Formalized Proposal

I aim to expedite the process of determining All-NBA Teams for a given season. The NBA determines, out of all the players in the league, which players are the best at their position. The All-NBA Teams consist of two guards, two forwards, and one center. There is an All-NBA First Team, as well as a second team. The first team is the higher honor.

The program I will create will speed up this process by allowing the user to enter player data for the top four players at each position group, and then an interactive GUI will sort those players into an All-NBA First Team and an All-NBA Second Team. As player entries are added, the user will be able to see the data entered into a list in the GUI.

Time/Change Logs

3/23 Stand Up:

Summary: Decided what the initial plan for the program would be, what the program would do, and how it would be implemented

"The problem I would solve is making it easier to generate the best possible basketball lineup with a given pool of available players.

The lineup generator would use a linked list data structure, and players entered in via user input will be placed in a stack. There will be a different linked list for the 3 different positional groups: Guard, forward, and center. The players will enter with their player efficiency rating as their talent measurement. The data will be sorted using an insertion sort, with the best players added to the final roster. For added functionality, I'm going to add an option to sort regardless of position which 5 players are the best."

4/6 Stand Up:

Summary: Began work on the program by establishing a skeleton outline of the classes and methods. Finished the Player class, and finished the insertion sort method.

"So far, I have established the classes that I will be using and I have finished my Player class. Doing the player class wasn't very difficult, because it is similar to a node class. I have also added a method for insertion sorting, as well as a driver that adds a Player to a linked list with attributes based on user input. Some input validation is included, but not much. I don't have any blockers yet, but I imagine I will at some point. I don't need help yet, but I know where to look if I do. I plan to work on my main Builder class next, and see if I can get the general idea working."

4/13 Stand Up:

Summary: Made a switch in data structure choice for the final team list, from a Linked List to an Array. Made more progress in the main class.

"This week, I switched one of the data structures I will use from a linked list to an array, because I feel like it will make more sense for what I'm trying to put together. I've done some work on my classes, and they aren't fully done yet but they're getting close. I don't need help yet but putting together a GUI might be difficult. I plan to finish up my classes soon and work on the GUI."

4/20 Stand Up:

Summary: Slow week without much progress, continued work on main class

"I'm still working on implementing the roster building methods in my main class. I didn't get as much done this week as I would like to, mainly because I was working ahead in other courses, so I guess that's my only blocker so far. I don't need help as of right now, but I may look online for resources that would help with making a GUI. I plan to finish up my main class, and start working on testing after that."

4/20 Retro:

Summary: Change in sorting type was a good choice, and the array data structure would be better for the final team list because of the fixed size.

"What has gone right in my project so far is I think I made a good decision with the type of sort that I chose, and choosing to make a separate Node class with its own properties. What went wrong, which ended up benefiting me, was the fact that I chose the wrong data structure at first. I initially chose a linked list, but I realized that having a fixed size for my data structure would work better and so I switched to an array data structure. I'm not completely done with my classes yet, but I plan to finish those in the next few days, then I plan to work on the testing, then the GUI. I don't really have anything to add in terms of what I will stop doing, but maybe that'll change by the time I finish."

4/26 Stand Up:

Summary: Finished main code application, and began work on the GUI

"I have pretty much finished everything other than input validation and the GUI. The only blocker I have so far is that I am struggling with the GUI, so I will need to figure that out somehow. I do need help, and I have looked online for some tips, but not much of it makes sense. If there are still available office hours between now and the time the project is due, I will try to get some help there. I plan to finish the GUI, finish the input validation, final testing, and then work on the presentation."

4/28 Stand Up:

Summary: Finished GUI, began work on needed input validation and the assignment report and manual

“I basically have everything done, the GUI works better than I thought it would. The last things I need to finish are some input validation and finish up the report, manual, and redo the video presentation. The GUI was a blocker for me, but I was able to overcome it and figure it out. I don't think I need help at this point. I plan to finish up the stuff that I need next.”

4/28 Retro:

Summary: GUI came together well after early struggles, video presentation was not as good as it could’ve been, so I realized I should redo it. Shifted focus from the code portion of the assignment to the written portion

“Like I mentioned in this week’s standup, I was surprised by how well the GUI came together, considering I'm not the most comfortable with them yet. The code is 99% done aside from a few things that I will finish up. Originally, I would've said the GUI is what went wrong, but I guess I don't have anything to add right now for what went wrong. I guess the fact that I need to redo my video presentation would be what went wrong. I will keep working on it up until the 1st and finish up my input validation, and some additional testing. At this point, I should probably just stop putting time into the GUI and really make sure my code is what it needs to be, as well as put more time into the report and manual.”

Lessons Learned

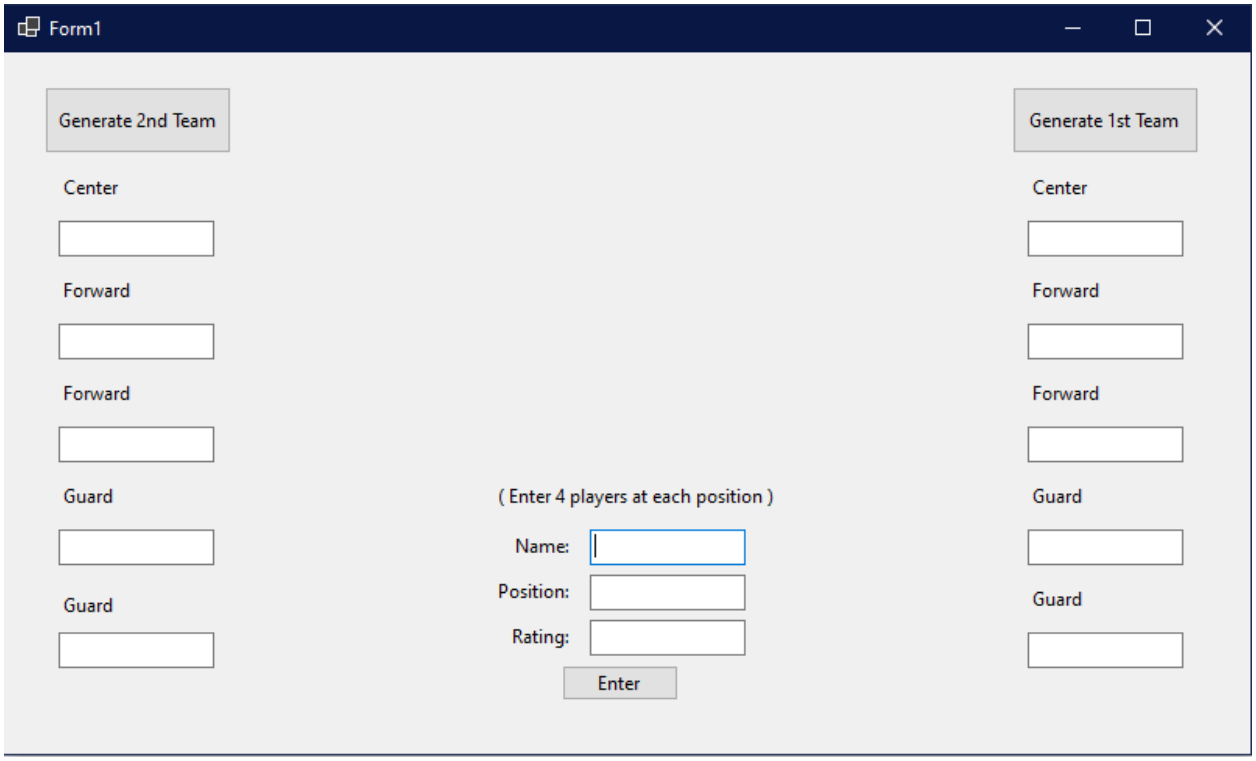
The scope of the project stayed very similar throughout the entire process, mainly because I didn’t want to waste time changing everything halfway through. I did end up making some changes anyway though. Initially, I wanted the additional functionality besides making the best team to be a sort that chose the top team regardless of position. I changed this additional function into the second team feature instead.

The blockers I encountered early on were mainly related to the data structure that I chose. At first, I chose a linked list data structure to hold the final team selections. But, I switched to an array structure because the fixed length ended up working better. The other blocker I had later on was the GUI. Up to that point, I wasn’t very comfortable making GUI’s. After a few hours of struggle and deciding between an MVC and a windows form app, I decided I would do a form app. The form app ended up not being very difficult to work with at all.

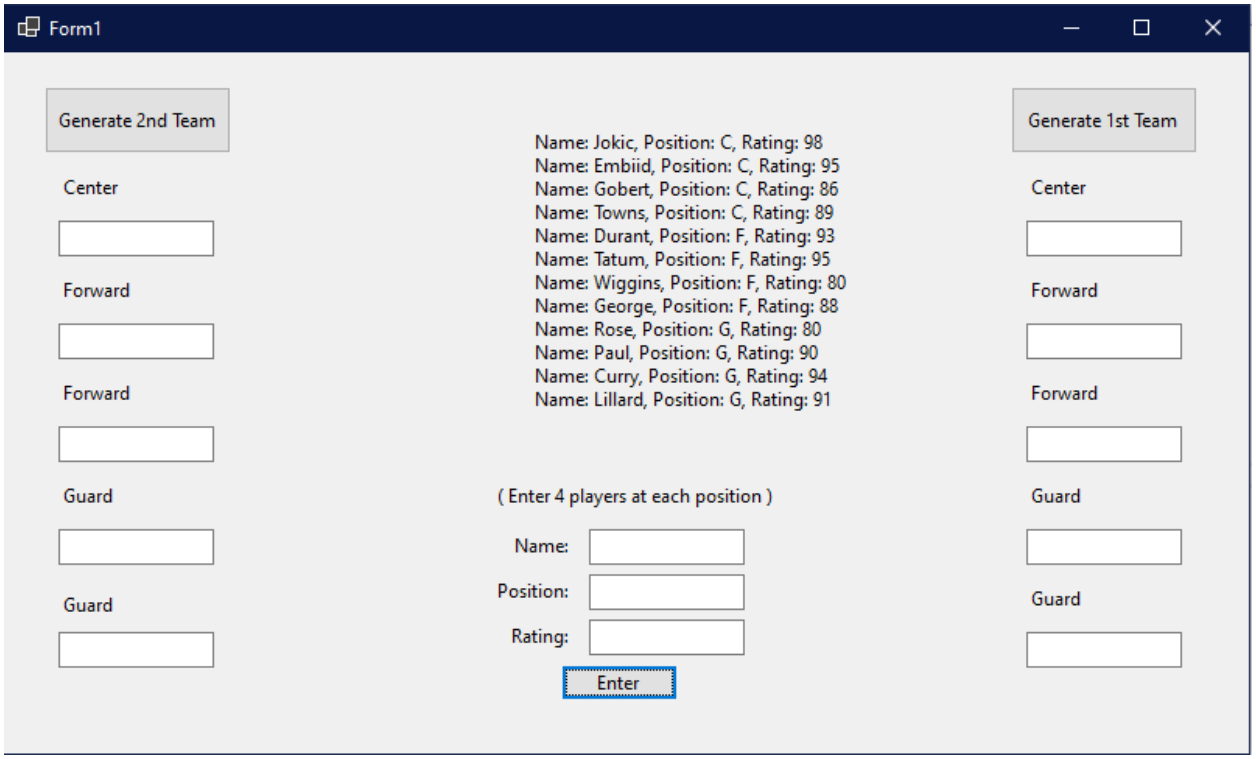
CODE including comments

The project, All-NBA Team Roster Builder is at
https://github.com/jesuisdolfin/Data-Structures-Final-Project

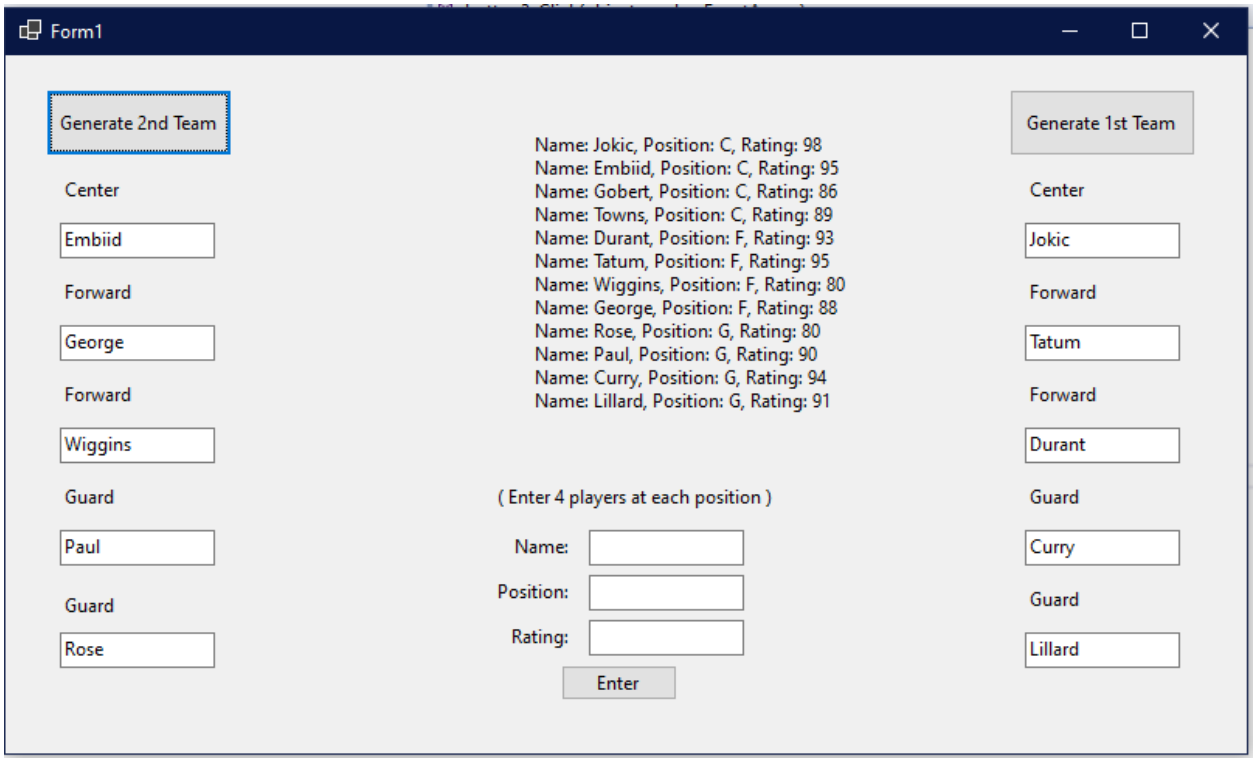
User's Manual



This is the basic GUI for the roster builder program. The user will enter in the player data in the middle 3 text boxes, and then press the “Enter” button to add that player.



As players are entered, their entries will show up in list format above the user input boxes. Be sure to only enter alphabetical characters in the name box, only “C”, “F”, or “G” for the position box, and only enter an integer for the rating box.



The two buttons in the top-right and top-left of the GUI are the buttons that will sort the entered players in the list into an All-NBA First Team and All-NBA Second Team. The results will be shown in the text boxes underneath the buttons.

Conclusion/Summary

Modularity: I made sure to modularize my code well by including a separate class just for the player and their properties. This way, I could just refer to that class in my builder class. I also used an insertion sort method inside of another method, instead of just copying the sort code and including it, which would have added twenty or so extra lines of code to that method.

Efficiency: Using a linked list for each position group of players was a good way to increase efficiency, mainly because the program doesn't know how many players at each position the user will enter, it only knows that it needs a minimum of four. The player class also increased efficiency, because it allowed for the program to consolidate all the data for one player into one object instead of multiple variables.

Robustness: The program includes basic input validation, throwing exceptions for illegal inputs in player entries. Driver file with sample inputs used as well, also those were used to test the program before I implemented the GUI.

Usability: The code accomplishes what it was created to do, which was sort a given list of players into the two best possible teams. The GUI, while it could use some small improvements, is readable to the user and easy to understand. The user's manual shows each element of the GUI and what it does. There is plenty of spacing as well, so each section of the GUI is easily identified.

Should be Readable: The code has plenty of descriptive comments, while also not having too many comments. The indentation is consistent, as well as the curly bracket placement.

Elegant: The code used was very consistent in variable names being descriptive and formatted similarly. The comments were only used when they would need to be used, and the code was modularized to reduce the amount of lines in each class

Overall, the program is a good tool to help increase efficiency in determining the All-NBA Teams. The sorting algorithms behind it make sense, and the GUI is well-organized and is easy to work with as a user. I'm happy with the way it turned out, but there are additions that could be made to it that would be an improvement.

Future versions of the program will hopefully include a better-looking GUI element for the added player entries, instead of just a label that expands. Also, I want to add the player rating besides the name on the final lists, instead of just the name. A flaw with the program is the fact that it assumes that one measurable rating exists to sort players, which is not realistic. I may try to find a way to address this and maybe add an algorithm that would determine the player rating, rather than the user entering it in.