Final Group Project – Connect Three

Lukas Fuller

12/10/2017

Professor Andrew Mehri

**Abstract**

The purpose of the final project was for the students to understand how to implement the game “Connect Four” and use the concepts they learned during the course to accomplish this. We created a game Connect Three, and used Inheritance for each item as well as vectors for handling player’s positioning on the board or grid. During the project, the major difficulty I faced was figuring out how to get my item, blocker, to work with the code created by my classmates. I worked with Zalika in order to come up with the idea for the item, but looking at the code reminded me how important it can be to use comments for defining each variable for your future self and other programmers that look at your code. Especially in a scenario when multiple programmers are going to need to change and add features to the code.

**Intro**

The main idea for the final project was to create the game connect three with our own set of rules and items to mix the game up. We used the code professor Mehri provided us to get an idea of how the game would operate in the C++ environment. Then we modified the code to suit our ruleset including: changing the win condition from the first person to connect four wins to the first to reach 5 points wins, creating functions for each item in the store, a point system for each score so that the player can use each point to purchase an item from the store, and modifying the Grid.h and main.cpp files to operate using vectors. We were specifically vague in the proposal to how the game would be played, since at the time we were unsure of how we wanted the game to end up. The final product reflects our efforts to use everything we learned this semester with each member working on their own piece of the code.

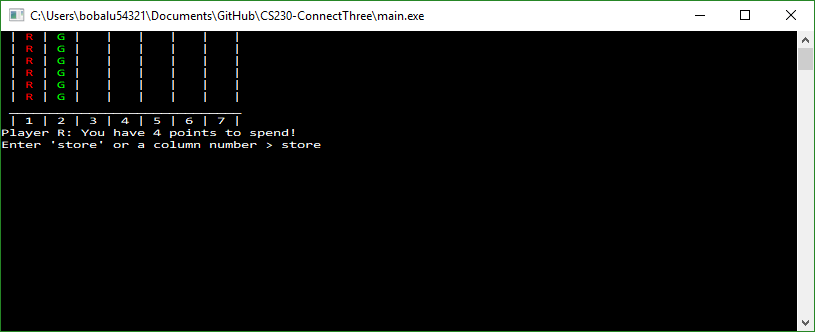
**Procedure/Methodology**

My slice of the code, that I worked on with Zalika, included the Blocker.h file, the implementation in the main.cpp with the blocker item, and the general idea of the blocker function. It started as a morphing item, that could change a piece on the board to the opposing player’s, then into a blocker item that the player could set pieces on top of, and finally into a blocker item that prevents any pieces from being placed in a column. It also required the same amount of points as the boulder item, since it wouldn’t be fair if a player could block a column for 4 points and destroy it for 2. This is more of what we focused on, the game’s balance of our item, as well as implementing it into the code.

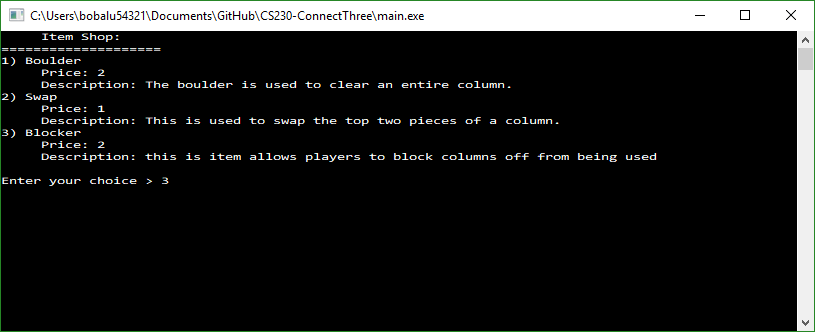
How the Blocker.h file works is by first creating the Blocker class and inheriting Item from the Item.h file using Inheritance to display the blocker’s price and description. After creating the class Blocker, it then clears the screen, calls the Grid::view function, asks the user which column to block, clears the screen, and ends by calling the Grid::block function. The Grid::block function operates by iterating through the matrix and inserting an ‘X’ in each row within the user defined column. Lastly, in the main.cpp file in case 3, the blocker function is called and compares the user’s points to see if they can purchase the blocker, if the answer is yes then the function calls the “use” function which essentially runs all of the code in the Blocker.h file and subtracts the required points to purchase the item from the player’s score. This propitiates the game making it more interesting, while adding a fair feature that each player can use once they reach 2 points.

**Results**

After running the main.cpp file, reaching 4 points, and typing “store”:



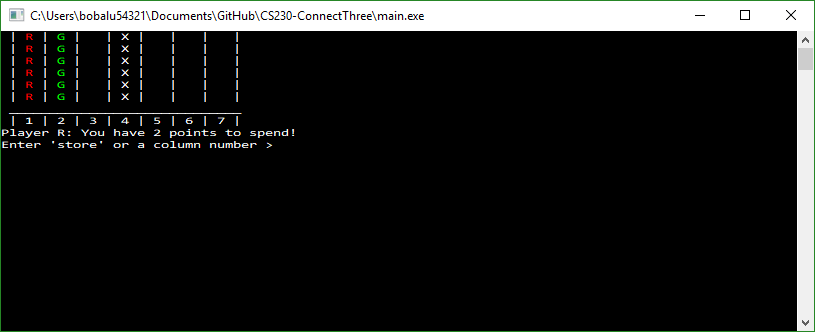
Then typing “3” in order to use the blocker item:



Blocking column 4:



And displaying the result after using the blocker item:



**Analysis**

In this lab we tried to create a game of Connect Four with a spin on it. I would certainly say that designing an item pales in comparison to the development of the grid itself and calling the grid’s functions in main, however the major difficulty I had with my portion of the code was inserting it into the main.cpp file and the Grid.h file. This is mainly because of my unfamiliarity to the other person’s code, and figuring out which variables I needed to use in order to properly initiate the blocker item in the code. It’s always a difficulty when operating in a group of programmers working on the same code. Always ensuring it is easy to follow and insert new functions in the future is certainly an important concept that I think every programmer can always get better at by using comments.

**Conclusion**

During the project, I used Inheritance and vectors in order to call Item class with its set of functions within the grid that used vectors to handle the player’s current positions on the board. I also had to figure out how to implement a function in someone else’s code with their own variables that need to be defined. It wasn’t that hard of a task since they helped with identifying what each item did, however it highlighted for me the importance of using comments in order to define what each variable does. It’s not only to look pretty or look neat for a grade, but so that in the future when you build upon the code, it isn’t difficult to figure out how to add more functionality to it. Thus, making the process more intuitive for yourself in the future or fellow programmers that are working on the same project as yourself.

**References**

Dale, N., Weems, C., & Richards, T. (2018). C plus data structures. Burlington, MA: Jones & Bartlett Learning.