Spring 2019

Mobile Application: Peril

Michael Prough
msp66@zips.uakron.edu

Michael Prough

Honors Project

Mobile Application:

Peril

Contents

Introduction

In recent times, applications have shifted and adapted to work on many different devices from PCs to mobile devices and smart watches. In return, some devices have become more favorable due to their mobility and ease of use, such as smartphones. Some of the applications being made for these devices include mobile video games, which are becoming increasingly popular for their simplicity and the luxury to be played anywhere at any time. Which inspired me to learn how to make a mobile game using coding techniques and resources I have learned in my major.

Tools

For my application, I originally planned to use Android Studio as the IDE to make the program. But, as I started creating the structure, I learned that Unity would allow me to build the executable file for any supported device which included devices like Android TV and iPhone. Using Unity also allowed me to use C# instead of Java and XML as the scripting language, which I was more familiar with. So I made the application using Unity as the IDE with C# as the scripting language and for the database language I used standard SQL. Unity was used to handle the look of the UI and the placement of objects that are too complex for creating in scripts. C# was used to program the objects to behave and change when certain cases were met. SQL was used to access the database that was used to store the scores and other values from the game, as well as to insert scores. There were a few major issues or challenges i had to face in order to make the game however. I had to focus on what the game's rules were or how it would play, how the character would move, how the enemy AI would behave, and a chunk system. Additional challenges that I had to face were the scoreboard system, control schemes, and assets for the graphical look.

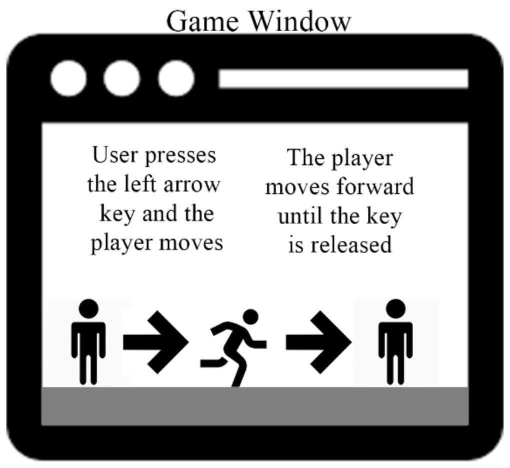
Concepts

In terms of game rules or game design, my game was to be reflective of a platformer and an infinite runner. The idea is the user controls a character that has to navigate multiple obstacles including jumps and enemies. The user is given a virtual joystick and buttons that are labeled attack and jump. If the player falls through a hole, the player dies. If the player collides with an attacking enemy, they take damage. If their health goes to 0 when colliding with the attacking enemy, the game is over. The level will generate chunks and enemies infinitely until the player dies, while also destroying chunks behind the player. While the game is playing there will be music which the user can opt to turn off using a button in the upper screen. Once the user dies they are given the option to give the scoreboard system a user name for your score. Then the system will insert the score into the database. The user will then be taken to the scoreboard scene to see the top 25 scores and their score, granted they didn't make it within the top 25. The user will then be taken to a game over screen where they can opt to go back to the start screen or restart the game. The game flows well with this architecture and the rules are simple to understand.

Assets

The first big challenge involved with making the game was finding assets for each object in the game. In order to find relatively cheap or free assets, I used itch.io to find sprites for the adventurer and enemy sprites, as well as music files. For the background images, i bought a background pack from the unity asset store. The last few assets I had to acquire was a text font for the game, which I also found through the itch.io store. All the assets are either free for use or I bought and have a license for.

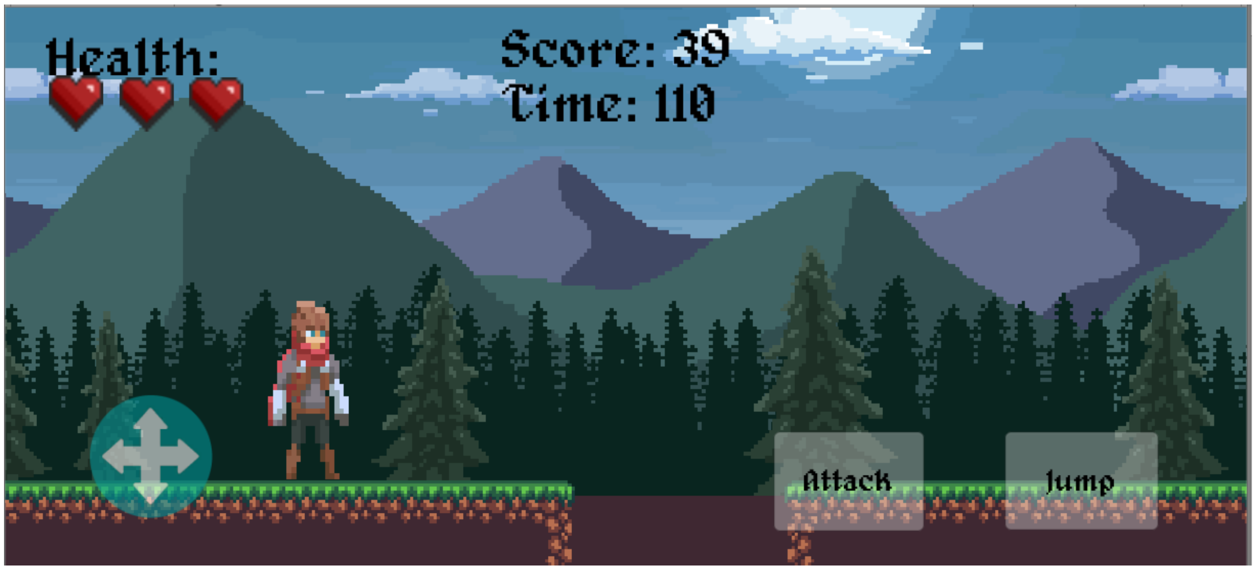
Character Movement


Game Window
User presses the left arrow key and the player moves
The player moves forward until the key is released

For the movement system, there are multiple aspects including player movement, background movement, and enemy movement. The player can move forward and backwards, as well as being able to jump and attack. However, a restriction on the player is that they cannot move backwards past the camera in order to force them to move on and to think strategically about how to proceed. Otherwise the rest of the movement is not restricted. While the player moves, the background and platforms appear to move with the character. The background is made of multiple images which create a moving effect known as parallax. This effect can make it feel as if the trees and clouds or any other feature moves with the player while other aspects can move at a slower rate. For my application, there are three copies of five aspects to 6 aspects depending on the screen being used. Generally the ground, trees and clouds or any image that represents a foreground image move relatively faster than the background elements such as the sky, sun, or mountains. There are generally 3 copies for the reason that each copy is moved to be in front of the player and to create a seamless transition. If there was a singular copy, certain aspects would cut off or missing. For the platforms the player walks on, they are called Tiles which lie in a tile set that controls their boundaries and their settings. The tile set is stationary and does not move. It only appears this way because the player actually moves in the space and so the more it moves in the space, the farther it gets away from the tiles at any point. Another aspect of movement is the enemy objects. Once created an enemy object will patrol the area by walking around on the platform it is spawned on. It will walk back and forth with pauses in between and will detect the edges of the platform so it does not fall off. If the player moves within a certain range of the enemy, the enemy will detect the player and will move towards them as long as they can.

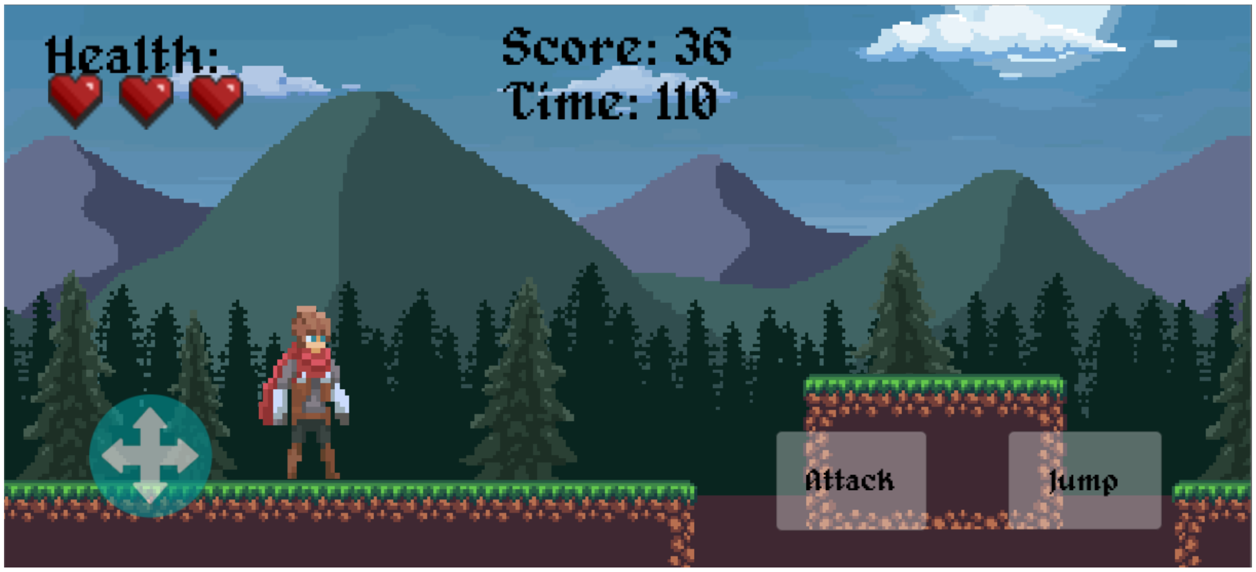
Chunk Generation

Another challenge of creating the game was the infinite chunk generation. I had to create a system to generate and delete chunks of tiles at certain points during the game. The way the generation works is that while the player moves forward, the script will determine when the player hits a tile whose x position is divisible by 16. The script will then look ahead 16 tiles of the player to see if there exists any tiles. If there is none, the script will determine what type of structure will be generated in that space. There are 3 possibilities, one possibility is a hole is made in the terrain, another is a floating platform is made, and the other is a just a flat platform. The first two possibilities have some slight variable factors. In the first possibility, the land is generated normally with a hole somewhere within the block.



In game screenshot of a chunk hole

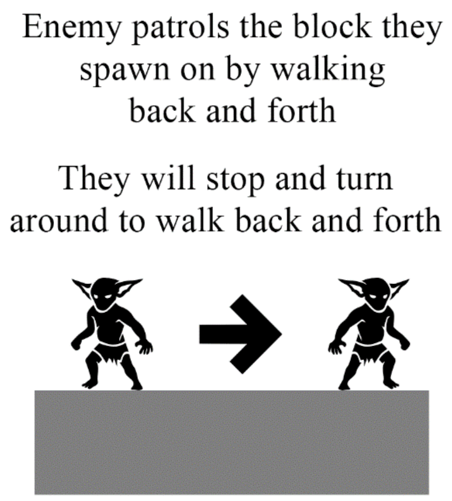
The hole can either be 4 or 5 tiles long depending on a random chance. For the second possibility, within the block a floating platform is created that is surrounded on both sides by small holes to force the player to time their jump correctly.



In game screenshot of a platform possibility

The platform can be either 2 or 3 blocks higher than the normal height of the ground and the holes on both sides can be 2 or 3 blocks wide, which is all decided based on random chance. Another aspect of the generation is the creation of enemy objects. Much like the structures, the enemy will spawn based on random chance. So not every block will have an enemy spawned on it but there exists a chance. If an enemy is to be spawned, the chunk will find a random tile that is occupied by a ground tile, and spawn it there. So an enemy could be spawned on a floating structure or on the ground. However, as the tile set is constantly expanded with the generation, it will eat up storage eventually and slow down the process. In order to combat this, I created a chunk deletion function. It operates similarly to the block generation function where it determines when the player hits a tile whose x position is divisible by 16. The script then looks 32 tiles behind the player, and if there any tiles behind the player past 32 blocks, the script deletes the tiles. Enemies however are not deleted the same as tiles. Enemies are automatically deleted if they are behind the camera a certain number of spaces. As the tiles are deleted or generated, the tile set will automatically increase or decrease the boundaries around the tiles that exist after each generation or deletion.

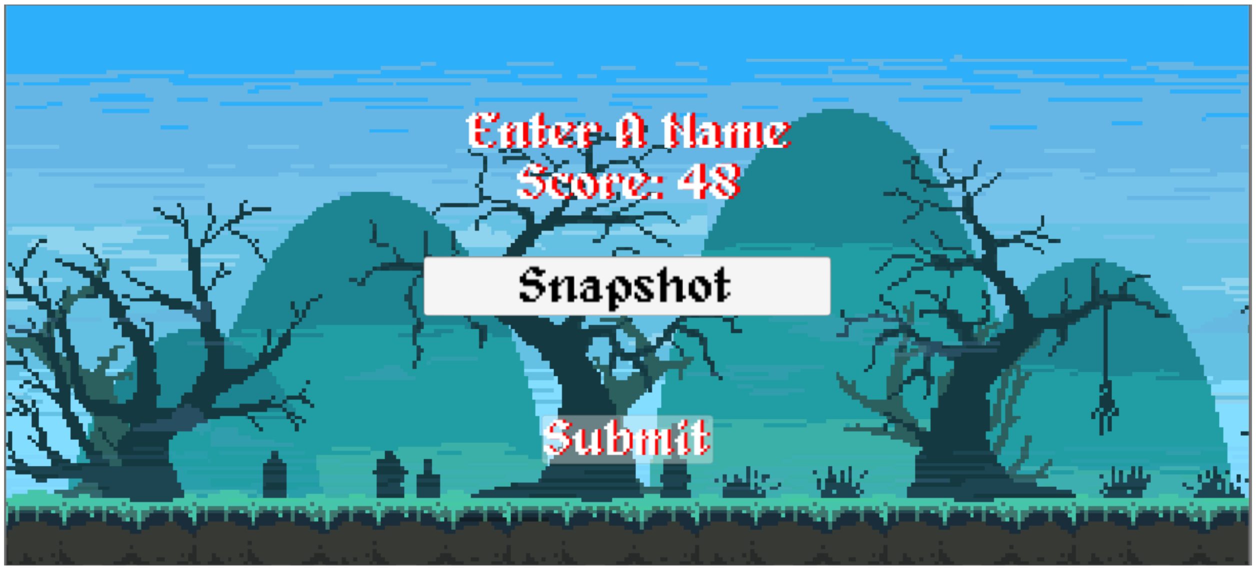
Enemy A.I.


Enemy patrols the block they spawn on by walking back and forth
They will stop and turn around to walk back and forth

The next aspect of the game I had to focus heavily on was the enemy AI. For enemy intelligence I had to carefully lay out how I wanted them to behave. The way I programmed the enemies to function is to patrol the area they spawn on while they can't find the player. Eventually once the player moves far enough within the detection range of the enemy, they will abandon their patrol routine. They will then rush towards the player until they are within attacking range. They can then launch an attack every 2 seconds to damage the player. If the enemy however collides with the character while they are attacking, they will die and activate a death animation. Which allows the user to then walk through them while they die. All motions of the enemy have animations which are controlled through various variables that change based on mathematical calculations or based on other objects' statuses. The AI script overall works relatively well and performs exceptionally well for my first attempt at AI behavior.

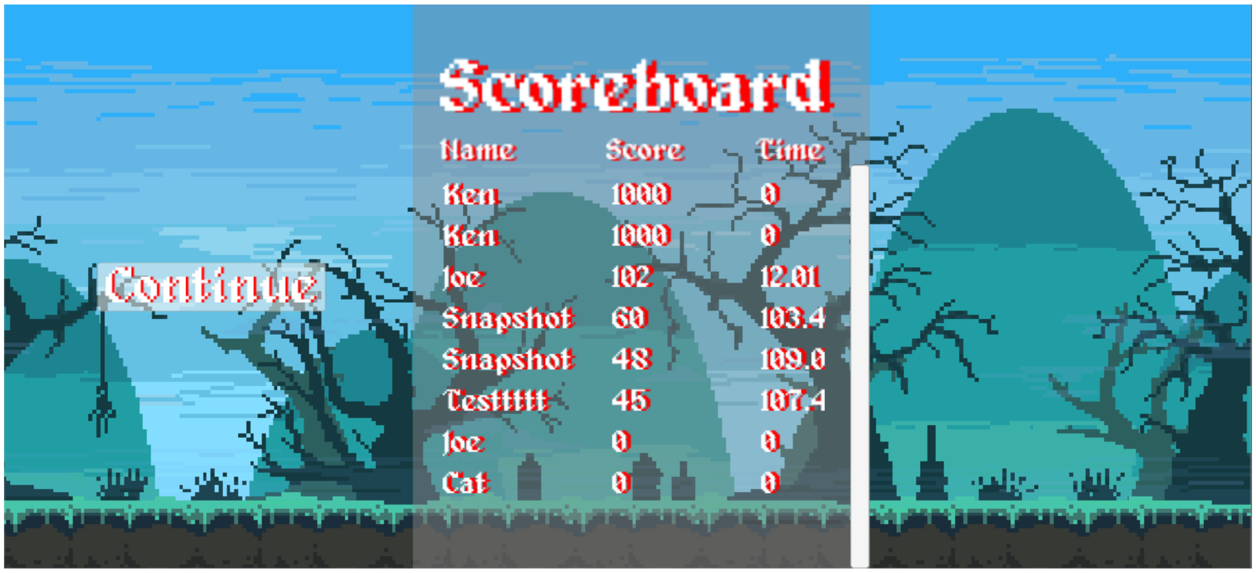
Scoring System

Once the game ends, the user is given the ability to see their score and how they compare to past runs in a scoreboard using SQLite and SQL code. Once the player dies, they are brought to a scene to enter a username or ID they wish to use to identify themselves on the leaderboard. They are also shown their score on this scene as well. Once they click the button to submit their score, it is inserted into the database using C# and SQL to access the database and insert the values they have.



In game screenshot for submitting a name

They are then taken to the scoreboard scene where the user can see a list of past scores from other users and themselves. The scoreboard will show the top 25 scores at the time and then if the user's score is not present in the top 25, it will display their score at the bottom.

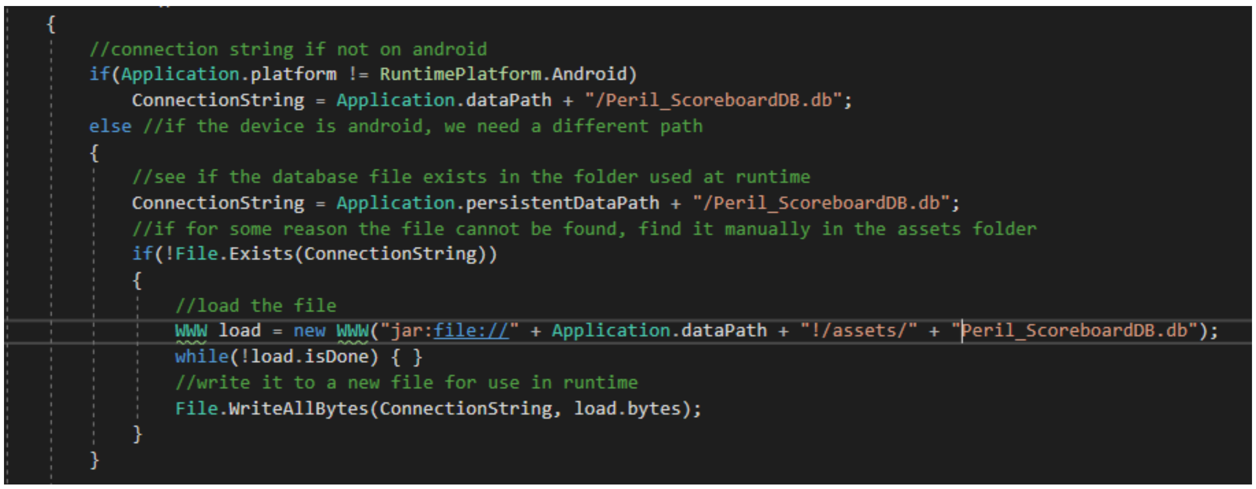


Scoreboard

Name	Score	Time
Ken	1000	0
Ken	1000	0
Joe	102	12.01
Snapshot	60	103.4
Snapshot	48	109.0
Testtttt	45	107.4
Joe	0	0
Cat	0	0

In game screenshot of the scoreboard

This allows for a sense of competition and gives the game more playability. This is achieved by giving the player a goal to accomplish rather than the user running through the game just for the sake of fun. Which can increase a user's interest in the application and give it more run time. A big issue with creating this however was accessing the database for the game to find the data and insert values. SQLite is used by having the database exist as a file within the game's assets where it can be used locally at any time regardless of whether the user is connected to the internet or not. Each device however stores assets and files different for use, and they have different means of accessing those files. To access the database on a PC is relatively simple and the path to find the file requires no special work. However, for Android, the app requires special files within the assets for the database to work and the path to access the database is different. So I had to create conditions to determine if the device being used was a PC or Android. Depending on the device, the path to find the database would change. I also added a special case where if the Android device could not find it within the directory given by the path, it could create the database from the compressed assets folder it has and be written into memory. Which essentially equates to creating a usable database file from what was stored for the application to use.

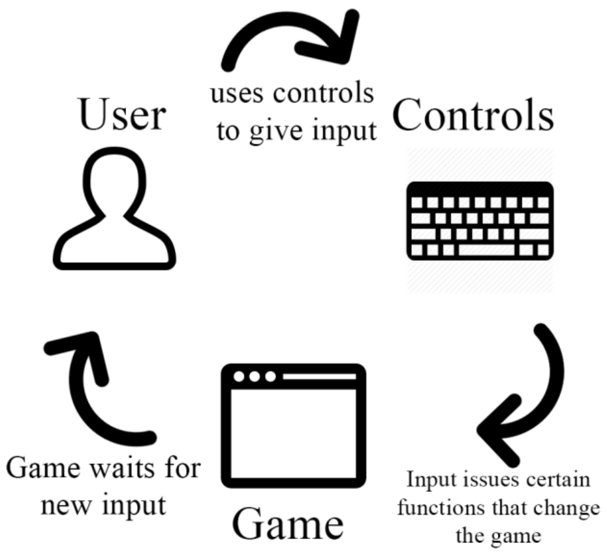


```
{
    //connection string if not on android
    if(Application.platform != RuntimePlatform.Android)
        ConnectionString = Application.dataPath + "/Peril_ScoreboardDB.db";
    else //if the device is android, we need a different path
    {
        //see if the database file exists in the folder used at runtime
        ConnectionString = Application.persistentDataPath + "/Peril_ScoreboardDB.db";
        //if for some reason the file cannot be found, find it manually in the assets folder
        if(!File.Exists(ConnectionString))
        {
            //load the file
            WWW load = new WWW("jar:file://" + Application.dataPath + "!/assets/" + "Peril_ScoreboardDB.db");
            while(!load.isDone) { }
            //write it to a new file for use in runtime
            File.WriteAllBytes(ConnectionString, load.bytes);
        }
    }
```

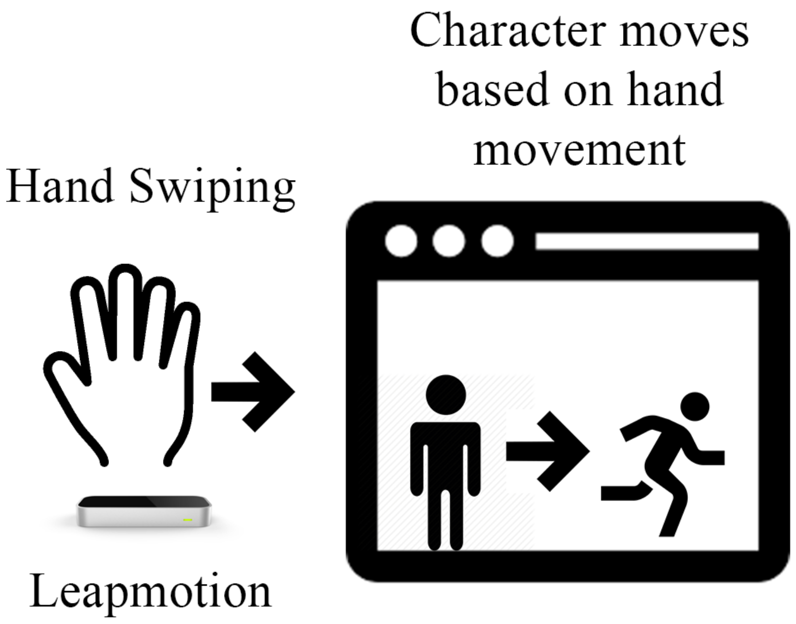
Code from the game on how to find the path to the database for both PC and Android

Creating this code took a lot of trial and error as well as research. The approach to achieve this has changed with each version of unity and update on Android so I had to experiment and try adding different required files until I found a combination that worked. In the end, both PC and Android have access to the database and create the elements listed above to create a scoreboard.

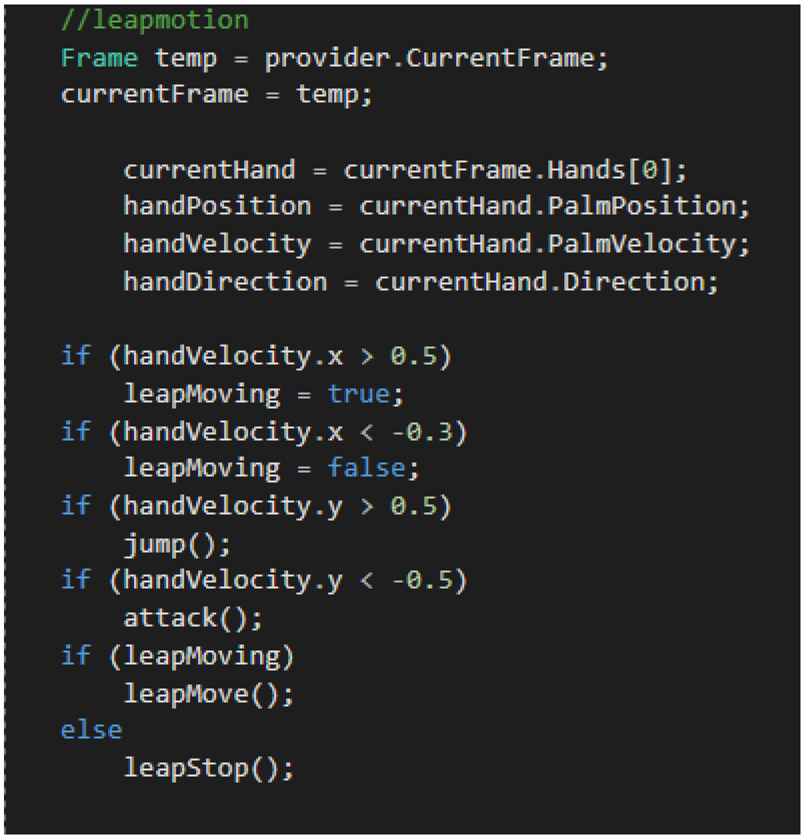
Controls


User
uses controls to give input
Controls
Input issues certain functions that change the game
Game
Game waits for new input

Another way that was designed to test the user’s skills was giving them the ability to use different control schemes. The different control schemes can make the game harder or easier depending on what the user is comfortable with. It also expands the potential user population by giving other users on different devices the ability to play the game. There are 4 different control schemes available for the game including PC controls, motion controls, and the combination of landscape or portrait orientation on mobile devices. Each control scheme barring the mobile device scheme has different methods of allowing the player to play the game.   The PC control scheme uses the conventional key combination of W, A, S, D and space to move about, while using Enter to attack. The user can navigate the screens with clicking the left mouse button. For the motion control scheme, the user will have to load a different version of the game on a PC and connect a Leap Motion device to their PC. The Leap Motion is a device that uses two cameras to detect a user's hands and track them. The developer can then use the tracked hands to detect if they do gestures, such as swiping or pinching.



The way the user interfaces with Peril when using the motion control scheme is that a user can navigate the menus by generally swiping to the right. If there are multiple buttons in the scene, the first option will be clicked if swiped right while the second option will require the user to swipe to the left. For controlling the character, swiping to the right will force the character to start moving. The user can stop the character by swiping to the left. They can also make the character jump by swiping upwards and make the character attack by swiping down.



```
//leapmotion
Frame temp = provider.CurrentFrame;
currentFrame = temp;

    currentHand = currentFrame.Hands[0];
    handPosition = currentHand.PalmPosition;
    handVelocity = currentHand.PalmVelocity;
    handDirection = currentHand.Direction;

if (handVelocity.x > 0.5)
    leapMoving = true;
if (handVelocity.x < -0.3)
    leapMoving = false;
if (handVelocity.y > 0.5)
    jump();
if (handVelocity.y < -0.5)
    attack();
if (leapMoving)
    leapMove();
else
    leapStop();
```

Code from the game on how the hands were tracked and the conditions for functions to occur.

These control schemes allow for users to interact in a new way with the game and it often is rather challenging to master the controls, giving the control scheme a sense of difficulty. The last control scheme is technically two in one, landscape and orientation are similar in that controlling them is the same but the orientation changes how the game looks and can slightly change how it feels due to a difference in perception. If some users who were well accustomed to landscape orientation switched to portrait orientation, they could be thrown off by the look and feel of the orientation. But changes between the two are relatively small, as I took some precautions to try and make sure most essential elements scaled properly.

Future Work

In the future, I would to continue working on Peril to make it more playable and add features upon it. As playable as Peril is, there are some bugs that appear at certain times under certain conditions that are not always repeatable. I would like to iron those out so the game can feel smooth and polished, so that it can be played as it was originally intended to. Another factor I would wish to change would be to work with another database like MySQL instead of SQLite. SQLite provides ease of access and allows the user to connect to the database when not online but it is limited in terms of scale. The database has to be pushed to the users in order for them to receive scores from other users. Using a database like MySQL that requires the user to connect through the internet would be essentially easier and more convenient for the user rather than having constant downloads. It would also allow for different data types as SQLite only offers currently five different data types. I could include values such as dates and user IDs which can help separate users better on the scoreboard. I would also include new features that I never had the chance to add like power ups or new enemies. This could potentially change how the game is played and allow for users to enjoy various strategies of playing.

Conclusion

In conclusion, the mobile application was made with many individual tools and had multiple challenges. The first challenge was finding assets, which was solved by using online resources such as Unity Store and Itch.io. The second challenge was the concept of the game which was met by combining the movement system of a platformer with infinite chunk generation from an infinite runner. The next challenge was the Character movement. There the character was given a system to which they could freely move back and forth, and jump. The next issue was how to handle chunk generation in the game. The chunk generation was handled by randomly generating different types of chunks every 16th tile during the game. The next challenge was the Enemy Artificial Intelligence and how the enemy should act. This was met by giving them idle and patrol routines, as well as a way to detect the player. From there they could chase and even damage them. Following this issue, was how to handle the scoring system for the application. This was handled by using C# in an accordance with SQL to access the SQLite database so that scores could be written to and retrieved. The last issue that was presented was the control schemes for the game. Multiple control schemes were introduced in the game in order to increase its playability. These control schemes included leap motion controls as well as PC controls. These challenges were solved through various means and by using multiple sets of tools to create the application, as well as to help it be viewed as a mobile game.

References

"Core Tilemap Concepts." Unity, Web, Feb. 2017

unity3d.com/learn/tutorials/topics/2d-game-creation/core-tilemap-concepts?playlist=17093

EN, N3K. *"Scrolling & Parallax (Unity 2D) - Game Mechanics - Unity 3D."* YouTube, Web, 16 July 2016

www.youtube.com/watch?v=QkisHNmcK7Y

"Legendary JRPG Battle Music Pack FREE by YouFulca." Itch.io, Web, 13 Aug. 2018

youfulca.itch.io/legendary-jrpg-battle-music-pack

"Rvros." Itch.io, Web

rvros.itch.io/

Studios, inScope. *"1. Unity Tutorial: High Score with SQLite - Intro."* YouTube, Web, 30 Aug. 2015

www.youtube.com/watch?v=laspFwXGprg.

Technologies, Unity. *"Welcome to the Unity Scripting Reference!"* Unity, Unity Technologies, Mar. 2018

docs.unity3d.com/ScriptReference/.

"The Basic Components of Interaction." Unity Modules: Interaction Engine, Web

leapmotion.github.io/UnityModules/interaction-engine.html.

Tsoi, Hewett. *"Alagard."* Alagard Font, Web, 2 July 2013

www.dafont.com/alagard.font.

“Unity Asset Store - *The Best Assets for Game Making.”* Unity Asset Store - The Best Assets for Game Making, Web

assetstore.unity.com/.