Joe Eacott

11/26/18

EECE5167 User Interface

**NBA Stat Tracker**

**I. Background/Introduction**

The Stat Tracker application is a tool for users who participate in fantasy basketball or are interested in NBA statistics. Since 2015 there have been more than 50 million people who play or participate in fantasy sports and on average are 18 years or older. Of these individuals, the average player spends $556 dollars over the course of a year. (<https://fsta.org/research/industry-demographics/>) This information tells us that there does exist a need for tools that offer players a pleasant experience when it comes to participating in online fantasy sports: after all, most users are interested in positive outcomes!

The millions of people who do play use many avenues to facilitate participation such as online web applications provided by familiar websites like Yahoo or ESPN. These websites allow users to connect with others, create a team from a pool of players (known as a draft), and then play against one another to see who has the best team. The difficult part of this process is knowing who to draft as there are many considerations when an individual pick a player for their fantasy team. This drafting process is usually an education decision on why a player is chosen over another. The education decision is formulated from viewing statistics about a player, looking at historical data, and making predications on how a player will be playing for the year. As demonstrated, gathering statistics about players is the most important part when deciding who to draft.

Now comes the fun part; the rest of the season to improve and play against one another! Once a player (or user) has a team, most often they’ll look to improve it over time as players get injured and must be dropped, other players they thought would do well aren’t and should be traded, or any scenario where a decision to change the team must be made. Once again, this is almost always done by comparing historical data and live data to determine what is a players’ fate. Like the online web applications provided by Yahoo or ESPN that you facilitate participation through, this data is usually found on their site too.

**II. Problem Statement:**

However, sometimes a user is given too much data. Sometimes a user is not present with enough data, too. There are often a few steps involved when looking for a statistic for a player or it is not easily accessible. Sometimes users must have multiple windows open when searching for this information and being able to see their team at the same time. In the end users who are power users or those who end up becoming serious about fantasy basketball will often find themselves using multiple sources and sites for information and decision making. On the flip side, users who want a clean tool that is ad free, only present with relevant information (and not articles about players, coaches, or even videos that can hog screen real-estate), and completely customizable does not exist. We must make our own! But how well can we do?

**III. Proposed Approach/solution**

We can certainly give it our best shot: the NBA Stat Tracker is an application that gives users all the freedom they can ask for. It will include information about the night, the teams that are playing, and a fresh screen to load players and categories into. The GUI is easy to use, easy to maintain, and can be expanded to allow users to have an enormous amount of information at their fingertips. Some of the most important decisions to be made during the season require a user to compare historical data of a player along side live data of a player – a simple task that requires either a complex website full of other, non-relevant information, or the use of multiple windows or tabs to see and compare exactly the information they want. But we shouldn’t stop there, as the GUI should be pleasing to the eye with consistency and expandability.

Note: It should be stated here that when using online web application such as Yahoo or ESPN, these websites are extremely complex with profile creation, credit/debit card processing portals for betting, ability to allow millions of requests per second, and many other features that give it credibility. The Stat Tracker application demonstrated here is designed for the purpose of statistic analysis, better presentation than the platforms, and user centered customization. **It is not an actual gambling application**.

**List of functionalities**

When thinking about a tool that users can use, it must have a few important features:

1. The first is the ability to view any player in a league and see any statistic from a global perspective. This means that a player should be searchable and all historical (or live) data should be viewable and customizable to a users’ interest level.
2. The second important feature is that as players are creating a team (drafting), they should be able to load their entire team in a single view, so they can see how their players are doing, both on a night-per-night basis and from a season-wide basis. This means that as a sports season advances over time, the information displayed should be able to show how players are doing on any given night but also how they are doing for the whole season. This purpose is so users can have all the information they want, on the fly, but presented in a very easy manor.
3. The third important feature is that players should be able to compare any number of players against each other. As a season is progressing, a user’s player may get injured or may be underperforming and a user may want to trade with someone else or pick up a player from free-agency to replace them. This will need to be done on the web application, through which the draft is taking place, but this tool will give accurate results.

Beyond these important features, there are other many secondary features that users should have that do fall into functional features. These features include things such as:

* ability to sort players/numbers either in a descending order or ascending order
* re-order players around their table without a need for re-inputting players
* an ability to clear the table everything if they wish
* an ability to clear a single player or category if they wish
* have a global search to view simple or advanced statistics
* change the current season or year for the purpose of historical statistics

Apart from functional features, there is a list of non-functional features that are equally important to users. This goes along with the user interface as a whole:

* familiar data entry through excel like tables
* familiar dialogs and shortcuts for advanced users
* having a game log to show how a player has done in the last few games
* injury report showing if a player can play or not
* colorized information to indicate a player is currently playing
* allow users to scale the application correctly when using different display settings
* allow users to change the theme from a traditional light theme to a dark theme

**IV. Design**

In order to achieve these functional and non-functional requirements, the use of a familiar yet extensive user interface must support a wide range of objects that users can interact with. The framework that is used to develop the Stat Tracker is called PyQt. The PyQt framework is a widget and event driven framework with familiar widgets like pushbuttons, radio buttons, pick-lists, check boxes, line entry, tables, labels, etc. The widgets are connected to one another through signals and an interaction with a widget “emits” a signal and is then processed sequentially.

This basic model that Qt offers follows very closely with the OAI model, so it will do nicely. The OAI model tells us that there is an interconnection between objects and actions. This makes sense, as an object is something that users interact with, and this interaction occurs through actions taken on the object. The objects in this case are the very programming objects that PyQt offers. Typically, the OAI model the workflow is to select the object graphically, such as using a pointing device like a mouse, and then performing an action on the selected object. The result/effect of the action is shown to the user graphically, too. The objects (widgets) are the interface objects. The interface actions are usually decomposed based on the interface object that is in use.

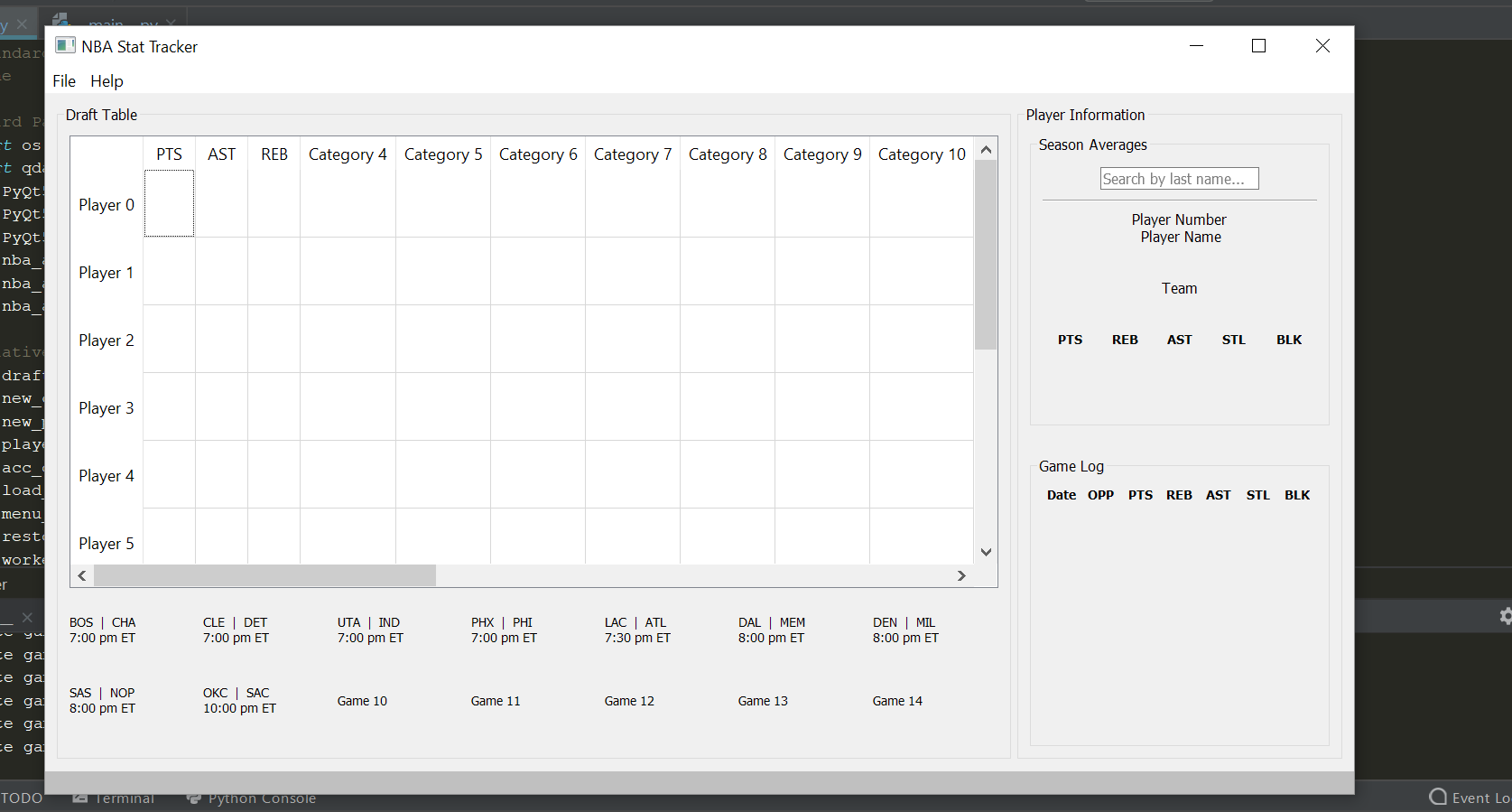
For example, accessing the Menu Bar object has a hierarchy that is more in-depth than a global player search object. Using the Menu Bar, users can open File 🡪 Options, Clear, Export, or Exit. Taking them into the Options menu brings up another dialog with options to change the theme, the font, or the season year. Using the global player search object, in contrast, only requires a user to type in a last night and select a player from a list. The depth here is a level or two less.

This application is designed around the golden rules of UI design and the LUCID methodology. The LUCID design approach gives us a framework to follow. First, create a vision for the application. This has been done through being an active fantasy player and NBA enthusiast. Second, analyze the user needs and development requirements. From using the ESPN online web application for many years, I have seen it grow into a successful application but find myself still criticizing what it lacks. Third, create a design concept and implement a key screen prototype. The GUI has been under construction for quite some time now with many re-works and has finally come to a final prototype. Fourth, refine the prototype to account for design issues and constantly improve it. With the help of my roommate and fellow peers who are UI gurus, the design has had features added and removed that adhere to the main workflow and create an eye-pleasing environment. Fifth, implement the product making late design changes where required. This has entailed removing some slightly older features that no longer support the main workflow. Finally, the product can be rolled out with all the final design changed implemented.

Through this methodology, the golden rules of UI have been highlighted in the design. Consistency has been addressed through colors, font, and headers being of equal size and weight. Universal usability has been addressed through keeping the GUI free from dramatic events and themes. Informative feedback has been addressed through giving users tool tips on objects to offer more information on what a widget can do and through use of a status bar. Dialogs yield closure by giving users fast results and easy to revert changes. Error prevent has been handled by considering use corner cases to ensure users do not run into errors when using the application or looking for specific players. Easy reversal of actions is handled by giving users an ability to clear a player or category by right-clicking on a header and just clicking remove, or users can clear the entire table through file 🡪 clear table. Users are always in control as the GUI was designed to be a customizable environment for users to player with. Finally, short-term memory load has also been addressed by giving users the ability to export and re-load their draft table on a per-session basis.

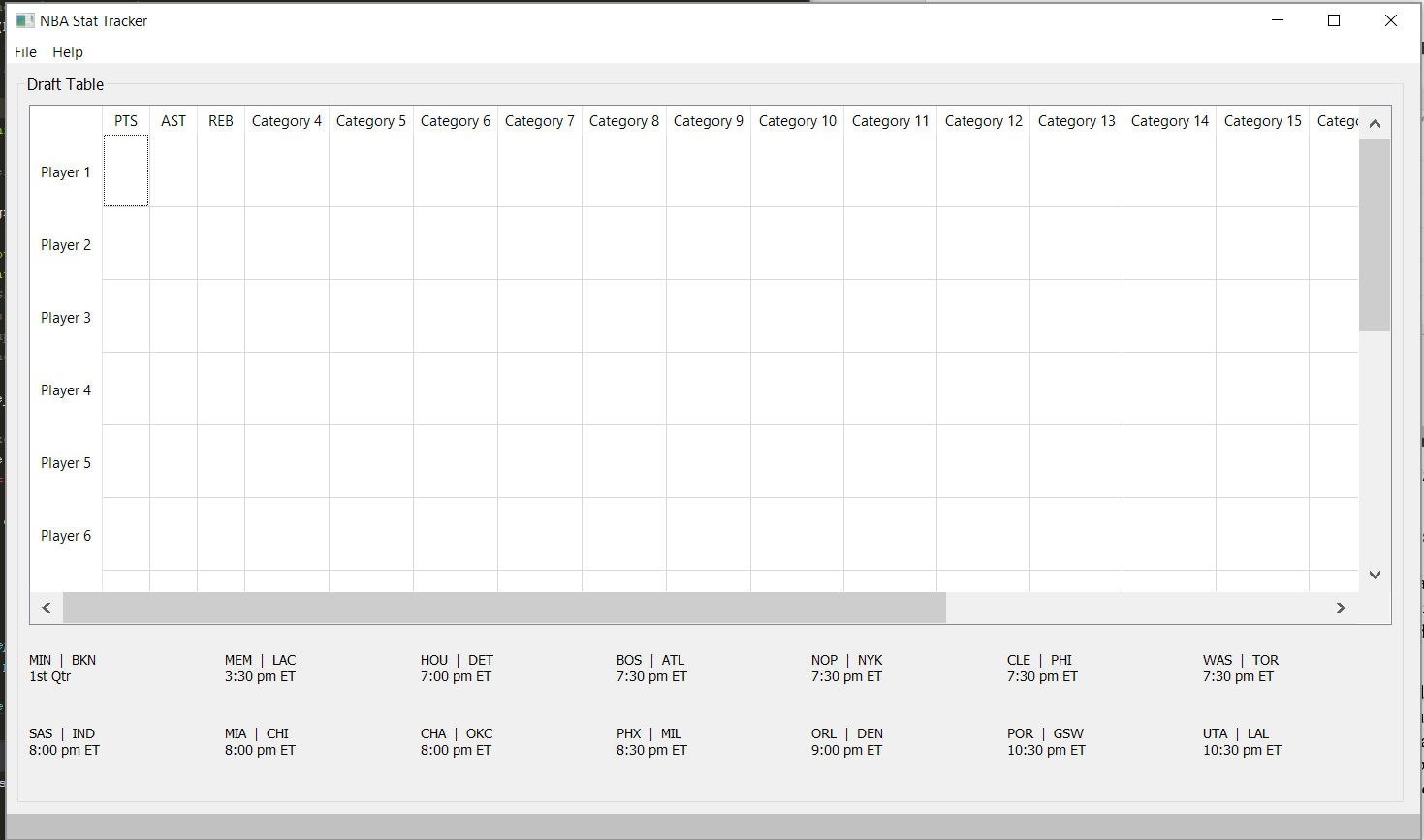
**V. Implementation, Issues, and Results**

The implementation of the application will consist of a main window that will hold the interface objects. This main window will have two sub-windows, one to hold an excel like table and the current games going on for the day, and a second for a global search. A menu bar and status bar will also be included in the main window. The menu bar will hold simple actions such as File, View, Help, etc. and the status bar will alert users when an action takes place. These objects are placed from most important to least important, according to a traditional left to right workflow. This can be better seen below in figure 1.

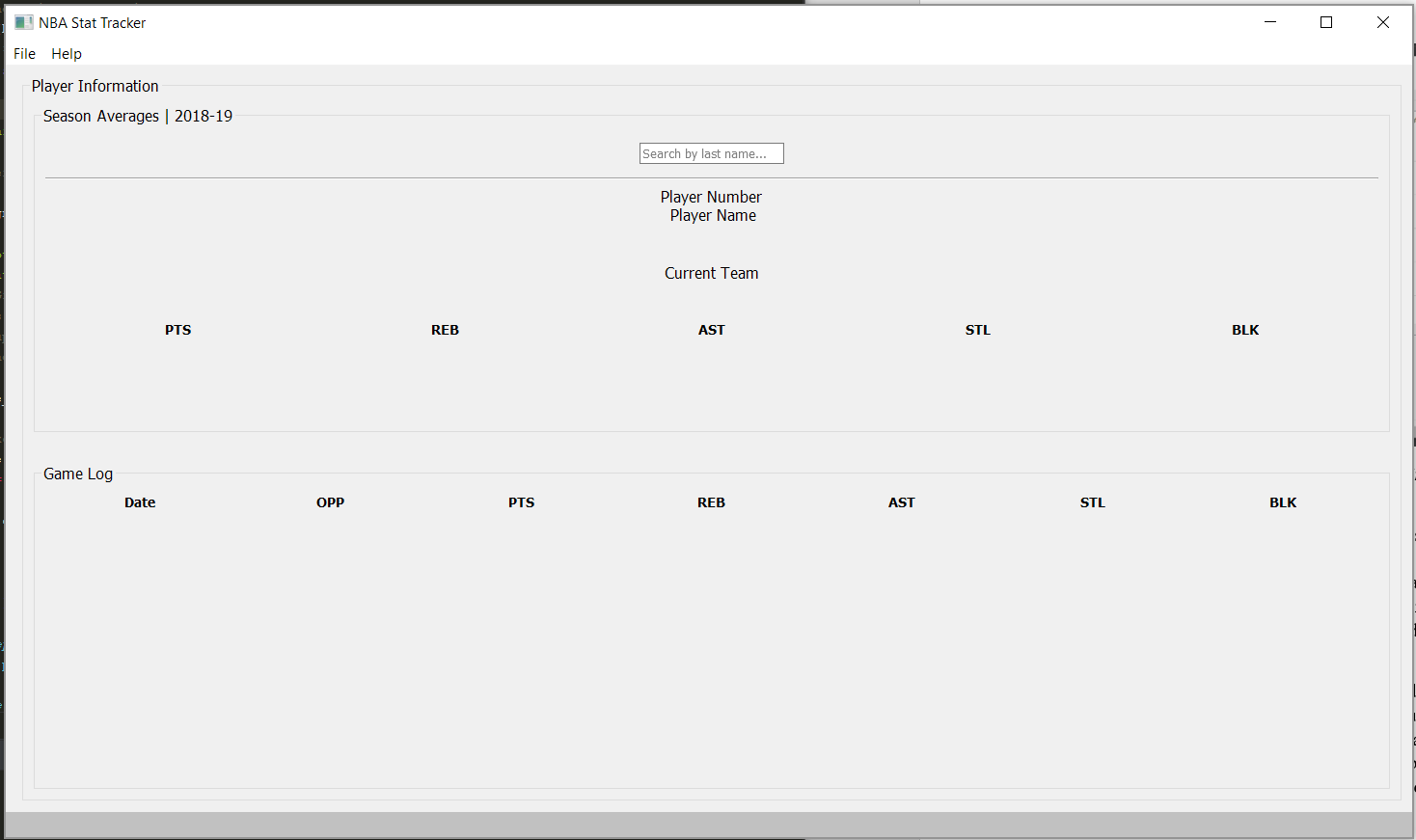


*Figure 1 above showing the main GUI design.*

The main workflow allows players to see the draft table covering the most real-estate, while the Player Information tab and list of games below the table take up less real-estate. Users can change the real-estate taken up by increasing or decreasing the widths by clicking on the divide line between the two sub-windows and dragging either left or right. Users can also completely remove the draft table object or player information object by dragging it “off” the screen. Once removed, it can be added again by dragging in the opposite direction. Figure 2/3 demonstrates this.

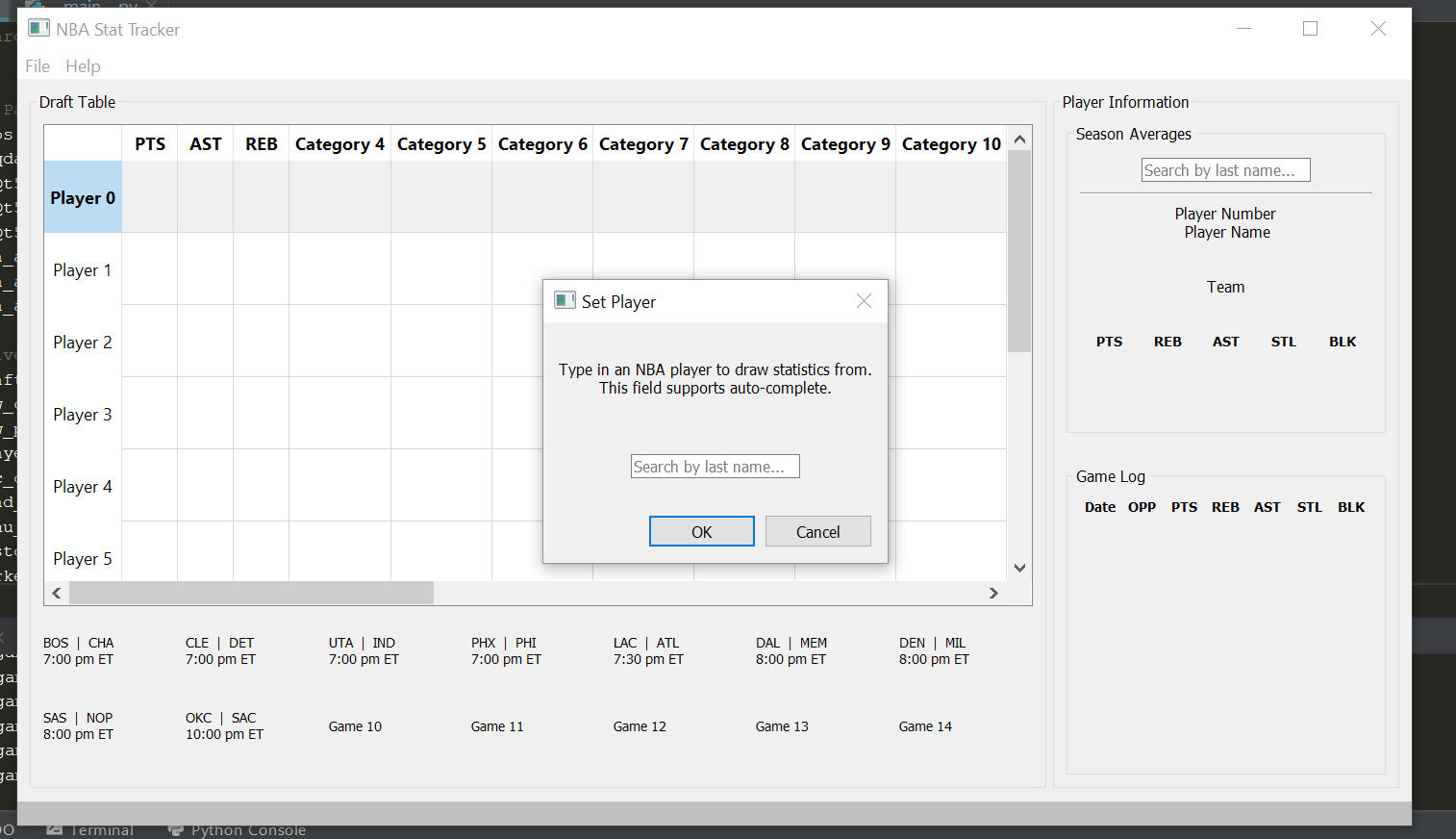


*Figure 2 showing the draft table exclusively.*

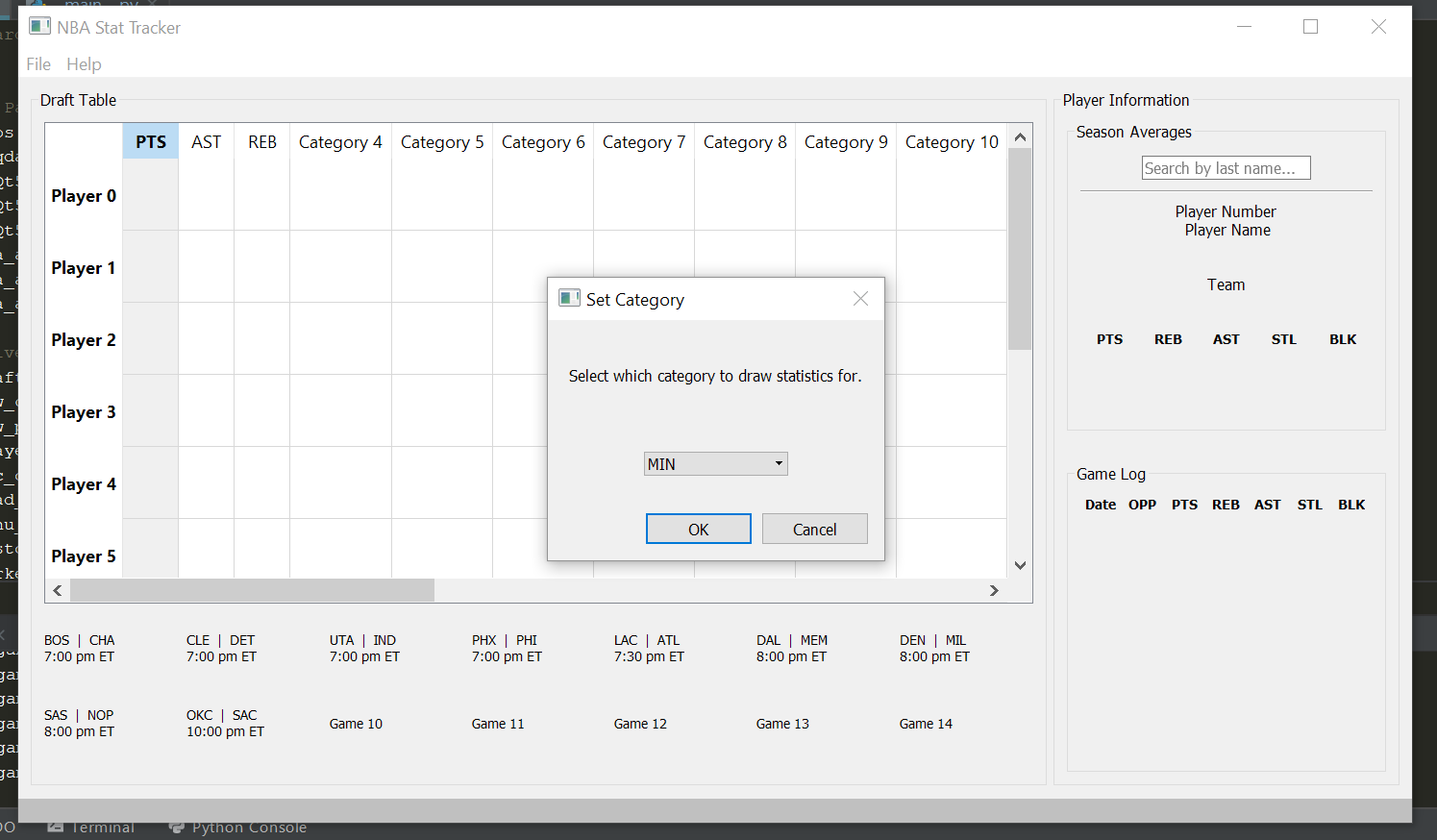
**

*Figure 3 showing the player information exclusively.*

As seen on the left, the Draft Table tab includes a familiar cell like interface with players on the left and categories on the top. Users must double-click on the header cell, such as “Player 0” or “Category 4” to open up a dialog. The Dialog allows users to pick and choose which player or category they want, respectively. These two dialogs can be seen in figure 4 and figure 5. As players and categories are inputted into these headers, information will automatically update.

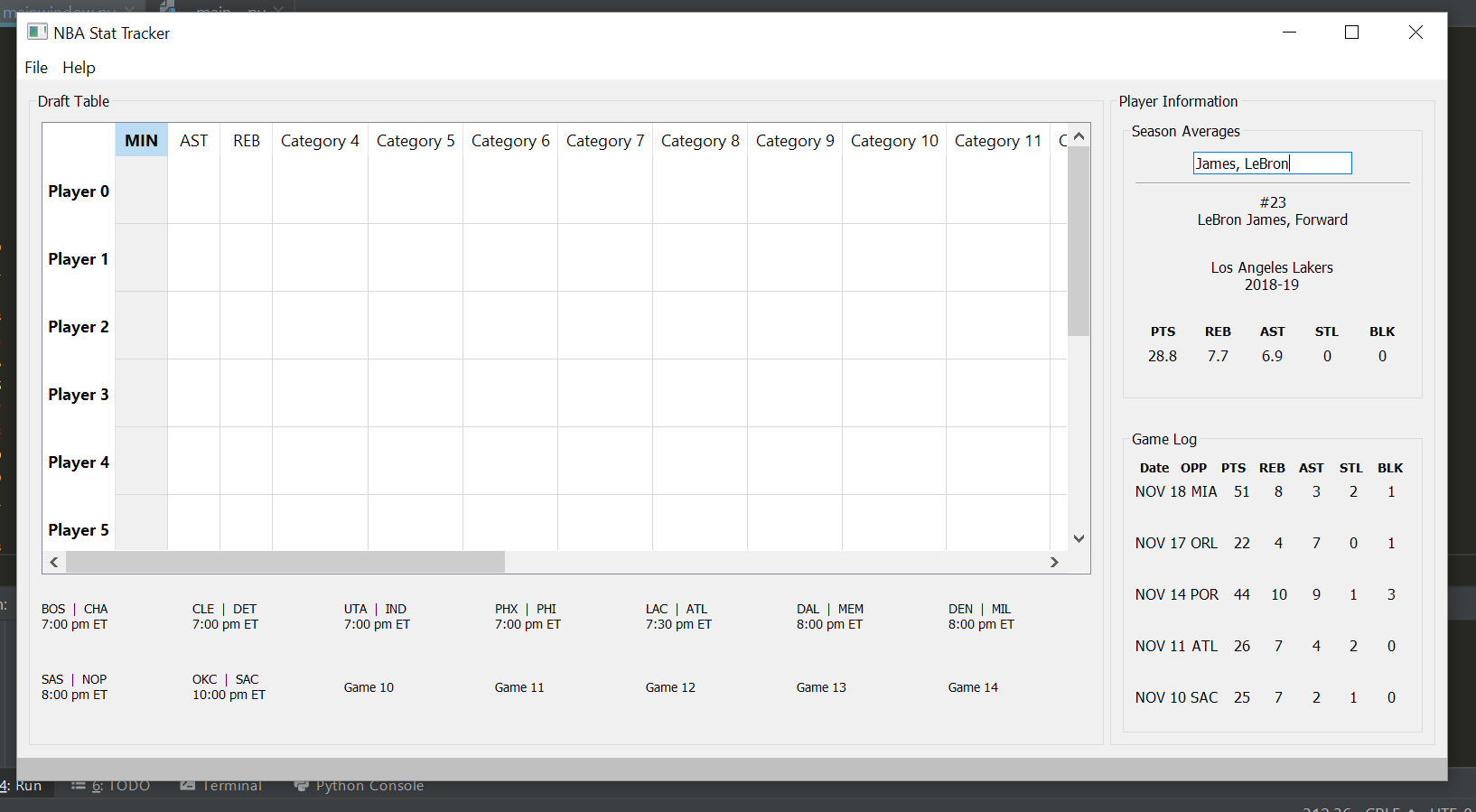


*Figure 4 above showing player entry with auto-complete.*



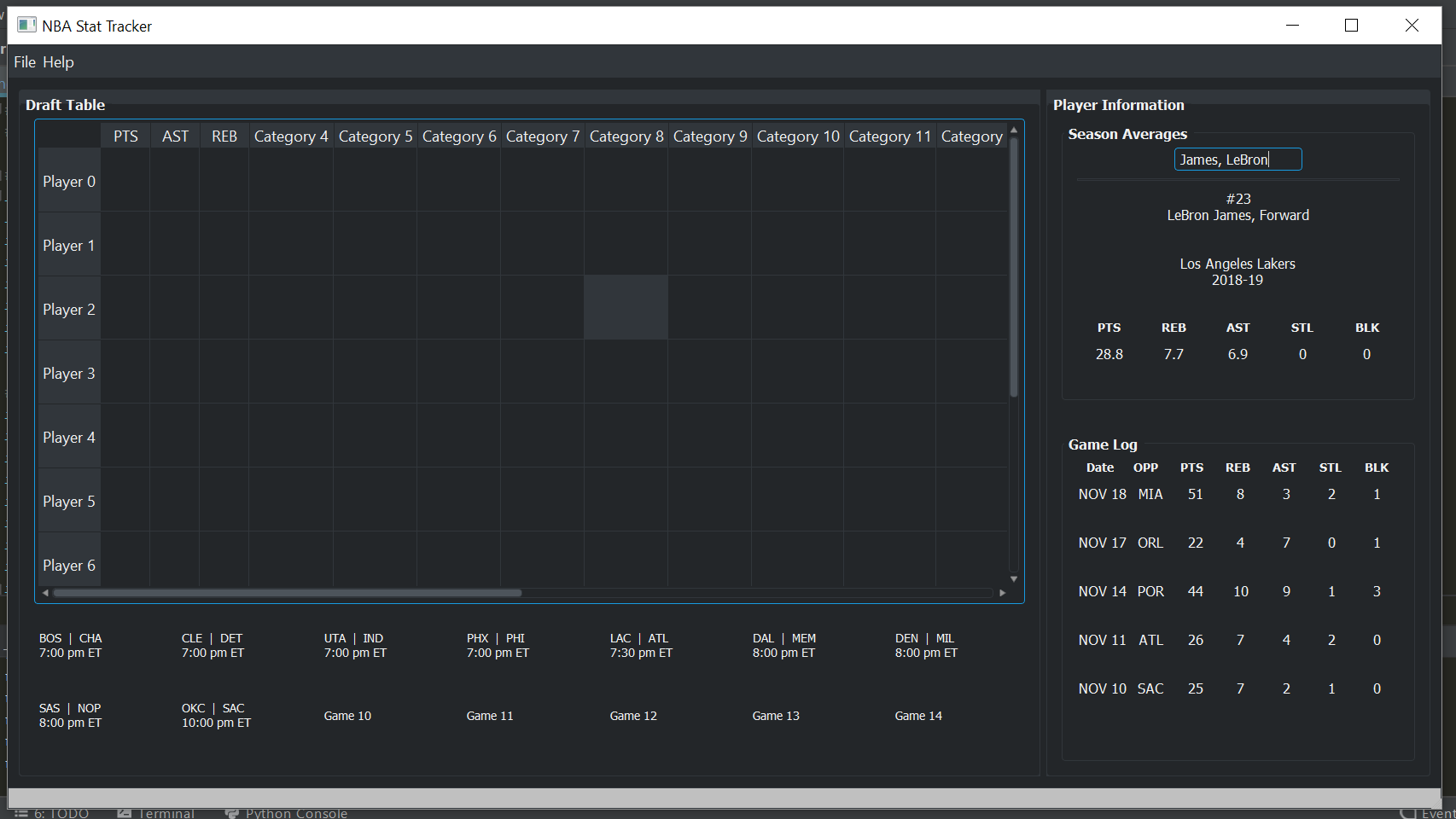
*Figure 5 above showing category entry with a pick-list option.*

The next aspect of the main window that users are presented with is the Player Information tab. This tab allows players to search for players on the fly without needing to input a player into their table. The purpose of this player information tab is to give users freedom from the draft table in two ways. The first way is that players can isolate that single player and see how their season is going. They can compare season averages to a players’ last few games. The second way users can take advantage of this is so they can see either simple or advanced data on them. Although only 5 statistics are shown below a player (points, rebounds, assists, steals, blocks) – a user can “pop out” the window to see much more advanced statistics like a shot-chart, how well they did during a game with an efficiency rating, injury report, etc. This can be seen in Figure 6.



*Figure 6 above showing a global search for a player.*

The user interface design choices were made to keep users in control when they want to see advanced things but keep a simple design. The font selection is clear and clean with variable font and bolded text to highlight different areas of the GUI. Users also have an option to change the theme and the font on the fly. In File 🡪 Options, users can change the theme to a dark or light theme, as well as increase or decrease the global font scaler. A demonstration of this is shown below in figure 7.



*Figure 7 above showing a dark theme as an option for the user.*

The next aspect of the application is its ability to retrieve data and display it concisely that makes sense to the user. The information is gathered from a public API which the NBA has created and hold the rights to. This endpoint is stats.nba.com and it is used extensively in the application. In order to achieve the speed that the user expects and without the GUI freezing during these network calls, each request is made in a separate thread and the data is returned to the GUI.

PyQt offers their own threads and workers, so incorporating it into the GUI was seamless. First, a worker object is created in the background that is to go and retrieve data from the endpoint. This handshake will take place and the data returned to the worker is independent of the GUI itself. In order to connect them to get the information safely back to the main GUI is to connect them using signals and slots. The worker is controlled through two signals, a start and finished signal which emit during its lifetime. The finished signal is the important one, and when it fires the GUI automatically grabs ahold of this worker’s data to guarantee its safe and won’t be lost. From the front-end perspective, the information is just magically handed to it during different interval periods and the user is notified by use of the status bar that a request came back good or bad.

As for peer recognition all the feedback came from my roommate. Together we dreamed of this application coming to life and this semester I was able to make it happen. My roommate made a lot of suggestions, the larger ones being addition of a game log and the ability to sort (I will finish this one day), and even deciding between single-clicks and right-clicks for dialog access. All of which became very important in the final design.

**Issues**

Although there were many highlights in the application, there were a few things that are still in the works and have proved to be quite difficult. A simple function for a user to have is the ability to click on a category to change the sorting of the numbers in the column. Let’s suppose users want to sort by the first category, points. The use case of this is to sort all the players loaded into the GUI to see who has the highest amount of points scored, be it for the season as an average or who scored the most points in a night.

Assume that the headers Player 0, Player 1, Player 3, etc. have loaded players. Initially they will be out of order, but a single click on the header “PTS” (points) should sort them in a descending order, and then ascending on the second click. This was actually very difficult to perform, as a user may not have sequentially filled users, i.e. a user may have first entered a player into the “Player 0” slot, then a player in the “Player 4” slot. The notion here is that the GUI should re-order the two players, put them in the first and second slot numerically correct.

However, with the single click action reserved for a sort and a double click reserved for another action (entry) the application had an extremely hard time differentiating between single clicks happening every short interval versus an actual double-click. The way I thought to solve this problem was to create an algorithm that would first store all the information within the draft table in the correct order, re-order the data stored how the user requested by the correct category (players and their data would be re-ordered according to which value was higher on a per player basis), then wipe the draft table, re-insert the sorted data and back fill the table with default information that the user has not changed.

Although this worked in theory, the draft table started to behavior very unpredictably. The column that the user requested to change a category for would reflect in the incorrect one, and single clicking would no longer sort. Although the logic was developed to implement the custom sort feature it proved to be rather very difficult to achieve and the undesirable behavior from testing it made the user experience poor. Who would have thought that such a simple idea would turn into a complex mess?

**Testing**

When it came to testing, this process was straight forward. My roommates and I would use the application over time and think of what is important to a user. We followed a simple approach of analyzing the application in its current state and deciding what worked and what didn’t, all while considering the end goal and main requirements. Each major feature that was added would be used in common cases and corner cases until it was refined and worked without failure. As we would test, more ideas would surface for further implementation. This process would repeat, and even sometimes a feature would be removed in favor of another. This type of refinement worked very well for this application and environment; over-time, the application became very usable for the target audience and was even shared amongst others online for feedback and issues. This refinement process ended up getting me to the end goal of a tested application, with many major and minor revisions that became more stable for my users.

**VII. Conclusion and Remarks**

I think the most important aspect to user interface, and something I personally struggle with, is removing myself as a developer and always considering the end user. As someone who loves to program and be creative through the ability to program and understanding of GUI frameworks, its not really what I think is going to be “cool” or an interesting feature – the product is for users and consumers. What I learned from this class that can help me reinforce this is the golden rules of user interface design. This is important to me because it allows my creative interests to continue to create things for users but enables my designs to always follow rules that are user centered.

The other very important thing I learned from this class is how wide-spread the user interface extends and the how important the **experience** is for them. If a user uses a bank ATM, they certainly do not want to be confused but want to feel excited to use it. For medical professionals the user experience needs to be easy to learn but also complex as it will probably serve a wide range of features. Before this class, I only considered user interface to be the user interface, not the experience along with it. I learned that users have emotions, likes and dislikes, and even accessibility discrepancies that all must be tailored to when designing. The experience ultimately decides how well perceived the user interface will be, even if the designer thinks the user interface is as good as it will get. To me, this is important thing to keep in mind always.

Finally, as I submit the NBA Stat Tracker, I intend on putting more work in the application to develop more for the user. This includes things such as shot-chart, pop-out windows that give advanced information on players, and ability to hook to other online tools for even more features and information. The applications code lives in my GitHub here <https://github.com/xeacott/stat-tracker> and can always be cloned for interest or a pull-request can be made for additions.