NFL Fantasy Football Mock Draft Simulator

Jayden Slotnick and Payton Glynn

AERSP 424

4/12/2024

I. Introduction

This report will outline the methodology of the Fantasy Football Mock Draft Simulation code. The goal of this project was to utilize existing rankings to create a mock draft simulation environment for the user. Fantasy football is a widely played game by millions of Americans. The game of fantasy football includes drafting real football players onto a fake fantasy team where they score points for you. Typically, there are fantasy football leagues of about 8-12 people, and the rosters drafted consist of major football positions such as quarterback, running back, wide receiver, and tight end. The league settings of every fantasy football league are slightly different, and this code provides the user flexibility to adjust key parameters such as league members, roster limits, and positional limits. In addition, this code will give the user the option to control a team and draft players themselves or have an automated process do it for them. This tool gives the user the ability to simulate three draft scenarios which are a standard league, a PPR (points per reception) league, and a half-PPR (half a point per reception) league. In summary, this code gives a unique user experience for mock drafting and is a potential tool to use before the football season to give a player practice drafting a team.

II. The “Ranking” Class

The purpose of the Ranking class is to take a user input on a particular draft a user would like to run (standard, PPR, half-PPR). The user is prompted for the input of the draft type, and the draft type is inputted into the ranking class. Then an object is created for each league type. The class will then output the rankings of the players for the desired league type. The main function contained in this class is the originalOperate function. This function operates by opening the CSV of the correct league type using ifstream. The rankings are then copied over into a vector container. This process is completed for each league type with 3 if statements so it will only run

the chosen league type. Lastly, operator overloading is used to help print the output of the container. The format of the output is… Player: XXX, Position XXX. The XXX in each would be replaced with real NFL players and their respective positions. The concept of function overloading is also used by the functions operate() and operate(int val). Therefore, if there is no input, the code will run the standard case which is the league case. If there is an input of a league type, the code will run the case specified. At the conclusion of the class, each of the CSV files are closed and the player rankings for the chosen league type is outputted.

III. The "League" Class

The League class is a base class where different classes can be derived from it. In the league class, there are two inputs that are the league name and number of members of the league. Within the class, there are several virtual functions which each have a different purpose. Each virtual function is called within the base league class with the actual logic for the functions written in the derived class(es). The first function is the getLeagueInfo() virtual function which can be used to get information about the league such as position limits and round limits. A derived class will utilize this function to create the logic for intended operation. The second function is the addMembers() virtual function which is intended to be used to add league members to a vector container. The next function is the getLeagueName() function which returns the name of the league. Lastly, there are a series of virtual get functions called getQbLimit(), getRbLimit(), getWrLimit(), getTeLimit(), and getRoundLimit(). These functions are intended to return the values of the quarterback limit, running back limit, wide receiver limit, tight end limit, and round limit respectively.

IV. The "CustomLeague" Class

The CustomLeague class is a derived class based on the League base class. This derived class is designed to give the user a unique input to how they want the league to be structured. CustomLeague starts with declaring some private variables such as the position limits, the league members, and the maximum league members that the code can support. The constructor for the class takes in the inputs of the league name and the number of members. The main function within this derived class is the getLeagueInfo() function. This function starts by prompting the

user to enter a league name that consists of letters and spaces. If the user enters a wrong input, they will be prompted to re-enter the league name. This user input is checked by using a regex pattern of uppercase letters, lowercase letters, and spaces. The user is then prompted to enter the quarterback, running back, wide receiver, tight end, and round limits. There are two lambda functions that are called positionLimitValidation and roundLimitValidation that check the validity of the user inputs. Next, the user is prompted to enter the league members with a limit of 16 league members supported by this code. The next function is a virtual function (std::vector<std::string> addMembers()) to add league members to a container. Based on the user input of the number of members, the user will be prompted to enter names for each of the league members in order of the desired draft order. Lastly, there are 5 integer virtual accessor functions (getQbLimit(), getRbLimit(), getWrLimit(), getTeLimit(), getRoundLimit()) which return the values of the quarterback, running back, wide receiver, tight end, and round limits respectively.

V. The "Draft" Class

The overall purpose of the draft class is to have the user or computer select from a list of players to fill out their fantasy football team according to the rules of the draft. This class consists of private members that initialize important variables to be used later. These include the positional and round limits, player average salaries, containers for the players, and many more. The draft constructor receives the inputs of league members, position limits, round limit, and draft type. In its body, the constructor fills the team player map with the league members, and then chooses which CSV file to open based on the user input. The team player map is used throughout the code to keep track of the team and their corresponding players and positions. It then resizes the positional limit containers depending on the number of league members. Next is the displayTopPlayers function. This shows function overloading because there are two instances of the function. The first instance is considered the default application and displays the top ten players available to the user. The second instance allows the user to input the number of players they would like to see, and then outputs that number of players. After this the promptForMorePlayers function allows the user to see a different number of top players if they were not initially satisfied with their choice. This is useful when searching for a player in a certain position. The function makes sure to check that the user inputs are valid for the situation

and do not cause any unwanted output. The updatePositionCount function then adjusts the position limit container so no drafter exceeds the limit on any position.

Next up is the getUserPick function, which allows the user to pick the player they would like to draft. This function receives an input from the user, checks that it is a positive number, and checks that this selection does not exceed the positional limits. If the input passes all these cases, the player is added to the team's roster. The largest function in the draft class is draftSimulation. This function simulates the computer's pick for the other members of the draft, as well as calling the getUserPick function to have the user make their own pick. The computer makes its pick through using probability rules which will be described later. If the player chosen by the computer plays a position that has already met its limit, the code will output "Moving to next player" and draft a player that will not exceed the limit. If the CSV file runs out of players needed to fill a position, the computer will forfeit that pick and say "No more of the position desired available, voiding pick". Another cool feature of this function is the ability to have a snake draft. A snake draft is where the draft order switches at the start of every other round. For example, if a team picked last in the first round, they would pick first in the second round. This is done by checking if the round number is divisible evenly by two at the start of every round.

The computer's pick was done with a probability method to add randomness to the draft to simulate a real-life draft scenario. The logic for the probability implementation was completed with a function called pickRandomizer. This function aligns the top four players available with the probabilities 40%, 30%, 20%, and 10% respectively. The player is then picked based on these percentages. This was done to represent some of the chaos that may occur in a real-life draft where the top player available will not always be chosen. Next the picks on each team are displayed with the outputTeamPicks function. This function iterates through every player and outputs their picks in order of QB, RB, WR, and TE. This function also keeps track of the estimated average salary of a team which may be important or interesting to the user. Finally, the last function in this class is the operate function. The operate function gives the user the choice to automatically draft their picks for them if they choose. It also handles the case where the draft file cannot be opened for any reason. Lastly, this function calls all other important functions in the class and allows the draft to run smoothly.

VI. The “main” cpp

The main cpp file is what links all the headers together in this project and is where the main functionality exists. The file begins by including all four of the headers, which allows it to instantiate classes from them. The main file has a global variable totalDrafts that allows the user to keep track of how many consecutive drafts they have completed. The main function prompts the user to enter which draft format they would like to run. Based on what the user inputs, the function instantiates an object from each of the “Ranking,” “CustomLeague,” and “Draft” classes. Additionally, in the CustomLeague object, a thread is created to run the functionality of the CustomLeague. After this, the user is asked if they want to complete another draft and if so which type. The process described is done for all three draft types (standard, ppr, and half ppr). This means the “Ranking,” “CustomLeague,” and “Draft” classes all have three objects instantiated from them. Finally, the number of drafts completed is shown to the user, and the code has finished.

VII. C++ Features Used

- Primitive data types – Used variables for char, int (signed and unsigned), double, float, string.
- Global and static variables – Used throughout the code to keep track of variables such as number of drafts completed and others.
- Conditional and iterative statements – Multiple instances of if and if-else statements, and while, do-while, and for loops throughout.
- Functions – Several functions in each class. Each function is described in the above report.
- Preprocessor – Used #include, #ifndef, #endif, #define in multiple files
- Pointers – Used to access the memory address from where the league is created.
- References – Used to pass into a function by reference several times.
- Function overloading – Used to have a default case and a case where a user enters an input to a function. For example, outputting 10 players available by default when it is the

users turn to pick, then allowing the user to enter how many they want to see if they want to see more.

- Lambda functions – usage in the CustomLeague class to simplify actions that were repeated multiple times.
- Containers – map containers, vector containers, and map containers within map containers to keep track of players and teams.
- Classes – 4 classes, Ranking, League, CustomLeague, Draft
- Objects – three objects created for each class based on league type
- Operator overloading – Used in the Ranking class to help format the output for the top player ranking that is outputted.
- Inheritance – Derived class CustomLeague inherits from base class league
- Virtual function/overriding – Usage in the base class league, overriding in the derived class.
- GUI (text based) – How user interacts with the system to draft their team.
- File System handling – Usage of ofstream and ifstream to read from files.
- Regular expression – Used to check if the leagueName string only contains uppercase and lowercase letters and spaces.
- Concurrent threading – 3 threads without resource sharing to run the CustomLeague classes.

VIII. Conclusions

The code in this project provides a unique drafting experience for a user. While there are platforms that exist to provide mock drafts, few have quite the capability that this code provides. Some interesting features include position limits and full flexibility of team size, number of league members, and number of drafts completed. In addition, this code can work with any CSV file with the same format as the files used in this project. Therefore, any user can create a CSV file of rankings and draft based on those rankings. Few, if any, platforms provide that capability. In summary, this project provides a user with a great mock drafting experience to prepare them for the upcoming fantasy football league year.