Bucket Buddy
Software Design Document

Rafael Braga

Robert Howe

Jason Nance

3/10/2015

Revision History			
Date	**Version**	**Author**	**Change Description**
1/22/2015	0.1	Jason Nance	Initial skeleton, project description section
1/29/2015	0.2	Robert Howe	Added activity diagram and initial class diagram
2/1/2015	0.3	Jason Nance	Added some use cases and activity diagrams with descriptions of each
2/1/2015	0.4	Rafael Braga	Added the second class diagram and description of some use cases
2/1/2015	0.5	Robert Howe	Added roster use case, roster activity diagram, and descriptions
2/4/2015	0.6	Jason Nance	Added definitions of terms
2/4/2015	0.7	Rafael Braga	Added purpose of project
3/7/2015	0.8	Jason Nance	Added detailed class diagram and description as well as some sequence diagrams and descriptions
3/8/2015	0.9	Robert Howe	Added a few sequence diagrams and descriptions
3/10/2015	1.0	Rafael Braga	Added a few sequence diagrams and descriptions

Table of Contents

Introduction

Purpose

The main motivation of this project is to build a smooth and efficient application that is pleasant to use. Sometimes you want to keep data from your favorite team and do not wish to use the archaic form of pen and paper. For this reason, we are proposing an easier and more organized new way of following teams. Applications focused in sports are common; however, they usually cost money. "Bucket Buddy" was idealized taking into consideration basketball fans and professionals in the area who want to keep personal data about their favorite teams, players and games but do not wish to spend money on it. Our goal is build an effective implementation which is good enough to equal paid applications in quality.

Scope

We will need to create multiple activities in our app that will provide an interface for the user to store data relevant to a non-professional basketball game. To that end, we will need to design and create several classes along with their attributes and methods. An efficient database will need to be implemented that stores all of the profile information and statistics in a logical and straightforward manner, making data retrieval easy.

Goals

The goal of this project will be to have an app that will enable the user to effectively and efficiently store information on basketball players and a team as a whole. The user will be able to retrieve that data in a similar effective manner. The storage and retrieval of the data will take place with an intuitive and sleek interface.

Definitions

Some definitions of terms useful for understanding the domain of Bucket Buddy follow. A full specification of the rules of basketball is beyond the scope of this document, but it will be useful to define some of the statistics and game information that a user can record.

Scoring - a shot that goes through the hoop (is “made”) from within the 3 point line awards the shooter’s team 2 points; a shot made from outside the 3 point line is worth 3 points. A made free throw is worth 1 point
Field Goals Attempted (FGA) - the number of shots attempted by a player or team
Field Goals Made (FGM) - the number of shots made by a player or team
3 Pointers Attempted (3PA) - the number of shots attempted from beyond the 3 point line by a player or team
3 Pointers Made (3PM) - the number of shots made from beyond the 3 point line by a player or team
Offensive Rebound - when a player retrieves the ball after a shot attempt on the opponent’s basket

Defensive Rebound - when a player retrieves the ball after a shot attempt on their team's basket
Assist - when a player passes to a player on their team, and that player proceeds to score
Turnover - when a player's actions cause the other team to gain possession of the ball via steal, offensive foul, out of bounds, or other event resulting in a change of possession
Steal - when a player takes possession of the ball from a player on the other team
Block - when a player makes contact with the ball during a shot attempt, causing it to miss
Foul - when a player commits one of several actions forbidden by the rules, such as making contact with an opponent while the opponent is shooting or pushing an opponent in an effort to get a rebound
Substitution - switching one or more of the 5 players on the court for one or more players sitting on the team's bench. Substitutions can only occur at specified times while the ball is dead
Timeout - when a team has possession of the ball, they can call a timeout to stop the game clock. Each team has a limited number of timeouts per half

Project Description

Overall Description

Bucket Buddy will consist of three main components. The first, the roster menu, will allow a user to create, modify, or delete a roster made up of players, which themselves can be created, modified, and deleted. This menu should be simple and easy to navigate. Secondly, the user should be able to select a roster and begin collecting statistics and game information for that roster during a game using an intuitive graphical interface. The interface will allow the user to undo or correct any mistaken input. These statistics will be stored in a database on the phone's hard drive for later viewing. The final component will be a display showing statistics already collected for a given roster or player, allowing the user to quickly assess the performance over time or per game of a team or individual. This display should be accessible from the roster menu and while recording statistics during a game.

Functional Requirements

1. The user must be able to create, modify, and delete a roster of players.
    a. The user must be able to create, modify, and delete players within rosters.
2. The user must be able to view statistics for each player on a given roster.
3. The user must be able to view statistics for a given roster as a whole.
4. The user must be able to choose a roster and begin recording statistics and game information in a game with that roster via a graphical interface.
    a. Statistics include (but are not limited to) points, field goals attempted/made (including their location), 3 point shots attempted/made, rebounds (offensive

and defensive), assists, turnovers, steals, blocks, fouls, and free throws attempted/made.

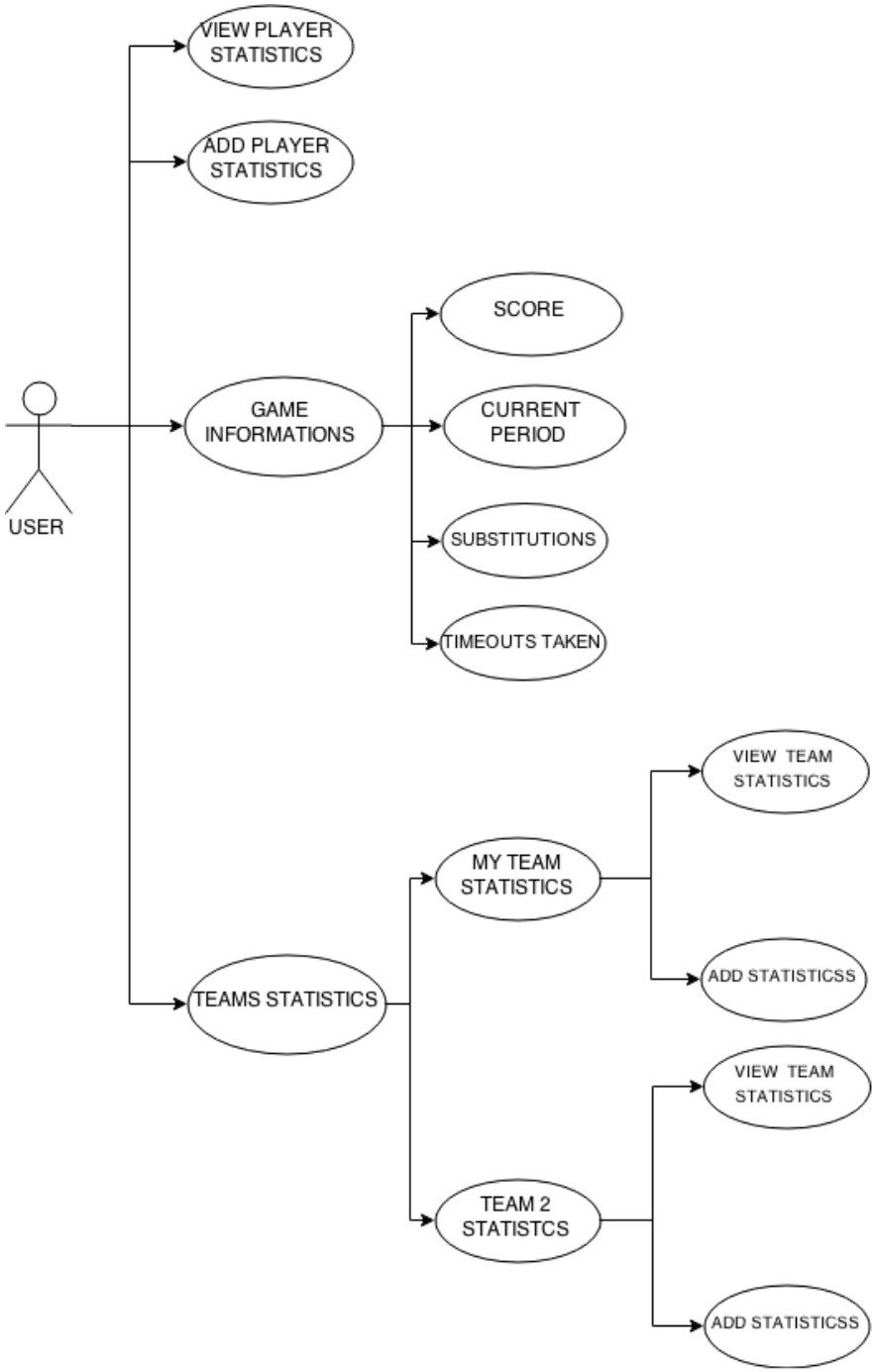
b. Game information includes (but is not limited to) substitutions, timeouts taken, current period (half or quarter), and the other team's score.
c. The user must be able to undo and/or correct a recent statistic entry in the event of a mistake.
d. The user must be able to view all aforementioned statistics from within the game interface.

Nonfunctional Requirements

1. The roster menu must be simple and easy to navigate.
2. The statistics collection graphical interface must be intuitive, aesthetically pleasing, and performant.
3. The app must not cause an unnecessarily large drain on the battery.
4. The app must store data efficiently and minimize the amount of hard drive usage.
5. The statistics reports must be useful and aesthetically pleasing.

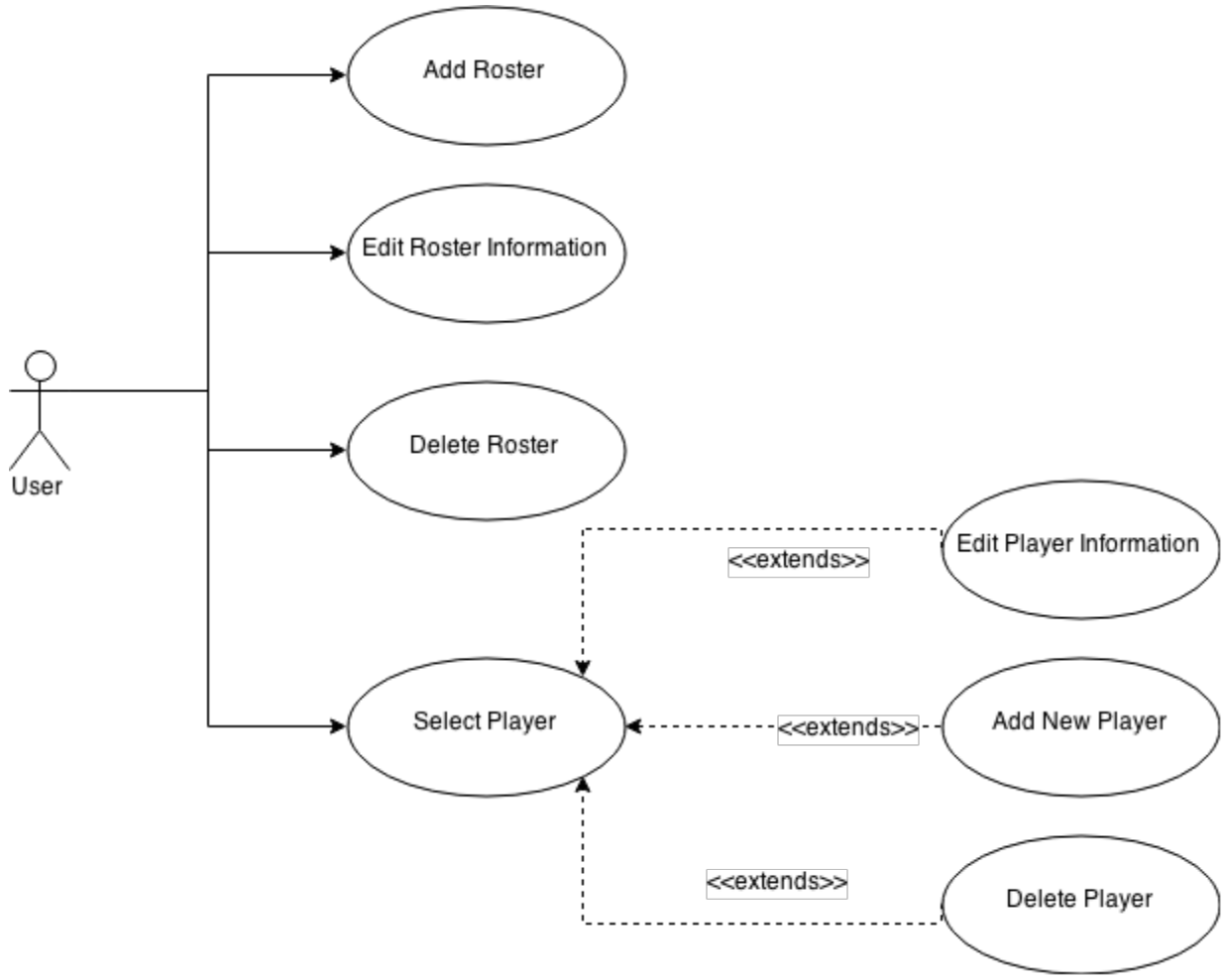
Diagrams

Use Case Diagrams



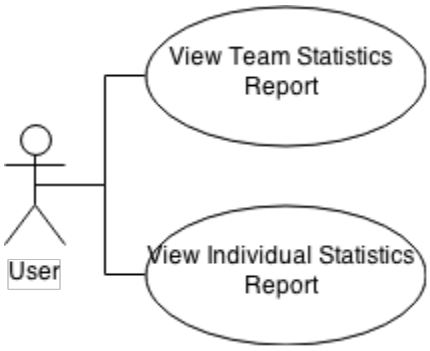
Use Cases - Game Screen

For the Game Screen view, the user will be able to view, add and edit information about the current game and players.



Use Cases - Roster Menu

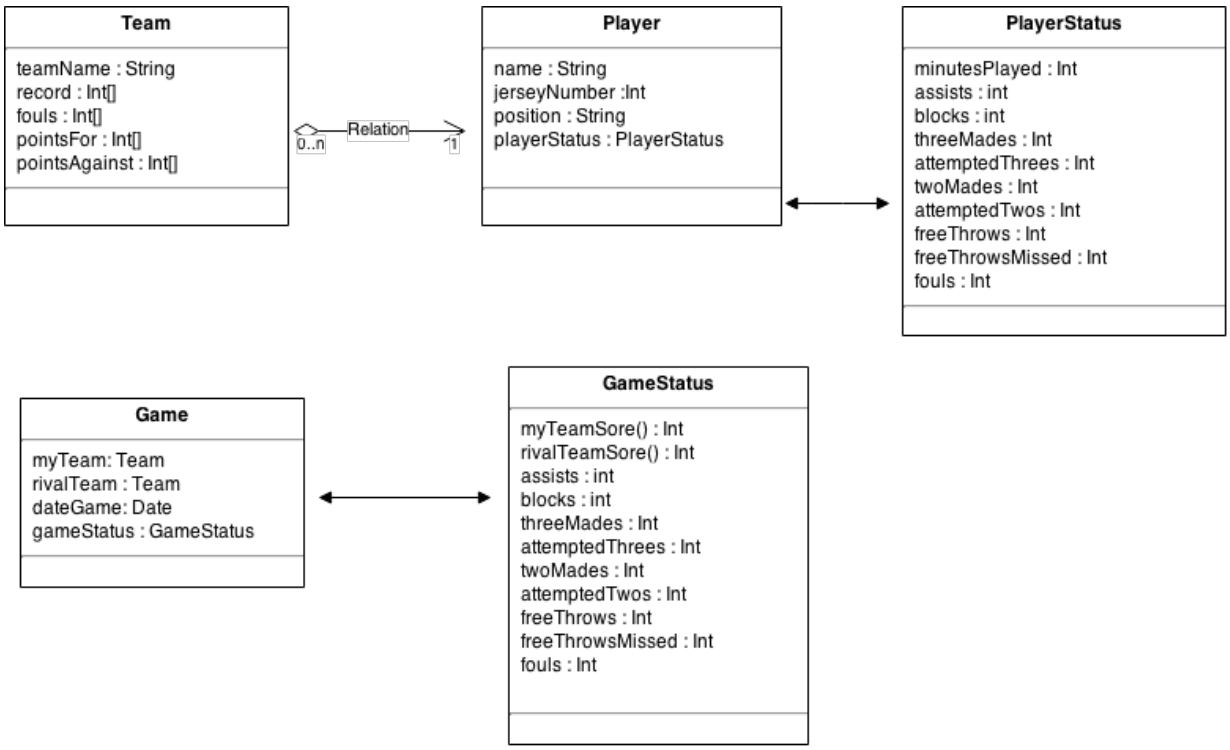
From the roster menu, the user can choose to add a new roster, edit a current rosters name, delete a current roster or choose an individual player in which they can edit that players personal information, delete that player, or add a new player.



Use Cases - View Statistics

From the view statistics screen, the user can choose to view a report of their team's statistics or statistics of an individual.

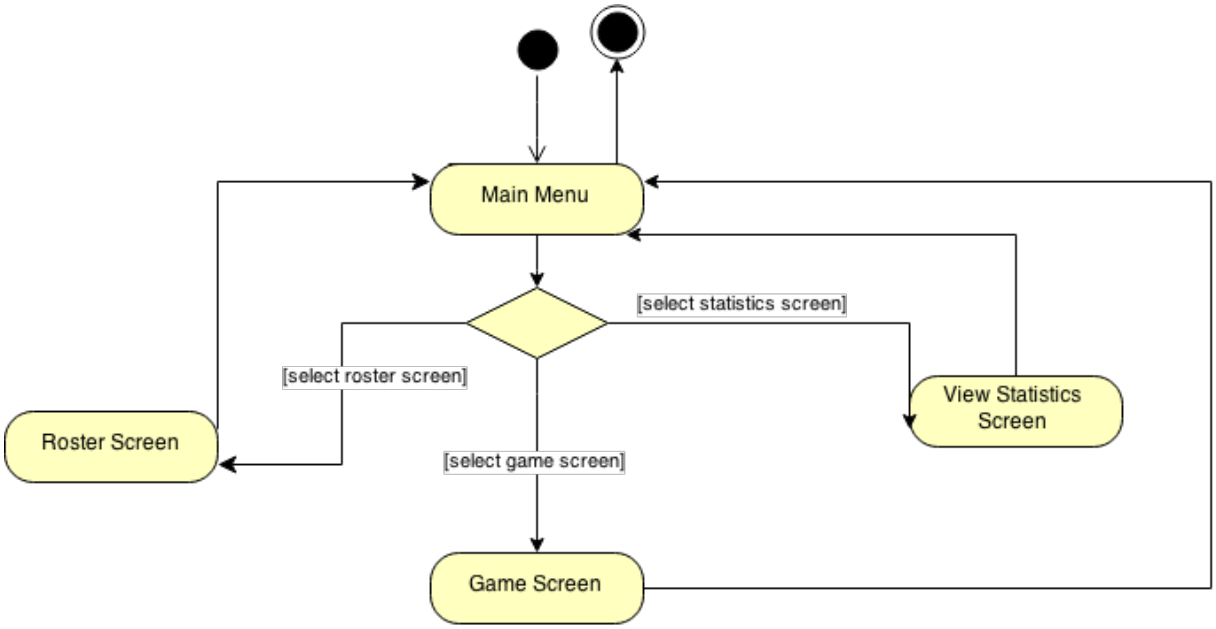
Class Diagram



Class Diagram

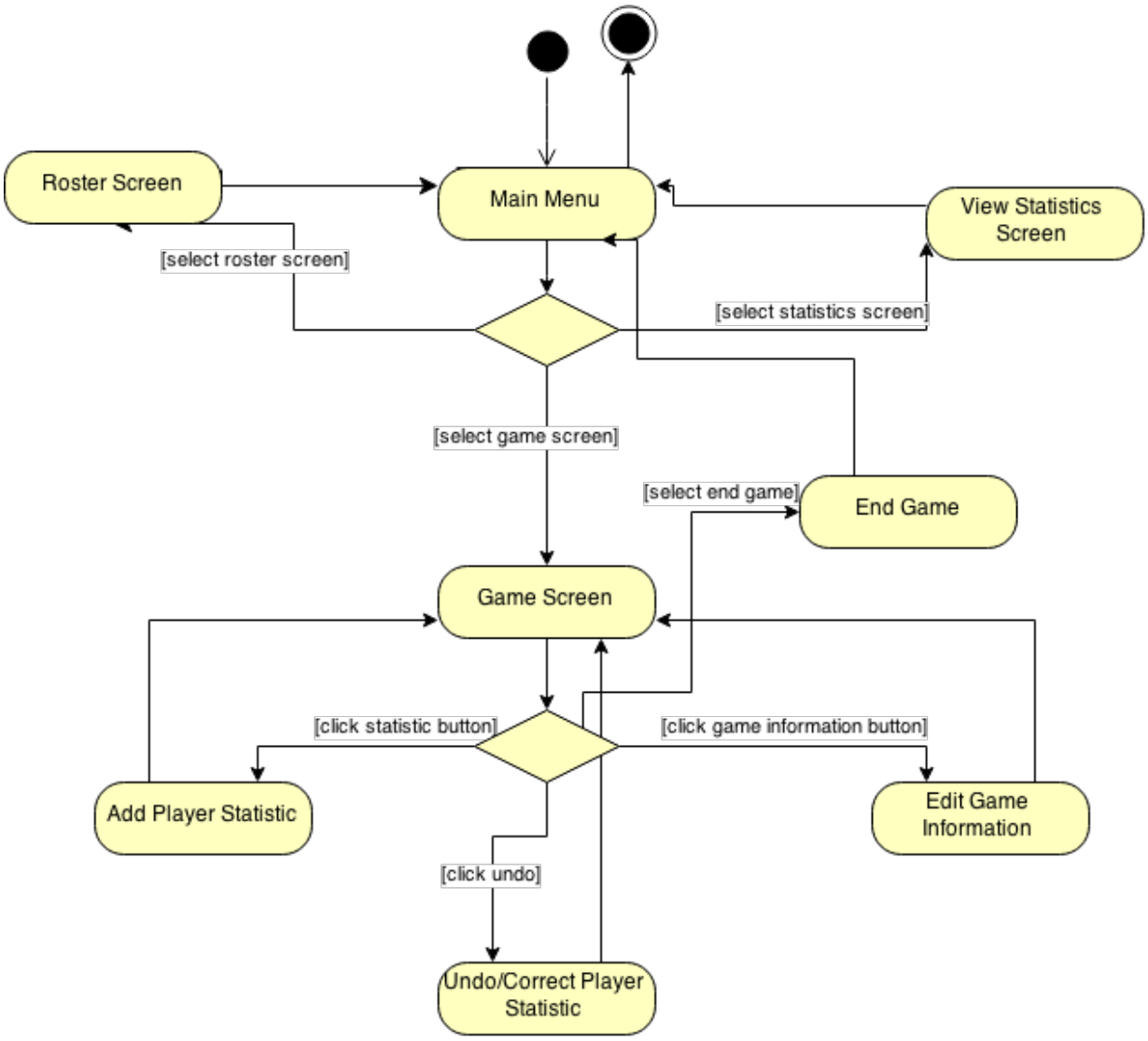
The class diagram describes the system, its attributes, and its methods, giving us a wide vision of the software and relation between the objects.

Activity Diagrams



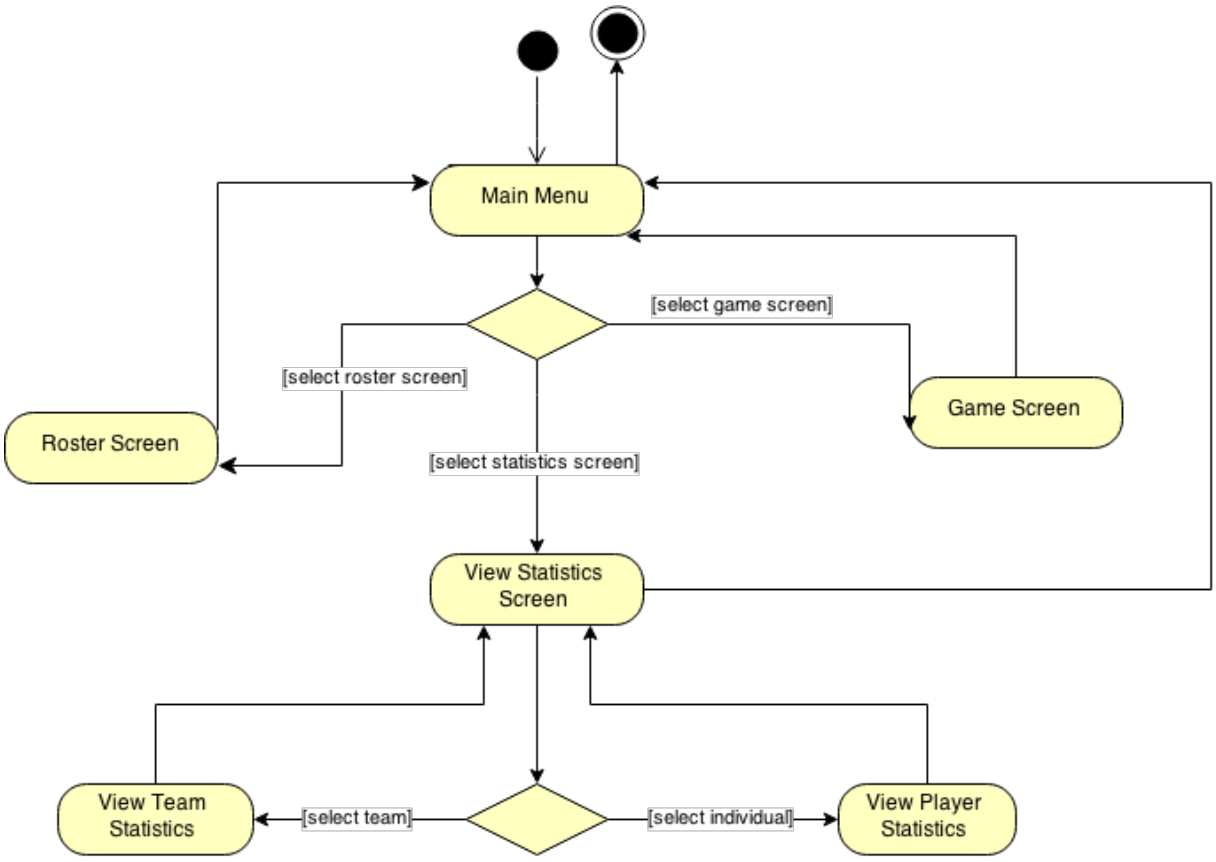
Activity Diagram - Main Menu

From the main menu, the user can choose to go to the roster screen, the game screen, or the view statistics screen. Once inside any of those screens, the user can optionally perform a number of actions before exiting back to the main menu and leaving the app.



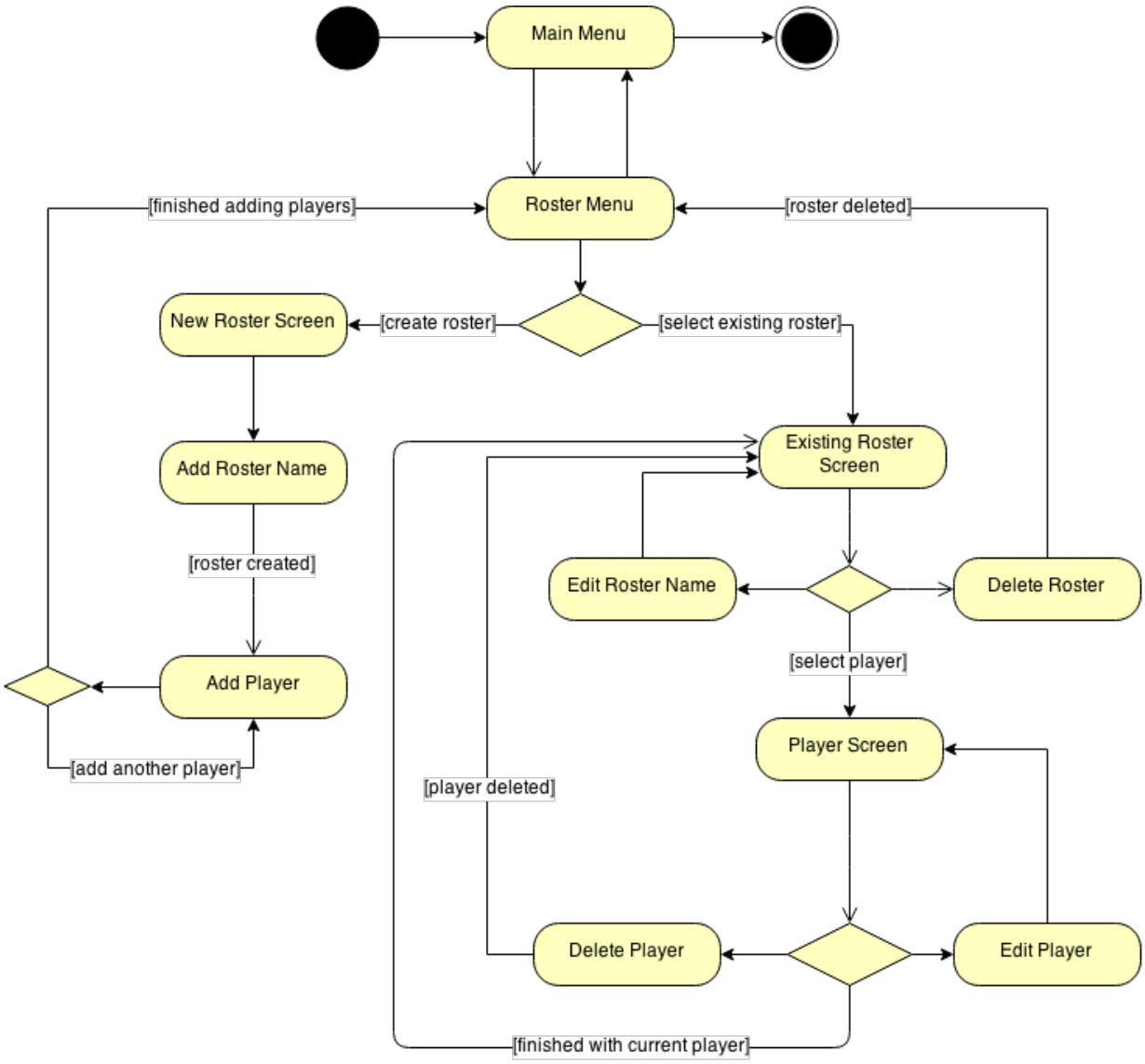
Activity Diagram - Game Screen

In the game screen, the user can perform several actions that will assist them in recording statistics. By clicking any one of the buttons designated for recording statistics, the user can add a player statistic. The user can also undo or correct a player statistic with the undo button should they make a mistake. Options available by pressing a game information button will allow the user to edit information about the game (as opposed to individual player statistics). Finally, once the game is over, the user can choose to end the game and will be returned to the main menu.



Activity Diagram - View Statistics

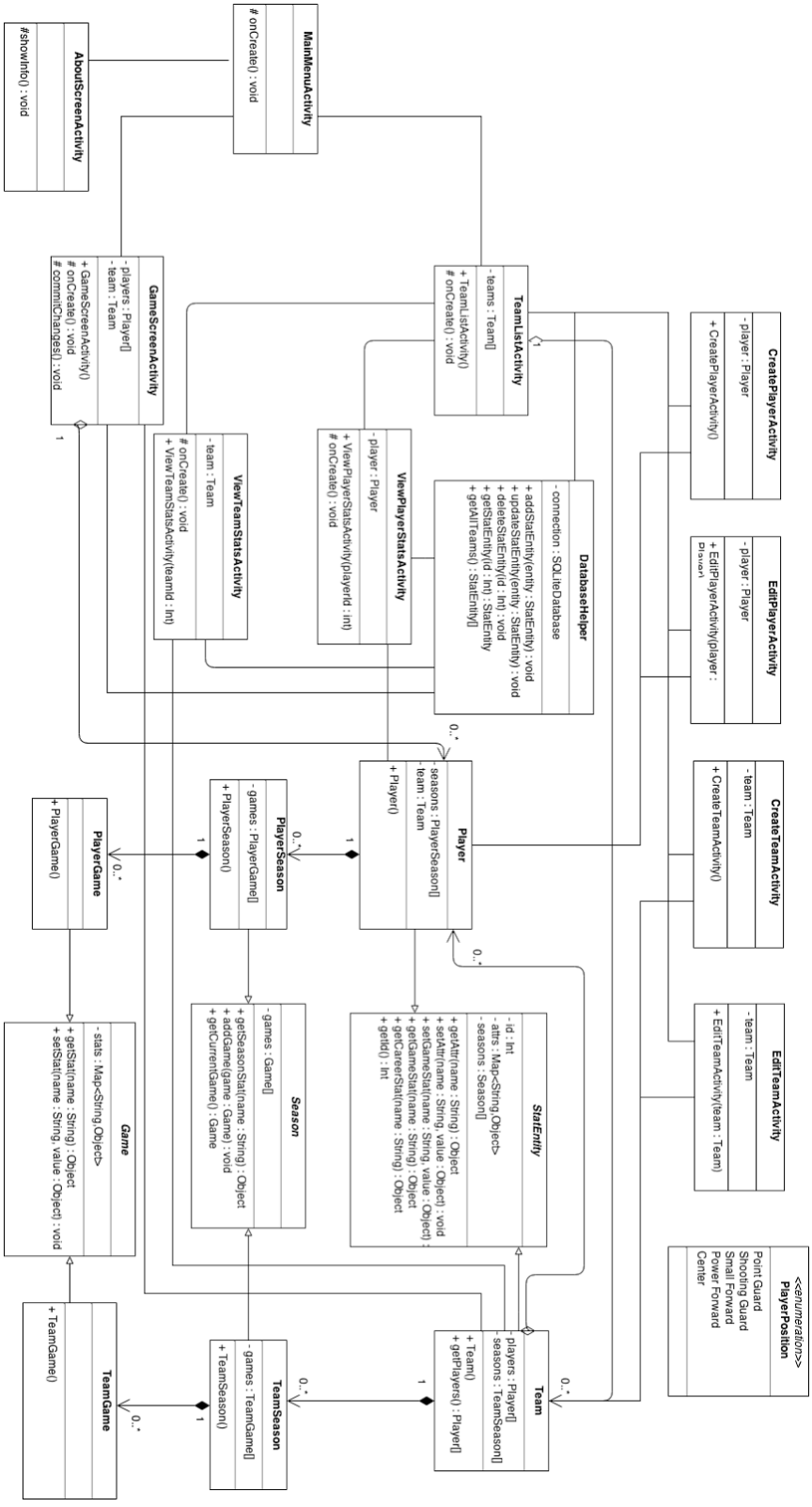
Upon reaching the view statistics screen, a user can choose a team -- allowing them to view statistics about the team -- or they can choose a player to see their individual statistics. Either of these choices will allow the user the option to return to the view statistics screen, from which they can view other statistics reports or return to the main menu.



Activity Diagram - Roster Menu

Upon reaching the roster menu screen, a user can choose an existing team (roster) or choose to create a new team. Choosing an existing team allows them to edit the team name, delete the team, or choose a specific player on that team, at which point they can edit that players personal information or delete that player. Choosing to create a new roster allows the user to create the team name then add as many players onto the roster as they see fit.

Design Class Diagram



The class diagram on the previous page gives a much more complete picture of the implementation of Bucket Buddy. On the left side and top of the diagram are the viewing components, such as the main menu, which leads to 3 other menus (a simple “about” menu, Team List, and Game Screen).

The Team List is associated with multiple activities for creating and editing/deleting players and teams as well as the DatabaseHelper, which will provide the interface into the SQLite database. The Teams List also allows a user to view statistics for the selected player or team, each of which require association with the DatabaseHelper and the player/team classes.

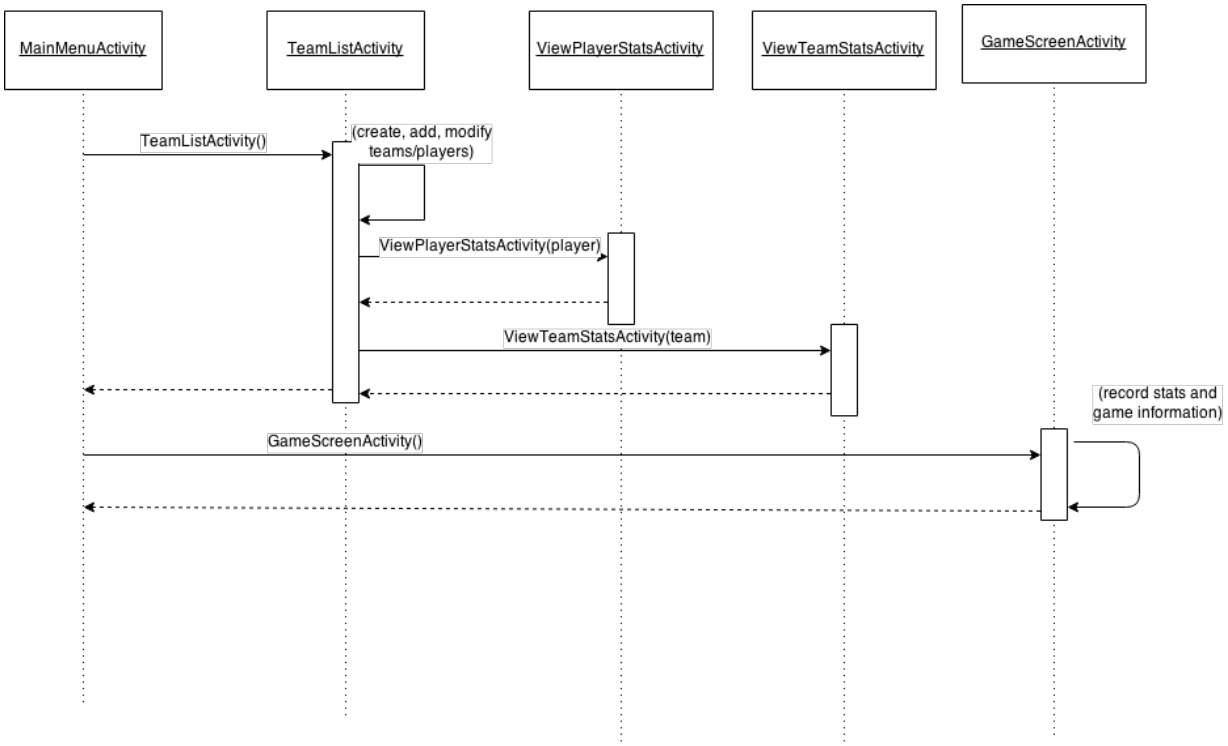
The Game Screen keeps track of players and teams, and it maintains an association with the DatabaseHelper to commit player/team statistics to the database after a game is over.

Towards the right of the diagram are classes intended to store data about teams and players in an intermediate format between the views and the database. StatEntity is a generalization of Players and Teams intended to cut down on code duplication; it provides a wrapper around “attrs”, which maps player or team attribute names to their values (such as “lastName”: “Smith”, or “teamName”: “Crimson Tide”).

StatEntities contain a list of Seasons (Player or Team). Seasons in turn contain Games (also Player or Team). Games also function as a wrapper over a map, this time mapping statistic names to values instead of attributes (ex. “points”: 8, or “timeouts”: 5). Games expose methods to manipulate and view statistics, and Seasons in turn allow client code to aggregate these statistics up to the season level. Season statistics can then be further aggregated to determine career statistics for a player or team. Seasons also provide access to the current game, which will be used when recording statistics in the game screen.

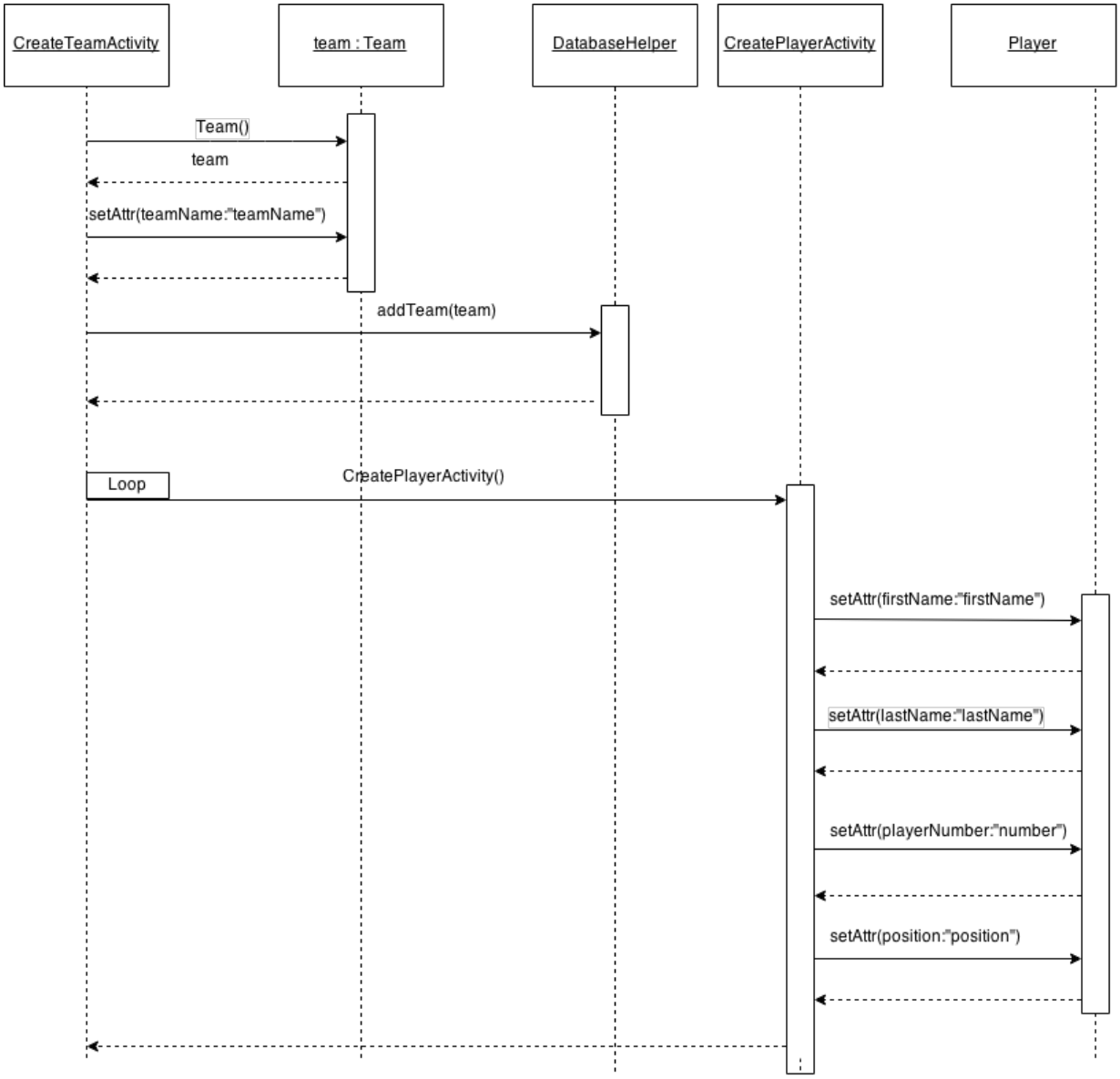
We also plan to make use of a simple enumeration to describe the different positions players can play; these are displayed at the top right.

Sequence Diagrams



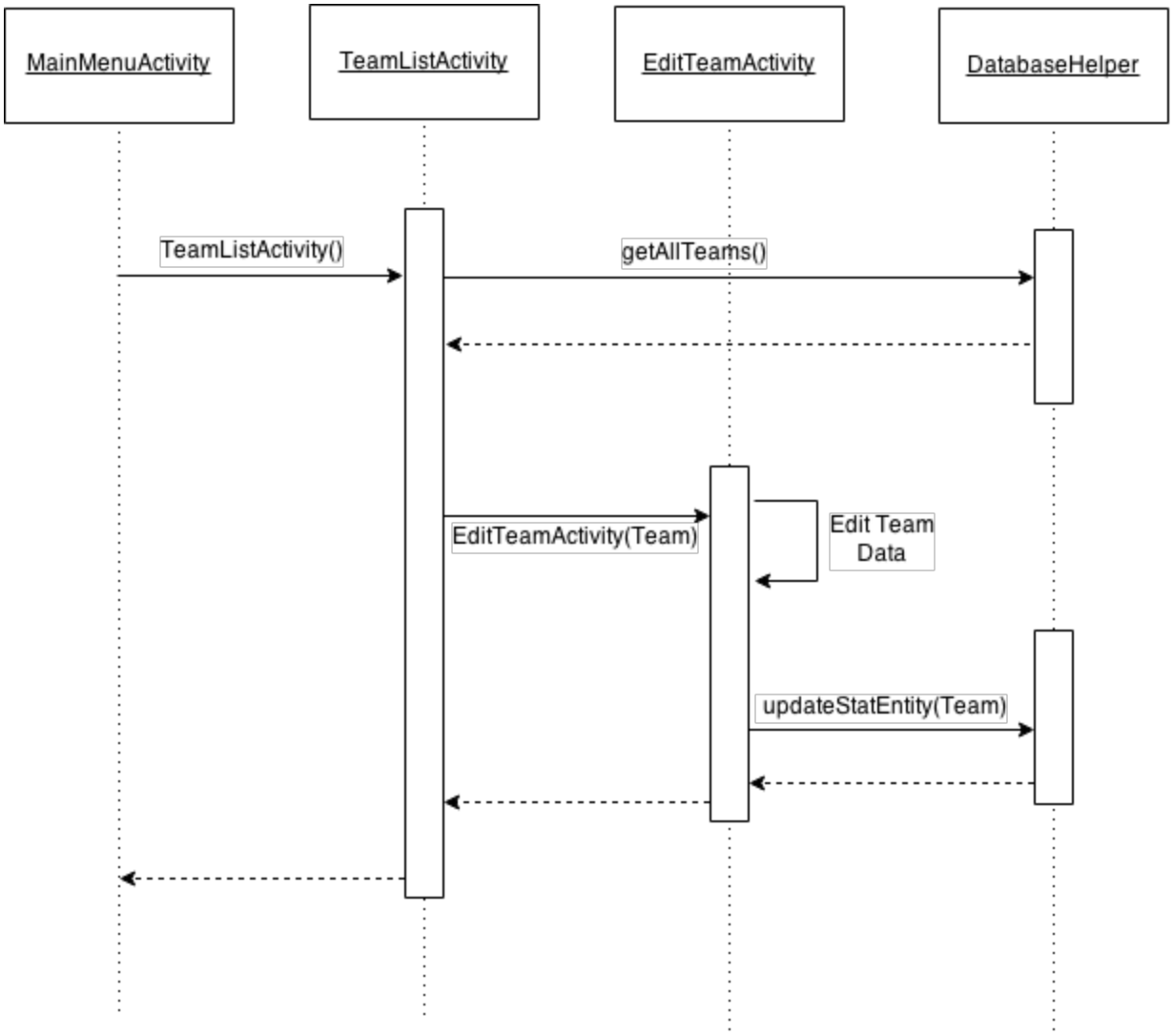
Sequence Diagram - Main Menu

This sequence diagram shows a very high/level overview of the basic functionality of the app. A button press at the main menu spawns the TeamsListActivity, from which the user can create or edit teams and players. The user can also view any stored statistics on these teams and players from the Teams List. Upon returning to the main menu, the user can also choose to spawn the GameScreenActivity, from which they can record stats and information on a game.



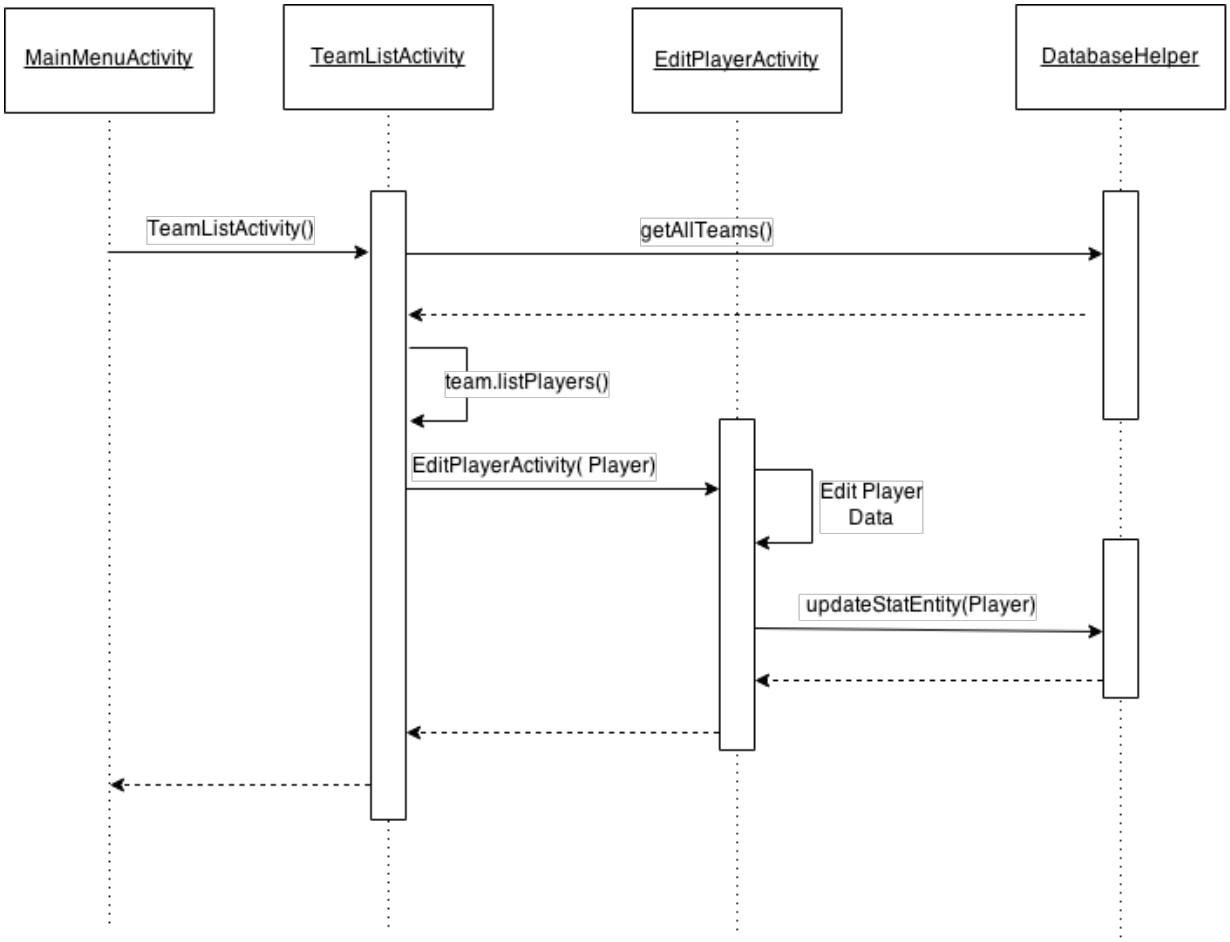
Sequence Diagram - Create Team/Player

This diagram shows the sequence of creating a team, which also involves the sequence of creating players for the team. From TeamListActivity (not shown), an option will exist to create a new team which will spawn CreateTeamActivity. The first step in creating a team will be to give it a name, at which point a team object with the given name will be instantiated and the team will be automatically added to the database of teams. Next, CreatePlayerActivity will be called from CreateTeamActivity and the user will be able to repeatedly create new players for the new team until they are finished.



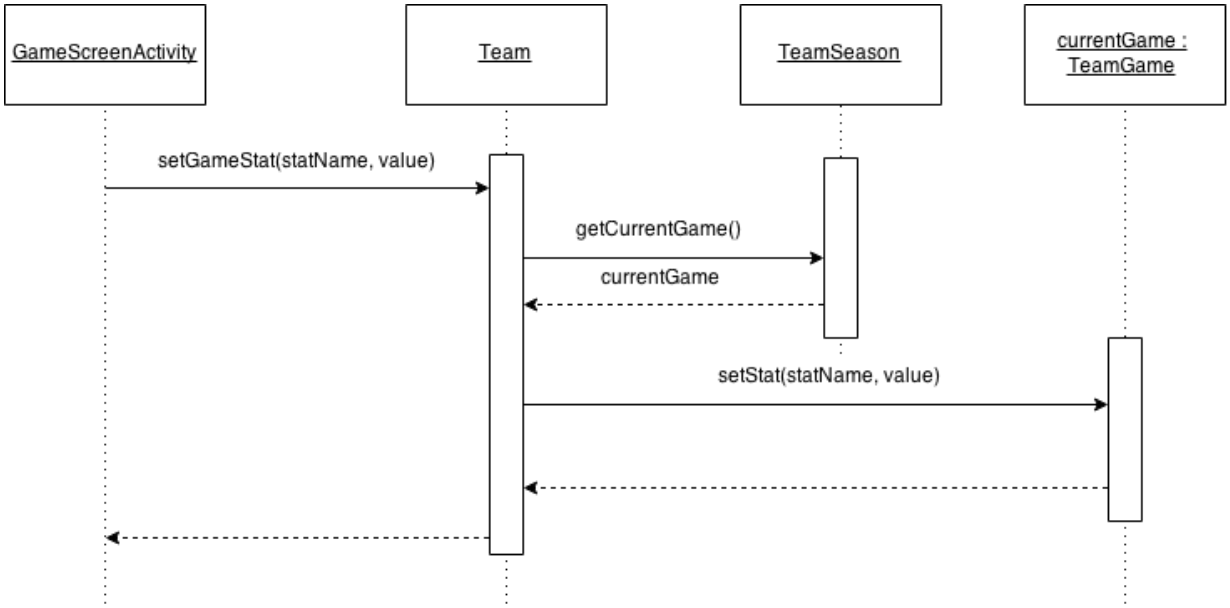
Sequence Diagram - Edit Team

This sequence diagram shows us the process of editing a team that has been inserted into the database. While the control flow is with TeamListActivity(), the user chooses in the teams list the team that he wants to edit. By clicking on the team, EditTeamActivity() will be launched, allowing the User to edit all the data about the team. After this, the EditTeamActivity will be in charge of saving the modified data in the database using the Database Helper. Then, the flow is returned to TeamListActivity, and the User can keep using the App.

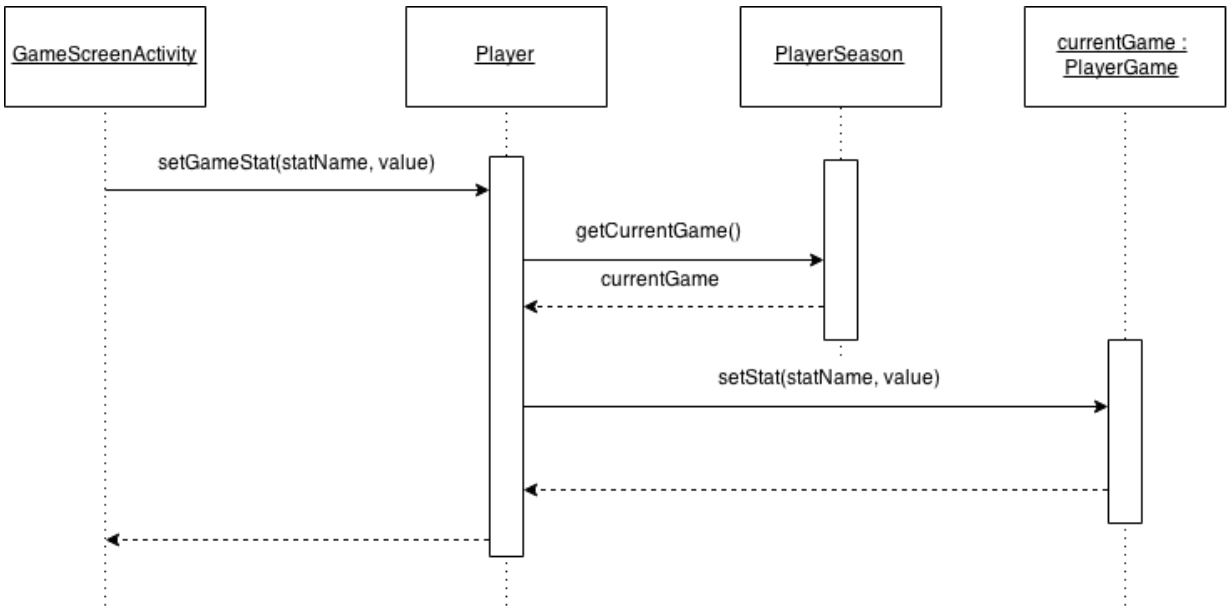


Sequence Diagram - Edit Player

The sequence diagram for an editing a player is very similar to editing of a team. The User will be able to access the list of players from each team listed in TeamListActivity(). When choosing edit a player, EditPlayerActivity will be launched, and it will allow the User to edit the player's data. After that, the activity will use the DatabaseHelper to save the data in the database, and the flow will be returned to TeamListActivity().

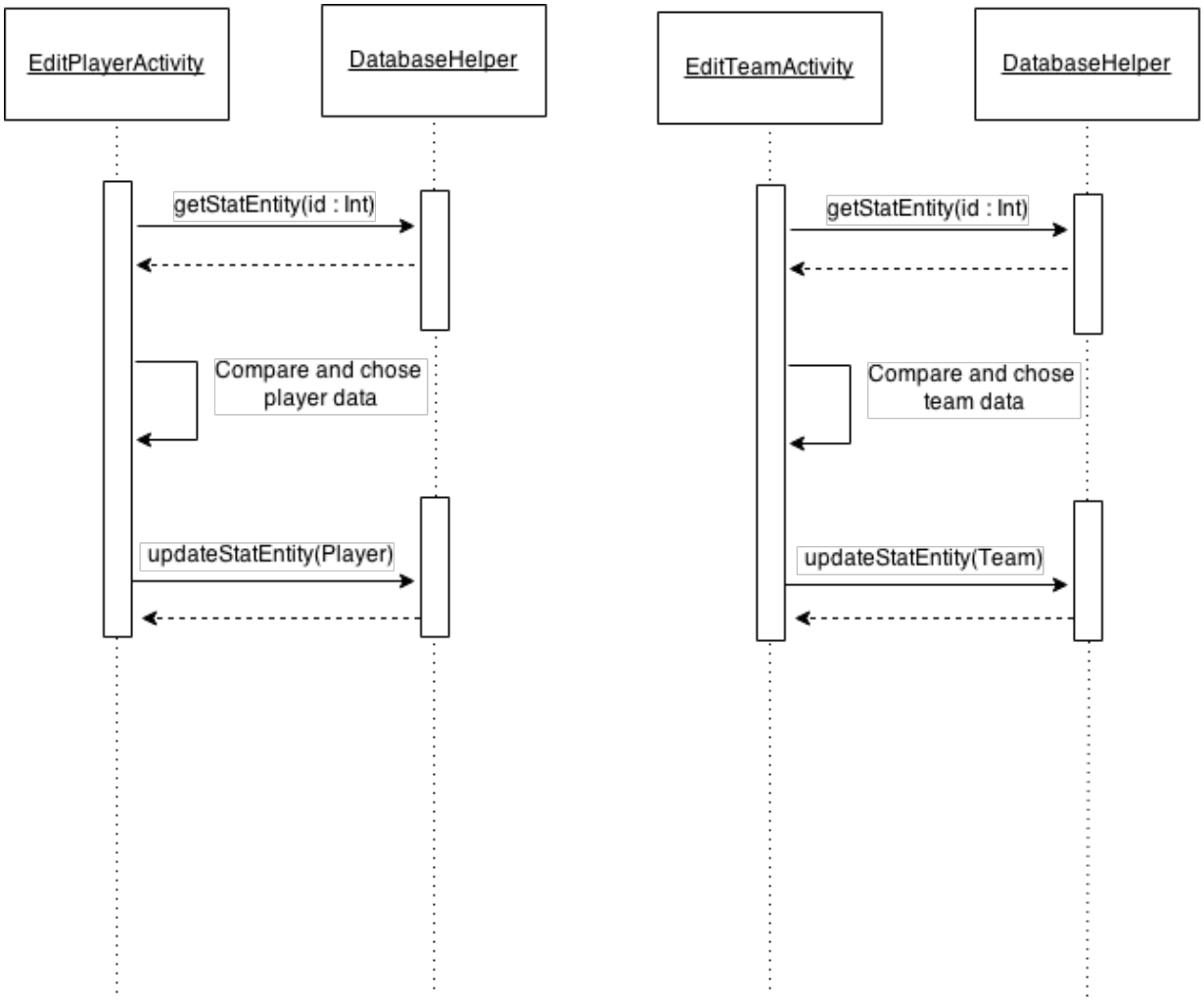


Sequence Diagram - Add Team Stat



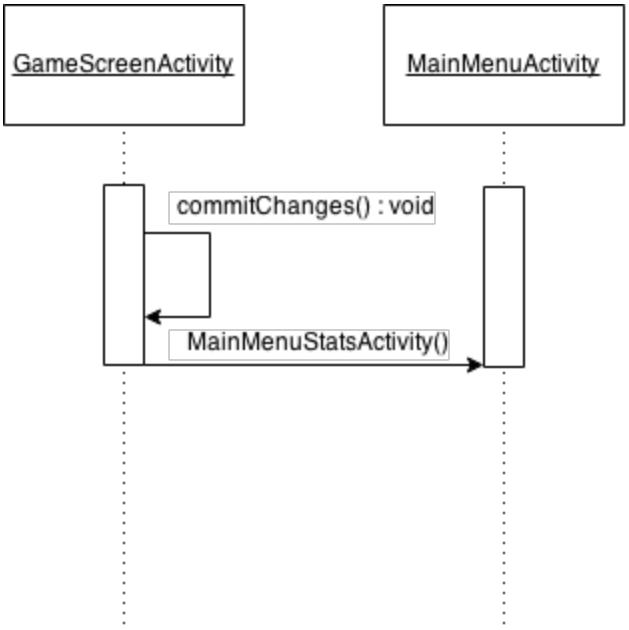
Sequence Diagram - Add Player Stat

The sequence diagrams to add a player stat and to add a team stat are identical except one is for player stats and the other is for team stats. The GameScreenActivity will host all of the stats available to add. Once a stat is selected, the user will have to choose which player or which team to add the stat, depending on what stat is selected. Once the the user makes that selection, the setGameStat() method will be called on the Player class which will then retrieve the current game and then call setStat() with the new value to be added to that players game for that game.



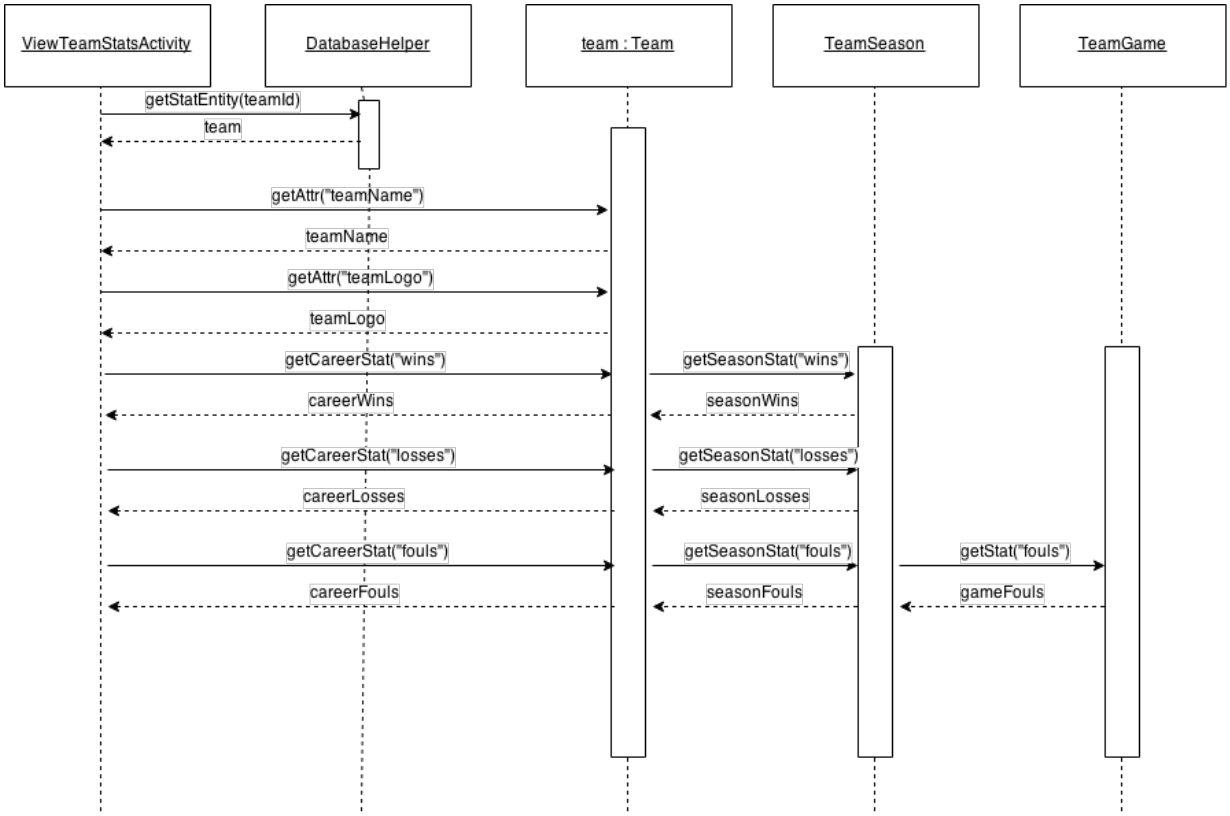
Sequence Diagram - Undo/Correct Stat

The sequence diagram to fix issues when doing an edit of a team or an edit of a player is almost the same. When the User is using EditPlayerActivity or EditTeamActivity, he will have the option to undo. The option to undo pulls from the database the current team's data or player's data and allow the user to compare the data stored with the data that he is editing and choose which one remains, saving it in the database.



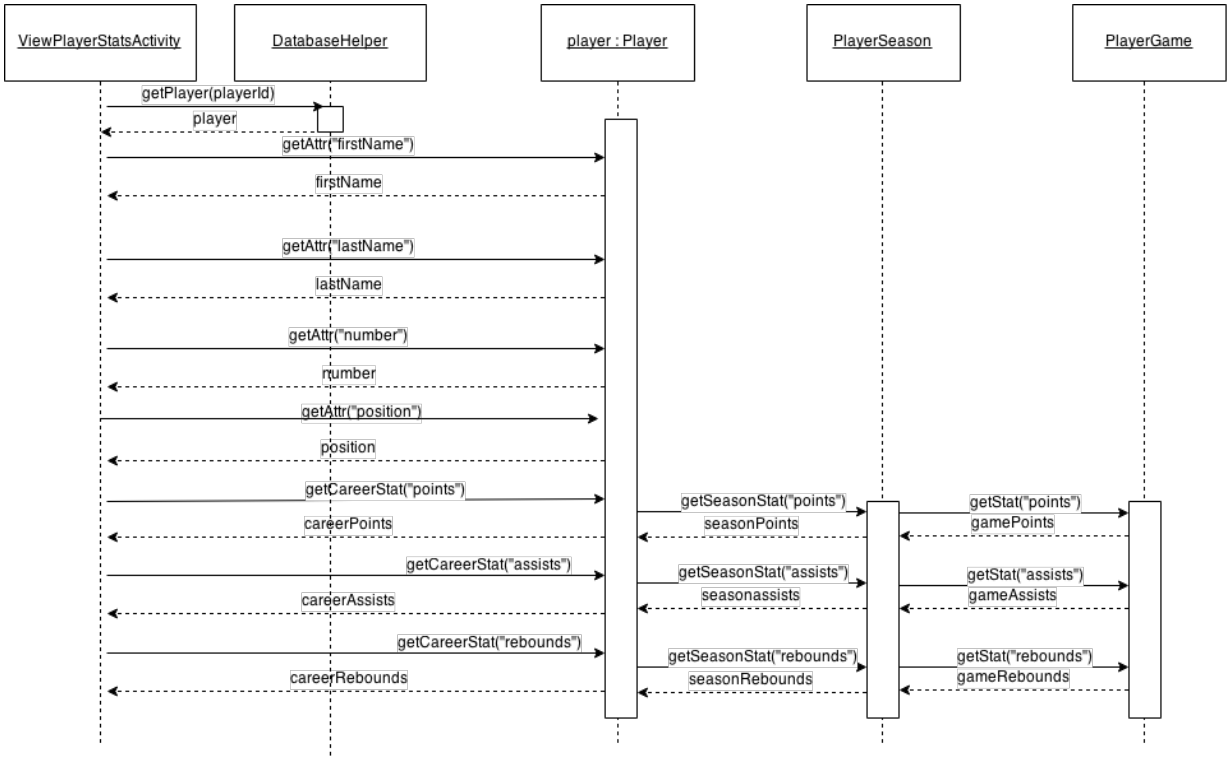
Sequence Diagram - End Game

The sequence diagram to end a game is the simplest of all. The user is in the GameScreenActivity, where all the data about the game are updated. When he has finished updating the data, the User clicks on a button that finish the game. After that, the GameScreenActivity saves the data in the database using the commitChanges() method. Then, the flow is returned to MainActivity().



Sequence Diagram - View Team Stats

Viewing stats about a given team (labeled “team” in the diagram) entails first retrieving the information about that team from the database using the team’s ID. Once that is done, the team attributes can be accessed and displayed, and some career stats about the team can be aggregated. “Wins” and “losses” must be totaled for each season, and “fouls” must be totaled for each game within each season.



Sequence Diagram - View Player Stats

Once the user has decided to view the stats of a given player (noted in the diagram as “player”), the ViewPlayerStatsActivity must retrieve all info about that player from the database using the correct playerId. The activity can then gather and display the player’s attributes as well as some career statistics. To get statistics over the player’s career, the player object must aggregate the given stats over all seasons and games.