Joel Stennett

EN 3233

Final Project Report

December 17, 2019.

**Introduction:**

For this project, I created an ultimate frisbee game to be played on the computer. I am on the ultimate frisbee team here at JBU, and I really enjoy playing it. My original idea was to create a sports game like basketball. However, I wanted to create a frisbee game because I love the game of frisbee, and there are not any frisbee computer games out there. Originally, my plan was to create a text based frisbee game, where the user could choose what happens based on three different options provided by the computer. Whether the action is successful or not is based on probability. I modeled the game based on different video games I have played, namely Madden and NBA 2k. To do this, I wanted to add features where the user could create their own players and teams, as well as simulate or play games. I also wanted a way for an entire season to be played or simulated. I also wanted to implement a way to keep stats of the games, because I love numbers, and I always appreciate looking at stats of games.

In this project, I implemented many of these parts of the project, but I was not able to make them all work together cohesively as I would have liked. In the end, I allowed the user to choose four different methods of gameplay: simulate one game, simulate one season, play one game, or play full season. These methods do exactly what they say they do. As of right now, I do not have a GUI connected to the program, because I had a lot of troubles connecting it to my program. Instead, I created a simple console application that is formatted in a nice way.

**Methods:**

To start, I created the basic infrastructure of the program. First, I created a Player class. I gave the Player class a lot of different properties, as well as attributes. In Madden and NBA 2k, players have a lot of different attributes and skills. I thought that by giving each player these attributes, I could make the gameplay depend on each of these attributes, so that the player who had the better attributes would do better than the player who had the worse attribute. However, this is not implemented yet. I have the infrastructure in place to do this, but I would have to alter my Game class. In my Player class, I also have two methods, one for reading in a list of first names, and another for reading in a list of last names. I have these methods, because my program creates random players, so these methods are used in the Create class to create a random name for each player. I used a list of random first and last names that I found and put into a csv file.

The next class I created was a Team class. This class also uses many different properties to represent different attributes that I needed it to have. It contains many different lists of players which are used for different purposes, because a team has the players on an entire team, as well as players who play only offense, and players who play only defense. Now that I think about it, I think that I probably could use inheritance to clean up the Team Class, as well as other classes, but I am not very familiar with how to use inheritance, and I needed the time to add many more features to my code. That can be done in a future version of this project. The Test class also holds many properties that will be useful in the Game class, because in the game, the program needs to keep track of who has the disc, and who started, as well as which team is better. The Team class also has a few properties that are kept track of in the Season class, like total points and wins and losses. These are necessary for sorting teams for standings and keeping track of winners.

The Team class also has a lot of methods that it uses. The first few methods deal with the actual formation of the team. Like the Player class, the Team class reads in a csv file of team names and team mascots, which I created a list based on locations in the USA, and a few mascots that I have. There are about 30 of each that are read in. There is also a method which fills the Team class with randomly created players. Essentially, this method creates 14 random players, and then puts it onto one team. There are also methods for calculating how good the team is, by calculating the player’s overall attributes, and then comparing them to another team. Another method that is important is the CompCreateLines class. In ultimate frisbee, teams are split into two lines, one is the O (offense) line, and the other is the D (defense) line. The O line always starts the point on offense, and the D line always starts the point on defense. This method splits the team into two lines, as well as splits the players into positions. In frisbee, there are two main positions, handler and cutter. There are three handles and 4 cutters on the field at one time. This method creates separate lines, filling two lines with players.

The next class that I created was a Create class. This class was used by many of the other classes. In this class, I created methods to randomly create teams, and players, randomly generate player names and team names and mascots, creating a line from a list of players, or a team from a list of players, and then put those all together to construct a full team. This class was built throughout the whole program, because I needed different methods to create and fill different players and teams. In this class, I also have methods for the user to create a player or create a team. Essentially, the program just asks the user to enter the different attributes, as well as the team name and mascot. I do not have this functionality working with the rest of the program. The methods work, but I have not taken the time to figure out where to incorporate it. It would be remarkably simple to add, I just have not created the rest of the program to the point where it would be helpful to add it.

The next class that I created was the Game class. This class was by far the most complex class that I created, as is evident because it is more than 1000 lines of code. The overview of this class is twofold, I want the user to be able to both play a game manually or simulate a game. Those are the two types of games that I want the user to be able to play. The game class has a lot of different properties, because there are many different aspects to each game. I created a constructor that takes in two teams, as well as an integer for what type of gameplay the user wants. This creates a unique game and tells the class whether it is a simulation or whether it will be a manual game.

Before I explain the Game class further, I must explain how a game of ultimate frisbee works. There are two teams, who start on opposite sides of the field. One team throws it to the other team, which starts the point. There are seven players on the line for each team. The offensive team catches the disc thrown by the defense, and plays. They throw it to each other, until they get down to the other teams endzone, or they throw a turnover. When a player catches the disc, they are not allowed to move anymore, they must stand still and throw the disc to another player. If the offense turns it over, the defense then picks it up where it landed, and is now on offense. This continues until one team scores, by completing a pass in the other team’s end zone. No substitutions are made during the point, but when the point is over, the team switches who is on the field. When a team scores, the defensive line goes onto the field and pulls to the offensive line on the other team. This is the very basics of ultimate frisbee and is really all that is needed to know for this program.

I started with the simulated game type. Basically, my design for this would be based solely on probability. The game would have two teams, who would have two different overall skill levels. The first method that is called is FullGame. The first method called inside here is StartGame. This method first fills 14 player properties, that will be used in the play game property. Next, the CoinFlip method is called. Essentially, this method randomly chooses who starts the game with the disc. To keep track of this, I have two Boolean properties in Team that are StartPointWithDisc. If it is true, it tells the program that the team started the point with the disc, and if it is false, then it didn’t. Next, the program calculates the difference in the team’s skill level, which is really important for simulating the game. It then calculates the probability of who will win the game based on this overall skill level, and then sets the score for each team to 0.

Next, the program goes into a while loop which checks whether the score for either team is 15. In ultimate frisbee, games are played to 15. Inside of the while loop, the method that is called is PlayPoint. This takes us to a method that first prints the scoreboard, and then puts the seven people on the line for each team depending on who is starting the point with the disc. There is also a property called discLocationY. This basically keeps track of what yardline the frisbee is on. A frisbee field is 110 yards long total. Each of the two end zones are 20 yards long, while the field between the endzones is 70 yards long. Next, the gametype determines the next method. The game type comes from a user input from the Main Menu class, which will be talked about later. If the game type is 1, it means the user wants to play the game manually. If the game type is a 2, the user wants to simulate the game.

First, I will explain how the simulate game works. This is done by calling the simulate point winner method. This method uses a lot of different if else statements. First, it determines who starts with the disc. In this if statement, it then determines who is the better team. If the better team is the team with the disc, then it a random number is generated. This is then compared to the probability that the team will win. This probability is determined earlier and is based on the team’s overall skill level. Essentially, if your team is better than the other team, there is a higher probability that you will win the point. However, in frisbee, you also have a better chance of winning the point if you start the point with the disc. I give 10 more points to the probability if the team starts with the disc and is the better team. However, if the better team is the team that doesn’t start with the disc, then there is a lower probability that the team starting with the disc will win the point. The program does this with both instances of whether the team starting with the disc is team one or team 2, and which team is better. If the teams are perfectly equal, then the team starting the point with the disc will have a better probability of winning the game. The game uses a randomly generated number between 1 and 100, and if the number generated is less than the probability, than the winner is the team who started the point with the disc, otherwise, the other team wins. This winner is returned as the point winner.

The play point winner method is much different, although it uses some of the same concepts. For this method, the yardsGainedDownfield property is vital, as this will tell us how far a pass will go. This method starts the same, by seeing who has the disc to start the point. If it is team 1, the disc starts on the 0-yard line. It then prints out a message to the user determining who they want to catch the pull. The program gives the user three options, all of which are handles, so the people who throw the disc most often. It asks for a numerical input, and if anything, else is input, an exception is thrown, and the question is asked again. Next, this user’s choice is used in a switch statement. Depending on the choice, a player is chosen to have the disc, and the Catcher property, which is of the Person type is set to the player. This Catcher property is used for print statements, as well as keeping stats. Next, we go into a while statement, which will only end when the point is over. First, it checks if the team has the disc because of a pull, or because the other team turned it over. If they turned it over, then I assign the first player to have the disc. Else, it then uses if statements based on the Catcher to determine who has the disc. This then is turned into an integer, which is used in a switch statement. I realize now that this isn’t very elegant, but I cannot use a non-constant property in a switch statement, and I didn’t have time to explore better ways to do this. Then depending on who caught the disc, the choose player to throw to method is called. This method is called when the user’s team has the disc. The determine player to throw to method is called when the other team has the disc. This method takes three player inputs. These changes depending on who caught the disc, but each player has the option of 1 handle, or 2 cutters to throw it to, and it depends on where they are on the field. It then asks the user for a number input of who they want to throw it to. Each player also represents how far of a throw it would be. If they throw to a handle, then it is a throw that is between negative 5 yards and positive 5 yards downfield, as is what happens in real frisbee. This throw has the highest probability of completion. If the player chooses the first cutter, it is an intermediate throw, which is between 5 and 12 yards gained, but has a lower probability of completion. The third options are for a deep throw to a cutter, which is anywhere between 12 and 50 yards gained. However, this method has the lowest probability of it being completed. Then a random number is generated, and depending on the probability of the completion, the pass is either complete or incomplete. If incomplete, the teamTurnedOver property is set to true. If completed, then the Thrower property is set to the Catcher property which is mostly used for stats and printing, and the Catcher is the user’s choice. Next, the DetermineYardsGained method is called. This method takes whatever throw was thrown, and randomly generates how far it was thrown. This does it whether it was complete or not, because even on incompletions, the disc still goes the same distance, it just hits the ground instead of being caught. Then the discLocationY property is altered based on how far it was thrown.

The next step is to go to an if else statement, determining whether the team turned it over or not, if they had, then it prints that out and returns to the while loop. If they had not, then it checks the discLocation to determine if the disc went out of bounds or not, or if the disc was in the endzone. If it was out of bounds, then a turnover occurred. If not, then a point winner is determined, and the Thrower gets an assist stat, and the Catcher gets a goal stat. Then the team with the disc is set to Team0, which essentially is just a trigger to return the winner of the point.

If the non-user’s team has the disc, then the determinePlayerToThrowTo method is called. This does the same exact thing as the previous method, but instead of the user choosing who to throw it to, it randomly chooses a player to do it.

Just as a disclaimer for this entire project, the probabilities used are not optimal to real life performance. I tried altering them as much as I could to make it seem most like results that would happen in real life, but in all honesty, it would take a ton of research to determine the real probabilities. I also was wanting to implement better algorithms, using individual player skill level to determine whether a throw is caught or not, but I ran out of time. However, I have the infrastructure to do this, so it would be decently simple to implement later.

After the point winner is returned, DetermineNextPointsInfo is called. Essentially, this determines who pulls the disc, and updates the scoreboard, essentially resetting everything that we need to be updated and then the function is run again, until one team gets to 15 points. When the game is over, each Team’s total points are updated for the season, as well as total points against, as well as their wins and losses.

As I stated earlier, the Game class is my longest class, and I am proud of it. I encountered many different problems, and I had to problem solve so that the game would be seamless. I want to continue to add to this class to make it more robust, and better. It would be cool to add Artificial Intelligence, or at least just make it so that individual player’s skill has an impact on who wins the point, but I don’t have time for that, as I had a lot of features I wanted to add. However, in future versions of the game, I will be adding those.

The next class that I created was a Season class. The purpose of this class was to allow the user to play an entire season manually or simulate an entire season. Originally, I was going to have a separate League class, which would keep track of team’s records, and the schedule. However, I realized that this would be redundant as a separate class for a league, so I chose to combine it with the season class. Seasons can vary between all different sports, even in frisbee. For this game, the season consists of ten teams, and has a 9-week season, allowing each team to play all of the other teams once. At the end of the season, there is a tournament that includes all of the teams. The format of this is that the 7th seed and the 10th seed, and the 8th seed and the 9th seed play against each other. These are play-in games, so that the actual tournament only has 8 teams. This is a basic version of playoffs for the end of the season. However, there are many other types of playoffs or tournaments that I can choose to incorporate, as well as many other different types of seasons that I could build.

To accomplish these goals, I created the Season class that contains 10 teams, as well as Lists which are used as Standings and the Sorted Standings. In this class, I have a variety of methods. I have a method that fills the class with random teams, essentially constructing the class to be used in the program. In the future, I will include a method that will allow the user to create a league with their own teams, and choose who they want to be in it, but that will come in further implementations.

The most important methods for the Season class are the Simulation methods. Now, these do not necessarily mean that the games are being simulated, it depends on what the user chooses, but that was what I chose to name the classes. The first method that I created was Simulate Week. What this does is it simulates one week of games of frisbee within the season, meaning that every team plays one game according to the schedule. In this method, I have a switch case that is based on what week it is in the season. In each case statement, I have 5 game constructors, with the different teams based on what week it is. This is how I created the schedule. I figured out, while doing a bit of research how to make it so that every team played each other once, and then hard coded it into each case. This might not be the most elegant solution, as it took a long time to figure it out, as well as to type in the teams into the program, but it is effective, because it is correct. Looking ahead, I could create a function that has an algorithm for creating the schedule, making sure that teams don’t play each other twice, but that could be tricky, because there are many different possibilities of schedules, but not many that allow it to work correctly. I suppose machine learning could help with that, but that is beyond the scope of my project. After the games are created, then the games are all played through. How the game is actually played depends on what the user chose at the beginning of the program. The games can either be simulated or they can be played manually.

The Simulate Week method is called inside of the Simulate Season method. Essentially, there is just a for loop that runs and increments a counter which is passed inside the Simulate Week method, which simulates the week. After all of the weeks are simulated in the season, the Calculate Team Point Differential method is called for each team. What this method does is calculates the difference between the points the team has scored, and the points that other teams score on them. This is used to calculate the standings at the end of the season. Next the Print Standings method is called. This method first calls the Calculate Standings method, which orders the Standings (which is a List property in the Season class) first by the number of wins, and then by the Total Point Differential, which was determined above. It then prints out these standings for the user.

We return to the Simulate Season method. The next method that is called is FinishAndPrintSeasonResults. After the standings are printed out, the Simulate Tournament method is called. This method is like the Simulate Season class, but instead of simulating 9 weeks, it simulates 4 rounds of a ten-team tournament. This requires some more hard coding, because the brackets have to be played a certain way. The bracket is shown below in Figure 1.



Figure 1: Sample Tournament Bracket

The Simulate Tournament method enacts this very bracket, and then returns a winner, which is then printed out for the user in the FinishAndPrintResults method. This is essentially everything that is in the season class.

I added a couple methods that might be used in future versions of the project that deals with free agents. Free agents are a big part of any sports season, as they are players that are not on any team and can be picked up by any team to have them play for them. My thought with this was to allow the user to choose a player to add to the team. The infrastructure to do this is in place, because I have a method that creates 50 random players, that are used as free agents. I have not created methods to allow the user to add them to a team, but that will come in future versions of the program.

That concludes the biggest classes that I created for this project, and the most important. The other methods that I will describe below are important, but are mostly more niche, and deal with specific items to allow the program to function well.

One problem that I had with this project was that I wanted to use global constants as I tested my project. Unfortunately, I could not figure out how to do this, so I created class just for this, called FullProgram, meaning that it was important for the full program. In this, I have two constants set, Verbosity and GameType. Verbosity determines what is printed out to the user. In the building stage of my project, I used print statements numerous times to help me debug errors, to show me what was actually happening. This is an important part of building a project, especially in the beginning stages. However, it gets really annoying once I move past these debugging stages. I did not want to delete these lines though, in case I wanted to look at them again. So, I created the Verbosity constant, to help determine what is printed out. I have three possible verbosities, 1, 2 or 3. 1 represents the least verbose. This means that the program only prints out the important info that the user would want to see. This is the verbosity that is shown in the Results section, and that the user will see. If I set Verbosity to 2, then more information is printed out. This means that more information about what is happening behinds the scene, and more information about each game is printed out, but none of it is especially important. When Verbosity equals 3, then everything is printed out to the Console. This is used for debugging, especially for the Simulate Game methods. I used print statements to look at the probability, and make sure that everything worked properly. It was necessary at the beginning of the project to use these, but now that those methods work, I don’t need the debugging statements anymore, which is why I have the verbosity.

The other constant is GameType. This was used originally to determine whether the games played would be simulations or manual play. This was used to test each function, but then I switched it over to ask the user for how they wanted to play the games, so it is now useless.

The next class that I created was the Test class. This was my main class, because it contained the Main method, which was called every time the program ran. The reason I called this class Test, is because this was what ran the code, and allowed me to test every part of the program. At first, I would test individual methods and classes, but now I just use it to start up the program I created. In the main method, I create three objects, one for MainMenu, which I will talk about later, one for Season, and one for Create. In this program, I run it in a while loop, so that the program only ends when they want to end the program. I first call the MainMenu class, which is a basic that prints out a basic main menu for the user. This menu gives the user 4 options: play a single game, simulate a single game, play a full season, and simulate a full season. The computer reads in the input and sends that back to the Main method. This is then used in a switch statement, that is a state machine that determines where the program will go next. If the user chose to play a single game, the method generates a random game, and then calls the method that allows the user to play the game. If the user chose to simulate a single game, then the same thing happens, but this time the game is simulated. If the user chooses a three, then the method to simulate a season is called, denoting that the user wants to play the season, not just simulate which is what the fourth options was. After the user chooses, and the program does what they want, the computer than asks whether they want to play again. The user inputs either a 1 or a 2 to play again, and the computer doesn’t except any inputs besides integers. This is what the Test class does.

The next class that I built was the Print class. This class is basic, it just contains many different methods to print information out to the user. Right now, it contains a method to print out a team roster, a method to print out the offensive line of a team, and a method to print out the defensive line of a team.

The last method that I wrote was to perform a Unit Test, which was a requirement for this project. I was not familiar with unit tests, and I did not know how to incorporate it into my program. What I should have done is used this test earlier in my project, when I was having a lot of troubles and needed a lot of debugging print statements. This would have helped me understand where my errors were and condensed my code. However, for the purpose of learning how to use the unit test, I created a unit test project in visual studio. The test I was going to run was to test the random number generator, which I was using in my simulate game methods. I wanted to make sure that it was doing what I thought it was doing. Essentially, I tested whether the number generated was actually between 1 and 100. After running the test, it was determined that the random number generator actually worked, which was a good confirmation, and I got to learn about how to use the Visual Studio unit tests.

The UML Class Diagrams for all of my classes are shown below in Figures 2 and 3. These show all of the fields, properties, methods that each class uses, and are explained above.

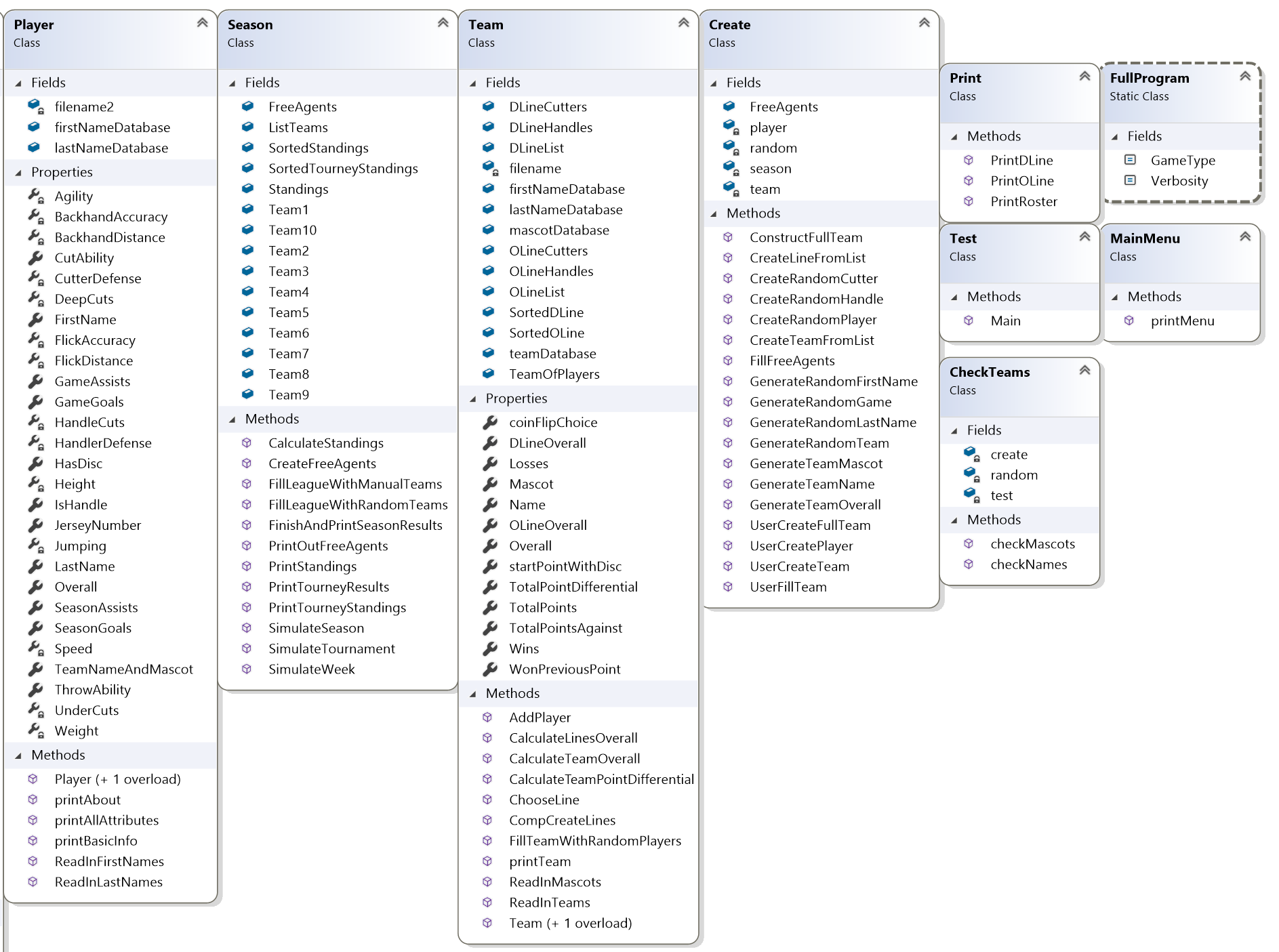


Figure 2: UML Class Diagrams Part 1

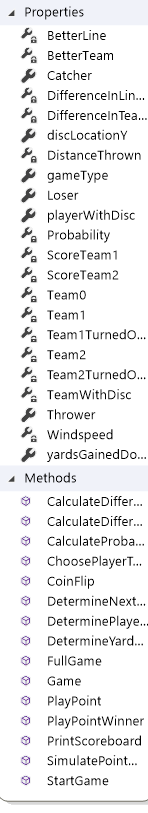


Figure 3: UML Class Diagrams Part 2

**Results:**

It is difficult to gauge the success of my project. The expanse of this project is massive, and would require many more hours to fully complete, and to add all of the features that I want to someday. This is a really good thing, because I know that this project can be made better, and I am excited to continue to work on this program in my free time. In the meantime, let me explain what my program does, and what works and doesn’t work.

The program that I have built allows the user to choose from four options of what to play. The four options are to: play a game, simulate a game, play a season, and simulate a season. This is done using the Main Menu method, which is print out to the console, and then a user input is read in from the keyboard.

Before I move forward, I must explain something. I did not create a Graphical User Interface (GUI) using a Windows form for this project. I spent numerous hours trying to do this, but I never had any luck with it. I am unfamiliar with GUI’s for the most part, and I had a lot of trouble with understanding how to set it up, and how to have it interact with my current program. When I started this, I was almost done with the rest of my program, which meant that I would have had a lot I needed to convert into a GUI format. When I started having trouble, I decided to just give up, and design my console application in a better way, so that it looked nice for the user.

I think the biggest problem I encountered with a GUI was trying to connect it to my project. It was hard for me to find how to connect it to a GUI, and how to receive user input and then display that information. I am sure it is actually pretty simple, but when I looked online, I never seemed to find what I needed. I did try though. In retrospect, I should have gone into Dr. Gilmour’s office hours, and sat with him trying to figure it out until I could figure it out and stopped having errors. If someone who had experience with GUI’s showed me how to connect it, I am sure I could have figured it out, and I would have had a Windows Form that connected with my program. However, this was not a huge deal for me, because I had spent a lot of time in making my program very quality, and so I was not as worried about the interface with the user, even though I recognize it’s importance. My plan is to continue this project in Mobile Web Applications next semester, where I can turn this program into an app that can be played on a computer or a phone.

Continuing with what I was saying before, the Main Menu method prints out a game menu for the user to allow them to choose what they want to do. This screen is shown below in Figure 2.

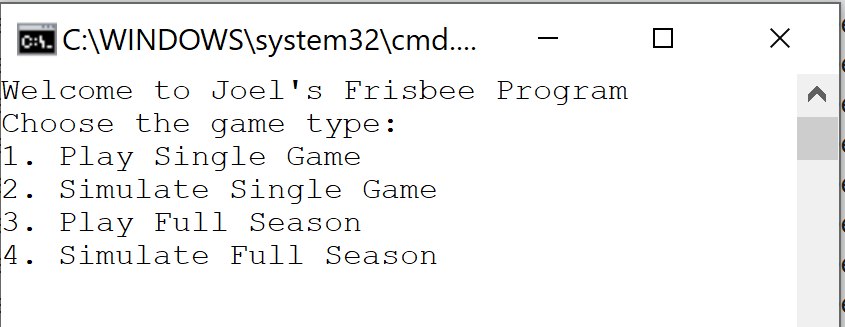


Figure 4: Main Menu

Each of the options that the user can choose connects to a valid method, and it will not break based on users input. I used a try exception here to make sure the user inputs an integer, and not anything else, and continues to ask the user until they do input a number. This exception is shown below in Figure 3.

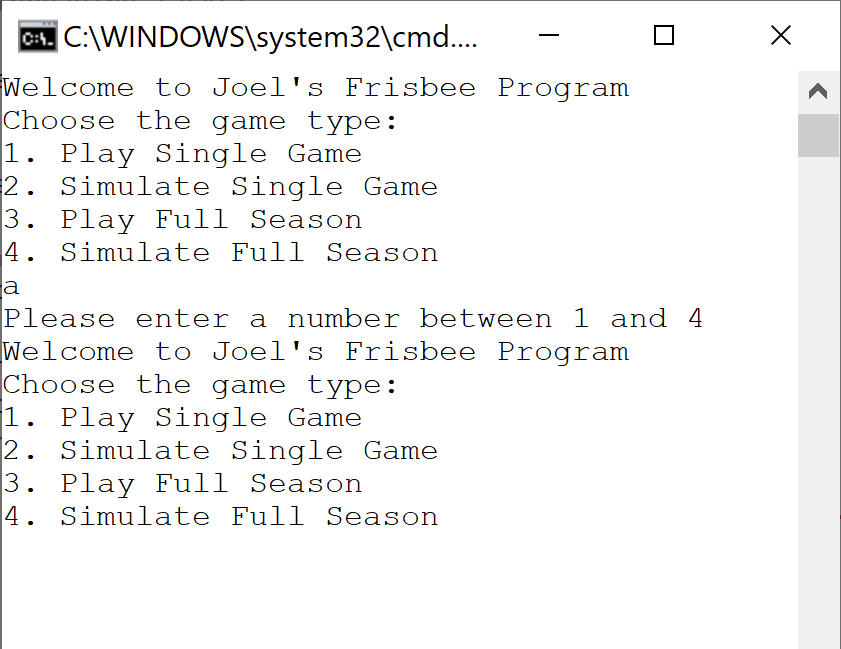


Figure 4: Invalid Input Entered

As you can see, the letter a is input, and the main menu is printed again prompting the user to enter a number. I will first show you the first game type, playing a single game. To do this, the number 1 is input. When this happens, a new game is created, with two random teams generated. The names do not really make sense, but future implementations may include a smarter method for creating team names, based on their location. That is not a worry of mine right now though. There is a lot that is going on behind the scenes here, but that was explained above in the Methods. In this example, San Francisco won the coin flip, and will receive the pull, meaning that they will start the point with the disc. For this program, the user’s team will always default to the first team created. Later, I will allow them to choose which team they want to be, but that will be a future feature.

The user is asked who they would like to receive the pull. Their three options are the three handles on their team’s O Line. I accidentally type in a letter, and the program catches that, and again asks the user for an input. This is shown below in Figure 4.

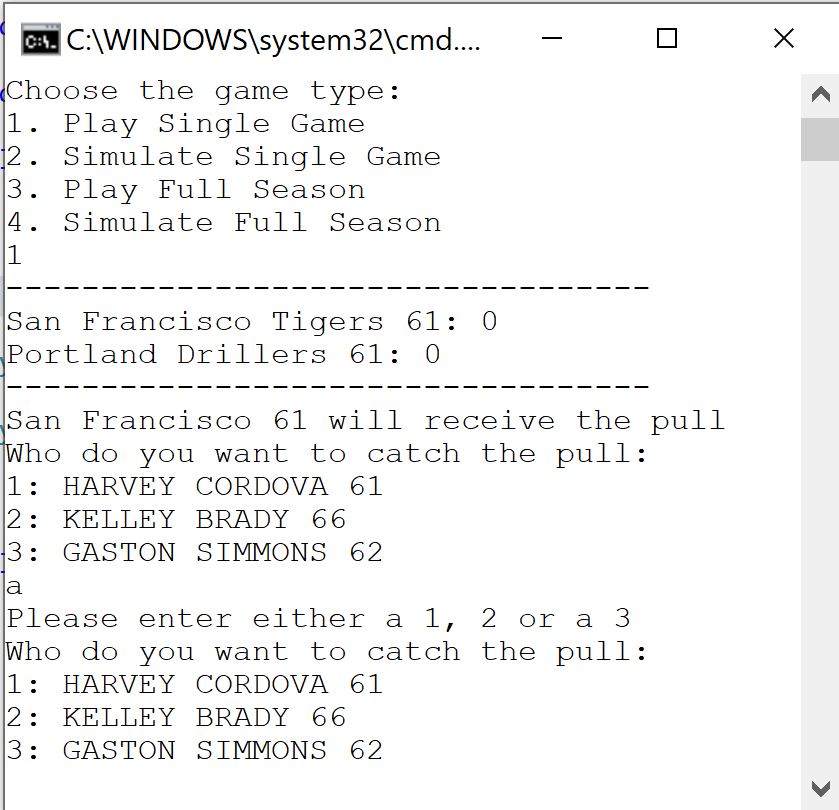


Figure 5: Invalid Player Choice Entered

I then enter the number 2, because he has the best player overall (the number next to their last name). Right now, that number does not mean anything, but it will affect the program in future implementations. When the number 2 is entered, the computer prints out that he has caught the disc on the 0-yard line, because the disc is always caught on the goal line on a pull in this game (not in real life though). It then prints out a message asking the user who they want to throw it to, as well as what type of throw it would be.

If you remember from earlier, the handle reset is a throw with a high catch probability, that goes between -5 yards and 5 yards upfield. An intermediate cutter throw has a lower catch probability, that goes between 5 yards and 12 yards upfield. The deep cutter throw has a comparably low catch probability, and that throw goes between 12 yards and 50 yards downfield.

For this example, I enter a 1. Notice what the program says. On a completion, it states that the pass is completed to the player, and where the disc is. This is shown in Figure 5 below.

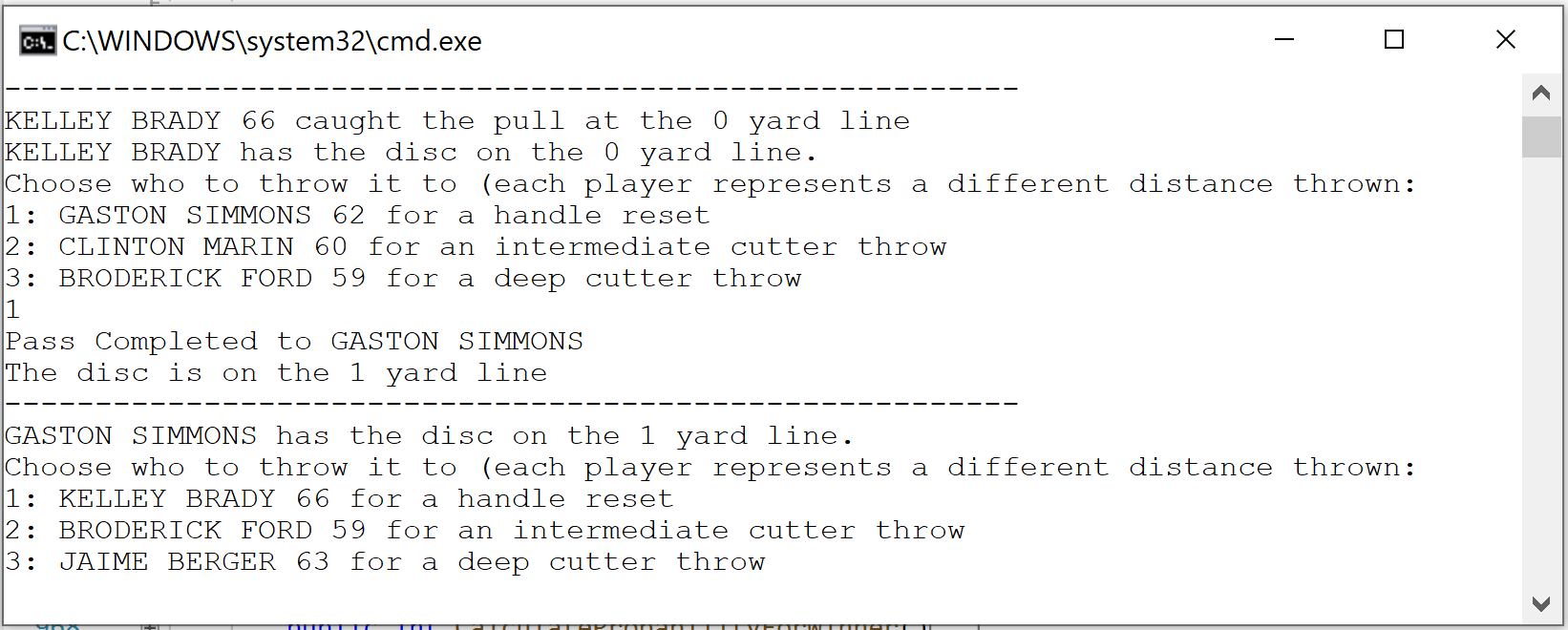


Figure 6: Handle Reset Throw

The program then asks for another player to throw it to. This time, I want to throw it a bit further, but still with a good chance of completion, so I enter a 2. I think this should be a completion, but then I receive a message that there is indeed a User Turnover, meaning that the other team gets the disc. It then says the disc is on the 6-yard line, meaning the computer only has to move forward 6 yards to score. This is shown below in Figure 6.

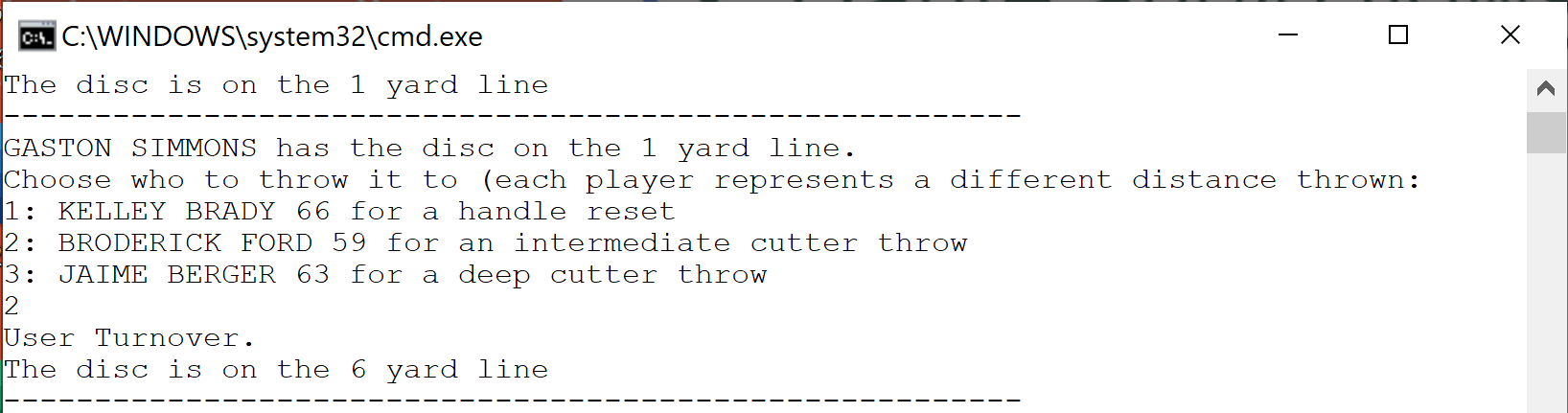


Figure 7: User Turnover on an Intermediate Cutter Throw

Next, the program prints out what the computer decides to do. This looks generally the same as when the computer has the disc, so I altered the outline from dashes to asterisks. The program runs, and the first pass is a completion back to the 7- yard line. The time that it takes to complete this play by the computer happens ridiculously fast, so I have a delay statement included in the code, so that the user can actually track what the computer has done. The next play is a computer turnover on the 4-yard line, which means that the user gets the disc back, starting on the 4-yard line. This is shown in Figure 7 below.

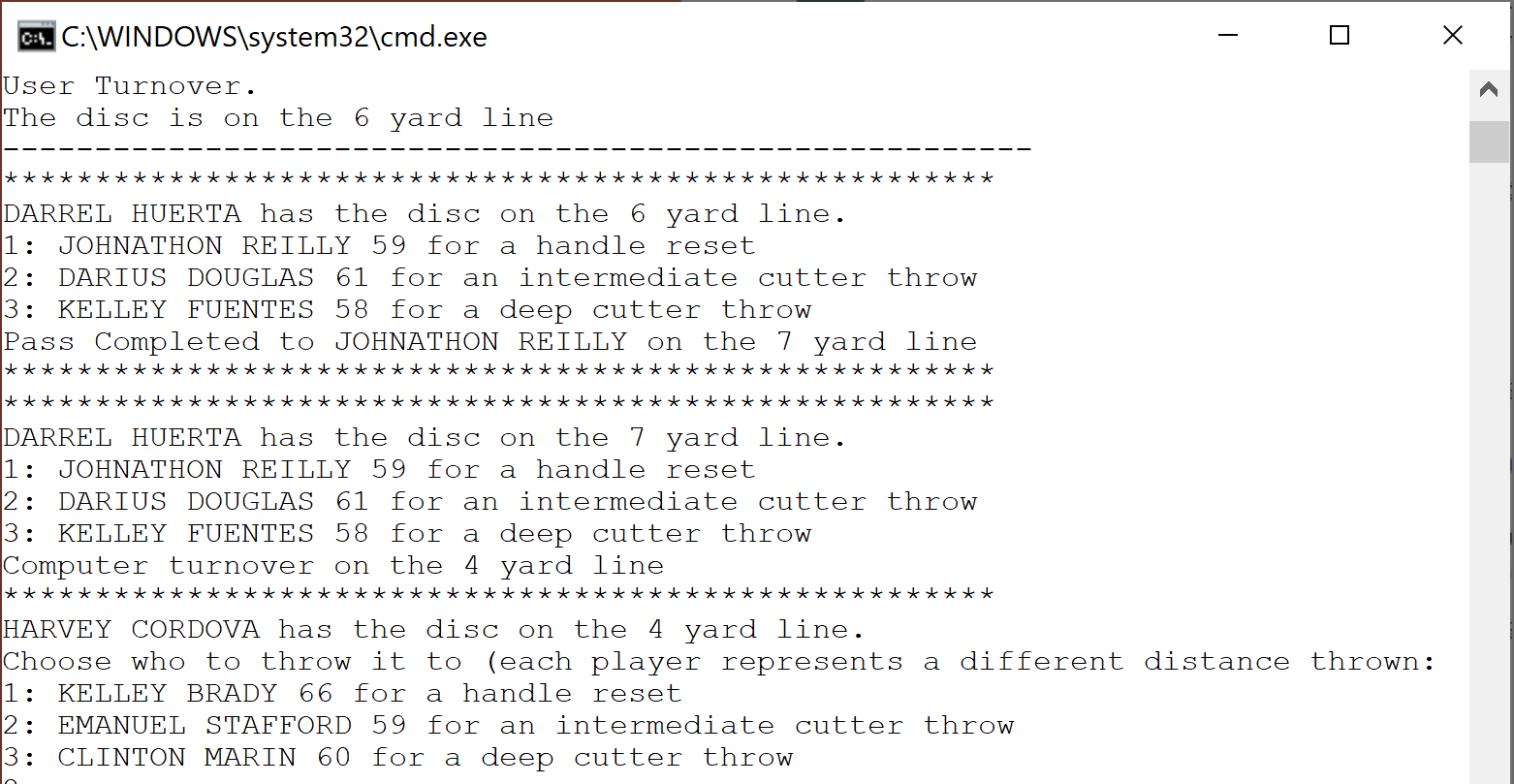


Figure 8: Computer Plays

Next, it switches back to asking the user what they want to do. This time, I enter a 2, hoping to move the disc down the field a little bit, but still have a good probability of a completion. It turns out that this time, the disc is completed to the 11-yard line. Now seems like a great time for a deep throw, so I enter a 3. I get lucky again, and the disc is completed all the way down to the 55-yard line! I now just want a short pass, so I enter a 1, and the disc is completed to the 58-yard line. I then choose a 2, hoping I can score. It is completed, but only to the 68-yard line, so I choose a 1 again. Unfortunately, this results in a turnover, on the 63-yard line This is all shown below in Figure 8.

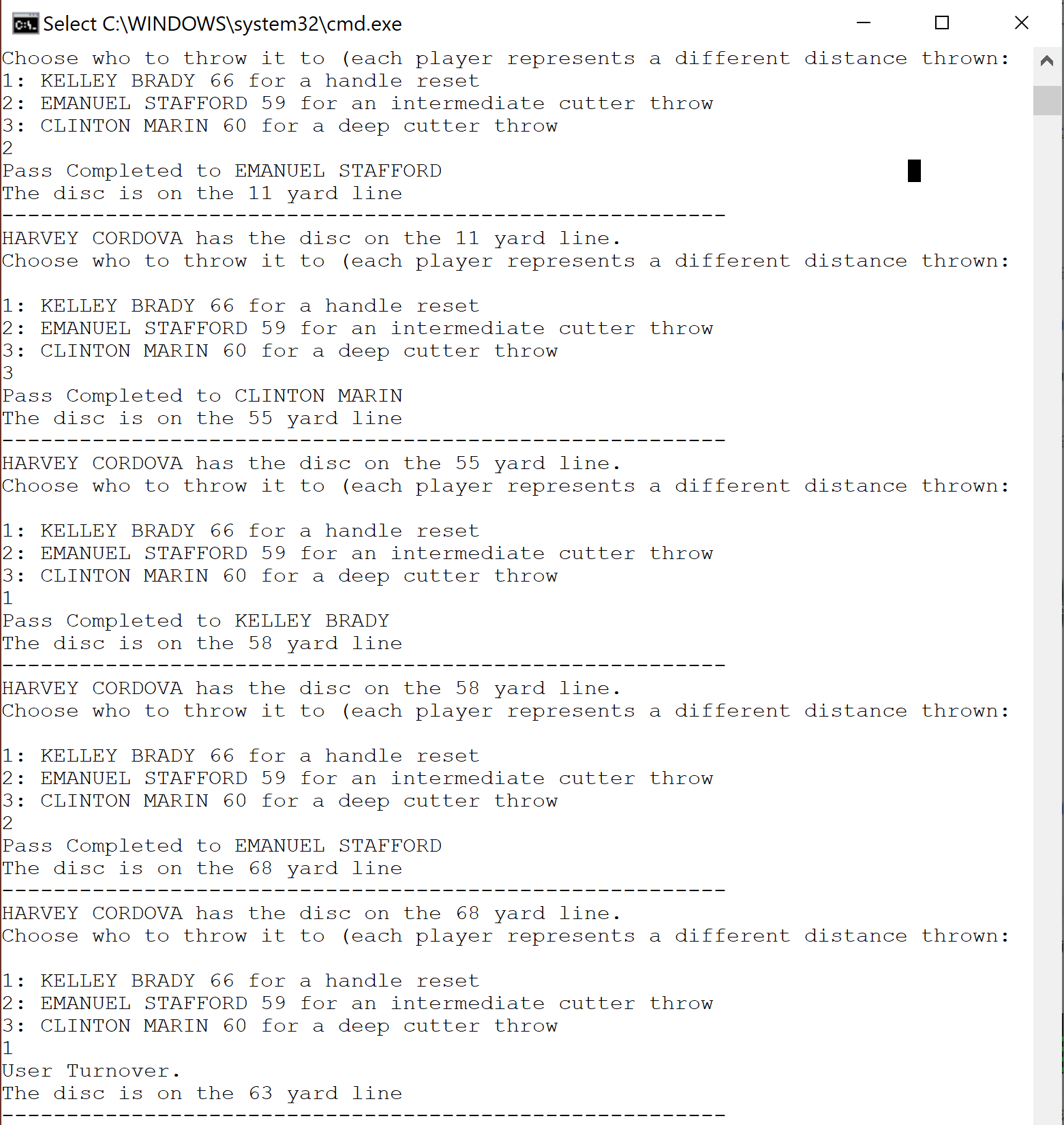


Figure 8: User Plays

Next, the computer completes 15 throws in a row, and then turns it over on the 8-yard line. This is one issue that I need to alter in future programs. Most teams do not just throw handle resets the whole game. I need to make the computer a bit smarter, so that they throw more medium to deep throws. However, that will come in later implementations. I also need to make it smarter, so that when it is close to the end zone, it doesn’t throw a deep throw, because that doesn’t happen in real life either, or else it would be a turnover. Perhaps I could alter the deep throw so that the farthest it could go is 5 yards out of bounds. That will be something I alter in future implementations.

So now the user picks it up on the 8-yard line. I now skip ahead, to the end of the point. During the point, the computer ends up winning the point, as is shown in Figure 9. This now starts a new point, where Team1 is receiving the pull again, because they were just scored upon.

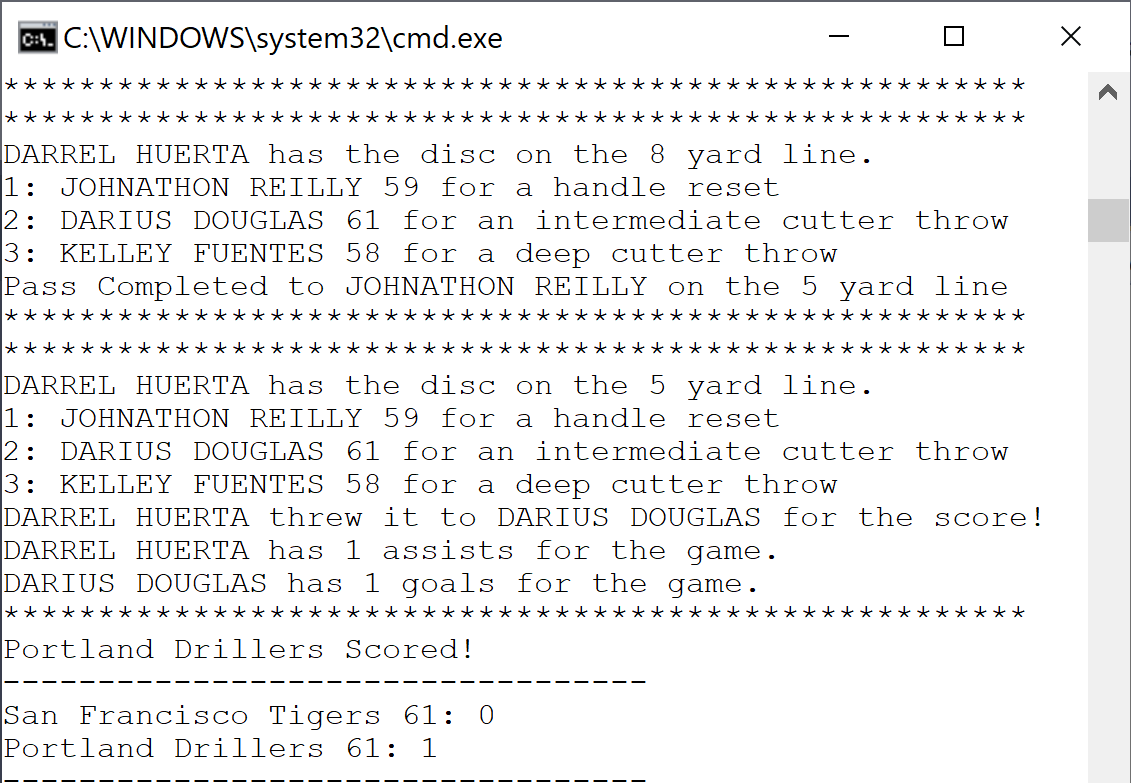
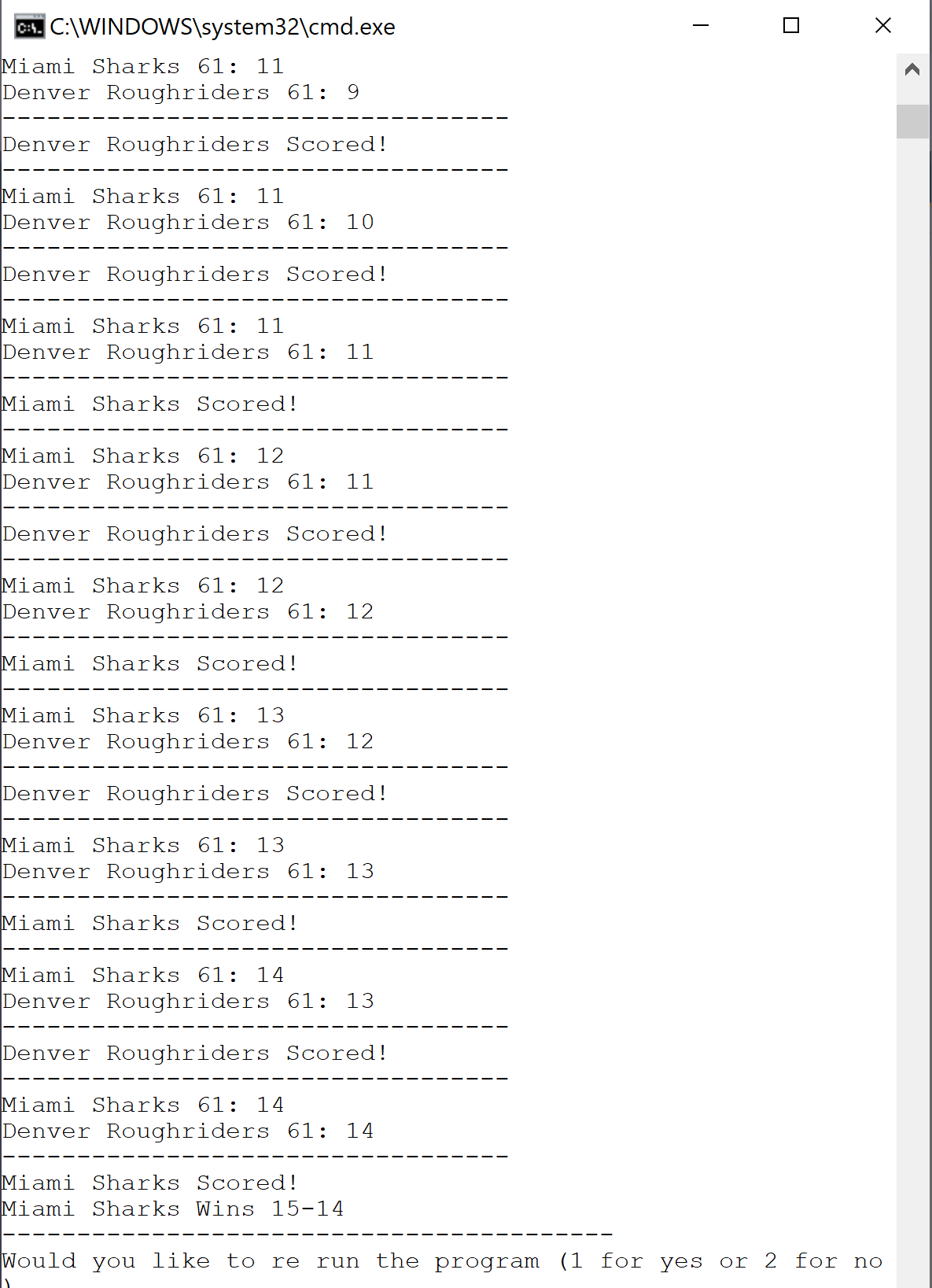


Figure 9: End of First Point

This continues like this until the end of the game, when one team reaches 15 points. The end of the game will be easier to show in the Simulate a full game method, because the play game method takes a long time to play. However, it all works great, and is a lot of fun!

After the play game method, the program asks the user if they want to run the program again. If the user inputs a 1, then the MainMenu is printed again. If any other number is entered, then the program ends. This time the user wants to simulate a full game, so they enter a two when asked what game type they want to play. When this is chosen, the computer immediately simulates a game, printing the scoreboard after each point, until the game is ended when a team reaches 15 points. This is shown below in Figures 9, 10, and 11.

Figures 10,11,12: Simulate Game Results

As can be seen, a full game is simulated, and ends when one team reaches 15.

The program asks to be re-run, and so I enter another 1. This time, I then choose the 3rd option, which was to play the full season. For time purposes, I will not demonstrate this, because that would take forever. Honestly, I don’t know why I have that, besides it being I wrote the code for it. It takes much too long time to play at once, because even one game takes forever to play manually. I will alter this in the future. I also want to learn how to save this game, so that I don’t lose my progress, or team stats when I exit the program. I don’t know how to save the game in the way that video games save, but I want to learn, so that I can save my games. This would mean that play season would be feasible, because you could save between games, and continue with it. Figure 12 demonstrates that entering a 3 does indeed lead you to play a full season.

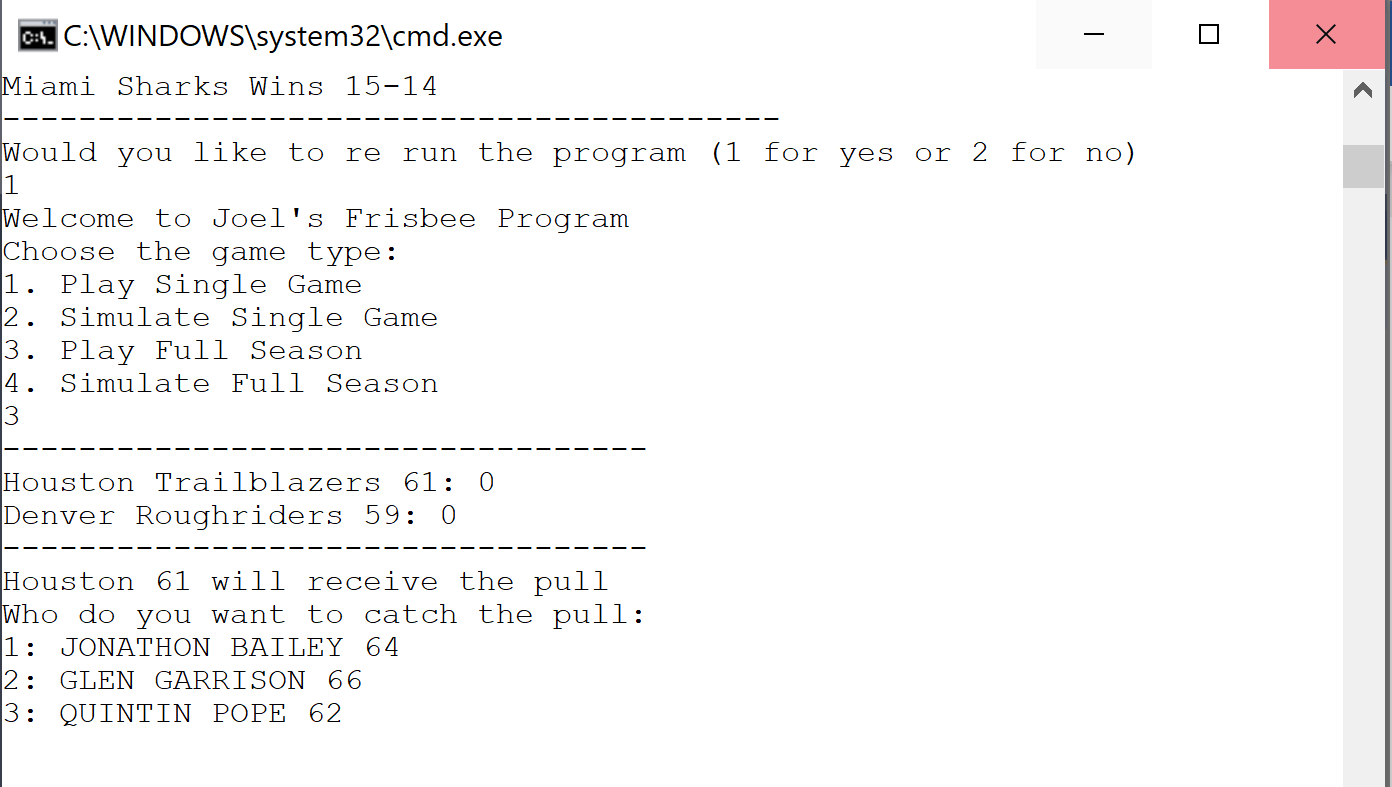


Figure 13: Manual Play Season

The last method to choose is to simulate a full season. This is my most satisfying gameplay, and it goes really fast. When I enter a 4, the simulate full season method is called. When this happens, scores of games just start rolling down the screen, as every single game in the season is being simulated point by point. Obviously, I won’t show this here, because that would take up 20 pages, but that is what happens. In the code, I have it pause for five seconds when the season is complete, so that it can print the end of the season standings. Then it continues into the tournament, where it simulates all of the games, and then prints out the tournament winner,

The end of season standings is shown below in Figure 13. Notice how the standings are sorted, first by team wins, and then by total point differential. There are interesting numbers that come out here. Because sometimes teams with 3 wins have better point differentials than teams with 7 wins. This is something that can be adjusted as time goes on with the program, but I don’t know how to improve that as of right now. The winner of the entire tournament is shown below in Figure 14.

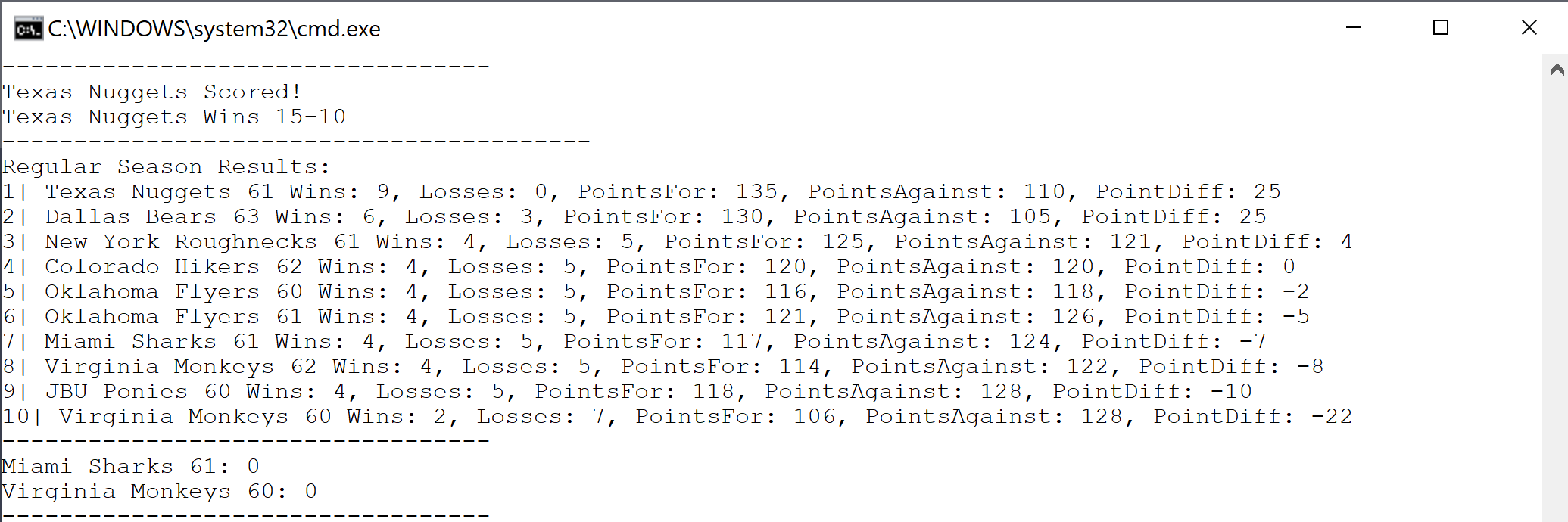


Figure 14: End of Regular Season Results

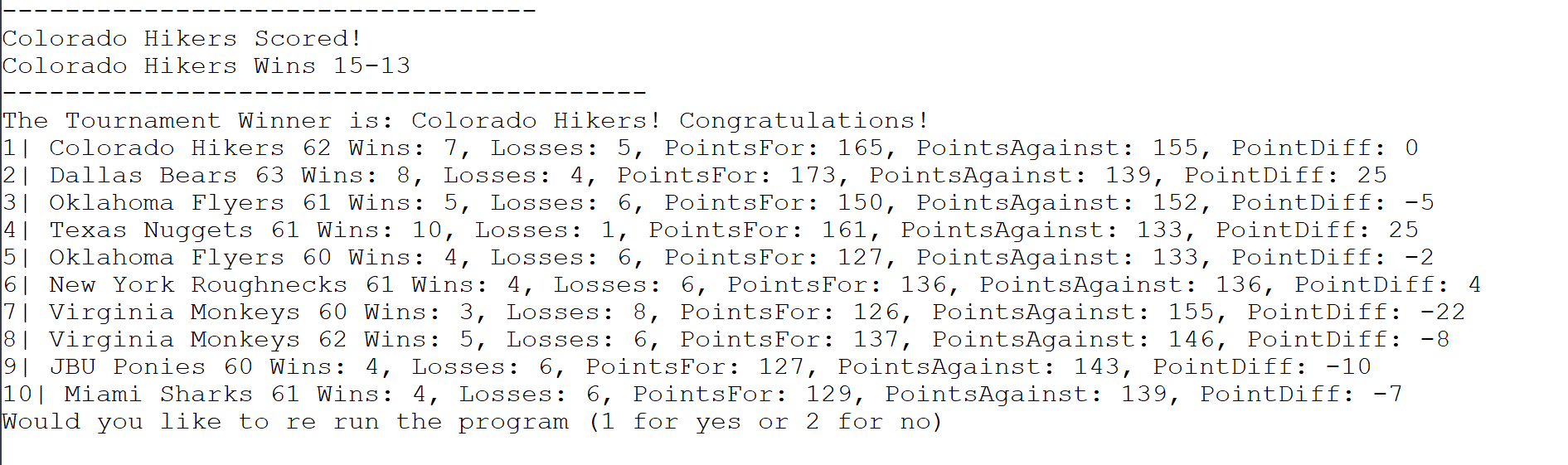


Figure 15: End of Season Tournament Results

It is evident that the same team who dominated the regular season only got 4th place. This is normal, but I don’t know how to improve the numbers behind the program, so that normal stuff happens more normally. It will require research, and more playing around with numbers.

The last step is to run the unit test that I created. I did this and found that the random number generator does indeed work correctly. This is shown below in Figure 15.

A screenshot of a cell phone

Description automatically generated

Figure 16: Unit Testing Results

Now, I have tested everything, and don’t want to play the game anymore, so I enter a 2 when prompted, and the program ends. As is evident, my program works. There are a few places that need to be much more robust and could be changed to make it more user friendly, and fun. However, the basic infrastructure is there for the whole game, and many more features can be added.

**Conclusions:**

To conclude, I successfully create an ultimate frisbee game. I wanted to create a game that simulated an ultimate frisbee game. I did this, allowing the user to choose from 4 different game options. They are allowed to simulate either a game, or a season, or they can play a game, or play a full season. The program has some areas that are vulnerable, but those are rare.

The cool thing about this program, is that it can be improved upon mightily. As you noticed as you read this report, there are many new features and implementations that I hope to add to the project. I want to make it more advanced in the game play, so that it is smarter, and plays like a real video game. I want to add a GUI to my project eventually, and then turn the program into an app that users could download onto their cell phones and tablets. I also someday hope to add graphics to the game, perhaps using Unity, or other software to create Graphics so that it can be more than text based. This would be really hard, but if I decide I want to do game development, then that would be awesome to work on.

I loved working on this project. I really love computer programming, especially in C#. It was such a unique experience to create my own idea, and then create it out of scratch. I did not use code from the internet, everything written is my own code. This project was a lot of work, I probably put over 60 hours of work into the project, but it was worth it. I loved working on it, and I am really proud of what I have created. I know that I still have a far way to go before I would consider this game finished, but that is because I have a high standard for my work. I am excited to work on this project in my free time, and to show it to potential companies that I can work for.

Eventually, I want to use this game to teach people how to play frisbee, because not many people know how to play. If kids could start playing my game, and then they go play it themselves because of the game, that would be incredible. I doubt this would ever happen, but who knows? I have looked, and there aren’t any ultimate frisbee games on the app store right now to my knowledge, maybe this is the first.

For me, the hardest part of the programming was debugging. I would write code, and then test, but since I had so much codes, it took forever to find the errors, and to understand why it was not working correctly. It was good practice in how to use Visual Studio effectively, as well as how to search for those errors. I also had a lot of trouble with a GUI. I need to teach myself how to create a GUI, but with a simple project, before moving on to tougher projects. I started trying to use a GUI too late in my project, and it would have just taken to much time to incorporate.

Next time, I would come up with a list of items that I wanted to incorporate, and then focus on those. I definitely got distracted by methods that I didn’t even end up using in my program because it would have been to much work to incorporate it into my program. If I had not wasted that time, I could have included more features into my program. This was a good experience to have now. I need to keep a clear vision of where I am going, and how I want to reach that destination, otherwise, I will get myself lost.

For next year, I would say keep it mostly the same. It was really important to let us choose our final projects, and it allowed me to choose something that I enjoyed doing, which was a game changer for me. I would say, in terms of requirements, make it more flexible. I know a lot of people had trouble trying to figure out where they would incorporate the testing into their final project. I know it is valuable, and I am glad I learned how to do it, but maybe talk about it more in class, right before the final project, or make a homework assignment about it. It just seemed like an unnecessary timewaster for my project, because I was doing so many different things. Other than that, I loved the project, and the class as a whole.

Appendix:

**Player Class:**

using System;

using System.Collections.Generic;

using System.IO;

namespace Frisbeev01

{

public class Player

{

// Player Traits

public string FirstName { get; set; }

public string LastName { get; set; }

public string JerseyNumber { get; set; }

public int Overall { get; set; }

private int Height { get; set; } // In inches

private int Weight { get; set; } // In pounds

private string filename2 = "Names.csv";

public List<string> firstNameDatabase = new List<string>();

public List<string> lastNameDatabase = new List<string>();

public bool IsHandle { get; set; } // If is true, they are a handler

public bool HasDisc { get; set; } // If true, they have the disc

private int Speed { get; set; }

private int Jumping { get; set; }

private int FlickDistance { get; set; }

private int FlickAccuracy { get; set; }

private int BackhandAccuracy { get; set; }

private int BackhandDistance { get; set; }

private int CutterDefense { get; set; }

private int HandlerDefense { get; set; }

private int Agility { get; set; }

private int HandleCuts { get; set; } // Not sure how to use it

private int UnderCuts { get; set; } // Under and deep cuts determine cut ability

private int DeepCuts { get; set; }

public int ThrowAbility { get; set; }

public int GameGoals { get; set; }

public int SeasonGoals { get; set; }

public int GameAssists { get; set; }

public int SeasonAssists { get; set; }

public int CutAbility { get; set; }

public string TeamNameAndMascot { get; set; }

public Player()

{ // Empty Constructor

;

}

public Player(string fn, string ln, string jn, int s, int j, int fd, int fa, int ba, int bd, int cd, int

hd, int ag, int hc, int uc, int dc)

{ // Constructor

FirstName = fn;

LastName = ln;

JerseyNumber = jn;

Speed = s;

Jumping = j;

FlickDistance = fd;

FlickAccuracy = fa;

BackhandAccuracy = ba;

BackhandDistance = bd;

CutterDefense = cd;

HandlerDefense = hd;

Agility = ag;

HandleCuts = hc;

UnderCuts = uc;

DeepCuts = dc;

ThrowAbility = (fd + fa + bd + ba) / 4;

CutAbility = (uc + dc) / 2;

Overall = (s + j + fd + fa + ba + bd + cd + hd + ag + hc + uc + dc) / 12;

}

public void printAbout(Player player)

{

Console.Write($"{FirstName} {LastName}, {JerseyNumber} ");

}

public void printAllAttributes()

{

Console.WriteLine("-------------------------------");

Console.Write($"First Name: {FirstName} \nLast Name: {LastName} \nJersey Number: {JerseyNumber}

\nOverall: {Overall} \n");

Console.Write($"Speed: {Speed} \nJumping: {Jumping} \nFlick Distance: {FlickDistance} \nFlick

Accuracy: {FlickAccuracy}\n");

Console.Write($"Backhand Accuracy: {BackhandAccuracy} \nBackhand Distance: {BackhandDistance}

\nCutter Defense: {CutterDefense} \nHandler Defense: {HandlerDefense}\n");

Console.Write($"Agility: {Agility} \nHandle Cuts: {HandleCuts} \nUnder Cuts: {UnderCuts} \nDeep

Cuts: {DeepCuts}\n");

}

public void printBasicInfo()

{

Console.WriteLine($"{FirstName} {LastName} {JerseyNumber} {Overall} ");

}

public List<string> ReadInFirstNames()

{

using (StreamReader reader = new StreamReader(filename2))

{

while (!reader.EndOfStream)

{

// Read in individual lines, and then separate by commas

string line = reader.ReadLine();

string[] fields = line.Split(',');

// Add these fields to a job object, which is added to list

firstNameDatabase.Add(fields[0]);

}

return firstNameDatabase;

}

}

public List<string> ReadInLastNames()

{

using (StreamReader reader = new StreamReader(filename2))

{

while (!reader.EndOfStream)

{

// Read in individual lines, and then separate by commas

string line = reader.ReadLine();

string[] fields = line.Split(',');

// Add these fields to a job object, which is added to list

lastNameDatabase.Add(fields[1]);

}

return lastNameDatabase;

}

}

}

}

**Team Class:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace Frisbeev01

{

public class Team

{

public string Name { get; set; }

public string Mascot { get; set; }

public List<Player> TeamOfPlayers = new List<Player>();

private string filename = "TeamList.csv";

public List<string> teamDatabase = new List<string>();

public List<string> mascotDatabase = new List<string>();

public List<string> firstNameDatabase = new List<string>();

public List<string> lastNameDatabase = new List<string>();

public List<Player> OLineList = new List<Player>();

public List<Player> DLineList = new List<Player>();

public List<Player> SortedOLine = new List<Player>();

public List<Player> SortedDLine = new List<Player>();

public List<Player> OLineHandles = new List<Player>();

public List<Player> OLineCutters = new List<Player>();

public List<Player> DLineHandles = new List<Player>();

public List<Player> DLineCutters = new List<Player>();

public int Overall { get; set; } // How good a team is (0-100)

public int OLineOverall { get; set; }

public int DLineOverall { get; set; }

public bool startPointWithDisc { get; set; } // Determines who starts each point on offense

public int coinFlipChoice { get; set; } // Either 1 or 2. 1 is heads, 2 is tails

public bool WonPreviousPoint { get; set; }

public int Wins { get; set; }

public int Losses { get; set; }

public int TotalPoints { get; set; }

public int TotalPointsAgainst { get; set; }

public int TotalPointDifferential { get; set; }

// Constructors

public Team()

{

;

}

public Team(string n, string m)

{

Name = n;

Mascot = m;

TotalPoints = 0;

TotalPointsAgainst = 0;

}

public void CompCreateLines()

{ // Computer creates an O line and D line. There are 7 on each line, 3 handles and four cutters.

// Each team has fourteen people

int countHandlesO = 0; // When there are three handles on O line, then i will put handles on d line

int countCuttersO = 0; // When there are 4 cutters on O line, then I will put cutters on d line

for (int i = 0; i < 14; i++)

{ // Adds 3 handles to OLine

Player player = TeamOfPlayers[i];

if (player.IsHandle == true)

{

if (countHandlesO == 3)

{

DLineList.Add(player);

DLineHandles.Add(player);

}

else

{ // Adds handles 3 times

OLineList.Add(player);

OLineHandles.Add(player);

countHandlesO++;

}

}

else

{

if (countCuttersO == 4)

{

DLineList.Add(player);

DLineCutters.Add(player);

}

else

{ // Adds cutters 4 times

OLineList.Add(player);

OLineCutters.Add(player);

countCuttersO++;

}

}

}

// SortedStandings = Standings.OrderBy(Team => Team.Wins).ThenBy(Team =>

Team.TotalPointDifferential).ToList();

//SortedOLine.OrderBy(Player => Player.IsHandle).ThenBy(Player => Player.ThrowAbility);

CalculateLinesOverall();

}

public void AddPlayer(Player player)

{ // Adds a player to a team

TeamOfPlayers.Add(player);

}

public List<string> ReadInTeams()

{ // This reads in the list of teams from a .csv file

using (StreamReader reader = new StreamReader(filename))

{

while (!reader.EndOfStream)

{

// Read in individual lines, and then separate by commas

string line = reader.ReadLine();

string[] fields = line.Split(',');

// Add these fields to a job object, which is added to list

teamDatabase.Add(fields[0]);

}

return teamDatabase;

}

}

public List<string> ReadInMascots()

{ // This reads in a list of mascots from a .csv file

using (StreamReader reader = new StreamReader(filename))

{

while (!reader.EndOfStream)

{

// Read in individual lines, and then separate by commas

string line = reader.ReadLine();

string[] fields = line.Split(',');

// Add these fields to a job object, which is added to list

mascotDatabase.Add(fields[1]);

}

return mascotDatabase;

}

}

public void printTeam()

{

Console.WriteLine($"{Name} {Mascot} {Overall}");

}

public void FillTeamWithRandomPlayers()

{ // Creates 14 random players, 6 of which are handles and 8 of which are cutters

// It then puts them into a team.

Create create = new Create();

Player player = new Player();

Season season = new Season();

for (int i = 0; i < 6; i++)

{ // Add 6 handles

AddPlayer(player = create.CreateRandomHandle());

}

for (int i = 0; i < 8; i++)

{

AddPlayer(player = create.CreateRandomCutter());

}

CalculateTeamOverall(); // Gives the team an overall rating based on each player

}

public List<Player> ChooseLine(bool isOLine)

{ // If on offense, then OLine is sent out, if on defense, DLine is sent out

List<Player> sevenOnField = new List<Player>();

if (isOLine == true)

{

sevenOnField = OLineList;

}

else

{

sevenOnField = DLineList;

}

return sevenOnField;

}

public void CalculateTeamOverall()

{

int sum = 0;

int count = 0;

foreach (Player player in TeamOfPlayers)

{

sum = sum + player.Overall;

count = count + 1;

}

Overall = (sum / count);

}

public void CalculateLinesOverall()

{

int sum = 0;

int count = 0;

foreach (Player player in OLineList)

{

sum = sum + player.Overall;

count = count + 1;

}

OLineOverall = (sum / count);

sum = 0;

count = 0;

foreach (Player player in DLineList)

{

sum += player.Overall;

count += 1;

}

DLineOverall = (sum / count);

}

public void CalculateTeamPointDifferential()

{

TotalPointDifferential = TotalPoints - TotalPointsAgainst;

}

}

}

**Game Class:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace Frisbeev01

{

public class Game

{

Random random = new Random();

private int ScoreTeam1 { get; set; }

private int ScoreTeam2 { get; set; }

private int Windspeed { get; set; } // Randomly chooses from array of windspeeds during game before each

point

private Team Team0 { get; set; } // This will be used as a team for between points, so that I can

exit while loops

private Team Team1 { get; set; }

private Team Team2 { get; set; }

private bool Team1TurnedOver { get; set; }

private bool Team2TurnedOver { get; set; }

private int BetterTeam { get; set; } // 1 means team1 is better, 2 means team2 is better

private int BetterLine { get; set; }

private int DifferenceInTeamOverall { get; set; }

private int DifferenceInLineOverall { get; set; }

private Team TeamWithDisc { get; set; }

private int Probability { get; set; }

public Team Loser { get; set; }

private int DistanceThrown { get; set; } // 1 means short throw, 2 means medium, 3 means long

private List<Player> OLineOnField = new List<Player>();

private List<Player> DLineOnField = new List<Player>();

public Team Winner = new Team();

public Team OLineTeam1 = new Team();

public Team OLineTeam2 = new Team();

public Team DLineTeam1 = new Team();

public Team DLineTeam2 = new Team();

public Team sevenOnTeam1 = new Team(); // These will be the seven players on the field for each point

public Team sevenOnTeam2 = new Team();

Print print = new Print(); // This class is for printing out info like rosters

//These next properties are for the manual play point

public int yardsGainedDownfield { get; set; } // 20 yard endzones count as -20 -> 0. 70 yard field

counts as 0->70. last endzone is 70->90

public int discLocationY { get; set; }

public int playerWithDisc { get; set; }

public int gameType { get; set; } // If 1, it is play game, 2 is simulate game

Player p1t1 = new Player();

Player p2t1 = new Player();

Player p3t1 = new Player();

Player p4t1 = new Player();

Player p5t1 = new Player();

Player p6t1 = new Player();

Player p7t1 = new Player();

Player p8t1 = new Player();

Player p9t1 = new Player();

Player p10t1 = new Player();

Player p11t1 = new Player();

Player p12t1 = new Player();

Player p13t1 = new Player();

Player p14t1 = new Player();

// Team 2

Player p1t2 = new Player();

Player p2t2 = new Player();

Player p3t2 = new Player();

Player p4t2 = new Player();

Player p5t2 = new Player();

Player p6t2 = new Player();

Player p7t2 = new Player();

Player p8t2 = new Player();

Player p9t2 = new Player();

Player p10t2 = new Player();

Player p11t2 = new Player();

Player p12t2 = new Player();

Player p13t2 = new Player();

Player p14t2 = new Player();

public Player Thrower { get; set; }

public Player Catcher { get; set; }

//

public Game(Team t1, Team t2, int type)

{

Create create = new Create();

gameType = type;

Team1 = t1;

Team2 = t2;

ScoreTeam1 = 0;

ScoreTeam2 = 0;

DifferenceInTeamOverall = CalculateDifferenceInTeamOverall(); // Changes property to difference in

team overall

OLineTeam1 = create.CreateLineFromList(t1.OLineList, true);

OLineTeam2 = create.CreateLineFromList(t2.OLineList, true);

DLineTeam1 = create.CreateLineFromList(t1.DLineList, false);

DLineTeam2 = create.CreateLineFromList(t2.DLineList, false);

}

// The game Progression goes like this.

// Game.SimulateFullgame is called by other method

// StartGame happens, then coinflip. This determines who has the disc to start

// Next, PlayPoint is called in a while loop while the game isn't over.

// Playpoint calls printscoreboard, and then it sets the lines according to who starts with the disc. (If Team1 starts with disc, Team 1's Oline starts

// If the program is simulating, then SimulatePointWinner is called. This method uses probability to

determine who won the point.

// If the program is not a simulation, then PlayPointWinner is called. This method allows the user to

play the point.

// After the point is played, DetermineNextPointsInfo is called. This method takes the winner of the

point, switches who starts with the disc, and then determines if game is over or not

// This ends the game while loop, and then each teams info is updated in terms of score and wins and

such.

public void StartGame()

{ // Gives intro, prints roster, calculates team's overall difference, finds windspeed, then starts

game

// It also creates two team objects of seven on o line and seven on dline

p1t1 = OLineTeam1.OLineHandles[0];

p2t1 = OLineTeam1.OLineHandles[1];

p3t1 = OLineTeam1.OLineHandles[2];

p4t1 = OLineTeam1.OLineCutters[0];

p5t1 = OLineTeam1.OLineCutters[1];

p6t1 = OLineTeam1.OLineCutters[2];

p7t1 = DLineTeam1.DLineCutters[3];

p1t2 = DLineTeam1.DLineHandles[0];

p2t2 = DLineTeam1.DLineHandles[1];

p3t2 = DLineTeam1.DLineHandles[2];

p4t2 = DLineTeam1.DLineCutters[0];

p5t2 = DLineTeam1.DLineCutters[1];

p6t2 = DLineTeam1.DLineCutters[2];

p7t2 = DLineTeam1.DLineCutters[3];

if (FullProgram.Verbosity == 2)

{

Console.WriteLine($"Welcome to Tonight's Matchup Between {Team1.Name} {Team1.Mascot} and

{Team2.Name} {Team2.Mascot}");

Console.WriteLine($"We hope the matchup will be close as {Team1.Name} has an overall:

{Team1.Overall} while {Team2.Name} has an overall: {Team2.Overall}");

}

if (FullProgram.Verbosity == 3)

{

print.PrintRoster(Team1);

print.PrintRoster(Team2);

}

CoinFlip();

DifferenceInTeamOverall = CalculateDifferenceInTeamOverall();

if (FullProgram.Verbosity == 3)

Console.WriteLine($"The better team is {BetterTeam}");

Probability = CalculateProbabilityForWinner();

ScoreTeam1 = 0;

ScoreTeam2 = 0;

}

public void CoinFlip()

{ // Determines who starts the game with the disc

Team1.coinFlipChoice = 1;

Team2.coinFlipChoice = 2;

int coinFlip = random.Next(1, 2);

if (coinFlip == 1)

{

Team1.startPointWithDisc = true;

Team2.startPointWithDisc = false;

if (FullProgram.Verbosity == 2)

Console.WriteLine($"Team 1 won the disc flip");

}

else

{

Team1.startPointWithDisc = false;

Team2.startPointWithDisc = true;

if (FullProgram.Verbosity == 2)

Console.WriteLine($"Team 2 won the disc flip");

}

}

public void FullGame()

{ // Takes two teams, and simulates a full game.

// All return values are used as properties and returned there and read later

// Returned as game, so that it can be referenced later

StartGame();

int winner = 0;

while (ScoreTeam1 != 15 && ScoreTeam2 != 15)

{

PlayPoint();

if (ScoreTeam1 == 15)

{

Console.WriteLine($"{Team1.Name} {Team1.Mascot} Wins {ScoreTeam1}-{ScoreTeam2}");

Console.WriteLine("-----------------------------------------");

Team1.TotalPoints += ScoreTeam1;

Team2.TotalPoints += ScoreTeam2;

Team1.TotalPointsAgainst += ScoreTeam2;

Team2.TotalPointsAgainst += ScoreTeam1;

Winner = Team1;

Loser = Team2;

winner = 1;

}

if (ScoreTeam2 == 15)

{

Console.WriteLine($"{Team2.Name} {Team2.Mascot} Wins {ScoreTeam2}-{ScoreTeam1}");

Console.WriteLine("-----------------------------------------");

Team1.TotalPoints += ScoreTeam1;

Team2.TotalPoints += ScoreTeam2;

Team1.TotalPointsAgainst += ScoreTeam2;

Team2.TotalPointsAgainst += ScoreTeam1;

Winner = Team2;

Loser = Team1;

winner = 2;

}

if (winner == 1)

{

Team1.Wins += 1;

Team2.Losses += 1;

break;

}

else if (winner == 2)

{

Team1.Losses += 1;

Team2.Wins += 1;

break;

}

}

}

public bool PlayPoint()

{ // Plays through a full point.

// If GameType is 1, it simulates it and Does calculations and chooses point winner

// If GameType is 2, the user plays it and a winner is chosen

bool gameOver = false;

int winner = 0;

PrintScoreboard();

if (Team1.startPointWithDisc == true)

{

sevenOnTeam1 = OLineTeam1;

sevenOnTeam2 = DLineTeam2;

discLocationY = 0;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

Console.WriteLine($"{Team1.Name} {Team1.Mascot} Has the Disc to Start ");

}

else if (Team2.startPointWithDisc == true)

{

sevenOnTeam1 = DLineTeam1;

sevenOnTeam2 = OLineTeam2;

discLocationY = 70;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

Console.WriteLine($"{Team2.Name} {Team2.Mascot} Has the Disc to Start ");

}

if (FullProgram.Verbosity == 3)

Console.WriteLine("Choosing Point winner in play point");

if (gameType == 1)

{

winner = PlayPointWinner();

}

else

{

winner = SimulatePointWinner();

}

gameOver = DetermineNextPointsInfo(winner);

return gameOver;

}

public int SimulatePointWinner()

{ // Simulates who wins the point based on probability and team overall

int determiner = random.Next(1, 100); // Will help determine winner

int winner = 0;

if (Team1.startPointWithDisc == true)

{ // + 10 probability for starting with disc

if (FullProgram.Verbosity == 3)

Console.WriteLine($"Determiner: {determiner} and Probability: {Probability} ");

if (BetterTeam == 1)

{ // If betterTeam

if (determiner <= (Probability + 10))

{ // Team 1 has better chance to hold as better team

winner = 1;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team1.Name} wins: {Probability + 10} while

receiving pull as better team");

Console.WriteLine("winner = 1 by holding as better team");

}

}

else

{ // Team 2 breaks and wins the point as worse team

winner = 2;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team2.Name} wins: {Probability - 10} by

breaking as worse team");

Console.WriteLine("winner = 2 by by breaking as worse team");

}

}

}

else if (BetterTeam == 2)

{ // If BetterTeam = 2

if (determiner <= (100 - Probability) + 10)

{ // Team 1 holds and wins the point as worse team

winner = 1;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team1.Name} wins: {100 - Probability + 10}

while receiving pull as worse team");

Console.WriteLine("winner = 1 by holding as worse team");

}

}

else

{ // Team 2 breaks as the better team

winner = 2;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team2.Name} wins: {100 - Probability + 10} by

breaking as better team");

Console.WriteLine("winner = 2 by breaking as better team");

}

}

}

else

{ // Teams are equal

if (determiner <= (Probability + 10))

{ // Team 1 has better chance to hold as equal team

winner = 1;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team1.Name} wins: {Probability + 10} while

receiving pull as equal team");

Console.WriteLine("winner = 2 by beating the odds and breaking as an equal team");

}

}

else

{ // Team 2 breaks and wins the point as equal team

winner = 2;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team2.Name} wins: {Probability - 10} while

breaking as equal team");

Console.WriteLine("winner = 2 by beating the odds and breaking as an equal team");

}

}

}

}

else if (Team2.startPointWithDisc == true)

{ // If team 2 starts with disc

if (FullProgram.Verbosity == 3)

Console.WriteLine($"Determiner: {determiner} and Probability: {Probability} ");

if (BetterTeam == 2)

{

if (determiner <= (Probability + 10)) // Gives Team2 better chance of holding

{ // Team 2 holds and wins point as better team

winner = 2;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team2.Name} wins: {Probability + 10} while

receiving pull as better team");

Console.WriteLine($"winner = {Team2.Name} by holding as better team");

}

}

else

{ // Else Team 1 breaks and wins the point as worse team

winner = 1;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team1.Name} wins: {Probability - 10} while

breaking pull as worse team");

Console.WriteLine($"winner = {Team1.Name} by breaking as worse team");

}

}

}

else if (BetterTeam == 1)

{ // Team 2 is not Better team

if (determiner <= 100 - Probability + 10)

{ // Team 2 holds and wins the point as worse team

winner = 2;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team2.Name} wins: {100 - Probability + 10}

while receiving pull as worse team");

Console.WriteLine("winner = 2 by holding as the worse team");

}

}

else

{ // Team 1 breaks and wins the point as better team

winner = 1;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team1.Name} wins: {100 - Probability + 10} by

breaking as equal team");

Console.WriteLine("winner = 1 by breaking as an equal team");

}

}

}

else

{ // Teams are equal

if (determiner <= (Probability + 10)) // Gives Team2 better chance of holding

{ // Team 2 holds and wins point as equal team

winner = 2;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team2.Name} wins: {Probability + 10} while

receiving pull as equal team");

Console.WriteLine("winner = 2 by holding as equal team");

}

}

else

{ // Else Team 1 breaks and wins the point as equal team

winner = 1;

if (FullProgram.Verbosity == 2 || FullProgram.Verbosity == 3)

{

Console.WriteLine($"Probability that {Team1.Name} wins: {Probability - 10} while

breaking as equal team");

Console.WriteLine("winner = 1 by holding as equal team");

}

}

}

}

return winner;

}

public int PlayPointWinner()

{ // This is in contrast to Simulate Point winner

// This method allows the user to play the point.

// This will be text based, giving the user a couple options of where to throw.

// Probability will be used to determine whether it is a score, and then uses probability

// This assumes the user is team1

int winner = 1; // Placeholder until I'm finished with it

bool pointOver = false; // This turns true when someone scores

// Initializes 14 players

// p stands for player t stands for team the number stands for which player and which team

// This is the Lineup

/\*

\* 1 2 3

\* 4 5 6 7

\*/

// Team 1

int userChoice = 0;

yardsGainedDownfield = 0;

/\*

Include a try catch exception here later

\*/

int numCatcher = 0;

if (Team1.startPointWithDisc == true)

{

TeamWithDisc = Team1;

discLocationY = 0;

Console.WriteLine($"{Team1.Name} {Team1.Overall} will receive the pull");

bool isCorrect = false;

while (isCorrect == false)

{

Console.WriteLine("Who do you want to catch the pull: ");

Console.WriteLine($"1: {p1t1.FirstName} {p1t1.LastName} {p1t1.Overall}");

Console.WriteLine($"2: {p2t1.FirstName} {p2t1.LastName} {p2t1.Overall}");

Console.WriteLine($"3: {p3t1.FirstName} {p3t1.LastName} {p3t1.Overall}");

if (!int.TryParse(Console.ReadLine(), out userChoice))

{

Console.WriteLine("Please enter either a 1, 2 or a 3");

isCorrect = false;

}

else

{

isCorrect = true;

}

}

switch (userChoice)

{

case 1:

p1t1.HasDisc = true;

Catcher = p1t1;

numCatcher = 1;

break;

case 2:

p2t1.HasDisc = true;

Catcher = p2t1;

numCatcher = 2;

break;

case 3:

p3t1.HasDisc = true;

Catcher = p3t1;

numCatcher = 3;

break;

default:

p1t1.HasDisc = true;

Catcher = p1t1;

numCatcher = 1;

break;

}

discLocationY = 0;

}

else

{

Console.WriteLine($"{Team2.Name} {Team2.Overall} will receive the pull");

discLocationY = 70;

TeamWithDisc = Team2;

switch (userChoice)

{

case 1:

p1t2.HasDisc = true;

Catcher = p1t2;

numCatcher = 1;

break;

case 2:

p2t2.HasDisc = true;

Catcher = p2t2;

numCatcher = 2;

break;

case 3:

p3t2.HasDisc = true;

Catcher = p3t2;

numCatcher = 3;

break;

default:

p1t2.HasDisc = true;

Catcher = p1t2;

numCatcher = 1;

break;

}

}

Console.WriteLine("--------------------------------------------------------");

Console.WriteLine($"{Catcher.FirstName} {Catcher.LastName} {Catcher.Overall} caught the pull at the

{discLocationY} yard line");

while (pointOver == false)

{

while (TeamWithDisc == Team1)

{

if (Team2TurnedOver == true)

{ // Earlier, there wouldnt be a catcher, because team2 had caught it. This allows for a

turnover, and the team picks it up

Catcher = p1t1; // Later have it randomly choose who picks it up

numCatcher = 1;

}

Console.WriteLine($"{Catcher.FirstName} {Catcher.LastName} has the disc on the

{discLocationY} yard line.");

Console.WriteLine($"Choose who to throw it to (each player represents a different distance

thrown: ");

Random probability = new Random();

if (Catcher == p1t1)

{

numCatcher = 1;

}

else if (Catcher == p2t1)

{

numCatcher = 2;

}

else if (Catcher == p3t1)

{

numCatcher = 3;

}

else if (Catcher == p4t1)

{

numCatcher = 4;

}

else if (Catcher == p5t1)

{

numCatcher = 5;

}

else if (Catcher == p6t1)

{

numCatcher = 6;

}

else if (Catcher == p7t1)

{

numCatcher = 7;

}

switch (numCatcher)

{ // This makes sure they can't throw it to themselves

// Other options will be 1 handle, and one intermediate cutter, and one deep cutter

case 1:

Catcher = ChoosePlayerToThrowTo(p2t1, p4t1, p5t1);

discLocationY += DetermineYardsGained();

break;

case 2:

Catcher = ChoosePlayerToThrowTo(p3t1, p5t1, p6t1);

discLocationY += DetermineYardsGained();

break;

case 3:

Catcher = ChoosePlayerToThrowTo(p2t1, p6t1, p7t1);

discLocationY += DetermineYardsGained();

break;

case 4:

Catcher = ChoosePlayerToThrowTo(p1t1, p3t1, p5t1);

discLocationY += DetermineYardsGained();

break;

case 5:

Catcher = ChoosePlayerToThrowTo(p2t1, p4t1, p6t1);

discLocationY += DetermineYardsGained();

break;

case 6:

Catcher = ChoosePlayerToThrowTo(p2t1, p5t1, p7t1);

discLocationY += DetermineYardsGained();

break;

case 7:

Catcher = ChoosePlayerToThrowTo(p3t1, p5t1, p6t1);

discLocationY += DetermineYardsGained();

break;

default:

Catcher = ChoosePlayerToThrowTo(p2t1, p4t1, p5t1);

discLocationY += DetermineYardsGained();

break;

}

if (Team1TurnedOver == true)

{

Console.WriteLine($"User turnover on the {discLocationY} yard line");

}

else

{

Console.WriteLine($"The disc is on the {discLocationY} yard line");

if (discLocationY > 90)

{ // Turnover because it is out of bounds

TeamWithDisc = Team2;

Console.WriteLine("Team 1 turned it out of the other endzone!");

}

else if (discLocationY < -20)

{ // If it goes out of other endzone

TeamWithDisc = Team2;

Console.WriteLine("Team 1 turned it out of their own endzone!");

}

else if (70 < discLocationY && discLocationY <= 90 && Team1TurnedOver == false)

{ // PointScored for Team1

Catcher.GameGoals += 1;

Thrower.GameAssists += 1;

Console.WriteLine($"{Thrower.FirstName} {Thrower.LastName} threw it to

{Catcher.FirstName} {Catcher.LastName} for the score!");

Console.WriteLine($"{Thrower.FirstName} {Thrower.LastName} has {Thrower.GameAssists}

assists for the game.");

Console.WriteLine($"{ Catcher.FirstName} {Catcher.LastName} has {Catcher.GameGoals}

goals for the game.");

winner = 1;

pointOver = true;

TeamWithDisc = Team0; // Exits while loop

Team1.startPointWithDisc = false;

Team2.startPointWithDisc = true;

}

}

Console.WriteLine("--------------------------------------------------------");

}

while (TeamWithDisc == Team2)

{

if (Team1TurnedOver == true)

{ // Earlier, there wouldnt be a catcher, because team2 had caught it. This allows for a

turnover, and the team picks it up

Catcher = p1t2; // Later have it randomly choose who picks it up

}

if (discLocationY > 70 )

{ // If the disc is turned over in the endzone, or beyond the endzone, set it on goal line

discLocationY = 70;

}

if (discLocationY < 0)

{ // If disc is turned over in other endzone, it is brought up to goal line

discLocationY = 0;

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine($"{Catcher.FirstName} {Catcher.LastName} has the disc on the

{discLocationY} yard line.");

Random probability = new Random();

//Console.WriteLine($"numCatcher {numCatcher}");

if (Catcher == p1t2)

{

numCatcher = 1;

}

else if (Catcher == p2t2)

{

numCatcher = 2;

}

else if (Catcher == p3t2)

{

numCatcher = 3;

}

else if (Catcher == p4t2)

{

numCatcher = 4;

}

else if (Catcher == p5t2)

{

numCatcher = 5;

}

else if (Catcher == p6t2)

{

numCatcher = 6;

}

else if (Catcher == p7t2)

{

numCatcher = 7;

}

switch (numCatcher) // This only allows three options for user

//switch (Catcher)

{ // This makes sure they can't throw it to themselves

// Other options will be 1 handle, and one intermediate cutter, and one deep cutter

case 1:

Catcher = DeterminePlayerToThrowTo(p2t2, p4t2, p5t2);

discLocationY -= DetermineYardsGained();

break;

case 2:

Catcher = DeterminePlayerToThrowTo(p3t2, p5t2, p6t2);

discLocationY -= DetermineYardsGained();

break;

case 3:

Catcher = DeterminePlayerToThrowTo(p2t2, p6t2, p7t2);

discLocationY -= DetermineYardsGained();

break;

case 4:

Catcher = DeterminePlayerToThrowTo(p1t2, p3t2, p5t2);

discLocationY -= DetermineYardsGained();

break;

case 5:

Catcher = DeterminePlayerToThrowTo(p2t2, p4t2, p6t2);

discLocationY -= DetermineYardsGained();

break;

case 6:

Catcher = DeterminePlayerToThrowTo(p2t2, p5t2, p7t2);

discLocationY -= DetermineYardsGained();

break;

case 7:

Catcher = DeterminePlayerToThrowTo(p3t2, p5t2, p6t2);

discLocationY -= DetermineYardsGained();

break;

default:

Console.WriteLine("Entered default in switch case for determining player to throw");

Catcher = DeterminePlayerToThrowTo(p2t2, p4t2, p5t2);

discLocationY -= DetermineYardsGained();

break;

}

if (Team2TurnedOver == true)

{

Console.WriteLine($"Computer turnover on the {discLocationY} yard line");

}

else

{

if (discLocationY > 90)

{ // Turnover because it is out of bounds

TeamWithDisc = Team1;

discLocationY = 70;

Console.WriteLine("Team 2 turned it out of the other endzone!");

}

else if (discLocationY < -20)

{ // If it goes out of other endzone

TeamWithDisc = Team1;

discLocationY = 0;

Console.WriteLine("Team 2 turned it out of their own endzone!");

}

else if (-20 < discLocationY && discLocationY <= 0 && Team2TurnedOver == false)

{ // PointScored for Team1

Thrower.GameAssists += 1;

Catcher.GameGoals += 1;

Console.WriteLine($"{Thrower.FirstName} {Thrower.LastName} threw it to

{Catcher.FirstName} {Catcher.LastName} for the score!");

Console.WriteLine($"{Thrower.FirstName} {Thrower.LastName} has {Thrower.GameAssists}

assists for the game.");

Console.WriteLine($"{ Catcher.FirstName} {Catcher.LastName} has {Catcher.GameGoals}

goals for the game.");

winner = 2;

pointOver = true;

TeamWithDisc = Team0;

}

else

{

Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName} on the

{discLocationY} yard line");

}

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

if (TeamWithDisc == Team0)

{

return winner;

}

}

return winner;

}

public Player DeterminePlayerToThrowTo(Player p1, Player p2, Player p3)

{ // this simulates the point for the computer, which the user is facing

Console.WriteLine($"1: {p1.FirstName} {p1.LastName} {p1.Overall} for a handle reset");

Console.WriteLine($"2: {p2.FirstName} {p2.LastName} {p2.Overall} for an intermediate cutter throw");

Console.WriteLine($"3: {p3.FirstName} {p3.LastName} {p3.Overall} for a deep cutter throw");

Random probability = new Random();

int potentialCatcher = probability.Next(1, 3);

switch (potentialCatcher)

{ // DistanceThrown is backwards from other ChoosePlayer DistanceThrown, because other team is

going the other direction

case 1:

int probailityOfCatch = probability.Next(0, 100);

if (probailityOfCatch > 10)

{ // Then the catch is complete, and yardage gained will

Thrower = Catcher;

Catcher = p1;

DistanceThrown = -1;

//Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName}");

Team2TurnedOver = false;

}

else

{ // Turnover

Thrower = Catcher;

// Add a turnover to their stats right here

TeamWithDisc = Team1;

DistanceThrown = 1;

//Console.WriteLine($"Computer Turnover on the {discLocationY} yard line.");

Team2TurnedOver = true;

}

break;

case 2:

probailityOfCatch = probability.Next(0, 100);

if (probailityOfCatch > 25)

{ // Then the catch is complete, and yardage gained will

Thrower = Catcher;

Catcher = p2;

DistanceThrown = -2;

//Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName}");

Team2TurnedOver = false;

}

else

{ // Turnover

Thrower = Catcher;

// Add a turnover to their stats right here

TeamWithDisc = Team1;

DistanceThrown = 2;

//Console.WriteLine($"Computer Turnover on the {discLocationY} yard line.");

Team2TurnedOver = true;

}

break;

case 3:

probailityOfCatch = probability.Next(0, 100);

if (probailityOfCatch > 45)

{ // Then the catch is complete, and yardage gained will

Thrower = Catcher;

Catcher = p3;

DistanceThrown = -3;

//Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName}");

Team2TurnedOver = false;

}

else

{ // Turnover

Thrower = Catcher;

// Add a turnover to their stats right here

TeamWithDisc = Team1;

DistanceThrown = 3;

//Console.WriteLine($"Computer Turnover on the {discLocationY} yard line.");

Team2TurnedOver = true;

}

break;

default:

probailityOfCatch = probability.Next(0, 100);

if (probailityOfCatch > 10)

{ // Then the catch is complete, and yardage gained will

Thrower = Catcher;

Catcher = p1;

DistanceThrown = -1;

//Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName} on the

{discLocationY");

Team2TurnedOver = false;

}

else

{ // Turnover

Thrower = Catcher;

// Add a turnover to their stats right here

TeamWithDisc = Team1;

DistanceThrown = 1;

Team2TurnedOver = true;

//Console.WriteLine($"Computer Turnover on the {discLocationY} yard line.");

}

break;

}

Thread.Sleep(1000); // Pauses program for 0.1 seconds so that user can see what computer does

return Catcher;

}

public Player ChoosePlayerToThrowTo(Player p1, Player p2, Player p3)

{

Console.WriteLine($"1: {p1.FirstName} {p1.LastName} {p1.Overall} for a handle reset");

Console.WriteLine($"2: {p2.FirstName} {p2.LastName} {p2.Overall} for an intermediate cutter throw");

Console.WriteLine($"3: {p3.FirstName} {p3.LastName} {p3.Overall} for a deep cutter throw");

int potentialCatcher = int.Parse(Console.ReadLine());

Random probability = new Random();

switch (potentialCatcher)

{

case 1:

int probailityOfCatch = probability.Next(0, 100);

if (probailityOfCatch > 10)

{ // Then the catch is complete, and yardage gained will

Thrower = Catcher;

Catcher = p1;

DistanceThrown = 1;

Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName}");

}

else

{ // Turnover

Thrower = Catcher;

// Add a turnover to their stats right here

Team1TurnedOver = true;

TeamWithDisc = Team2;

DistanceThrown = -1;

Console.WriteLine("User Turnover.");

}

break;

case 2:

probailityOfCatch = probability.Next(0, 100);

if (probailityOfCatch > 25)

{ // Then the catch is complete, and yardage gained will

Thrower = Catcher;

Catcher = p2;

DistanceThrown = 2;

Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName}");

}

else

{ // Turnover

Thrower = Catcher;

// Add a turnover to their stats right here

TeamWithDisc = Team2;

DistanceThrown = -2;

Console.WriteLine("User Turnover.");

Team1TurnedOver = true;

}

break;

case 3:

probailityOfCatch = probability.Next(0, 100);

if (probailityOfCatch > 40)

{ // Then the catch is complete, and yardage gained will

Thrower = Catcher;

Catcher = p3;

DistanceThrown = 3;

Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName}");

}

else

{ // Turnover

Thrower = Catcher;

// Add a turnover to their stats right here

TeamWithDisc = Team2;

DistanceThrown = -3;

Console.WriteLine("User Turnover.");

Team1TurnedOver = true;

}

break;

default:

probailityOfCatch = probability.Next(0, 100);

if (probailityOfCatch > 10)

{ // Then the catch is complete, and yardage gained will

Thrower = Catcher;

Catcher = p1;

DistanceThrown = 1;

Console.WriteLine($"Pass Completed to {Catcher.FirstName} {Catcher.LastName}");

}

else

{ // Turnover

Thrower = Catcher;

// Add a turnover to their stats right here

TeamWithDisc = Team2;

DistanceThrown = -1;

Console.WriteLine("User Turnover.");

Team1TurnedOver = true;

}

break;

}

return Catcher;

}

public int DetermineYardsGained()

{

int numYardsGained = 0;

Random probability = new Random();

switch (DistanceThrown)

{

case 1:

numYardsGained = probability.Next(-5, 5);

discLocationY += numYardsGained;

break;

case -1:

numYardsGained = probability.Next(-5, 5);

discLocationY -= numYardsGained;

break;

case 2:

numYardsGained = probability.Next(5, 12);

discLocationY += numYardsGained;

break;

case -2:

numYardsGained = probability.Next(5, 12);

discLocationY -= numYardsGained;

break;

case 3:

numYardsGained = probability.Next(12, 50);

discLocationY += numYardsGained;

break;

case -3:

numYardsGained = probability.Next(12, 50);

discLocationY -= numYardsGained;

break;

default:

numYardsGained = probability.Next(1, 10);

discLocationY += numYardsGained;

break;

}

return numYardsGained;

}

public bool DetermineNextPointsInfo(int winner)

{ // this is used for changing who has posession next time

// It also returns a boolean for determining if game is over

bool gameOver = false;

if (winner == 1)

{

Team1.startPointWithDisc = false;

Team2.startPointWithDisc = true;

ScoreTeam1 = ScoreTeam1 + 1;

discLocationY = 70;

if (FullProgram.Verbosity == 1)

Console.WriteLine($"{Team1.Name} {Team1.Mascot} Scored! ");

}

else if (winner == 2)

{

Team1.startPointWithDisc = true;

Team2.startPointWithDisc = false;

ScoreTeam2 = ScoreTeam2 + 1;

discLocationY = 0;

if (FullProgram.Verbosity == 1)

Console.WriteLine($"{Team2.Name} {Team2.Mascot} Scored! ");

}

else

{

Console.WriteLine("Default winner case entered.");

}

// 15 is when the game is over

if (ScoreTeam1 == 15)

{

gameOver = true;

}

else if (ScoreTeam2 == 15)

{

gameOver = true;

}

else

{

gameOver = false;

}

return gameOver;

}

public int CalculateProbabilityForWinner()

{

int probability;

return probability = DifferenceInTeamOverall + 50;

}

public int CalculateDifferenceInTeamOverall()

{ // Calculates the difference between the team's overall skill

int differenceInTeamOverall = 1;

if (Team1.Overall > Team2.Overall)

{

BetterTeam = 1;

differenceInTeamOverall = Team1.Overall - Team2.Overall;

}

else if (Team1.Overall < Team2.Overall)

{

BetterTeam = 2;

differenceInTeamOverall = Team2.Overall - Team1.Overall;

}

else

{

BetterTeam = 0;

differenceInTeamOverall = 0;

}

return differenceInTeamOverall;

}

public int CalculateDifferenceInLinesOverall()

{ // Calculates the difference between the team's overall skill

int differenceInTeamOverall = 1;

if (Team1.OLineOverall > Team2.DLineOverall)

{

BetterLine = 1;

DifferenceInLineOverall = Team1.OLineOverall - Team2.DLineOverall;

}

else if (Team1.OLineOverall < Team2.DLineOverall)

{

BetterTeam = 2;

differenceInTeamOverall = Team2.Overall - Team1.Overall;

}

else

{

BetterTeam = 0;

differenceInTeamOverall = 0;

}

return differenceInTeamOverall;

}

public void PrintScoreboard()

{ // Prints the score of the current game

if (FullProgram.Verbosity == 1)

{

Console.WriteLine("----------------------------------");

Console.WriteLine($"{Team1.Name} {Team1.Mascot} {Team1.Overall}: {ScoreTeam1}");

Console.WriteLine($"{Team2.Name} {Team2.Mascot} {Team2.Overall}: {ScoreTeam2}");

Console.WriteLine("----------------------------------");

}

}

}

}

**Season Class:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace Frisbeev01

{

public class Season

{

public List<Team> ListTeams = new List<Team>(10);

public List<Team> Standings = new List<Team>();

public List<Team> SortedStandings = new List<Team>();

public List<Team> SortedTourneyStandings = new List<Team>();

public List<Player> FreeAgents = new List<Player>();

public Team Team1 = new Team();

public Team Team2 = new Team();

public Team Team3 = new Team();

public Team Team4 = new Team();

public Team Team5 = new Team();

public Team Team6 = new Team();

public Team Team7 = new Team();

public Team Team8 = new Team();

public Team Team9 = new Team();

public Team Team10 = new Team();

public void FillLeagueWithRandomTeams()

{

Create create = new Create();

ListTeams.Add(Team1 = create.ConstructFullTeam());

ListTeams.Add(Team2 = create.ConstructFullTeam());

ListTeams.Add(Team3 = create.ConstructFullTeam());

ListTeams.Add(Team4 = create.ConstructFullTeam());

ListTeams.Add(Team5 = create.ConstructFullTeam());

ListTeams.Add(Team6 = create.ConstructFullTeam());

ListTeams.Add(Team7 = create.ConstructFullTeam());

ListTeams.Add(Team8 = create.ConstructFullTeam());

ListTeams.Add(Team9 = create.ConstructFullTeam());

ListTeams.Add(Team10 = create.ConstructFullTeam());

//for (int i = 0; i < 10; i++)

//{

// Team team = create.ConstructFullTeam();

// if (FullProgram.Verbosity == 2)

// {

// Console.WriteLine($"Team Name: {team.Name} Team Overall: {team.Overall}");

// Console.WriteLine($"OLine Overall: {team.OLineOverall}, DLine Overall:

{team.DLineOverall}");

// }

// ListTeams.Add(team);

//}

}

public void SimulateSeason(int userChoice)

{

FillLeagueWithRandomTeams();

ListTeams.Add(Team1);

ListTeams.Add(Team2);

ListTeams.Add(Team3);

ListTeams.Add(Team4);

ListTeams.Add(Team5);

ListTeams.Add(Team6);

ListTeams.Add(Team7);

ListTeams.Add(Team8);

ListTeams.Add(Team9);

ListTeams.Add(Team10);

for (int i = 0; i < 10; i++)

{

Standings.Add(ListTeams[i]);

}

for (int i = 1; i < 10; i++) // Play each week of the 9 game season

{

SimulateWeek(i, userChoice);

}

// Creates teams for keeping track of stats and info between games

Team1.CalculateTeamPointDifferential();

Team2.CalculateTeamPointDifferential();

Team3.CalculateTeamPointDifferential();

Team4.CalculateTeamPointDifferential();

Team5.CalculateTeamPointDifferential();

Team6.CalculateTeamPointDifferential();

Team7.CalculateTeamPointDifferential();

Team8.CalculateTeamPointDifferential();

Team9.CalculateTeamPointDifferential();

Team10.CalculateTeamPointDifferential();

//PrintStandings();

FinishAndPrintSeasonResults(userChoice);

}

public Team SimulateTournament(int userChoice)

{

Game game1 = new Game(SortedStandings[6], SortedStandings[9], userChoice); // Round 1 play in game

game1.FullGame();

SortedTourneyStandings.Add(game1.Loser);

Game game2 = new Game(SortedStandings[7], SortedStandings[8], userChoice); // Round 1 play in game

game2.FullGame();

SortedTourneyStandings.Add(game2.Loser);

Game game3 = new Game(SortedStandings[0], game2.Winner, userChoice); // Round 2 Left Top Bracket

game3.FullGame();

SortedTourneyStandings.Add(game3.Loser);

Game game4 = new Game(SortedStandings[1], game1.Winner, userChoice); // Round 2 Right Top

Bracket

game4.FullGame();

SortedTourneyStandings.Add(game4.Loser);

Game game5 = new Game(SortedStandings[2], SortedStandings[5], userChoice); // Round 2 // Right

Bottom Bracket

game5.FullGame();

SortedTourneyStandings.Add(game5.Loser);

Game game6 = new Game(SortedStandings[3], SortedStandings[4], userChoice); // Round 2 // Left

Bottom Bracket

game6.FullGame();

SortedTourneyStandings.Add(game6.Loser);

Game game7 = new Game(game3.Winner, game6.Winner, userChoice); // Semifinals Left Bracket

game7.FullGame();

SortedTourneyStandings.Add(game7.Loser);

Game game8 = new Game(game4.Winner, game5.Winner, userChoice); // Semifinals Right Bracket

game8.FullGame();

SortedTourneyStandings.Add(game8.Loser);

Game game9 = new Game(game7.Winner, game8.Winner, userChoice); // Championship Game

game9.FullGame();

SortedTourneyStandings.Add(game9.Loser);

SortedTourneyStandings.Add(game9.Winner);

SortedTourneyStandings.Reverse();

return game9.Winner;

}

public void SimulateWeek(int week, int userChoice)

{

switch (week)

{ // Simulates 5 games for 9 weeks, so everyone plays each other once

case 1:

Game game1 = new Game(Team1, Team10, userChoice);

Game game2 = new Game(Team2, Team9, userChoice);

Game game3 = new Game(Team3, Team8, userChoice);

Game game4 = new Game(Team4, Team7, userChoice);

Game game5 = new Game(Team5, Team6, userChoice);

game1.FullGame();

game2.FullGame();

game3.FullGame();

game4.FullGame();

game5.FullGame();

break;

case 2:

Game game6 = new Game(Team7, Team2, userChoice);

Game game7 = new Game(Team6, Team3, userChoice);

Game game8 = new Game(Team5, Team4, userChoice);

Game game9 = new Game(Team10, Team9, userChoice);

Game game10 = new Game(Team8, Team1, userChoice);

game6.FullGame();

game7.FullGame();

game8.FullGame();

game9.FullGame();

game10.FullGame();

break;

case 3:

Game game11 = new Game(Team9, Team8, userChoice);

Game game12 = new Game(Team1, Team7, userChoice);

Game game13 = new Game(Team6, Team2, userChoice);

Game game14 = new Game(Team5, Team3, userChoice);

Game game15 = new Game(Team4, Team10, userChoice);

game11.FullGame();

game12.FullGame();

game13.FullGame();

game14.FullGame();

game15.FullGame();

break;

case 4:

Game game16 = new Game(Team4, Team3, userChoice);

Game game17 = new Game(Team10, Team8, userChoice);

Game game18 = new Game(Team9, Team7, userChoice);

Game game19 = new Game(Team6, Team1, userChoice);

Game game20 = new Game(Team2, Team5, userChoice);

game16.FullGame();

game17.FullGame();

game18.FullGame();

game19.FullGame();

game20.FullGame();

break;

case 5:

Game game21 = new Game(Team10, Team6, userChoice);

Game game22 = new Game(Team7, Team5, userChoice);

Game game23 = new Game(Team2, Team1, userChoice);

Game game24 = new Game(Team8, Team4, userChoice);

Game game25 = new Game(Team9, Team3, userChoice);

game21.FullGame();

game22.FullGame();

game23.FullGame();

game24.FullGame();

game25.FullGame();

break;

case 6:

Game game26 = new Game(Team5, Team9, userChoice);

Game game27 = new Game(Team8, Team6, userChoice);

Game game28 = new Game(Team7, Team10, userChoice);

Game game29 = new Game(Team3, Team2, userChoice);

Game game30 = new Game(Team1, Team4, userChoice);

game26.FullGame();

game27.FullGame();

game28.FullGame();

game29.FullGame();

game30.FullGame();

break;

case 7:

Game game31 = new Game(Team3, Team1, userChoice);

Game game32 = new Game(Team9, Team4, userChoice);

Game game33 = new Game(Team8, Team5, userChoice);

Game game34 = new Game(Team7, Team6, userChoice);

Game game35 = new Game(Team10, Team2, userChoice);

game31.FullGame();

game32.FullGame();

game33.FullGame();

game34.FullGame();

game35.FullGame();

break;

case 8:

Game game36 = new Game(Team6, Team4, userChoice);

Game game37 = new Game(Team5, Team10, userChoice);

Game game38 = new Game(Team1, Team9, userChoice);

Game game39 = new Game(Team2, Team8, userChoice);

Game game40 = new Game(Team3, Team7, userChoice);

game36.FullGame();

game37.FullGame();

game38.FullGame();

game39.FullGame();

game40.FullGame();

break;

case 9:

Game game41 = new Game(Team8, Team7, userChoice);

Game game42 = new Game(Team4, Team2, userChoice);

Game game43 = new Game(Team10, Team3, userChoice);

Game game44 = new Game(Team1, Team5, userChoice);

Game game45 = new Game(Team6, Team9, userChoice);

game41.FullGame();

game42.FullGame();

game43.FullGame();

game44.FullGame();

game45.FullGame();

break;

default:

Console.WriteLine("Default Case entered");

break;

}

}

public void PrintTourneyResults(Team team)

{

Console.WriteLine($"The Tournament Winner is: {team.Name} {team.Mascot}! Congratulations!");

for (int i = 0; i < 10; i++)

{

Team team1 = SortedTourneyStandings[i];

Console.WriteLine($"{i + 1}| {team1.Name} {team1.Mascot} {team1.Overall} Wins: {team1.Wins}, Losses: {team1.Losses}, PointsFor: {team1.TotalPoints}, PointsAgainst: {team1.TotalPointsAgainst}, PointDiff: {team1.TotalPointDifferential}");

}

}

public void FinishAndPrintSeasonResults(int userChoice)

{

Console.WriteLine("Regular Season Results: ");

CalculateStandings();

for (int i = 0; i < 10; i++)

{

Team team = SortedStandings[i];

Console.WriteLine($"{i + 1}| {team.Name} {team.Mascot} {team.Overall} Wins: {team.Wins}, Losses: {team.Losses}, PointsFor: {team.TotalPoints}, PointsAgainst: {team.TotalPointsAgainst}, PointDiff: {team.TotalPointDifferential}");

}

Thread.Sleep(5000); // Pauses program for 0.1 seconds so that user can see what computer does

Team team1 = SimulateTournament(userChoice);

PrintTourneyResults(team1);

}

public void PrintTourneyStandings()

{

}

public void PrintStandings()

{

Thread.Sleep(1000);

Console.WriteLine("Final Standings: ");

CalculateStandings();

for (int i = 0; i < 10; i++)

{

Team team = SortedStandings[i];

Console.WriteLine($"{i + 1}| {team.Name} {team.Mascot} {team.Overall} Wins: {team.Wins}, Losses: {team.Losses}, PointsFor: {team.TotalPoints}, PointsAgainst: {team.TotalPointsAgainst}, PointDiff: {team.TotalPointDifferential}");

}

}

public void CalculateStandings()

{ // Calculates the end of the year standings based on wins and then point differential. This determines the seeding for the tourney

SortedStandings = Standings.OrderBy(Team => Team.Wins).ThenBy(Team => Team.TotalPointDifferential).ToList();

SortedStandings.Reverse(); // Flips list around, because for some reason, it was backwards

}

public void CreateFreeAgents()

{ // Creates 50 random players for there to be free agents

// Someday, users will be able to add free agents to their team

Create create = new Create();

Player player = new Player();

for (int i = 0; i < 50; i++)

{

player = create.CreateRandomPlayer();

FreeAgents.Add(player);

}

}

public void PrintOutFreeAgents()

{ // Prints out the players that are not on a team

FreeAgents = FreeAgents.OrderBy(Player => Player.Overall).ToList();

FreeAgents.Reverse();

foreach (Player player in FreeAgents)

{

player.printAllAttributes();

}

}

public void FillLeagueWithManualTeams()

{ // Allow user to create teams and then create a league of them

;

}

}

}

**Create Class**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Frisbeev01

{

public class Create

{

Player player = new Player();

Team team = new Team();

//private int seed = DateTime.Now.Ticks.GetHashCode();

//public int seed = 10000;

Random random = new Random();

Season season = new Season();

public List<Player> FreeAgents = new List<Player>();

public Player UserCreatePlayer()

{ // Allows thee user to create their own player, with their own abilities

Console.WriteLine("First name: ");

string fn = Console.ReadLine();

Console.WriteLine("Last name: ");

string ln = Console.ReadLine();

Console.WriteLine("Jersey Number: ");

string jn = Console.ReadLine();

Console.WriteLine("Speed: ");

int s = int.Parse(Console.ReadLine());

Console.WriteLine("Jumping: ");

int j = int.Parse(Console.ReadLine());

Console.WriteLine("Flick Distance: ");

int fd = int.Parse(Console.ReadLine());

Console.WriteLine("Flick Accuracy: ");

int fa = int.Parse(Console.ReadLine());

Console.WriteLine("Backhand Accuracy: ");

int ba = int.Parse(Console.ReadLine());

Console.WriteLine("Backhand Distance: ");

int bd = int.Parse(Console.ReadLine());

Console.WriteLine("Cutter Defense: ");

int cd = int.Parse(Console.ReadLine());

Console.WriteLine("Handle Defense: ");

int hd = int.Parse(Console.ReadLine());

Console.WriteLine("Agility: ");

int ag = int.Parse(Console.ReadLine());

Console.WriteLine("Handle Cuts: ");

int hc = int.Parse(Console.ReadLine());

Console.WriteLine("Under Cuts: ");

int uc = int.Parse(Console.ReadLine());

Console.WriteLine("Deep Cuts: ");

int dc = int.Parse(Console.ReadLine());

Player player1 = new Player(fn, ln, jn, s, j, fd, fa, ba, bd, cd, hd, ag, hc, uc, dc);

FreeAgents.Add(player1);

//Console.WriteLine("Create RandomPlayer finished");

return player1;

}

public Team UserCreateTeam()

{ // User creates a team

Console.WriteLine("Team Name: ");

string name = Console.ReadLine();

Console.WriteLine("Team Mascot: ");

string mascot = Console.ReadLine();

Team team = new Team(name, mascot);

return team;

}

public void UserFillTeam()

{ // User chooses players to fill team with

;

}

public void UserCreateFullTeam()

{ // puts players into full team. This calls create and fill team funcitons

;

}

public List<Player> FillFreeAgents()

{

return FreeAgents;

}

public Player CreateRandomHandle()

{

string fn = GenerateRandomFirstName();

string ln = GenerateRandomLastName();

string jn = Convert.ToString(random.Next(0, 100));

int s = random.Next(40, 80);

int j = random.Next(40, 80);

int fd = random.Next(60, 80);

int fa = random.Next(60, 90);

int ba = random.Next(60, 90);

int bd = random.Next(60, 90);

int cd = random.Next(40, 80);

int hd = random.Next(40, 80);

int ag = random.Next(40, 80);

int hc = random.Next(60, 90);

int uc = random.Next(40, 60);

int dc = random.Next(40, 60);

Player player1 = new Player(fn, ln, jn, s, j, fd, fa, ba, bd, cd, hd, ag, hc, uc, dc);

player1.IsHandle = true;

if (FullProgram.Verbosity == 3)

Console.WriteLine("Create RandomPlayer finished");

return player1;

}

public Player CreateRandomCutter()

{

string fn = GenerateRandomFirstName();

string ln = GenerateRandomLastName();

string jn = Convert.ToString(random.Next(0, 100));

int s = random.Next(40, 80);

int j = random.Next(40, 80);

int fd = random.Next(40, 70);

int fa = random.Next(40, 70);

int ba = random.Next(40, 70);

int bd = random.Next(40, 70);

int cd = random.Next(40, 80);

int hd = random.Next(40, 80);

int ag = random.Next(40, 80);

int hc = random.Next(40, 80);

int uc = random.Next(60, 90);

int dc = random.Next(60, 90);

Player player1 = new Player(fn, ln, jn, s, j, fd, fa, ba, bd, cd, hd, ag, hc, uc, dc);

player1.IsHandle = false;

if (FullProgram.Verbosity == 3)

Console.WriteLine("Create RandomPlayer finished");

return player1;

}

public Player CreateRandomPlayer()

{ // Creates a player with good attributes

string fn = GenerateRandomFirstName();

string ln = GenerateRandomLastName();

string jn = Convert.ToString(random.Next(0, 100));

int s = random.Next(40, 80);

int j = random.Next(40, 80);

int fd = random.Next(40, 80);

int fa = random.Next(40, 80);

int ba = random.Next(40, 80);

int bd = random.Next(40, 80);

int cd = random.Next(40, 80);

int hd = random.Next(40, 80);

int ag = random.Next(40, 80);

int hc = random.Next(40, 80);

int uc = random.Next(40, 80);

int dc = random.Next(40, 80);

Player player1 = new Player(fn, ln, jn, s, j, fd, fa, ba, bd, cd, hd, ag, hc, uc, dc);

player1.GameAssists = 0;

player1.GameGoals = 0;

player1.SeasonAssists = 0;

player1.SeasonGoals = 0;

if (FullProgram.Verbosity == 3)

Console.WriteLine("Create RandomPlayer finished");

return player1;

}

public string GenerateRandomFirstName()

{

List<string> listFirst = player.ReadInFirstNames();

int index = random.Next(1, 1000);

string firstName = listFirst[index];

return firstName;

}

public string GenerateRandomLastName()

{

List<string> listLast = player.ReadInLastNames();

int index1 = random.Next(1, 1000);

string lastName = listLast[index1];

return lastName;

}

public Team GenerateRandomTeam()

{

int index = random.Next(1, 24);

string teamName = GenerateTeamName(index);

string mascotName = GenerateTeamMascot(index);

Team team1 = new Team(teamName, mascotName);

if (FullProgram.Verbosity == 3)

{

Console.WriteLine("Generate RandomTeam finished");

Console.WriteLine($"team.Name = {team1.Name} , mascot = {team1.Mascot}");

}

return team1;

}

public Game GenerateRandomGame(int userChoice)

{

Team team1 = ConstructFullTeam();

Team team2 = ConstructFullTeam();

Game game1 = new Game(team1, team2, userChoice);

return game1;

}

public string GenerateTeamName(int index)

{

//Random random = new Random(DateTime.Now.Ticks.GetHashCode());

List<string> listTeams = team.ReadInTeams();

string teamName = listTeams[index];

if (FullProgram.Verbosity == 3)

{

Console.WriteLine("Generate Team name finished");

Console.WriteLine($"teamName = {teamName}");

}

return teamName;

}

public string GenerateTeamMascot(int index)

{

//Random random = new Random(DateTime.Now.Ticks.GetHashCode());

List<string> listMascots = team.ReadInMascots();

string teamMascot = listMascots[index];

if (FullProgram.Verbosity == 3)

{

Console.WriteLine("Generate Team mascot finished");

Console.WriteLine($"teamMascot = {teamMascot}");

}

return teamMascot;

}

public int GenerateTeamOverall(int index)

{

int overall = random.Next(0, 100);

return overall;

}

public Team ConstructFullTeam()

{

team = GenerateRandomTeam();

team.FillTeamWithRandomPlayers();

team.CompCreateLines();

return team;

}

public Team CreateLineFromList(List<Player> players,bool isOline)

{

Team team = new Team();

for (int i = 0; i < 7; i++)

{

team.AddPlayer(players[i]);

if (isOline == true)

{

if (players[i].IsHandle == true)

{

team.OLineHandles.Add(players[i]);

}

else

{

team.OLineCutters.Add(players[i]);

}

}

else

{

if (players[i].IsHandle == true)

{

team.DLineHandles.Add(players[i]);

}

else

{

team.DLineCutters.Add(players[i]);

}

}

}

return team;

}

public Team CreateTeamFromList(List<Player> players)

{

Team team = new Team();

for (int i = 0; i < 14; i++)

{

team.AddPlayer(players[i]);

}

return team;

}

}

}

Full Program Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Frisbeev01

{ // This is used solely for global constants. It chooses how much info to print out to the user, as well as

what gameplay is being played

static class FullProgram

{

public const int Verbosity = 1; // 1 means only important info. 2 means more info. 3 means literally everything

public const int GameType = 2; // 1 means simulation, 2 means manual play

}

}

Print Class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Frisbeev01

{

public class Print

{

public void PrintRoster(Team team)

{ // Prints Roster. Sometimes the OLine or DLine will be passed in, and it will be used to print out

the line

Console.WriteLine("----------------------------------");

team.printTeam();

foreach (Player player in team.TeamOfPlayers)

{

player.printAllAttributes();

}

Console.WriteLine("----------------------------------");

}

public void PrintOLine(Team team)

{

Console.WriteLine("----------------------------------");

team.printTeam();

foreach (Player player in team.OLineList)

{

player.printBasicInfo();

}

Console.WriteLine("----------------------------------");

}

public void PrintDLine(Team team)

{

Console.WriteLine("----------------------------------");

team.printTeam();

foreach (Player player in team.DLineList)

{

player.printBasicInfo();

}

Console.WriteLine("----------------------------------");

}

}

}

**Main Menu Class**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Frisbeev01

{

public class MainMenu

{

public int printMenu()

{

int answer = 1;

bool isCorrect = false;

while (isCorrect == false)

{

Console.WriteLine("Welcome to Joel's Frisbee Program");

Console.WriteLine("Choose the game type: ");

Console.WriteLine("1. Play Single Game");

Console.WriteLine("2. Simulate Single Game");

Console.WriteLine("3. Play Full Season");

Console.WriteLine("4. Simulate Full Season");

if (!int.TryParse(Console.ReadLine(), out answer))

{

Console.WriteLine("Please enter a number between 1 and 4");

isCorrect = false;

}

else

{

isCorrect = true;

}

}

return answer;

}

}

}

**Test Class:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Frisbeev01

{

public class Test

{

public static void Main()

{

//FrisbeeGui.MainWindow;

//Create create = new Create();

//Team team1 = create.GenerateRandomTeam();

//Team team2 = create.GenerateRandomTeam();

//CheckTeams check = new CheckTeams();

//team2.Name = check.checkNames(team1, team2);

//team2.Mascot = check.checkMascots(team1, team2);

//create.FillTeamWithRandomPlayers(team1);

//create.FillTeamWithRandomPlayers(team2);

//Game game1 = new Game(team1, team2);

//game1.startGame();

MainMenu menu = new MainMenu();

Season season = new Season();

Create create = new Create();

int endProgram = 1;

while (endProgram == 1)

{

int userChoice = menu.printMenu();

switch (userChoice)

{

case 1:

// User chooses to Play a single game

Game game1 = create.GenerateRandomGame(1);

game1.FullGame();

break;

case 2:

// User chooses to Simulate a single game

Game game2 = create.GenerateRandomGame(2);

game2.FullGame();

break;

case 3:

// User chooses to play a full season

season.SimulateSeason(1);

//season.SimulateTournament(userChoice);

break;

case 4:

// User chooses to simulate a full season

season.SimulateSeason(2);

//season.SimulateTournament(userChoice);

break;

default:

break;

}

bool isCorrect = false;

while (isCorrect == false)

{

Console.WriteLine("Would you like to re run the program (1 for yes or 2 for no)");

if (!int.TryParse(Console.ReadLine(), out endProgram))

{

Console.WriteLine("Please enter a 1 to run the program again, or a 2 to stop the

program");

isCorrect = false;

}

else

{

isCorrect = true;

}

}

}

//season.CreateFreeAgents();

//season.PrintOutFreeAgents();

}

}

public class CheckTeams

{

Random random = new Random(DateTime.Now.Ticks.GetHashCode());

Create create = new Create();

Test test = new Test();

public string checkNames(Team team1, Team team2)

{// Checks to make sure that both teams don't have the same name

string name = team2.Name;

while (team1.Name == team2.Name)

{ // Makes sure that both teams aren't the same name

// Changes team 2 because we are okay with team1 being the sme

int index = random.Next(1, 24);

name = create.GenerateTeamName(index);

}

if (FullProgram.Verbosity == 3)

Console.WriteLine($"Check Mascots {team2.Name} {team2.Mascot}");

return name;

}

public string checkMascots(Team team1, Team team2)

{ // Checks to make sure teams dont have same mascot

string mascot = team2.Mascot;

while (team1.Mascot == team2.Mascot)

{

int index = random.Next(1, 24);

mascot = create.GenerateTeamName(index);

}

if (FullProgram.Verbosity == 3)

Console.WriteLine($"Check Mascots {team2.Name} {team2.Mascot}");

return mascot;

}

}

}

**Unit Test Class:**

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Frisbeev01;

namespace UnitTestProject2

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestsRandomNumGenerator()

{ // This method tests whether simulate point winner works correctly

// Essentially, it generates a random number, and then determines if it is less than the probability

Random random = new Random();

int randomNum = random.Next(1, 100);

bool expectedBool = true;

bool actualBool = false;

if (randomNum > 100 || randomNum < 1)

{

actualBool = false;

}

else

{

actualBool = true;

}

Assert.AreEqual(expectedBool, actualBool);

//, "The random number was not generated accurately");

}

}

}