Assessment 201.2

2nd-year student Michael Mahuika

ID: 270389266

During the process of learning Unity and C#, I found that Unity has many native functions that aid with the use of C#. This helps with the coding side of things, as the native functions are simple and easy to use once you know how to use them. I faced many troubles during this prototyping process, such as getting the player to flip during left and right movement, making the player jump and come back down, handling the physics of different platforms, and organizing all the files. There were other functions I was in the process of adding, but I couldn't wrap my head around them 100%.

Since I had already learned C++ in my first year, C# was very similar. Additionally, Unity's native functions helped make the process easier. I started with some simple grounding of the player, giving the player movement and a simple attack. These were all handled in two different scripts: playerAttack and playerMovement, both of which used Rigidbody, Box Collider, and Ray Casting. I managed to create a simple player and ground that the player could move back and forth on, as well as jump and hang on the wall.

A challenge I faced during the prototyping process was flipping the player as they turned left and right. This was more of a personal preference and likely wasn't necessarily a big deal, but I felt it was worth addressing early on. This was solved by using transform in an if-statement for my horizontalInput variable with 0.01f and -0.01f. This meant that when the player moves right (> 0), they will transform orientation to face right, and when the player moves left (< 0), they will transform orientation to face left.

Another challenge I had was making the player jump and come back down. I created the function IsGrounded, which was overloaded with Physics2D.BoxCast. This function takes six arguments, with the most important being boxCollider.bounds.center, boxCollider.bounds.size, and the layer specified as an input method in Unity itself. All of this meant that jumping would only happen when the player is on a specified layer. The jumping function was also passed with a serialized field for the jump power, allowing the jumps to be controlled in distance. Unity also has a "gravity" preset, and with both of those together, I managed to achieve a good balance for jumping.

I had some minor issues with 2D physics. I tried to use high friction for 'slime' and low friction for ice; however, it wasn't working at first. After some adjustments, I managed to get the 'slime' to slow the player down and the ice to make the player slide slightly. It would be worth looking into more code for this, as I am not 100% satisfied with the way it turned out. The bounce works great, my biggest issue was trying not to make it too bouncy, as even a decimal change can mean the difference between jumping too high or too low.

I wanted to add enemies and health into the game to give it some more challenging features and make it more enjoyable. I also wanted to add some traps and expand the map. However, for a prototype, I am pleased with its current state. I will keep working on it in my spare time and, one day, have a completed game.

References:

Unity Asset Store. (n.d.). Pixel Adventure 1. Retrieved July 5, 2024, from https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360

Unity Asset Store. (n.d.). Dragon Warrior Free. Retrieved July 2, 2024, from https://assetstore.unity.com/packages/2d/characters/dragon-warrior-free-93896

Unity Asset Store. (n.d.). Ninja Sprite Sheet Free. Retrieved June 28, 2024, from https://assetstore.unity.com/packages/2d/characters/ninja-sprite-sheet-free-93901

Github link: https://github.com/poppedkorn/Ice-Jump/tree/main

I will also provide some videos of testing process'