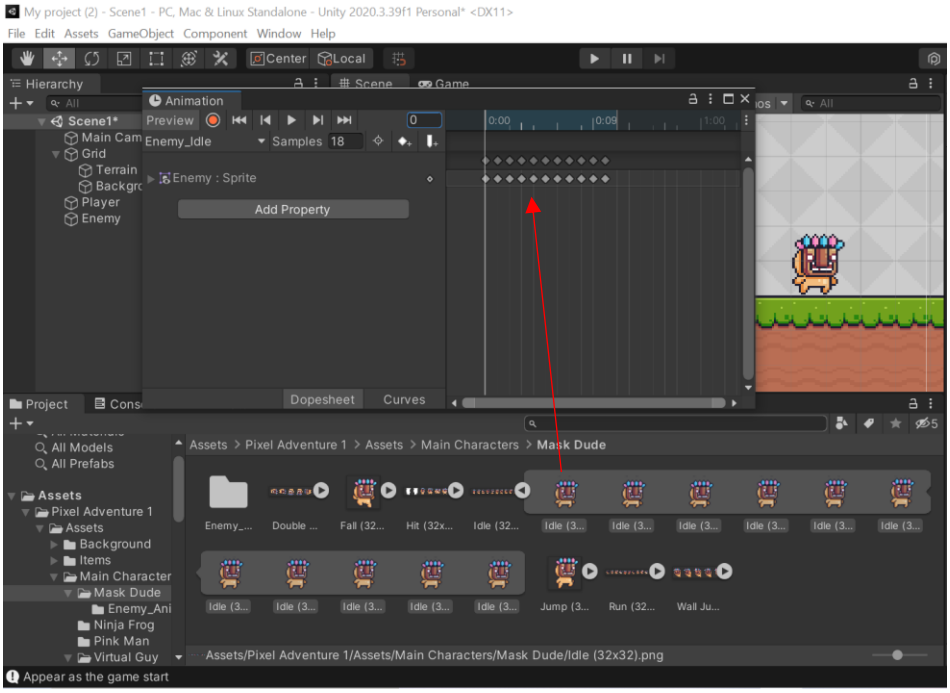
f. Drag Idle sprite to the enemy inspector under Sprite Renderer



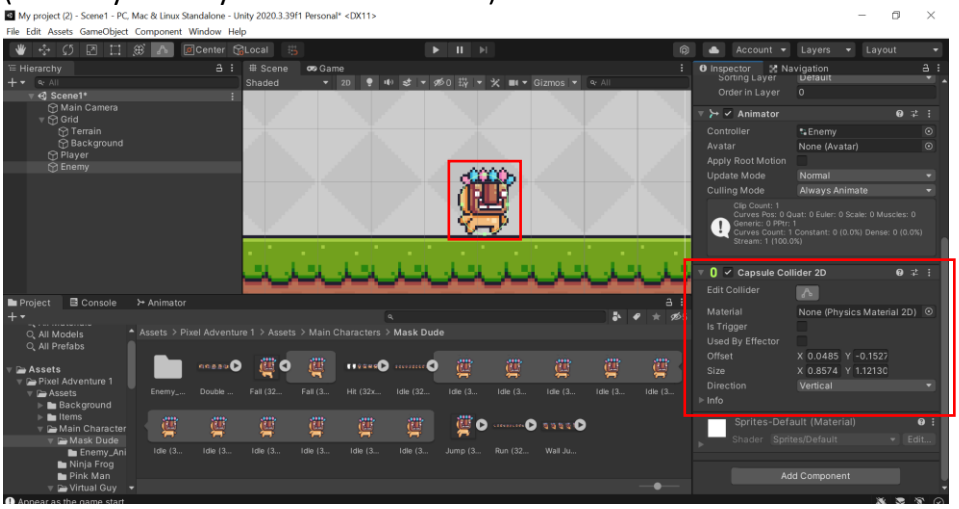
g. Create the animation for the enemy and rename to Enemy_Idle

h. Drag the animation in the player to create the Animator Controller

i. Drag all the idle sprite to the animation tab and change the sample rate to 18 or any suitable sample (You may follow Labsheet 3: Animation)



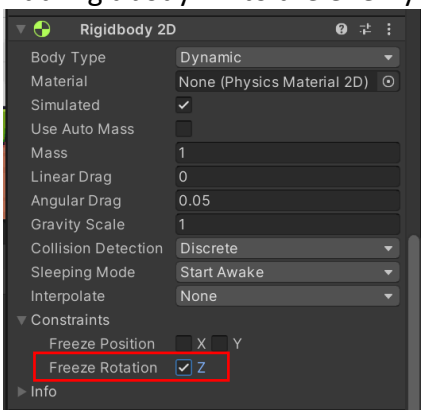
j. Add collider for the enemy by click on Add Component at the Inspector tab (You may use any suitable 2D collider)



k. Resize the collider use to make sure the collider border fit nicely to the enemy

l. Play the game and observe if the player able to jump over the enemy
Tips: Change the jump force if the player unable to jump over the enemy

m. Add Rigidbody 2D to the enemy and tick Freeze Z under Constraints

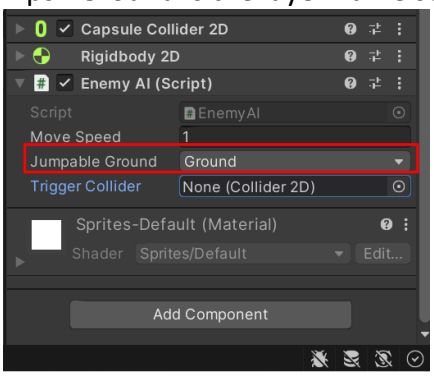


n. Create new C# script name EnemyAI and attach to the enemy in the hierarchy

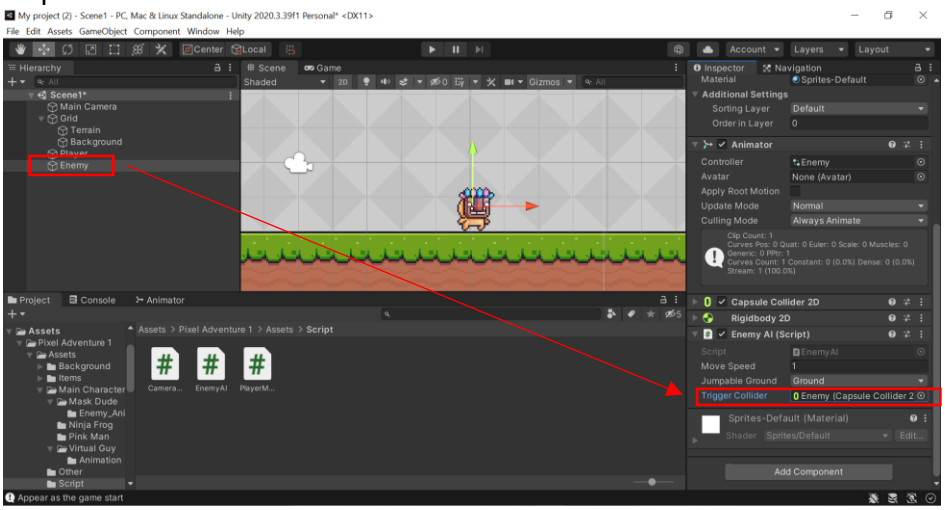
o. Double click to open the script and add the following codes

```
public class EnemyAI : MonoBehaviour
{
[SerializeField] private float moveSpeed = 1f;
[SerializeField] private LayerMask jumpableGround;
private Rigidbody2D rb;
[SerializeField] private Collider2D triggerCollider;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        rb.velocity = new Vector2(moveSpeed, rb.velocity.y);
    }

    void FixedUpdate()
    {
        if (!triggerCollider.IsTouchingLayers(jumpableGround))
        {
            Flip();
        }
    }

    private void Flip()
    {
        transform.localScale = new Vector2(transform.localScale.x * -1, transform.localScale.y);
        moveSpeed *= -1;
    }
}
```

p. Click on enemy in the hierarchy and change the Jumpable Ground to Ground
Tips : Ground is the layer name set for Terrain tilemap



q. Drag the Enemy game object in the hierarchy to the Trigger Collider in the Inspector tab

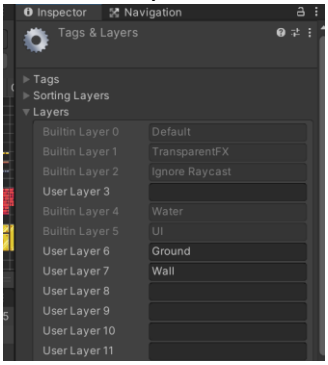


r. Play the game and observe the enemy movement

s. Add a new tilemap and rename as Wall

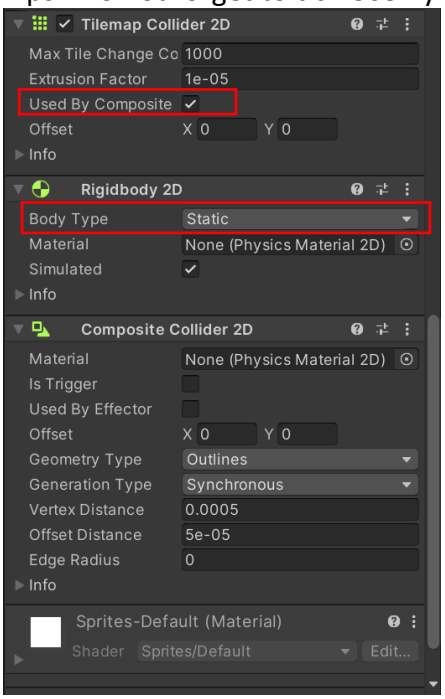
t. Create the tile palette for the Wall and design the tilemap as desire

u. Add new layer and rename to Wall



v. Click on Layer in the Wall inspector tab and choose Wall as the name of the layer

w. Click on Add Component in the Wall inspector tab and add Tilemap Collider 2D and Composite Collider 2D to the Wall
Tips : Do not forget to tick Use By Composite under Tilemap Collider 2D



x. Change the Body Type (Rigidbody 2D) to Static to disable gravity scale for the Wall game object as shown above

y. Double click EnemyAI script and add the following code

```
public class EnemyAI : MonoBehaviour
{
[SerializeField] private float moveSpeed = 1f;
[SerializeField] private LayerMask jumpableGround;
[SerializeField] private LayerMask wall;
private Rigidbody2D rb;
[SerializeField] private Collider2D triggerCollider;
```

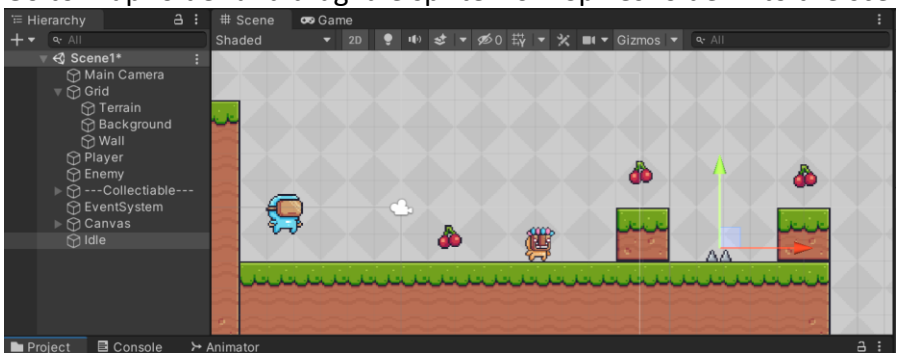
```
void FixedUpdate()
{
    if (!triggerCollider.IsTouchingLayers(jumpableGround) || triggerCollider.IsTouchingLayers(wall))
    {
        Flip();
    }
}
```

z. Play the game and observe the result

D) Player Death

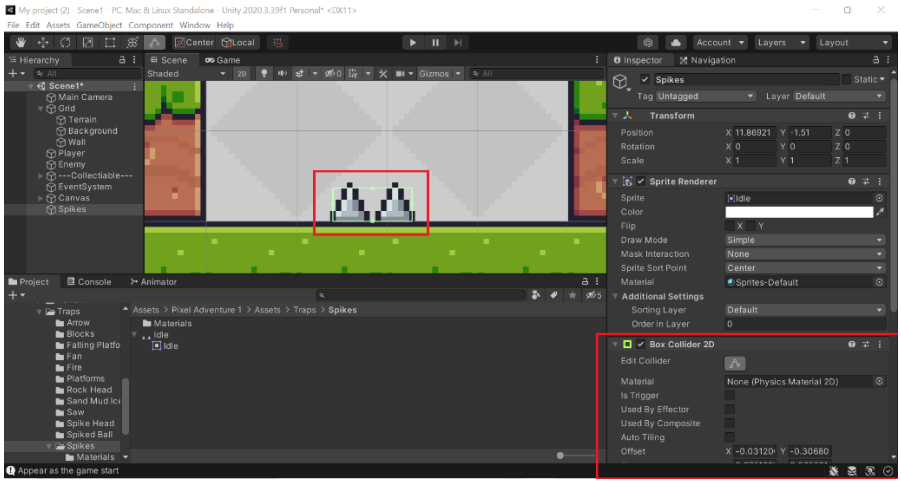
1. Create an obstacle

a. Go to Trap folder and drag Idle sprite from Spikes folder into the scene



b. Position the spikes as desire

c. Add a Box Collider 2D to the spikes and resize the collider boundary as desire

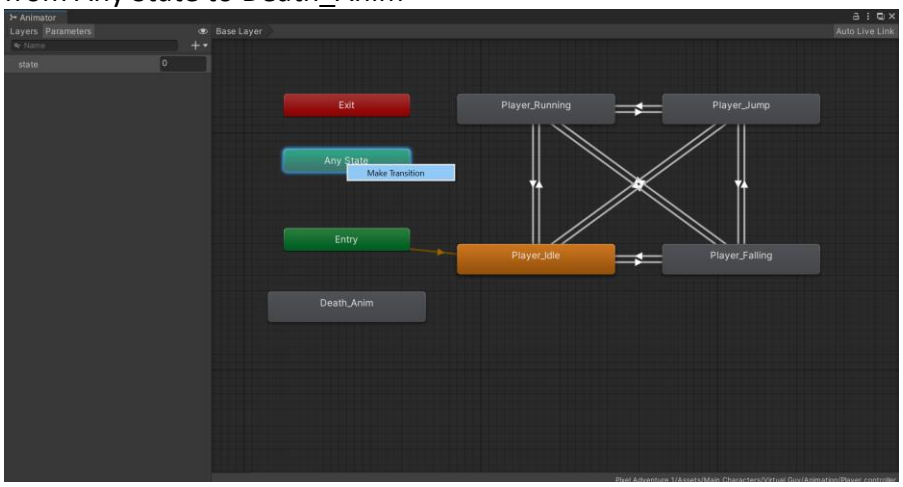


d. Add a new tag called Trap to the Spikes game object

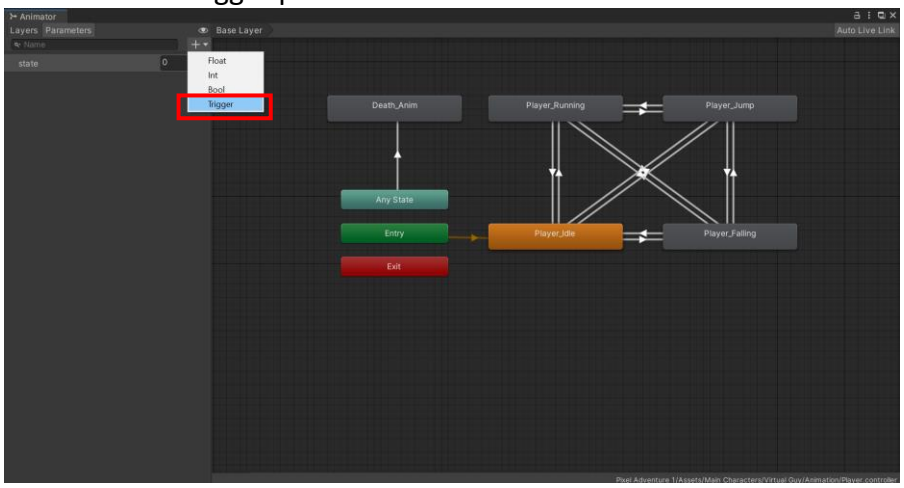
e. Create animation called Death_Anim to the Player game object in order to make the player disappear when the player collide with any obstacles
Notes: use Appearing & Disappearing sprite in Main Character folder

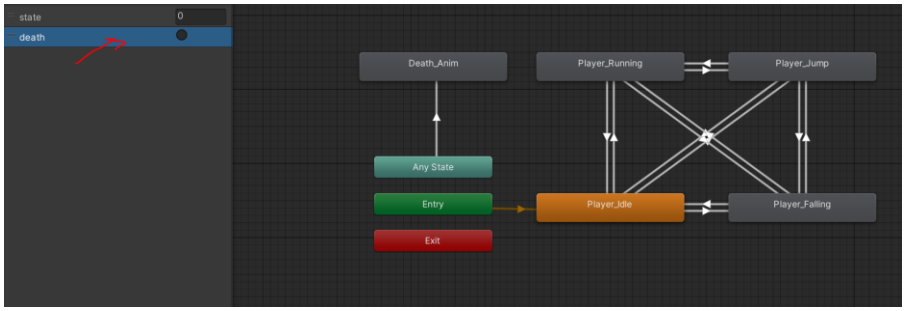
f. Drag the disappearing sprites collection into the Death_Anim and change the sample rate to 18
Notes: Do not forget to change the pixel per unit for Disappearing & Appearing sprite to 16

g. Next open the animator controller for the player and make transition from Any State to Death_Anim

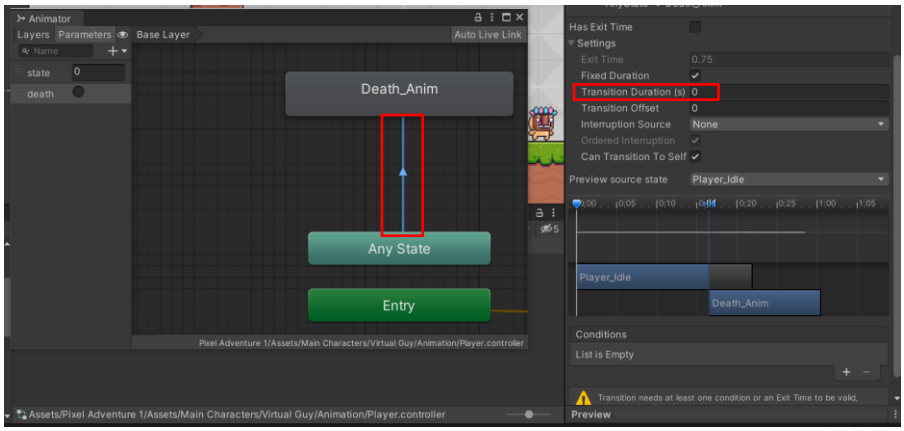


h. Next create a trigger parameter called death in the animator tab

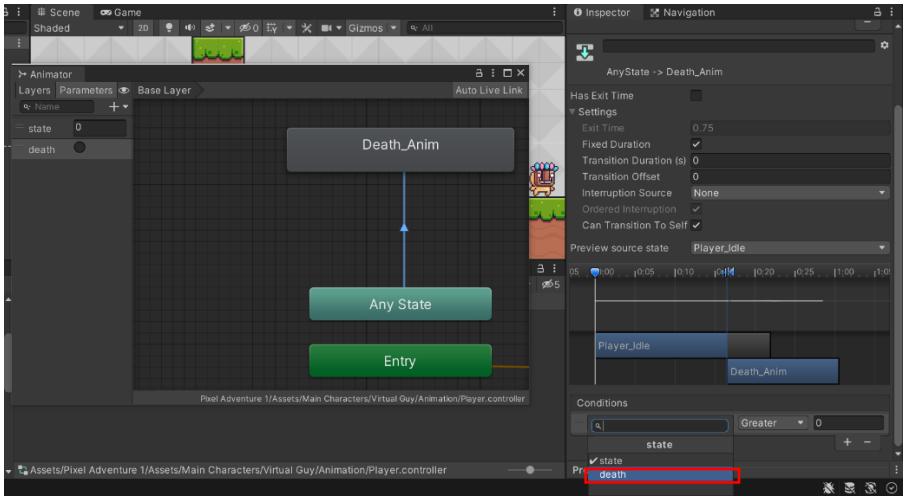


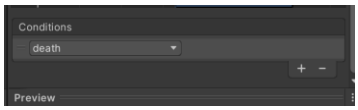


i. Then click on the transition arrow from Any State to Death_Anim and set the transition duration to 0



j. Next set the Conditions at the bottom and add death





k. Next create a new C# script name Player_Life and add the following codes

```
Player_Life.cs*
Miscellaneous Files                                   Player_Life
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Player_Life : MonoBehaviour
6   {
7       private Rigidbody2D rb;
8       private Animator anim;
9       // Start is called before the first frame update
10      void Start()
11      {
12          anim = GetComponent<Animator>();
13          rb = GetComponent<Rigidbody2D>();
14      }
15      // Update is called once per frame
16      void Update()
17      {
18
19      }
20      private void OnCollisionEnter2D(Collision2D collision)
21      {
22          if(collision.gameObject.CompareTag("Trap"))
23          {
24              Die();
25          }
26      }
27      private void Die()
28      {
29          rb.bodyType = RigidbodyType2D.Static;
30          anim.SetTrigger("death");
31      }
32  }
33
```

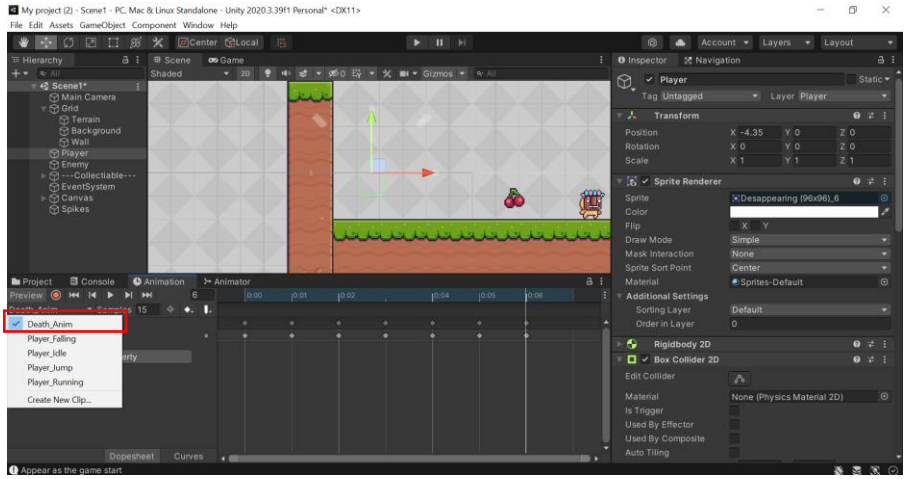
l. Play and observe the result
Notes: The player will disappear when hit the spikes and the body type change to static (no movement)

2. Restart level once the player died

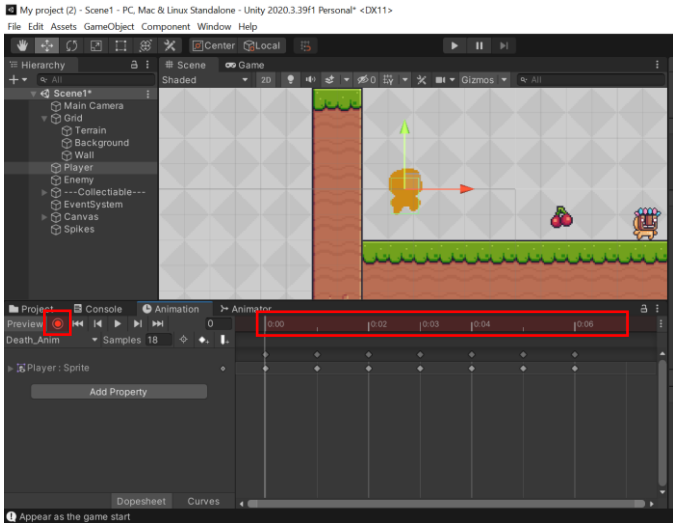
a. Add the following codes to Player_Life script in order to reload player to the current scene

```
Player_Life.cs*
Miscellaneous Files                                   Player_Life
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
6   public class Player_Life : MonoBehaviour
7   {
8       private Rigidbody2D rb;
9       private Animator anim;
10      void Start()
11      {
12          anim = GetComponent<Animator>();
13          rb = GetComponent<Rigidbody2D>();
14      }
15      void Update()
16      {
17
18      }
19      private void OnCollisionEnter2D(Collision2D collision)
20      {
21          if(collision.gameObject.CompareTag("Trap"))
22          {
23              Die();
24          }
25      }
26      private void Die()
27      {
28          rb.bodyType = RigidbodyType2D.Static;
29          anim.SetTrigger("death");
30      }
31      private void Restart()
32      {
33          SceneManager.LoadScene(SceneManager.GetActiveScene().name);
34      }
35  }
```

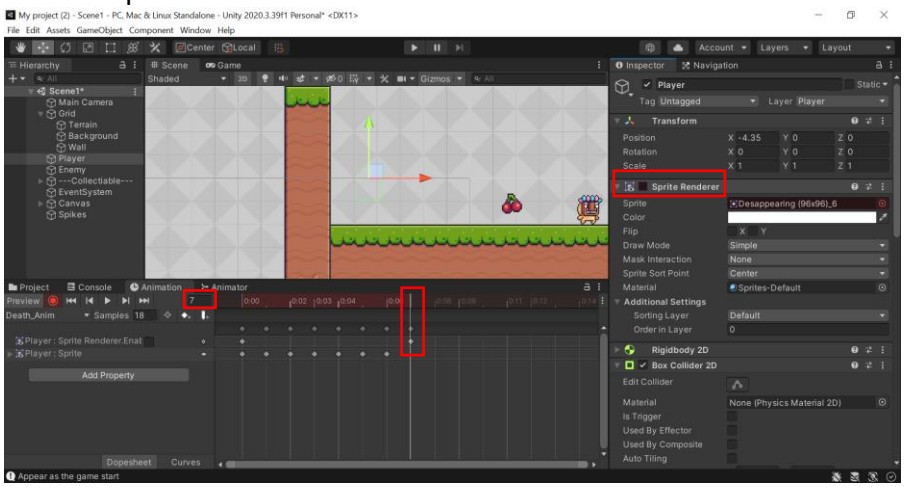
b. Click on Player animation tab and choose Death_Anim



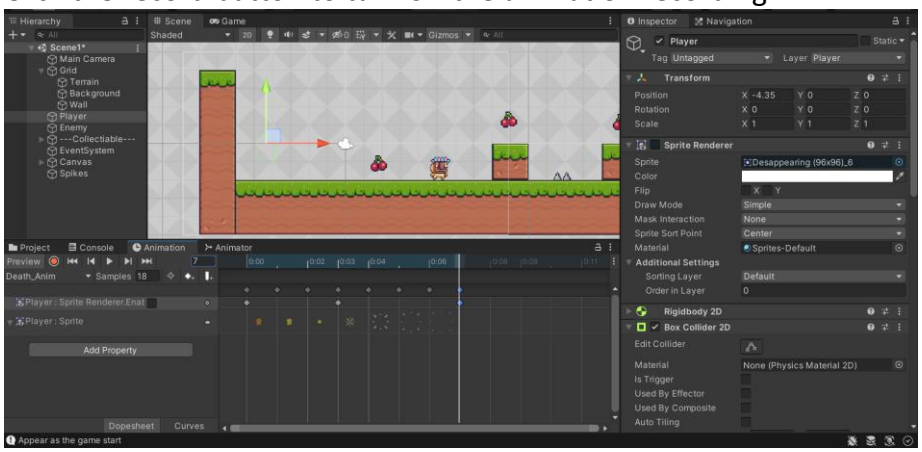
c. Click on Record button to add keyframe



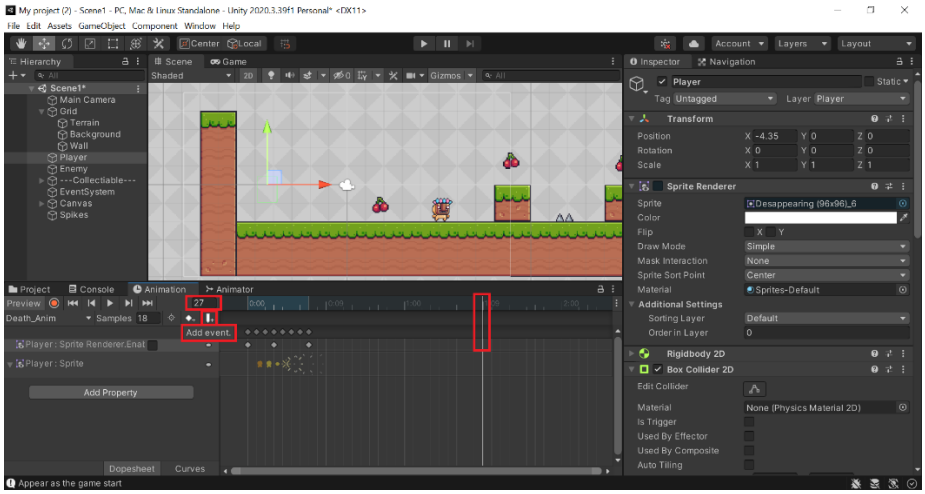
d. Go to the last keyframe (0:07) and turn off the Sprite Renderer by untick at the inspector tab



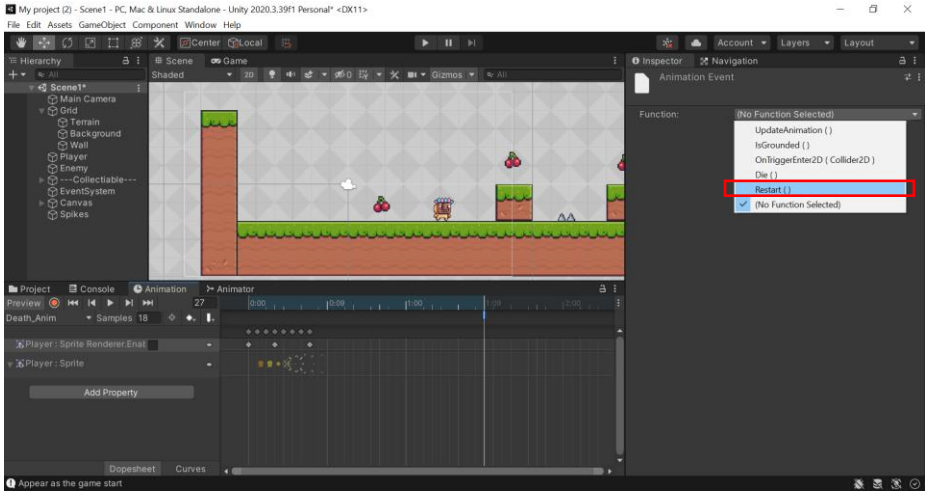
e. Click the record button to turn off the animation recording



f. Next to call the Restart() method, click on add animation event at keyframe 27 as shown below



g. Then choose Restart() at Function: as shown below



h. Change the tag of all obstacles and enemy in the game to Trap

i. Play the game

Appendix

References & Resources

REFERENCES

Franz, L. (2020). 2D Game Development with Unity 1st Edition. CRC Press. (ASIN: B08L5L7TC3)

Halpern, J. (2019). Developing 2D games with Unity: independent game programming with C#. NY, NY: Apress. (ISBN: 13:978-1484237717)

Jarred, C. &. Louis, S. (2020). MonoGame Mastery: Build a Multi Platform 2D Game and Reusable Game Engine 1st ed. Apress. (ISBN: 1484263081)

Felicia, P. (2020). A Beginner's Guide to 2D Platform Games with Unity: Create a Simple 2D Platform Game and Learn to Code in the Process. (n.p.): Independently Published. (ISBN : 979-8555335074)

Jackson, S. (n.d.). Mastering Unity 2D Game Development: Become an Expert in Unity3D's New 2D System, and Then Join in the Adventure to Build an RPG Game Framework!. United Kingdom: Packt Publishing Limited

Special Resources to Deliver The Course :

- https://tinyurl.com/2DGameeBookResources. Resources

JUMP INTO 2D PLATFORM GAME CREATION

e ISBN 978-967-0074-20-7

POLITEKNIK METro KUALA LUMPUR
(online)