Term ID: 220212 Subject Code: INT209

CA – 2

NAME: ALAN JAMES

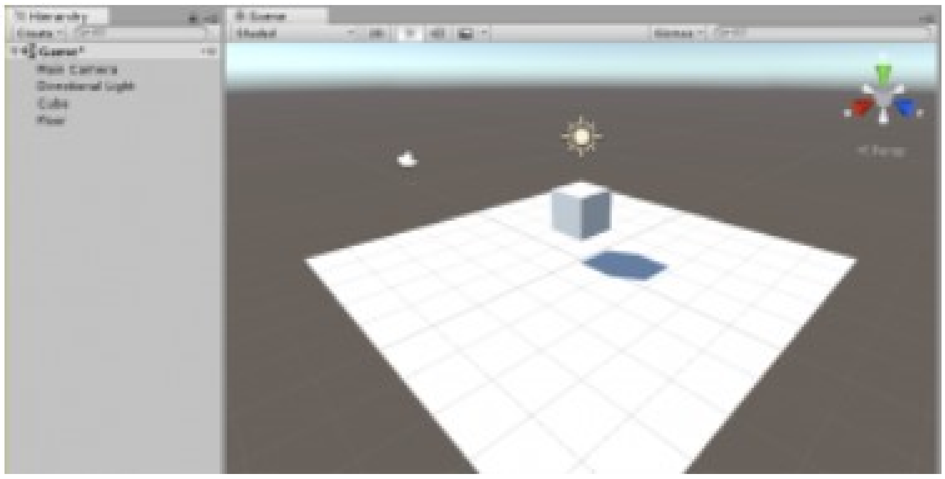
REG NO: 11903972

SECTION: K19KH

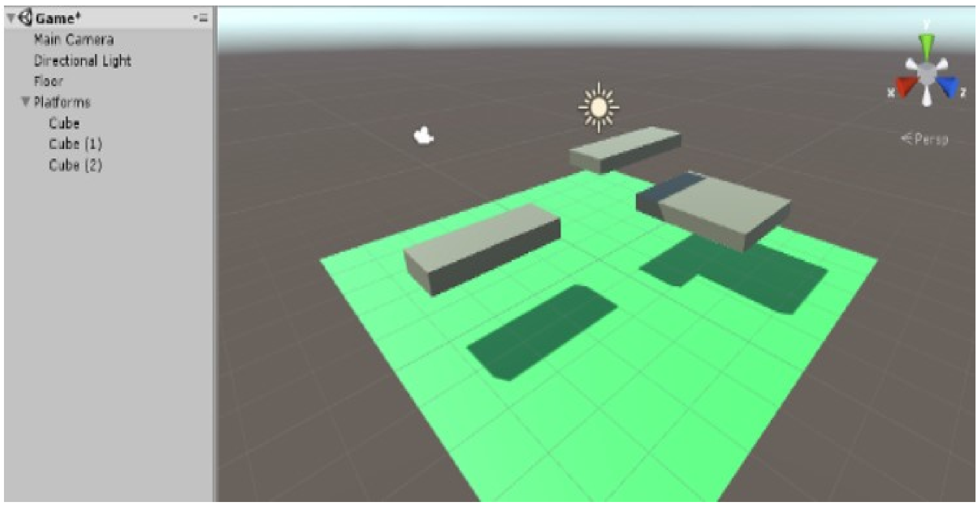
ROLL NO: A08

SUBMITTED TO: MR. SUDHANS SHEKAR PANDEY

Step 1: In the Hierarchy Window, right click and select 3D Object – Cube. Click on the cube and change the position values. Let’s go ahead and create the floor of the game. For that, we will use a plane. Right click on your scene in the Hierarchy Window and select 3D Object – Plane. This will bring up a plane into our scene. From the Hierarchy Window, we can rename this object by right clicking – Rename, by selecting it and pressing F2, or by single-clicking on it after we’ve selected it. Call it “Floor”.

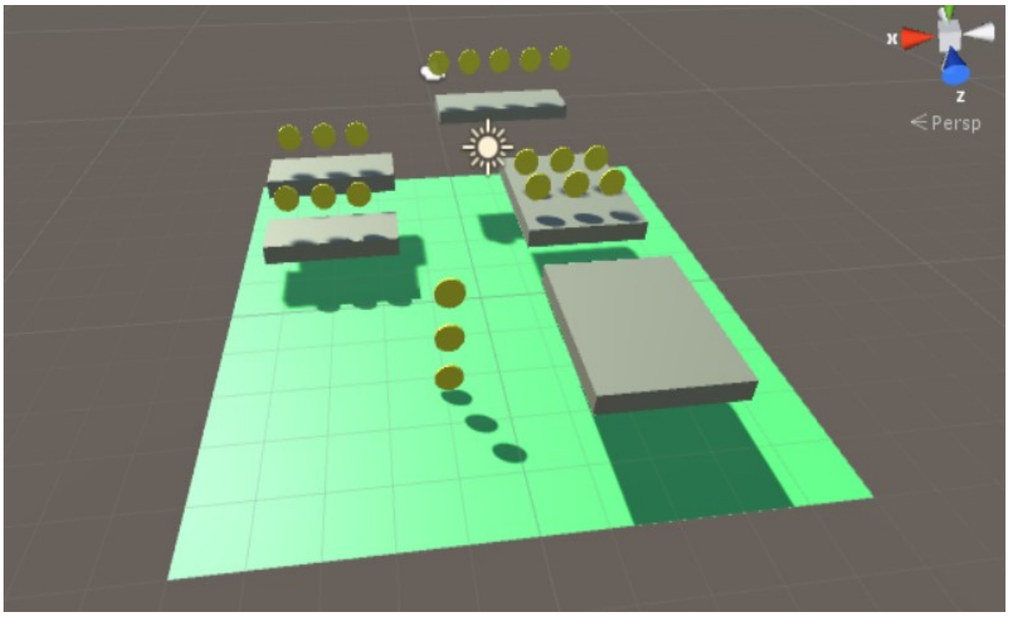


Step 2: Make 2-3 cubes and scale it to different sizes. Also change the color of the Floor. In the Hierarchy Window right click and select Create Empty. Rename this new object to “Platforms”. Drag and drop all the platforms you created into this object. Notice that even though Platforms is empty.

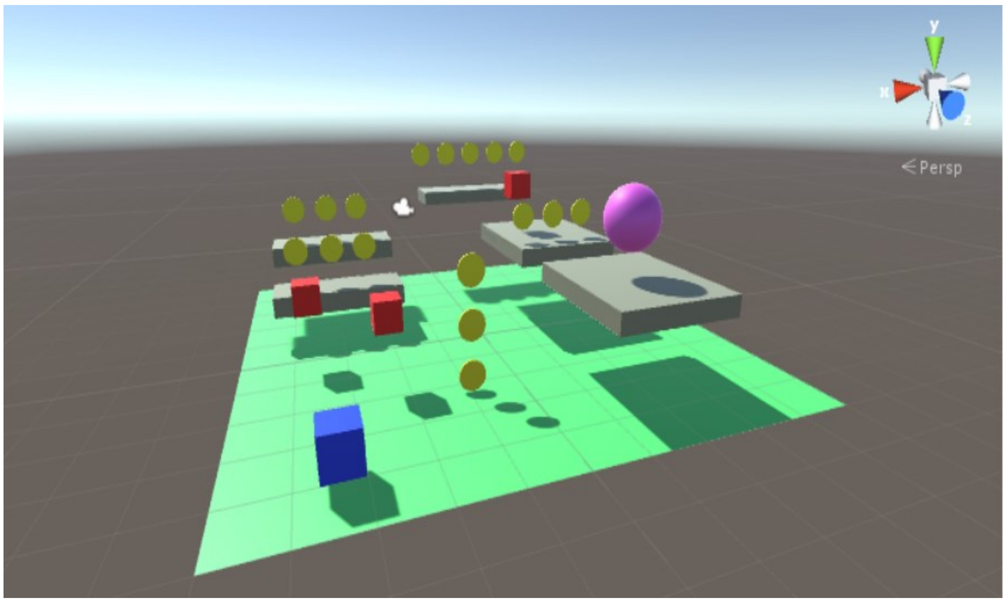


Step 3: Make a coin by scaling down the cylinder on the appropriate axis. Rename the cylinder to “Coin”. Drag the Coin material into your

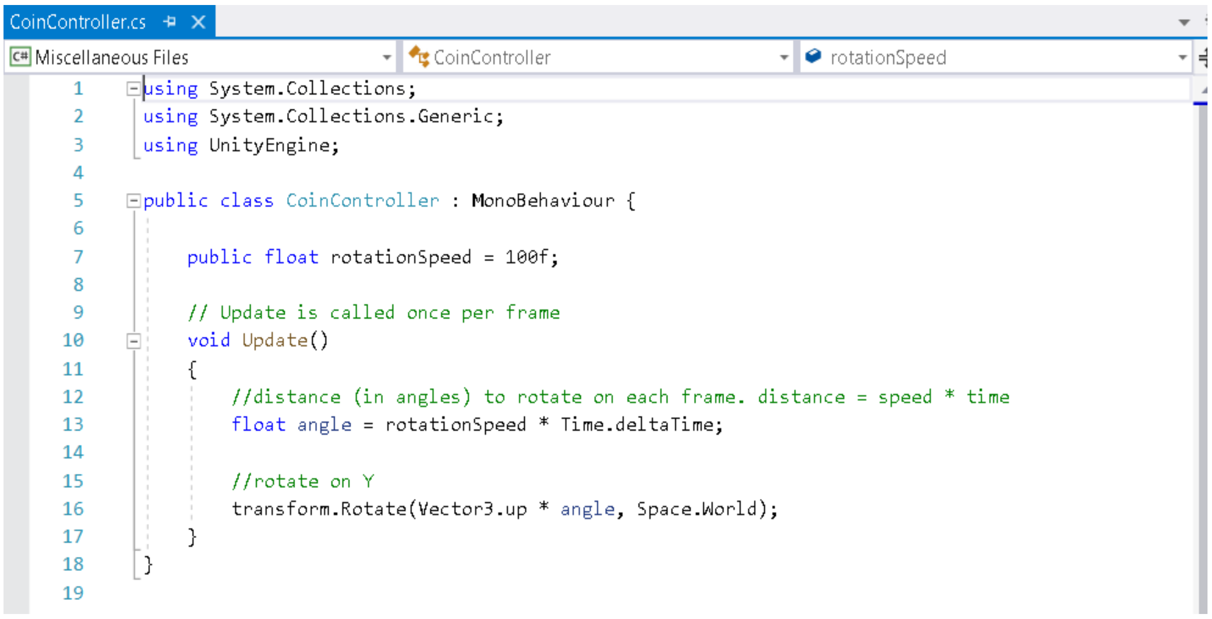
coin. Create a prefab of the same, and recreate it different times as needed.



Step 4: Make the Player and enemies using different cube positions, scales, and colors. Our Player is Blue Cube and enemies are the Red ones.

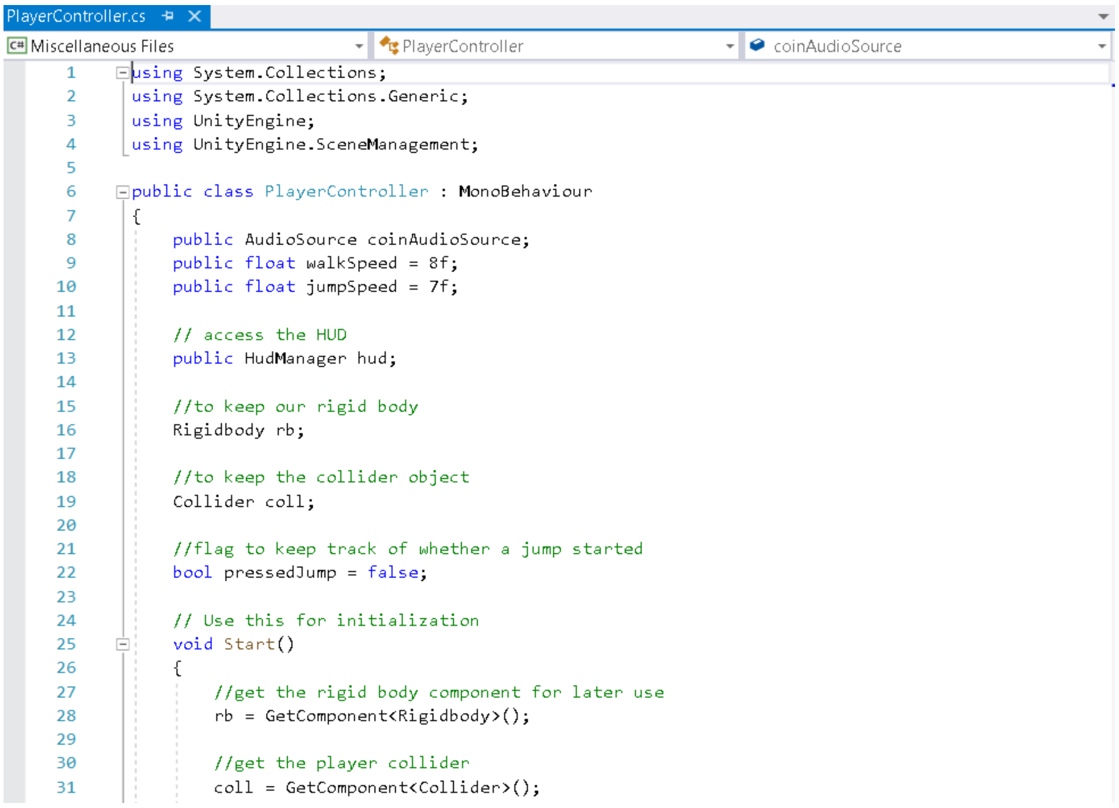


Step 5: Make a script for coin rotation. We need rotation on the Y axis. Attach that to the Coin prefab.

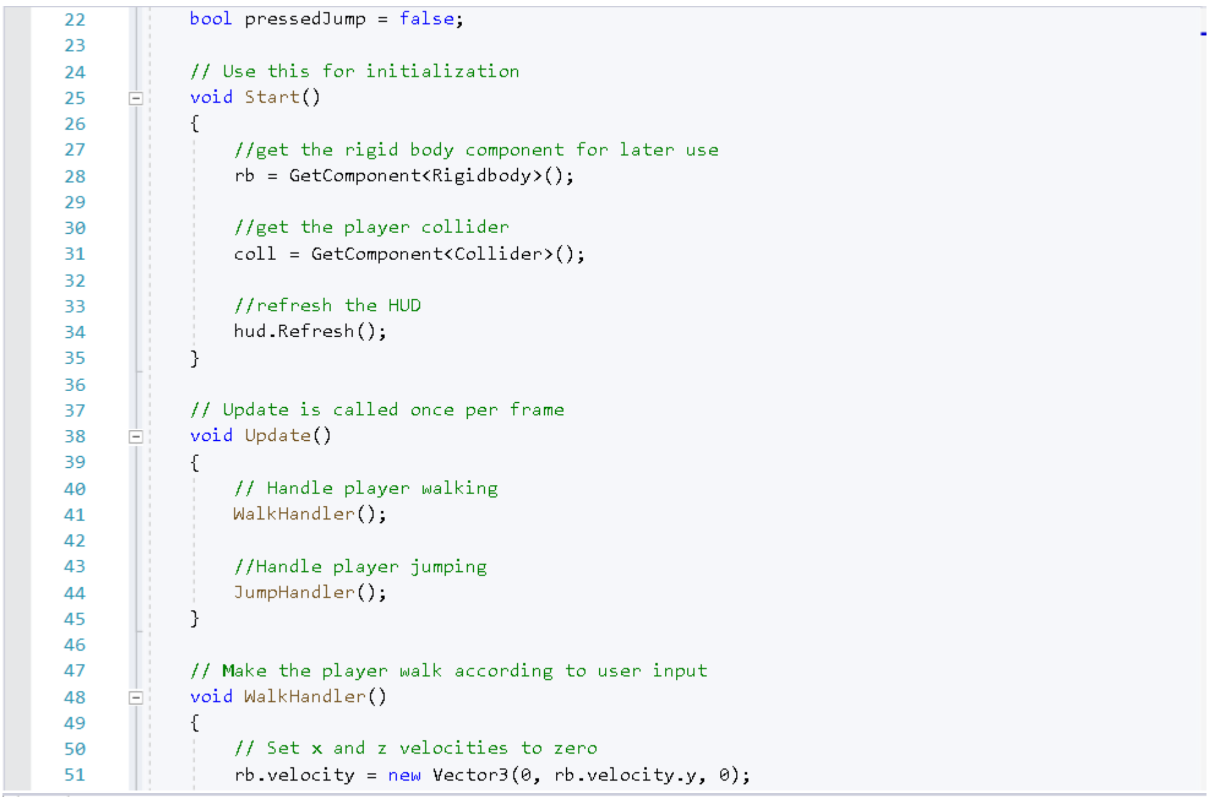


```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CoinController : MonoBehaviour {

    public float rotationSpeed = 100f;

    // Update is called once per frame
    void Update()
    {
        //distance (in angles) to rotate on each frame. distance = speed * time
        float angle = rotationSpeed * Time.deltaTime;

        //rotate on Y
        transform.Rotate(Vector3.up * angle, Space.World);
    }
}
```

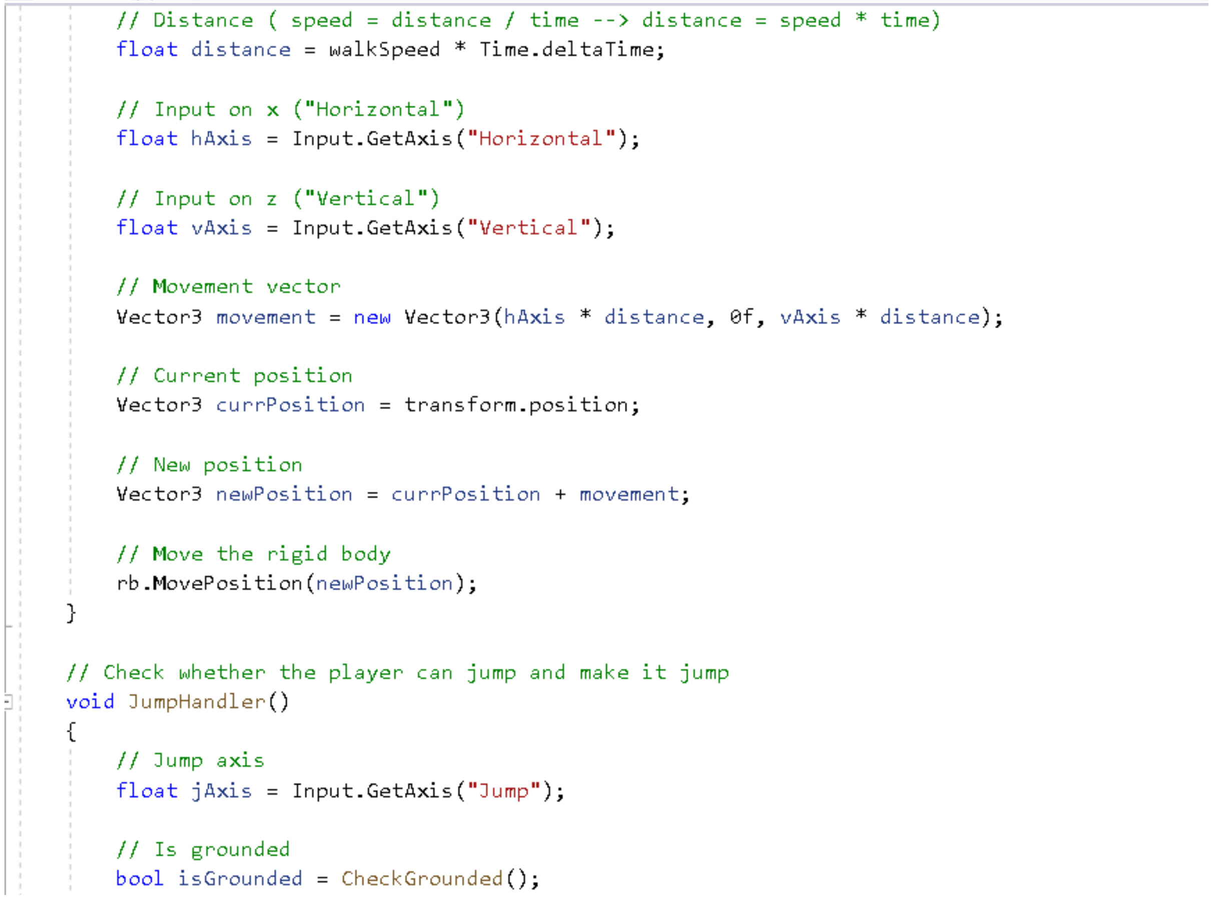
Step 6: In the Inspector click Add Component, select Physics – RigidBody. Create a new script in the Scripts folder and call it PlayerController. We'll start by adding some public properties for the walking and jumping speed of our player. Attach the script to the Player prefab.



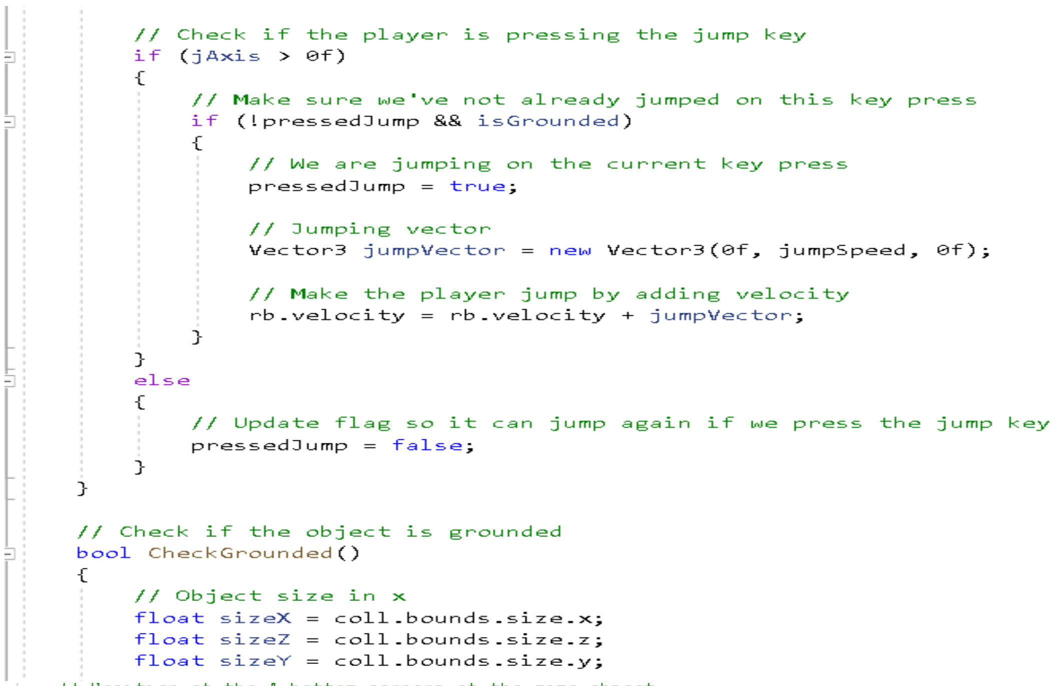
```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PlayerController : MonoBehaviour
{
    public AudioSource coinAudioSource;
    public float walkSpeed = 8f;
    public float jumpSpeed = 7f;

    // access the HUD
    public HudManager hud;

    //to keep our rigid body
    Rigidbody rb;

    //to keep the collider object
    Collider coll;

    //flag to keep track of whether a jump started
    bool pressedJump = false;

    // Use this for initialization
    void Start()
    {
        //get the rigid body component for later use
        rb = GetComponent<Rigidbody>();

        //get the player collider
        coll = GetComponent<Collider>();
```



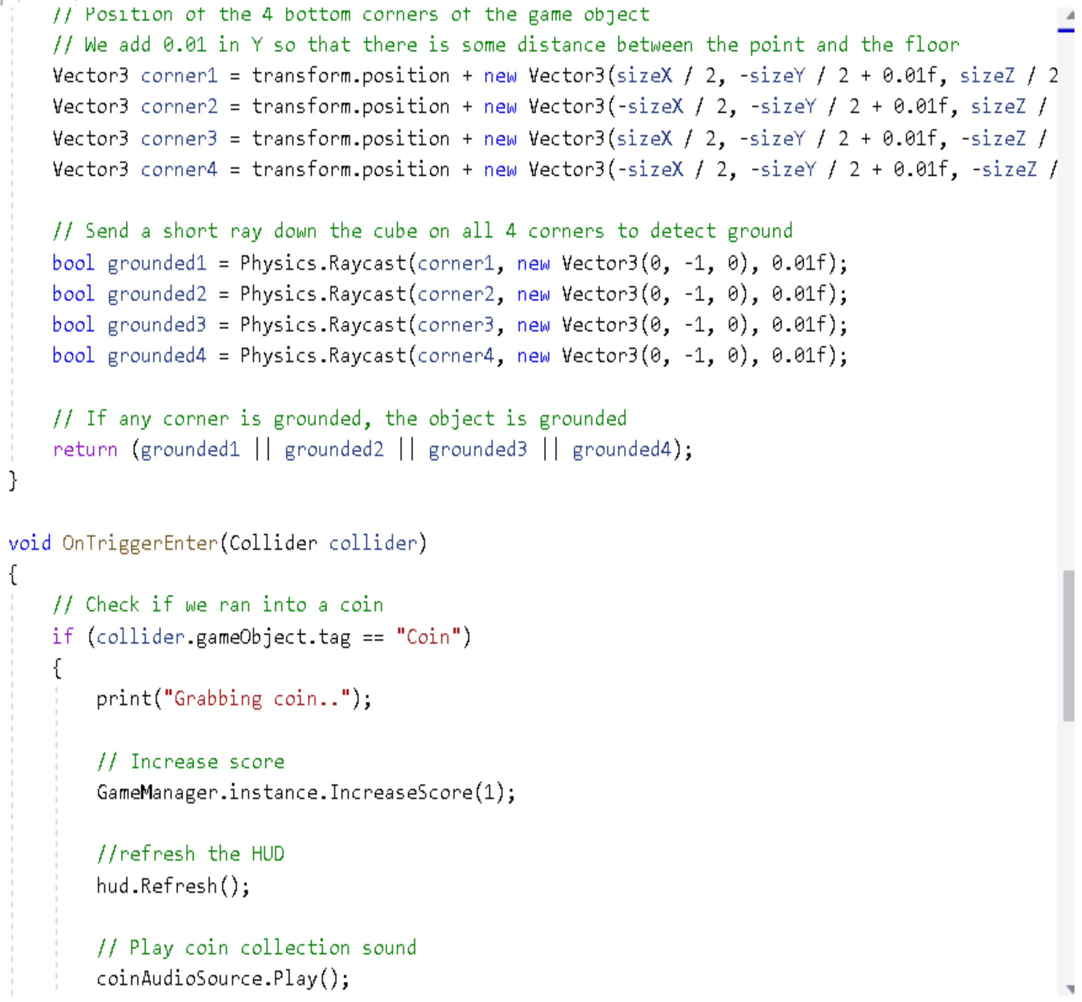
```csharp
    bool pressedJump = false;

    // Use this for initialization
    void Start()
    {
        //get the rigid body component for later use
        rb = GetComponent<Rigidbody>();

        //get the player collider
        coll = GetComponent<Collider>();

        //refresh the HUD
        hud.Refresh();
    }

    // Update is called once per frame
    void Update()
    {
        // Handle player walking
        WalkHandler();

        //Handle player jumping
        JumpHandler();
    }

    // Make the player walk according to user input
    void WalkHandler()
    {
        // Set x and z velocities to zero
        rb.velocity = new Vector3(0, rb.velocity.y, 0);
```



```csharp
        // Distance ( speed = distance / time --> distance = speed * time)
        float distance = walkSpeed * Time.deltaTime;

        // Input on x ("Horizontal")
        float hAxis = Input.GetAxis("Horizontal");

        // Input on z ("Vertical")
        float vAxis = Input.GetAxis("Vertical");

        // Movement vector
        Vector3 movement = new Vector3(hAxis * distance, 0f, vAxis * distance);

        // Current position
        Vector3 currPosition = transform.position;

        // New position
        Vector3 newPosition = currPosition + movement;

        // Move the rigid body
        rb.MovePosition(newPosition);
    }

    // Check whether the player can jump and make it jump
    void JumpHandler()
    {
        // Jump axis
        float jAxis = Input.GetAxis("Jump");

        // Is grounded
        bool isGrounded = CheckGrounded();
```

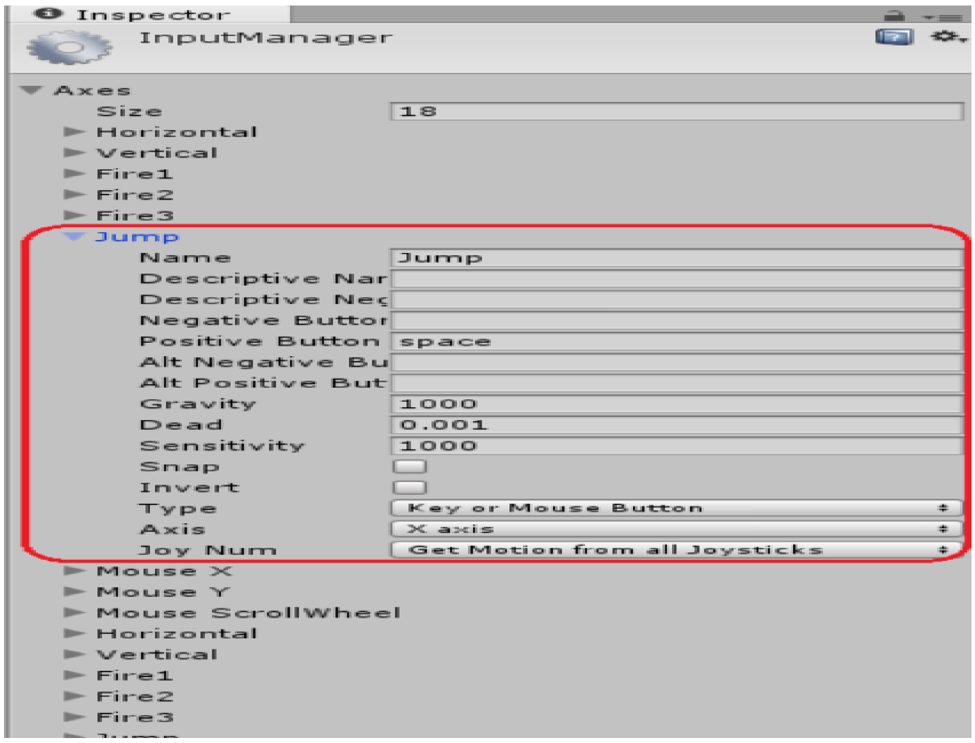


```
        // Check if the player is pressing the jump key
        if (jAxis > 0f)
        {
            // Make sure we've not already jumped on this key press
            if (!pressedJump && isGrounded)
            {
                // We are jumping on the current key press
                pressedJump = true;

                // Jumping vector
                Vector3 jumpVector = new Vector3(0f, jumpSpeed, 0f);

                // Make the player jump by adding velocity
                rb.velocity = rb.velocity + jumpVector;
            }
        }
        else
        {
            // Update flag so it can jump again if we press the jump key
            pressedJump = false;
        }
    }

    // Check if the object is grounded
    bool CheckGrounded()
    {
        // Object size in x
        float sizeX = coll.bounds.size.x;
        float sizeZ = coll.bounds.size.z;
        float sizeY = coll.bounds.size.y;
```



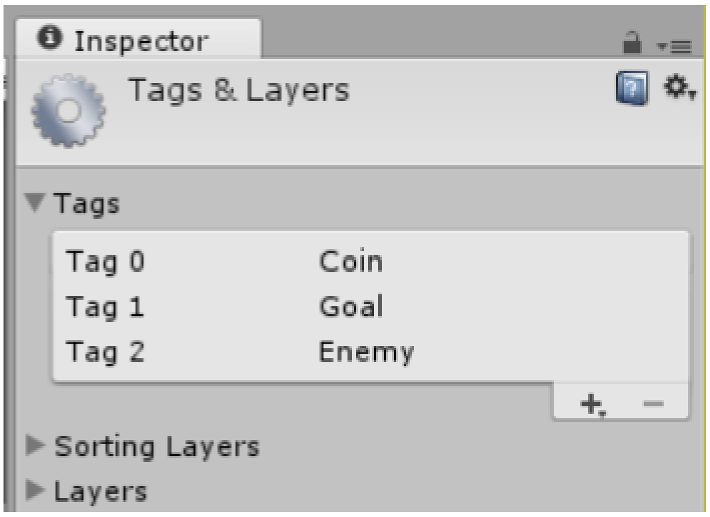
```
    // Position of the 4 bottom corners of the game object
    // We add 0.01 in Y so that there is some distance between the point and the floor
    Vector3 corner1 = transform.position + new Vector3(sizeX / 2, -sizeY / 2 + 0.01f, sizeZ / 2
    Vector3 corner2 = transform.position + new Vector3(-sizeX / 2, -sizeY / 2 + 0.01f, sizeZ /
    Vector3 corner3 = transform.position + new Vector3(sizeX / 2, -sizeY / 2 + 0.01f, -sizeZ /
    Vector3 corner4 = transform.position + new Vector3(-sizeX / 2, -sizeY / 2 + 0.01f, -sizeZ /

    // Send a short ray down the cube on all 4 corners to detect ground
    bool grounded1 = Physics.Raycast(corner1, new Vector3(0, -1, 0), 0.01f);
    bool grounded2 = Physics.Raycast(corner2, new Vector3(0, -1, 0), 0.01f);
    bool grounded3 = Physics.Raycast(corner3, new Vector3(0, -1, 0), 0.01f);
    bool grounded4 = Physics.Raycast(corner4, new Vector3(0, -1, 0), 0.01f);

    // If any corner is grounded, the object is grounded
    return (grounded1 || grounded2 || grounded3 || grounded4);
}

void OnTriggerEnter(Collider collider)
{
    // Check if we ran into a coin
    if (collider.gameObject.tag == "Coin")
    {
        print("Grabbing coin..");

        // Increase score
        GameManager.instance.IncreaseScore(1);

        //refresh the HUD
        hud.Refresh();

        // Play coin collection sound
        coinAudioSource.Play();
        // Destroy coin
        Destroy(collider.gameObject);
    }
    else if (collider.gameObject.tag == "Enemy")
    {
        // Game over
        print("game over");

        SceneManager.LoadScene("Game Over");
    }
    else if (collider.gameObject.tag == "Goal")
    {
        print("goal reached");

        // Increase level
        GameManager.instance.IncreaseLevel();
    }

}
```

Step 7: Unity comes with an Input Axis called “Jump”, which activates with the spacebar by default. Go to Edit – Project Settings – Input to double check it’s there on your end.



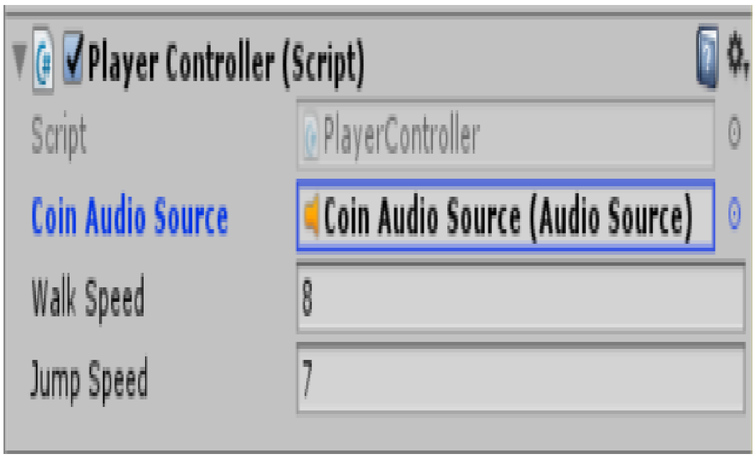
```
Inspector
InputManager
▼ Axes
    Size                    18
  ► Horizontal
  ► Vertical
  ► Fire1
  ► Fire2
  ► Fire3
  ▼ Jump
      Name                  Jump
      Descriptive Nar
      Descriptive Neç
      Negative Buttor
      Positive Button       space
      Alt Negative Bu
      Alt Positive But
      Gravity               1000
      Dead                  0.001
      Sensitivity           1000
      Snap                  ☐
      Invert                ☐
      Type                  Key or Mouse Button
      Axis                  X axis
      Joy Num               Get Motion from all Joysticks
  ► Mouse X
  ► Mouse Y
  ► Mouse ScrollWheel
  ► Horizontal
  ► Vertical
  ► Fire1
  ► Fire2
  ► Fire3
```

Step 8: We now are going to make the coins disappear after the Player goes through it. We need to make our coins a trigger collider. When another object collides with a coins, a trigger event will be fired, and the physics properties of the object won't be affected, just as if you were going through thin air! Select the coin prefab, find the component Capsule Collider in the Inspector, and check Is Trigger. To detect whether the trigger object the player ran into is a coin we can give our coin prefab a tag that identifies it. Select the coin prefab and in the Inspector, in Tag go and select Add Tag.



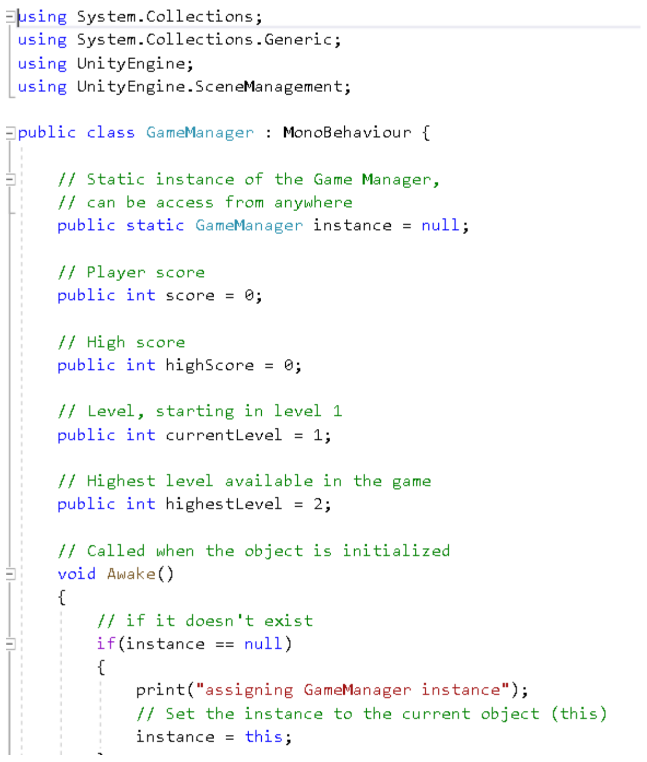
Tag 0	Coin
Tag 1	Goal
Tag 2	Enemy

Step 9: In the Hierarchy Window, right click and select Audio – Audio Source, rename it to "Coin Audio Source". Select the newly created object. In the Inspector, drag the coin.ogg file to AudioClip. Uncheck the box Play On Awake so that the sound doesn't play each time the game starts.

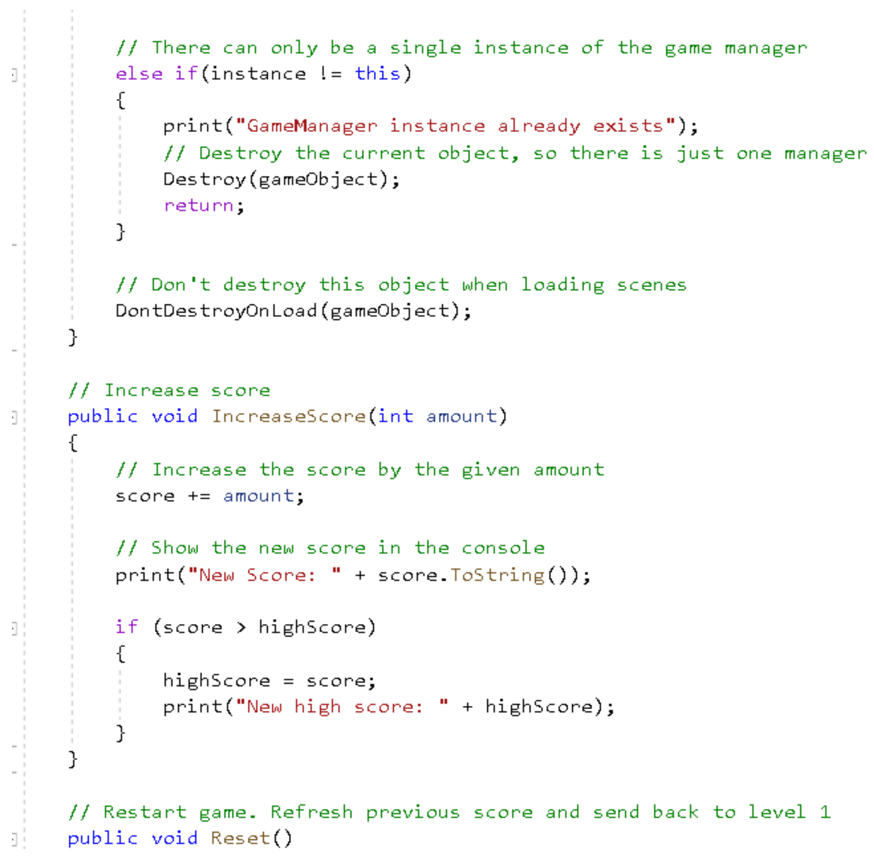


Script	PlayerController
Coin Audio Source	Coin Audio Source (Audio Source)
Walk Speed	8
Jump Speed	7

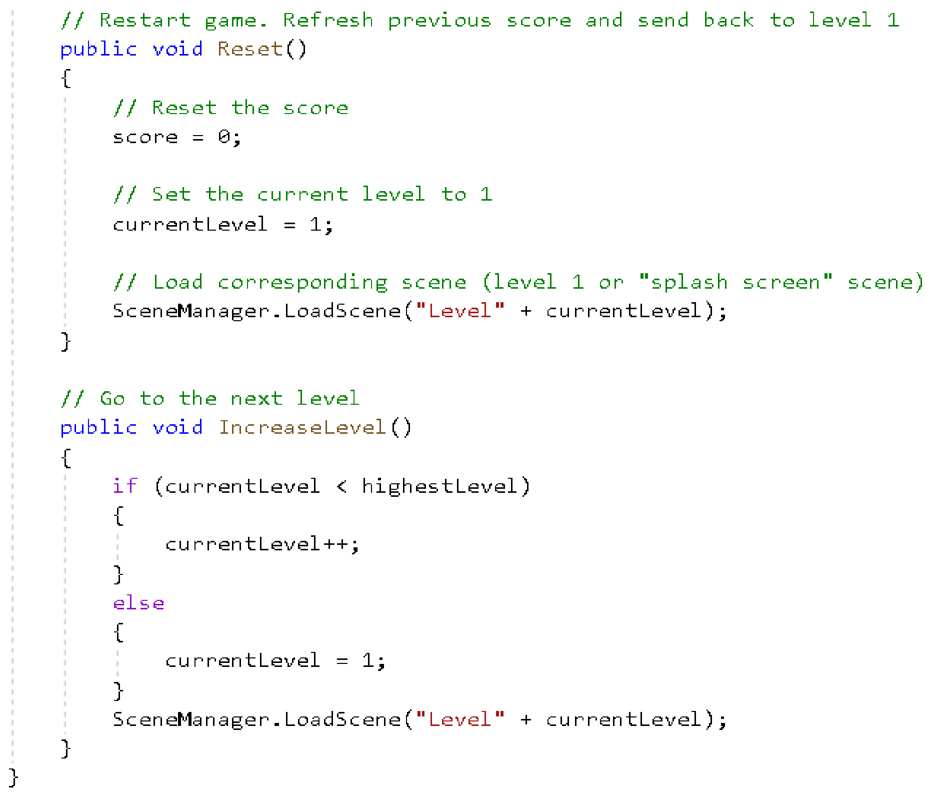
Step 10: Now we need to increase or decrease the score using the following script.



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

public class GameManager : MonoBehaviour {

    // Static instance of the Game Manager,
    // can be access from anywhere
    public static GameManager instance = null;

    // Player score
    public int score = 0;

    // High score
    public int highScore = 0;

    // Level, starting in level 1
    public int currentLevel = 1;

    // Highest level available in the game
    public int highestLevel = 2;

    // Called when the object is initialized
    void Awake()
    {
        // if it doesn't exist
        if(instance == null)
        {
            print("assigning GameManager instance");
            // Set the instance to the current object (this)
            instance = this;
```

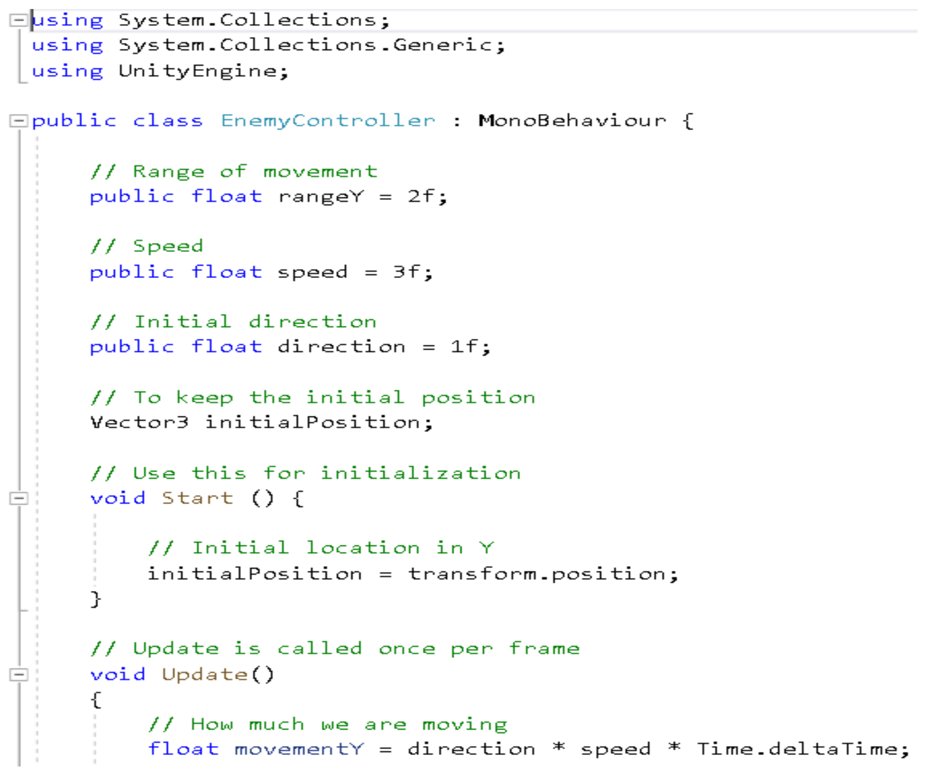


```
        // There can only be a single instance of the game manager
        else if(instance != this)
        {
            print("GameManager instance already exists");
            // Destroy the current object, so there is just one manager
            Destroy(gameObject);
            return;
        }

        // Don't destroy this object when loading scenes
        DontDestroyOnLoad(gameObject);
    }

    // Increase score
    public void IncreaseScore(int amount)
    {
        // Increase the score by the given amount
        score += amount;

        // Show the new score in the console
        print("New Score: " + score.ToString());

        if (score > highScore)
        {
            highScore = score;
            print("New high score: " + highScore);
        }
    }

    // Restart game. Refresh previous score and send back to level 1
    public void Reset()
```

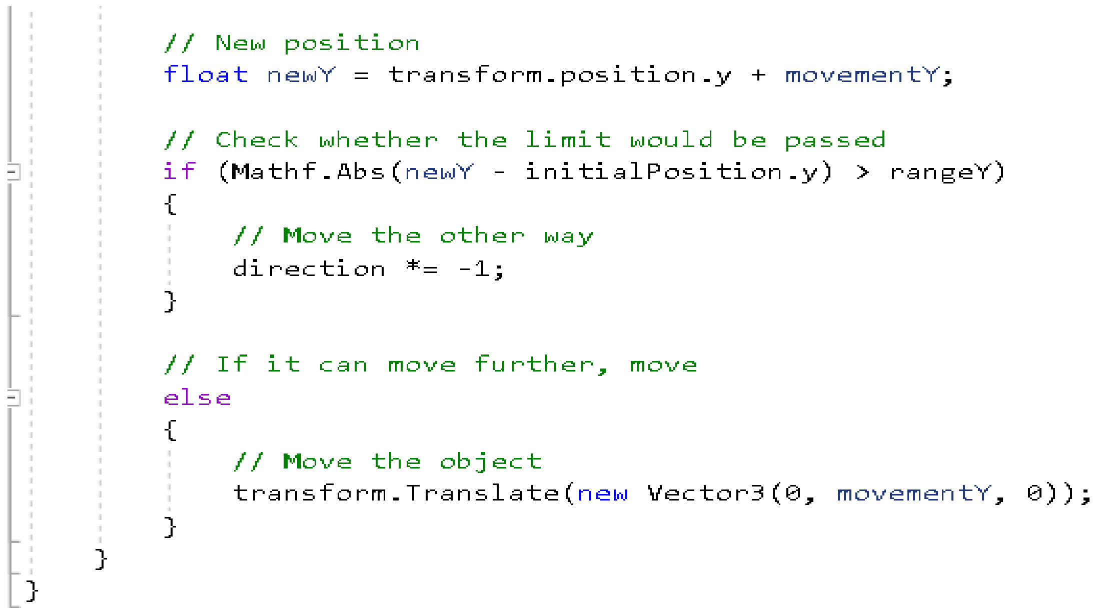


```
    // Restart game. Refresh previous score and send back to level 1
    public void Reset()
    {
        // Reset the score
        score = 0;

        // Set the current level to 1
        currentLevel = 1;

        // Load corresponding scene (level 1 or "splash screen" scene)
        SceneManager.LoadScene("Level" + currentLevel);
    }

    // Go to the next level
    public void IncreaseLevel()
    {
        if (currentLevel < highestLevel)
        {
            currentLevel++;
        }
        else
        {
            currentLevel = 1;
        }
        SceneManager.LoadScene("Level" + currentLevel);
    }
}
```

Step 11: Enemy movement should also be initiated. For that, we will create a script for their movement and attach it to that Prefab.

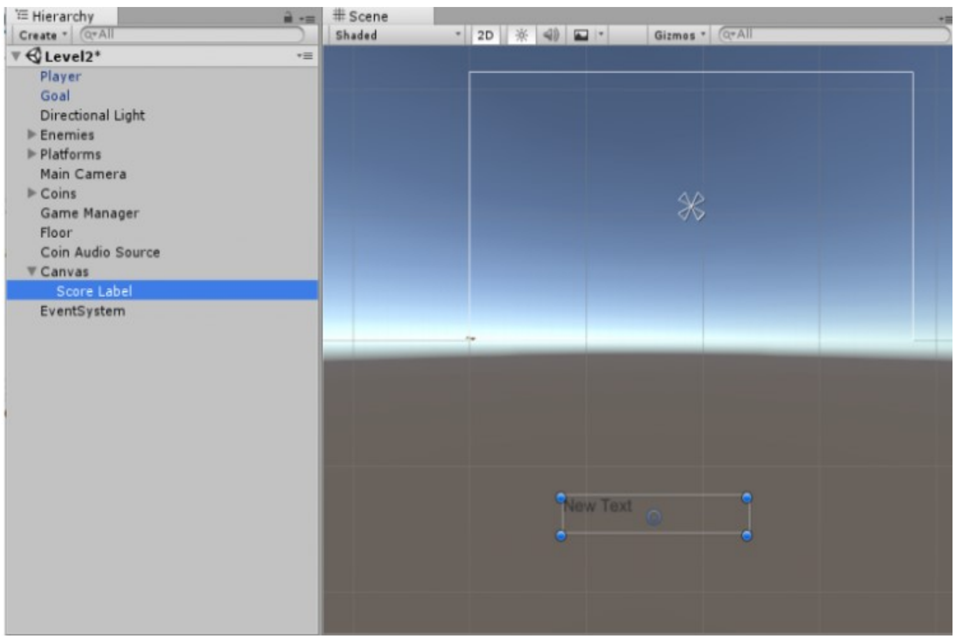


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyController : MonoBehaviour {

    // Range of movement
    public float rangeY = 2f;

    // Speed
    public float speed = 3f;

    // Initial direction
    public float direction = 1f;

    // To keep the initial position
    Vector3 initialPosition;

    // Use this for initialization
    void Start () {

        // Initial location in Y
        initialPosition = transform.position;
    }

    // Update is called once per frame
    void Update()
    {
        // How much we are moving
        float movementY = direction * speed * Time.deltaTime;
```



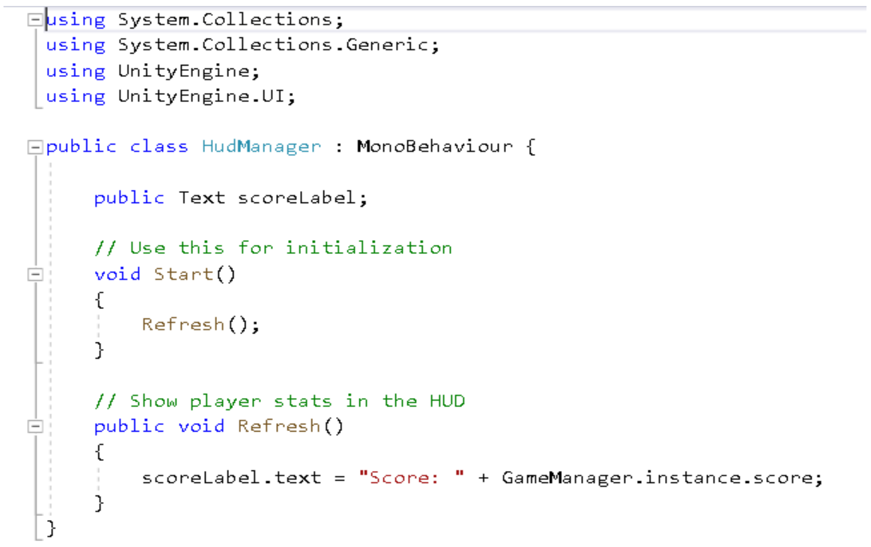
```
        // New position
        float newY = transform.position.y + movementY;

        // Check whether the limit would be passed
        if (Mathf.Abs(newY - initialPosition.y) > rangeY)
        {
            // Move the other way
            direction *= -1;
        }

        // If it can move further, move
        else
        {
            // Move the object
            transform.Translate(new Vector3(0, movementY, 0));
        }
    }
}
```

Step 12: Add another level by copy-pasting the necessary things. Start by adding a text element in the Hierarchy Window. Right-click, UI – Text. This will create a Canvas element, with a Text element inside. Rename the text element to “Score Label”. In the Scene View, click 2D, select the text and press f.



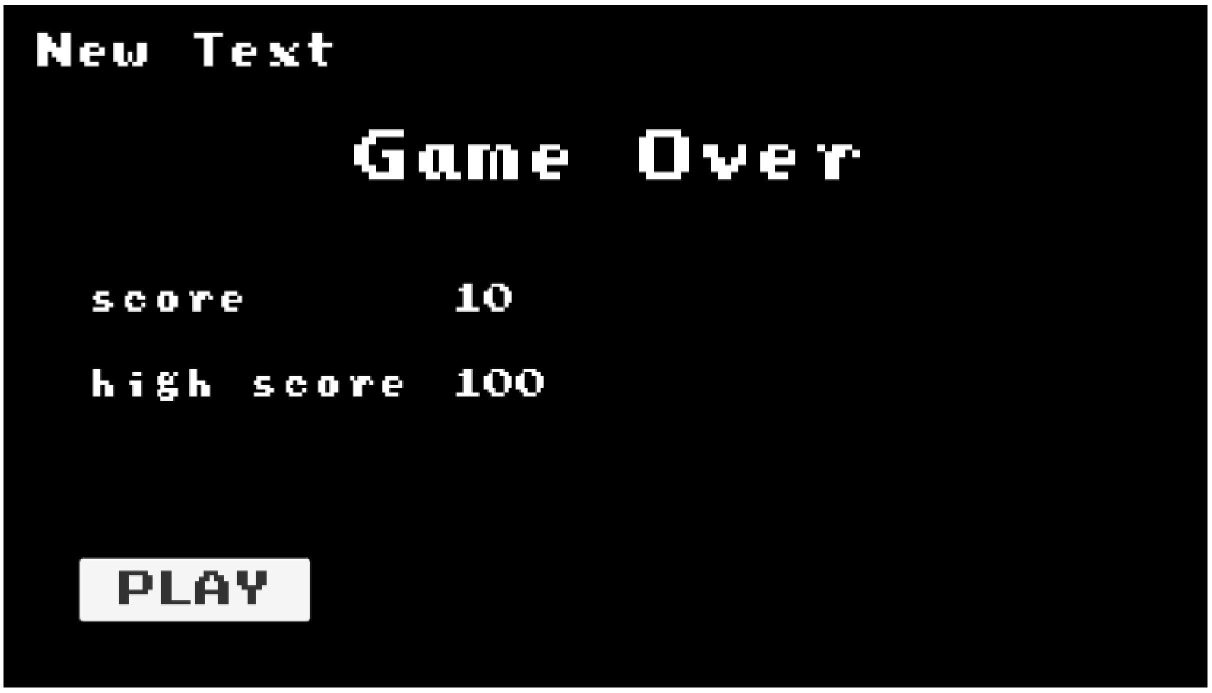
```
Level2*
    Player
    Goal
    Directional Light
  ▶ Enemies
  ▶ Platforms
    Main Camera
  ▶ Coins
    Game Manager
    Floor
    Coin Audio Source
  ▼ Canvas
      Score Label
    EventSystem
```

Step 13: We’ll create a new object that will take care of managing the HUD. Create an empty object called Hud Manager. Create a new script and name it HudManager. Drag this new script onto the Hud Manager object in the Hierarchy View.

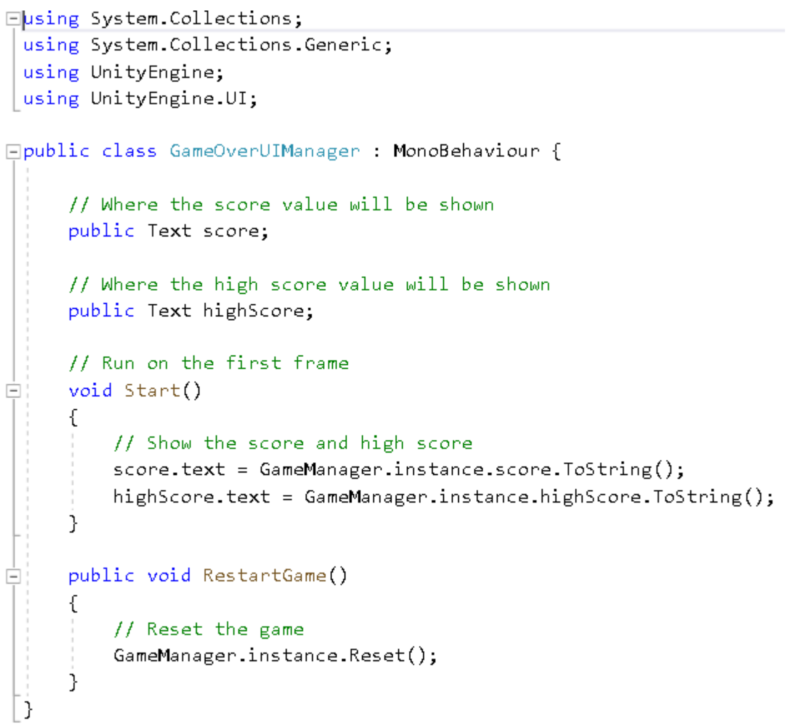


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HudManager : MonoBehaviour {

    public Text scoreLabel;

    // Use this for initialization
    void Start()
    {
        Refresh();
    }

    // Show player stats in the HUD
    public void Refresh()
    {
        scoreLabel.text = "Score: " + GameManager.instance.score;
    }
}
```

Step 14: We make some more texts like Game Over Screen too.



Step 15: Create a new script called GameOverUIManager. This new class will have public variables so that we can pass on the text elements where we want to show the score and high score. Also, it will have a public method to restart the game, which we can use in our Play button.



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

public class GameOverUIManager : MonoBehaviour {

    // Where the score value will be shown
    public Text score;

    // Where the high score value will be shown
    public Text highScore;

    // Run on the first frame
    void Start()
    {
        // Show the score and high score
        score.text = GameManager.instance.score.ToString();
        highScore.text = GameManager.instance.highScore.ToString();
    }

    public void RestartGame()
    {
        // Reset the game
        GameManager.instance.Reset();
    }
}
```

Step 16: Save the project and export it or Build it for yourself as a game.