**Work in Progress Report 3**

Names: Abhinav Agrahari and Matthew Meade

Major developments/breakthroughs(reference specific code please):

We were able to create Laser guns that shoot at the player every *n* seconds. We were also easily able to create images of Al Gore that pop up at the user’s previous death location, a hard mode (Master Mode), and a frontend menu. We added an initial/basic menu to Luke Skywalker, and then updated it (graphically and with content) as needed in later releases.

The array of PVectors for a Sprite’s direction was very useful. Besides using it for updating the hero (user) we also used it to update the moving saws. It really cut down on the long “if” structure we would have had when the time came to change the direction that the saws were moving.

To create a side character that follows the main player, we repurposed the fXstart and fYstart variables in Sprite to set to the position of the hero. Matt created follow(), as seen in Obi-Wan Kenobi and above (later).

The latest and most stable version of *Don’t Give Up* is version 6.5 Qui-Gon Jinn.

Major Challenges/setbacks( reference specific code please):

We had avoided using “atan2()” for the eyes that looked at the user by using vectors. This time we couldn’t do that because we had an image of our Laser Gun that we had to rotate so that it faced the player. We aren’t able to use the rotate function if we passed it a PVector, so we had to use “atan2()”. In “Laser\_Gun” you can see the atan2() function finally at work. One of the most frustrating things about this function was that if you input the variables in the wrong order it works, but if you don’t then it doesn’t work. The processing website says the atan2() function requires an input of a y and x value (in that order) to properly work. When we tried it, that didn't work; we found a website (in the List of Sources document) that switched the order of the inputs, and it worked!

We were also getting a problematic indexOutOfBounds error every time a bullet hit a platform in our main project. After two days, we found out that the bullet was being removed but the rest of the for loop was still trying to check if it was hit. To solve this we used a boolean variable to “skip” the rest of the loop (through the if structure) as break would have stopped the for loop, which we did not want to happen.

Another thing that was a bit time consuming was inputting and saving names. Loading and saving a text file with names was easy enough; the time consuming part was designing the architecture. If you tried to input a name ( but you didn’t type anything) then the program broke. This was fixed as described in the scratch section.

We were very stumped before on what to add to our game to include Parallax Scrolling, so we decided to go with a window in the level. Out of it you can see clouds and fog moving past the moon. We did this by translating the images by different rates relative to the “XCameraOffset” which is used for scrolling.

We were able to create falling platforms by creating a boolean bActivateGravity and set it to true after *n* milliseconds of coming in contact with the user. The gravity function is then simply called on the falling platform.

At first, we thought the code was über-lagging because of parallax scrolling. When the user jumped, and the game scrolled along the y-axis, everything slowed down. We were really frustrated and realized that in our update function for the camera, we forgot to round “fYCamOffset” so the game was translating by decimal numbers causing it to lag. So we have now declared it as an int instead of a float to keep things simple (it became “nYCamOffset”).

Any modifications to your specifications/release schedule:

We made some noticeable changes: one was switching the names for version 4.0 with version 4.1. Most of the remaining modifications that we added were rearranging the incremental features of the releases.

We are able to do 42 levels, but we don’t see the point of doing so as it is very time consuming / redundant. We will now have only 20 levels, the final one being a boss fight. This will allow us to focus more on the cutscenes, animations, and the final boss fight. Below is a reworked plan of our releases.

|  |  |  |
| --- | --- | --- |
| 4.0 | Emperor Palpatine | Spikes strategically placed in the levels. |
| 4.1 | Ewok | LaserGuns that shoot at you and will kill you. |
| 4.2 | Han Solo | Moving Saws. |
| 5.0 | Jabba the Hut | Al Gore - a picture of Al Gore appears instead of blood when the user dies - Get it? *Gore*… Al *Gore* |
| 5.1 | Jar Jar Binks | (Inspirational) text appears every level near “the eye” |
| 6.0 | Luke Skywalker | Frontend Menu to open credits and settings. Settings will allow you to turn audio off. |
| 6.1 | Mace Windu | User inputs his/her name before beginning the game, and that name appears every time “The Eye” says something.   * ex. “You can do it Bob!” |
| 6.2 | Obi-Wan Kenobi | Side character (a butterfly) follows the player |
| 6.3 | Padame | “Master Mode”   * Extreme mode where you restart the entire game if you ever die * Will be accessed using the menu |
| 6.4 | Princess Leia | Parallax Scrolling - the middleground(and foreground) scrolls faster than the background |
| 6.5 | Qui-Gon Jinn | Falling Platforms and 9 more levels   * some levels will have laser guns, while others will have spikes, or saws |
| 6.6 | R2D2 | 6 more levels   * some levels will have laser guns, while others have spikes, or saws, or falling platforms, or an altered field of gravity, or a combination of the above. |
| 6.6.1 | Sebulba | Animated player and animated side character (spritesheet) |
| 6.6.2 | Stormtrooper | Door becomes animated |
| 6.7 | Teebo | 1 more level   * this final level will be a boss fight * the crab will be able to shoot the boss to be able to defend himself |
| 6.8 | Yoda | Cutscenes in between levels |

**Description of your scratch/test program:**

*NOTE:* For this scratch you will have to run it and close it multiple times to truly see the functionality of the scratch.

Describe the generic concept you needed to test out:

Mace Windu (one of our releases) required file input and output to be able to detect whether or not a user had played based on a name inputted. This scratch is called “loadAndSaveStrings”

Source any web site/book that helped you with that concept:

The sites that helped us with file input and output are as follows; all of them are from the Processing website:

<https://processing.org/reference/loadStrings_.html>

<https://processing.org/reference/saveStrings_.html>

<https://processing.org/reference/String_equals_.html>

Describe the code and the lesson that you learned from it:

The program first loads a text file (.txt) into a String array using the function loadStrings()

When we went to loop through the array after the name was entered to check if the user had already played we found out that(according to the Processing Reference website) if we do something like

if (sNameEntered **==** sNameInList)

it will return false. We would instead have to use

if (sNameEntered.**equals**(sNameInList)

We tested that statement, and it seemed to work both ways. We chose to use equals() just to be on the safe side in case something was going terribly wrong with using the equality operator (==) with Strings and we did not know about; it also seemed more… fancier.

After that, the program cycles through the array while checking if the entered name is part of the array.

If the name is not found, it saves the array of Strings as a text file using saveStrings().  
 We learnt that co-ordinating different parts of a project can be a trouble, but if done right should work out in the end.

Describe any challenges that you enjoyed in integrating this scratch code into your major project:

*bNameHasFocus*: a variable for checking whether the input box was clicked on / currently selected or not.

*bHasEnteredName*: a variable for checking whether or not the user attempted to make the program accept the name they entered.

There were no challenges integrating this scratch program into our main project. However, there were many impediments in designing the architecture for this scratch.

If you tried to input a name (but you didn’t type anything) then the program broke. The textWidth() function was outputting an error that it could not calculate the width of the displayed string (because technically the string had no width.

This was fixed by checking if the entered name was greater than 0. If it wasn’t, the variable bNameHasFocus is set to false, bHasEnteredName is set to true and the program continues on from there (displays a message on the screen depending on the name inputted).

In our main project, the variable bHasEnteredName is not set to true until the length of the inputted name is greater than 0.