RAIDER PARKOUR

HAYDEN TROXELL TYLER WELLS IMMANUEL SABWAMI DIPESH PANERU ADAM LADUE

ELEVATOR PITCH

Raider Parkour will be a 3D parkour game that involves a user controlling a character jumping across challenging platforms to reach the ultimate finish line.

INTRODUCTION

We used the Unity engine to develop our game and used blender for the 3D assets

OVERVIEW

UI DESIGN

PLAYER CONTROLS

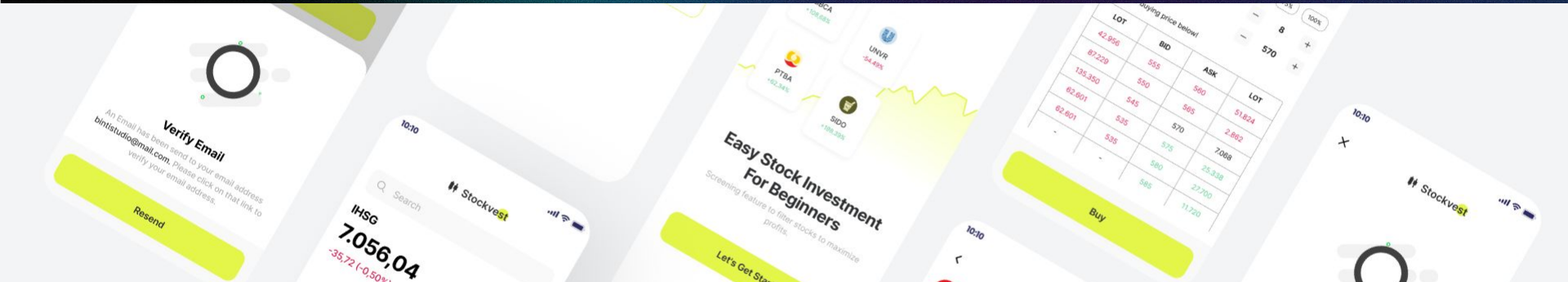
3D DESIGN

PLAYER INFO

AUDIO | TESTING

RAIDER PARKOUR

UI DESIGN



USER STORY

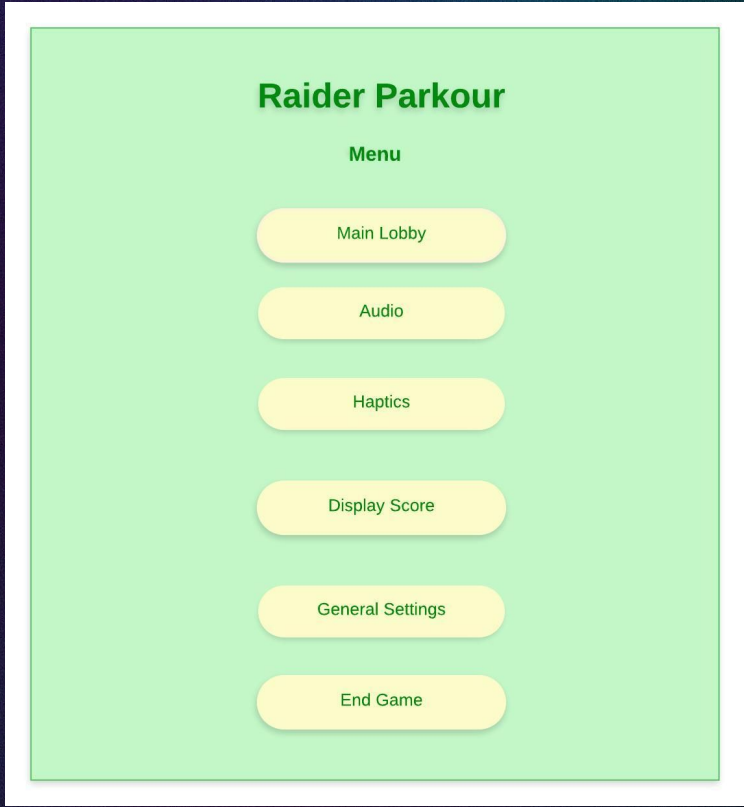
As a developer I want to develop the in-game toggle events(start menu) to help ensure a straight and fun direction for a player to achieve goals and rewards so that they are obligated to be more 'engaged'.

REQUIREMENT

INTERACTIVE START MENU

The game will have a start menu on load up that allows a player to choose between options to start or quit game. It will be very easy to navigate and look appealing to the user.

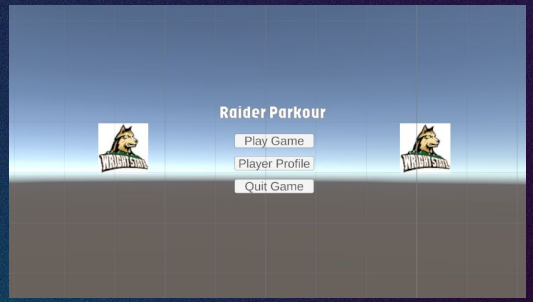
DESIGN



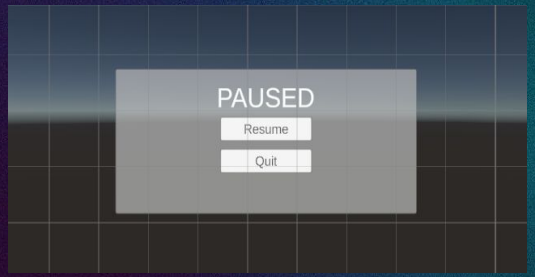
UI Design

IMPLEMENTATION

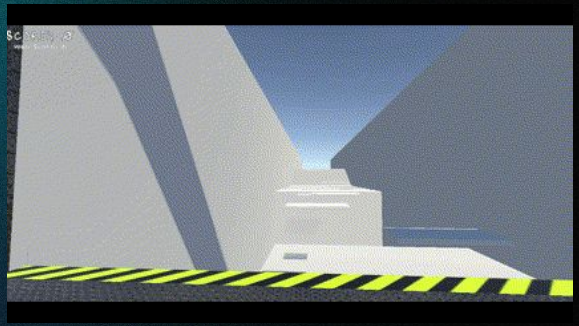
Main Menu



Pause Menu



Game Over Menu



UI Design

PLAYER CONTROLS

USER STORY

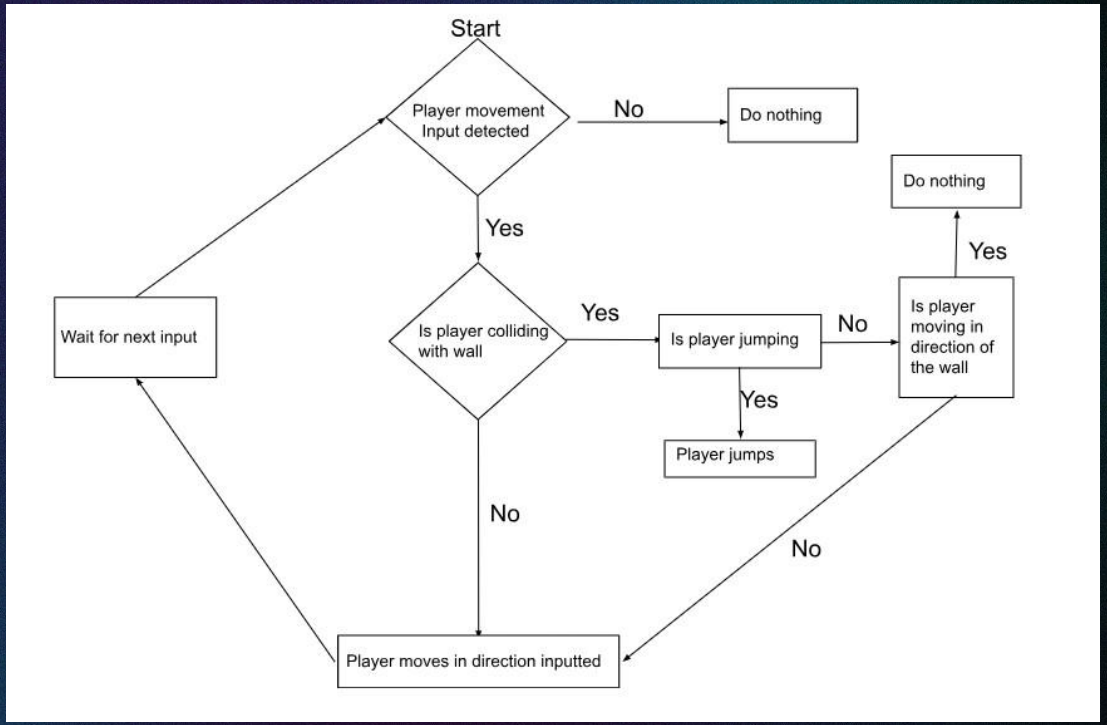
As a developer, I want to develop the backend logic that will insure smooth collision detection, which will allow the player to enjoy the game and its environment without having any player model clipping issues.

REQUIREMENT

FLUID GAME CONTROLS

The system shall provide continuous monitoring of in-game player movement and environmental interactions to ensure smooth and seamless gameplay.

DESIGN

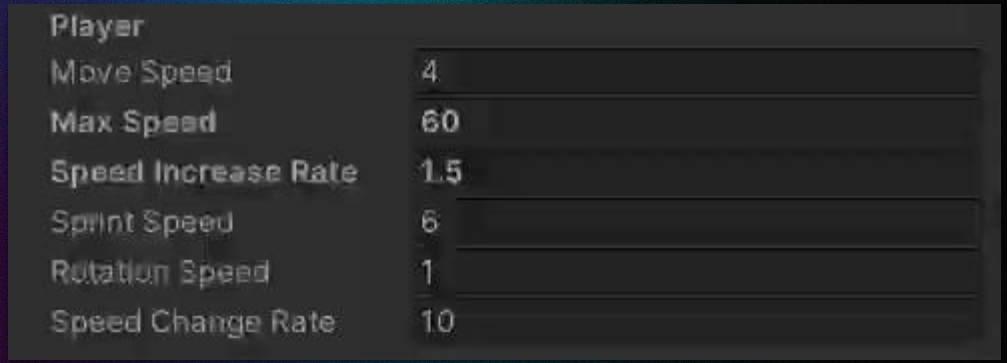


PLAYER CONTROLS

IMPLEMENTATION

```
private void IncreaseSpeedOverTime(){
    // Gradually increase speed over time until it reaches the maximum speed
    if (MoveSpeed < maxSpeed){
        MoveSpeed += speedIncreaseRate * Time.deltaTime;
        MoveSpeed = Mathf.Clamp(MoveSpeed, 0f, maxSpeed);
    }

    // Ensures SprintSpeed also increases over time
    if (SprintSpeed < maxSpeed){
        SprintSpeed += speedIncreaseRate * Time.deltaTime;
        // Throttles SprintSpeed so it does not pass maxSpeed
        SprintSpeed = Mathf.Clamp(SprintSpeed, 0f, maxSpeed);
    }
}
```



PLAYER CONTROLS

3D DESIGN

USER STORY

As a developer, I want to implement custom-made 3d assets for the game to give it more character and to separate from any other similar projects built using our workflow.

REQUIREMENTS

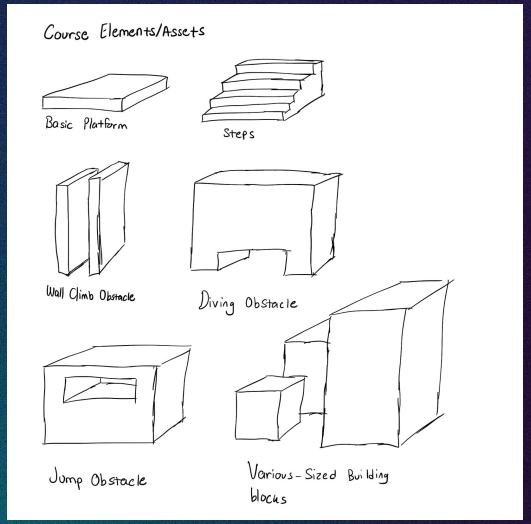
CUSTOM MADE 3D ASSETS

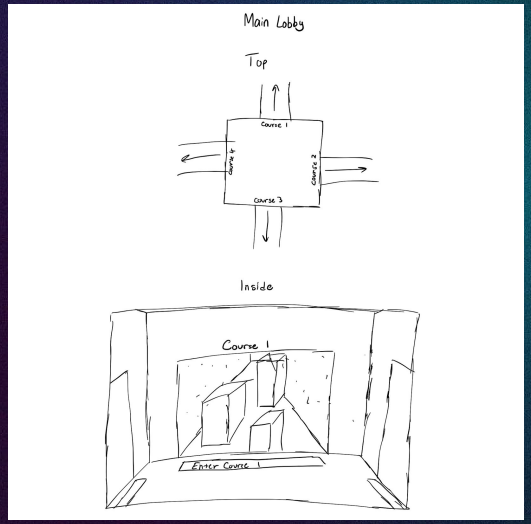
The game shall include custom-made/externally created parkour structures.

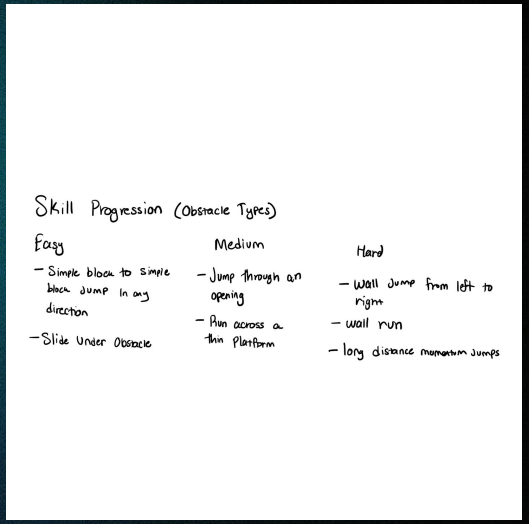
SKILL PROGRESSION

The difficulty level of each course shall gradually increase the further the player progresses through it.

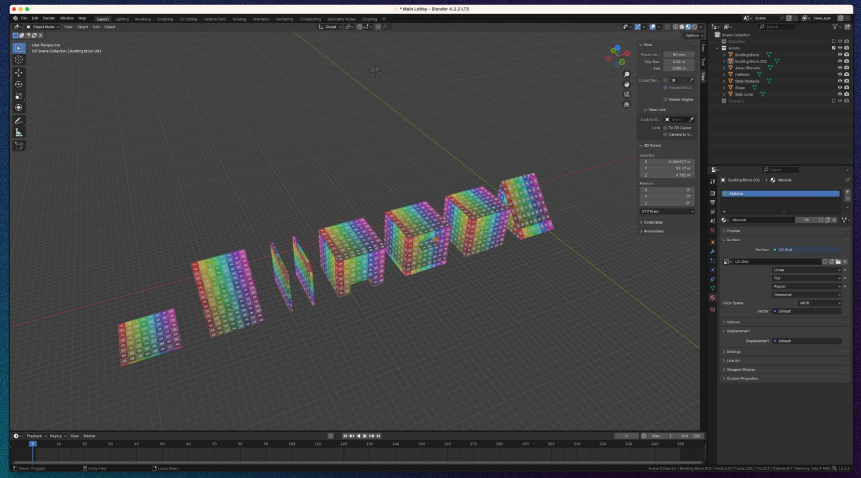
DESIGN


Course Elements/Assets
Basic Platform
Steps
Wall Climb Obstacle
Diving Obstacle
Jump Obstacle
Various-Sized Building blocks

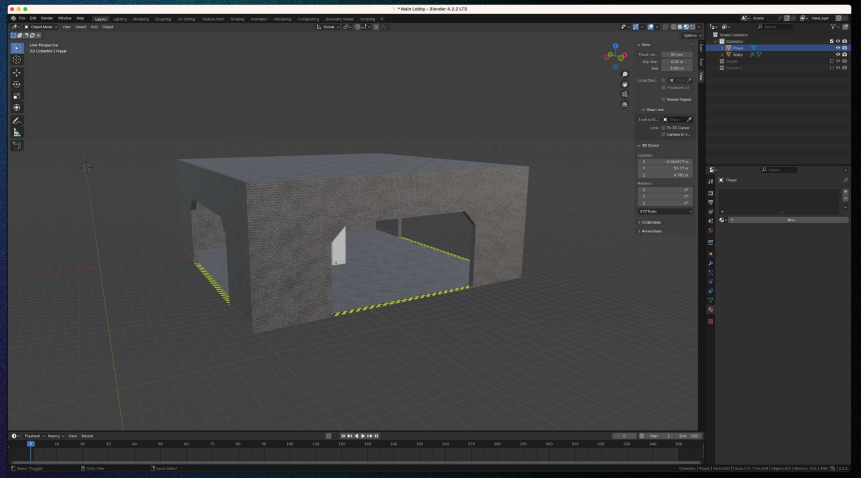

Main Lobby
Top
Course 1
Course 2
Course 3
Course 4
Inside
Course 1
Enter Course 1


Skill Progression (Obstacle Types)
Easy
- Simple block to simple block jump in any direction
- Slide under obstacle
Medium
- Jump through an opening
- Run across a thin platform
Hard
- Wall jump from left to right
- wall run
- long distance momentum jumps

IMPLEMENTATION



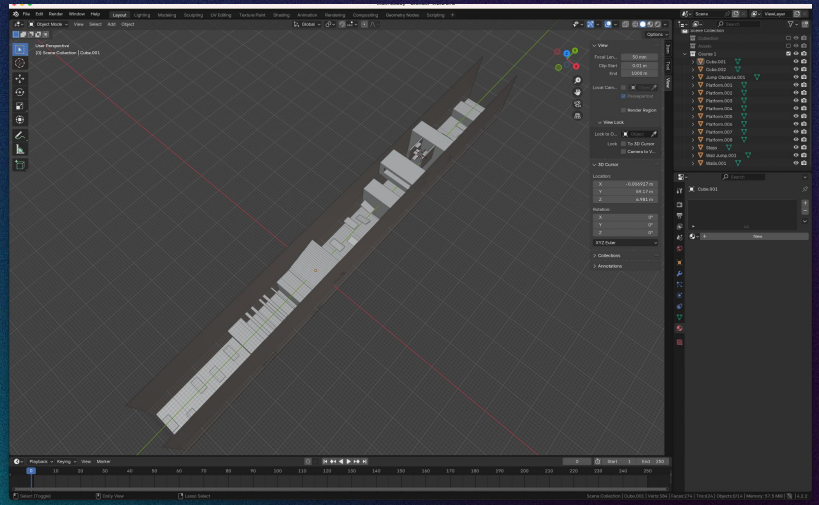
BASIC BUILDING BLOCKS



MAIN LOBBY

3D DESIGN

IMPLEMENTATION

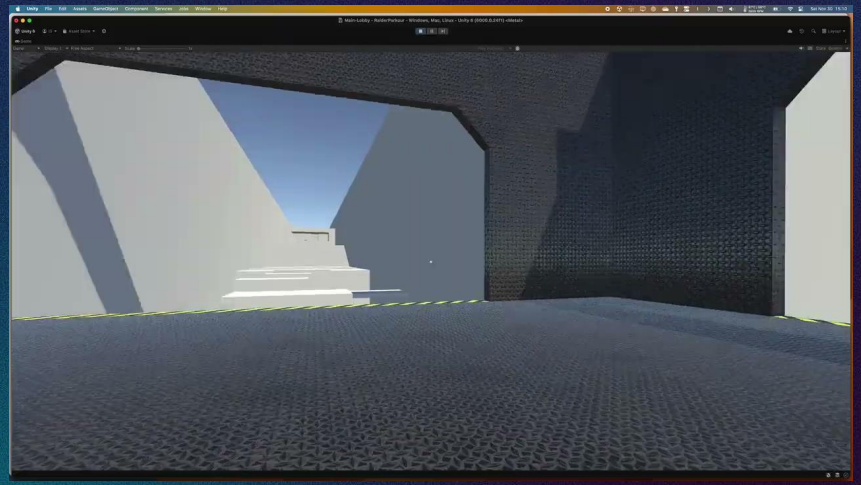


COURSE BLENDER MOCKUP

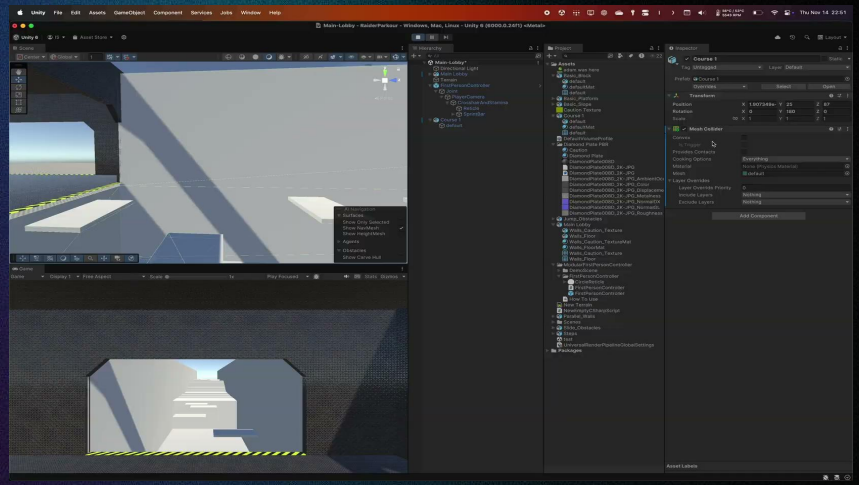
COURSE UNITY IMPLEMENTATION

3D DESIGN

IMPLEMENTATION



MAIN LOBBY



COURSE TEST RUN

3D DESIGN

PLAYER INFO


5.340
0.450
30,393
2.440
2.750
1.600
56,512
0.000
1.830
2.290
128,544
0.000
1.600

USER STORY

As a developer, I want to implement a database system that stores player scores and personal records, so that players can track their progress and compare their best performances across different runs with themselves and other players.

PLAYER INFO

REQUIREMENTS

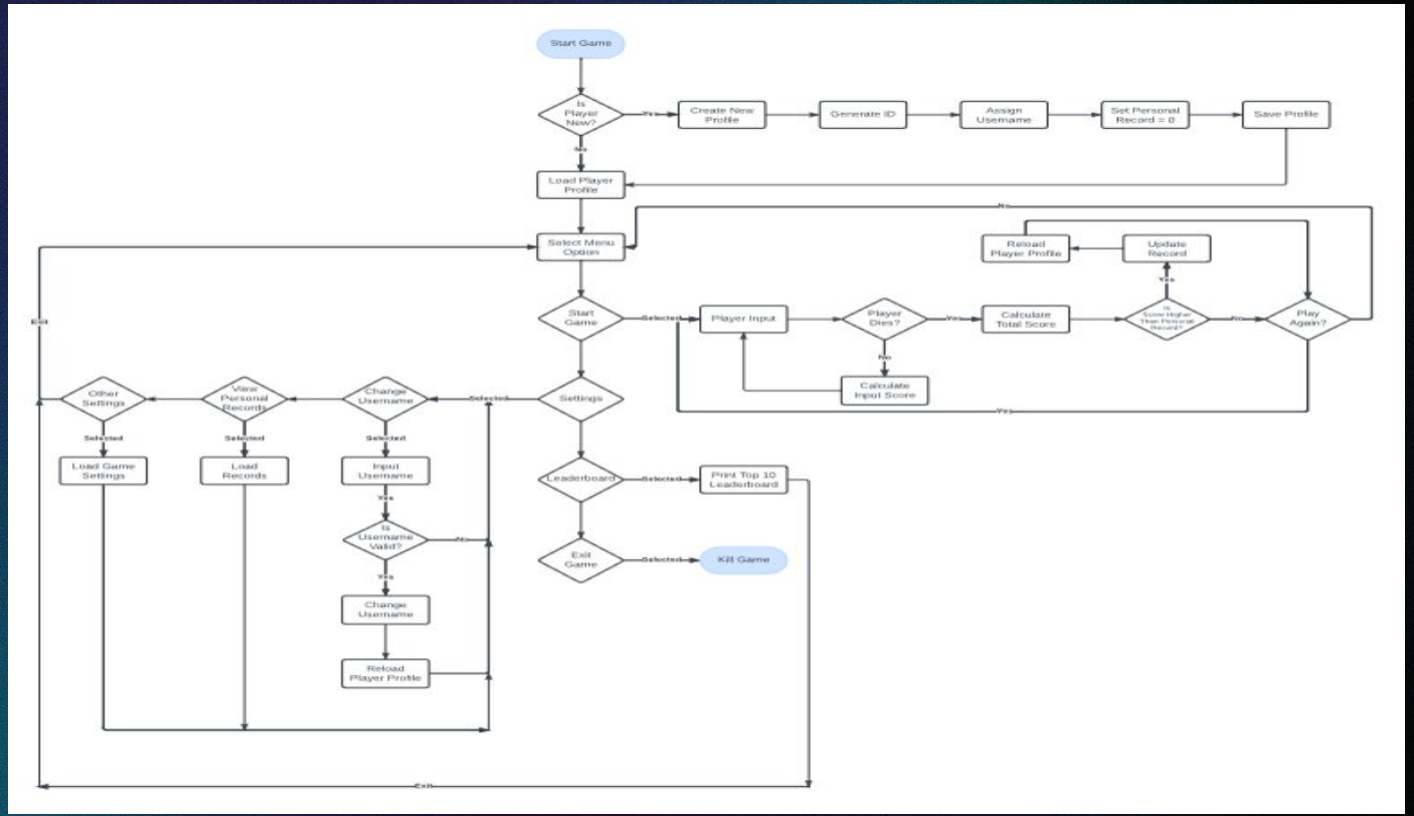
PLAYER PROFILES

The system shall create a new player profile for a new player when the game is first launched, give the profile a unique PlayerID and a default username, and contain the personal record of the player. The system shall also allow the user to change their username and prevent the user from having a username longer than 15 characters long, containing ONLY letters or numbers.

PLAYER SCORES

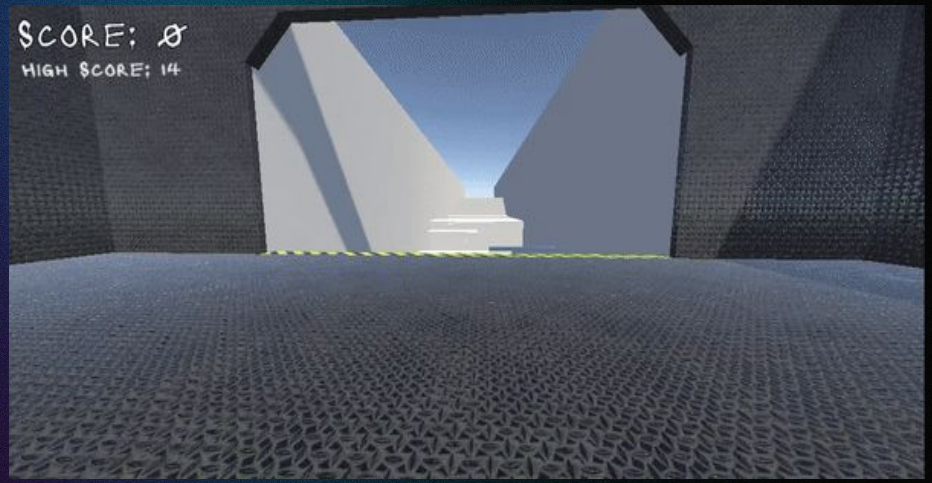
The system shall correctly keep track of the players score throughout the level and shall update the player's profile personal record upon death, if current score is larger than the current personal record

DESIGN



PLAYER INFO

IMPLEMENTATION



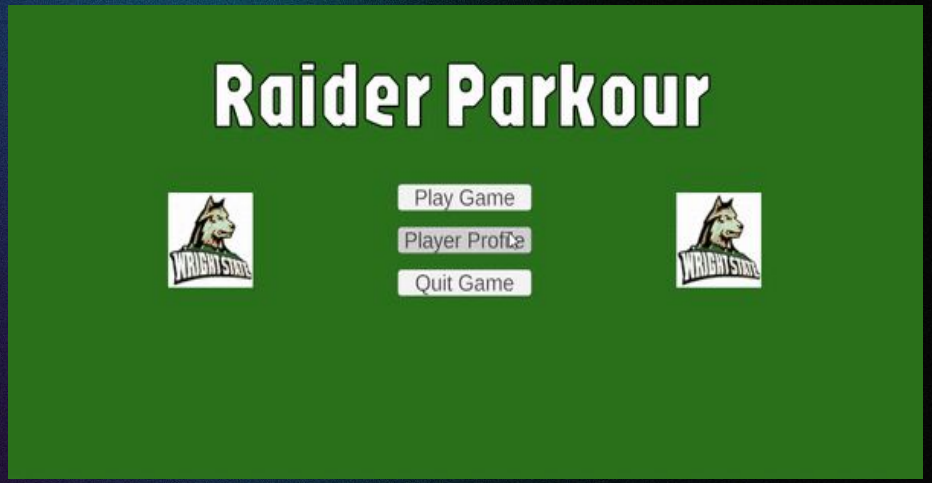
SCORE TRACKING

HIGHSCORE UPDATING

PLAYER INFO

IMPLEMENTATION

CHANGING USERNAME



INVALID USERNAMES

PLAYER INFO

AUDIO

USER STORY

As a Developer, I want to implement feedback when the player performs certain action to make the game more immersive.

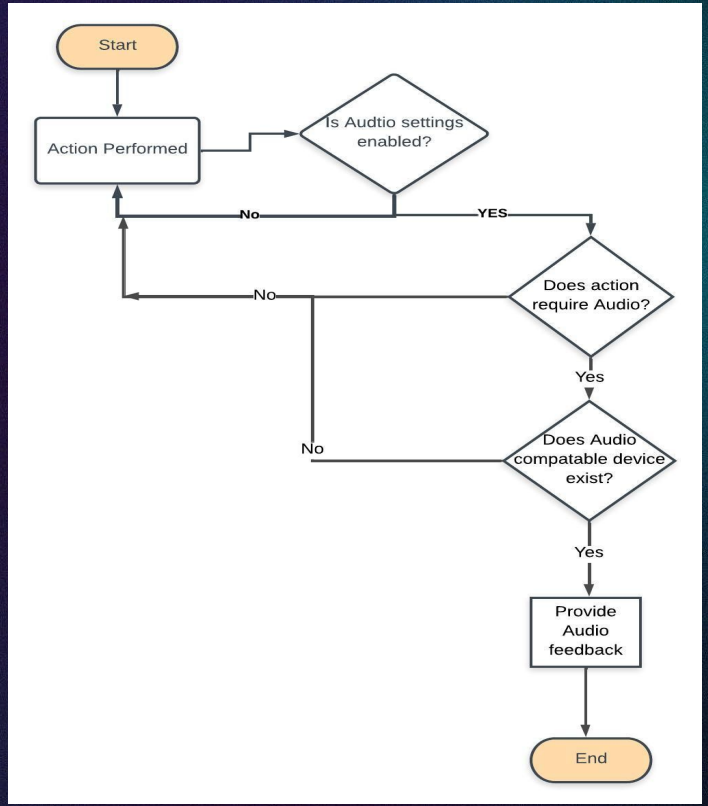
AUDIO

REQUIREMENT

AUDIO FEEDBACK

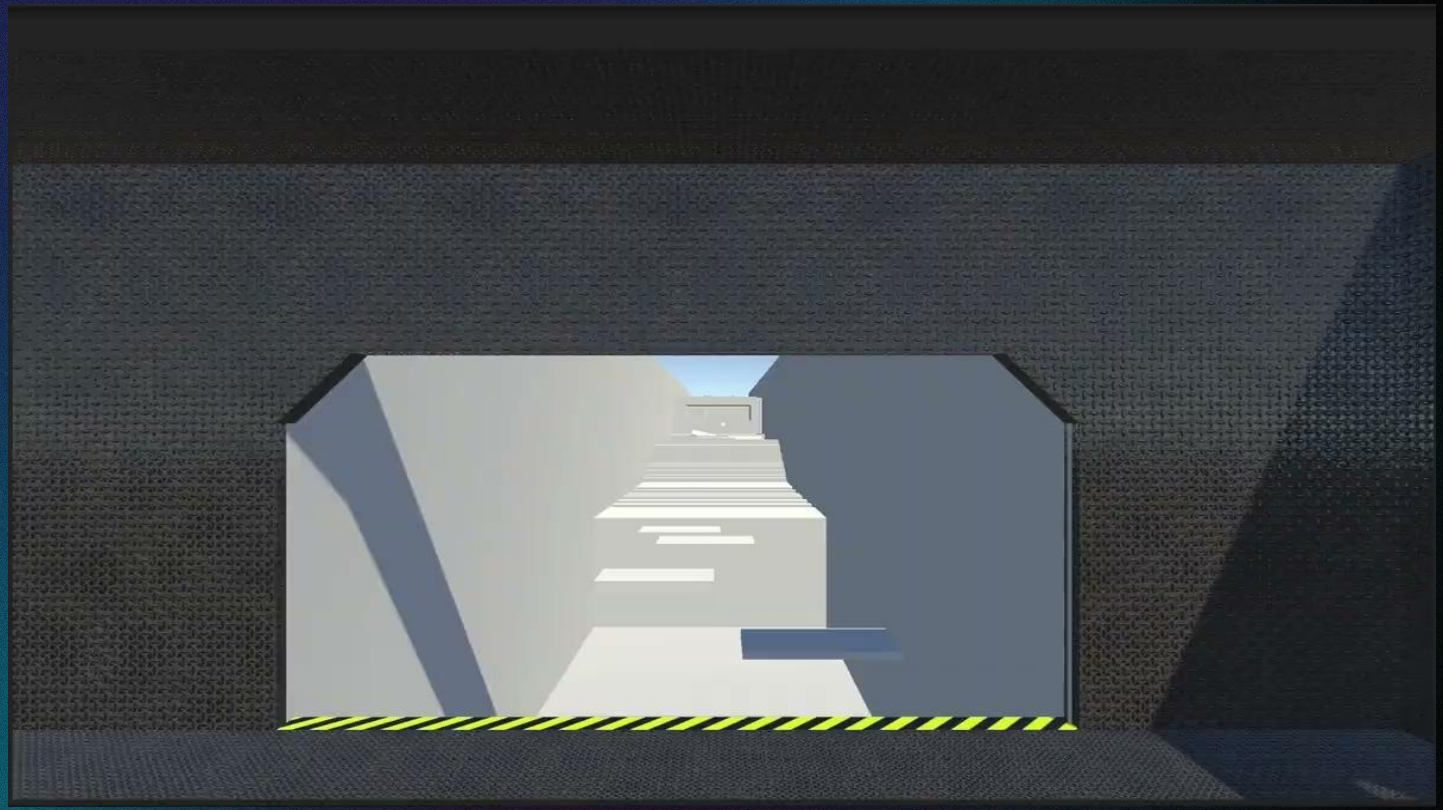
The game shall provide audio feedback during some special circumstances such as jumping, landing , being killed e.t.c.

DESIGN



AUDIO

IMPLEMENTATION



AUDIO

TESTING

USER STORY

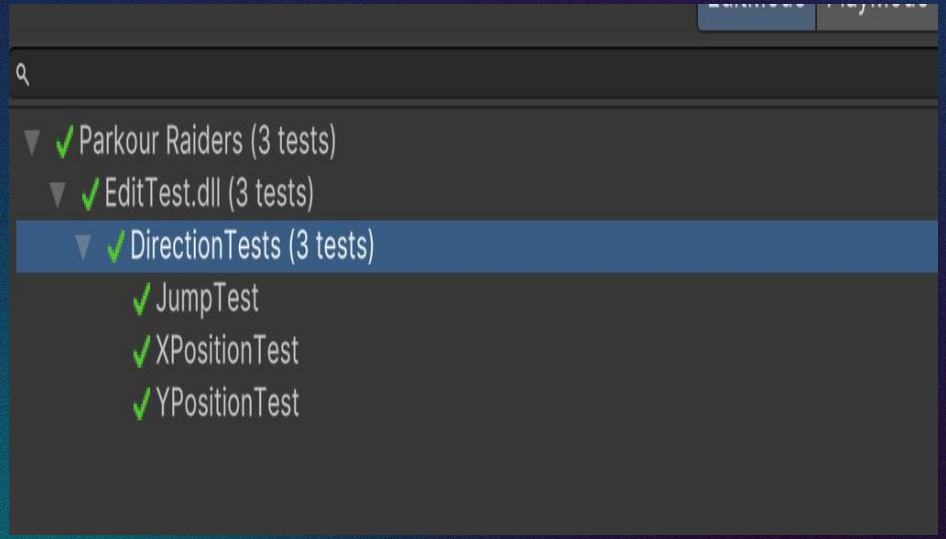
As a Tester, I want to be able to play the game without being "Killed" (i.e game restarting) in any case, so that I can focus on the actual testing of the game instead of being focused on not being killed so that the game does not restart.

REQUIREMENT

NO-KILL MODE

No kill mode in the game shall allow the player to play the game without being killed

IMPLEMENTATION




```
public class DirectionTests
Click to collapse the range.
    [Test]
    0 references
    public void XPositionTest(){
      var gameObj = new GameObject();
        var player = gameObj.AddComponent<FirstPersonController>();

        Assert.NotNull(player);

        var playerOldPosition = new Vector3(0,0,0);
        player.transform.position = playerOldPosition;

        Vector3 targetVelocity = new Vector3(1, 0, 0);
        player.movePlayerXYDirection(targetVelocity);

        var playerFinalPosition = player.transform.position;

        Assert.AreNotEqual(playerOldPosition, playerFinalPosition);
    }
```

IMPLEMENTATION

```
[Test]
0 references
public void YPositionTest(){
    var gameObj = new GameObject();
    var player = gameObj.AddComponent<FirstPersonController>();

    Assert.NotNull(player);

    var playerOldPosition = new Vector3(0,0,0);
    player.transform.position = playerOldPosition;

    Vector3 targetVelocity = new Vector3(0, 1, 0);
    player.movePlayerXYDirection(targetVelocity);

    var playerFinalPosition = player.transform.position;

    Assert.AreEqual(playerOldPosition, playerFinalPosition);
}

[Test]
0 references
public void JumpTest() {
    var gameObj = new GameObject();
    var player = gameObj.AddComponent<FirstPersonController>();

    Assert.NotNull(player);

    var playerOldPosition = new Vector3(0,0,0);
    player.transform.position = playerOldPosition;

    player.Jump();

    Assert.AreEqual(playerOldPosition, player.transform.position);
}
```

TESTING

CONCLUSION

We successfully developed a first person 3D game using Unity and Unity Hub to be able to simultaneously work on separate game features on different branches.