Manual

Five Crowns C++

Author: Frank Ryan

Course: CMPS 366 01

Submitted: 10/1/2019

**Bug Report:**

There seems to be one possible bug that I believed I fixed, but there have not been many scenarios to test my solution. The bug is found when the player asks for help when trying to go out. When the human asks for help here, and his hand contains a wild card for the round that can also assist in a run, then the computer finds a run that is incorrect. Here it an example of one time it happened:

Human Hand: 9H XH QH J2 KH

When the human asked for help to go out, the computer recommended this:

The computer found a run of … 9H QH XH QH KH J2

I believe I fixed this bug by adding a function that checks for any wild duplicates in the hand. The function’s name is searchUsedWild.

Although I believed I fixed this, I have not created many opportunities to truly test this case.

**Feature Report:**

Missing Features:

All features were implemented.

Extra Features:

I did not add any extra features.

**Description of the data structures/classes:**

The primary data structure I used in this program was vector. To be more specific, I used vectors of type <Card>, which is an object of class Card. To understand this more, I will now discuss the functionalities of my classes. (Total of 7 classes)

Card class: Used to create card objects, primarily inside the Deck class constructor. In each card object, there are three properties/parameters. The first is the card’s value, the second is the suite, and the third is the deck number. In this class I have the following functions, which are mostly getters:

string getVal: returns the value of the card as a string

int getNumValue: returns the value of the card as an int

string getSuite: returns the suite of the card as a string

int getDeck: returns the deck number of the card

Deck class: Used to deal cards and search cards. In the deck constructor, I create every Card object in each deck. I put all these cards into a private vector<Card>. In this class I have the following function:

void shuffle: shuffle all cards in the private Card vector.

void printDeck:print all cards in the private Card vector.

vector<Card> dealCards: deal out cards in the private Card vector.

vector<Card> drawPile: create draw pile as Card vector.

vector<Card> discardPile: create discard pile as Card vector.

string printCardHand: prints out a Card vector.

bool searchDeck: searches the deck for an inputted Card object.

Game class: Meant to be created and called in the main function of program. Displays the main menu and proceeds to iterate through rounds of the game by calling the Round class. This class has the following functions:

void startGame: should be called in main, begins the game of Five Crowns. Can start a new game, load a game, or exit program. If starting a game, calling the Round class and iterate through rounds. If loading a game, creates an ifstream to take in text file and then call Round class functions.

Round class: Used to start rounds in a game. Contains functions that handle one round at a time. The Player, Human, and Computer objects use this class to interact with the game. This class has the following functions:

int startRound1: simulates a single round of a game. Calls deck functions to receive cards for players and piles. Then continues to call Human and Computer object to simulate their turns. Waits for a Human or Computer object to go out. Once that happens, this function ends.

int startSavedGame: similar to startRound1 function. Called when a player wishes to load a game in the Game class function startGame. Places all data of text file into correct storage, then simulates one round. Once a player goes out on this round, the startRound1 function is meant to be called and iterated until the game reaches past 11 rounds.

Player class: Class that contains functionalities of a player in the game. It is the parent class of Human and Computer classes. Many computer functionalities are placed inside this class in order for a Human object to use if he/she wishes to ask for help from the computer. This class has the following functions that can be called from Computer and Human objects:

vector<Card> dropCard: drops a card from a Card vector.

vector<Card> pickDrawCard: picks the top draw card for a player.

vector<Card> pickDiscCard: picks the top discard card for a player.

void printCards: prints out a Card vector.

vector<Card> checkComputerGoOut: checks if a player can go out with their hand. Calls Player functions checkBook, checkRun, checkOutPossibilites, checkOutThree, checkOutFour. This is the largest function of the Player class. It considers the round number, and determines what sizes of books and runs to look for. It returns a null hand if the player can go out.

vector<vector<Card>> checkBook: used only in checkComputerGoOut. Checks for all books of a certain size and places them into a double Card vector.

vector<vector<Card>> checkRun: used only in checkComputerGoOut. Checks for all runs of a certain size and places them into a double Card vector.

bool checkDuplicates: checks if a two Card vectors have the same card.

int checkOutPossibilities: checks if two double Card vectors contain a duplicate card. Uses checkDuplicates function.

int checkOutThree: checks if three double Card vectors contain a duplicate card. Uses checkDuplicates function.

int checkOutFour: checks if four double Card vectors contain a duplicate card. Uses checkDuplicates function.

bool searchUsedWild: searches a Card vector if it contains a particular wild card.

vector<Card> checkPossibleBook: checks for the largest book inside a Card vector.

vector<Card> checkPossibleRun: checks for the largest run inside a Card vector.

Human class: Inherits from the Player class. Contains functions for a human player. Calls Player functions when the human player asks for help. This class contains functions:

vector<Card> startTurn: begins and goes through a complete human player’s turn.

int searchDiscCard: searches the human’s hand for an inputted card to discard.

bool checkHumanGoOut: called when the human wishes to input cards to go out.

bool searchGoOutCards: searches the human’s hand for all inputted cards to go out with.

bool checkRun: checks to see if inputted cards are a run.

bool checkBook: checks to see if inputted cards are a book.

int startLastTurn: begins and goes through a complete human’s players turn once the other player goes out.

vector<Card> humanLastGoOut: called when the human wishes to go out or put down cards before the round is over.

bool checkEqualHand: checks is two Card vectors are equal.

Computer class: This class inherits from the Player class. It is needed in order to have a computer player. It mostly utilizes functions from the Player class, but also has important functions of its own. The functions it has includes:

vector<Card> startTurn: this function goes through a computer player’s turn. It uses functions in the Player class to determine its moves. This includes using functions: checkComputerGoOut, checkPossibilities, checkPossibleBook, and checkPossibleRun. In terms of logic, the computer looks to add to his largest book if he can, otherwise, he adds to his largest run if he has one. The checkComputerGoOut determines whether the computer goes out or not.

int startLastTurn: this function is very similar to the startTurn function in the Computer class. However, at the end of this function, the leftover card points are added up and returned.

int checkFinalHand: this function looks for any books or runs in the computer’s hand before it goes out.

**Log:**

Over the course of three weeks, I was able to keep to the recommended milestones for the project. In the first week, I completed the basic classes and was able to go through all the rounds with all functionalities of the human player. By the end of the second week, I finished the computer class and its logic, as well as serialization. In total, I spent more time on the project during the second week than any other. The third week consisted of error checking and documentation.

Milestone #1 - 9/17/2019

**In this milestone, I briefly outline the creation process of my classes and their functions. As of now in my project, I have created the overall layout of the game, as a human can now play against a computer and complete a full game. However, I am now working on the computer’s logic, which has proven to be difficult.**

After planning the layout of the classes, I started creating my project the weekend it was assigned. I decided to start off by creating the Deck and Card classes to initialize all cards and two decks. I then went onto the Game and Round classes to loop into 11 rounds of the game, and finally created the Player, Human, and Computer classes to be used inside the Round class.

Initially, in my Round class, I had a function called startRounds(), that would loop through all rounds of the game. However, I decided that it would be best to loop through round in the Game class, so I then changed the startRounds() function to startRound(), which only dealt with one round for the game, to be called 11 times. I also decided it would be best to leave the humanPoints and computerPoints inside the Game class as well, which will be added to inside the startRound() after each round is over (This is by passing humanPoints and computerPoints as a parameter for startRound().

Inside my Human and Computer classes, I have startTurn() and startLastTurn() that will be called inside the startRound() function. startTurn() is called when the other player has not gone out yet, and startLastTurn() is called when the other player has gone out. I first decided to create all functionalities of the Human class and work on the Computer class later. The startTurn() function in the Human class goes through all options that the human player will have when playing the game. It first asks if the player wants to make a move, ask for help, quit, or save the game. So far, I have implemented the functions make a move and quit the game. I now created an input string that asks for the user to either pick up from draw pile or from discard pile. From this, I have created two functions in the Player class, one called pickDrawCard() and one called pickDiscCard(), which can both be used by the Human and Computer class. After the user picks to choose from discard or draw, I then ask the user to discard a card. The user input the cards he wants to discard, and I have a function in the Player class that searches for that card in his hand and puts in on the discard pile.

From here I made an option for the user to go out, and if he wants to, I take in his input on runs/books in his hand and pass them to a goOut() function in Human class, which analyzes his input and determines whether he inputted either books or runs. This function took quite long compared to all other function creations I have made previously, primarily because I was not sure how I wanted to create the algorithm to check for runs and books. Finally, if the user’s input is correct, then the winningCondition parameter in Human’s startTurn() function is changed to 1, and thus the Round class recognizes that and calls the computer’s startLastTurn(). In both startLastTurn() functions, I add up the points from any cards that do not equal a run or a book, and return that integer to the Round class, which is then sent back to the Game class to add to the computer/human variables, and the next round then begins.

All in all, I have made decent progress with my project so far, as I now only have to work on the Computer class functionalities, and serialization. I have also made sure to error check all my code so far, such that a user’s input cannot crash my code. The hardest part of the Computer’s logic is evaluating his hand and determining whether or not it can go out, as there are so many possibilities. So, as of a couple days ago, I have been experimenting with different algorithms to achieve this.

Milestone #2 - 9/24/2019

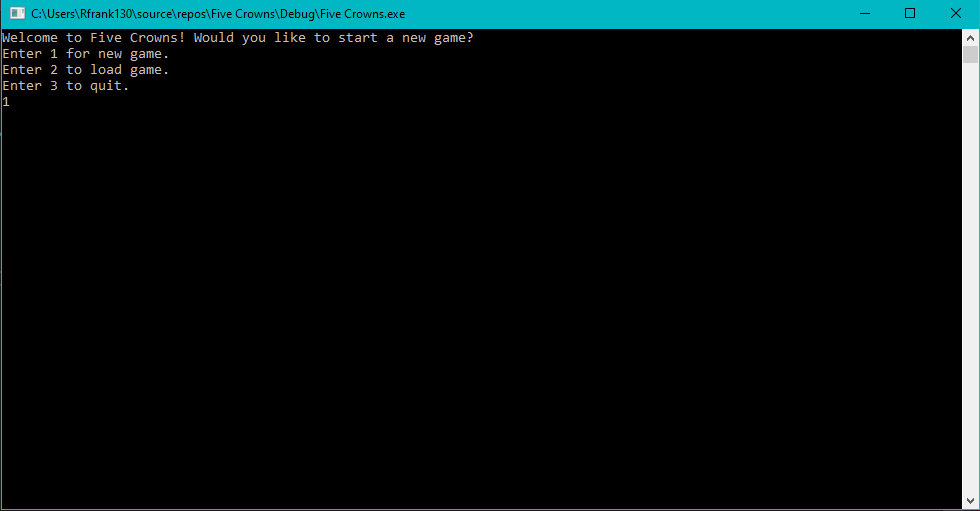
Now that we are two weeks into the assigned C++ project, I seem to be on track with my milestones. Over this last week, I have completed the Computer class and his logic, which gave me the most difficulty. I had a lot of planning of the Computer’s algorithms, and many failed. I originally tried to create a recursive algorithm for the Computer’s goOut() function, but I ran into too many problems. I eventually decided to take a different approach, and put all the Computer’s hand’s possible books and runs and placed them inside vectors. I then compared all possibilities that the hand has, checked for any duplicate cards in each possible run and book, and if there were no duplicates, the Computer player goes out.

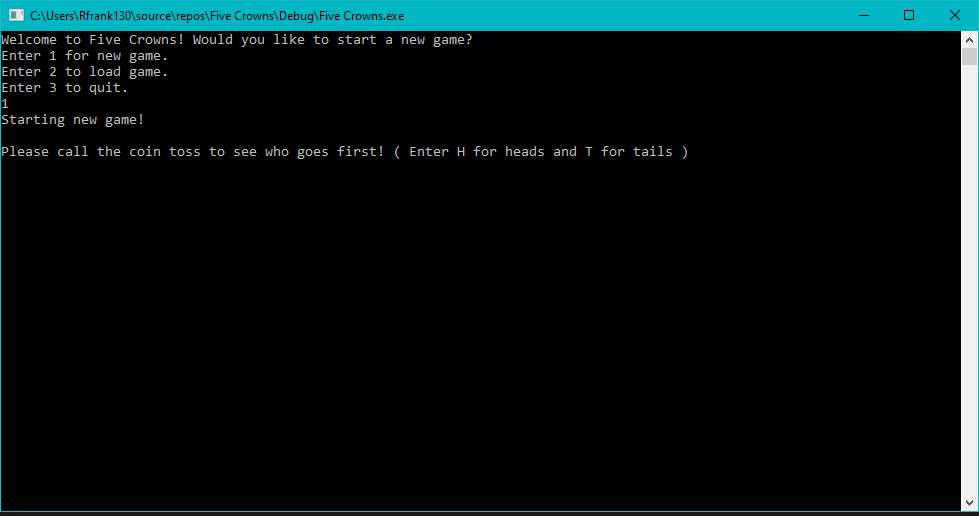
So, I check these runs and books depending on which round it is. As an example, since the players are dealt 3 cards on the first round, I only check for runs and books of 3. However, when the round is 4, where 6 cards are dealt, I must check for runs and books of 6, but also check for books of 3, and runs of 3. I then check these combinations, which are: book of 3 and book of 3, book of 3 and run of 3, and run of 3 and run of 3.

This is the main functionality of my Computer’s goOut() algorithm, and seems to work the best compared to my others. Along with the Computer’s logic, I was also able to finish the serialization part of the project, which did not give me too much trouble. Now that those two tasks are down, all I have left to do is the human Help Mode, which will essentially be the human player calling some Computer functions. Once this is done, I will be onto the documentation aspect of the project, and I should be able to complete my project by the first deadline on October 1st.

How to Run the Program: In order to begin the program, create a Game object and have it call startGame(). This will begin the program and bring you to the main menu.

**Screen Shots:**

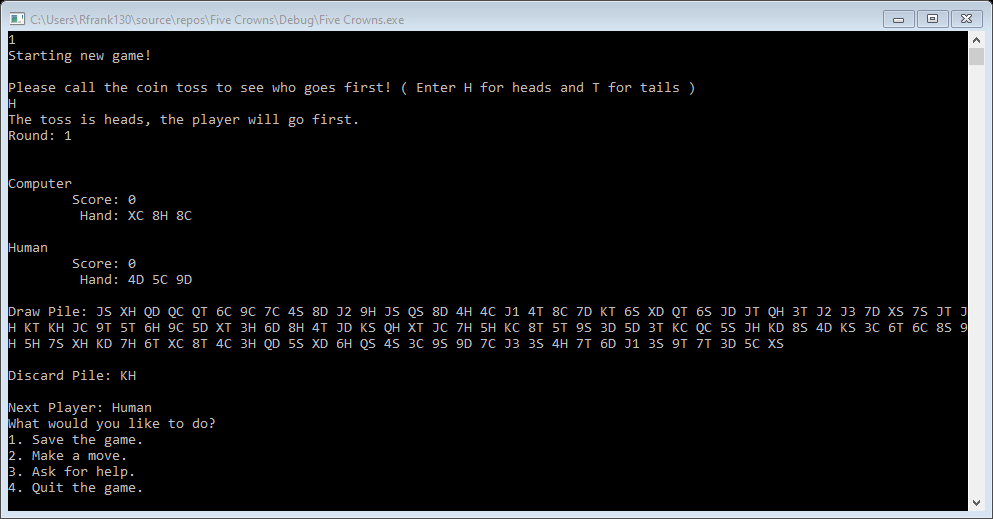
Human Turn without Computer Help: Starting the game at the main menu. To begin a new game, enter “1” into the command prompt as such.

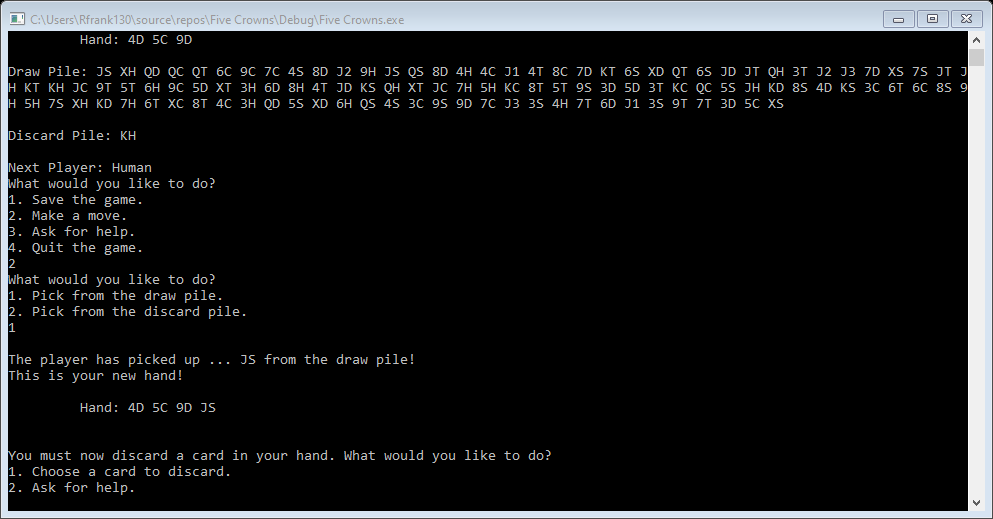
The game will now ask you to call a coin toss to go first. Enter either “H” for heads or “T” for tails.

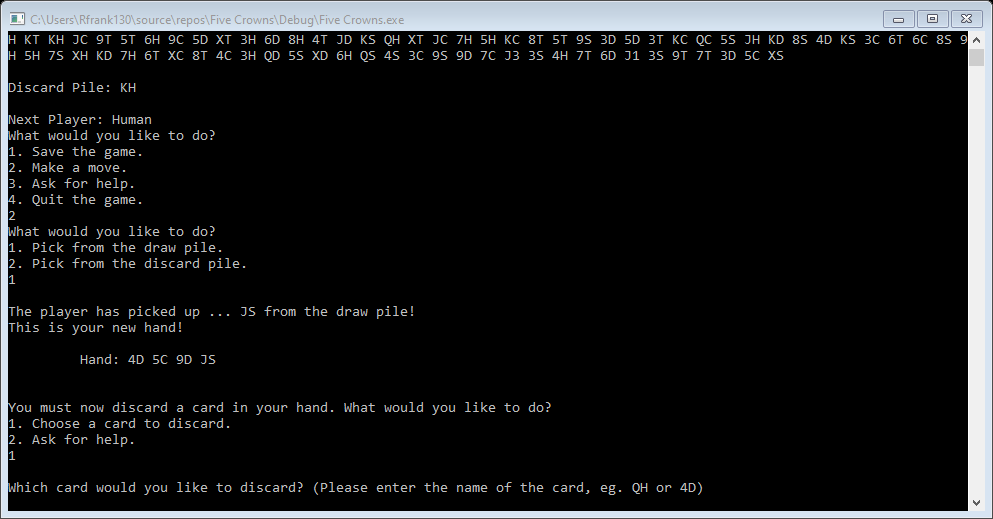
After calling the coin toss, the game will begin by stating who won the coin toss and will go first. Then, the cards will be dealt to each player, and the draw and discard pile will be displayed. This will be the main screen for the game as each round continues.

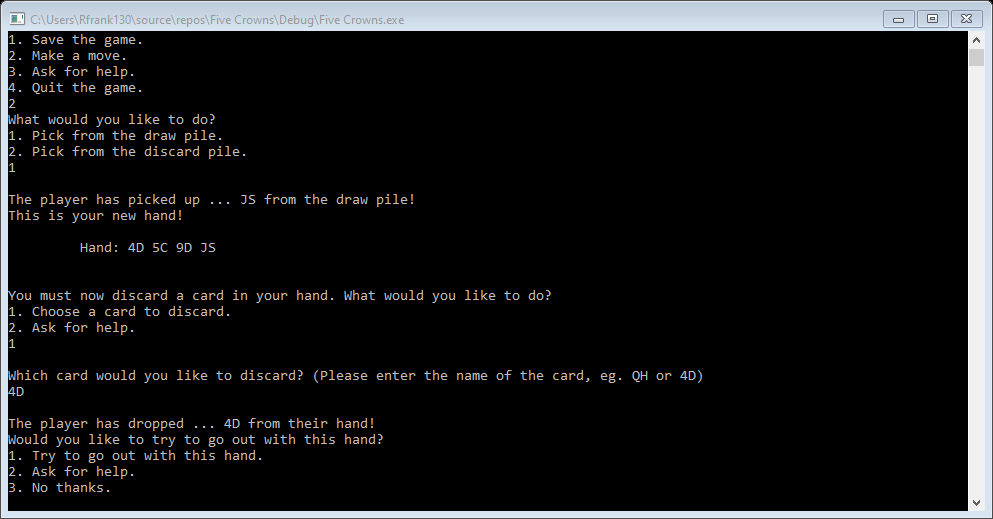
On this screen, you will be prompted to enter what you would like to do. Your options are:

1. Save the game, 2. Make a move, 3. Ask for Help, 4. Quit the Game.

In this part of the manual, we will continue without asking the computer for help. The manual will later display what to do when asking the computer for help.

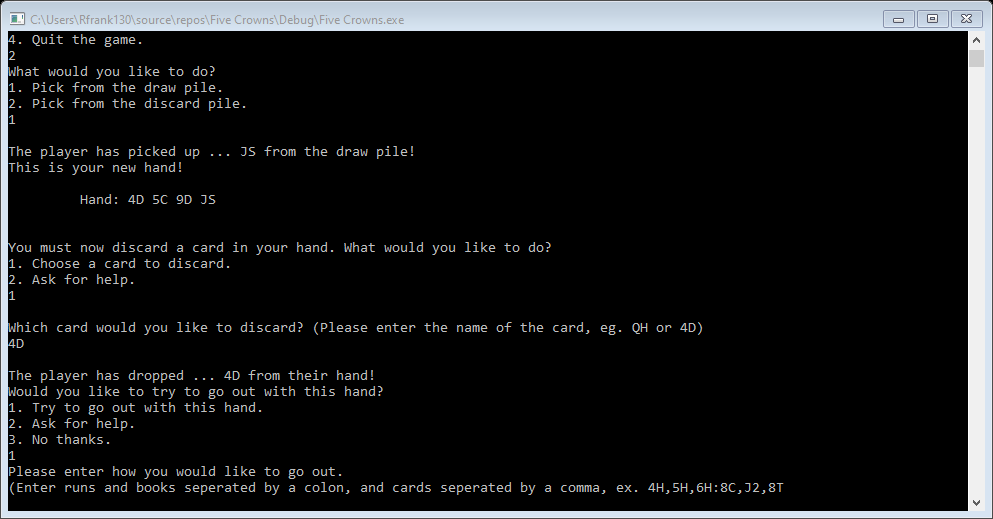
Now, we just enter “2” into the command prompt to make a move. The game now asks if we would like to pick from the draw or discard pile. I will enter “1” for the draw pile.

The game will now prompt the human to either choose a card to discard from their hand, or ask the computer for help. Since we will cover the computer help later in the manual, we will select “1” to choose a card to discard.

The game now asks us to enter which card we would like to discard from our hand. From this, I will discard “4D” from my hand.

The game now asks the user three prompts.

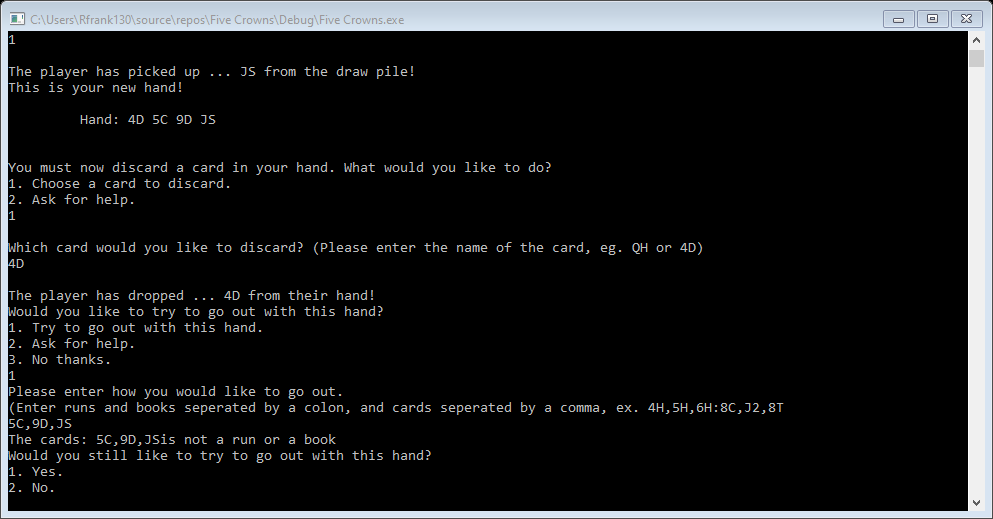
1.Try to go out with this hand, 2. Ask for help, 3. No thanks.

Say we enter “1” in the command prompt to try to go out.

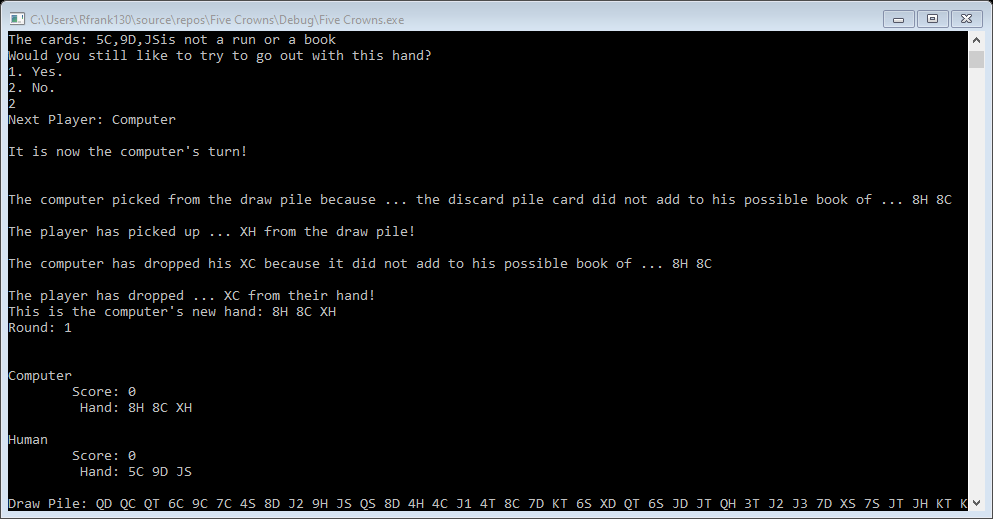
The game now asks us to display our combination of runs and books. To do this, we must separate our books and runs by a colon, and our cards by a comma.

Ex. 5D,6D,7D:8H,8C,8T

We will enter cards in our hand now, but they will not be a book nor a run.



The game does not recognize this as a book or run, so asks us if we want to enter cards again or say no to end our turn. Because entering “1” for yes would bring us back to the other step we just went through, we will enter “2” for no and end our turn.

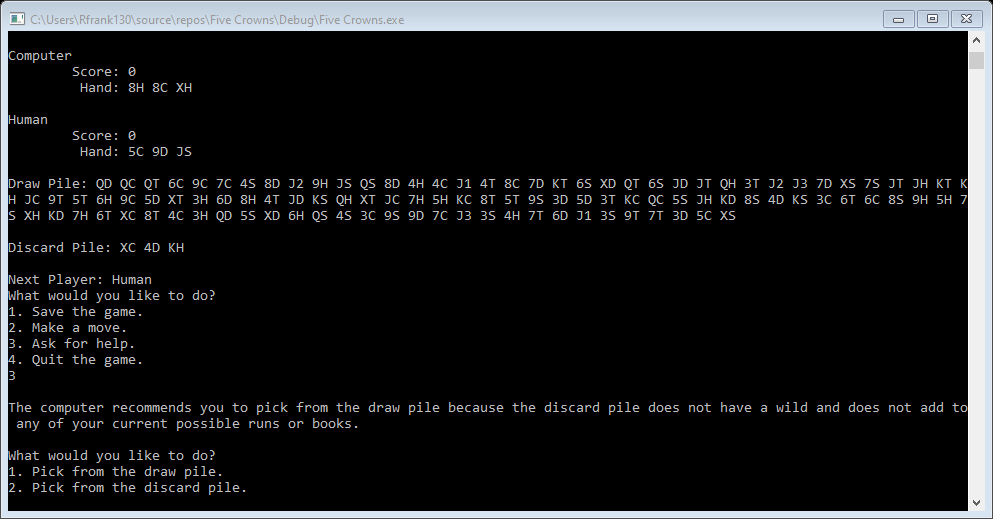


Our turn is now ended, so the computer takes his turn.

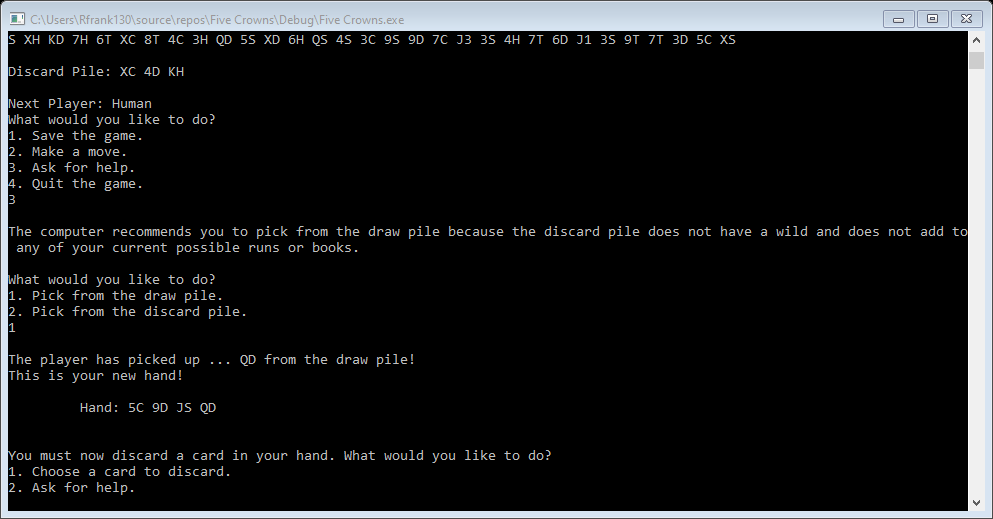
Asking Computer for Help

We have now gone through a whole turn for the human player, but now, we will show what to do when the human asks for help.

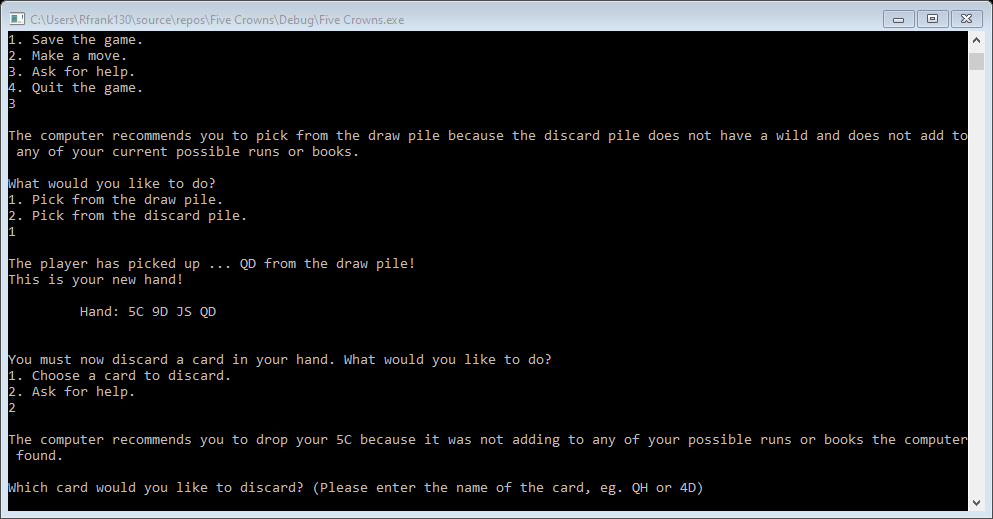
The first time a human can ask the computer for help is at the main screen of the board game. The human can ask for help by entering “3” in the command prompt.



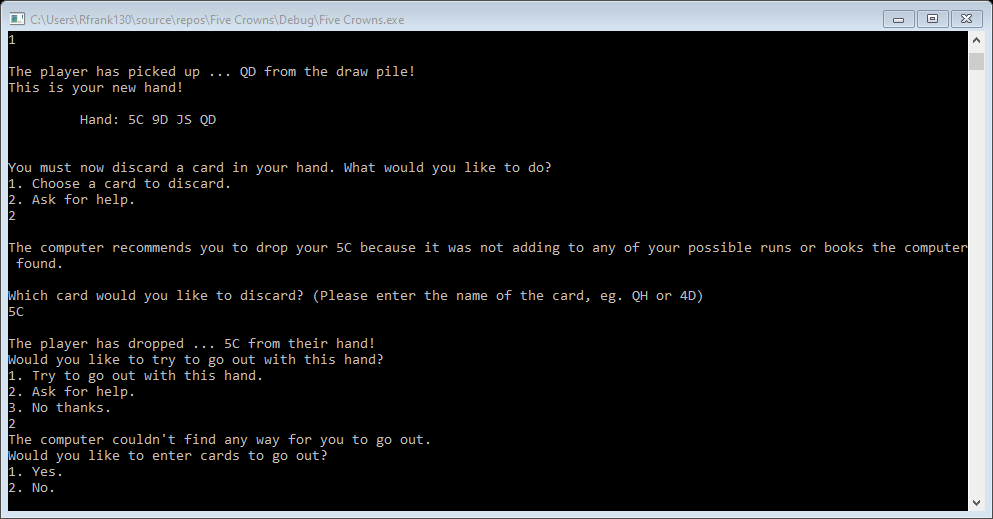
By entering “3”, the computer now gives its recommendation. In this example, the computer tells us to pick from the draw pile. Now, by entering “1”, we can pick from the draw pile.



We have now picked from the draw pile and have received QD. We can again ask for help by entering “2”. This will recommend a card to discard.



In this example, the computer recommends us to drop the 5C from our hand. So, as noted before, the game brings up a screen to enter a card from our hand. I will enter 5C into the command prompt. The game will then give us a screen to ask for help, where I will enter “2” into the command prompt.

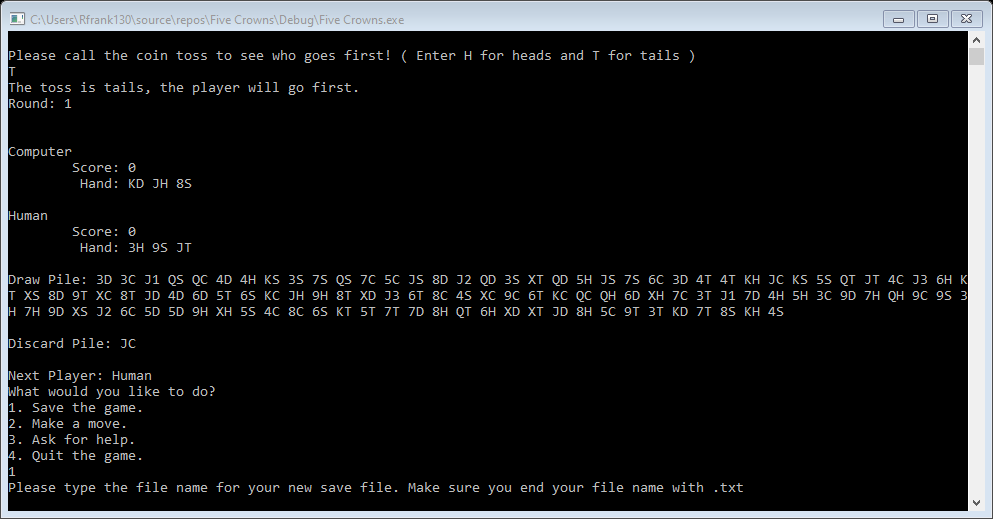


After entering “2” to ask for help to go out, the computer has found now way for me to go out. It will then ask if I would like to try to go out or not. If I decide to try to go out, it will give me the screens we have noted previously in the manual. If not, the game will end my turn and move to the computer’s turn.

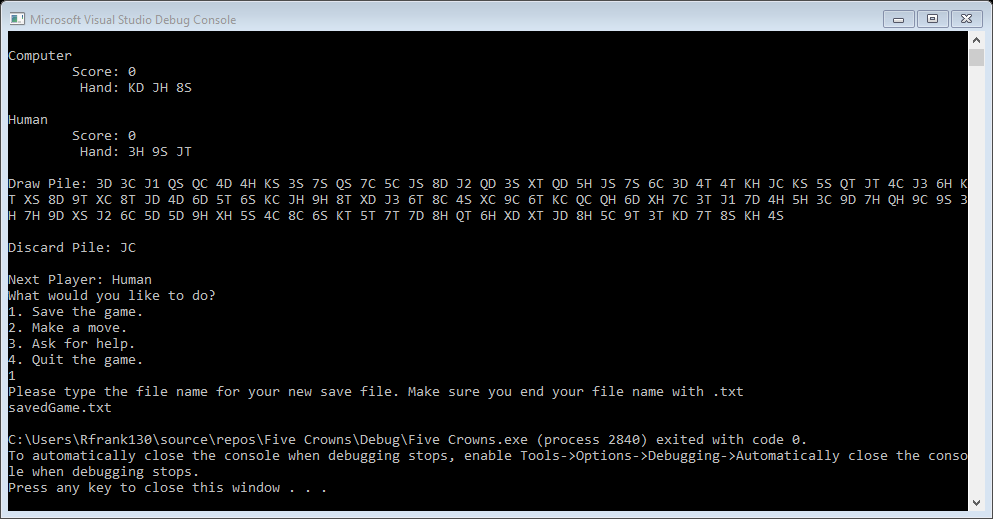
Serialization: Now we will go over how to save a game as a text file. We will then show you how to load a saved text file.

Saving a game…

At the beginning of a human’s turn, the human can decide to save the game as a text file by entering “1” into the command prompt, as such.



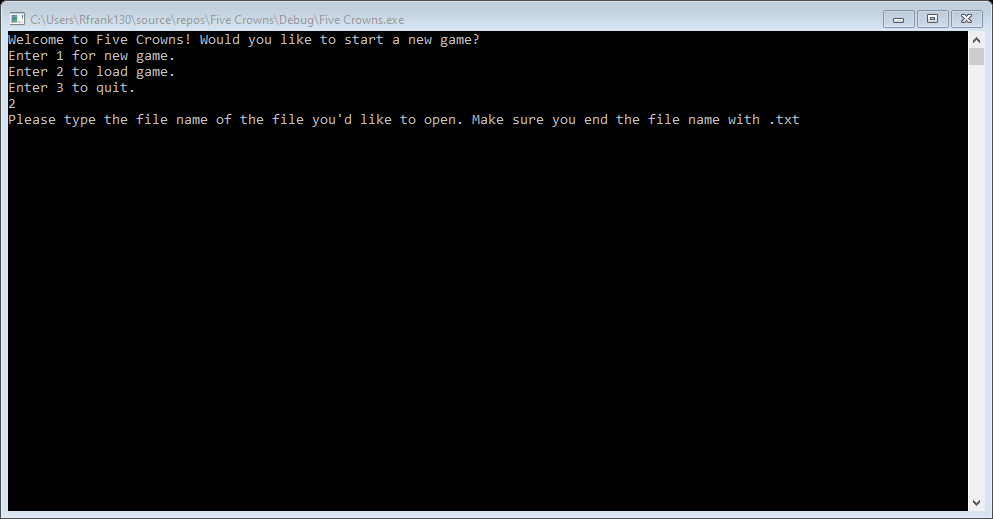
The game will now ask for a text file to save the game as. Note that you must save your game ending in a .txt.



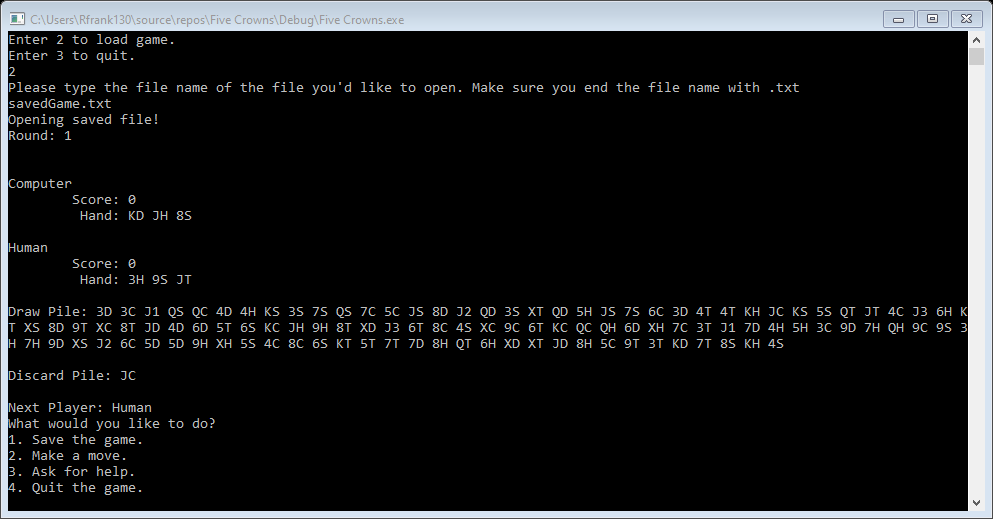
In this example, I saved my game as file name savedGame.txt. Once I have done that, the program exists.

Loading a game…

In the main menu screen, you can enter “2” into the command prompt to load a saved text file game. It will then bring up a prompt asking for the text file.



In this example, I will load my savedGame.txt that I just created. All I have to do is enter it into the command prompt as such.



After entering my saved game text file, the game brings me to the main board game screen, and I can now continue my game.