Marco Chiang

12/6/12

ECE 373

Project 2

Go Fish Implementation for Project 2

New Go Fish features:

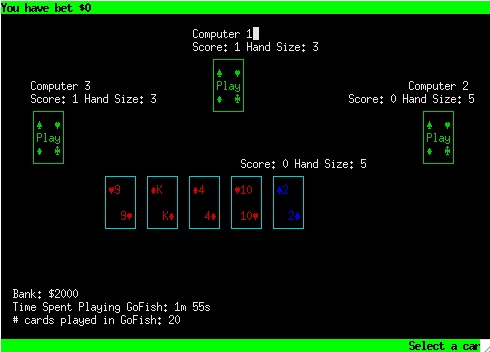
* Real time display of time spent in game.
* Banking system.
* Number of cards played.
* Advertisements.
* Partial Resizing (with bugs)
* Quit game in go fish menu

Figure 1: main menu screen



Figure 2: Go Fish Menu



Figure 3: In game play 

Implementation:

1. The real time display in game – I create an instance of the Timer class which I wrote and once I select the number of players I start the timer using timer(obj).startTimer(); Then when the game starts I display with mvprintw(posx,posy,str) where the str variable has the value timer(obj.endTimer(); This refreshes all the time and shows the difference from when the clock starts and when it ends.
2. The banking system (declared an extern bank object)– Whenever you click on Go Fish from the main screen it asks for an initial bid. If you win you get 3 times’ the amount of what you bet. If you lose, you lose 3 times’ the amount of what you bet. When you get into the Go Fish menu you’ll be asked to bet another amount, which is determined by round. For example, if you ask for a card and you “go fish” you subtract your in-game betting amount. If you get a card that is a pair, you win the in-game betting amount. The bank is displayed on the screen.
3. I wrote a card counter class with a struct, which includes int cardCounters for all games and I declare an extern object of the cardCounter class (links with the one created in main). Whenever the deck deals a card to a player I increment the card counter for go fish. For example if you start a new game with 4 players the # of cards played will be 20 (5 cards per player \* 4 = 20). By the end of the game the card counter will have reach 52 and the deck will be empty. The main menu will display the accumulative count for # of cards played in go fish and so will the in game one.
4. John Fuller wrote an advertisement class that reads in a text file and sequentially displays each line until the end of file then it re opens the text file and starts back from the top. I update the advertisement after every round.
5. Resizing – the in-game has some resizing compatibility include redrawing computer cards, scores, hand size, top banner, Bank, Time spent playing, and # of cards played. This is the feature that has bugs in it. If you try to resize the terminal it will erase the portions of the game not mentioned and the bottom banner will get distorted. Sometime the game reaches a seg-fault, which is odd since it seems to be random occurrences.
6. The user has the option to quit from the go fish menu game if they don’t want to play anymore. However, if the user has already entered a bet, they will lose money as if they lost the game.

Testing Methods:

- I started creating test case directories for my game including makefiles and test scripts for the classes I wrote (CardCounter.cpp GameTimer.cpp and GoFish.cpp). My plan was to have the tester run testscript\_(cardcounter, gametimer, or gofish).sh and that would compile the make files specific to each class. Then it would copy the input files I planned on creating into the testing directory and compare the output results with the expected output solutions in AutomatedTesting/testcases\_(class name)/\*.txt.

Conclusions:

Other than the resizing feature, the Go Fish has no noticeable bugs that I can think of. Integration was simple enough since John and I split up the new features. He took bank and advertisement and I took card counter and timer. We then just included the header files for the classes and implemented the necessary nitty gritty details like which instances of the object class to pass through to our games.