

James Fitzgerald
Eleazar Fernando
Jonathan Lau

Final Project Paper – Texas Hold'em

Abstract

The purpose of the project is to provide an interface for 2 players to compete in a game of Texas Hold'em through network connection. Texas Hold'em poker has become the most popular form of poker recently and has enjoyed an increased fan base thanks to the media. We feel that we could create a user-friendly two player interface that can be easily accessed and enjoyed by the common Windows user. Some key terms to bear in mind (which will be explained more in the following pages) include: **hand evaluator function, packets, the Hand, Player, Gameplay, Deck and Card classes.**

Our goal is to create a user-friendly interface for two players to compete in a game of Texas Hold'em poker through network connection. We feel that it would be a great idea to have a program that can fully capture the essence of the real-life game through an easy-to-use interface. Keeping this in mind, we defined specific program capabilities and developed the program according to these guidelines. First of all, we decided that a Graphical User Interface should be used. A colored and graphical-intensive interface not only appeals to all audiences visually, it also makes game play much easier and enjoyable. We decided to use the form utility in Microsoft Visual C++ 2003 to create the GUI that displays the player's hand, the community cards and chips on the table. As in a real-life Texas Hold'em game, the user of our program has the option to call, raise or fold during his/her turn, facilitated by the respective 'call', 'raise' and 'fold' buttons. A betting counter is also present to keep track of the amount of money each player has and the size of the pot in each game. To make the program even friendlier to the users, we have decided to implement an evaluator that automatically determines the winner after all the cards are dealt. Finally, to further encourage the spirit of competition among two players, we have implemented a chat function for the players to exchange messages (read: trash-talk).

The target platform for our program is Microsoft Windows since we developed the whole program with Microsoft Visual C++ 2003 and established connection using Winsock. The potential user can be anyone that enjoys the game of Texas Hold'em who uses a computer with MS Windows.

With the goals and capabilities of the program defined, we started developing

the program accordingly (please refer to the UML diagram). We created five main classes that interact with each other. First of all, we have a Card class that, as its name implies, represents one distinct playing card out of a deck of 52 different playing cards. Each card object holds its respective rank and suit which can be retrieved by the member functions `get_rank()` and `get_suit()`; the graphical image of the card itself is also stored in the Card class. We then move on to the Deck class, which consists of 52 private card members from the Card class, all stored in a vector. The Deck class also contains a call to `random_shuffle()` function (from the algorithm header) which, randomizes the order of cards in the vector (shuffles the deck) when the deck is created.

After we have created a deck and card objects, we move on to creating a Player class. The Player class interacts directly with the Hand class and the Gameplay Class. Basically, two Player objects are created at the start of the game to represent the two competing players. Each player has a 'hand', a list of pointers that point to their respective card objects (the private member `my_hand`). The private member `money` keeps tracks of the money a player possesses while `next_player` points toward the upcoming player. The Hand class contains functions that represent every winning combination in a poker game (e.g. straight, flush, etc.). The private members of this class include a map that references the type of winning combination to the actual combination of cards, a list of cards, as well a variable `besthand` which holds the type of the best possible combination from the specific player's hand. The Gameplay class, on the other hand, contains all the functions needed for the game to flow properly – the functions for dealing cards (`playerdeal()` & `tabledeal()`), the hand evaluator as well as the `winner()` function. Private members of the Gameplay class include a vector containing pointers to the two player objects, a pointer to the deck object, as well as socket elements necessary for connection.

A sample flow of the program would be as follows: The Deck is shuffled by randomizing the vector of card pointers using the `shuffle()` function. The `playerdeal()` function is then invoked, and two card pointers (pocket cards) are pushed into each player's `my_hand` list using the `add_card()` function. A betting round then occurs with the `make_bet()` function. Money is transferred from the player to the pot using the `take_money()` function. A player has the option of calling, raising or folding. Assuming that both players remain in the game, 3 cards will be dealt onto the table using `tabledeal()` (the flop) and another betting round would occur. This process will occur similarly two more times, with two additional community cards (the turn and the river) being dealt and pushed into the players' hands. Finally, assuming that neither player has folded, the `hand_evaluator()` and `winner()` functions will be called and the winner will be determined and be rewarded with the pot added to his/her

account using the `add_money()` function.

Facilitating communication between the two player terminals was a major challenge for this project. We adapted the client/server model and used Winsock for connections. When the program is started, one user sets up as the server; the other player (the client) then connects to the server by entering the server's IP address. The chat function, the action of the previous player (and the amount of the bet) and the winning conditions all require connectivity between the server and the client. The transfer of data between the terminals is facilitated by a custom protocol implemented in the packet class. A packet consists of the following contents: the first byte denotes the type of the packet, i.e. chat message, bet amount, etc. The second byte acts as the sender while the 3rd to the 7th bytes show the size of the packet's data portion. The actual data is contained from the 8th byte onward.

The two major data structures that we have utilized in this program are maps and lists. The most prominent use of the map structure was to support the hand evaluator function. The hand evaluator is a key component of our project. As a private member of the Hand class, the map structure we implemented has keys of `E_TYPE` (the type of trick e.g. flush, pair) and values of `list<CARD*>` (combinations of card pointers that constitutes tricks). By utilizing the hand evaluator functions (`straight()`, `flush()`, etc.) in conjunction with the map, the possible combinations (values of the map) are compared with the player's current hand and the type of hands present are evaluated. The best hand (hand with the highest rank) is then stored in the `besthand` variable and both players' `besthand` variables are compared to determine the winner.

Basically, we felt that the map structure was the natural choice in this case since the ability of having a key-value pair referencing was important for the evaluation and comparison of the rank of the current hand. We considered alternatives such as sets but did not find it feasible. A set of pointers stored in a set that points to a vector/list might serve our purpose but such implementation is not practical since it would be overtly tedious; also, unlike maps, sets do not allow duplicate keys which are necessary for the correct implementation of our evaluation table in this case.

The second data structure we utilized was the list. The main application of the list in our program was as a medium to hold the 7-pointer long hand of each player. While vectors and lists share a lot of similarities, we ultimately chose to use lists for a number of reasons. First of all, we decided to use iterators as the means of accessing data. The `stl` list class contains more built-in functions regarding the use of iterators (such as the `splice` function) than its vector counterpart. Although we did not, in this context, utilize all built-in functions, the potential utility such functions provide was attractive. Secondly, compared to the vector, the list is much more efficient in inserting and deleting data at arbitrary positions. While the general insert and erase

operations in a vector have a running time $O(n)$, the list has a running time of $O(1)$. Finally, the list class provides additional functions for manipulating elements in the front of the container, i.e. `pop_front()` and `push_front()` which the vector class lacks.

In conclusion, we have learnt a lot in this project. Among them are the effective use of lists and maps, making use of the form utility and graphic display functions in Microsoft Visual C++ as well as how to construct a server/client connection using Winsock.

References

- 1) Winsock Programmer's FAQ
<http://tangentsoft.net/wskfaq/>

Biographies



Jonathan Lau - Senior Science-Computing (not to be confused with Computer Science) major. Still has no idea why Data Structures is part of his major's requirement but did end up learning a lot from this class.



James Fitzgerald is junior computer engineering major. He is from the great state of Rhode Island and enjoys data structures, for breakfast or for lunch. He one day hopes to do product development at an embedded computer systems company, preferably his own.



Eleazar Fernando is a junior computer engineering major from the mediocre state of Florida. He enjoyed this data structures course as it helped him with his programming skills. He one day hopes to be a fireman.