

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award
of Bachelor of Technology Degree in Information Technology

By
Vaibhav Vishal

38120087



DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND
TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119. TAMILNADU

JANUARY 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" Grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119.

Phone: 044 - 2450 3150 / 51 / 52 / 54 / 55 Fax: 044 - 2450 2344

www.sathyabama.ac.in



SCHOOL OF COMPUTING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of who carried out the project entitled "ROLLER MADNESS" under our supervision from August 2020 to November 2020.

Internal Guide

M.S INDIRA K

Head of the Department

Dr. R. Subhashini M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I “VAIBHAV VISHAL” hereby declare that the Project Report entitled “**ROLLER MADNESS**” done by me under the guidance of “M.S INDIRA K”, at Sathyabama Institute of Science and Technology (Deemed to be University), Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119 is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 21/01/2021

Place: Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge our sincere thanks to Board of management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey our thanks to **Dr. T. SASIKALA M.E., Ph.D., Dean, School of Computing** and **Dr. R. SUBHASHINI M.E., Ph.D., Head of the Department, Department of Information Technology** for providing us the necessary support and details at the right time during the progressive reviews.

I would like to express our sincere and deep sense of gratitude to our Project Guide **M.S INDIRA K**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express our thanks to all Teaching and Non-teaching staff members of the Department of **INFORMATION TECHNOLOGY** who were helpful in many ways for the completion of the project.

TRAINING CERTIFICATE

MICHIGAN STATE UNIVERSITY

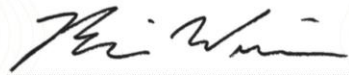
Nov 23, 2020

Vaibhav Vishal

has successfully completed

Introduction to Game Development

an online non-credit course authorized by Michigan State University and offered through
Coursera



Brian M. Winn
Associate Professor
Media and Information

COURSE
CERTIFICATE



Verify at coursera.org/verify/X7E2V337CYES

Coursera has confirmed the identity of this individual and their
participation in the course.



Inspiring and empowering
future professionals

Vaibhav Vishal

Data Analytics Consulting Virtual Internship

Certificate of Completion
August 23rd, 2020

Over the period of August 2020, Vaibhav Vishal has completed practical task modules in:

Data Insights
Data Insights and Presentation
Data Quality Assessment

Deborah Yates
National Managing
Partner People
Performance and
Culture

Tom Brunskill
CEO, Co-Founder of
Forage

Enrolment Verification Code h6szX9xX9gXPTsQbu | User Verification Code 5LfoSRq8D3rM8jGvb | Issued by Forage

ABSTRACT

ROLLER MADNESS

Roller Madness Game project is developed using “UNITY GAME ENGINE”. The language used for the development of this project is “C#”. The project file contains Assets such as C# scripts, prefabs, textures, sounds, animations, models, scenes and many more. Roller Madness is a 3D game specially designed for PCs. The gameplay Graphics is good enough for PCs and controls are simple for the users.

CHAPTER NO	TITLE NAME	PAGE NO
	ABSTRACT	7
	LIST OF ABBREVIATION	9
1	INTRODUCTION	9
	1.1 GENERAL	9
	1.2 OUTLINE OF THE PROJECT	10
2	AIM AND SCOPE	11
	2.1 AIM	11
	2.2 PROBLEM STATEMENT	11
	2.3 SCOPE	11
3	SYSTEM ANALYSIS AND DESIGN	11
	3.1 PLATFORM	11
	3.2 SOFTWARES USED	12
	3.3 ALGORITHM	14
	3.4 PROJECT DESCRIPTION	14
4	RESULT AND DISCUSSION	15
	4.1 SYSTEM REQUIREMENTS	15
	4.2 GAME ARCHITECTURE	15
	4.3 MODULES	15
	4.4 RESULT	17
5	CONCLUSION AND FUTURE WORK	17
	5.1 CONCLUSION	17
	5.2 FUTURE WORK	18
	5.3 REFERENCE	18
	APPENDIX	18
	A.SCREENSHOTS	18
	B.SOURCE CODE	21

LIST OF ABBREVIATIONS

ABBREVIATIONS	EXPANSION
HTML	HYPERTEXT MARK-UP LANGUAGE
C#	C Sharp (Microsoft programming language)
UNITY	A Game Development Engine

CHAPTER 1

INTRODUCTION

1.1GENERAL

Roller Madness Game project is developed using Unity Game Engine. The language used for the development of this project is “C#”. The project file contains Assets such as C# scripts, prefabs, textures, sounds, animations, models, scenes and many more. Roller Madness is a 3D game specially designed for PCs. The gameplay Graphics is good enough for PCs and controls are simple for the users.

1.2 OUTLINE OF THE PROJECT

Talking about the gameplay, Roller Madness game is all about moving a rolling ball with full control avoiding the obstacles and without falling outer game surface. In this game, the player has to control the rolling ball in order to collect yellow coins. There are some cubes which make the ball bounce back. The cube boxes

fall to the game surface and it tries to stop the ball. Where ever the ball moves, the boxed cubes follow and if it touches the ball then the game is over. The gameplay is not so simple, controlling the ball is a bit hard as it moves with extra force and the can dodge two or more boxed cubes in order to destroy it. After Collecting 5 coins, the level is complete. Likewise in another level, there are more obstacles with high difficulties in collecting coins.

CHAPTER 2

AIM AND SCOPE

2.1 AIM

The main objective of this project is to collect a defined number of coins (displayed on a screen) to proceed to the next level. If you fall off the grid or collide with an enemy you are dead and have to replay the level.

2.2 PROBLEM STATEMENT

This project is designed for Entertainment purpose.

2.3 SCOPE

The game development industry have now become a ubiquitous part of our lives with the increasing penetration of technology. Interestingly, the cost of playing online games has plummeted owing to the affordable internet connection

CHAPTER 3

SYSTEM ANALYSIS

3.1 PLATFORM

3.1.1 UNITY 3D

Unity is a game engine developed by Unity Platforms. It's cross-platform and designed to get your vision not just up and running but also allow you to monetize it. It was initially designed for Mac OS but has expanded to over 25 other platforms, making it a flexible option for game developers. Unity3D is designed to

democratize game development. Users have access to features that allow real-time graphics development in both 3D and 2D environments. It offers drag-and-drop functionality with C# integration. Game design is accessible, and new features are released regularly. The Unity Asset Store gives you access to prebuilt features, including terrain, backgrounds, and even period specific weapons you can use to build your perfect game.

3.1.2 USER INTERFACE

The user interface (UI) is the point of human-computer interaction and communication in a device. This can include display screens, keyboards, a mouse and the appearance of a desktop. It is also the way through which a user interacts with an application or a website

3.2 SOFTWARES USED

3.2.1 CODE EDITORS USED

VISUAL STUDIO CODE

Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and MacOS. It includes support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring.

3.2.2 PROGRAMMING LANGUAGES USED

HTML

Hypertext Markup Language is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies

such as Cascading Style Sheets and scripting languages such as JavaScript.

C#

C# (C-Sharp) is a programming language developed by Microsoft that runs on the .NET Framework. C# is used to develop web apps, desktop apps, mobile apps, games and much more.

JAVASCRIPT

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript, often abbreviated as JS, is a high-level, interpreted scripting language that conforms to the ECMAScript specification. JavaScript has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

3.2.3 LIBRARIES AND FRAMEWORKS USED

ASSETS

Shorthand for anything that goes into a video game – characters, objects, sound effects, maps, environments, etc.

SCRIPTS

script language is a programming language for a special run-time environment that automates the execution of tasks; the tasks could alternatively be executed one-by-one by a human operator. Scripting languages are often interpreted, rather than compiled.

3.3 ALGORITHM

Unity3D is a powerful tool for creating games. In this module, you will create your first actual game from start to finish, called Roller Madness. This ball rolling game will introduce you to the concepts of physics, health and damage, pickups, enemies, spawners, a user interface, and a variety of visual and audio effects. This is your right of passage into game development!

One of the reasons we use Unity3D is its visual editor which makes creating interactive games accessible to both creative and technical individuals. In this module, you will create your first Unity3D project from start to finish. Using a variety of graphical and audio assets and a library of scripts, you will create a simple model of our solar system. By the end of the module, you should have a good understanding of the Unity3D editor and the workflow for creating games.

3.4 PROJECT DESCRIPTION

Roller Madness game is all about moving a rolling ball with full control avoiding the obstacles and without falling outer game surface. In this game, the player has to control the rolling ball in order to collect yellow coins. There are some cubes which make the ball bounce back. The cube boxes fall to the game surface and it tries to stop the ball. Where ever the ball moves, the boxed cubes follow and if it touches the ball then the game is over. The gameplay is not so simple, controlling the ball is a bit hard as it moves with extra force and the can dodge two or more boxed cubes in order to destroy it. After Collecting 5 coins, the level is complete. Likewise in another level, there are more obstacles with high difficulties in collecting coins

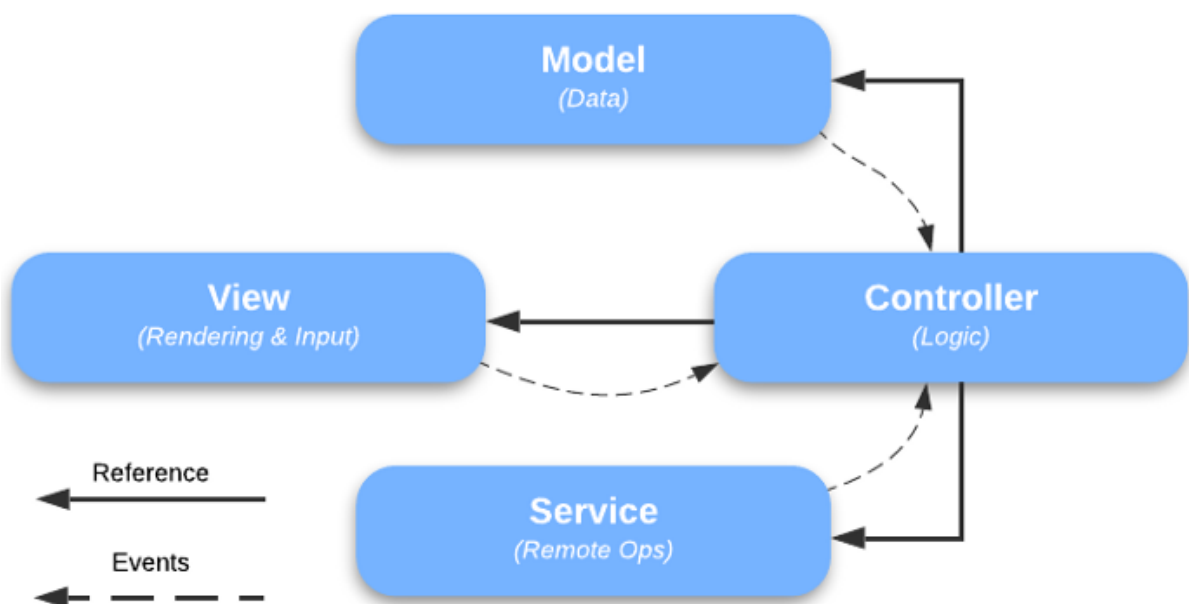
CHAPTER 4

RESULTS AND DISCUSSION

4.1 SYSTEM REQUIREMENTS

- Operating System: Windows/Mac
- RAM: 4GB
- Processor: 64x 1.0Ghz processor
- ROM: 8GB

4.2 GAME ARCHITECTURE



4.3 MODULES

BENEFITS OF MODULARIZATION

- It will be easier to debug.
- You will have reusable code.
- Readability will be increased.
- The code will be more reliable.

- Control over visibility on high level.

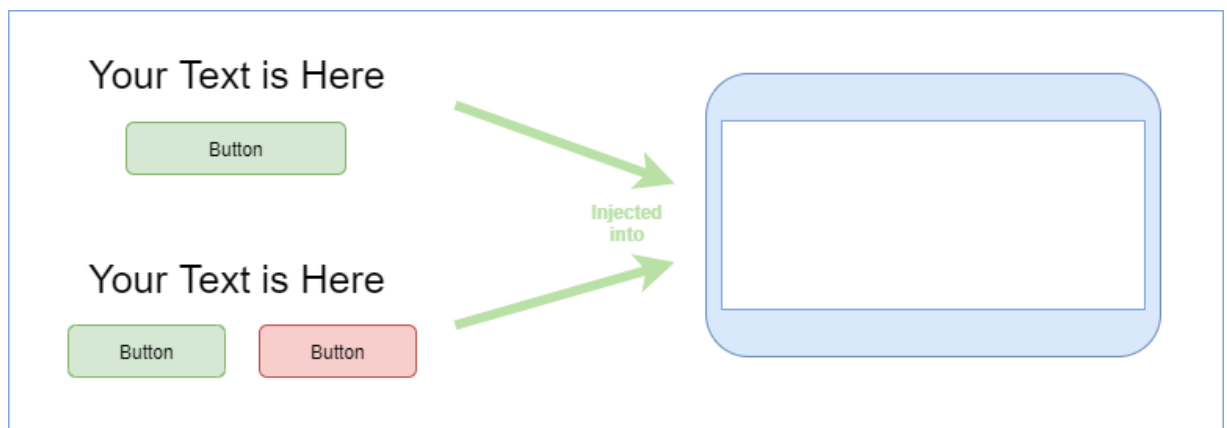
USED MODULES

Scene Module

It seems that we can divide this popup as a Panel and a Content. We can reuse the same panel and inject different contents into it. This way, we would never need to create a new panel. Here is an illustration below.



Control Manager Module



Script Module

The behaviour of GameObjects is controlled by the Components that are attached to them. Although Unity's built-in Components can be very versatile, they can provide to implement your own gameplay features. Unity allows to create our own Components using scripts. These allow to trigger game events, modify Component properties over time and respond to user input in any way we like.

4.4 RESULT

In this game, the players have to control the rolling ball in order to collect yellow coins. There are some cubes which make the ball bounce back. The cube boxes fall to the game surface and it tries to stop the ball. Where ever the ball moves, the boxed cubes follow and if it touches the ball then the game is over

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

Roller Madness Game project is developed using Unity Game Engine. The language used for the development of this project is "C#". The project file contains Assets such as C# scripts, prefabs, textures, sounds, animations, models, scenes and many more. Roller Madness is a 3D game specially designed for PCs. The gameplay Graphics is good enough for PCs and controls are simple for the users. In this game, the players have to control the rolling ball in order to collect yellow coins. There are some cubes which make the ball bounce back. The cube boxes fall to the game surface and it tries to stop the ball. Where ever the ball moves, the boxed cubes follow and if it touches the ball then the game is over

5.2 FUTURE WORK

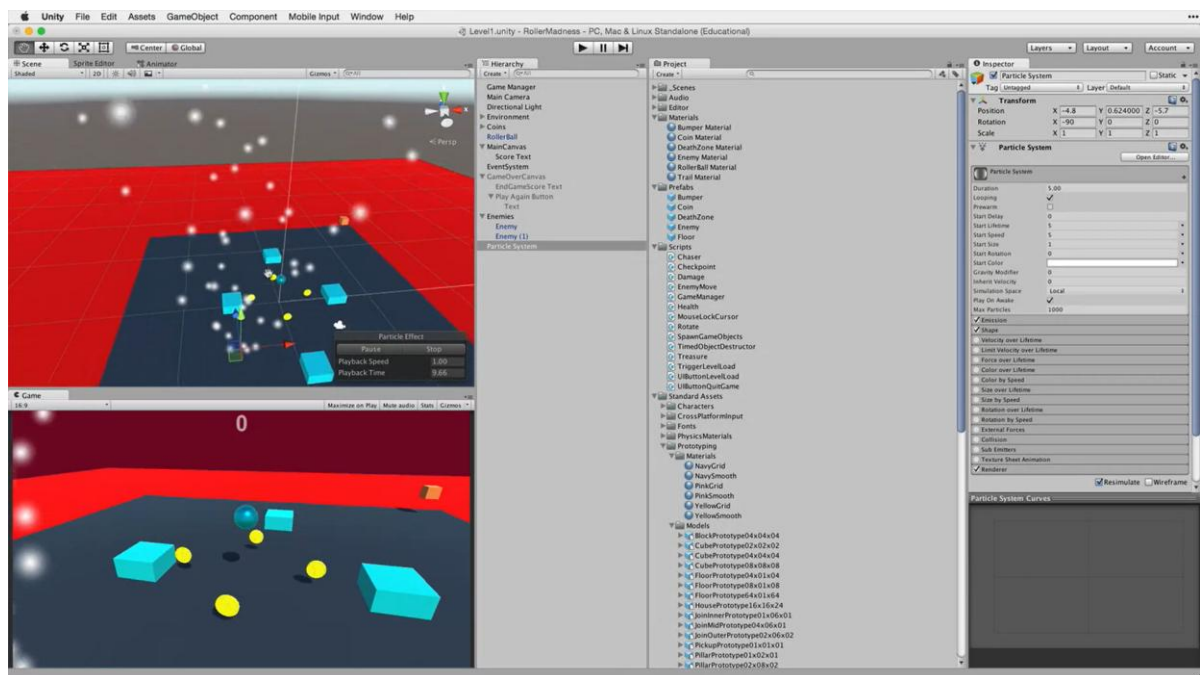
The game development industry have now become a ubiquitous part of our lives with the increasing penetration of technology. Interestingly, the cost of playing online games has plummeted owing to the affordable internet connection.

5.3 REFERENCES

1. Coursera.org

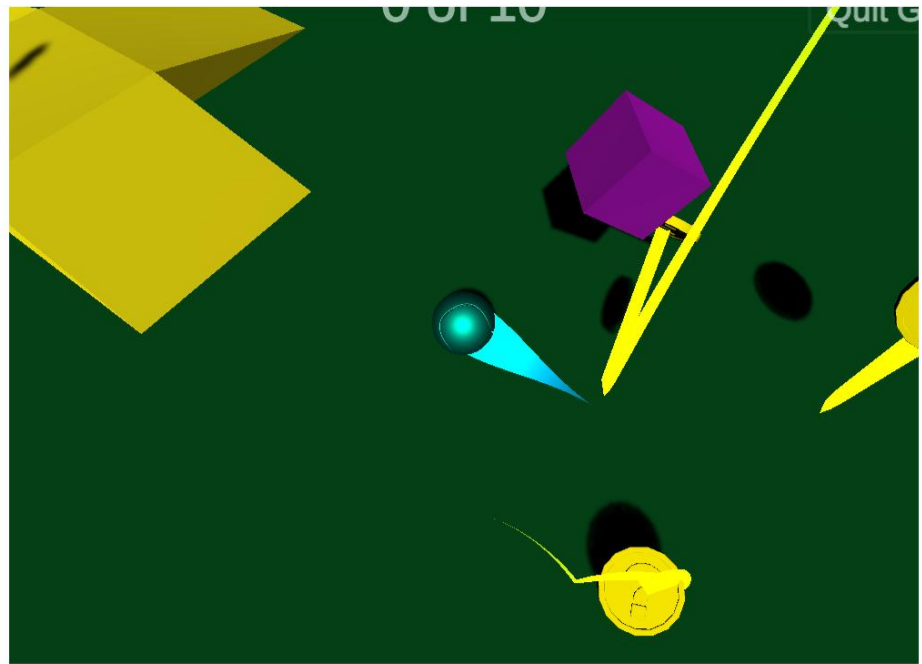
APPENDIX

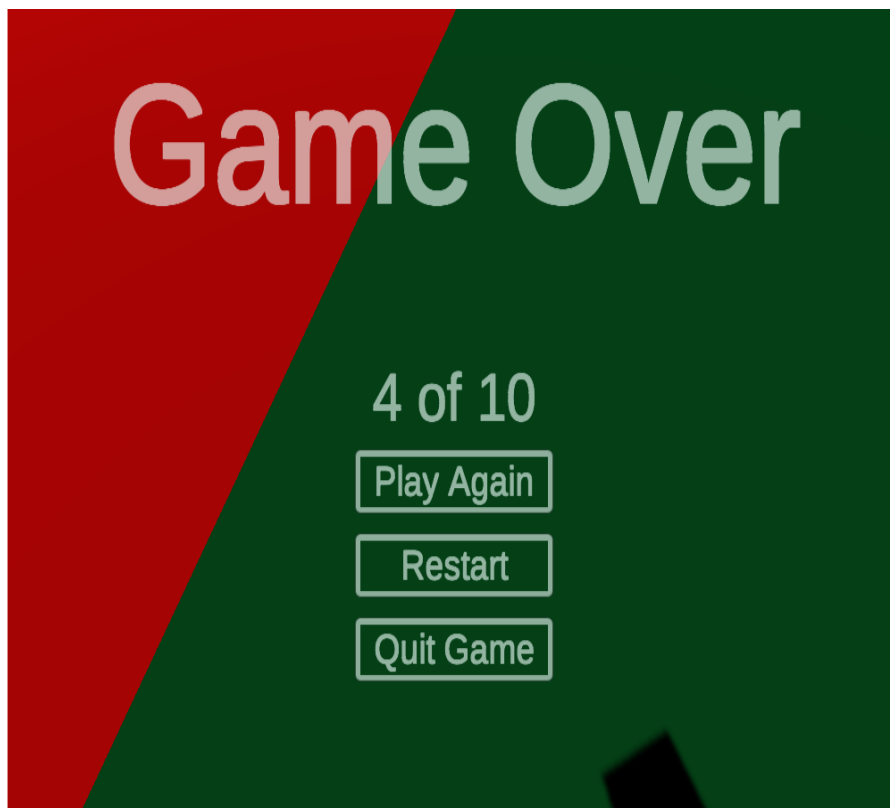
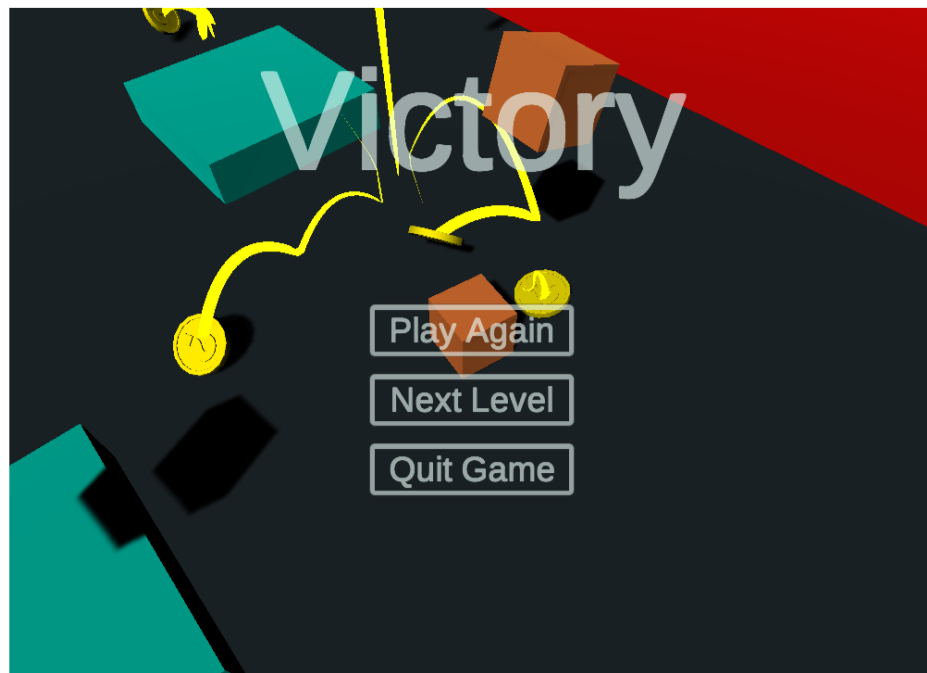
A.SCREENSHOTS





WebGL Roller Madness





B.SOURCE CODE

1. Heath.cs

```
using UnityEngine;
using System.Collections;

public class Health : MonoBehaviour {

    public enum deathAction {loadLevelWhenDead,doNothingWhenDead};

    public float healthPoints = 1f;
    public float respawnHealthPoints = 1f;          //base health points

    public int numberOfLives = 1;                    //lives and variables for respawning
    public bool isAlive = true;

    public GameObject explosionPrefab;

    public deathAction onLivesGone = deathAction.doNothingWhenDead;

    public string LevelToLoad = "";

    private Vector3 respawnPosition;
    private Quaternion respawnRotation;

    // Use this for initialization
    void Start ()
    {
        // store initial position as respawn location
        respawnPosition = transform.position;
        respawnRotation = transform.rotation;

        if (LevelToLoad=="") // default to current scene
        {
            LevelToLoad = Application.loadedLevelName;
        }
    }

    // Update is called once per frame
    void Update ()
    {
        if (healthPoints <= 0) {                      // if the object is 'dead'
            numberOfLives--;
            // decrement # of lives, update lives GUI

            if (explosionPrefab!=null) {
                Instantiate (explosionPrefab, transform.position, Quaternion.identity);
            }

            if (numberOfLives > 0) { // respawn

```

```

        transform.position = respawnPosition; // reset the player to respawn p
osition
        transform.rotation = respawnRotation;
        healthPoints = respawnHealthPoints; // give the player full health again
    } else { // here is where you do stuff once ALL lives are gone)
        isAlive = false;

        switch(onLivesGone)
        {
            case deathAction.loadLevelWhenDead:
                Application.LoadLevel (LevelToLoad);
                break;
            case deathAction.doNothingWhenDead:
                // do nothing, death must be handled in another way elsewhere
                break;
        }
        Destroy(gameObject);
    }
}

public void ApplyDamage(float amount)
{
    healthPoints = healthPoints - amount;
}

public void ApplyHeal(float amount)
{
    healthPoints = healthPoints + amount;
}

public void ApplyBonusLife(int amount)
{
    numberOfLives = numberOfLives + amount;
}

public void updateRespawn(Vector3 newRespawnPosition, Quaternion newRespawnRotation)
{
    respawnPosition = newRespawnPosition;
    respawnRotation = newRespawnRotation;
}
}

```

2. Chaser.cs

```

using UnityEngine;
using System.Collections;

//[RequireComponent(typeof(CharacterController))]

public class Chaser : MonoBehaviour {

    public float speed = 20.0f;
    public float minDist = 1f;
    public Transform target;

    // Use this for initialization

```

```

void Start ()
{
    // if no target specified, assume the player
    if (target == null) {

        if (GameObject.FindWithTag ("Player")!=null)
        {
            target = GameObject.FindWithTag ("Player").GetComponent<Transform>();
        }
    }
}

// Update is called once per frame
void Update ()
{
    if (target == null)
        return;

    // face the target
    transform.LookAt(target);

    //get the distance between the chaser and the target
    float distance = Vector3.Distance(transform.position,target.position);

    //so long as the chaser is farther away than the minimum distance, move towards
    it at rate speed.
    if(distance > minDist)
        transform.position += transform.forward * speed * Time.deltaTime;
}

// Set the target of the chaser
public void SetTarget(Transform newTarget)
{
    target = newTarget;
}
}

```

3. Damage.cs

```

using UnityEngine;
using System.Collections;

public class Damage : MonoBehaviour {

    public float damageAmount = 10.0f;

    public bool damageOnTrigger = true;
    public bool damageOnCollision = false;
    public bool continuousDamage = false;
    public float continuousTimeBetweenHits = 0;

    public bool destroySelfOnImpact = false;    // variables dealing with exploding on i
    mpact (area of effect)
    public float delayBeforeDestroy = 0.0f;
    public GameObject explosionPrefab;
}

```

```

private float savedTime = 0;

void OnTriggerEnter(Collider collision) // used for things like
bullets, which are triggers.
{
    if (damageOnTrigger) {
        if (this.tag == "PlayerBullet" && collision.gameObject.tag == "Player") // i
f the player got hit with it's own bullets, ignore it
            return;

        if (collision.gameObject.GetComponent<Health> () != null) { // if the hit ob
ject has the Health script on it, deal damage
            collision.gameObject.GetComponent<Health> ().ApplyDamage (damageAmount);

            if (destroySelfOnImpact) {
                Destroy (gameObject, delayBeforeDestroy); // destroy the object
whenever it hits something
            }

            if (explosionPrefab != null) {
                Instantiate (explosionPrefab, transform.position, transform.rotation
);
            }
        }
    }
}

void OnCollisionEnter(Collision collision) // this is used for
things that explode on impact and are NOT triggers
{
    if (damageOnCollision) {
        if (this.tag == "PlayerBullet" && collision.gameObject.tag == "Player") // i
f the player got hit with it's own bullets, ignore it
            return;

        if (collision.gameObject.GetComponent<Health> () != null) { // if the hit ob
ject has the Health script on it, deal damage
            collision.gameObject.GetComponent<Health> ().ApplyDamage (damageAmount);

            if (destroySelfOnImpact) {
                Destroy (gameObject, delayBeforeDestroy); // destroy the object
whenever it hits something
            }

            if (explosionPrefab != null) {
                Instantiate (explosionPrefab, transform.position, transform.rotation
);
            }
        }
    }
}

void OnCollisionStay(Collision collision) // this is used for damage over time thing
s
{
    if (continuousDamage) {

```

```

        if (collision.gameObject.tag == "Player" && collision.gameObject.GetComponent<Health> () != null) { // is only triggered if whatever it hits is the player
            if (Time.time - savedTime >= continuousTimeBetweenHits) {
                savedTime = Time.time;
                collision.gameObject.GetComponent<Health> ().ApplyDamage (damageAmount);
            }
        }
    }
}

```

4. Rotate.cs

```

using UnityEngine;
using System.Collections;

public class Rotate : MonoBehaviour {

    public float speed = 10.0f;

    public enum whichWayToRotate {AroundX, AroundY, AroundZ}

    public whichWayToRotate way = whichWayToRotate.AroundX;

    // Use this for initialization
    void Start () {

    }

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

        switch(way)
        {
            case whichWayToRotate.AroundX:
                transform.Rotate(Vector3.right * Time.deltaTime * speed);
                break;
            case whichWayToRotate.AroundY:
                transform.Rotate(Vector3.up * Time.deltaTime * speed);
                break;
            case whichWayToRotate.AroundZ:
                transform.Rotate(Vector3.forward * Time.deltaTime * speed);
                break;
        }
    }
}

```

5.GameManager.cs

```

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

public class GameManager : MonoBehaviour {

```

```

public static GameManager gm;

[Tooltip("If not set, the player will default to the gameObject tagged as Player.")]
public GameObject player;

public enum gameStates {Playing, Death, GameOver, BeatLevel};
public gameStates gameState = gameStates.Playing;

public int score=0;
public bool canBeatLevel = false;
public int beatLevelScore=0;

public GameObject mainCanvas;
public Text mainScoreDisplay;
public GameObject gameOverCanvas;
public Text gameOverScoreDisplay;

[Tooltip("Only need to set if canBeatLevel is set to true.")]
public GameObject beatLevelCanvas;

public AudioSource backgroundMusic;
public AudioClip gameOverSFX;

[Tooltip("Only need to set if canBeatLevel is set to true.")]
public AudioClip beatLevelSFX;

private Health playerHealth;

void Start () {
    if (gm == null)
        gm = gameObject.GetComponent<GameManager>();

    if (player == null) {
        player = GameObject.FindWithTag("Player");
    }

    playerHealth = player.GetComponent<Health>();

    // setup score display
    Collect (0);

    // make other UI inactive
    gameOverCanvas.SetActive (false);
    if (canBeatLevel)
        beatLevelCanvas.SetActive (false);
}

void Update () {
    switch (gameState)
    {
        case gameStates.Playing:
            if (playerHealth.isAlive == false)
            {
                // update gameState
                gameState = gameStates.Death;

                // set the end game score
            }
        }
    }
}

```

```

        gameOverScoreDisplay.text = mainScoreDisplay.text;

        // switch which GUI is showing
        mainCanvas.SetActive (false);
        gameOverCanvas.SetActive (true);
    } else if (canBeatLevel && score>=beatLevelScore) {
        // update gameState
        gameState = gameStates.BeatLevel;

        // hide the player so game doesn't continue playing
        player.SetActive(false);

        // switch which GUI is showing
        mainCanvas.SetActive (false);
        beatLevelCanvas.SetActive (true);
    }
    break;
case gameStates.Death:
    backgroundMusic.volume -= 0.01f;
    if (backgroundMusic.volume<=0.0f) {
        AudioSource.PlayClipAtPoint (gameOverSFX,gameObject.transform.positi
on);

        gameState = gameStates.GameOver;
    }
    break;
case gameStates.BeatLevel:
    backgroundMusic.volume -= 0.01f;
    if (backgroundMusic.volume<=0.0f) {
        AudioSource.PlayClipAtPoint (beatLevelSFX,gameObject.transform.posit
ion);

        gameState = gameStates.GameOver;
    }
    break;
case gameStates.GameOver:
    // nothing
    break;
}

}

public void Collect(int amount) {
    score += amount;
    if (canBeatLevel) {
        mainScoreDisplay.text = score.ToString () + " of " + beatLevelScore.ToString (
);
    } else {
        mainScoreDisplay.text = score.ToString ();
    }
}
}
}

```

6. Treasure.cs

```

using UnityEngine;
using System.Collections;

public class Treasure : MonoBehaviour {

    public int value = 10;
    public GameObject explosionPrefab;

    void OnTriggerEnter (Collider other)
    {
        if (other.gameObject.tag == "Player") {
            if (GameManager.gm!=null)
            {
                // tell the game manager to Collect
                GameManager.gm.Collect (value);
            }

            // explode if specified
            if (explosionPrefab != null) {
                Instantiate (explosionPrefab, transform.position, Quaternion.identity);
            }

            // destroy after collection
            Destroy (gameObject);
        }
    }
}

```

7. SpawnGameObjects.cs

```

using UnityEngine;
using System.Collections;

public class SpawnGameObjects : MonoBehaviour {

    public GameObject spawnPrefab;

    public float minSecondsBetweenSpawning = 3.0f;
    public float maxSecondsBetweenSpawning = 6.0f;

    public Transform chaseTarget;

    private float savedTime;
    private float secondsBetweenSpawning;

    // Use this for initialization
    void Start () {
        savedTime = Time.time;
        secondsBetweenSpawning = Random.Range (minSecondsBetweenSpawning, maxSecondsBetweenSpawning);
    }

    // Update is called once per frame
    void Update () {
        if (Time.time - savedTime >= secondsBetweenSpawning) // is it time to spawn again?
        {

```

```

        MakeThingToSpawn();
        savedTime = Time.time; // store for next spawn
        secondsBetweenSpawning = Random.Range (minSecondsBetweenSpawning, maxSeconds
BetweenSpawning);
    }
}

void MakeThingToSpawn()
{
    // create a new gameObject
    GameObject clone = Instantiate(spawnPrefab, transform.position, transform.rotati
on) as GameObject;

    // set chaseTarget if specified
    if ((chaseTarget != null) && (clone.gameObject.GetComponent<Chaser> () != null))
    {
        clone.gameObject.GetComponent<Chaser>().SetTarget(chaseTarget);
    }
}
}

```

8.TriggerLevelLoad.cs

```

using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class TriggerLevelLoad : MonoBehaviour {

    public string nameOfLevelToLoad = "";

    // Use this for initialization
    void Start () {

    }

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

    }

    void OnTriggerEnter (Collider other)
    {
        if(other.gameObject.tag == "Player" )
        {
            SceneManager.LoadScene(nameOfLevelToLoad);
        }
    }
}

```

9. UIButtonLevelToLoad.cs

```

using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class UIButtonLevelLoad : MonoBehaviour {

```

```
public string LevelToLoad;

public void loadLevel() {
    //Load the level from LevelToLoad
    SceneManager.LoadScene(LevelToLoad);
}
}
```

10. UIButtonQuitGame.cs

```
using UnityEngine;
using System.Collections;

public class UIButtonQuitGame : MonoBehaviour {

    public void quitGame()
    {
        //Closes the game
        Application.Quit();
    }
}
```