NFL Fantasy Football Mock Draft Simulator

Jayden Slotnick and Payton Glynn

# **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 they 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.

# **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.

# **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.

# **The “CustomLeague” Class**

The CustomLeague class is a derived class off 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.

# **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. 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 takes in the 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. It then resizes the positional limit containers depending on the number of league members. Next is the displayTopPlayers function. This displays function overloading because there are two instances. The first instance 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 of these cases, the player is added to the team’s roster.

# **Conclusions**

Add some conclusions about the code