### **FinalMarquis**

### **12/8/23**

### Formalized Proposal

Baseball is a game that has many variables to account for such as which inning it is, what the scores are, how many outs there are, etc. Two of these lend themselves well to being modeled as data structures that have been covered in this class. The first is that the batting order of each team can be modeled as a queue. “Popping” from the queue will give the next batter, and “peeking” beyond that will give the on-deck batter. Once a play is resolved, pushing the current batter back to the end of the queue maintains the batting order. The user will enter the team names and player names. Once these are put through input validation, a bubble sort will be performed on the player names to put the batting order in alphabetical order. The second is a stack of play outcomes that continually grows as the game progresses. By looking at the top of the stack, a last play summary can be provided on the user interface.

The program and user interface will be a digital scoreboard for a baseball game. It will post and refresh the innings, outs, scores, at-bat and on-deck players, what bases have a runner on, and the last play, as well as signaling which team is at bat. The classes include Game and Teams classes which have various methods.

### Time/Change Logs

Sprint 1: mid-to-end of October.

Adjusted project from being based on a basketball scoreboard to instead be based on baseball since there were places that a stack and queue data structure could make sense. Defined play by play and batting order classes.

Sprint 2: beginning-to-middle of November.

Drafted up additional classes, Play-by-Play and Batting Order which would create and maintain the (2) distinct data structures.

Sprint 3: middle-to-end of November.

Started UI work, had a decent amount of the main scoreboard screen done.

Sprint 4: beginning of December.

Bulk of project work was done here as I finished other course a bit early and could dedicate myself fully to this. Finished integrating UI and classes. Adding UI and message boxes to smooth out communicating to user (error message box, game over, entries saved). Added bubble sort method. Added baseball diamond tracking what bases are loaded. Plenty of user testing.

### Lessons Learned

There were several lessons learned while working on this project. First off, recalling how to choose, set up, and update the various Tkinter (user interface) widgets took some time as it wasn’t readily familiar. Building off that, understanding that a radio button callback needs to update variables in the context of a class since there is never a return value (that goes anywhere) when you call to a method via button action. The sorting algorithm was something that had slipped off of my radar until this week so it cut into some of the time I had originally planned to spend developing unit tests. Learning about TDD, I wish I would have tried to implement that ahead of time to make it very clear when the program is working successfully. However, there were many intermediate steps (such as the mechanics of outs, innings, tracking who is on base) that I underestimated how tricky they could be before being deep into integrating the UI and Classes.

### CODE including comments

My project, Awesome Project Titles is at [http://github.com](https/github.com)/myusername

### User's Manual

A screenshot of a computer

Description automatically generated

Pressing the run button on the “GUI.py” file will open the first window. In this window, the user will enter in team and player names for two baseball teams. To confirm entry of names, click “Save Teams”.

A screenshot of a computer

Description automatically generated

If the entries meet input validation requirements, user will see the “Entry success” message window. Click “OK” to proceed.

A screenshot of a computer

Description automatically generated

If the input is invalid, a prompt will appear indicating that the input was invalid and what the requirements are. After acknowledging the error message, the user will need to re-run “GUI.py”

A screenshot of a baseball scoreboard

Description automatically generated

The Baseball scoreboard screen appears which contains all of the game information including Outs, Inning, team names, team scores, which batter is at bat, which is on deck, play outcome options from an at-bat (out, single, double, triple, home run), and last play. The user will interact with this screen by selecting a play outcome, and clicking “Log Play”. The “Log Play” button will resolve all results of the play such as incrementing outs, scores, etc. The baseball diamond in the middle will update to reflect the bases that are loaded. The at bat and on deck batters will refresh. The baseball image indicates which team is currently at bat. Finally, the last play will update.

A screenshot of a computer

Description automatically generated

At game end, a “Game over!” message box will appear indicating which team won or if there was a tie and the score. Clicking “OK” ends the program.

### Conclusion/Summary

In terms of MERUSE principles, I believe this project has it’s strength and weaknesses. For Modularity, there are several classes, Teams, Game, PlaybyPlay, and BattingOrder. I believe the last two definitely make sense to keep separated since they are for distinct datatypes but I’m not sure if Teams and Game couldn’t have been combined. For Efficiency, the method that comes to mind is the bubble sort. Bubble sorting isn’t necessarily efficient, but in context of (2) lists of just 9 players to sort, it doesn’t really hinder anything. For Robustness, the program itself does not spit out any errors (from what I can tell). But the trade off is the code doesn’t appear Elegant throughout. There are several “if” statements that are just in place to dodge an exception being thrown depending on the game state. On Usability, the program does fulfill what I intended, it keeps track of a game of baseball from the at-bat and on-deck batters to the last play that occurred to the team scores as well. Finally, for readable, I think this is something that could have been improved. The “GUI.py” program in particular has a lot of code so perhaps that could be refactored to be simplified.

The baseball scoreboard project is successful in allowing a user to enter in team and player names and manage all the updates to a game of baseball while just asking the user to input plays one at a time. I could see this type of scoreboard being helpful in something like an announcer setting since the player coming up to bat and on deck are constantly being updated in the background, as well as the innings, outs, scores, bases loaded, etc. I’m proud of how it turned out visually and the code logic appears sound.

At times, future improvements become more apparent after working with the program for a little while. One improvement would be to add more to the player data to make a better decision in how to set the batting order. For instance, a player with a good batting average may be more advantageous to be assigned to a certain place in the batting order (rather than just sorting batting order alphabetically). A future version could be more detailed to track pitch count. Finally, future versions could actually track at-bat outcomes for each player, so that the user can see some player stats.