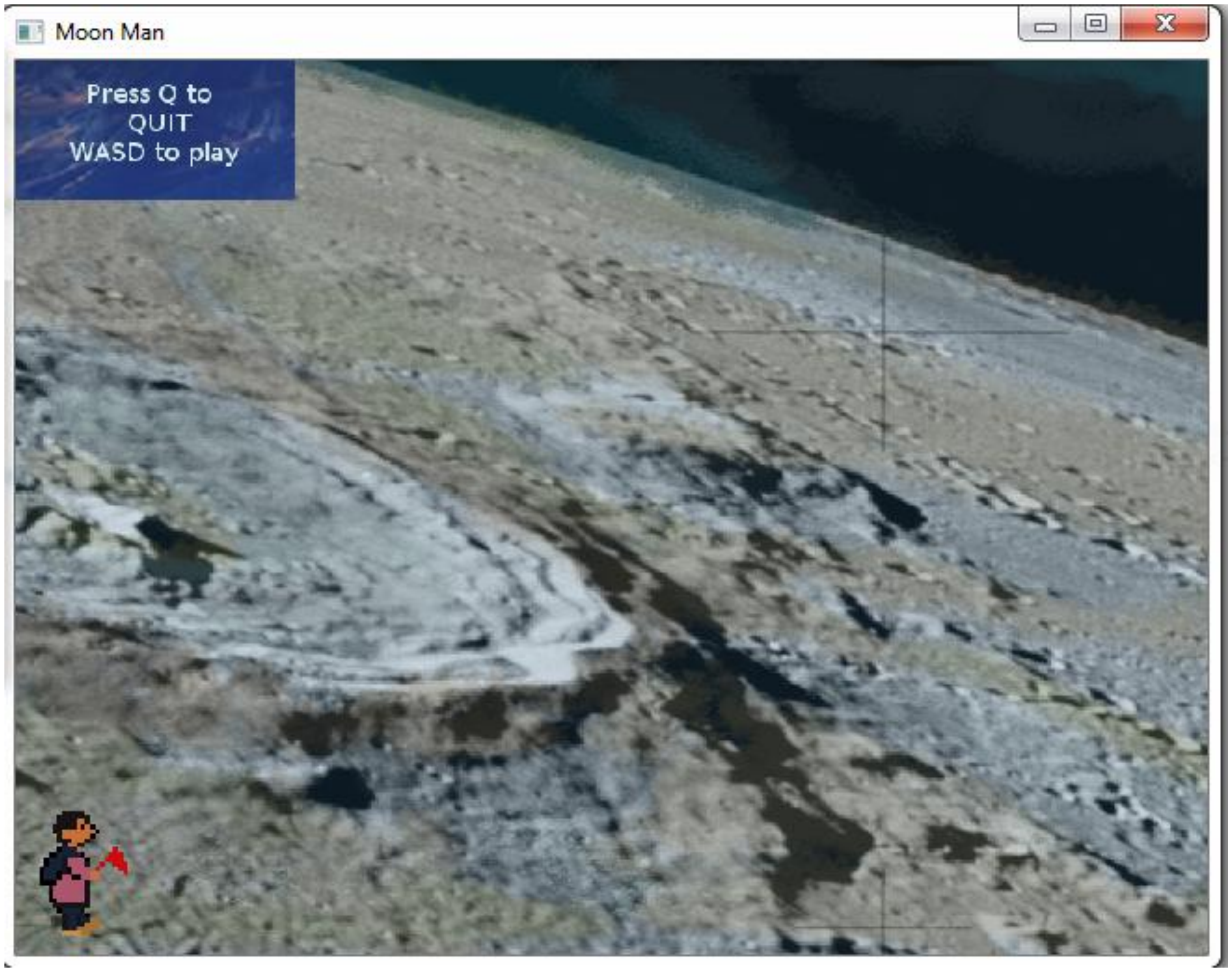
Krystal Maughan

CSIS 137 Advanced C++

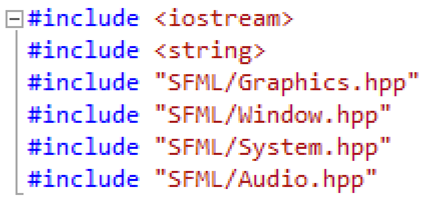
1. **Summarize your project**

   My project is called **“Moon Man: Buzz’s Adventure”**. It’s a game made with SFML (C++ library/ framework) where you would be on the surface of the moon and have to colonize it. You would have to deal with the
   Atmosphere, gravity, and some enemies/monsters that stand in the way and can kill you.



2. What it does

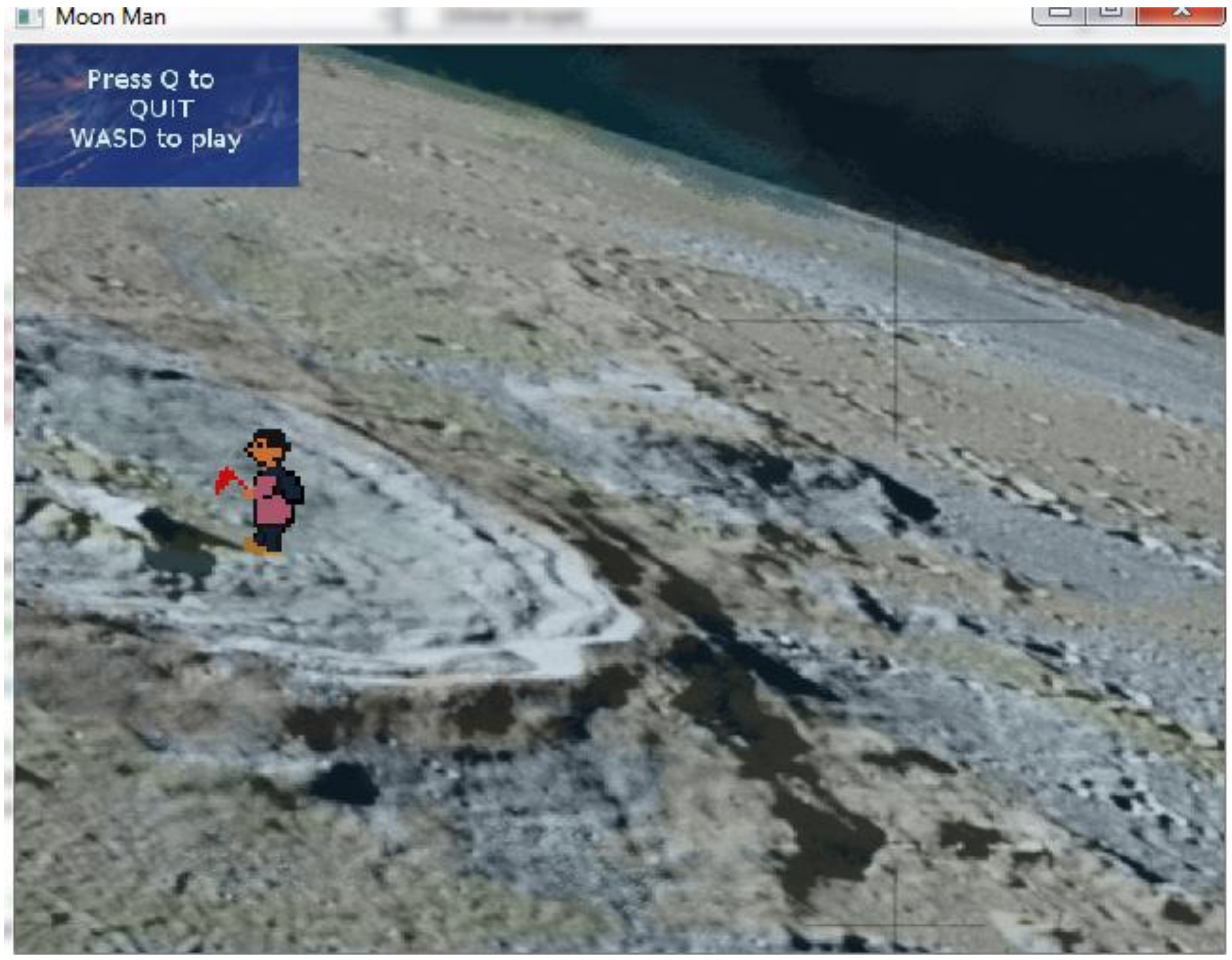
Currently, it uses SFML and C++.



```
#include <iostream>
#include <string>
#include "SFML/Graphics.hpp"
#include "SFML/Window.hpp"
#include "SFML/System.hpp"
#include "SFML/Audio.hpp"
```

3. How to use it

Load Files and Ctrl+5.
When the window comes up, the game loads. Use WASD to navigate "Buzz". Use P to pause the game. Use Q to quit the game.



4. **<u>Explain all class concepts you incorporated into your project</u>**

   I used:

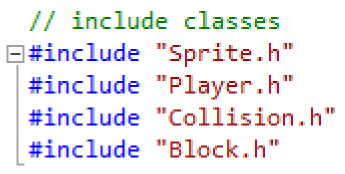
   Main.cpp

   Sprite: animation for sprite by creating bounding box and moving through sprite sheet vector arrays (sf::Vector2u) are unsigned images and you can access x and y of images, etc. It enables a circular traversal of the images so that the sprite array reaches its end and returns to the first index, thus providing an illusion of animation. It also controls player facing left or right based on keys pressed (in conjunction with player)

   Player: movement and input keys for walking

   Collision: check for collision and intersection of objects

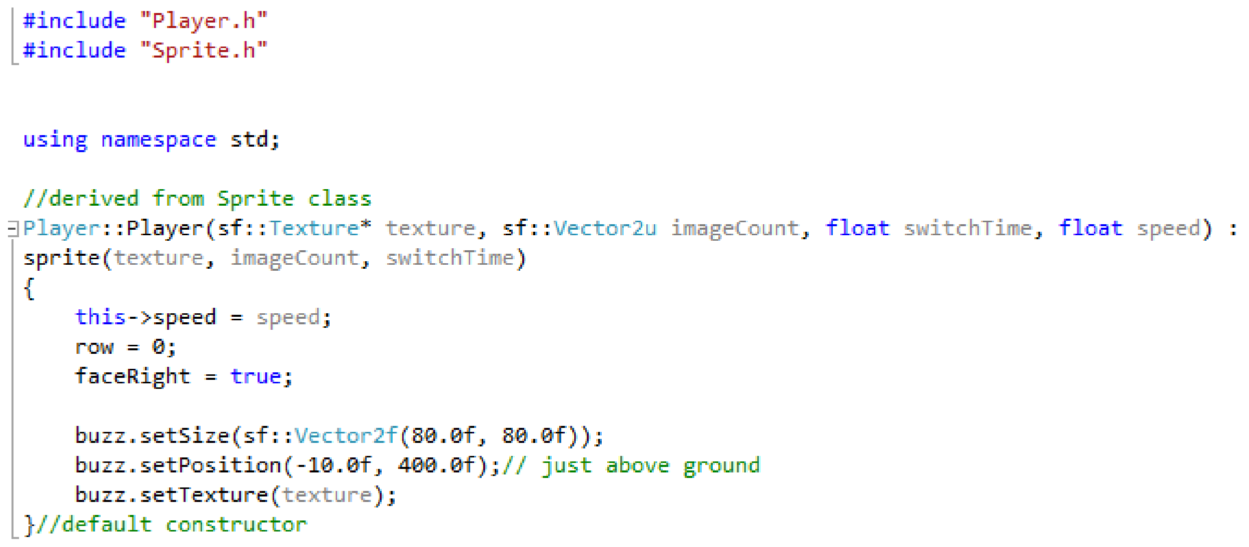
   Block: blocks that use collision (for ground and in future, other atmospheric randomly placed objects)



```
// include classes
#include "Sprite.h"
#include "Player.h"
#include "Collision.h"
#include "Block.h"
```

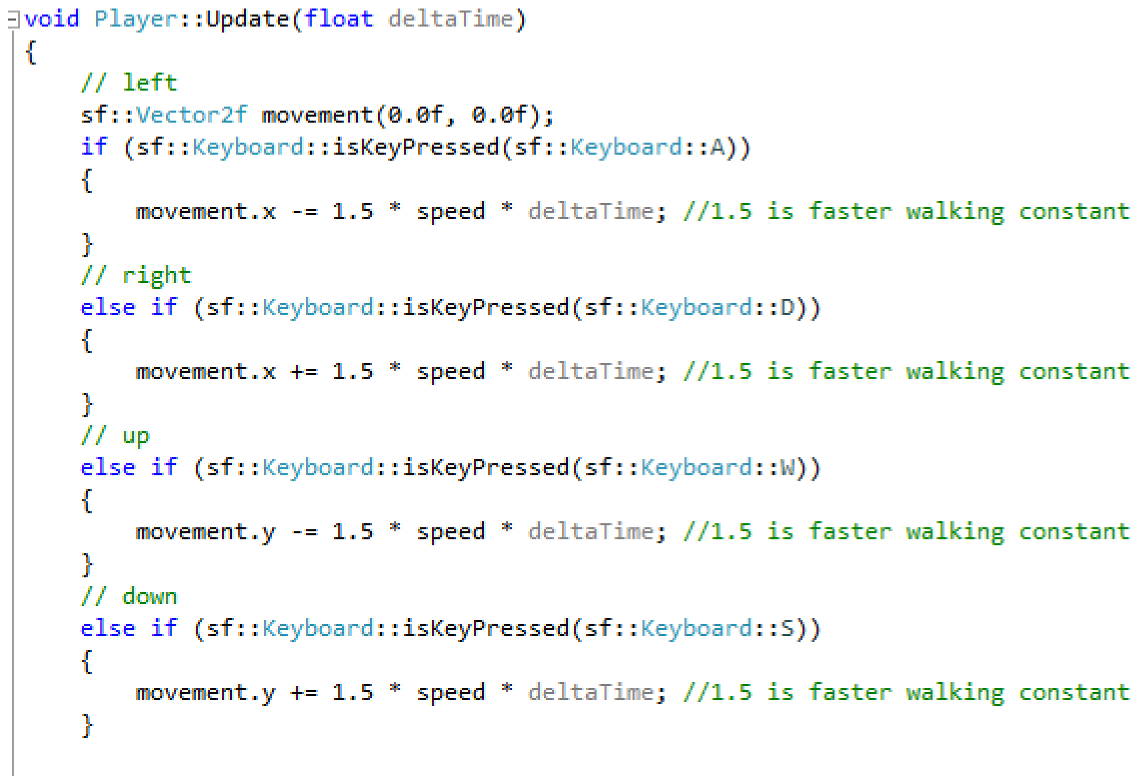
5. **<u>Explain any differences from your final project proposal and why.</u>**

   I didn't end up using Polymorphism, because I had my hands full with the exception handling (see extra credit), several files, trying to get collision and physics to work. But I was able to use Inheritance and Exception Handling (see extra credit), as well as Input Handling/ Key event listeners and I learned about deltas and clamping. Below is Derived class (Player: Sprite(base))



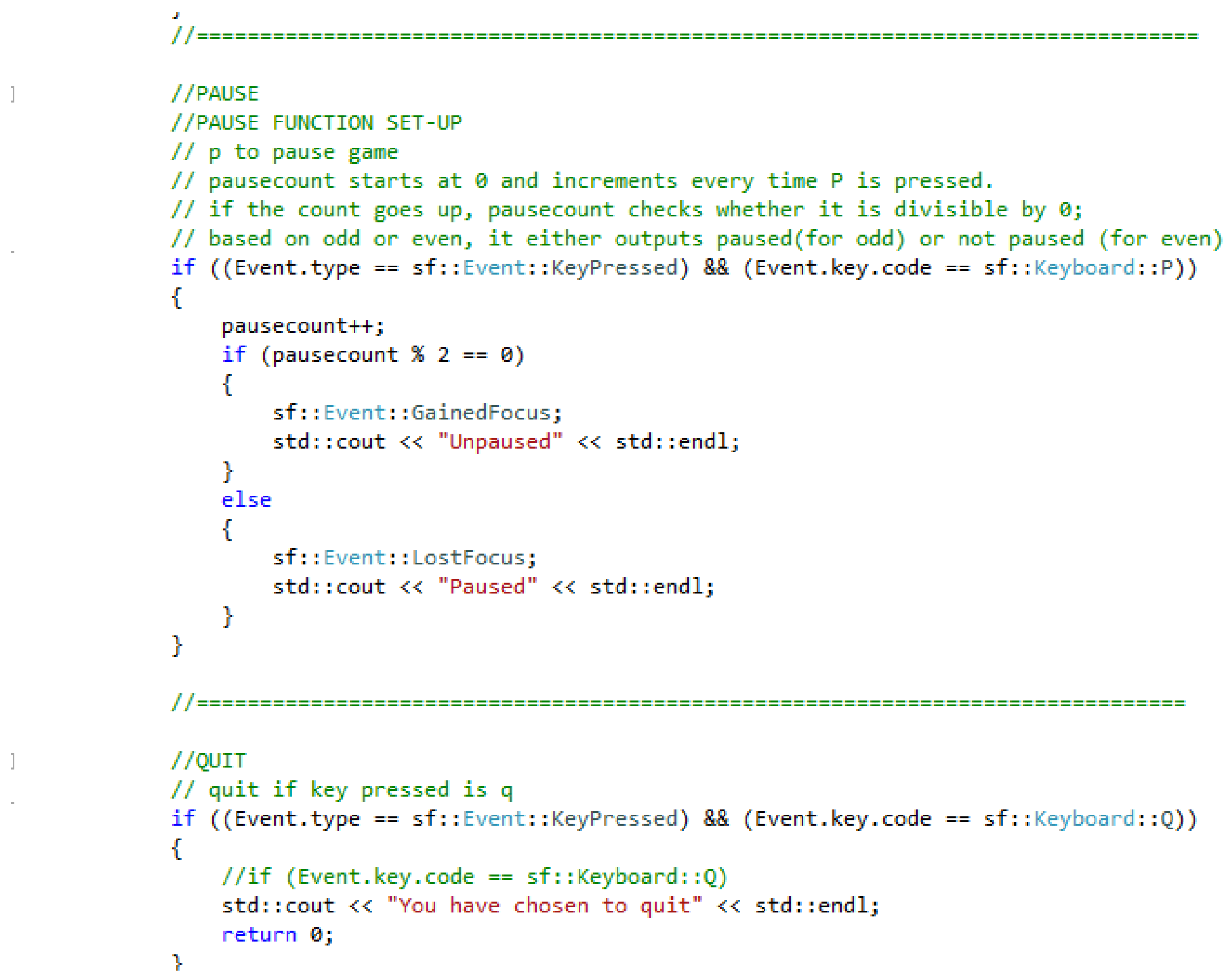
```
#include "Player.h"
#include "Sprite.h"


using namespace std;

//derived from Sprite class
Player::Player(sf::Texture* texture, sf::Vector2u imageCount, float switchTime, float speed) :
sprite(texture, imageCount, switchTime)
{
    this->speed = speed;
    row = 0;
    faceRight = true;

    buzz.setSize(sf::Vector2f(80.0f, 80.0f));
    buzz.setPosition(-10.0f, 400.0f);// just above ground
    buzz.setTexture(texture);
}//default constructor
```

Input Handling WASD Key Pressed



```
void Player::Update(float deltaTime)
{
    // left
    sf::Vector2f movement(0.0f, 0.0f);
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
    {
        movement.x -= 1.5 * speed * deltaTime; //1.5 is faster walking constant
    }
    // right
    else if (sf::Keyboard::isKeyPressed(sf::Keyboard::D))
    {
        movement.x += 1.5 * speed * deltaTime; //1.5 is faster walking constant
    }
    // up
    else if (sf::Keyboard::isKeyPressed(sf::Keyboard::W))
    {
        movement.y -= 1.5 * speed * deltaTime; //1.5 is faster walking constant
    }
    // down
    else if (sf::Keyboard::isKeyPressed(sf::Keyboard::S))
    {
        movement.y += 1.5 * speed * deltaTime; //1.5 is faster walking constant
    }
```

Pause button causes Screen to Freeze or be Inactive and also Unpaused based on Boolean function (pausecount counts and based on odd or even count, determines this)

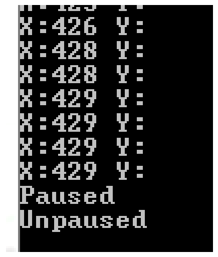


```
//================================================================================

//PAUSE
//PAUSE FUNCTION SET-UP
// p to pause game
// pausecount starts at 0 and increments every time P is pressed.
// if the count goes up, pausecount checks whether it is divisible by 0;
// based on odd or even, it either outputs paused(for odd) or not paused (for even)
if ((Event.type == sf::Event::KeyPressed) && (Event.key.code == sf::Keyboard::P))
{
    pausecount++;
    if (pausecount % 2 == 0)
    {
        sf::Event::GainedFocus;
        std::cout << "Unpaused" << std::endl;
    }
    else
    {
        sf::Event::LostFocus;
        std::cout << "Paused" << std::endl;
    }
}

//================================================================================

//QUIT
// quit if key pressed is q
if ((Event.type == sf::Event::KeyPressed) && (Event.key.code == sf::Keyboard::Q))
{
    //if (Event.key.code == sf::Keyboard::Q)
    std::cout << "You have chosen to quit" << std::endl;
    return 0;
}
```

6. **<u>Also say if you added anything</u>**

Things I did:

- Designed the character and background
- Set scale of each
- Added sound (Load Music) : affects functionality of game..need to optimize (see extra credit)
- Input Handling/ Key Event Listeners (Key Pressed)
- Pause Input Handling using Window Focus/ Lose Focus
- Quit button (press Q to quit)
- Mouse position movement (x and y)-> was helpful in debugging

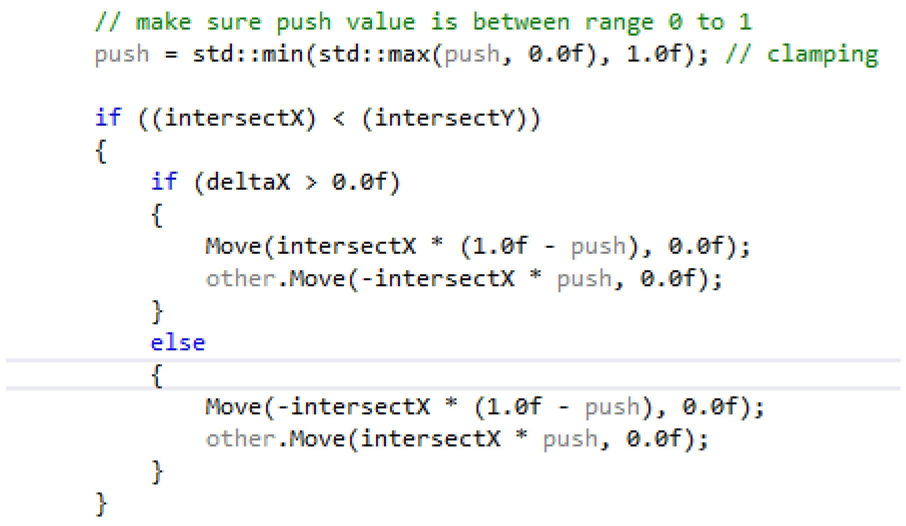


```
X:426 Y:
X:428 Y:
X:428 Y:
X:429 Y:
X:429 Y:
X:429 Y:
X:429 Y:
Paused
Unpaused
```

<u>Things I learned:</u>

AABB Collision (Axis Aligned Bounding boxes) -> how you can determine collision

Clamping (Max-Min function). -> https://en.wikipedia.org/wiki/Clamping_(graphics)



```
// make sure push value is between range 0 to 1
push = std::min(std::max(push, 0.0f), 1.0f); // clamping

if ((intersectX) < (intersectY))
{
    if (deltaX > 0.0f)
    {
        Move(intersectX * (1.0f - push), 0.0f);
        other.Move(-intersectX * push, 0.0f);
    }
    else
    {
        Move(-intersectX * (1.0f - push), 0.0f);
        other.Move(intersectX * push, 0.0f);
    }
}
```

Two-buffer system rendering-> draw images rendered your screen are two buffers that switch from front to back.

Clock provided with SFML is based on your internal system (POSIX), which enables your movement to work on other systems the way it is set up, instead of specifically to your machine. Used for delta functions to determine time passed, and therefore how to calculate movement.

<u>What I'd like to learn/ continue:</u>

How to do a bounding box and apply physics to the environment of wind that pushes the character away.

Add enemies

Joystick control, player score

Better Physics

Make it a two player game (WASD and arrow keys?)

Parallax for the background ie views and boundaries for viewing screen and allowing constrained scaling (small resolution, medium and large)

Better menu

Gravity and ability to jump high like the moon gravity (boosting/ attenuation)

Differences in texture physics: having less friction vs more

Something like randomly generated particles for rain or snow that randomly occur at various times during the game and push or suck the player away from it as an extra challenge.

Optimize sound