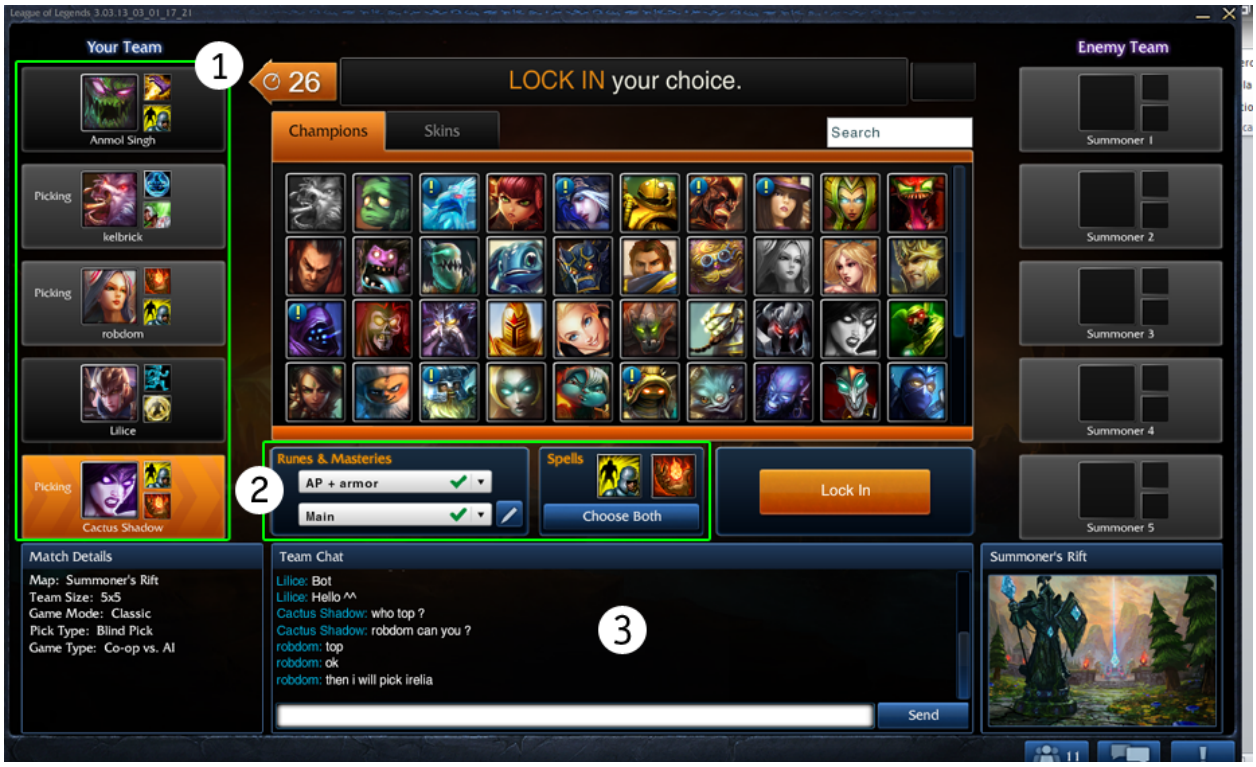
Project: Pre-Game Decision Making Application - League of Legends Team Counter Pick Report

-Background

League of legends is a multiplayer strategy battle game. The game consists of over 140 unique characters, known as champions who each have four completely unique abilities, that impact the game by dealing damage to enemies, healing allies, preventing enemies from preforming actions or augmenting allies. Before the game takes place there is a very important, pick and ban phase, which is the focus of our application.

During this time the players ban champions to prevent their opponents, or sometimes their teammates, from using those champions and then pick the champion they want to play. The decision of which champion to play is heavily impacted by what the opponents have already picked.

The reason we chose to create an application based around the pick and ban phase was due to the complexity of figuring out what champion would be the best go against the current opponents and the limited amount of time a person has to do it in. This app would simplify the decision while still giving the user a pool to choose from.

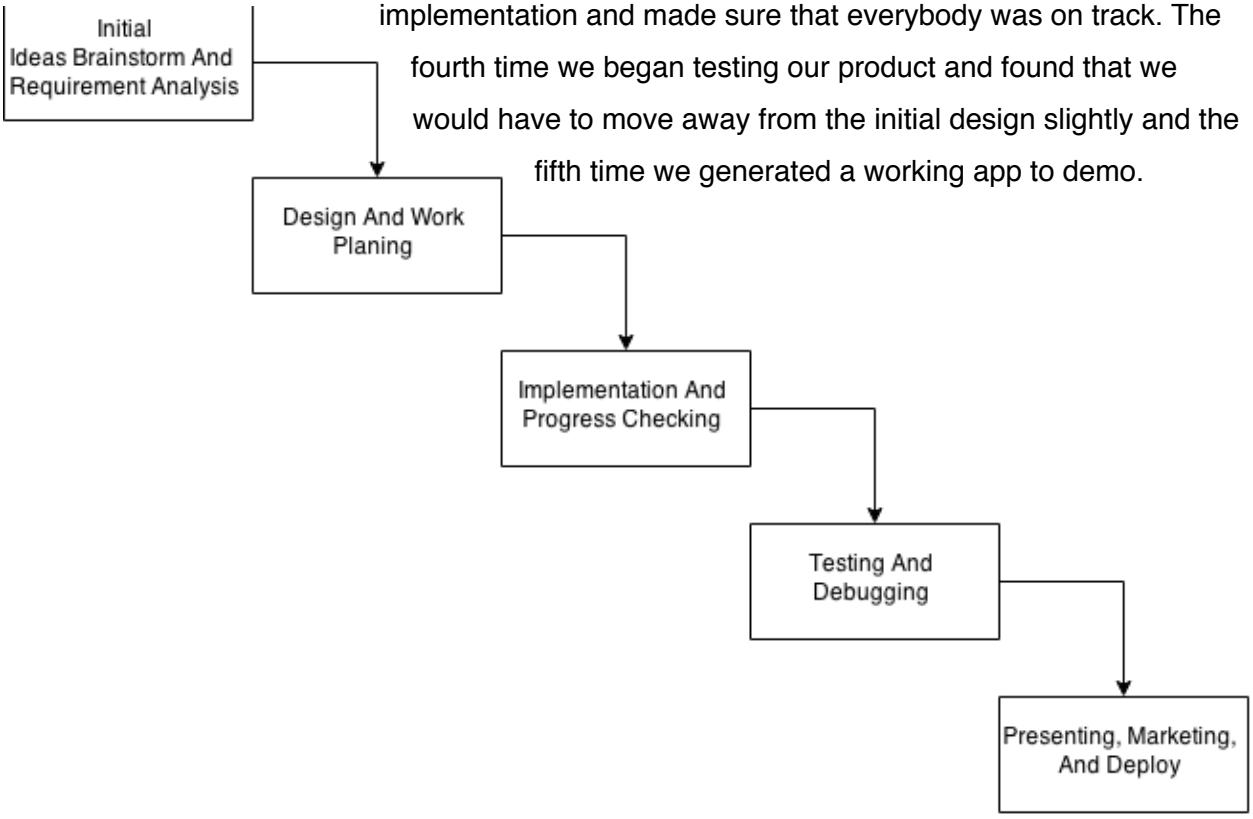


Picture source: http://esportresearch.net/complete-description-of-league-of-legends/
Explain more about the game.

-Development Model

When designing the app we decided to use the waterfall model. The project was spilt up enough and straightforward enough that the we could parts to people after we had the designed the project. The project was small enough that we would not need frequent meetings to check on everyone's progress.

The first time we met we discussed the requirements for the project and what we wanted to make it do. The second time we created the design for the project and separated out who was going to do what. The third time we checked on the progress of implementation and made sure that everybody was on track. The fourth time we began testing our product and found that we would have to move away from the initial design slightly and the fifth time we generated a working app to demo.



The waterfall model was successful as we were able to rely on each member to complete their part. Each member was caught up on their tasks at the planned times and able to communicate what they had completed to the group. Because of each member's efficiency, every stage of the model is successful, so is the whole model. If the code had been more complex we would have had difficulty without extra communication that would have been provided by an agile process, however due to the simplicity of the code the waterfall model worked.

-Requirement Analysis

User Interface Frontend

At our first meeting, we set out three main functional requirements. The first was the User Interface. We wanted to create an interface that resembled the league of legends client, so that it would be easy for the user to use. Since they should already know the layout of the client, it would make an easy transition into using our software. As with the league of legends client, the big box of champions would be in the middle of the screen. Both teams selection of champions would be on the left side of the screen. This would insure that the slight difference from client would make it distinguishable from the actual client. As selections are made in the champion select, the box in the middle would update with the possible counter picks to the opposing teams selections. The counter picks would be separated into three categories (good, okay, and bad). This would let the user know all the options they have in selecting the best possible pick.

A desirable function we wanted was the option to separate champions into the lanes the usually go to(top lane, jungle, mid lane, ADC, and support), which works like a classification filter to make the application more user-friendly. This would ensure the user can easily pick for the lane they wish to play, versus having to sort through the champions themselves.

In the end we made the user interface just like the league client. This includes having the enemies champion selections on the right side of the box in the middle. This was different from our original plan, but was at the same time easier for the user to use. The software was left plain white, so to be different from the league client.

We also didn't get the different tiers of counter picks implemented. Only the best picks get outputted in the window once the selection is made. Along with that is the fact we do not have the application automatically update whenever you put in a champion. You select all the champions that have been selected then submit the query and it outputs the possible counter picks.

We did not have enough time to implement our desirable function to separate the counter picks by the lane they correspond to.

Database Backend

The second functional requirement we came up with was a database of all the different champions. One table was supposed to hold the champions information. Such as name, lane, etc. A second table would be a massive table that has each champion on the column and row, so that each matchup would be represented. This would be used to factor in how each champion counters every other champion. An optional requirement would be a table that shows what certain summoner spells counter specific champions. This would tell the user when the enemy picks a certain champion, that a specific spell would be useful against them.

We eventually decided to not go with a database. We decided that setting up a database and connecting to it would take more time than necessary. We made it so that it downloads the information it needs from lolcounter.com and stores it into an excel spreadsheet. The program then accesses the information whenever the user requests a counter pick. After the initial download, there is no more needing to access the info. This makes it a better method than having a database in a single location. We also did not get time to implement the summoner spell counters.

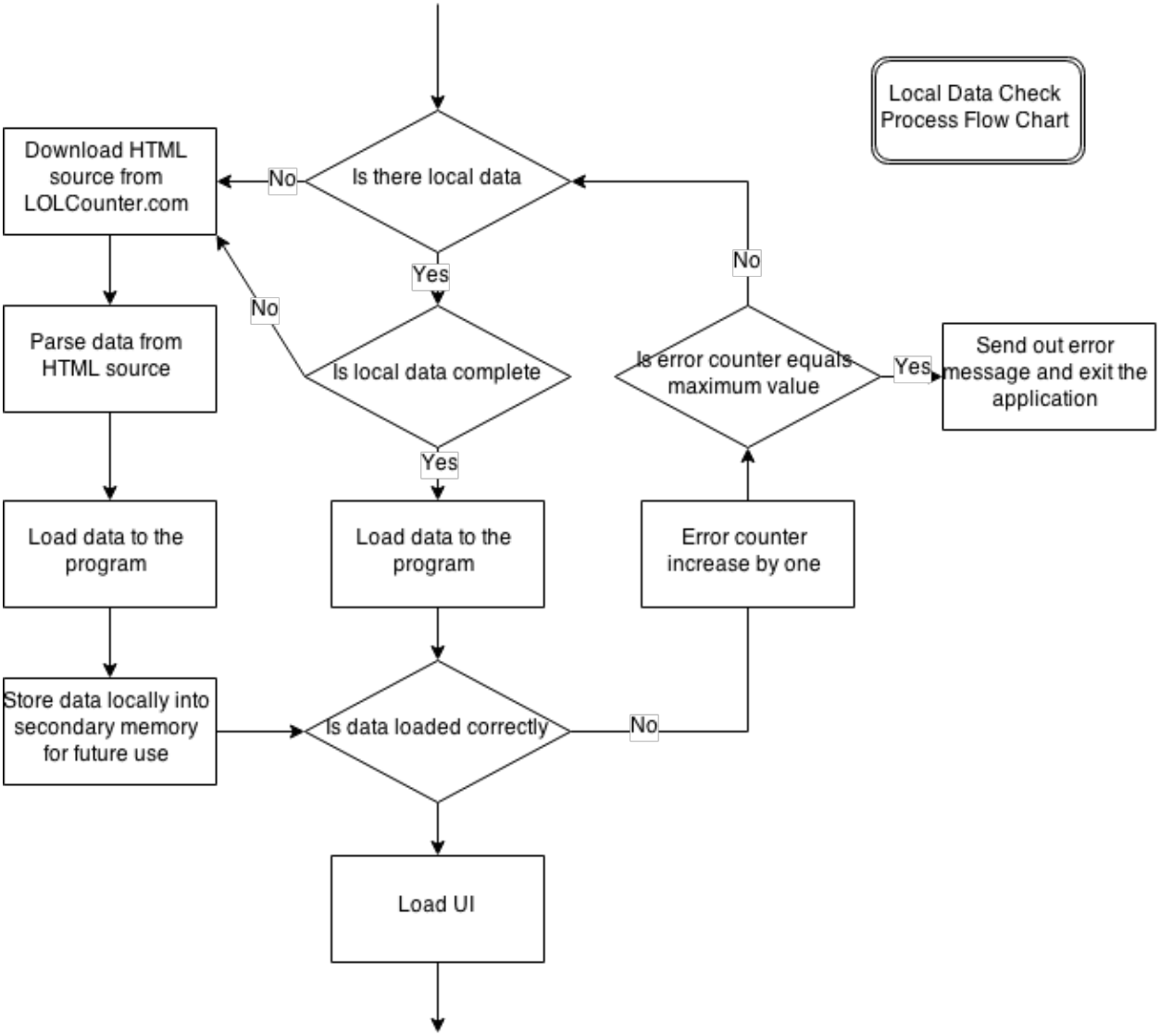
Result Calculating Algorithm

The final functional requirement we came up with is the counter pick algorithm itself. The original idea was that the algorithm would sort through the table full of values representing whether each champion countered each other or not. It would add up the values for all the champions listed and output the champions based on the three tiers(good, okay, and bad). The idea was to calculate the counters of the entire team, and not just an individual pick. So it would add and subtract the numbers based on the entire teams champion choice. The algorithm was also supposed to recommend champions based on the champions your team has already selected. This would add to the overall team synergy and help for an overall victory.

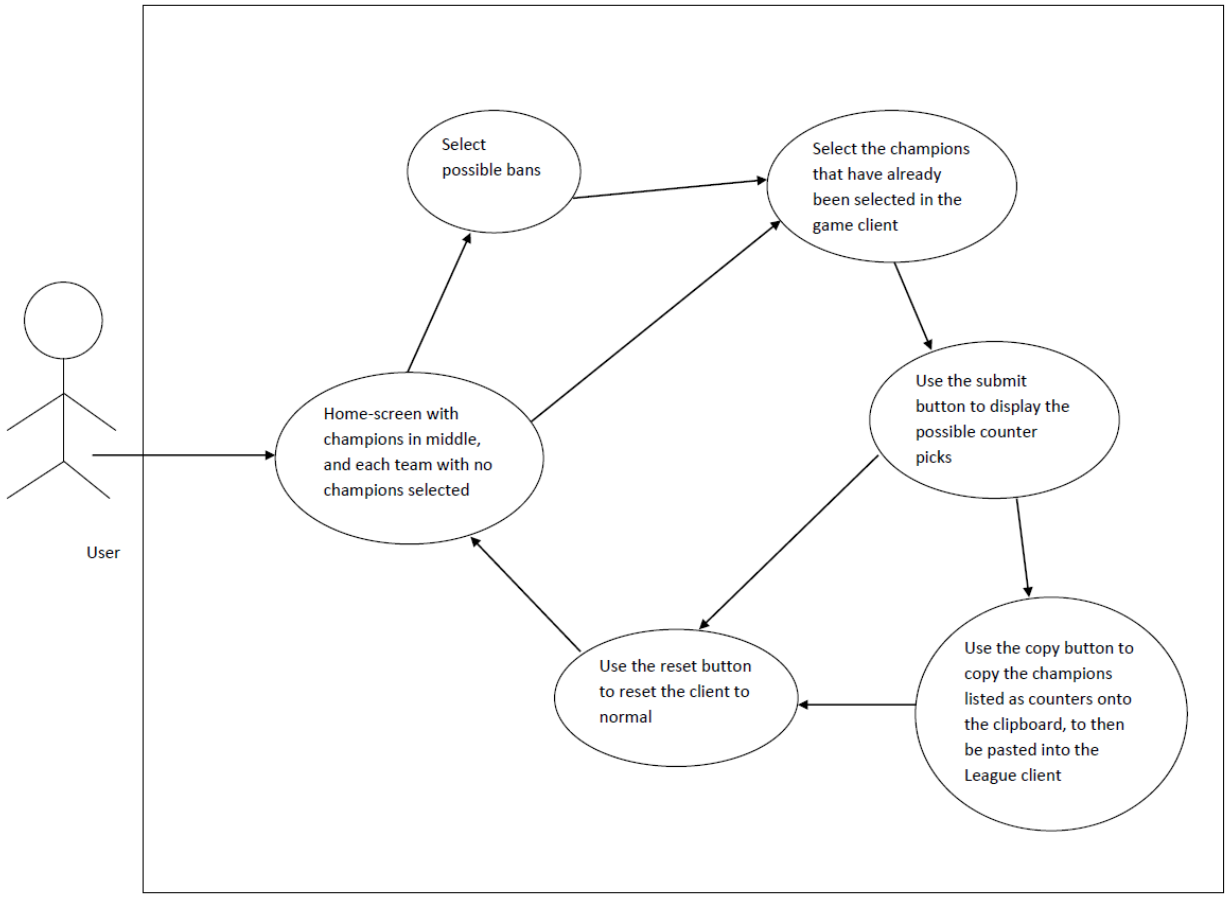
The algorithm was made, but instead of going through a table, it parses through an excel sheet that has downloaded all the information from lolcounter.com. It then, at this time, only outputs the best possible picks. We did not get around to adding the three different tiers, just the best. Also, we did not get around to adding the part about team synergy. So the algorithm does not take into account your teams picks at all. It only worries about what the opposing team has selected.

-Design

Local Data Check Process (before UI)
Flow Chart



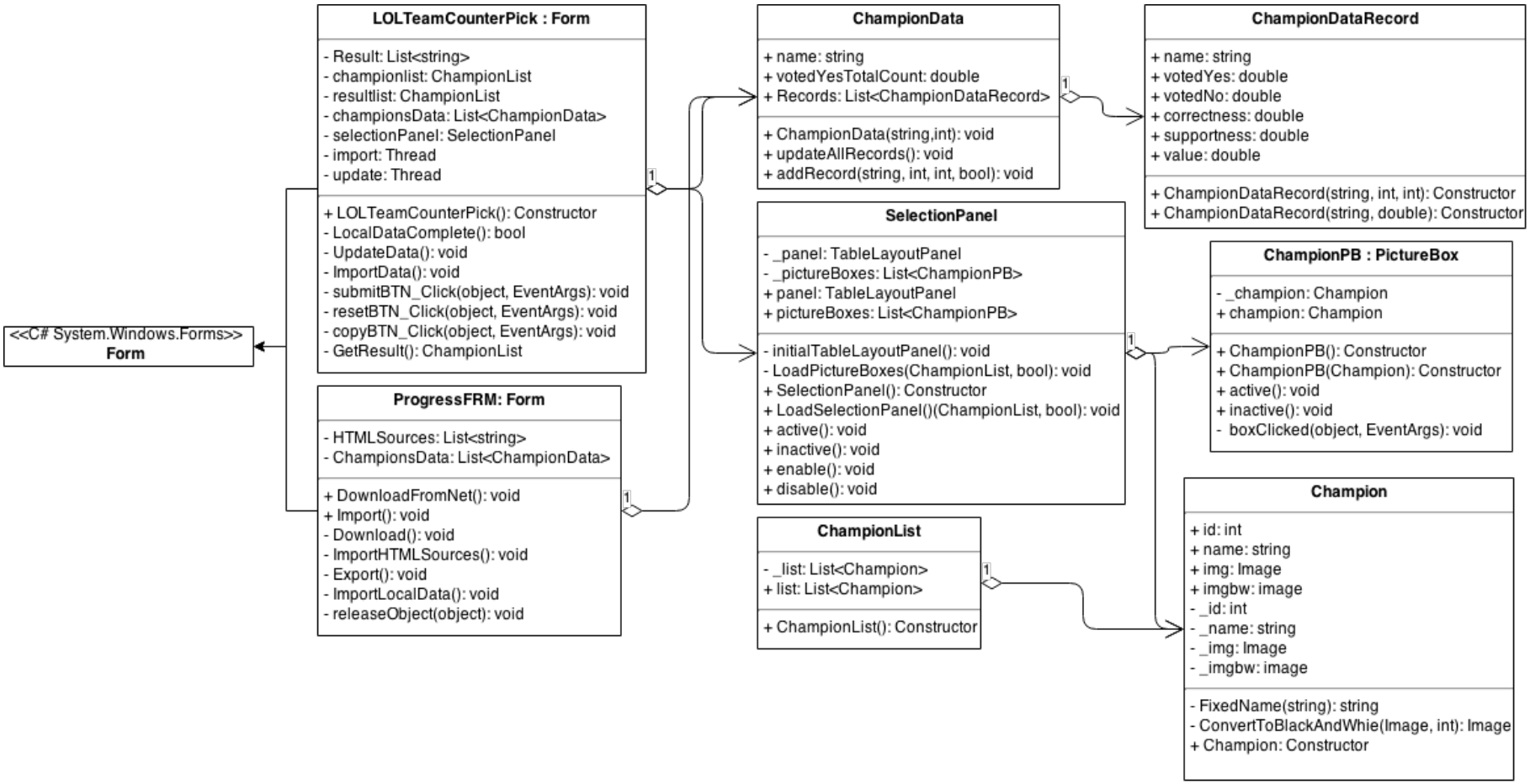
UI Diagram


Select possible bans
Select the champions that have already been selected in the game client
Use the submit button to display the possible counter picks
Home-screen with champions in middle, and each team with no champions selected
User
Use the reset button to reset the client to normal
Use the copy button to copy the champions listed as counters onto the clipboard, to then be pasted into the League client

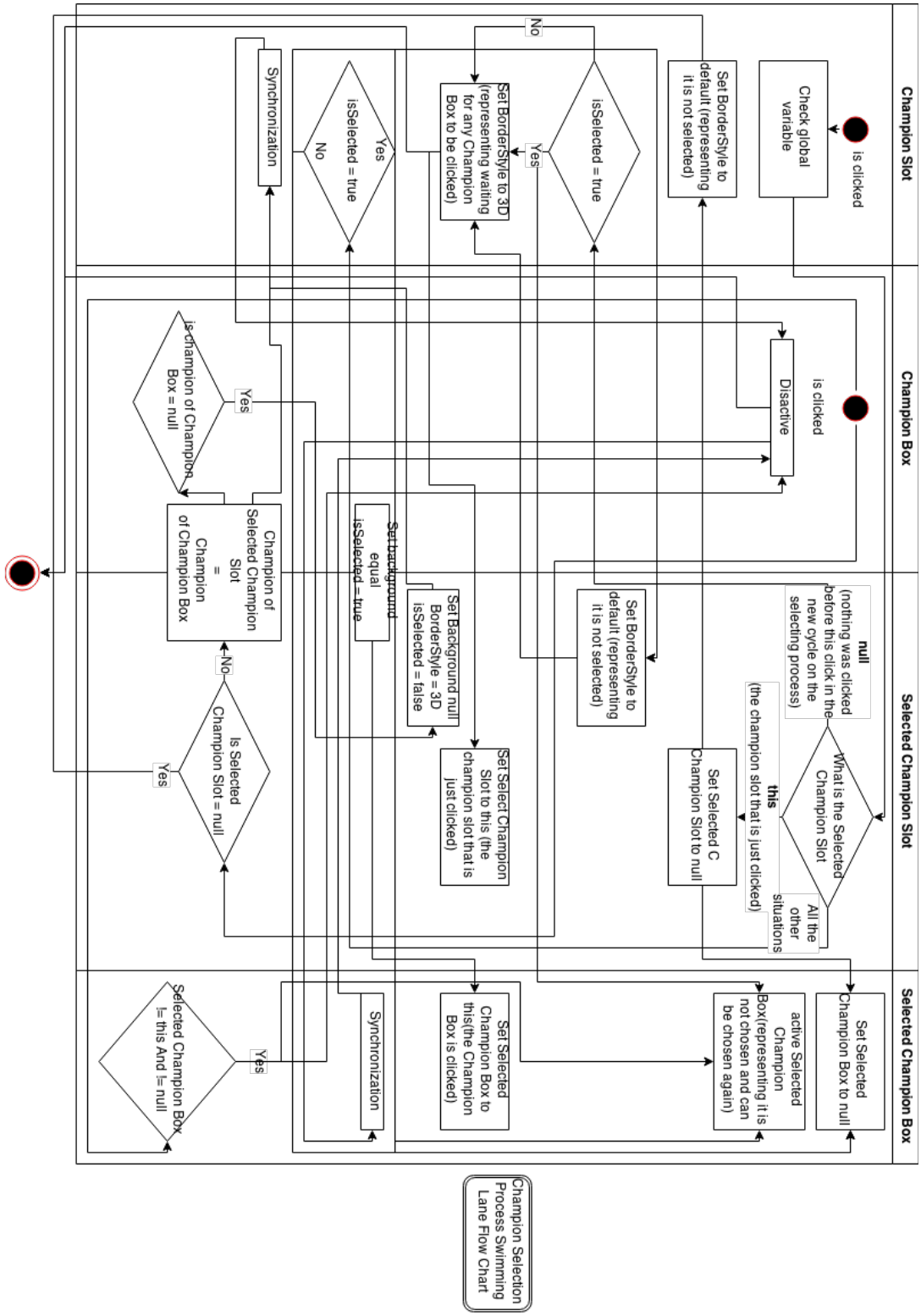
UI Design


LOL Team Counter Pick
Allies
Selection
Opponents
Ban
Ban
Submit
Reset
Copy
Control Buttons
The Blank in the middle bottom is for the future functions
Selection Pool And Result Display Area
Selection Area For Allies And Opponents
Ban Area For Allies And Opponents

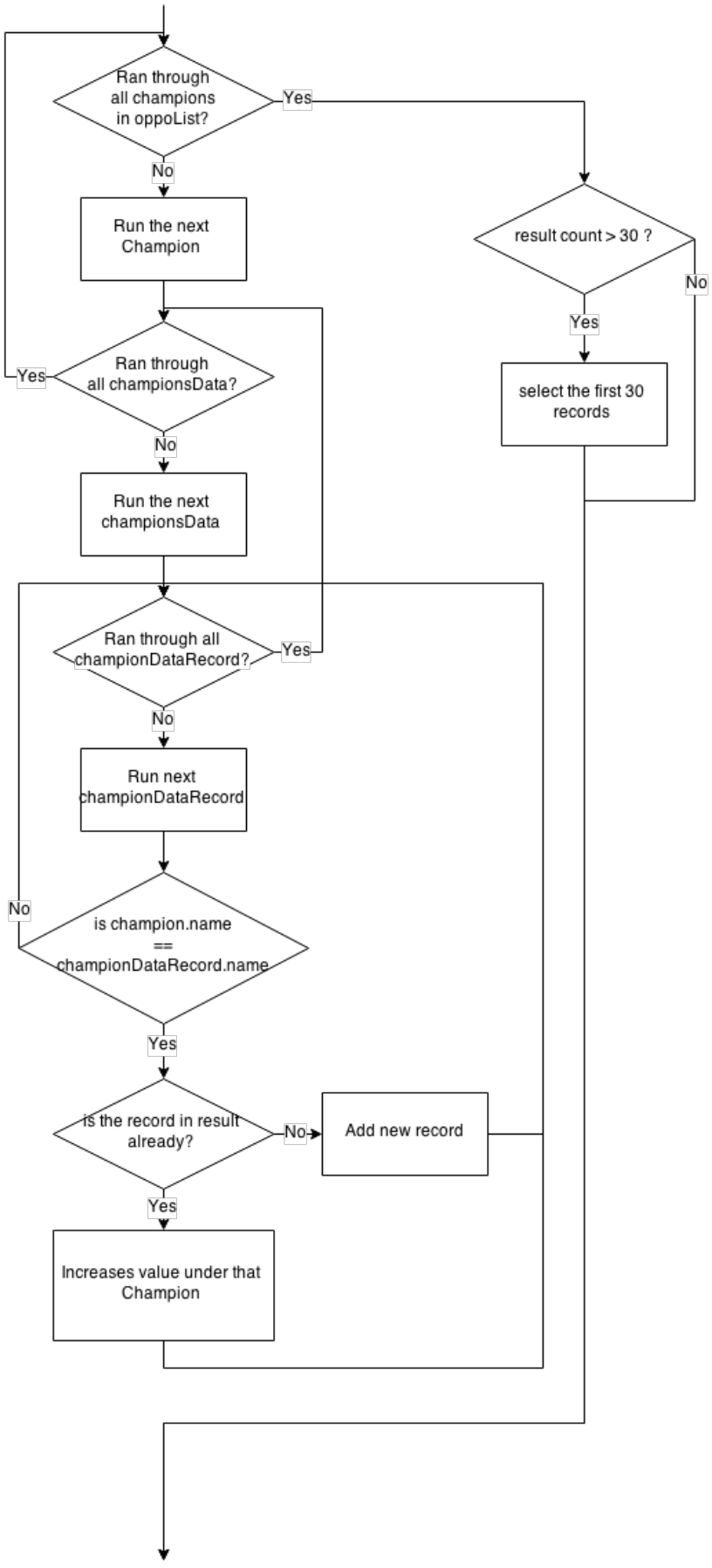
Class Diagram



Champion Selecting Process (After UI showed up and before submit selection)
Swimming Lanes Chart



Result Generating Algorithm
Flow Chart



Database Design

For our League of Legends counter pick application, we originally planned to use a database to store all of the information required for our selection algorithm. This included every champion's basic information, possible team roles, and relative match strength versus every other champion. Champion information and roles were retrieved from leagueoflegends.net (League of Legends' official website) and stored in the Champions and Roles tables respectively. Relative match strengths were to be determined using win percentages of online ranked matches. These percentages were retrieved from elophant.com and stored in the Matchups table. The relational models and schemas for this version of the database can be found elsewhere in this report (marked as "before refining").

In the original version of the database, the data representing champion's relative strengths were treated as a relation between one champion and another; so, each relation between two champions was its own tuple in the Matchups table. This meant the Matchups table would contain n^2 tuples where n is the number of champions (currently ~120). As a result, some queries were taking too long for our application to compute the optimal selection in less than one second (one of our functional requirements). Thus, the database needed to be refined.

In order to minimize the amount of time for queries to be processed, the number of tuples in the Matchups table needed to be reduced significantly (query complexity is based partially on the number of tuples to be searched). Because it's possible for any champion to oppose any other champion, the Matchups relation has total participation for all champions. Thus, the data represented in the Matchups relation could also be represented using a composite attribute on the Champion entity. This composite attribute would contain a separate atomic attribute for every champion and would indicate the relative strength of the entity against that particular champion. This change was implemented in the second version of the database, resulting in the refined relational models and schemas seen elsewhere in this report.

The refined version of the database was equivalent to the original in terms of functionality; both contained the exact same data. However, refining improved the database in both needed storage and retrieval speed. While the Champions and Roles tables were unaltered, meaning they required the same amount of storage space in both versions, the Matchups table, which was by far the largest, was reduced in size by roughly a factor of 10. This is largely because the champion's names took the most space to store and the original version repeated each champion's name for every relation in which it took part. Smaller storage space is helpful, but the more important benefit of the database refinement was its speed. The new way the matchups table was stored reduced its number of tuples by two orders of magnitude, giving it fewer than were required for the Roles table. This reduced the slowest query time from about .6 seconds to a little over .01 seconds.

-Implementation
with Codes attached in the following pages

-Testing and Debugging

There was no unit testing. The only testing we did is run the application with different inputs(different selection of Champions). And let League of Legends players determine if the application generates good outputs (since there are no actually standard to measure the goodness of the outputs). And It turns out it works great.

As for debugging, since we follow every stage of our model tightly, and it's a relative small application, there is not much bug we have found. And we fixed all bugs that we found.

-Presenting

Before the day of presentation, we have anything we need set up on our presenting laptop including, the slides and demo, as well as the local data that the application needs. During the day of presentation, everyone showed up on time and present our parts orderly as it was planned. Besides the slides, there was interact with the audiences through the demo, everything was successful.

-Conclusion

Since the project is relatively small, everything was planned out well ahead, and everyone follows track tightly, the whole project is perfectly complete throughout every stage and as a whole.