Poker Equity Analytic Tool

Scrum Master: Michael Bowen – Git ID mjb236

Software Developer: Matthew Landram – Git ID mcl46

Software Developer: Brandon Lockridge – Git ID brandondjango

Quality Assurance Manager: John Riesenberger – Git ID jriese

CS 1530 – SPRINT 4 DELIVERABLE      Due: July 22, 2015

SPRINT DESCRIPTION

The fourth and final sprint flew by entirely too quickly for our liking. We ended up needing to scramble a bit to even get the program to perform the calculation for which it was being designed. With all of the customization and blank Cards that needed to be considered, said calculation also proved to be more difficult to perform than anticipated.

We actually needed to take two entire separate attempts at getting the calculation working. The first one, while producing numbers, ended up proving to be inaccurate when the same scenario was run on an existing piece of software. We then scrapped that implementation and, admittedly hurriedly, threw together a calculation algorithm that would provide more accurate results. The time crunch, however, led to this section of code being ugly indeed. Given more time, this could be cleaned up and refactored to be much more readable, but the priority was to at least get a calculation performed.

The HandGenerator and CompleteHandGenerator classes were completed during the final sprint prior to implementing the equity calculation. The HandGenerator class was designed to generate all combinations of HoleHand objects from the data provided by the user. For example, given the hand AA, the HandGenerator would return a list of HoleHand objects representing each of the six combinations of ace pairs.

The CompleteHandGenerator picks up where the HandGenerator leaves off, by taking a HoleHand object and a PokerTable object, and returning a list of CompleteHand objects representing each of the 21 different ways of choosing five cards from the 7 cards contained in the hole hand and poker table.

So, lists of CompleteHand objects are generated for the player and the enemy, and each of the player’s hands is then compared to each of the enemy’s hands to come up with a percentage of winning.

Our experiences with the code reviews this sprint were much the same as in sprint three. Initial reviews were done informally, with feedback given through the Slack chatroom or email prior to merging new code. However, due to the time crunch we were feeling to get a product that might actually do what it was supposed to do, we were probably a bit more lax on what might pass review than otherwise would be the case. We simply needed to get a working product.

There were not many disagreements of note during this sprint. Everyone was in agreement on which user stories needed to be completed before the final sprint ended. Once that was discussed, there did not even feel like there was enough time to even bring up disagreements.

Unfortunately, the product did not come together in an appreciable way until very late in development. We wanted to send a prototype to the customer to get some feedback, but the program was not even functioning until very late in the sprint. As such, any feedback given by the customer would not have been able to be corrected for anyway. We decided instead to at least try to get a working calculation.

There were many challenges with the code during this last sprint. As touched on earlier, the actual calculation logic required two separate and different attempts to get working. It still may not even be totally accurate. It’s also not very fast, but we wanted something that worked at all. Had we more time, we would certainly focus on making the calculation faster and more accurate.

The CompleteHandGenerator class proved to be slightly more difficult than anticipated. We spent a good amount of time thinking about how to cleanly program a choose five of seven combination, but in the end, facing the deadline, the class was simply hard coded with a two dimensional array representing each of the 21 different combinations. This array was then accessed to create the list of CompleteHand objects. Not as sophisticated as would be desired, but it works.

As touched on before, the decision of which user stories to focus on was pretty simple and agreed upon by the group. We needed to knock out the top user stories in the backlog so that we would have program that worked at all, instead of just two month of non-fruitful work. This is touched upon again later in the document.

USER STORIES COMPLETED

1. As a poker player,

I want to be able to choose hand ranges, adjusted for the suit of the card,

So that I can get accurate results from equity calculations.

1. As a poker player,

I want to be able to compare two players’ hands to determine hand equity

So that I may make more informed decisions on poker plays.

1. As a poker player,

I want to be able to represent all combinations of hand ranges with logical sense

So that the equity results will be as accurate as possible.

1. As a poker player,

I want to be able to assign random cards at any point

So that I can evaluate situations with unknown cards.

REASONS FOR FOCUSING ON ABOVE USER STORIES

All of the user stories focused on during this sprint were integral to the purpose of the program. As such, the decision to focus on these particular stories was an easy one.

User story one was the crux of the problem described by the customer. This was the entire reason that the customer originally wanted the program created at all. Per the customer, the existing equity software did not allow for choosing of certain suits and combinations thereof. We designed the program to be customizable for any card-suit combo.

User story two was secondary in importance only to the above user story. The program could allow you choose the card-suit combos all you want, but without performing an actual calculation, those choices would have been moot. So, getting the program to actually perform a calculation was very important to us, and to the customer.

The third user story completed was similar to the first user story, in that it was a focus to allow the user to easily input all of the variations that s/he would like to analyze. The user of the program will be able to represent all of the multitudes of hand variations through text or graphical input.

Along the lines of user story one, user story four was implemented to allow for great customization of the variables to be considered. The user is able to leave cards blank (in a logical manner), to tell the program to check every possible combination for that particular card. Though, the program as it stands does not do the calculation as quickly as we would like. As such, leaving too many cards blank will result in very long response times. Perhaps this user story should be considered half complete.

The other user stories in the project backlog were decided to be axed due to time and their relative unimportance to the primary function of the program. The ability to import, export, and archive hands was deemed to be not very important to our purposes. Likewise, the ability to compare more than two hands was seen as not quite as important, as well as too time consuming to implement.

USER STORY BACKLOG

1. As a poker player,

I want to be able to compare more than two hands at a time

So that I am able to make more informed decisions about playing more than one opponent.

1. As a user of the poker equity tool,

I want to be able to see the equity from my current hand overlaying my poker client

So that I can manually calculate equity in games I am actively playing.

1. As a user of the poker equity tool,

I want to be able to import a history of played hands

So that I am able to analyze hands from past games.

1. As a user of the poker equity tool,

I want to be able to simply export a hand analysis

So that I can easily share my results with others.

1. As a poker player,

I want to have access to an archive of commonly played hands

So that I can quickly calculate equity in hands I frequently encounter.

DEFECTS

One major defect found during the testing of the program was that our first iteration of the equity calculation turned out to be entirely inaccurate. This was discovered by running identical scenarios on our software and on existing equity calculation software. This was difficult to test for programmatically, so the defect was found by testing as a user.

This defect led to writing entirely new code from scratch to tackle the calculation in a different manner. The resulting code is ugly and difficult to follow, but it works for our purposes. This is certainly an issue in which more time would have helped.

The GridGUI that allows the user to select hands with a graphical interface had slightly misaligned buttons that needed to be situated. The buttons were also colored in a way that made it difficult to determine whether or not the buttons were depressed. This issue was looked into, though as we used IDE GUI generation, it is somewhat difficult to get things working properly. But these defects are more aesthetic than functional.

GITHUB REPOSITORY

https://github.com/jriese/pokercalc