**Super Capsule**

**Game Application Development Project Report**

**Submitted by:**

**Student Name:** Hadia Naeem

**Student Name:** Syeda Naima Nadeem

**Student Name:** Noor Ayesha Malik

**Student Name:** Uroosa Ahmed Yar

**June 2023**

**Department of Computer Science and Software Engineering**

**Jinnah University for Women**

Table of Contents

[**1.** **GAME INTRODUCTION** 2](#_Toc137121287)

[**2.** **GAME STORYLINE** 3](#_Toc137121288)

[**3.** **SOURCE CODE** 3](#_Toc137121289)

[**Start Menu** 3](#_Toc137121290)

[**Player Movement** 3](#_Toc137121291)

[**Player Life** 5](#_Toc137121292)

[**Item Collector** 6](#_Toc137121293)

[**Way Point Follower** 6](#_Toc137121294)

[**Sticky Platform** 7](#_Toc137121295)

[**Rotate** 8](#_Toc137121296)

[**Finish** 8](#_Toc137121297)

[**End Menu** 8](#_Toc137121298)

[**4.** **FLOWCHART** 10](#_Toc137121299)

[**5.** **SCREENSHOTS** 11](#_Toc137121300)

[**Start Scene** 11](#_Toc137121301)

[**Level 01** 12](#_Toc137121302)

[**Level 02** 14](#_Toc137121303)

[**End Scene** 15](#_Toc137121304)

[**6.** **REFERENCE** 16](#_Toc137121305)

# **GAME INTRODUCTION**

Super Capsule is a single-player survival game played entirely through a radar interface. Players navigate a hostile landscape, learn about their surroundings, and unravel a classic story in the process. Super Capsule is a very minimalistic game. All you have to work with is the radar screen, which gives you vision around you, and your scan ability, which allows you to find your objective, hazards, power vents, air pockets, and the sound of your breathing. It's an excellent little design, which pushes just the right buttons to be a tense, atmospheric experience.

# **GAME STORYLINE**

Super Capsule is a game that can only be played on a pc/desktop. It originated from Super Mario. Play using used or arrow keys and jump over the moving red enemy object to kill it. Complete all the levels to finish it. This super capsule game contains two levels, moving enemies and coins to collect by the player/capsule. The player can kill enemies if the player is able to land above the enemy otherwise it will be killed by the enemy. If a player falls down/dies it will come back to the start of the level in which the player currently is. The game also contains start and end screens so one can play this game as many times they want. We also added different sound effects on the jump, die, collecting coins, etc.

# **SOURCE CODE**

## **Start Menu**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class StartMenu : MonoBehaviour

{

    public void StartGame()

    {

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

    }

}

## **Player Movement**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

    Rigidbody rb;

    [SerializeField] float movementSpeed = 6f;

    [SerializeField] float jumpForce = 5f;

    [SerializeField] Transform groundCheck;

    [SerializeField] LayerMask ground;

    [SerializeField] AudioSource jumpSound;

    // Start is called before the first frame update

    void Start()

    {

        rb = GetComponent<Rigidbody>();

    }

    // Update is called once per frame

    void Update()

    {

        float horizontalInput = Input.GetAxis("Horizontal");

        float verticalInput = Input.GetAxis("Vertical");

        rb.velocity = new Vector3(horizontalInput \* movementSpeed, rb.velocity.y, verticalInput \* movementSpeed);

        if (Input.GetButtonDown("Jump") && IsGrounded())

        {

            Jump();

        }

    }

    void Jump()

    {

        rb.velocity = new Vector3(rb.velocity.x, jumpForce, rb.velocity.z);

        jumpSound.Play();

    }

    private void OnCollisionEnter(Collision collision)

    {

        if (collision.gameObject.CompareTag("Enemy Head"))

        {

            Destroy(collision.transform.parent.gameObject);

            Jump();

        }

    }

    bool IsGrounded()

    {

        return Physics.CheckSphere(groundCheck.position, .1f, ground);

    }

}

## **Player Life**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class PlayerLife : MonoBehaviour

{

    [SerializeField] AudioSource deathSound;

    bool dead = false;

    private void Update()

    {

        if (transform.position.y < -1f && !dead)

        {

            Die();

        }

    }

    private void OnCollisionEnter(Collision collision)

    {

        if (collision.gameObject.CompareTag("Enemy Body"))

        {

            GetComponent<MeshRenderer>().enabled = false;

            GetComponent<Rigidbody>().isKinematic = true;

            GetComponent<PlayerMovement>().enabled = false;

            Die();

        }

    }

    void Die()

    {

        Invoke(nameof(ReloadLevel), 1.3f);

        dead = true;

        deathSound.Play();

    }

    void ReloadLevel()

    {

        SceneManager.LoadScene(SceneManager.GetActiveScene().name);

    }

}

## **Item Collector**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class ItemCollector : MonoBehaviour

{

    int coins = 0;

    [SerializeField] TextMeshProUGUI coinsText;

    [SerializeField] AudioSource collectionSound;

    private void OnTriggerEnter(Collider other)

    {

        if (other.gameObject.CompareTag("Coin"))

        {

            Destroy(other.gameObject);

            coins++;

            coinsText.text = "Coins: " + coins;

            collectionSound.Play();

        }

    }

}

## **Way Point Follower**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class WaypointFollower : MonoBehaviour

{

    [SerializeField] GameObject[] waypoints;

    int currentWaypointIndex = 0;

    [SerializeField] float speed = 1f;

    void Update()

    {

        if (Vector3.Distance(transform.position, waypoints[currentWaypointIndex].transform.position) < .1f)

        {

            currentWaypointIndex++;

            if (currentWaypointIndex >= waypoints.Length)

            {

                currentWaypointIndex = 0;

            }

        }

        transform.position = Vector3.MoveTowards(transform.position, waypoints[currentWaypointIndex].transform.position, speed \* Time.deltaTime);

    }

}

## **Sticky Platform**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class StickyPlatform : MonoBehaviour

{

     private void OnCollisionEnter(Collision collision)

    {

        if (collision.gameObject.name == "Player")

        {

            collision.gameObject.transform.SetParent(transform);

        }

    }

    private void OnCollisionExit(Collision collision)

    {

        if (collision.gameObject.name == "Player")

        {

            collision.gameObject.transform.SetParent(null);

        }

    }

}

## **Rotate**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Rotate : MonoBehaviour

{

    [SerializeField] float speedX;

    [SerializeField] float speedY;

    [SerializeField] float speedZ;

    void Update()

    {

        transform.Rotate(360 \* speedX \* Time.deltaTime, 360 \* speedY \* Time.deltaTime, 360 \* speedZ \* Time.deltaTime);

    }

}

## **Finish**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Finish : MonoBehaviour

{

    private void OnTriggerEnter(Collider other)

    {

        if (other.gameObject.name == "Player")

        {

            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

        }

    }

}

## **End Menu**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EndMenu : MonoBehaviour

{

    public void QuitGame()

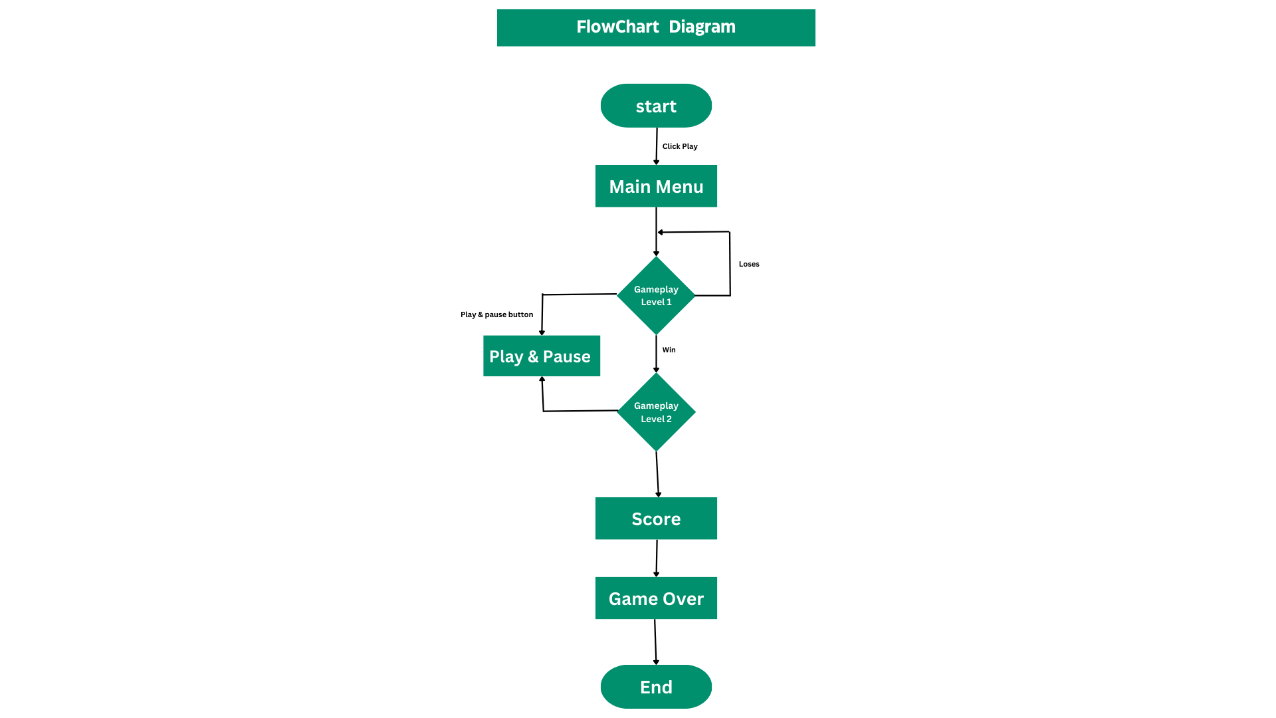
    {

        Application.Quit();

    }

}

# **FLOWCHART**



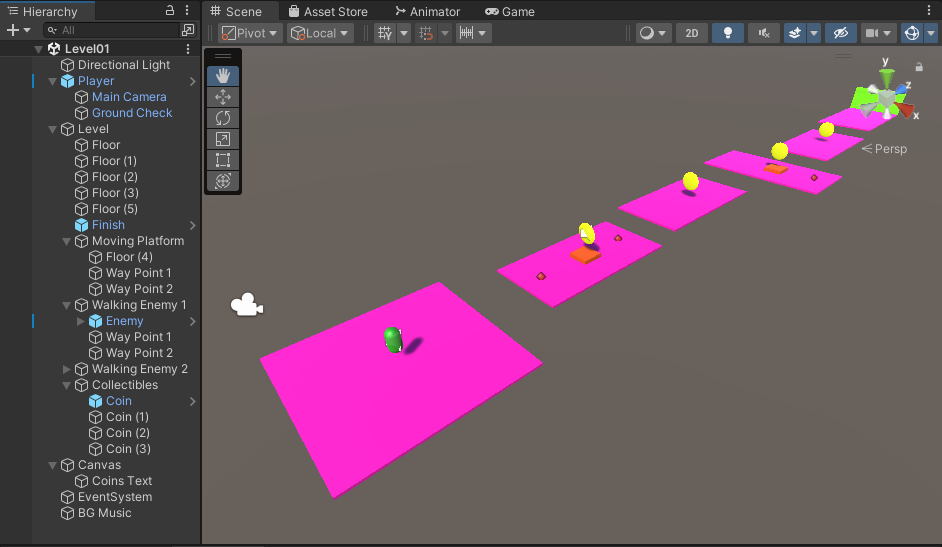
# **SCREENSHOTS**

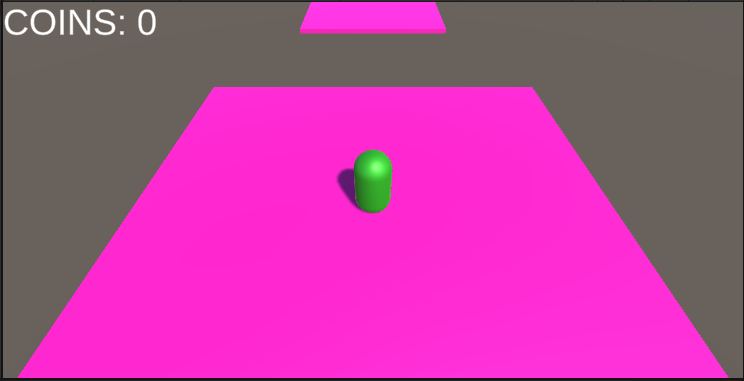
## **Start Scene**

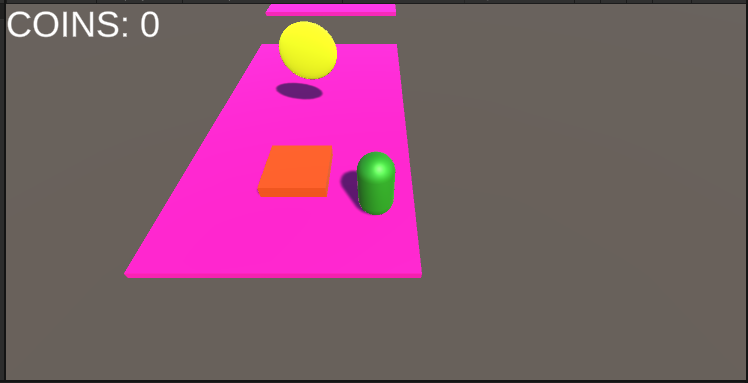


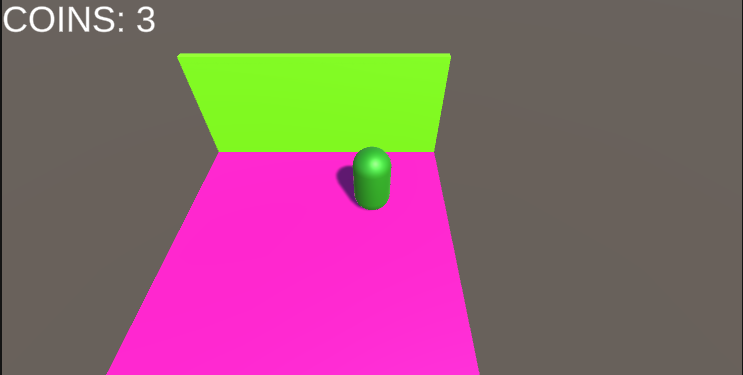


## **Level 01**

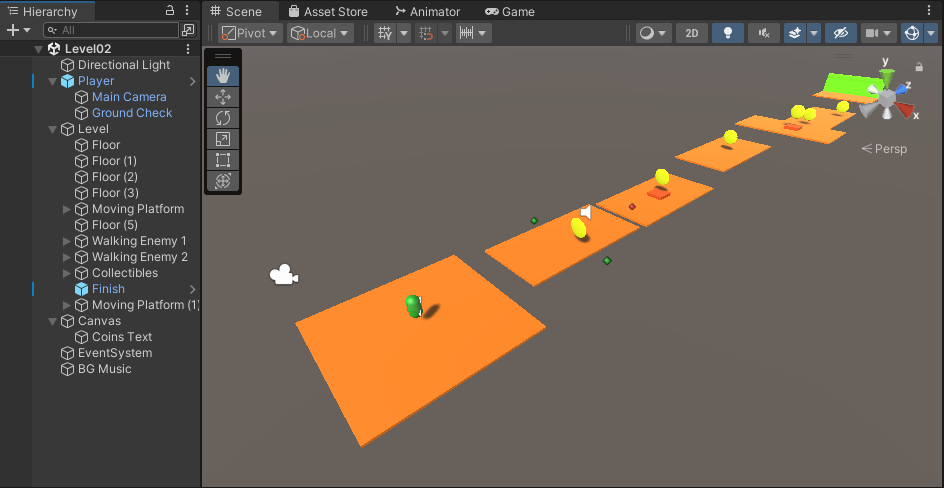


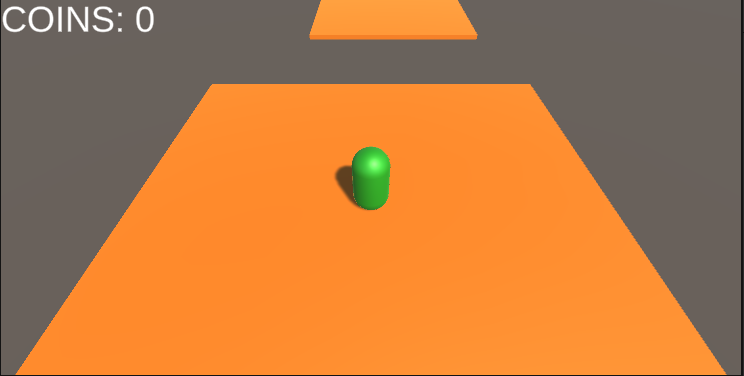


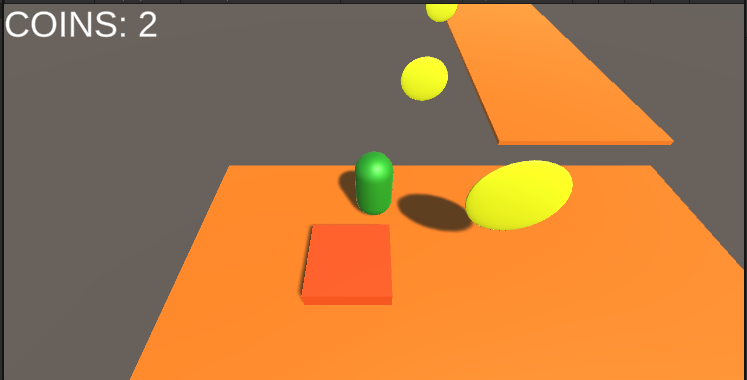




## **Level 02**







## **End Scene**

