Anne Schwartz

Final Project

12/18/13

Problem Specification and Design Document

1. For my project, I created a program for a user to player Go Fish against a computer player. In order to simulate the computer’s turn, I created a memory of previous requests by the player so that if the computer obtains a card with a rank that matches one in the memory, the computer will ask for that rank.
2. For this game, I have a menu with three different difficulty options. These difficulties are based on the amount of previous ranks requested by the player, that the computer remembers. After the initial deal in which each player gets 7 cards and the cards are sorted by orderC and orderP, it is the player’s turn. On the player’s turn, he or she is asked to enter the rank he/she would like from the computer (but this rank must be in his or her hand when the player requests it). If the computer has the requested rank, the card(s) is/are passed over to the player’s hand and the cards are sorted in order of rank (orderP and orderC). If the computer does not have the rank, the player is prompted to Go Fish. When the player clicks the ‘Draw’ button, the goFishP method is called which adds a card to the player’s hand. After each sort, the player’s hand is also checked for sets of 4 of a rank (checkForPointsP). If the hands does have all 4 cards of a rank, those cards are removed from the hand and a point is given the player.

For the computer’s turn, ranksFirstTime is called to create lists of all the ranks in the computer’s hand and an options list, which contains any ranks in the computer’s hand that matches the memory. Then, in cTurn, the computer either randomly choses a rank from options or if that list is empty, it chooses a random index in the rank list to request the rank in the index. After requesting a rank, the rank is removed from the list so that the computer will not request the same rank in one turn. If the player has the card, it is passed to the computer’s hand and the cards are sorted and checkForPointsC is called to check if the computer has any sets of 4 of a rank. When the player does not have the requested rank by the computer, the player instructs the computer to GoFish, which then calls the goFishC method to add a card to the computer’s hand. This method then calls the orderC method and checkForPoints.

The game is simulated by calling pTurn, which ends when goFishP is called. It then calls ranksFirstTime and cTurn for the computer to have a turn until goFishC is called, at which time it then returns to goFishP. This loop continues until the checkForEnd method finds that the game has ended, at which time self.checkEnd is returned as True and the game ends and the winner is announced. The checkForEnd method determines the game has ended if the draw pile is empty. If a player’s hand is empty, the player get’s another card from the deck (as if they are told to ‘GoFish’) and the game continues.

The lists of cards in each hand and the deck are universal variables (self.pHand, self.cHand and self.cards). By having them as universal variables, I can change the lists in different methods.

c. To test this program, I used a smaller deck so that the test games I played would be faster. I found instances that could cause the program to crash, including invalid rank entries and the player saying he or she has a rank requested by the computer that he/she does not actually have. I struggled to program the end of the game because many methods are called within method calls in the play method. This made it difficult to return the checkEnd to the proper place for the game to end. My original attempts kept displaying the winner but then trying to continue running the game from where the checkForEnd method had been called. I then discovered that by returning checkEnd, the methods would stop running and the while loop in the play method would acknowledge the end.

For a future version, I would make it so that the player can quit out of the game window in the middle of a game, rather than having to wait until the end. I would also find a way to have less happening in each method.