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 3 DELIVERABLE          Due: July 8, 2015

SPRINT DESCRIPTION

A good amount of the third sprint was spent on refactoring a large portion of the existing code to ensure that the additions to be made will integrate seamlessly. Classes that underwent major refactors include Card, Deck, and CardValue, as well as their respective test classes.

Refactors for these classes included updating comments for each method of the class, as well as integrating CardValue and Deck. This led to substantial code savings in the Deck class, as Deck initialization could now be done in just a few lines of code. Test cases were also more clearly labeled so that the team or possible outside users would have an easier time understanding the expectations and purpose of each method. After the refactoring process, the test suite was reran to ensure that nothing broke in the interim.

During the refactoring, deficiencies in the code base were spotted and listed for future fixes. These fixes, including allowing for simple checking of Card availability in the Deck and a function in the CardValue method to return the integer face value of the Card, were left until after refactoring was complete. After was finished, these features were added into the code base and the test suite reran, again to ensure that nothing happened to break.

Once refactoring and the above feature additions were complete, a few new classes were created to further aid in the eventual equity calculations. The HoleHand and CompleteHand classes were created to represent the different types of poker hands that a player may have during the course of a game. The HoleHand class represents the two cards that are in the player’s hand. The CompleteHand class is our representation of a complete, five-card poker hand.

The PokerTable class was designed to store the community cards that show up on the table – the three-card flop, the turn card, and the river card. The PokerTable class itself is not the most sophisticated class, nor are the HoleHand and CompleteHand classes, but they will play important roles in the overall goal of the program. The Cards in the PokerTable and HoleHand classes will be used to generate the different CompleteHands that each player will be able to create.

Work on a HandGenerator class that will generate the combinations of HoleHands and CompleteHands begun during sprint three, but the logic proved to be complicated and completion and integration of the HandGenerator will need to be pushed back to the final sprint.

Work on the HandVal class continued during sprint 2. This class will be our primary method of evaluating different poker hands by assigning a value to each hand based on the cards in that hand.

Some disagreements arose on which IDE GUI builder to use to build the GUI for the program. Mike built a prototype using Eclipse while Matt decided to put together a more in-depth GUI using the Netbeans interface. This was simple to reconcile as both interfaces were fairly similar and, as Matt was building a more robust GUI, we simply allowed him to continue working on his version.

During code review of the HandVal class, a brief disagreement on how flush hands were evaluated arose from a misunderstanding how multiple flush hands compare to each other. This disagreement was easily remedied by doing some research into hand values. It also gave us a better understanding on the task we faced in general.

Integrating everyone’s work together is a difficult and ongoing process. This is especially true with the GUI design, as using different IDEs leads to different formatting and it’s difficult to interface between them cleanly.  As stated, a lot of this sprint was integrating previous work during the refactoring process. It may be prudent to submit the HandVal class to a similar process, but heading into the final sprint, time constraints may make this an unrealistic goal.

The code review process certainly helped to demonstrate the importance of writing clear code and just how difficult it can be to understand code written by another person. That said, having another set of eyes (one that had not directly worked on a specific set of code) helped us find a few different problems, which will be touched on in the defect section of the document. Our code review process was perhaps a bit more informal than might be preferred. This may be primarily due to our collective unfamiliarity with GitHub. We could also use a more structured method of choosing who does the reviewing as we basically just used a volunteer process. One of us would read over the code submitted by the others and suggest changes and improvements via our Slack group or in person. Some issues found during further code reviews were stored in the Review directory of the Git repo. Changes could then be made prior to merging the branch.

Customer interaction still needs to be worked on and Mike takes the responsibility for the deficit on this front. He had planned to submit a GUI prototype to the customer for feedback, but it took longer than expected to get up and running. Also, our goal from the outset has been fairly straightforward. Unfortunately, it has proved difficult to get a demonstrable product to show the customer as we have not yet fully developed the ability to calculate equity.

Which leads directly into difficulties in writing the code. The HandVal and HandGenerator classes have proved to be more complicated than we might have hoped at the outset. There are simply tons of different possibilities for which to check. For example, notation for just one hole hand can come in multiple different forms and lengths, making parsing those values to create the HoleHand class difficult. The resulting generate method of the HandGenerator class, while not yet completed, seems like it will come together soon, but it will not be pretty code.

USER STORIES COMPLETED

5.      As a user of the poker equity tool,

I want to be able to choose cards using a graphical interface

So that it is simple to choose the cards I want to analyze.

REASONS FOR FOCUSING ON ABOVE USER STORIES

We did not specifically set out to complete the GUI for the user at the beginning of the sprint. In fact, we could have done better in defining which of the user stories we wanted to try to complete at the start of the sprint. While working toward the other user stories in the product backlog, it became apparent that the sheer complexity of the task would not allow us to get much done in the way of completing the user stories. This was disappointing as we thought the work done in the first two sprints would allow these stories to fall more quickly.

We continued to work on building on the groundwork laid in the previous two sprints, with the additions of the HoleHand, CompleteHand, and PokerTable classes. Also, work being done toward the HandVal and HandGenerator classes continued, but these were not completely finished and integrated, and with the resolution of these classes lies the core of the program being worked toward.

The desire to improve communications with customer Professor Go Boom and to have at least something concrete to show for the sprint led to a slight shift in goals during the course of the sprint. It was decided that building a GUI for the user would not only provide the customer with an idea of what we’re working on, it would also allow for easier manual tests of the system.

USER STORY BACKLOG

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.

2.   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.

3.   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.

4.   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.

5.       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.

6.       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.

7.       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.

8.       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.

9.      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

Because of its complexity, the refactoring process mentioned above led to Mike creating a ton of defects that were demonstrated by the test suite. Here, the tests we had written for the different classes really proved to be invaluable. The failure of any one test would allow us to check the changes we had made and either reverse them, or refactor them again to make the changes green.

While creating the CompleteHand class, tests were written to validate the functionality of the class’s getHighCard and getLowCard functions. The test for getHighCard went green easily enough, but the getLowCard method proved more difficult. The test asserted that the low card in the hand should be 4s, but the result ended up as Ah. This pointed to a logic error and, sure enough, we had used a greater than rather than a less than. This also may point to it being a good idea to reverse the values for the cards in the CardValue class. This should be revisited, time permitting.

Code reviews also provided valuable detection of defects. As hinted at before, these reviews turned up at least two major flaws in the code base that will need to be addressed moving forward.

First, while code reviewing the refactored code and the additional HoleHand, CompleteHand, and PokerTable classes, he noticed that none of the classes were actually drawing the Cards from the Deck (making them unavailable to the rest of the program). This was not something that could have specifically been tested for, at least not at this point, as it was simply logic that was not considered at the time of writing the code. Discussions on where and when to make cards unavailable will be important to have to solve our overall problem.

Second, Matt discovered an issue with the way that flushes were being evaluated with the HandVal class. The issue would arise when every card except the lowest card of two flush hands were the same. For example, one hand had KQT54 of hearts and the other had KQT53 of hearts. In this case the first hand would win instead of the two hands splitting the pot. This issue will need to be addressed moving forward to ensure accurate calculations.

One defect with the GUI design was the usage of regular buttons instead of toggle buttons for the hand selection grid.  This led to an entire deletion and restart of the GUI from scratch, as the event listeners all had to be changed.  Fortunately only five buttons had been created for testing so it wasn’t thousands of lines of code to be changed.  Interfacing between the Main and Grid interfaces has proven to be a slight challenge, and needs to be further refined in the last sprint.

We also decided to review some of the code a second time after the merge. Additional review turned up a few more issues that will need to be addressed in the coming sprint. Some of these issues can be found in the Review directory of the Git repo.

GITHUB REPOSITORY

https://github.com/jriese/pokercalc