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 2 DELIVERABLE      Due: June 24, 2015

SPRINT DESCRIPTION

The second sprint focused on laying the groundwork for the overall solution that we are trying to develop for calculating Texas Hold ‘Em style poker equity. We needed to come up with a way to represent the poker cards in the system and allow the user to input what card combinations he would like to test. To this end, two classes were developed – Card and Deck.

The Card class is our internal representation of a single poker card. Every poker card has two intrinsic attributes – value and suit. As such, the Card class allows for the storage of these two attributes. It includes methods for accessing the poker card’s value and suit, as well as a method for printing a String representation of the card.

The Deck class is how we represent the master deck of poker cards. Every deck of poker cards has exactly 52 cards – 2 through Ace in each suit of spades, clubs, hearts and diamonds. When the Deck class is instantiated, the constructor creates each of the 52 cards and stores them in an ArrayList. The ArrayList is then shuffled, thereby creating a randomly shuffled deck of cards for the poker game.

While they only related indirectly to the user stories, two static classes were created to allow for consistency in card representations and user input. These are the Translator and CardValue classes. They do not directly solve user stories, but they will lay an important foundation for the program overall.

The Translator class is designed to accept the user’s input and translate it into useable information for the program. The program will eventually accept String inputs from the user that represent the hands or range of hands that the player and enemy may have. The Translator class accepts this String, trims off any excess white space, breaks it into each individual hand representation and ensures that the input only includes valid characters. The individual String representations of the poker hands are then returned in an ArrayList.

The CardValue class stores constants for integer representations of the cards. While the program overall has not fully implemented this class, it should allow for relatively simple initialization of the Cards and Decks once completed. The CardValue class also includes methods that the Card class can access to get String representations of the Cards, thereby keeping output consistent.

The main program itself was coded with a simple Texas Hold ‘Em rules poker game. This was primarily done to demonstrate the rules were implemented correctly and to demonstrate progress to the customer.

Testing the classes, at least with the static class Translator, turned out to be slightly tricky. The class only has one public method to test, but that public method used private methods to assist in its function. So testing of the private methods was done indirectly through the calling of the public translate method. It took a while to figure out how to determine to test the private methods, but we were able to do so by checking for negative and null values.

After sprint one, our team realized that we had a shortcoming in communication with each other, having relied too heavily on long email chains that were nearly impossible to parse. We decided that using Slack for communication might help out with this area and it has been functioning well. John set up the team in Slack and connected it to GitHub, making it very easy to see when new commits were made to the code. Also, having a consistent chat room made communication simple. Mike, at least, found that having the Slack app on his smartphone made communication very easy and timely. All that said, we could still improve on frequency of communications.

Communication with the customer still needs to be improved. The customer went incommunicado and Professor Laboon agreed to stand in as the customer. Luckily, the designs we had for the project were neatly in line with Professor Laboon’s response on the questions directed toward the customer. Now that the customer issue has been resolved, we should be able to keep the customer abreast of what is happening with greater frequency and get more feedback for the final two sprints.

In general, we decided on a bottom-up approach to creating the poker equity tool. Not many user stories have been addressed in this specific sprint, but the work done on each little piece so far should lead to a domino effect-like take down of user stories in the future sprints. The program is also being designed with an event-driven architecture in mind. This makes sense given that the program relies on the user inputting the hands and telling the program to calculate the equity.

USER STORIES COMPLETED

1. As a player of Texas Hold ‘Em style poker,

I want the program to adhere to the rules of Texas Hold ‘Em

So that I can analyze the appropriate type of poker hands.

1. As a poker player,

I want to be able to choose a hand or range of hands for my player

So that I can compare it to the enemy hand.

1. As a poker player,

I want to be able to choose a hand or range of hands for my enemy

So that I can compare it to the player hand.

1. As a poker player,

I want to be able to customize the pool of community cards

So that I can analyze different scenarios.

1. As a poker player,

I want to be able to customize community cards at different points – flop, turn, river –

So that I am able to completely control scenarios.

1. As a user of the poker equity tool,

I want to be able to type hand ranges using standard poker card notation

So that I can enter any type of hand for analysis.

REASONS FOR FOCUSING ON ABOVE USER STORIES

You will notice that the above user stories do not include the most highly prioritized stories. Those stories are certainly the overall focus of the project, but the groundwork to completing those had to be laid first.

User story three had to be a priority because the program will require the rules of Texas Hold ‘Em poker to be followed so that the program can do its job. We will have failed to produce a valid solution to the problem if the program were to implement rules for another style of poker.

User stories four and five were important to focus on because they are the basis for user interaction. Being able to choose the cards in each player’s hands is integral to the operation of the program. The user is able to choose a hand or range of hands, separated by commas.

User stories six and seven basically follow directly from user stories three through five. The user of the program needs to be able to input the hands of the players, but also to customize the hands on the table. So the user will be able to choose each card available on the table. Those cards will eventually be used by the program to figure out the equity that each player has for that hand.

User story eight was important as it would give a consistent means of representing the cards. The user needs to be able to enter cards in a standard, logical manner. What we decided on (based on our interaction with the customer and playing around with similar software) was to use notation of the form VALUEsuit. For example, the Ace of Hearts would be represented Ah. The Translator class helps to allow the program to accept user input that is not correctly capitalized, and translates it into the correct standard notation.

Overall, all of the above user stories were of high priority because they help build the foundation of the program. With our bottom-up approach, getting these individual pieces together will eventually allow the program to come together as a whole.

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.

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.

1. 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.

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

Overall, the testing involved helped keep defects low. The main program was not rigorously tested due to time constraints, but this will be corrected in a future sprint.

When coding the CardValue class, unit tests helped to find some defects. One use defect was passing a null value to the getCardValue methods. The unit test testGetValueNull() was able to determine what would happen when the method was passed a null value. It reproducibly threw an error. The solution we came up with was to have the method return a negative value, namely -1, when the method received a null value. In this way, other areas of the program will be able to account for null value errors.

Another defect in the CardValue class was what happened when you passed it a value that was not a valid representation of a poker card. We were able to reproduce the error by giving it values not associated with card values or suits – in the subset {AKQJT98765432HDCS}. The expected result was to return -1, but the observed result was returning an uninitialized value. The testGetValueInvalidValue() test was written to check for invalid values and the method was updated to return -1 when an invalid card value was sent to it.

GITHUB REPOSITORY

https://github.com/jriese/pokercalc