CS 5150
March 16, 2018

Project Name: Basketball Statistics

Group Members:
Hyuckin David Lim (hdl39)
Kemal Hasan Atay (kha26)
Kevin Gregor - (krg43)
Kyle Brown - (kmb266)
Marcus Boeck-Chenevier - (mb2533)
Sarvar Dhillon - (sd763)
Yingjie (Natalie) Ding - (yd344)
Yiqi Yu - (yy757)

Milestone 2 Report

Report Objective: This report has four main objectives. The first is to review the project requirements as put forth by David Robertson (the client) in order to ensure that he and the group are in agreement with what constitutes success in this project. The second purpose is to review the preliminary user interface designed by the user interface mini-team and receive feedback on the design. During the presentation, the group received feedback from Professor Arms that the client must be more intimately involved in the user interface design process and due to this feedback the group has already begun soliciting feedback on the second iteration of the user interface. The third purpose for this report is to document the progress made by group including the creation of a database schema, the implementation of said schema, the design of a mock user interface and deciding on the technology that will be used to create the graphical user interface (GUI). The fourth and final objective of this report is to put forth the project time line that the group will follow in order to complete the third milestone. This includes a breakdown of tasks to

be completed by each of the mini teams, namely the user interface (UI) team and the database (DB) team.

Requirements: The following are the requirements according to the client that are necessary components of a successful software application for the Cornell Men's Basketball coaching staff and the group has deemed them feasible to complete. There was some back and forth with the client about what was possible due to time and technical constraints, and in the following list are the results of said negotiation along with an explanation of what each requirement means. The software application must...

- Show team and individual player statistics through a traditional boxscore. This is important so that the coaches can have an intuitive understanding of what this application will provide them. This requires understanding and creating a traditional boxscore.
- Show team and individual advanced statistics. This does not mean that the group will be creating its own statistical models for analyzing basketball performance, but rather, it will apply predefined, widely-accepted metrics and advanced statistical formulas to aid the coaches in analyzing team performance.
- Filter the above statistics based on a number of variables, including: time and score of a game; game location, whether it is being played at home, away, or at a neutral location; if the game resulted in a win or a loss.
- Handle different the units of the boxscore and advanced statistics. The statistical units must be available in “per 30 minute”, “per 40 minute”, “per 100 possessions”, and “per game”.

- Show team and player statistics based on different 2, 3, 4, and 5 man lineups. This is one of the main use cases of this application. The coaching staff will want to see which players have the most positive and negative impacts on the game. An example of this is being able to see the team statistics and player statistics when “Player X” and “Player Y” are on the court together. Another case the software must handle is if the coaches want to see the statistics when a specific player is not in the game. For example what are the team statistics if “Player X” and “Player Y” are on the court but “Player B” is not.
- Show team and individual boxscores and advanced statistics of all other NCAA men’s basketball teams.
- Be maintainable after the group has moved on from CS 5150 and is unavailable to offer technical support. This maintainability is one of the most important features of the software as the coaching staff will be using this to analyze team data next year when the group has been dissolved.

One important note is that the application will not be able to perform lineup filtering on teams other than Cornell. This is due to constraints in the data the group is able to gather. The client expressed interest in this feature, but it is not possible with the data currently available.

Respond to Feedback from Feasibility Study: In the interests of transparency, and to best accomplish our goal of creating a comprehensive, useful system according to the clients’ requirements and the development team’s available time, we would like to reflect on some earlier concerns about the project’s feasibility. Primarily, we would like to clarify that certain aspects of the project are already complete or were provided to us in very early stages of

development: the Client has provided a list of statistical models (along with their underlying formulas) that they would like implemented for advanced statistics, and the ESPN JSON parser had already been written prior to the beginning of the project - as such, two large components of the project are already complete. Further, we have made substantial progress towards some baseline backend and front-end systems (which we further outline below), which makes us confident that we can fully develop a system according to the requirements developed so far.

Project Scope

According to the requirements of our client, our team reached an agreement with the scope of the project after careful discussion. Basically, there are four main tasks we are going to finish within the scope. First is the data visualization. As required, the application should be able to clearly present the box score stats per player and per team to the users. Secondly, advanced data is to be calculated and displayed. Based on the basic data we get, we should build a model to compute the advanced data and display it on the user interface as wanted. Luckily, there are already many models available, so this would not be a problem. The third is the functional filters. For some time, the users want to see the stats in a specific period within a game rather than a whole game. Also, cases are that the users want to look up the stats of the home games or just last five or ten games. Thus, the application should have adequate filters to present wanted stats to the users. This is the most important part of the project. Lastly, the application would provide a lineup function to the users so that the users could see which lineup on court in a specific time have a better performance. As a result, they can make the adjustments in lineups reasonably. One point to mention is that the lineups are developed only for Cornell basketball team.

Due to the time concern and lack of the detailed stats, we probably would not cover these tasks in our project although they were brought up by the client in the premium features. First, since we have find the appropriate model to compute the advanced stats, we do need to invent new formulas to analyze basketball team and player statistics. Also, we won't implement the video editing and analysis software. What's more, the lineup function would only cover Cornell basketball team instead of each team in NCAA. Lastly, because we do not have the coordination information of the player's shot. It would be reasonable that we are not going to implement the short chart or shot location data representation.

Because we did not provide enough details in the feasibility plan, there may have been some misunderstandings about the scope of the whole project, which gave Professor an impression that this is a much bigger project than the time allows. However, since we have finished the two main parts of the project that we have got the basic data and an appropriate model, the team thinks that the project is workable on the whole.

Database Team Updates

Database Team – Progress:

First of all, we have found and utilized a public/free library online for scraping data from ESPN. It gathers detailed data for each game in the league: play-by-plays, box scores and other types of data that we will use in our project. After making a few modifications on the code, we have downloaded data files for over 6000 games.

We then analyzed these files in order to understand how ESPN formats and uses data. We filtered out which parts are necessary and which parts we can eliminate to increase efficiency of the database that we were going to create. Every one of these files contained excessive information that we will filter out later when the data is being transferred to the database. The rest of the data will be converted into the schema we have created before being stored.

We also started creating an XML parser to parse and convert data files for Cornell games. The data files we have downloaded from ESPN didn't have data for substitutions which we decided to use in our database schema, so we decided to use separate files that represent games played by Cornell submitted by our clients. After analyzing these files like we did the game files from ESPN, we begun coding the XML parser. The current parser we have can parse and convert the files, but it is not completed yet. There are a few changes that we need to make along with some improvements.

After reading through the XML and JSON files to see what data were available, we designed an entity-relationship diagram to represent how to store the relevant data in our database.

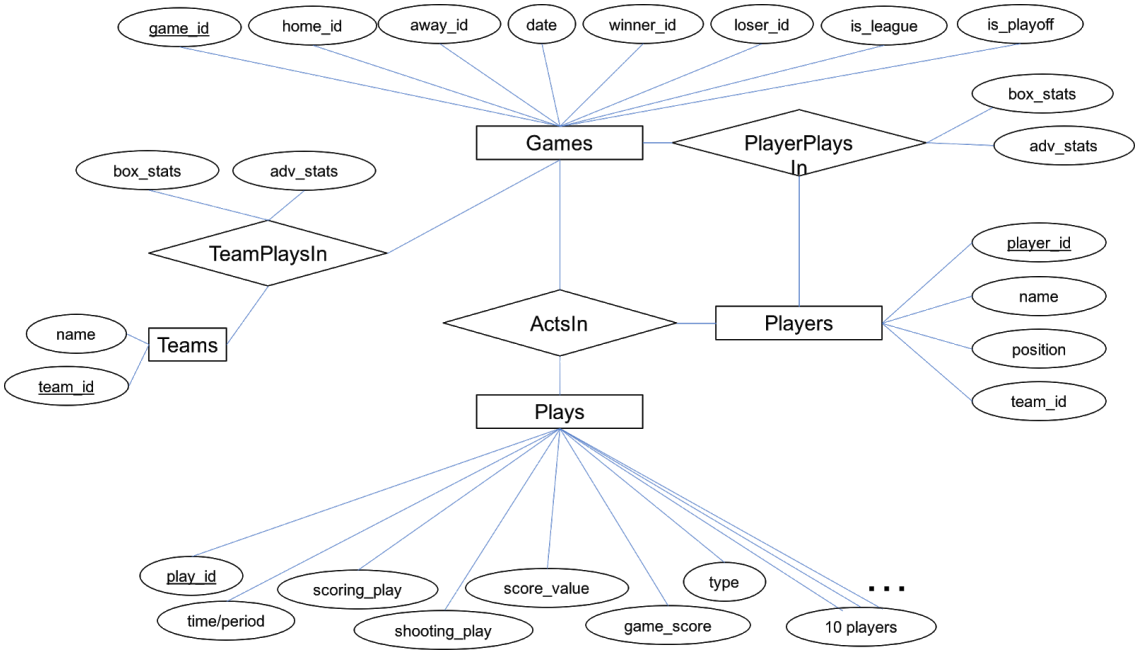

Figure 1

The entities in Figure 1 are Teams, Players, Games, and Plays. Each entity stores relevant attributes about itself, e.g. Players stores player_id (a unique identifier), name, position, and team_id. This allows our relational database management system (tentatively SQLite[1]) to efficiently filter all games that were played at home, or all plays that involved a made shot for example. The relationships in Figure 1, represented in diamonds, describe the relationships between the various entities. For example, when a player plays in a game, we can store that player's box and advanced statistics for that game.

One design decision we made was to include all ten players that are on the court for every play (this is available for Cornell data only). We needed a way to store lineup information, and

[1] We are choosing SQLite for our early development stages because it is lightweight (both in terms of memory and storage needs), and so we can create and tear down testing databases as needed. We can easily transition to another SQL-based database because we are leveraging Python SQLalchemy, which will convert our high-level Python database declarations to SQL.

found this option to be efficient with regards to both space and speed of filtering, given our plan to leverage Python's SQLAlchemy library for at least part of the filtering.

We believe that implementing this entity-relationship diagram will allow our database system to capture all the requirements discussed with the client. However, any modifications we may need to make in the future can easily be handled by appending or deleting attributes from the various tables. We have already instantiated the tables, and are in the process of developing a parser to populate the database with the relevant data from the XML and JSON files.

Database Team – Next Steps:

As mentioned above, we currently have a base XML parser (that is not yet robust enough to handle data such as OT, and has not yet been built to handle data files that are broken or missing some information). The XML parser converts the XML data to a python dictionary - we plan on having the JSON parser create a dictionary with the exact same format, except without lineup data (as this is only available in the XML parser). We have also created a database using the schema discussed above - more specifically, we use Python SQLAlchemy's ORM Model[2] to map from Python classes to SQL data tables, which allows us to programmatically generate and add data as it is parsed. We have begun preliminary testing writing, querying, and reading to the database using the SQLAlchemy wrapper. The next steps are to make the XML parser more flexible, create the JSON parser, design more robust unit tests to try and discover flaws in our schema, programmatically combine the parser and SQLAlchemy code to populate the database when a data file is parsed, and create a database with dummy data for the UI team to experiment

[2] Specification available here: http://docs.sqlalchemy.org/en/latest/orm/tutorial.html

with. Projected timelines for these tasks are discussed below. After these tasks are complete, we will begin writing queries that will serve as the backend processes for when the user applies a filter on the front-end.

User Interface Team Updates

User Interface Team – Progress:

For an application such as the one we're creating, usability is crucial. For this reason we had a majority of the team work to design an optimal user interface to fit the technical requirements while still being clean and usable.

To maximize the number of original ideas for a user interface, the UI team decided that every individual member of the team was to create their own preliminary mock-up based on the requirements sought out by the client. The following step was for all the members of the UI team to meet and evaluate the different UI mockups presented. These mockups were evaluated internally because we wanted to create a solid preliminary UI to present the client. In the team's efforts to put together the optimal UI we took the strongest elements/aspects of each individual mockup to create the next iteration. What the team then came up with was a general page layout with a vertical filter tab on the left of the page and the box score/advanced stats presented on the right. Furthermore, a three page structure was adopted, consisting of a "Players" page, a "Teams" page, and a "Games" page.

The "Players" page displays statistical breakdowns per player for a given team (Figure 2.1), whereas the "Teams" page would show a side-by-side of the statistics of different teams (Figure 2.2). Finally, the "Games" page would allow the user to focus on the statistical breakdown per game (Figure 2.3). For demonstration and testing purposes a semi-functional user interface was put together using the Adobe Xd. So far the testing and evaluation of the UI has been kept internal.

Figure 2: User Interface Prototype

Figure 2.1: Player stats

Figure 2.2: Team stats

Figure 2.3: Game stats

User Interface Implementation – Next Steps:

For our next steps, we will continuously revise our designs based on the feedback we have received and will receive over the next couple of weeks. Based on our user interface mockup and immediate feedback from the client, we will begin to write code which can make the prototype a real tool/platform.

The UI sub-team has two main goals moving forward:

Our first goal is to actually build a usable user interface for the application. We plan to use Electron - a framework that allows us to create a desktop application using web languages like JavaScript, HTML, and CSS. We will also continue to use Adobe XD to generate prototypes with clickable icons and show it to the client go gain feedback so that we can adjust our code accordingly. Based on our current design, we have three main pages of the application: Teams, Players and Games. The Teams and Players sections will be nearly identical, featuring the same statistics in both pages; however, the Teams tab will display statistics on a per-team basis while the Players tab will be on a per-player basis. The Games tab, on the other hand, will show box scores and other statistics associated with entire games. We will design each section according to different logic flows so that the clients are able to intuitively and easily access the data and filters. At the same time, these three pages should link to each other. For example, when the client is looking at box scores in the Game section, he will be able to easily see statistics for each player in that game. If the client wishes to see more details about the players, they can click on a player's name and be redirected to the Players page with the correct filters for that game and player already in effect. We will also implement a "compare" function, which will allow the

client to have several tabs of the application open so they can easily compare different statistical scenarios. Our team plans to complete this goal by March 23rd.

Our second goal, which will come after we have created the user interface in Electron, is to link our front-end and back-end. We will connect our database to the user interface, so that the application can continuously update and display data while responding to any filter requests. This step requires both the completion of the database setup by the database team and the completion of the user interface skeleton by the UI team. We plan to complete this integration step by April 9th, just before the next presentation and report.

Once we finish the integration, we will again perform several tests to ensure that the application is intuitive, the flow of data is smooth and correct, and the data is retrieved efficiently. And of course, we will give the application to our client(s) to test and give feedback along the way, which we will use to make revisions. The testing process will start as soon as possible and will be a continuous process until we hand over the final application.

Revised Schedule:

The Gantt Chart below shows our revised schedule for the rest of the project. By presentation 2, we hope to have the user interface fully completed and connected to the database. After presentation 2, we plan to work with the client to continue testing our application and adding more advanced filtering scenarios. The green bars represent work to be done by the UI team, while the blue bars represent work to be done by the database team, and the orange bar represents work that should be done by both teams together.