**AERSP 424 Final Project Report**

***The Ohio Trail***

By Robert Blanchard, Luke Fulton, and James Pifer

# Overview

Our group created a version of the game “The Oregon Trail” that is based in Ohio. It is simpler than the original Oregon Trail game, but it is intended to be much more difficult. The player can choose their name, and they must manage their health, food, and money. Each day, the player is given the options to rest, buy food, or travel. The timeline of the game is intended to be in the 1800’s, but there is comedic effect in having modern day scenarios occur. This is a text-based game as creating and using graphics would have been a much larger task.

To manage files between the three members, we used GitHub at <https://github.com/RobertJBlanchard/OhioTrail/tree/main>. We originally met and would work on one laptop together, but we had to split up once we had the base game down. We commented in the code to make it easier to read. We also spoke in an external group chat to help divvy up what everyone would be working on. ChatGPT was used for basic code organization and basic debugging. We also used ChatGPT to give us scenario ideas, but we ended up modifying most of them in the end.

# C++ Project Software Checklist

|  |  |  |
| --- | --- | --- |
| **Checklist** | **Potential** | **Completed** |
| Compilable | 1 | 1 |
| Runtime errors | 1 | 1 |
| Readability | 2 | 2 |
| Math / Technical Knowledge from other areas | 2 | 2 |
| OS independent | 1 | 1 |
| IDE independent | 1 | 1 |
| Backward / Forward Compatibility | 1 | 1 |
| Code Organization | 2 | 2 |
| Efficiency | 2 | 1 |
| Memory Leaks | 1 | 1 |
| Memory Usages | 1 | 1 |
| CPU Usages | 1 | 1 |
| Complexity and Creativity | 2 | 1 |
| All Primitive Data Types | 0.5 | 0.5 |
| Global and Static variables | 0.5 | 0.5 |
| Operators | 0.5 | 0.5 |
| Conditional Statements | 0.5 | 0.5 |
| Iterative Statements | 0.5 | 0.5 |
| Functions | 0.5 | 0.5 |
| Static functions | 1 | 1 |
| Preprocessor | 1 | 1 |
| Pointers | 1 | 1 |
| References | 1 | 1 |
| Function Parameter Passing | 0.5 | 0.5 |
| Containers | 2 | 1 |
| Struct / Class | 2 | 1 |
| Objects | 2 | 2 |
| Inheritance | 2 | 2 |
| Virtual Function / Overriding | 1 | 1 |
| Date & Time | 0.5 | 0.5 |
| **Total** | **50** | **31** |

# Procedures to Start the Game

* Unzip/Extract the compressed folder.
* Open the .sln file with Visual Studio.
* Hit run.
* Enter your name.
* Select 1, 2, or 3 depending on your choice.

# Code Explanation

The code starts with classes and two types of inheritance. Virtual functions are also used to create a math game event, along with another survival event. More classes are used before our vector stating how far the user is from cities in Ohio as they travel. A vector with strings is then used to store the text output seen for the standard events. A complicated event that requires you to answer aerospace math questions is below the standard events. The tolerance of the user’s answer is checked to determine the accuracy of the user’s answer. The output of the current location of the in-game character and the distance to the next city is outputted. The user’s date and time is also displayed. The next chunk of code is what randomly picks an event from above. It also ensures the same event does not occur twice in a row. An if else statement was used to determine the result of every corresponding event.

The main portion of the code displays “The Ohio Trail” as an ASCII graphic. The user is asked to input their name and is then shown an event with the results of their health, money, food, distance, and day shown. If the player has zero or less than zero of the necessities, they lose. This is determined by an if statement and will output to the player that they are a loser. A switch with multiple cases is shown is the player wants to buy food or rest depending on how much food or rest they want. There is a check to make sure the player doesn’t buy more food than they can afford. If the player makes it the distance and wins, their final scores will be shown before they are told that they still lose since they were mauled by a bear when they finally make it to their destination.

There may also be a few more points that I did not mention in this explanation that we counted towards our final total in the checklist above. There may also be a few points that we did not give ourselves credit for because we were not sure. For example, we did not credit ourselves with the Robustness point for handling bad inputs by the user which could cause errors, but under certain scenarios, there are parts of the code that identify a bad input.

# Future Plans

In the future, we would plan to add basic graphics. There is also the opportunity to always add more scenarios with different outcomes and types of interactions.