[[1]](#footnote-1)

Oregon Trail

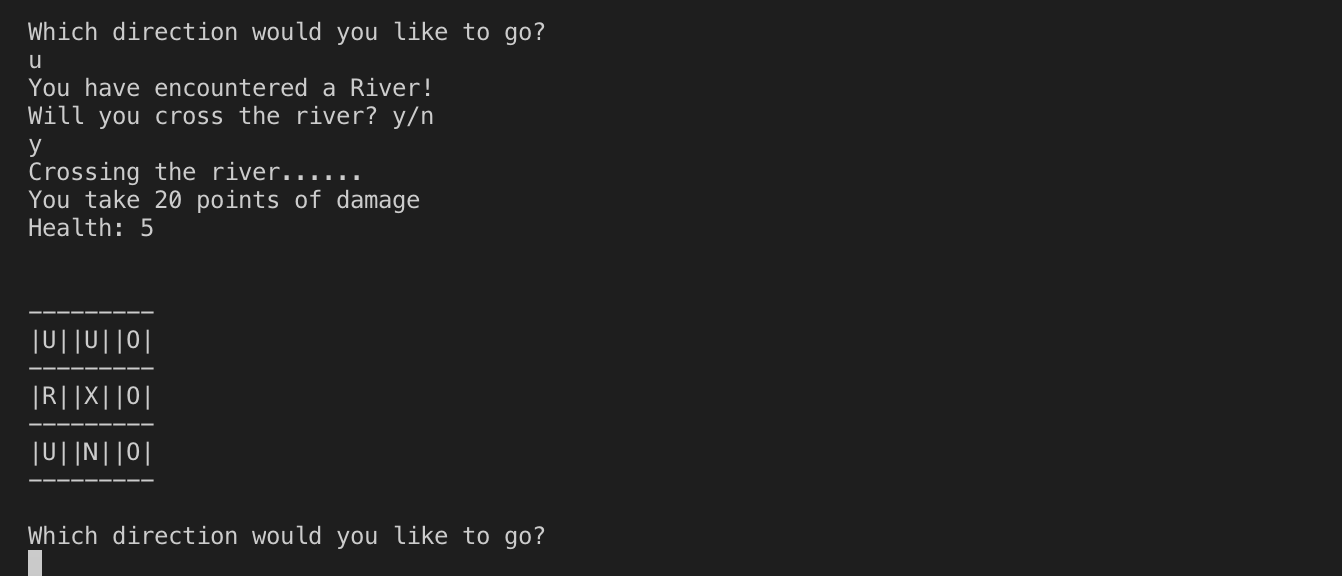
Reddington Reese Walters

*Overview*

The project was to design the old text-style game ‘Oregon Trail’ in C with an emphasis on file input/output, command line input/output, dynamic memory, structs, unions, and pointers. My methodology was to go into the project and complete as much of it as I could following the professors suggestions and use the data structures he required. The outcomes led to learning about the coding language, ‘C’, and becoming comfortable using all of the different aspects necessary for this project. Although this project took ~30 hours and many nights working and asking for help, I feel like it helped me be prepared for future projects.

# INTRODUCTION

T

he project called for created a doubly linked list of nodes which was very hard to implement and took many tries and a lot help to complete. For this project I set out to accomplish creating a game that is efficient in it’s memory management and was bug free. I do not believe I achieved the most efficient or bug free implementation of the game, but I accept that I learned much about the coding language and the various data structures used to help minimize memory usage.

# Application Design

Designing this application was extremely hard at first. Trying to understand the data structures in ‘C’ without any prior knowledge proved to be a challenge. I used the online tutorial provided on the class website to get a general idea of the types of data we would be using and manipulating. For the project, it called for ‘typedef structs’, ‘unions’, and ‘enums’. All of these were fairly simple to create, but using them proved to be more of a challenge. To create each tile of the game board, I created a 2d linked list of nodes that were placed at index’s based on their ‘x’ and ‘y’ value so they would be easily access later. One of the harder parts of the project was figuring our how to use this linked list to correctly link all the nodes together into a map using their directional pointers. After linking the nodes together, the next step was to create a dynamically allocated 3x3 viewport. This was by far the hardest to implement and trying to dynamically freeing nodes from memory then copying new nodes into memory proved to be a big challenge. After that was finally implemented, creating all of the game logic to handle movement and different tiles was simple and straight forward, yet it took a solid amount of time to handle every case.

# Results

My implementation gives good results! I would say it works about 90% of the time and I am able to traverse the board. I can make it to the finish and the game ends. I believe I am correctly dynamically allocating memory while traversing the board, and correctly copying nodes whenever necessary.

However, there are some instances where certain movement patterns will crash the game and throw a segmentation fault. Below are two screenshots of my game showing the boot up and a single traversal across a river.

# Future Work

I believe my application falls short in the way it dynamically allocates the viewport. I do no believe it is efficient. This was the hardest part of the project for me. In the end I had to ask a classmate how they structured their viewport and followed their structure and implementation. After I got help on creating the viewport, I was able to continue on the project and finish the logic of handling movement. The project is essentially finished, except for in a few cases my game segment faults. If I had more time I would be able to trouble shoot these bugs and flesh out the game to where it doesn’t segment fault, but right now I have no idea why it does so.

1. [↑](#footnote-ref-1)