

Game Development (Platformer game Development)

A Project Report

Submitted in partial fulfillment of the
Requirements for the award of the Degree of

BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)

By

Pawan Pramod Yadav

Roll Number: 1901421

Under the esteemed guidance of

Mrs. Rupali Jawale

(IT co-ordinator)



DEPARTMENT OF INFORMATION TECHNOLOGY

S.D.T. KALANI COLLEGE

(Affiliated to University of Mumbai)

ULHASNAGAR-421001

MAHARASHTRA

2019-2020

PROFORMA FOR THE APPROVAL PROJECT PROPOSAL

(Note: All entries of the proforma of approval should be filled up with appropriate and complete information. Incomplete proforma of approval in any respect will be summarily rejected.)

PRN No.:

Roll No: _____

1. Name of the Student

2. Title of the Project

3. Name of the Guide

4. Teaching experience of the

Guide _____

5. Is this your first submission?

Yes ☐

No ☐

Signature of the Student

Signature of the Guide

Date:

Date:

Signature of the Coordinator

Date:

S.D.T. KALANI COLLEGE (*Affiliated to*
University of Mumbai) **ULHASHNAGAR-**
MAHARASHTRA-421001

DEPARTMENT OF INFORMATION TECHNOLOGY



CERTIFICATE

This is to certify that the project entitled, “**Game Development (Platformer game Development)**”, is bonafied work of **Pawan Yadav** bearing Roll No: **(1901421)** submitted in partial fulfillment of the requirements for the award of degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

Internal Guide

Coordinator

External Examiner

Date:

College Seal

Abstract

A platform game is a subgenre of video game and can be of two types such as a puzzle game or an action game. The main concept of a 2D platformer is that the player will control a character or avatar and guide him through a series of platforms which can be grounded or elevated depending on the level design. The mission of such games is always to collect certain items (coins, rubies etc.) and avoid or fight the enemy characters. In this game we are creating a beautiful environment which will consists of grounded as well as air suspended platforms and a system to collect coins and multiple potions which will change the character or avatars behavior.

Through the development of this game we are trying to demonstrate the combination of creative thinking and problem solving by programming an exciting experience, for this we are going to use the plethora of functions and features provided by the Unity3D engine and some image and audio editing software. This game will be supported on platforms such as the latest versions of Windows. In future we will also continue the development of this project and make it available for android devices as well.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose ceaseless Cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. We are grateful to our project guide **Prof. Rupali Jawale** for the guidance, inspiration and constructive suggestions that helped us in the preparation of this Project.

DECLARATION

I hereby declare that the project entitled, “**Game Development (Platformer game Development)**” done at **S.D.T. KALANI COLLEGE**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of knowledge other than me, no one has submitted to any other university.

The Project is done in partial fulfillment of the requirements for the award of degree of **BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)** to be submitted as **semester V** project as part of our examination.

Name and Signature of the Student

TABLE OF CONTENTS

Chapter 1: Introduction	01
1.1 Background	02
1.2 Objectives	03
1.3 Purpose, Scope and Applicability	04
1.3.1 Purpose	04
1.3.2 Scope	05
1.3.3 Applicability	05
1.4 Achievements	05
 Chapter 2: Survey of Technologies	06
2.1 Existing System	06
2.2 Proposed System	06
2.3 Requirements Analysis	07
2.4.1 Functional Requirements	07
2.4 Hardware Requirements	08
2.5 Software Requirements	08
2.6 Justification of selection of technology	08
 Chapter 3: System Design	09
3.1 Basic Modules	10
3.2 Data Dictionary	11
3.3 ER Diagram	12
3.4 DFD/UML Diagrams	14

Chapter 4: Implementation and Testing	23
4.1 Code	23
4.2 Testing Approach	50
4.2.1 Unit Testing	51
4.2.2 Integration Testing	52
4.3 Gantt Chart	53
 Chapter 5: Results and Dissuptions	 54
 Chapter 6: Conclusion and Future Work	 70
 Chapter 7: References	 71

List of Figures

Sr. No.	Content	Page No
1	waterfall model	09
2	ER Diagram of Game	13
3	DFD Diagram of Starting a Game	15
4	DFD Diagram of Save & Load	15
5	DFD Diagram of Gamer interaction with game	16
6	Class Diagram	17
7	Use-Case Diagram	19
8	Activity Diagram	21
9	Sequence diagram	22
10	Gantt chart of 2D Platformer Game Development	53

LIST OF TABLES

Sr. No.	Content	Page No
1	Data Dictionary for Save and Load	11
2	Game general test case table	51
3	Game detailed test case table	52

Chapter 1

Introduction

Game development is the process of using graphics libraries, game engines and image editing software to display images on the screen and let the player manipulate the images on the screen to create experiences and tell the player a story.

“Platformers” or „platform games“ are games that mainly revolve around a character controlled by the player, which runs and jumps to avoid obstacles and/or to defeat enemies. Platformers are often classified as a sub-genre of action games, and are considered to be one of the first game genres. Despite being one of the first game genres, platform games have retained their popularity throughout the years. Platformers can be categorized into two distinct types; single screen platformers and scrolling-platformers.

Single screen platform games display each level as a single static screen. These games can have multiple levels, which become increasingly difficult as the player progresses. Most of the old arcade platformer games are single screen platformers due to technical limitations, for example, games like Donkey Kong and Burgertime.

Scrolling platformers can be identified by the scrolling game screen as the playable character approaches the edge of the game screen. Just like single screen platformers, scrolling platformers can have multiple increasingly difficult levels and can also feature boss levels. For example Castlevania, Sonic the Hedgehog and Super Mario Bros are scrolling platform games.

1.1 Background

Platform games originated in the early 1980s, which were often about climbing ladders as much as jumping, with 3D successors popularized in the mid-1990s. The term describes games where jumping on platforms is an integral part of the gameplay and came into use after the genre had been established, no later than 1983. The genre is frequently combined with elements of other genres, such as the shooter elements in *Contra*, Beat 'em up elements of *Viewtiful Joe*, adventure elements of *Flashback*, or role-playing game elements of *Castlevania: Symphony of the Night*.

While commonly associated with console gaming, there have been many important platform games released to video arcades, as well as for handheld game consoles and home computers. North America, Europe and Japan have played major parts in the genre's evolution. Platform themes range from cartoon-like games to science fiction and fantasy epics.

At one point, platform games were the most popular genre of video game. At the peak of their popularity, it is estimated that between one-quarter and one-third of console games were platformers, but have since been supplanted by first-person shooters. As of 2006, the genre had become far less dominant, representing a two percentage market share as compared to 15.0% in 1998, but is still commercially viable, with a number of games selling in the millions of units. Since 2010 a variety of endless running platformers for mobile devices have brought renewed popularity to the genre.

1.2 Objectives

To make a Side scrolling platformer game in which the user takes control of the character where the character performs following platformer movements like run, jump, attack, etc. To complete the game the player must traverse through levels. The levels are properly and carefully designed and give the game a new Life.

The game is based on an entertaining story where the player is the main character which is thrown into the dungeon and the dungeon is locked. To get out of the dungeon player must traverse through levels & clear the dungeon by defeating the Enemies of the dungeon to get to the outside world. The dungeon is full of monster's, trap's, etc. The higher level the player gets to the higher the difficulty rises accordingly.

The player character can perform platformer movements like run, idle, jump, double jump, attack, etc. In the game there are many types of enemies like patrolled, stationed & targeting enemies.

Following are the major objectives behind the new proposed system.

- To provide a simple 2D Adventures Game.
- To provide Realistic experience
- Great sounds effects

1.3 Purpose, Scope And Applicability

1.3.1 Purpose

The purpose of the project is to develop a 2D platformer game which anyone can play regardless of the age & gender. Another purpose is to develop a fun game which people can play & relieve their stress & have fun in their free time.

The purpose of any game is first to benefit all those who have opted-in. Even though the game may seem dysfunctional from the outside, the players all get something from playing in it.

Aim for Project:

The project that we have undertaken aims to develop a 2D Adventures game. The project “Game Development (Platformer Game Development)” includes the following functionalities:

- Realistic Experience
- 2D view
- Good Graphics
- Faster performance
- Offline Game
- No need for the Internet.

1.3.2 Scope

The scope of our Game will be limited to the functional design of the original Platformer game, along with our own selection of additions and modifications to the same. It will support saving and loading the states of games in progress, and also the ability to save or abandon a game in progress.

1.3.3 Applicability

- Makes boring free time fun.
- Training and improvement of mental abilities.
- Reduces stress.
- Makes playing fun.
- Provides better gaming experience.
- Makes you active again after a stressful day of work.

1.4 Achievements

- All the goals are achieved in this project successfully.

Chapter 2

Survey Of Technologies

2.1 Existing System

In the current system the platformer games are using the old graphics & old physics engine. In some cases the developer has to develop a new physics for the game which can be time consuming and sometimes it is filled with bugs. So the physics in game & graphics does not provide a realistic experience and in some games saving the game data is not present so playing & completing the game can be a headache sometimes.

2.1.1 Limitations of Current System:

- It is an offline game.
- No multiplayer mode supported.
- Saving user data is not present.
- The graphics & physics engine are old so the game does not provide a realistic experience.

2.3 Proposed System

To overcome the drawbacks of the existing system the proposed system has been evolved. This project aims to develop a game with realistic physics & graphics. It also help in reducing the complexity of game by providing proper & easy to use controls.

- The proposed system has three main aspects.
 - Scenes
 - Character
 - Storyline

It is an offline 2d Adventure game. The game is based on a player story, where the

player finds itself inside a Dungeon. The main aim of the player is to survive and find a way to get out of the Dungeon.

- Either player loss or win.
- As the game name suggests, The player will make a way to get out of the Dungeon.

2.3.1 Advantages of Proposed System:

- **GUI:** it does provide 2d graphics.
- **Controls:** Game provides easy controls
- **User-friendly:** The system should be user-friendly.
- **Experience:** Game provides realistic experience.

2.4 Requirement Analysis

2.4.1 Functional Requirements

- A functional requirement describes what a Game system should do.
- Functional requirements specify a function that a system or system component must be able to perform.
- It can be documented in various ways. The most common ones are written descriptions in documents and use cases.
- The user must be able to start the game & be able to choose options from the main menu.
- When clicking on play the game must start & load the scene.
- The game should run properly and controls must work properly without any errors & bugs.
- The user can select audio on/off from option menu & see controls options.
- **Player:** The player character should be able to perform 2d movements & actions like run, range attack, idle, etc.

- he user must be able to save and load the game as required.
- The user must be able to exit the game by choosing the Quit option from the menu.

2.5 Hardware Requirements

CPU	SSE2 instruction set support
Memory	4 GB Ram
Hard drive	Minimum 10 GB Free Space
Graphic Hardware	Graphics card with DX10 (shader model 4.0) capabilities

2.6 Software Requirements

Operating system	Windows 10 64 bit
Programming Language	C#
IDE	Microsoft Visual Studio 2019
Game Engine	UNITY
Database(Backend)	SQL
Image Editor/Creator	Gimp

2.7 Justification of Selection of Technology

- Everything is going on smoothly by using all the above H/W and S/W.
- The game is exported for **Windows Platform** but in future it can also be made available for **Android devices**.

Chapter 3

System Design

The 2D Platformer Game is designed using the **Waterfall Model** approach. The waterfall model is a sequential design process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance.

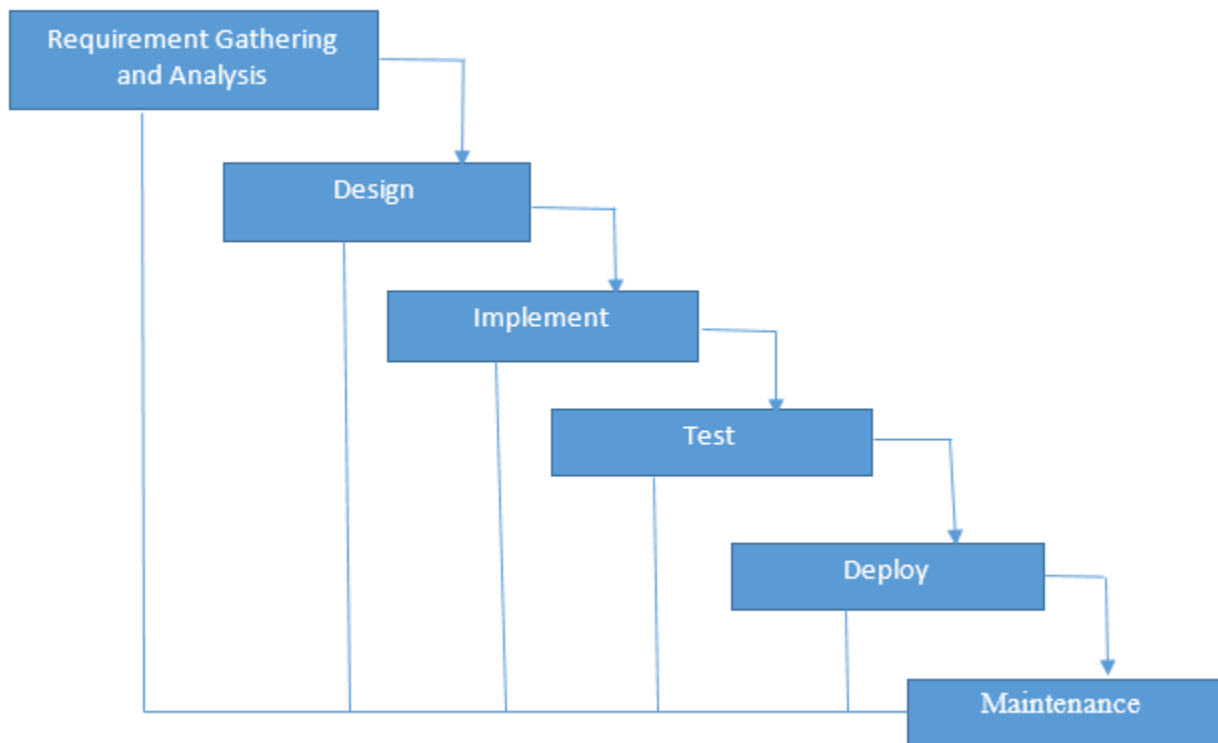


Fig 3.0 waterfall model

3.1 Basic Modules

Modules are used for dividing up the program into smaller, more easily understood, reusable parts. A module consists a single block of code that can be invoked in the way that a procedure, function, or method is invoked.

Advantages of modularizations:

- Smaller Component are easier to maintain
- Program can be dived based on functional aspect.
- Desired level of abstraction can be re-used again.
- Concurrent execution can be made possible.
- Desired for Security Aspect.

Following are the modules in the Platformer Game Development:

- 1) **Save:** The user can save the game data.
- 2) **Load:** The user can load the save data of the game from the no of saves.
- 3) **Settings:** The setting module includes the settings (sound, music & language).
- 4) **Menu:** It contains the character Status and shortcut of other modules (store, status, skills, settings, etc...).
- 5) **Map:** It contains the mini map of the current level.
- 6) **Controller:** The controller module includes buttons with which the user can control the player.
- 7) **Help:** It helps to give the Instructions and hints about the game.
- 8) **Quest:** It contains the current Quest (Task or Mission) which the player has to complete to clear the game.

3.2 Data Dictionary

A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary holds records about other objects in the database, such as data ownership, data relationships to other objects, and other data. Only database administrators interact with the data dictionary.

➤ Following are the table of the Platformer Game System:

Table for Save and Load:

Mygame.s3db







Field Name	Data Type	Constraints
save_id	INTEGER	PRIMARY KEY, NOT NULL
health	INTEGER	NOT NULL
coins	INTEGER	NOT NULL
level	VARCHAR(20)	NOT NULL
player_pos_x	FLOAT	NOT NULL
player_pos_y	FLOAT	NOT NULL
player_pos_z	FLOAT	NOT NULL
saved_bool	BOOLEAN	NOT NULL

3.3 ER Diagram

The ER or (Entity Relational Model) is a high-level conceptual data model diagram. Entity-Relation model is based on the notion of real-world entities and the relationship between them.

Entity relationship diagram displays the relationships of entity set stored in a database. In other words, we can say that ER diagrams help you to explain the logical structure of databases. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.

❖ Components of ER Diagram

Symbol	Name
	Entity
	Attribute
	Relationship
	Weak Entity
	Weak Attribute
	Weak Relationship

❖ Diagrams

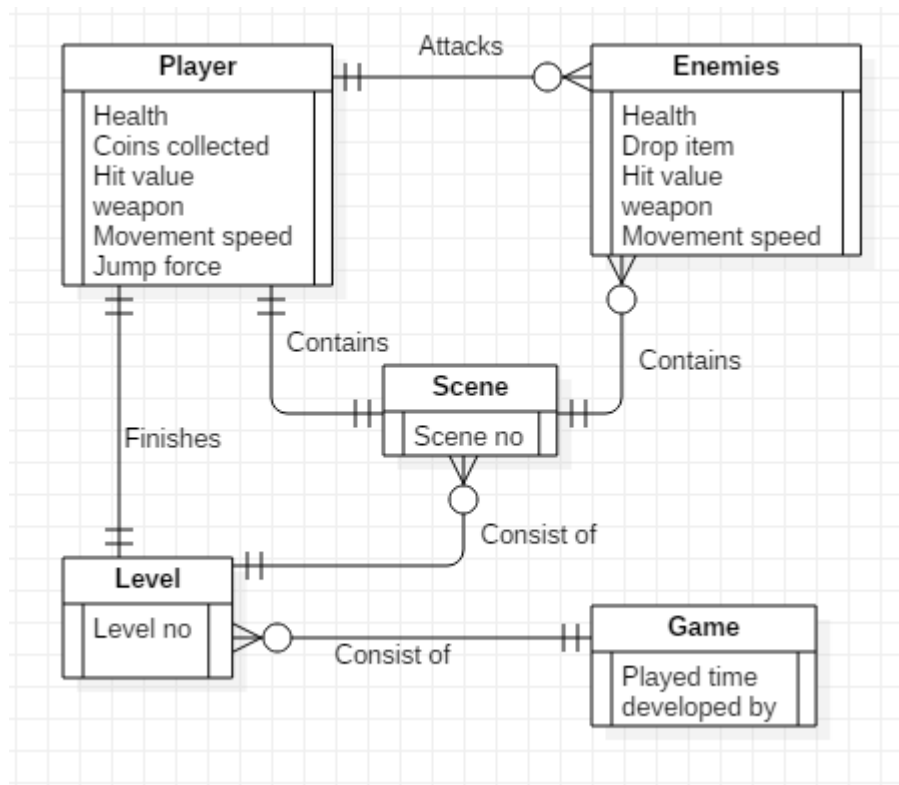


Fig 3.3.1 ER Diagram of Game

3.4 DFD/UML Diagrams

Data flow diagram is a graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Types of DFD: Data Flow Diagrams are either Logical or Physical.

- Logical DFD - This type of DFD concentrates on the system process, and flow of data in the system.
- Physical DFD - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and closer to the implementation.

❖ DFD Components

DFD can represent Source, destination, storage and flow of data using the following set of components -



- **Entities:** Entities are the source and destination of information data. Entities are represented by a rectangle with their respective names.
- **Process:** Activities and action taken on the data are represented by Circle or Round edged rectangles.
- **Data Storage:** There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow:** Movement of data is shown by pointed arrows. Data movement is shown from

the base of the arrow as its source towards the head of the arrow as destination.

❖ Diagrams

A. DFD Diagram of Starting a Game:

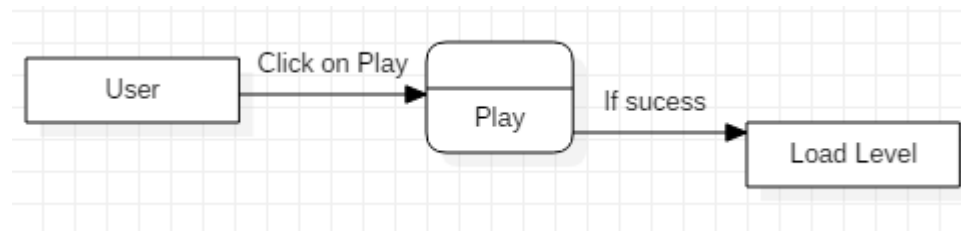


Fig 3.4.1 DFD Level 0

B. DFD Diagram of Save & Load:

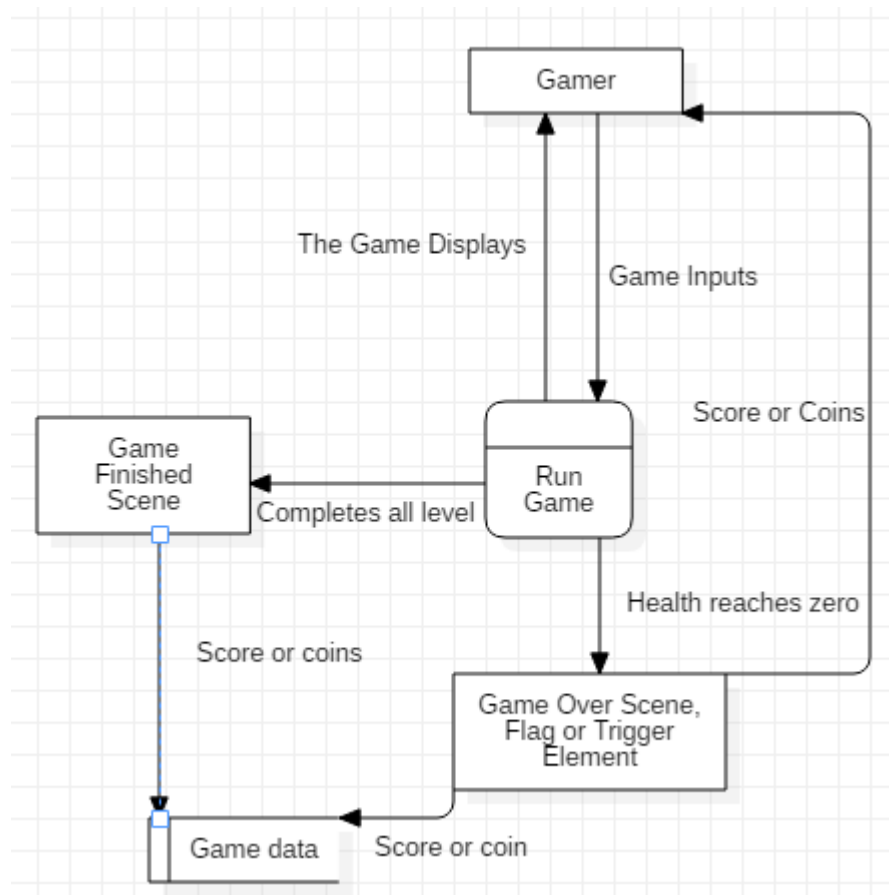


Fig 3.4.2 DFD Level 1

C. DFD Diagram of Gamer interaction with game:

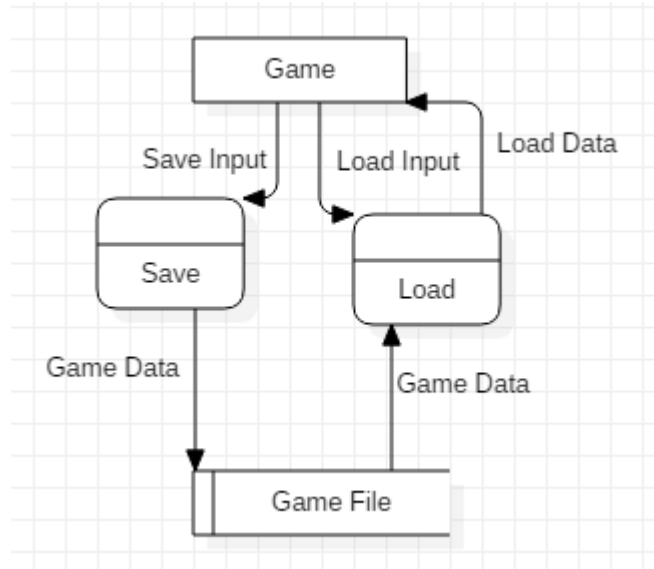


Fig 3.4.3 DFD of Gamer interaction with game

Procedural Design

Procedural design is best used to model programs that have an obvious flow of data from input to output. It represents the architecture of a program as a set of interacting processes that pass data from one to another.

Logic Diagrams

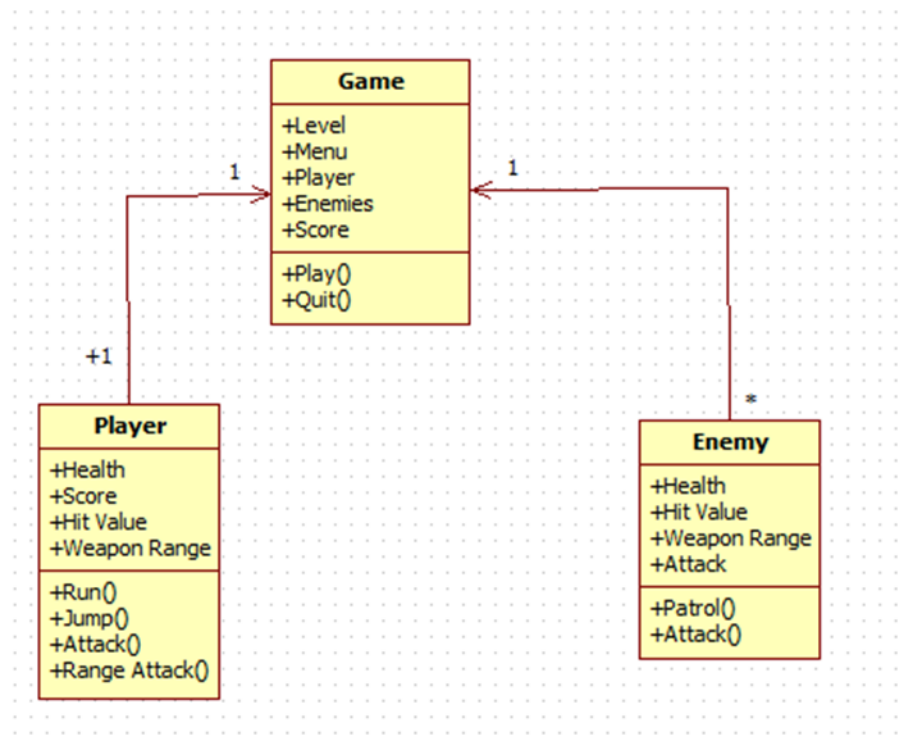


Fig 3.4.4 Class diagram

User interface design

The graphical user interface is presented (displayed) on the computer screen. It is the result of processed user input and usually the primary interface for human-machine interaction. The touch user interfaces popular on small mobile devices are an overlay of the visual output to the visual input.

The visual part of a computer application or operating system through which a client interacts with a computer or software. It determines how commands are given to the computer or the program and how data is displayed on the screen.

Use case Diagram

A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the term "system" refers to something being developed or operated, such as a mail-order product sales and service Web site. Use case diagrams are employed in UML (Unified Modelling Language), a standard notation for the modelling of real-world objects and systems.

➤ Use case notations

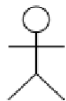

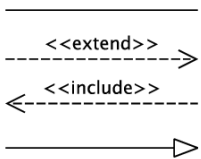
Symbol	Reference Name
	Actor
	Use case
	Relationship

Diagram:

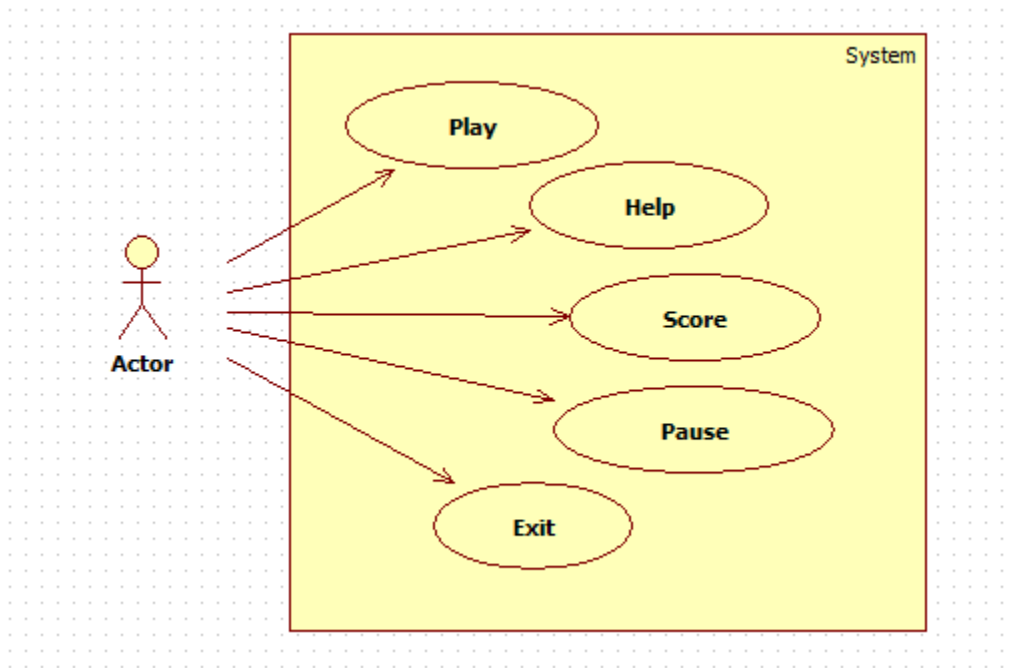


Fig 3.4.5 Use-Case Diagram

PLAY Subsystem:

- a) This subsystem allow user to interact with games.
- b) This subsystem also provides interaction with actor.
- c) This subsystem provides GUI to user.
- d) This subsystem is a combination of story, result and action object.

HELP Subsystem:

- a) This subsystem provides help regarding the game.
- b) This subsystem provides control information.
- c) Help the user to control the actor.

EXIT Subsystem:

- a) This Subsystem allows user to quit game.

Activity Diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

➤ Activity notations





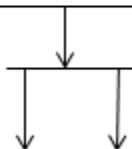
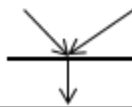

Sr. No	Name	Symbol
1.	Start Node	
2.	Action State	
3.	Control Flow	
4.	Decision Node	
5.	Fork	
6.	Join	
7.	End State	

Diagram:

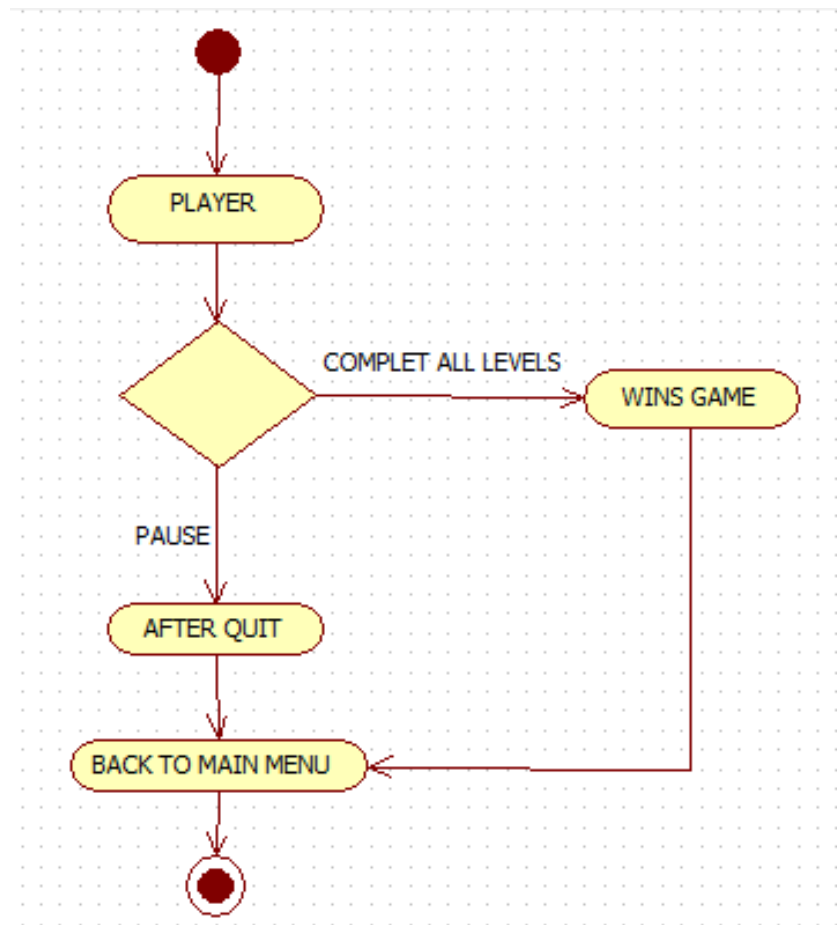


Fig 3.4.6 Activity Diagram

Sequence diagram

The sequence diagram shows object interaction arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagram are typically associated with Use-Case realization in the logical view of the system under development. Sequence diagram are sometimes called event diagram or event scenarios.

➤ Sequence notations

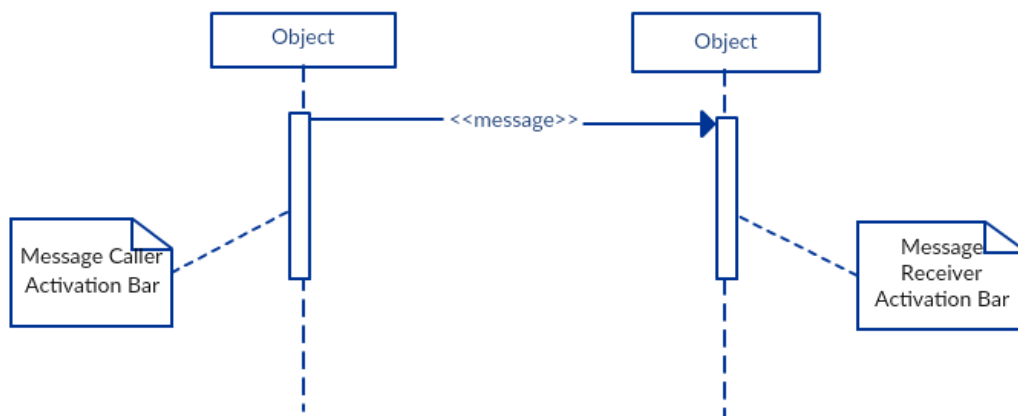


Diagram:

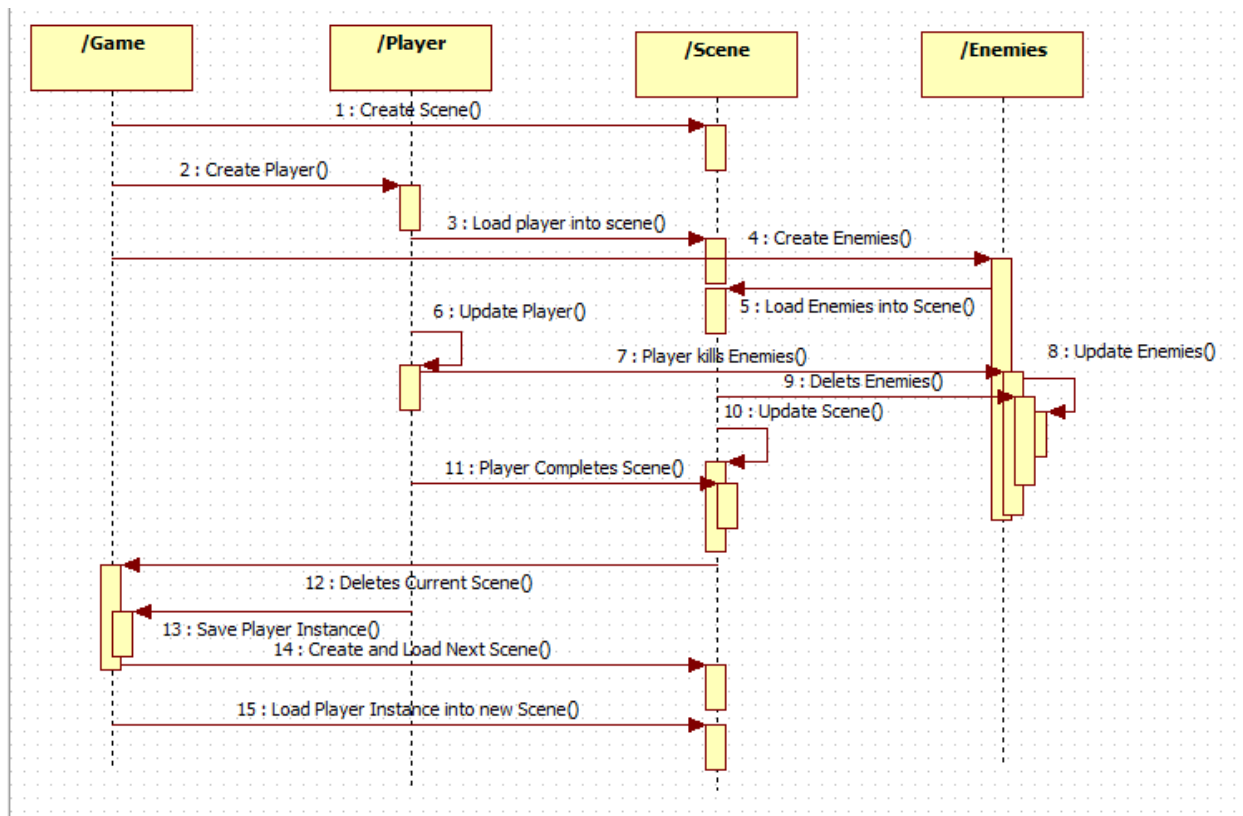


Fig 3.4.7 Sequence Diagram

Chapter 4

Implementation And Testing

4.1 Code

1. CharacterComponents.cs

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

public abstract class CharacterComponents : MonoBehaviour
{
    AudioManager am;
    public Rigidbody2D rb;
    public float speed;
    public float jumpForce;
    public float fallMultiplier = 2.5f;
    public float lowJumpMultiplier = 2f;
    public bool isGrounded = false;
    public Transform isGroundedChecker;
    public float checkGroundRadius;
    public LayerMask groundLayer;
    public Transform ShootPoint;
    public float rememberGroundedFor;
    float lastTimeGrounded;
    public GameObject jumpFx;
    public GameObject playerExplodFx;
    public GameObject Bullet;
    public int defaultAdditionalJumps = 1;
    int additionalJumps;
    public bool facingRight;
    public int health;
    public int maxHealth = 100;
    public bool dead = false;
    public int coin = 0;

    void Start()
    {
        am =
        GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
        health = maxHealth;
        rb = GetComponent<Rigidbody2D>();
        dead = false;
```

```

        //Bullet = GameObject.FindGameObjectWithTag("Bullet");
        facingRight = true;
        additionalJumps = defaultAdditionalJumps;
    }
    public void Jump()
    {
        if (Input.GetKeyDown(KeyCode.Space) && (isGrounded || Time.time -
lastTimeGrounded <= rememberGroundedFor || additionalJumps > 0))
        {
            rb.velocity = new Vector2(rb.velocity.x, jumpForce);
            additionalJumps--;
            Instantiate(jumpfx, transform.position, transform.rotation);
            am.Play("Jump");
        }
    }
    public void Move()
    {
        float x = Input.GetAxisRaw("Horizontal");

        if (Input.GetAxis("Horizontal") < 0 && facingRight ||
Input.GetAxis("Horizontal") > 0 && !facingRight)
        {
            changingDirection();
        }

        float moveBy = x * speed;

        rb.velocity = new Vector2(moveBy, rb.velocity.y);
    }
    public virtual void changingDirection()
    {
        facingRight = !facingRight;
        transform.localScale = new Vector3(transform.localScale.x,
transform.localScale.y, transform.localScale.z);
        if (facingRight)
            transform.eulerAngles = new Vector3(0, 0, 0);
        else
            transform.eulerAngles = new Vector3(0, -180, 0);
    }
    public void BetterJump()
    {
        if (rb.velocity.y < 0)
        {
            rb.velocity += Vector2.up * Physics2D.gravity * (fallMultiplier - 1) *
Time.deltaTime;
        }
    }

```

```

        else if (rb.velocity.y > 0 && !Input.GetKey(KeyCode.Space))
        {
            rb.velocity += Vector2.up * Physics2D.gravity * (lowJumpMultiplier - 1) *
Time.deltaTime;
        }
    }
    public void CheckIfGrounded()
    {
        Collider2D colliders = Physics2D.OverlapCircle(isGroundedChecker.position,
checkGroundRadius, groundLayer);

        if (colliders != null)
        {
            isGrounded = true;
            additionalJumps = defaultAdditionalJumps;
        }
        else
        {
            if (isGrounded)
            {
                lastTimeGrounded = Time.time;
            }
            isGrounded = false;
        }
    }

}
public void AttackHandler()
{
    if (Input.GetKeyDown(KeyCode.Z))
    {
        shoot();
    }
}

}
public void shoot()
{
    if (facingRight)
    {
        am.Play("Shoot");
        //GameObject tmp = (GameObject)Instantiate(Bullet, new
Vector3(transform.position.x, transform.position.y, transform.position.z),
Quaternion.Euler(new Vector3(0, 0, -90)));
        GameObject tmp = (GameObject)Instantiate(Bullet, ShootPoint.position,
Quaternion.Euler(new Vector3(0, 0, -90)));
        Instantiate(Resources.Load<GameObject>("Prefab/GunMuzzle") as
GameObject, ShootPoint.position, transform.rotation);
    }
}

```

```

        tmp.GetComponent<Bullet>().Initialize(Vector2.right);
    }
    else
    {
        am.Play("Shoot");
        //GameObject tmp = (GameObject)Instantiate(Bullet, new
Vector3(transform.position.x, transform.position.y, transform.position.z),
Quaternion.Euler(new Vector3(0, 0, 90)));
        GameObject tmp = (GameObject)Instantiate(Bullet, ShootPoint.position,
Quaternion.Euler(new Vector3(0, 0, 90)));
        tmp.GetComponent<Bullet>().Initialize(Vector2.left);
    }
}
public void TakeDamage(int damage)
{
    health -= damage;
}

public void check()
{
    if (health <= 0)
    {
        if (dead == false)
            Destroy(gameObject);
        am.Play("Explosion");
        Instantiate(playerExploadFx, transform.position, transform.rotation);
        Debug.Log("dead");
        health = 0;
        dead = true;
    }
    if (health > 100)
    {
        health = maxHealth;
    }
}
}

```

2. PlayerController.cs

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

public class PlayerController : CharacterComponents
{
    void Update()
    {
        Move();
        Jump();
        BetterJump();
        CheckIfGrounded();
        AttackHandler();
        check();
    }
}
```

3. Sound.cs

```
using UnityEngine;
using UnityEngine.Audio;
[System.Serializable]
public class Sound {

    public string name;

    public AudioClip clip;

    [Range(0, 1f)]
    public float volume;
    [Range(0, 1)]
    public float pitch;

    public bool loop;

    [HideInInspector]
    public AudioSource source;

}
```

4. AudioManager.cs

```
using UnityEngine.Audio;
using UnityEngine;
using System;
using UnityEngine.SceneManagement;

public class AudioManager : MonoBehaviour
{
    public Sound[] sounds;
    public static AudioManager instance;
    public float music=1;
    public float sfx=1;
    public string[] _music = {"Bg","Intro"};
    public string[] _sfx = {
"Key","Jump","Shoot","Pickup_coin","Powerup","Hurt","Hit" };
    void Awake()
    {
        if(instance==null)
        {
            instance=this;
        }
        else
        {
            Destroy(gameObject);
            return;
        }
        DontDestroyOnLoad(gameObject);
        foreach (Sound s in sounds) {
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;
            s.source.loop = s.loop;
            s.source.volume = s.volume;
            s.source.pitch = s.pitch;
        }
    }
    public void Start()
    {
        if (SceneManager.GetActiveScene().name == "mainmenu")
        {
            Play("Intro");
            Debug.Log("Intro playing");
        }
    }
    public void Play(string name)
    {
        Sound s=Array.Find(sounds,sound=>sound.name==name);
```

```

        if(s==null)
        {
            Debug.LogWarning("Sound: "+name+"not Found");
            return;
        }
        s.source.Play();
        Debug.Log("'" + s.name + " = " + s.source.isPlaying.ToString());
    }
    public void Stop(string name)
    {
        Sound s=Array.Find(sounds,sound=>sound.name==name);
        if(s==null)
        {
            Debug.LogWarning("Sound: "+name+"not Found");
            return;
        }
        s.source.Stop();
    }
    public void Pause(string name)
    {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        if (s == null)
        {
            Debug.LogWarning("Sound: " + name + "not Found");

            return;
        }
        s.source.Pause();
    }
    public void KeyPress()
    {
        Play("Key");
    }
    public void audioOption()
    {
        music = PlayerPrefs.GetFloat("music");
        sfx = PlayerPrefs.GetFloat("sfx");
        foreach (Sound s in sounds)
        {
            foreach(string m in _music)
            {
                if (m.Equals(s.name))
                {
                    s.source.volume = music;
                    s.volume = music;
                }
            }
            foreach (string sf in _sfx)

```

```

        {
            if (sf.Equals(s.name))
                s.source.volume = sfx;
                s.volume = sfx;
        }
    }
}

```

5. Scenemanager.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Scenemanager : MonoBehaviour
{
    LevelLoader LL;
    Transform SpawnPoint;
    PlayerController player;
    public bool dontDestroy = false;
    public bool load_ = false;
    public int h = 0;
    public int c=0;
    public string l="";
    public Vector3 pos = new Vector3(0,0,0);
    public void Awake()
    {
        LL=GameObject.FindGameObjectWithTag("LL").GetComponent<LevelLoader>();
    }
    void Update()
    {
        if (dontDestroy)
        {
            DontDestroyOnLoad(GameObject.FindGameObjectWithTag("SC"));
        }
        if (load_)
        {
            DB dB;
            dB = GameObject.FindGameObjectWithTag("SC").GetComponent<DB>();
            dB.dbRead();
            Debug.Log("load");
            h = dB.h;
            c = dB.c;
        }
    }
}

```



```

        l = dB.l;
        pos = dB.pos;
        if (load_)
        {
            SceneManager.LoadScene(l);
            load_ = false;
        }
    }
}
public void load()
{

    SpawnPoint = GameObject.FindGameObjectWithTag("SpawnPoint").transform;
    player.health = h;
    player.coin = c;
    player.transform.position = pos;
    Debug.Log("loaded");
    Destroy(gameObject);

}
private void OnLevelWasLoaded(int level)
{
    if (SceneManager.GetActiveScene().name != "mainmenu")
    {
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
        load();
    }
}
}

```

6. GameManager.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class GameManager : MonoBehaviour
{
    PlayerController player;
    Transform SpawnPoint;
    AudioManager AM;
    // Start is called before the first frame update
    void Start()
    {
        AM =
GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();

```

```

    player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    AM.Stop("Intro");
    AM.Play("Bg");
}

// Update is called once per frame
void Update()
{
    if (player.dead || player.health == 0)
    {
        //PlayerPrefs.SetInt("savecoin",1);
        //PlayerPrefs.SetInt("coin",player.coin);
        Invoke("reload", 2);
    }
}
public void reload()
{
    string scene = SceneManager.GetActiveScene().name;
    SceneManager.LoadScene(scene);
    //LL.LoadLevel(SceneManager.GetActiveScene().buildIndex);
    //SpawnPoint = GameObject.FindGameObjectWithTag("SpawnPoint").transform;
    //GameObject.FindGameObjectWithTag("Player").transform.position =
SpawnPoint.position;
}
public void save()
{
    int h = player.health;
    int c = player.coin;
    string l = SceneManager.GetActiveScene().name;
    float x = player.transform.position.x;
    float y = player.transform.position.y;
    float z = player.transform.position.z;
    bool s = true;
    DB dB;
    dB = GameObject.FindGameObjectWithTag("GM").GetComponent<DB>();
    dB.DBinsert(h,c,l,x,y,z,s);
}
}

```

7. DB.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Data;
using Mono.Data.Sqlite;
using System.IO;

public class DB : MonoBehaviour
{
    public int h = 0;
    public int c = 0;
    public string l = "";
    public Vector3 pos = new Vector3(0, 0, 0);
    public string database = "/game.s3db";
    // Start is called before the first frame update
    void Start()
    {
        TBcreate();
    }
    public void TBcreate()
    {
        string connection = Application.dataPath + database;
        Debug.Log(connection);
        IDbConnection dbcon = new SqliteConnection("Data Source = " + connection);
        dbcon.Open();
        IDbCommand dbcmd;
        IDataReader reader;
        dbcmd = dbcon.CreateCommand();
        string q_createTable = "CREATE TABLE IF NOT EXISTS [mygame] ([save_id]
INTEGER NOT NULL PRIMARY KEY,[health] INTEGER NOT NULL,[coins]
INTEGER NOT NULL,[level] VARCHAR(20) NOT NULL,[player_pos_x] FLOAT
NOT NULL,[player_pos_y] FLOAT NOT NULL,[player_pos_z] FLOAT NOT
NULL,[saved_bool] BOOLEAN NOT NULL)";
        dbcmd.CommandText = q_createTable;
        reader = dbcmd.ExecuteReader();
        dbcon.Close();
    }
    public void DBinsert(int h,int c,string l,float x, float y,float z,bool s)
    {
        DBdelete();TBcreate();
        string connection = Application.dataPath + database;
        IDbConnection dbcon = new SqliteConnection("Data Source = " + connection);
        dbcon.Open();
        IDbCommand dbcmd;
        IDataReader reader;
```

```

        dbcmd = dbcon.CreateCommand();
        string q_insert = "insert into
mygame(save_id,health,coins,level,player_pos_x,player_pos_y,player_pos_z,saved_bo
ol)" +
        "values(1,"+h+", "+c+", "+l+", "+x+ ", " + y + ", " + z + ", " +s+"");";
        dbcmd.CommandText = q_insert;
        reader = dbcmd.ExecuteReader();
        dbcon.Close();
    }
    public void DBdelete()
    {
        string connection = Application.dataPath + database;
        IDbConnection dbcon = new SqlConnection("Data Source = " + connection);
        dbcon.Open();
        IDbCommand dbcmd;
        IDataReader reader;
        dbcmd = dbcon.CreateCommand();
        string q_insert = "drop table mygame; ";
        dbcmd.CommandText = q_insert;
        reader = dbcmd.ExecuteReader();
        dbcon.Close();
    }
    public void DBupdate()
    {
        DBdelete();
        TBcreate();
    }
    public void dbRead()
    {
        string connection = Application.dataPath + database;
        IDbConnection dbcon = new SqlConnection("Data Source = " + connection);
        dbcon.Open();
        IDbCommand dbcmd;
        IDataReader reader;
        dbcmd = dbcon.CreateCommand();
        string q_read = "select * from mygame;";
        dbcmd.CommandText = q_read;
        reader = dbcmd.ExecuteReader();
        while (reader.Read())
        {
            h = int.Parse(reader[1].ToString());
            c = int.Parse(reader[2].ToString());
            l = reader[3].ToString();
            pos = new Vector3(float.Parse(reader[4].ToString()),
float.Parse(reader[5].ToString()), float.Parse(reader[6].ToString()));
        }
    }

```

```

        dbcon.Close()
    }
}

```

8. Menucontrol.cs

```

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

public class Menucontrol : MonoBehaviour
{
    AudioManager am;
    LevelLoader LL;
    public bool paused=false;
    //am =
    GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
    public GameObject pauseMenu;
    public void Awake()
    {
        if(SceneManager.GetActiveScene().buildIndex==0)
        {
            LL=GameObject.FindGameObjectWithTag("LL").GetComponent<LevelLoader>();
        }
    }
    private void Start()
    {
        am =
        GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
    }
    private void Update()
    {
        if
        (Input.GetKeyDown(KeyCode.Escape)&&(SceneManager.GetActiveScene().name!="
        mainmenu")&&!paused)
        {
            am =
            GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
            am.KeyPress();
            Pause();
            paused=true;
        }
    }
    public void NewGame()

```

```

{
    LL.LoadLevel(SceneManager.GetActiveScene().buildIndex + 1);
}
public void ContinueGame()
{
    Scenemanager sc;
    Debug.Log("continue");
    sc =
GameObject.FindGameObjectWithTag("SC").GetComponent<Scenemanager>();
    sc.dontDestroy=true;
    sc.load_ = true;
}

public void QuitGame()
{
    Debug.Log("Quit");
    Application.Quit();
}
public void Pause()
{
    paused=true;
    Debug.Log("paused");
    pauseMenu.SetActive(true);
    Time.timeScale = 0;
}
public void Resume()
{
    paused=false;
    Debug.Log("Continue "+paused);
    pauseMenu.SetActive(false);
    Time.timeScale = 1;
}
public void Restart()
{
    Time.timeScale = 1;
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
public void ExitGame()
{
    GameManager gm;
    gm =
GameObject.FindGameObjectWithTag("GM").GetComponent<GameManager>();
    gm.save();
    Time.timeScale = 1;
    am.Stop("Bg");
    SceneManager.LoadScene("mainmenu");
}

```

```

        //LL.LoadLevel(0);// 0 is for mainmenu
    }
    private void OnLevelWasLoaded()
    {
        if (SceneManager.GetActiveScene().name == "mainmenu")
        {
            am=
GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
            am.Play("Intro");
        }
    }
}

```

9. Optioncontrol.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Optioncontrol : MonoBehaviour
{
    [SerializeField]
    public Slider musSlider;
    public Toggle musToggle;
    public Slider sfxSlider;
    public Toggle sfxToggle;
    AudioManager am;

    void Start()
    {
        am =
GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
        if (gameObject.active == true)
            check();
    }
    public void OnSliderChange()
    {
        PlayerPrefs.SetFloat("music", musSlider.value);
        PlayerPrefs.SetFloat("sfx", sfxSlider.value);
        am.audioOption();
    }
    public void OnToggle()
    {
        if(sfxToggle.isOn)
        {
            PlayerPrefs.SetFloat("sfx",1);
            sfxSlider.gameObject.SetActive(true);

```

```

        sfxSlider.value = 1;
    }
    else
    {
        PlayerPrefs.SetFloat("sfx",0);
        sfxSlider.gameObject.SetActive(false);
    }

    if(musToggle.isOn)
    {
        PlayerPrefs.SetFloat("music",1);
        musSlider.gameObject.SetActive(true);
        musSlider.value = 1;
    }
    else
    {
        PlayerPrefs.SetFloat("music",0);
        musSlider.gameObject.SetActive(false);
    }
    am.audioOption();
}
public void check()
{
    musSlider.value = am.music;
    if (am.music == 0)
    {
        musToggle.isOn = false;
    }
    sfxSlider.value = am.sfx;
    if (am.sfx == 0)
    {
        sfxToggle.isOn = false;
    }
}
}

```


10. Bullet.cs

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

public class Bullet : MonoBehaviour
{
    public Vector2 direction;
    [SerializeField]
    public float speed=10;
    public Rigidbody2D myrigidbody;
    public GameObject bulletExplodeFX;
    AudioManager am;
    void Start()
    {
        am =
GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
        myrigidbody = GetComponent<Rigidbody2D>();
    }
    private void FixedUpdate()
    {
        myrigidbody.velocity = direction * speed;
        Destroy(gameObject, 2f);
    }
    public void Initialize(Vector2 direction)
    {
        this.direction = direction;
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        Instantiate(bulletExplodeFX, transform.position, transform.rotation);
        am.Play("Explosion");
        Destroy(gameObject);
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Enemy"))
        {
            Instantiate(bulletExplodeFX, transform.position, transform.rotation);
            am.Play("Explosion");
            Debug.Log("Enemy");
            Destroy(gameObject);
            //Destroy(collision.gameObject); //enemy.TakeDamage(damage);
        }
    }
}
```

11. buttonEvent.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class buttonEvent : MonoBehaviour
{
    AudioManager am;
    public Button _button;
    void Start()
    {
        am =
GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
        _button = gameObject.GetComponent<Button>();
        _button.onClick.AddListener(click);

    }
    void click()
    {
        AudioManager am;
        am =
GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
        am.KeyPress();
        Debug.Log("Clicked");
    }
}
```

12. CoinCollector.cs

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

public class CoinCollector : MonoBehaviour
{
    private PlayerController player;
    [SerializeField]
    public Text text;
    private void Start()
    {
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }

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

```

    {
        text.text = (player.coin).ToString();
    }
}

```

13. Collectable.cs

```

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

public class Collectable : MonoBehaviour
{
    private PlayerController player;
    private AudioManager am;

    void Start()
    {
        am =
        GameObject.FindGameObjectWithTag("AM").GetComponent<AudioManager>();
        player =
        GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            //play collection animation
            am.Play("Pickup_coin");//play collection music
            player.coin += 1;
            Destroy(gameObject);
            Instantiate(Resources.Load<GameObject>("Prefab/CoinFx")as
            GameObject,transform.position,transform.rotation);
        }
    }
}

```

14. CompleteGame.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
public class CompleteGame : MonoBehaviour
{
    public Text BlinkText;
    Color[] c;
    void Start()
    {
    }
    // Update is called once per frame
    void Update()
    {
        BlinkText.CrossFadeAlpha(0, 2, true);
    }
    public void GotoMainMenu()
    {
        SceneManager.LoadScene(0);
    }
}
```

15. damage.cs

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

public class damage : MonoBehaviour
{
    private PlayerController player;
    public int damage_ = 10;

    // Use this for initialization
    void Start()
    {
        player =
        GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }
    // Update is called once per frame
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {

```

```

        Debug.Log("player damaged");
        player.TakeDamage(damage_);
    }
}
}

```

16. DeadZone.cs

```

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

public class DeadZone : MonoBehaviour
{
    private PlayerController player;
    private void Start()
    {
        player =
        GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            player.health = 0;
        }
    }
}

```

17. DestoryoverTime.cs

```

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

public class DestroyoverTime : MonoBehaviour
{
    public float time = 2;
    void Start()
    {
        Destroy(gameObject, time);
    }
}

```

18. EnemyHealth.cs

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

public class EnemyHealth : MonoBehaviour
{
    public int health = 50;
    public int damage = 20;
    public GameObject enemyExplodeFx;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Bullet"))
            TakeDamage();
    }
    void TakeDamage()
    {
        health -= damage;
    }
    private void Update()
    {
        if(health<=0)
        {
            Instantiate(enemyExplodeFx, transform.position, transform.rotation);
            Destroy(gameObject);
        }
    }
}
```

19. HealthBar.cs

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

public class HealthBar : MonoBehaviour
{
    private PlayerController player;
    public Slider Health;

    // Start is called before the first frame update
    void Start()
    {
        player =
        GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }
}
```

```

// Update is called once per frame
void Update()
{
    Health.value = player.health;

}
}

```

20. LevelLoader.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
public class LevelLoader : MonoBehaviour
{
    [SerializeField]
    public Slider slider;
    public GameObject loadingscreen;
    public Text progressText;
    public void LoadLevel(int sceneIndex)
    {
        StartCoroutine(LoadAsynchronously(sceneIndex));
    }
    IEnumerator LoadAsynchronously(int sceneIndex)
    {
        AsyncOperation
operation=SceneManager.LoadSceneAsync(sceneIndex);
        loadingscreen.SetActive(true);
        while(!operation.isDone)
        {
            float progress=Mathf.Clamp01(operation.progress / 0.9f);
            slider.value = progress;
            progressText.text=progress * 100 + "%";
            yield return null;
        }
    }
}

```

21. MinMapCameraFollow.cs

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

public class MinMapCameraFollow : MonoBehaviour
{
    PlayerController player;
    private void Start()
    {
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }
    private void Update()
    {
        if(player.health>0)
        {
            transform.position = player.transform.position;
        }
    }
}
```

22. Patrol.cs

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

public class Patrol : MonoBehaviour
{
    public float speed;
    public float distance;
    public bool movingRight = true;
    public Transform groundDetection;
    public bool stop = false;
    PlayerController player;
    public int damage = 30;

    private void Start()
    {
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }
    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector2.right * speed * Time.deltaTime);
```



```

RaycastHit2D groundInfo = Physics2D.Raycast(groundDetection.position,
Vector2.down, distance);
if (groundInfo.collider == false)
{
    if (movingRight == true)
    {
        transform.eulerAngles = new Vector3(0, -180, 0);
        movingRight = false;
    }
    else
    {
        transform.eulerAngles = new Vector3(0, 0, 0);
        movingRight = true;
    }
}
if (stop)
{
    speed = 0;
}
}

void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        //take damage
        player.TakeDamage(damage);
    }
}
}

```

23. Spawn.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Spawn : MonoBehaviour
{
    //[SerializedField]
    public Transform[] spawnpoint;
    public float time=1.5f;
    public GameObject[] enemies;
    public int noE=0;
    public int noSp=0;
    public float countenemies=0,maxenemies=15;
    public PlayerController player;
}

```

```

public bool finalspawn=false;

// Start is called before the first frame update
void Start()
{
    if(finalspawn)
    {
        StartCoroutine(SpawnAnEnemy());
        noE=enemies.Length;
        noSp=spawnpoint.Length;
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
    }

}

IEnumerator SpawnAnEnemy()
{
    Vector2 Spawnpos=spawnpoint[Random.Range(0,noSp)].position;
    Instantiate(enemies[0],Spawnpos,Quaternion.identity);
    yield return new WaitForSeconds(time);
    if((countenemies!=maxenemies)&&(player.health!=0))
        StartCoroutine(SpawnAnEnemy());
    countenemies+=1;
    if(countenemies>maxenemies)
    {
        Debug.Log("Load");
        yield return new WaitForSeconds(1);

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

```

24. teleport.cs

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

public class teleport : MonoBehaviour
{
    PlayerController player;
    public int h = 0;
    public int c = 0;
    public Transform SpawnPoint;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
            h = player.health;
            c = player.coin;
            DontDestroyOnLoad(gameObject);
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        }
    }
    private void OnLevelWasLoaded(int level)
    {
        if (SceneManager.GetActiveScene().name.Equals("12"))
        {
            SpawnPoint =
GameObject.FindGameObjectWithTag("SpawnPoint").transform;
            GameObject.FindGameObjectWithTag("Player").transform.position =
SpawnPoint.position;
            player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
            player.health = h;
            player.coin = c;
            Destroy(gameObject);
        }
    }
}
```

4.2 Testing Approach

What is Test Case?

“A test case has a component that describes an input, action or event expected response, to determine if feature of an application is working correctly.”

Software testing can be stated as the process of validating and verifying that computer program/application/languages:

- Meets the requirements that guided its design and development.
- Works as expected.
- Can be implemented with the same characters.
- And satisfies the needs of stakeholders.

Why we Write test case??

A test case in software Engineering is a set of conditions or variable under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do.

The basic objective to write test case is to validate testing coverage for the application.

Test cases bring some sort of standardization and minimize the ad-hoc approach in testing.

Test Case to check if game is working properly or not:

Purpose	Check whether game is working properly or not.
Assumption	Assumption that must be met before the test case can be run.
Test Data	Play and observe a game.
Steps	Steps to carry out the test. See step formatting rules below: 1. Open a Game. 2. Play a Game.
Expected Results	Game is working properly
Actual results:	Test Data: Play and Complete all levels..
Comments:	If user completes one level, he jumps to next level.

Test Case to check Events of game:

- **Event:** Occurrence happening at a determinable time & place with/without human agents.
- **Trigger:** An occurrence that tells the system about the events
- **Source:** An external agent or actor that supplies information.
- **Activity:** Behavior that the system provides.
- **Response:** An output produced by the system that gives information.
- **Destination:** An external agents or actor that receives data from the system.

Sr. No.	Event	Trigger	Source	Activity	Response	Destination
1	Player starts a game	Player click on game icon	player	Game starts	Game start	Game
2	Player click on play button	Player created	player	Level 1 starts	Game initialization	Game
3	Level Transition	Player go from one level to other level	player	Level completion	Game Level changes	Game
4	Player Wins	Player completes level	player	Game completes	Game ends	Game
5	Player Loses	Player dies	player	Game over	Game ends	Game
6	Exit	Player Exits	player	Game closes	Game closes	System

4.3 Gantt Chart

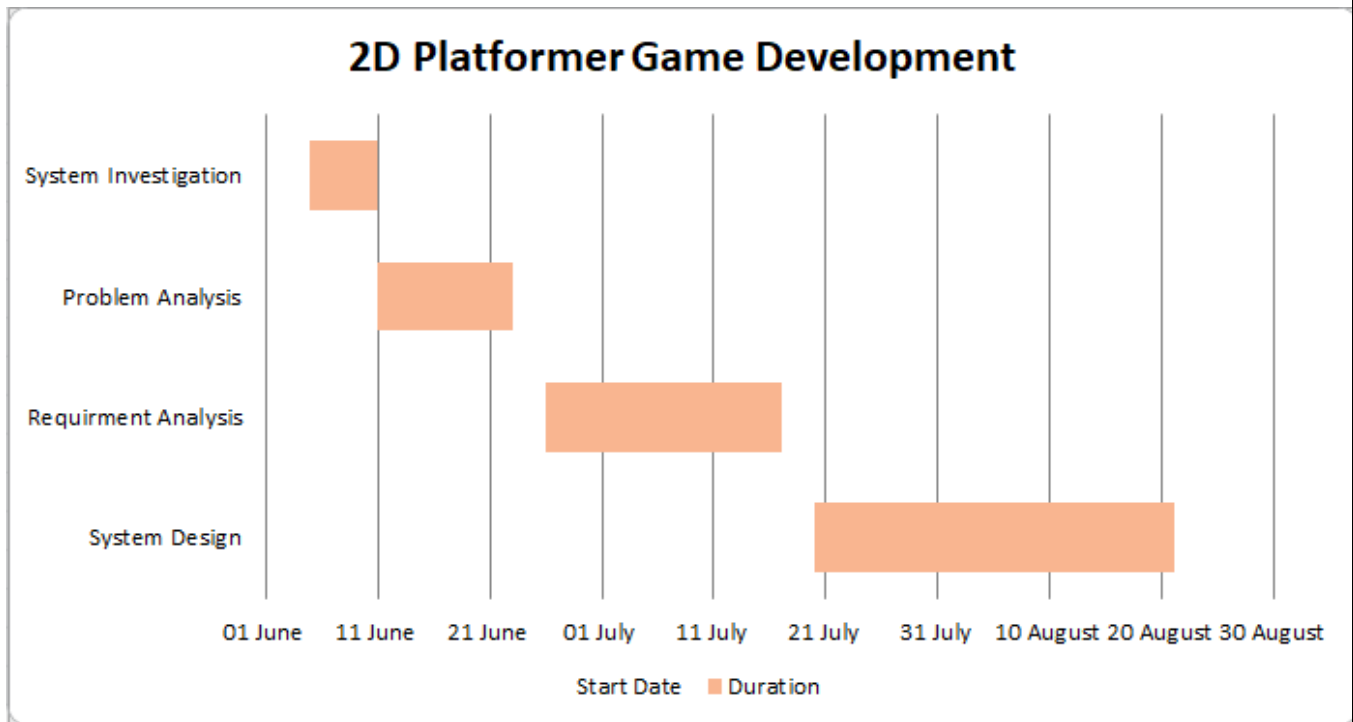


Fig 4.3.1 Gantt chart of 2D Platformer Game Development

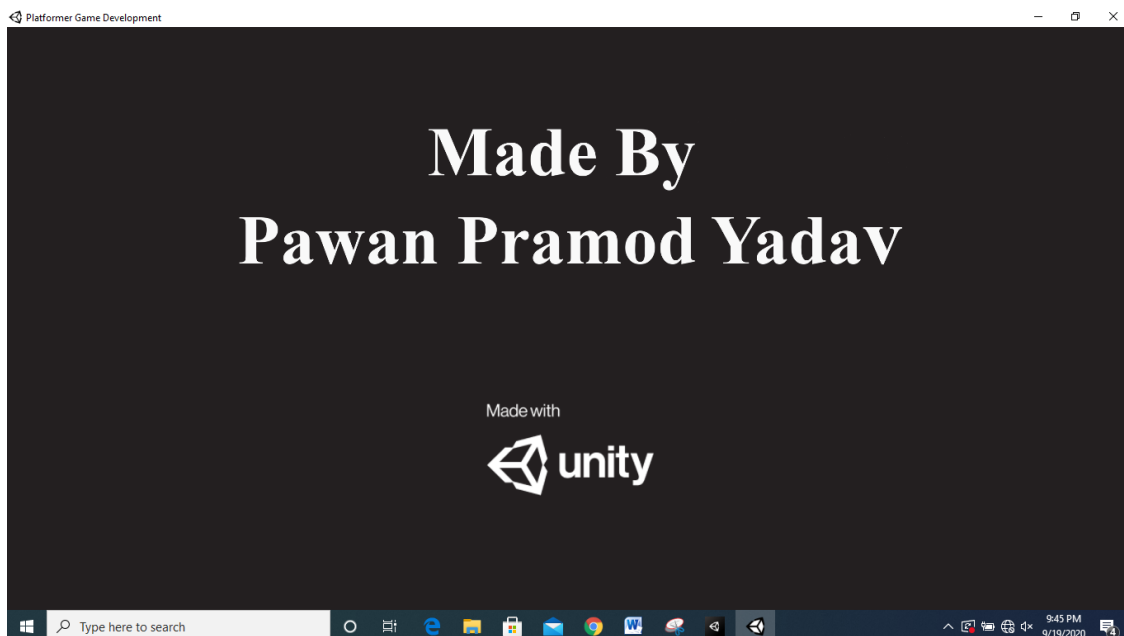
Chapter 5

Results And Discussions

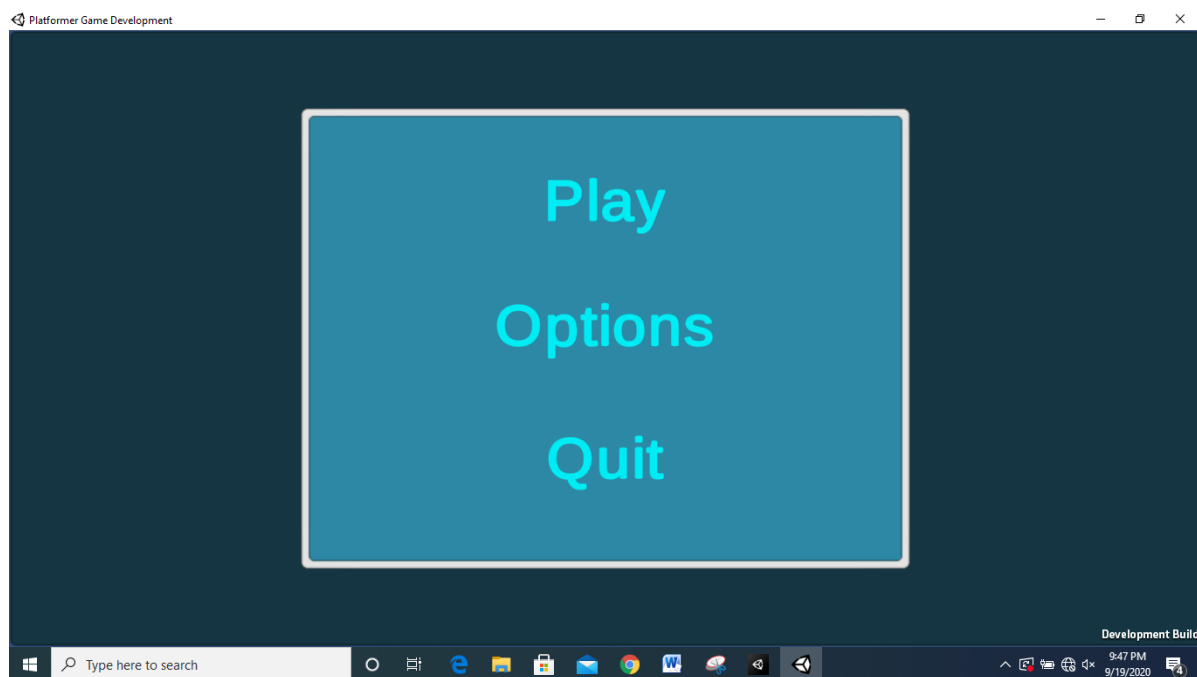
Game Title Screen



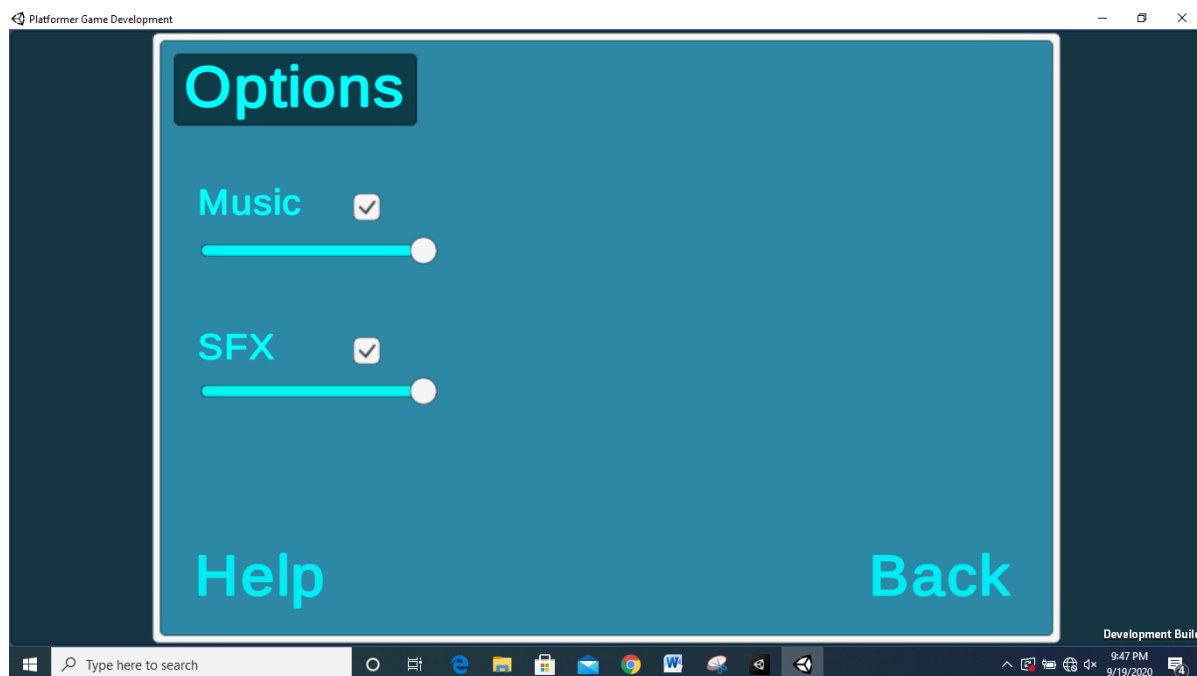
Game Author Screen



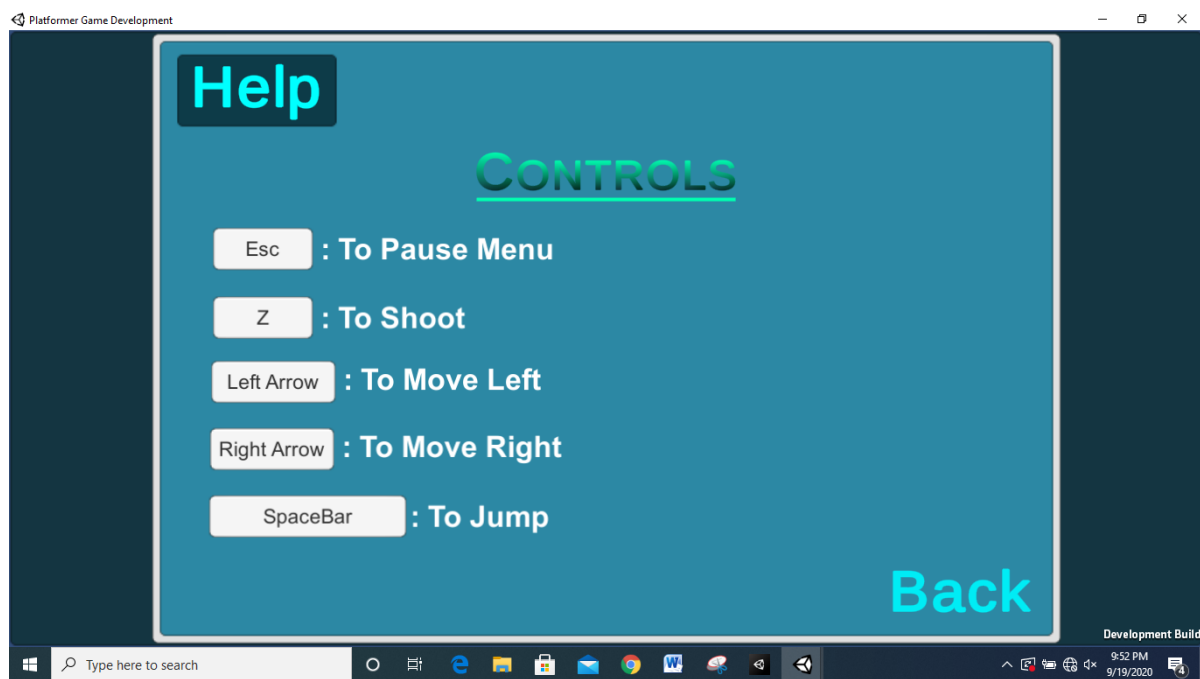
Main Menu



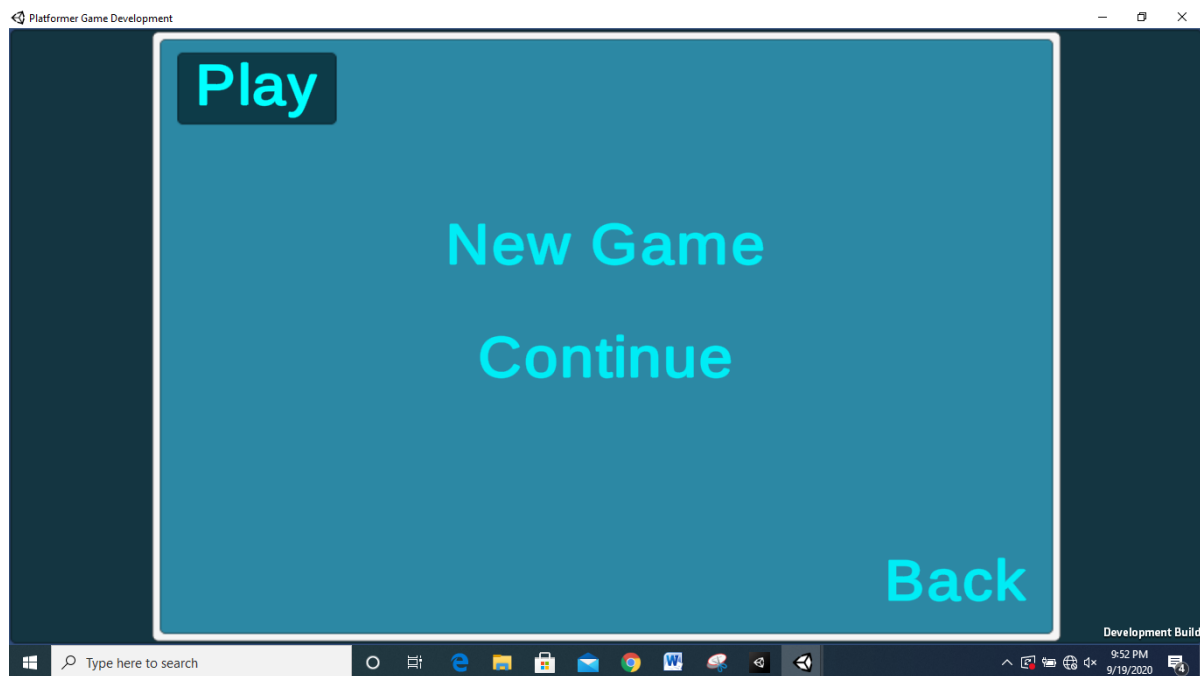
Option Menu



Help Screen



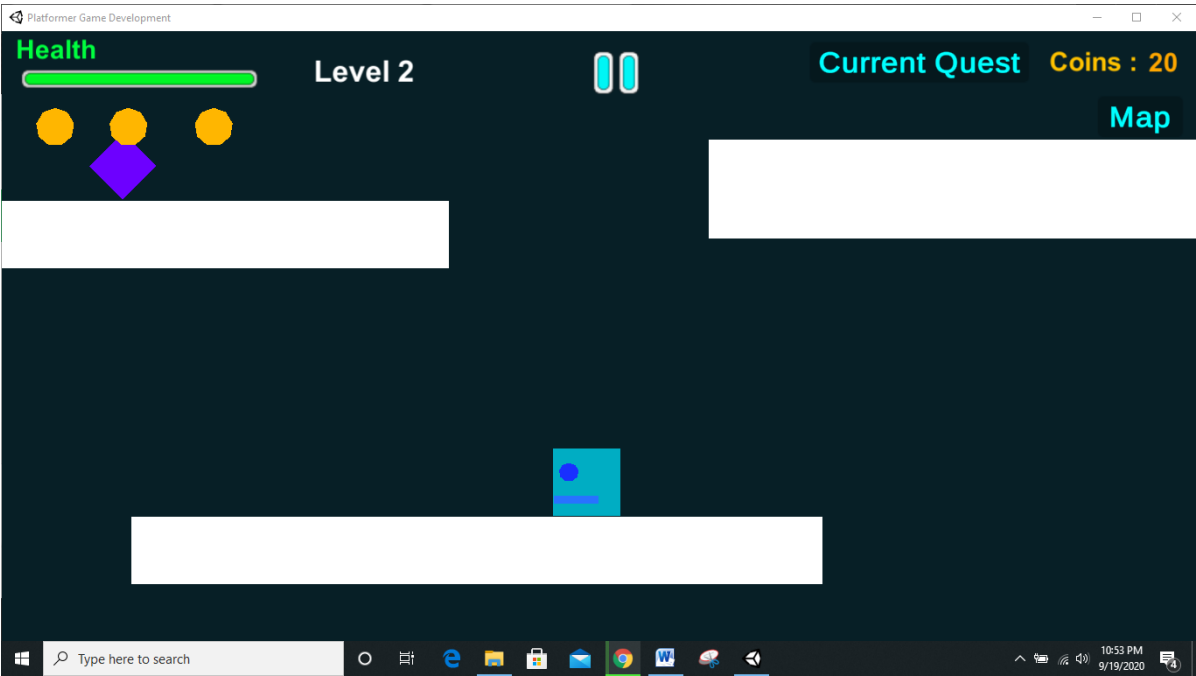
Play Menu



Level 1 Screen



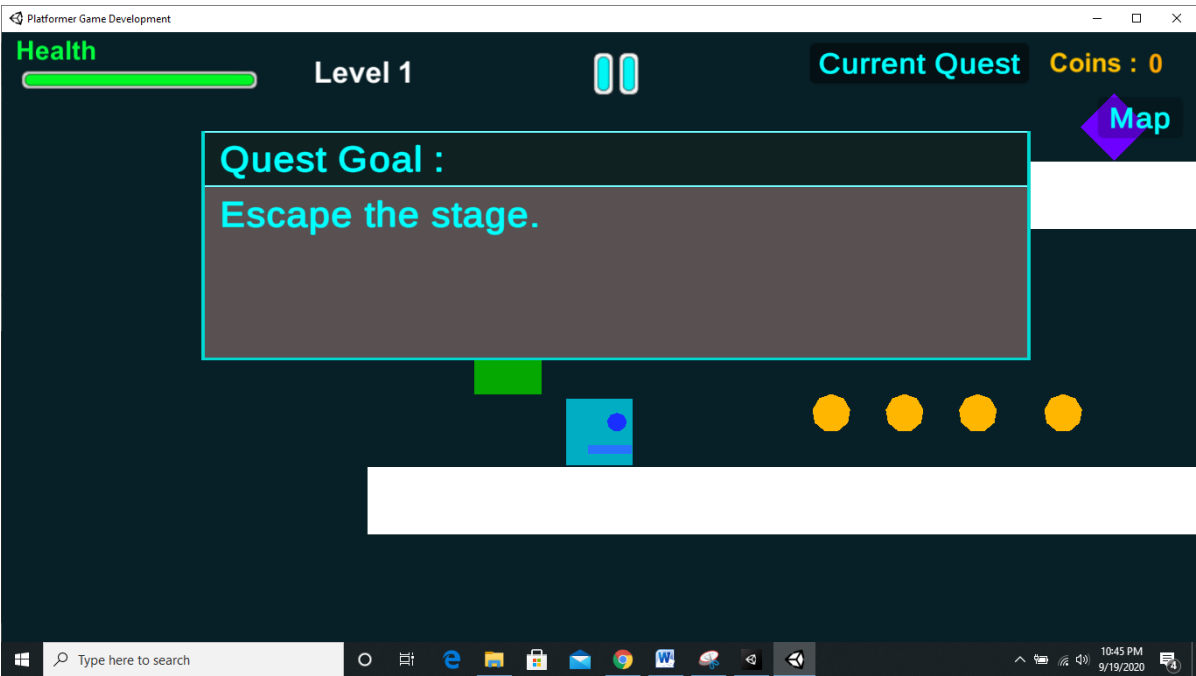
Level 2 Screen



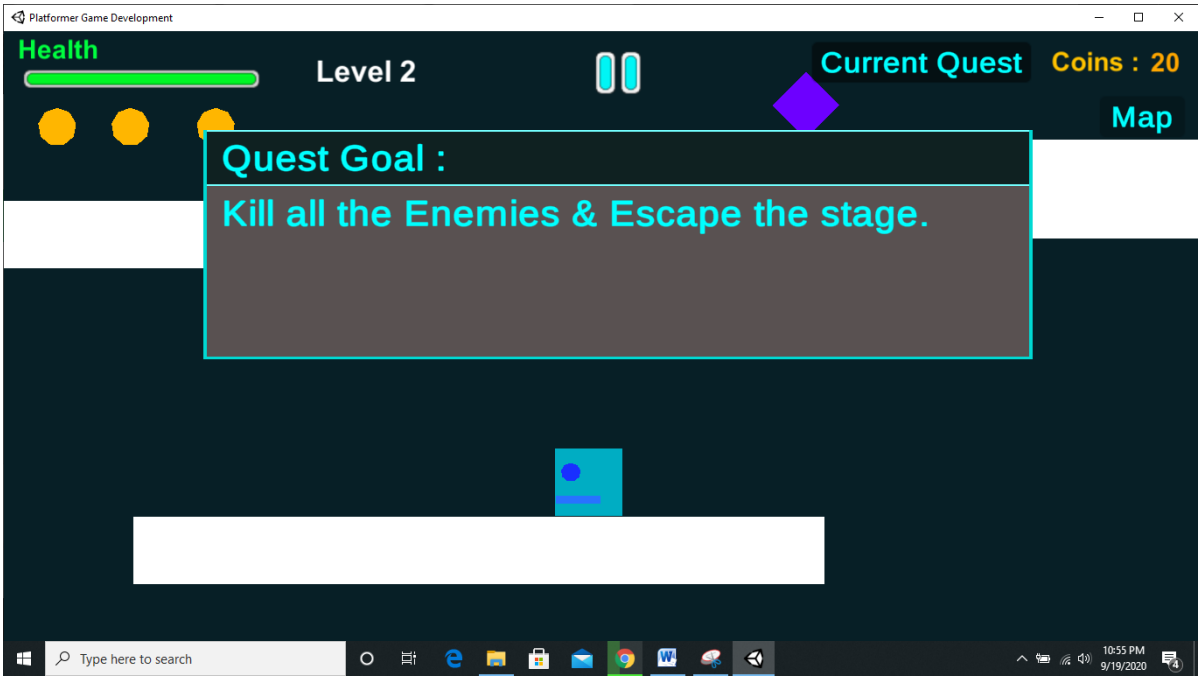
Level 3 Screen



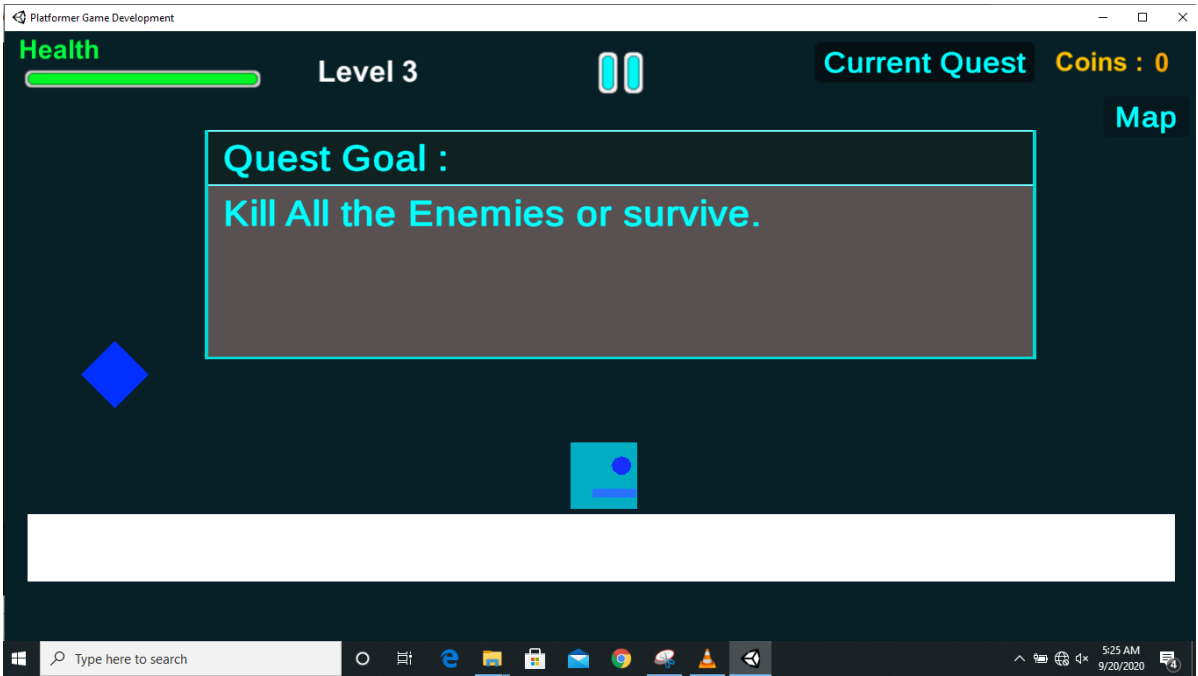
Level 1 Quest



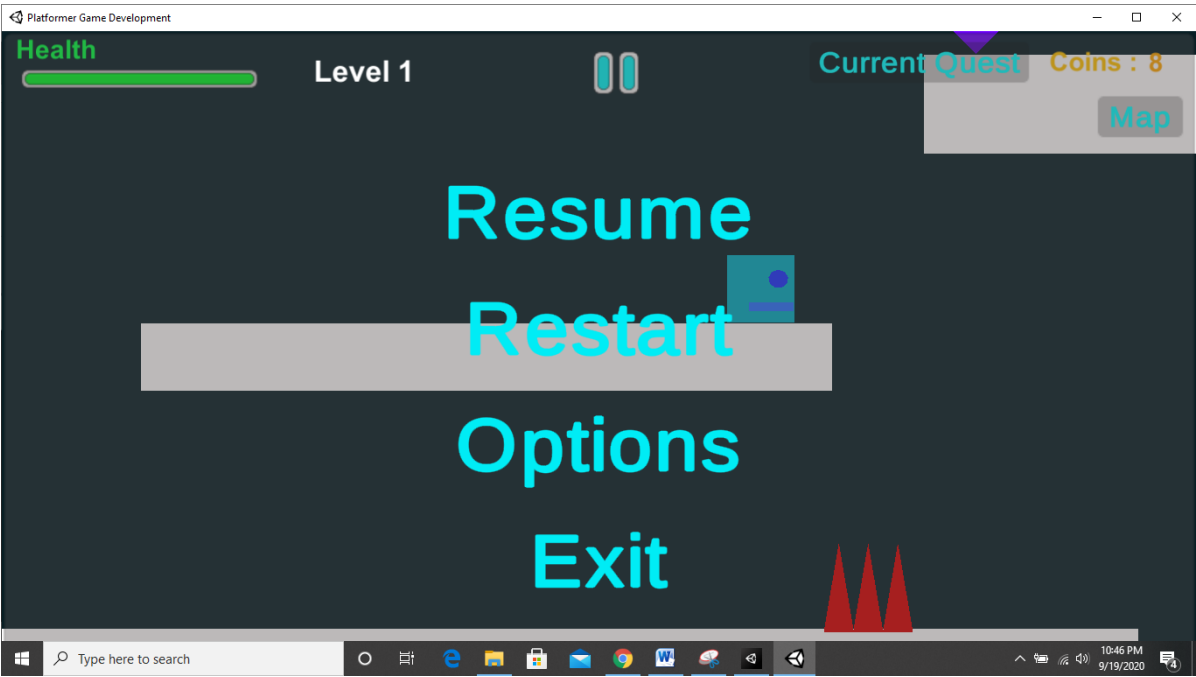
Level 2 Quest



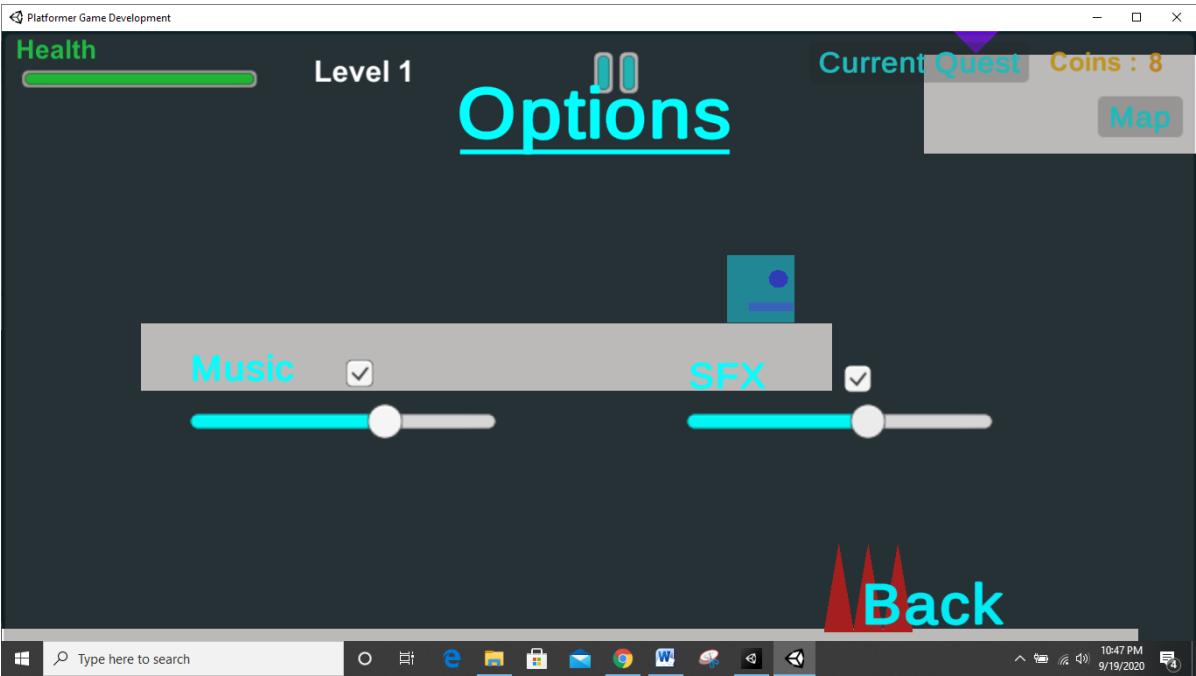
Level 3 Quest



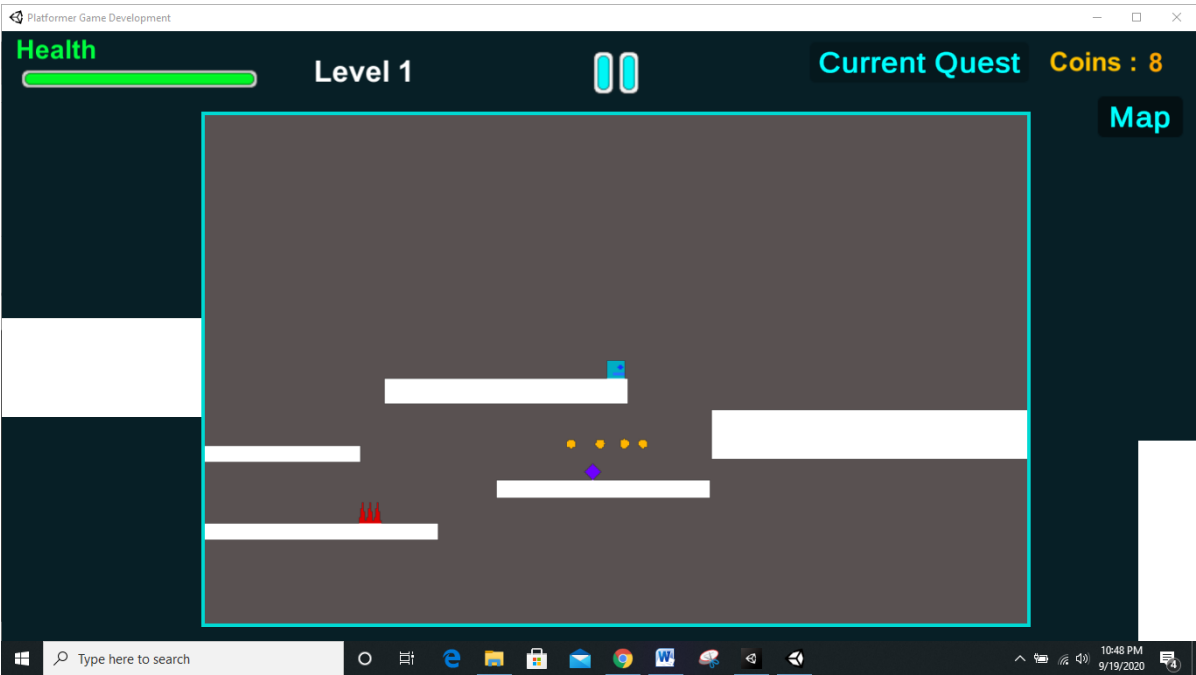
Pause Menu



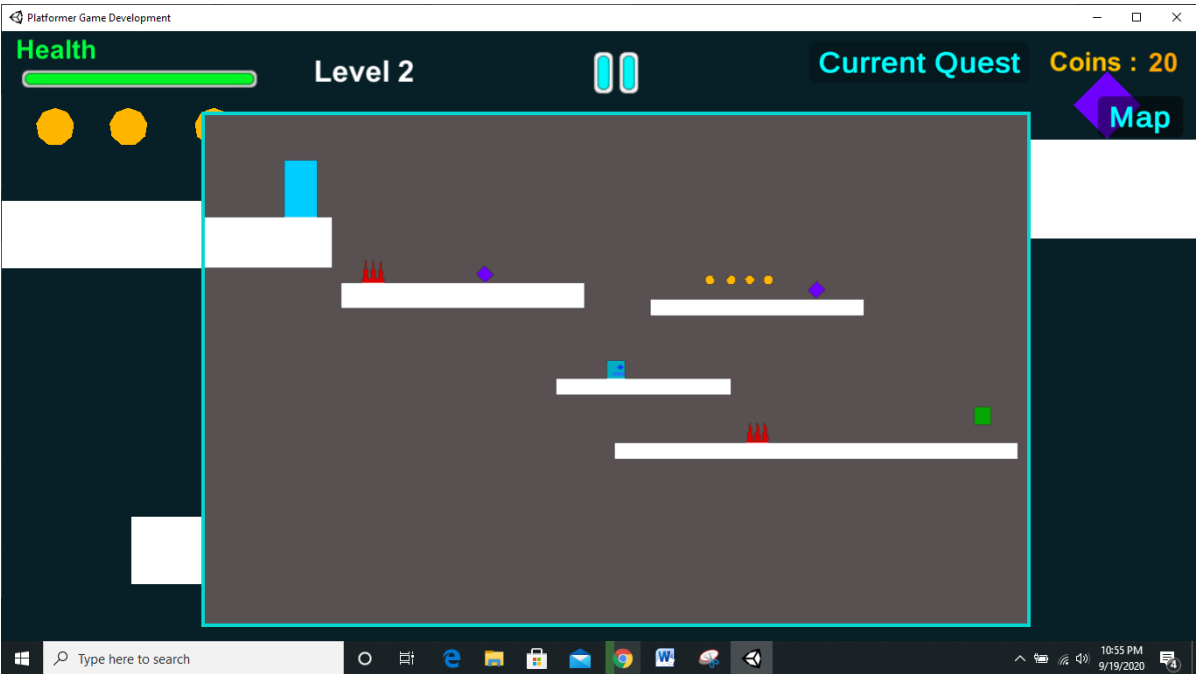
Option Menu in Pause Menu



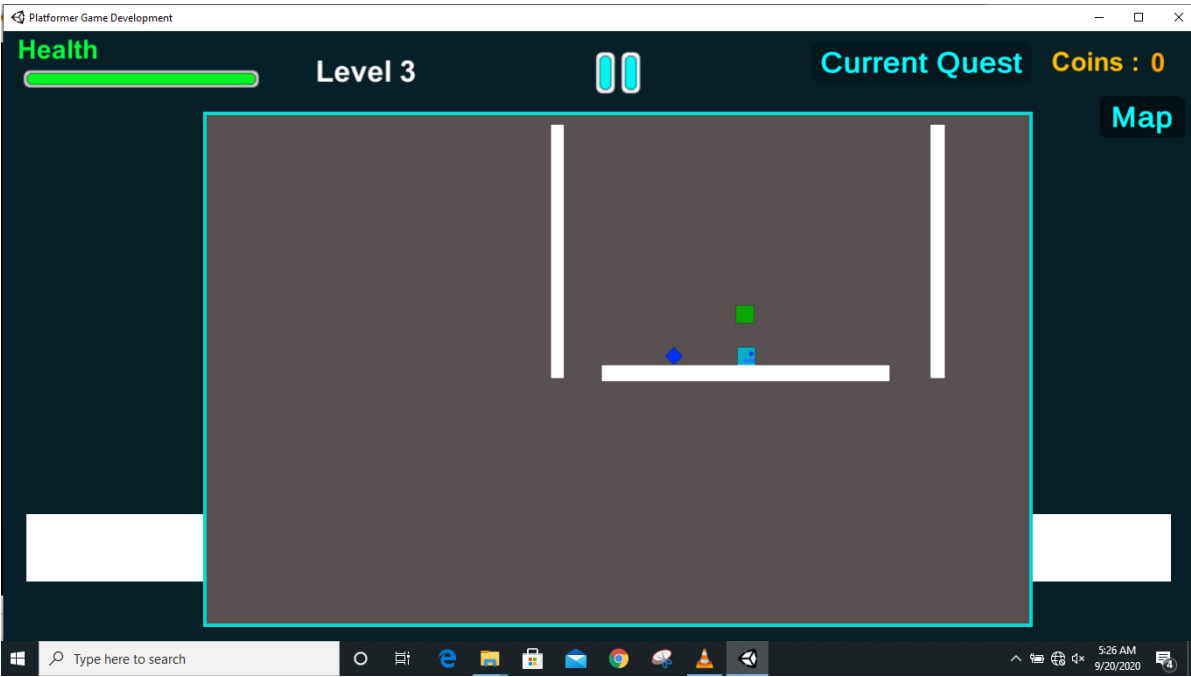
Mini Map Level 1



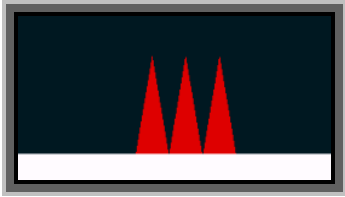
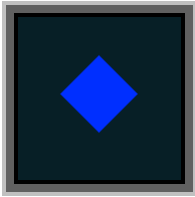
Mini Map Level 2



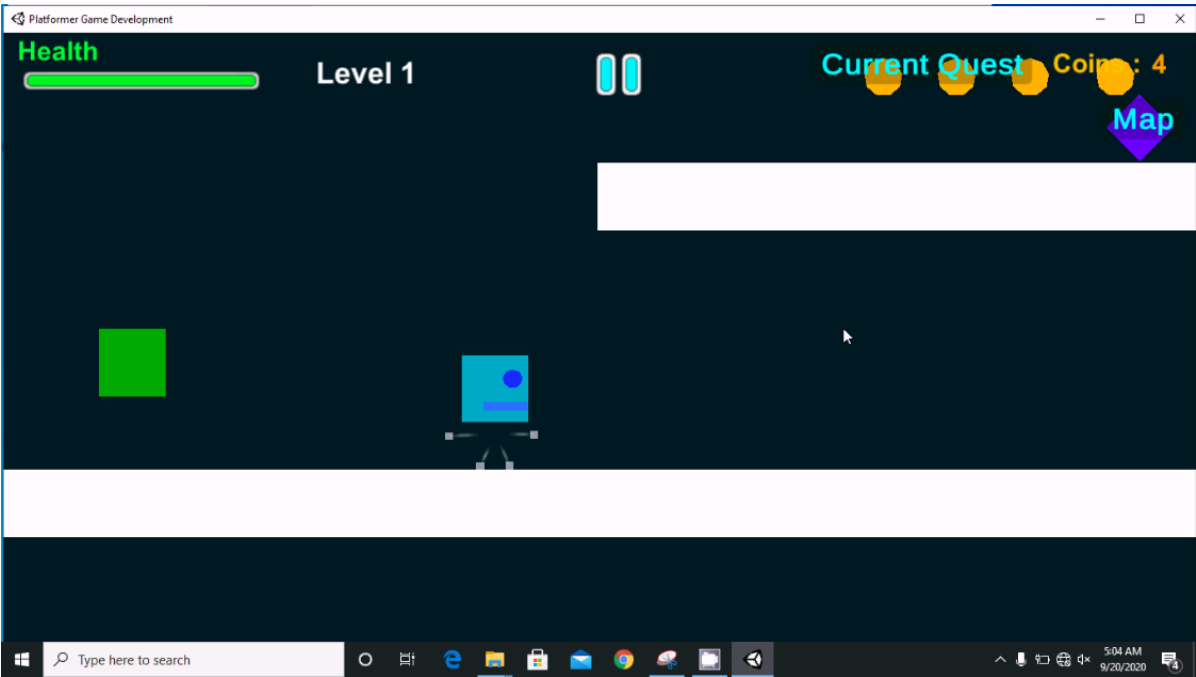
Mini Map Level 3



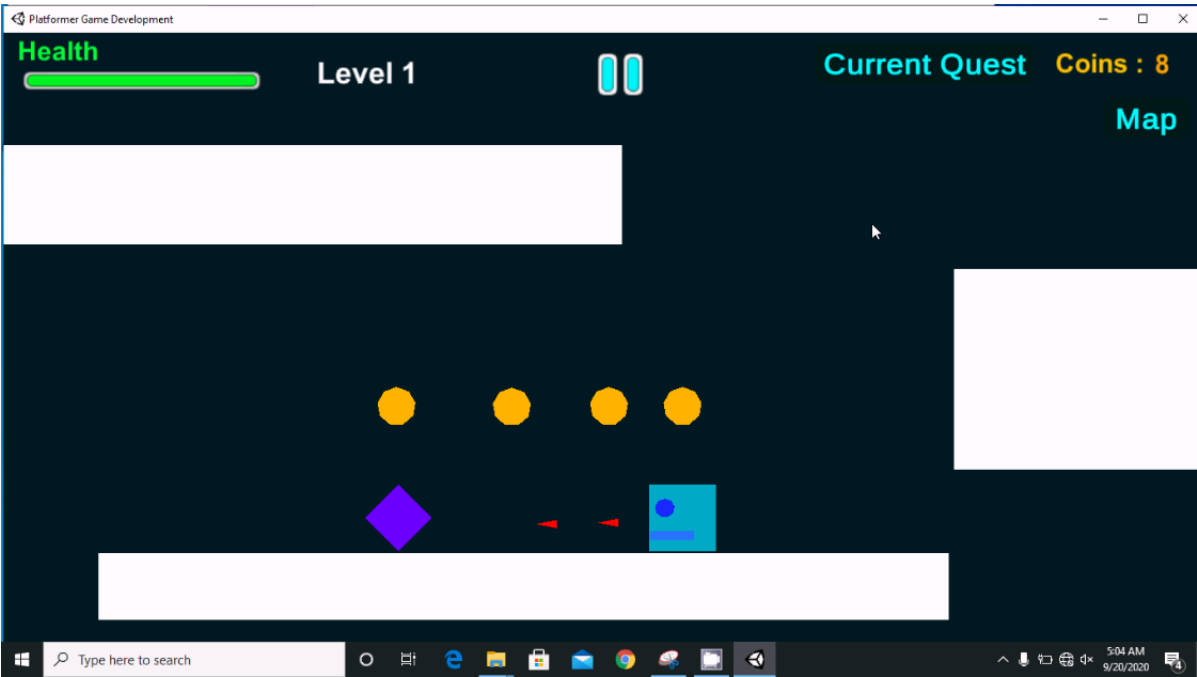
Enemy and Damager



Jump



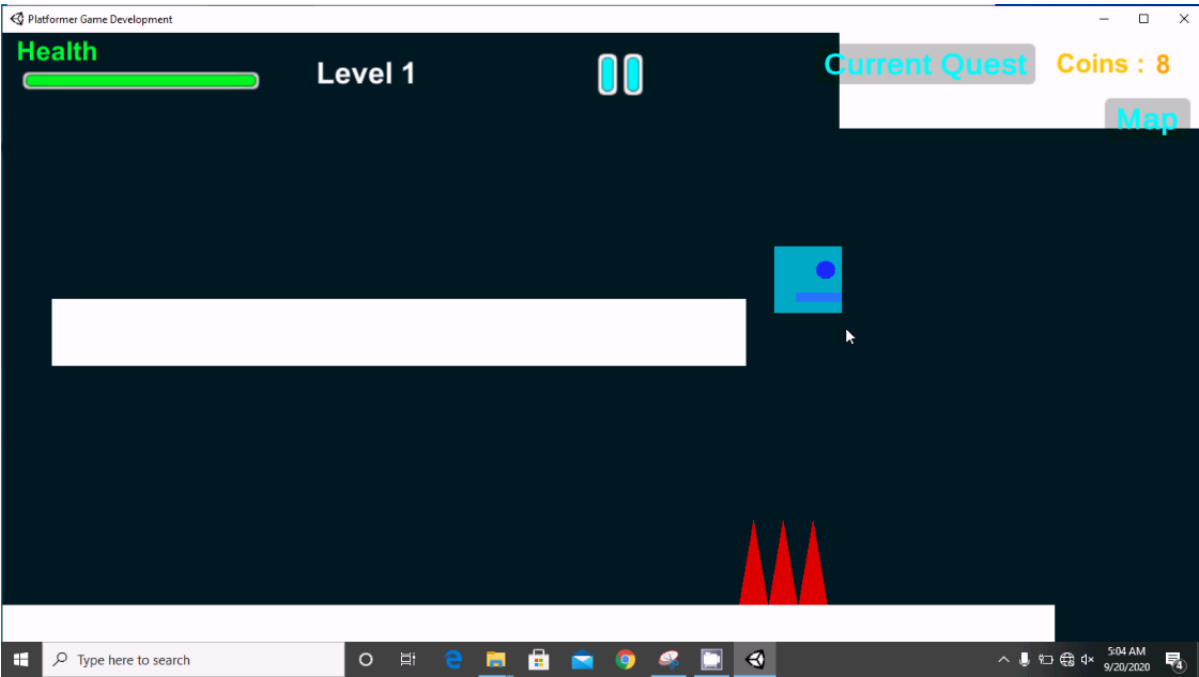
Shooting



Enemy Destruction



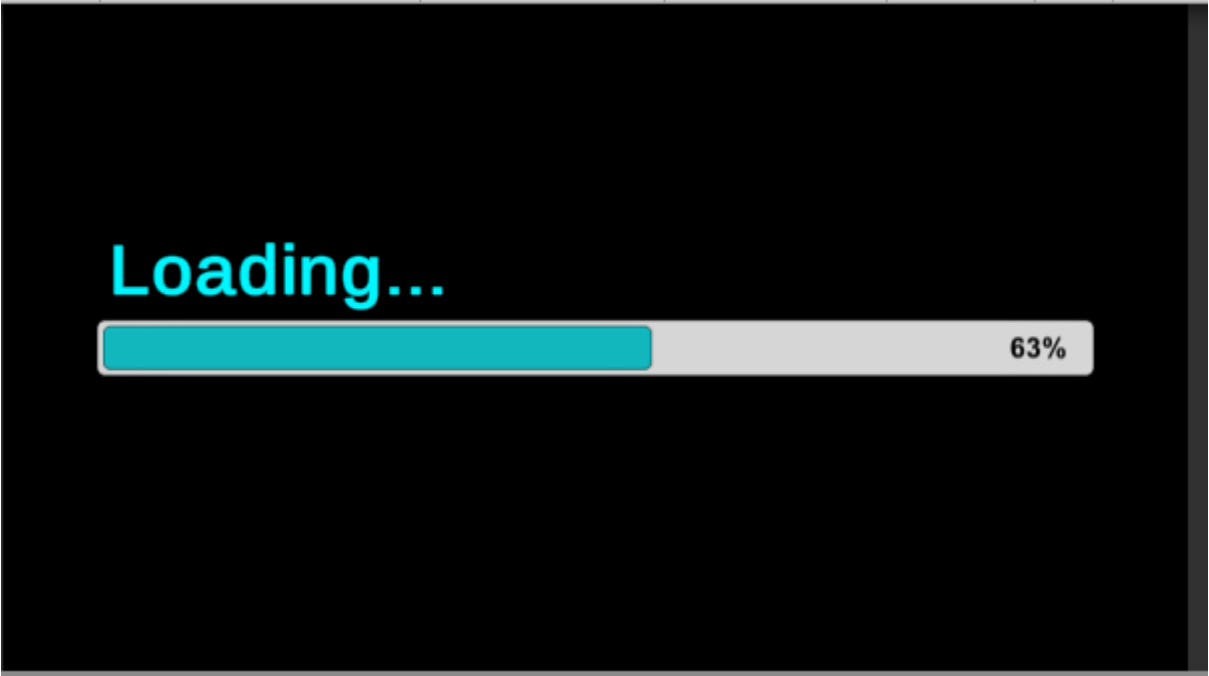
Damage Player



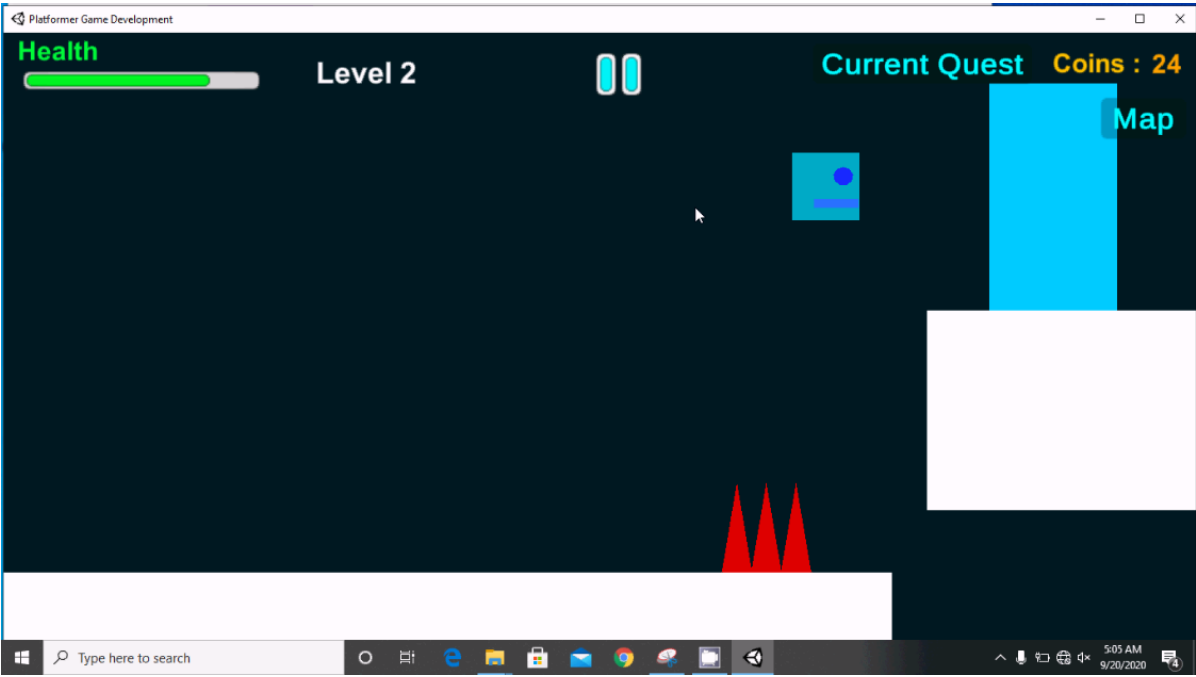
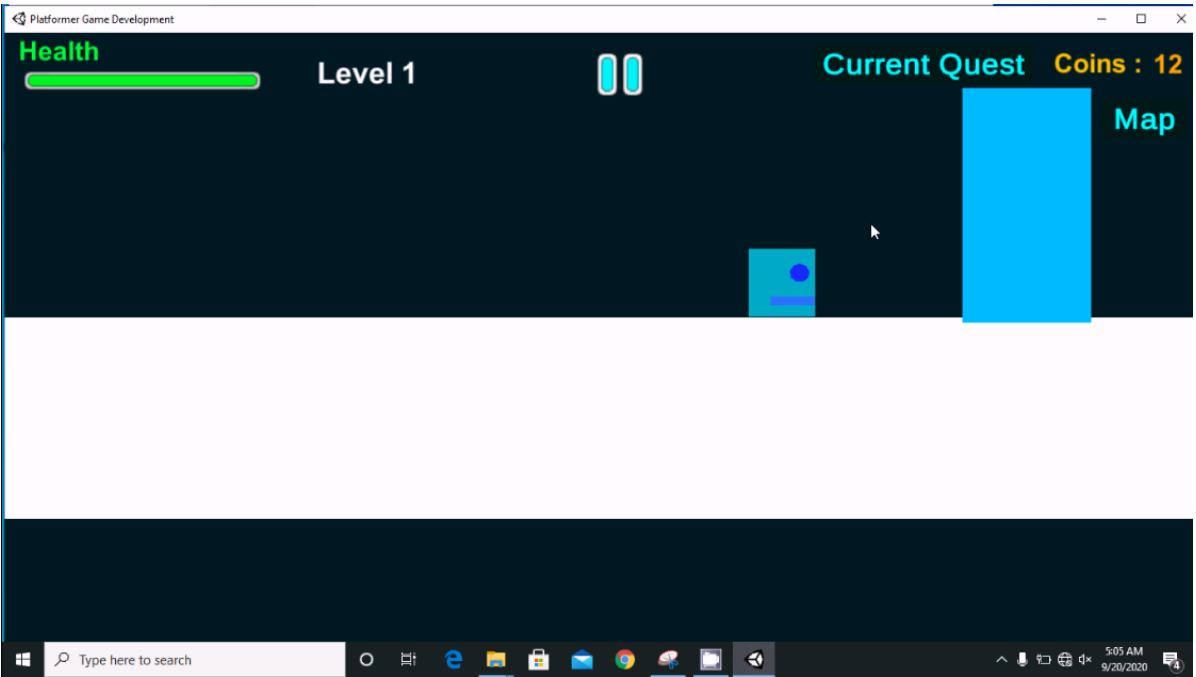
Coin Collector



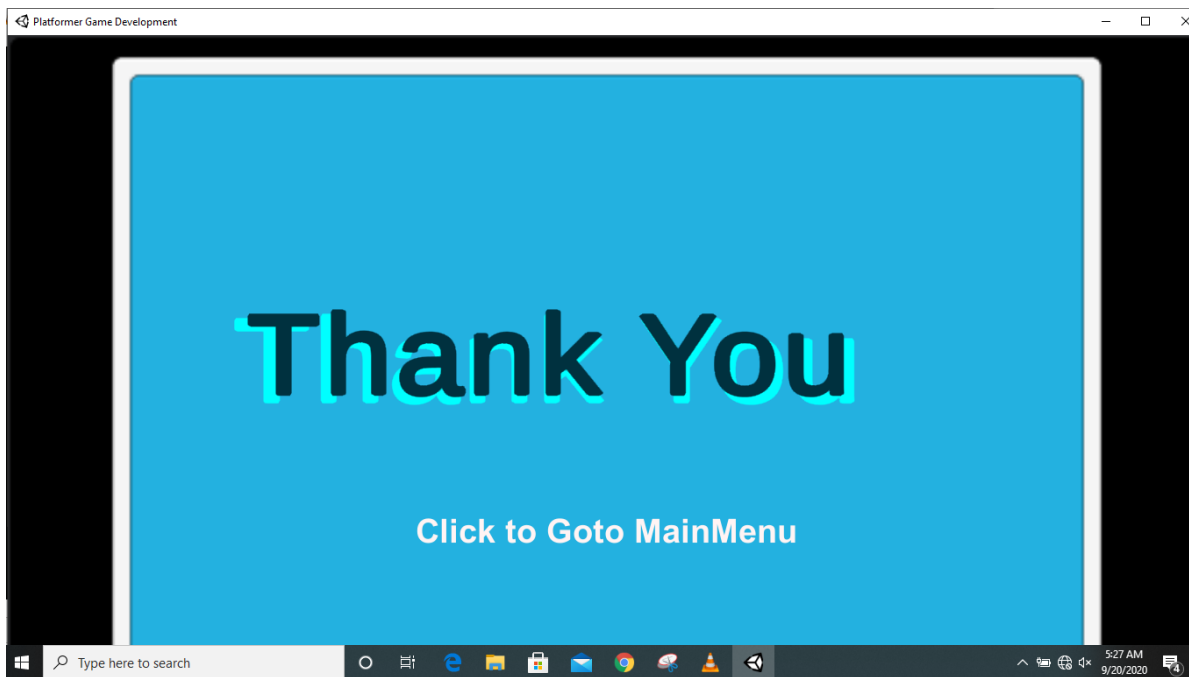
Level Loader



Teleporter



Game Complete Screen



Database Save

A screenshot of the DB Browser for SQLite application. The window title is "DB Browser for SQLite - E:\Pawan\unity\New folder\try_Data\game.s3db". The "Table:" dropdown is set to "mygame". The table data is displayed in a grid with columns: save_id, health, coins, level, player_pos_x, player_pos_y, player_pos_z, and saved_bool. The first row shows values: 1, 100, 0, 1, -7.103, -2.454659, 0.0, and True.

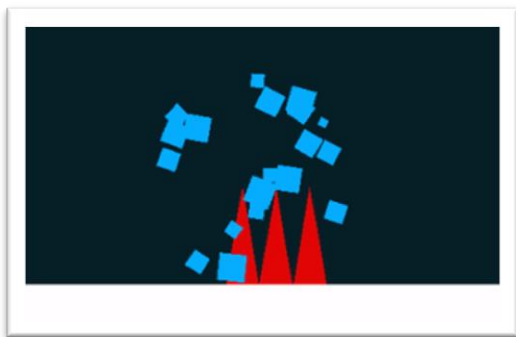
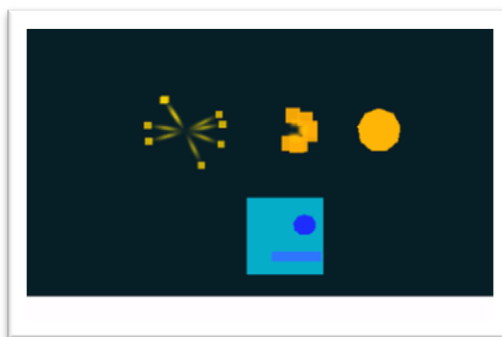
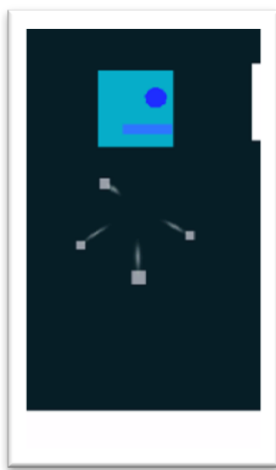
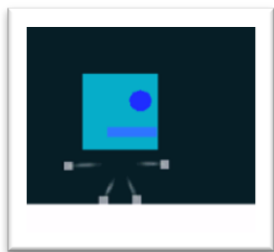
	save_id	health	coins	level	player_pos_x	player_pos_y	player_pos_z	saved_bool
1	1	100	0	1	-7.103	-2.454659	0.0	True

Database Table

A screenshot of the DB Browser for SQLite application showing the "Database Structure" tab. The "mygame" table is selected, and its structure is displayed in a table with columns: Name, Type, and Schema. The table structure is as follows:

Name	Type	Schema
mygame	CREATE TABLE [mygame] ([save_id] INTEGER NOT NULL PRIMARY KEY,[health] INTEGER NOT NULL	
save_id	INTEGER	"save_id" INTEGER NOT NULL
health	INTEGER	"health" INTEGER NOT NULL
coins	INTEGER	"coins" INTEGER NOT NULL
level	VARCHAR(20)	"level" VARCHAR(20) NOT NULL
player_pos_x	FLOAT	"player_pos_x" FLOAT NOT NULL
player_pos_y	FLOAT	"player_pos_y" FLOAT NOT NULL
player_pos_z	FLOAT	"player_pos_z" FLOAT NOT NULL
saved_bool	BOOLEAN	"saved_bool" BOOLEAN NOT NULL

Animations with Particle System



Chapter 6

Conclusion And Future Work

6.1 Conclusion

This game will take the player on a journey of adventure and exploration through multiple levels and scenarios. This system can be expanded in the future by adding more levels and enemy types, develop better controller for the enemy and NP 's (non-playable characters). Future versions of Unity3D engine will bring more features and opportunities to increase the efficiency of the code and add more feature.

6.1.1 Lessons Learned

First of all, I come to realize that keeping it simple is good. Having great core gameplay that is polished is very valuable. Sometimes we just keep inventing cool new features without realizing they just make the game unpleasantly complex and hard to approach. If the core mechanic cannot stand on its own then there is not much that new features can do to improve it.

Clearly I learned to implement the smooth tile-based model which is a good versatile method that can be used to make games other than platformers as well. Making the character move and jump can sound easy, but it can actually be very hard to make it properly.

I also came to notice, that there are many little aspects and features that I was not aware of before. For example the considerations for allowing the player to jump when it would normally be impossible or how many different ways there are to implement a camera. Reading the theory behind how cameras work in games helped me to realize the importance of what players see and how the camera movement affects them. I also learned how much variety there is in the different popular features cameras have in games and how they are tailor-made to fit the game.

6.2 Future Work

We will continue to Work in this Game and Improve it further by adding more Partical System Animation and More Attack Method. We will also Try to Make it Avaliable for Android as well. There will also be more Levels to Play in the Future.

Future Updates:

- More Levels
- Many types of enemies like Airborn.
- Available for Android Platform.

Chapter 7

References





7.1 Youtube Channels

- ❖ Inscope Studio channel (YouTube channel)
- ❖ Brackeys channel (YouTube channel)
- ❖ Unity channel (YouTube channel)
- ❖ Code Monkey(YouTube channel)

7.2 URLS

- 1) <https://unity.com>
- 2) <http://freeassests.com>
- 3) <http://www.divaportal.org/smash/get/diva2:692737/FULLTEXT01.pdf>
- 4) http://www.gamasutra.com/view/feature/1851/a_detailed_crossexamination.php
- 5) http://www.gamasutra.com/blogs/BenChong/20150112/233958/Endless_Runner_Games_How_to_think_and_design_plus_some_history.php
- 6) http://www.gamasutra.com/view/feature/167392/sad_but_true_we_cant_prove_when_.php
- 7) https://archive.org/stream/Electronic_Games_Volume_01_Number_11_1983.html

7.3 Books

-  Unity 2D Game Development Cookbook by **Claudio Scolastici**
-  Learning Unity 2D Game Development by Example by **Venita Perira**
-  Unity 2D Game Development by **Dave Calabrese**
-  Learn Unity for 2D Game Development by **Alan Thorn**