BUG REPORT

1) When computer makes a pair with its hand card, and the stockpile card matches two OTHER separate cards - the stockpile card is added to the pair made in the first move even if the values dont match.
The function assumes that when the stockpile card matches 2 cards it must be the stack pair from the first move, and makes a triple stack regardless of the values. as a result no cards are captured, and an incorrect triple
stack is made.

2) if the computer or human matches a triple stack or 3 cards with the hand card and then the stockpile card matches a card in the layout, the vector subscript goes out of range and the program crashes.
This is because the matched triple stack (now a quartet) is moved to the Capture Pile immediately, and the vector index that it occupied is empty, which triggers a crash when any vector iterating function tries to iterate over it.**

****Bug 2 existed at the time of the demo, but no longer exists as of submission. The fix was to simply consolidate the Layout after matching and capturing 3 cards in either human or computer first move scenario.**

FEATURE REPORT

MISSING FEATURES:

The Human and Computer Capture Piles are not sorted by value.

EXTRA FEATURES:

There is a full graphical representation of the game printed with ascii graphics. The screen is refreshed when either player plays, a round ends or begins, or any other user input functions (saving, help, quitting). Cards in the layout and the human hand have their index numbers displayed above them, and the stockpile card is denoted with 'SP' above it.

DATA STRUCTURES + CLASSES

DATA STRUCTURES:
vectors are the most used data structures outside of class objects in the program. There are many instances of vectors of vectors and also just vectors of class objects, such as vector<card>.

CLASSES:

Client - inherits gameboard, comp, human, and player. 23 Member functions, 1 constructor, and 2 protected data members. This class is responsible for being the 'dealer'. It handles the bulk of the instructions and function calling for the rest of the classes. It sets the display screen, handles the various conditions that determine what will happen (e.g. player turn or computer turn?), and handles all player input before turns. It also handles loading and saving.

gameboard - inherits card and player classes. 20 member functions, 2 constructors, 1 destructor, 1 private and 3 protected data members. This class is the 'table' which will played upon. a good game table has markings on it to indicate where to place things - spots for cards, currency, drinks etc. The data structures that are members of the gameboard class can be thought of as these spaces. They simply indicate where the objects should go. The client class is responsible for saying what should happen, but the gameboard class is the one doing all of the work, moving cards between data structures, dealing cards, printing objects to the console. It is the workhorse of the program, whereas the client is the brains.

card - inherits no classes. 2 member functions, 3 constructors, 2 protected data members. This class is relatively simple, it represents the card objects, which are comprised of a value and a suit. although the suit is not strictly necessary for GoStop to be played, the class is more portable with the added suit, and also the display is nicer.

player - inherits card class. 5 member functions, 1 constructor, 3 protected data members. This class is the parent class for the human and comp classes, it exists to bring about polymorphism revolving around the two.

human - inherits player class. 12 member functions. This class is used to call functions related to the human player. The human class holds its own capture pile, score, and hand, which is useful for allowing the score to persist between rounds. The functions of the human class have to do with the core gameplay and setting and getting its data members. When playing a game, you would ensure that you are following the rules (hopefully) when making moves - not anybody else. The design here is the same.

comp - inherits player class. 12 member functions. This class is the same as the human class, but it is for the computer player.

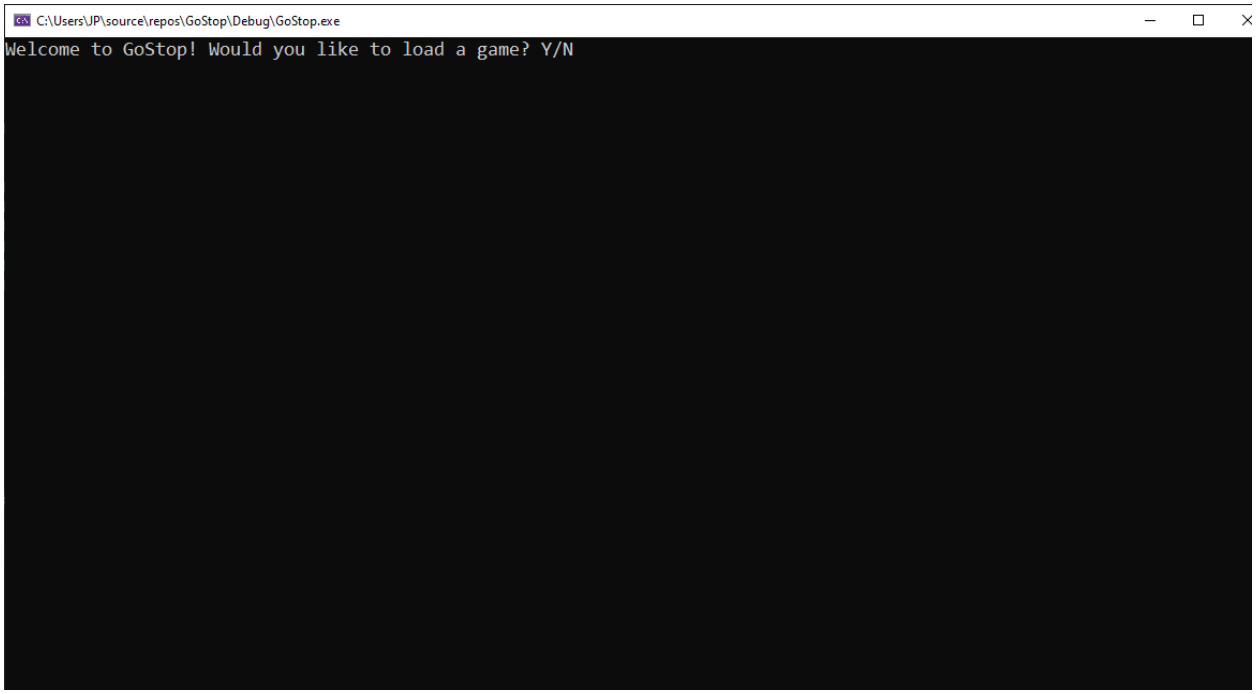
RUNNING THE PROGRAM:

To run the program, navigate to (yourfilepath)/James Giordano C++/bin and run GoStop.exe

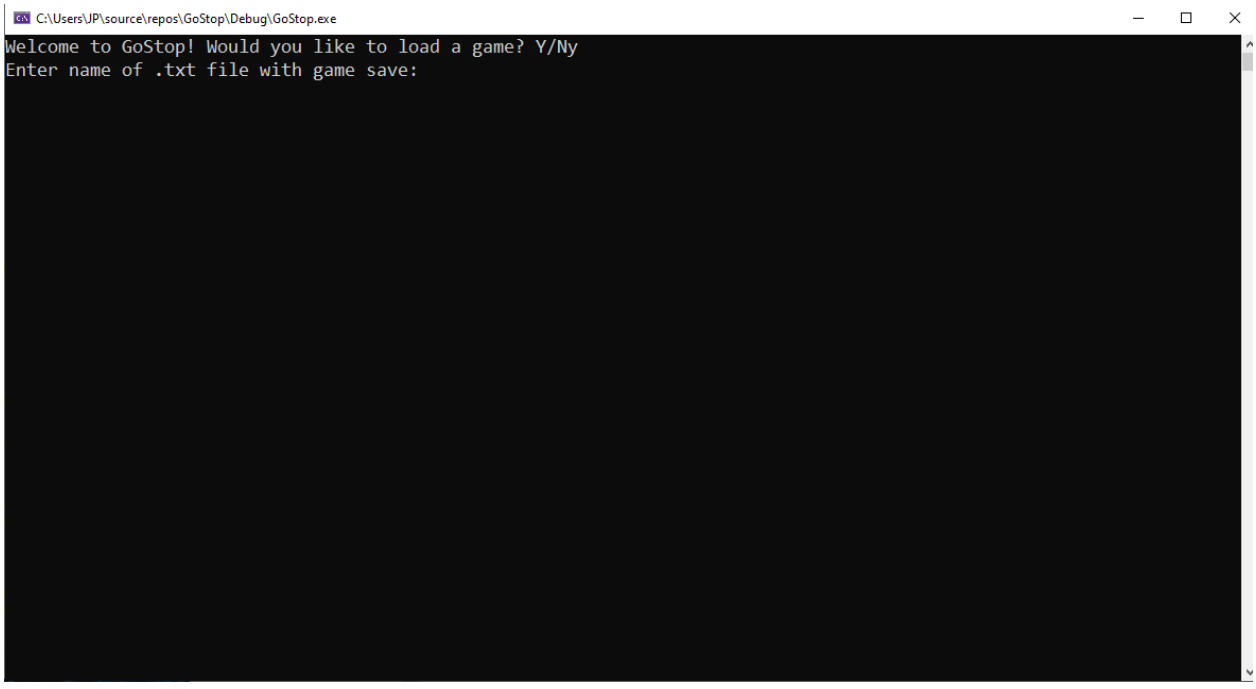
To run for debug open GoStop.sln in the src folder.

SCREENSHOTS:

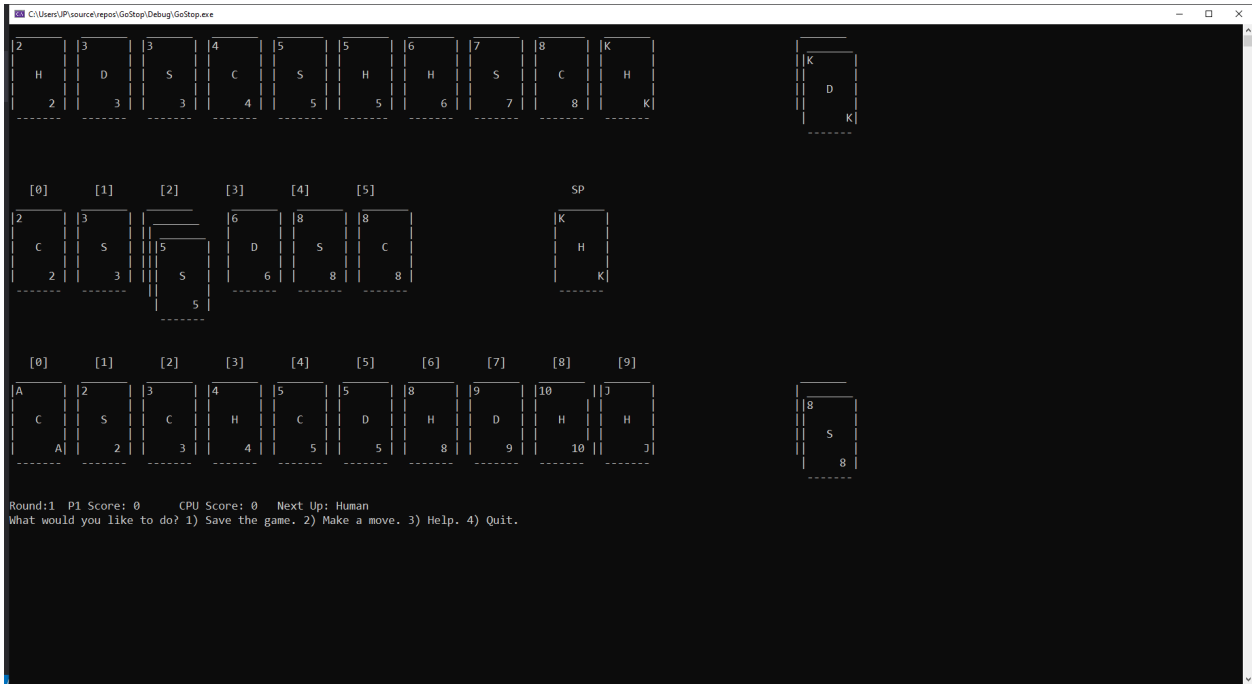
Loading a game, n will result in a fresh game being created.



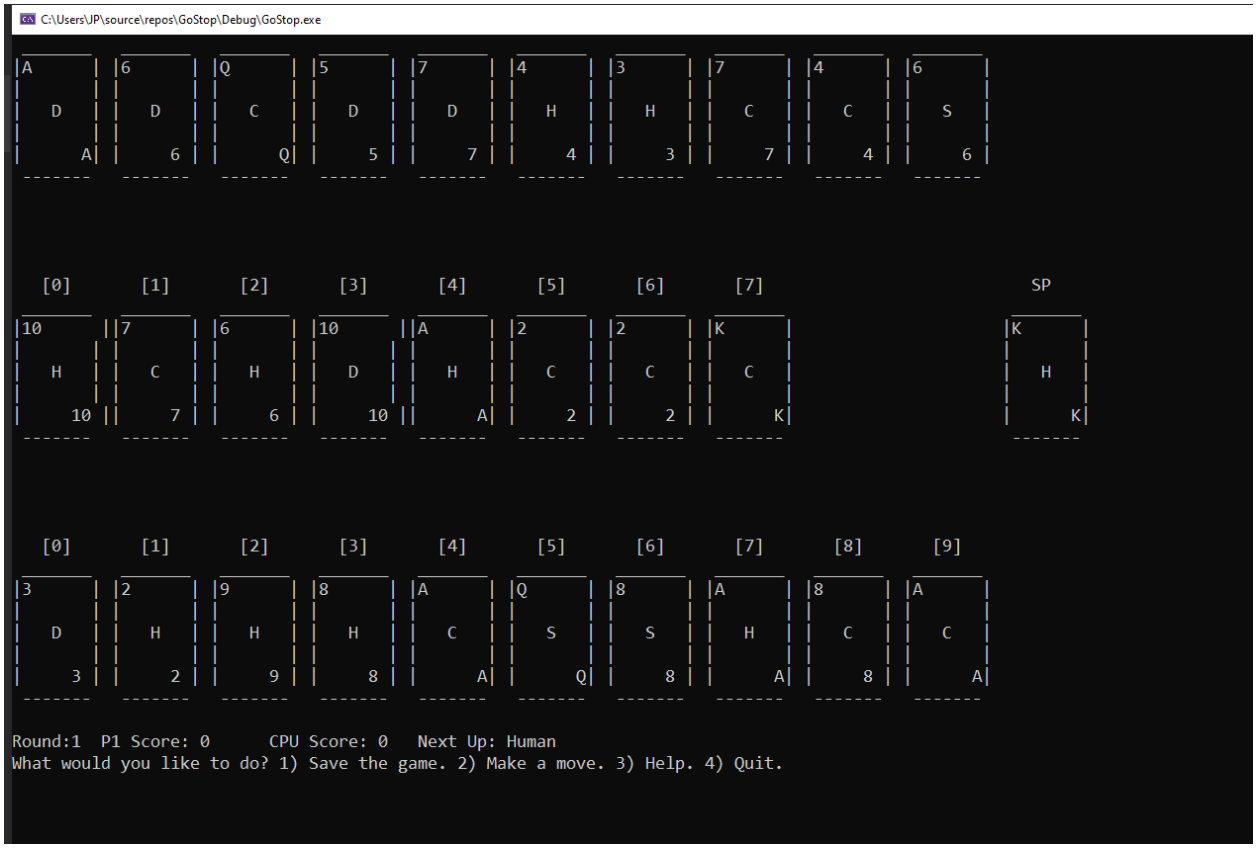
If yes, asks for a textfile name.



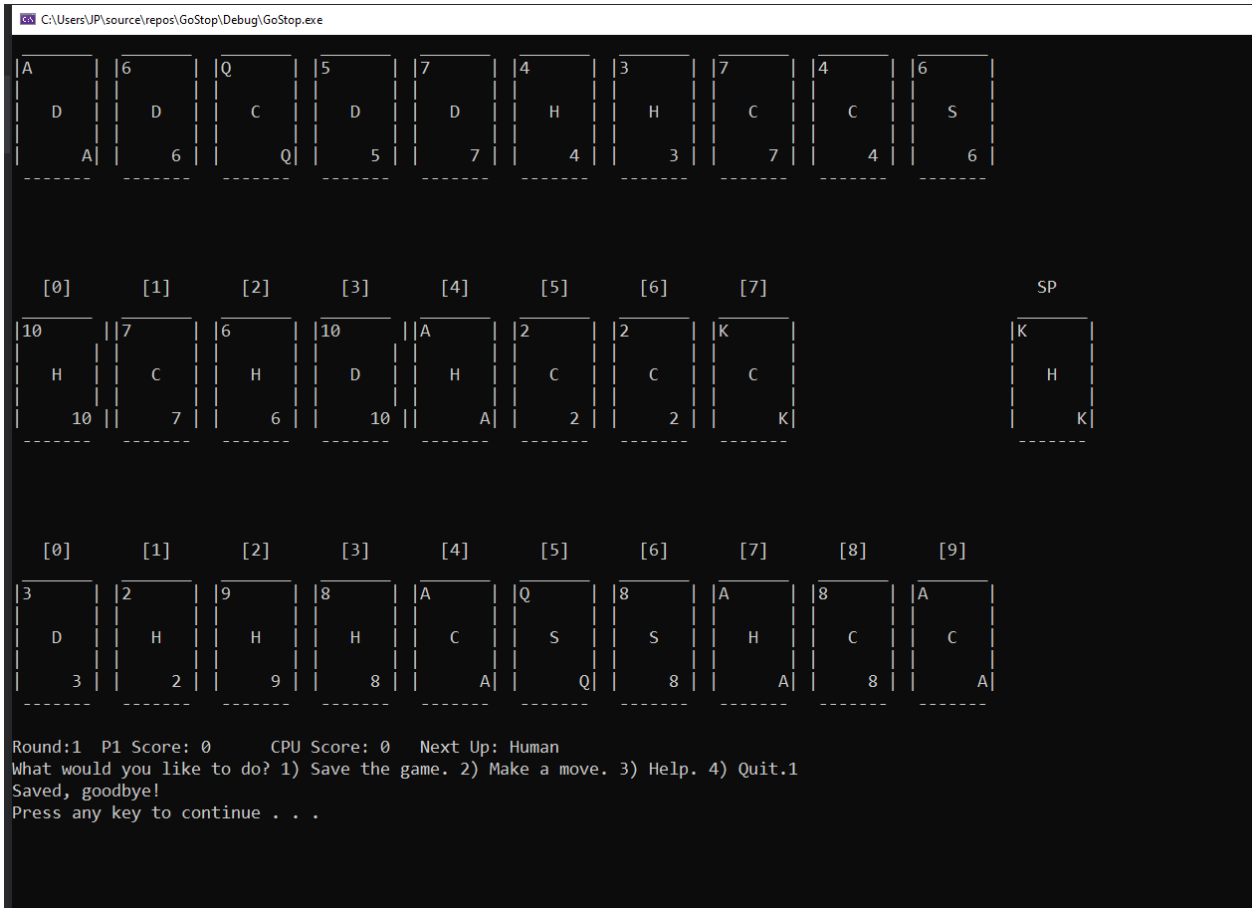
A loaded game will display as such.



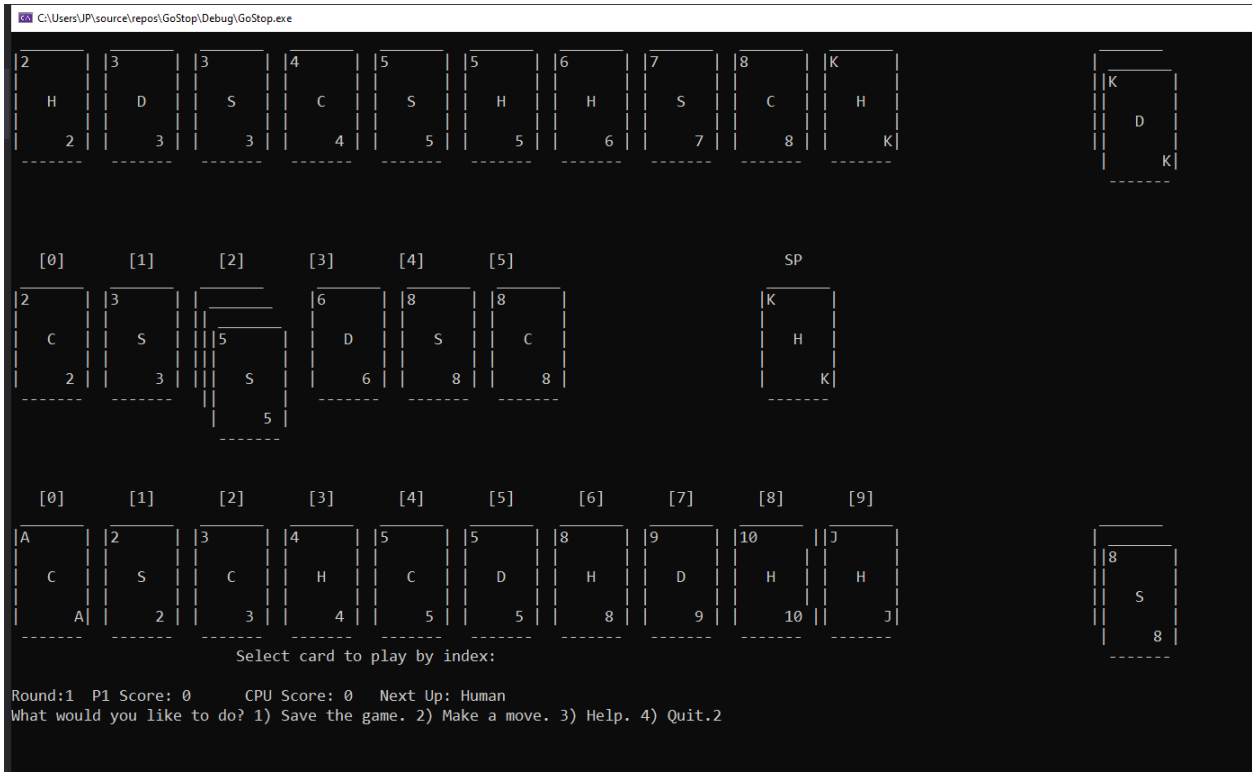
Anwering ‘n’ will start a new game. A new game will display as such.



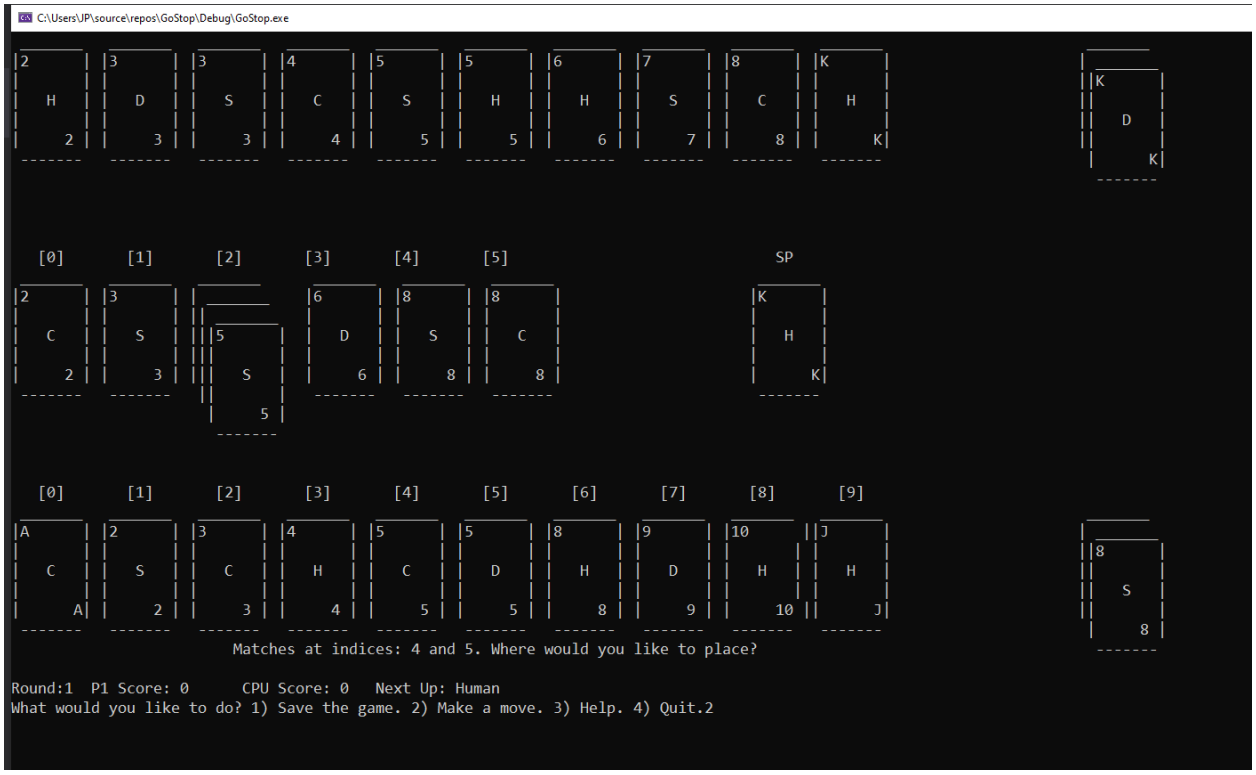
Inputting 1 saves the game to a file ‘savedGame.txt’ informs the player of success, and quits.


C:\Users\JP\source\repos\GoStop\Debug\GoStop.exe
[0] [1] [2] [3] [4] [5] [6] [7] SP
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
Round:1 P1 Score: 0 CPU Score: 0 Next Up: Human
What would you like to do? 1) Save the game. 2) Make a move. 3) Help. 4) Quit.1
Saved, goodbye!
Press any key to continue . . .

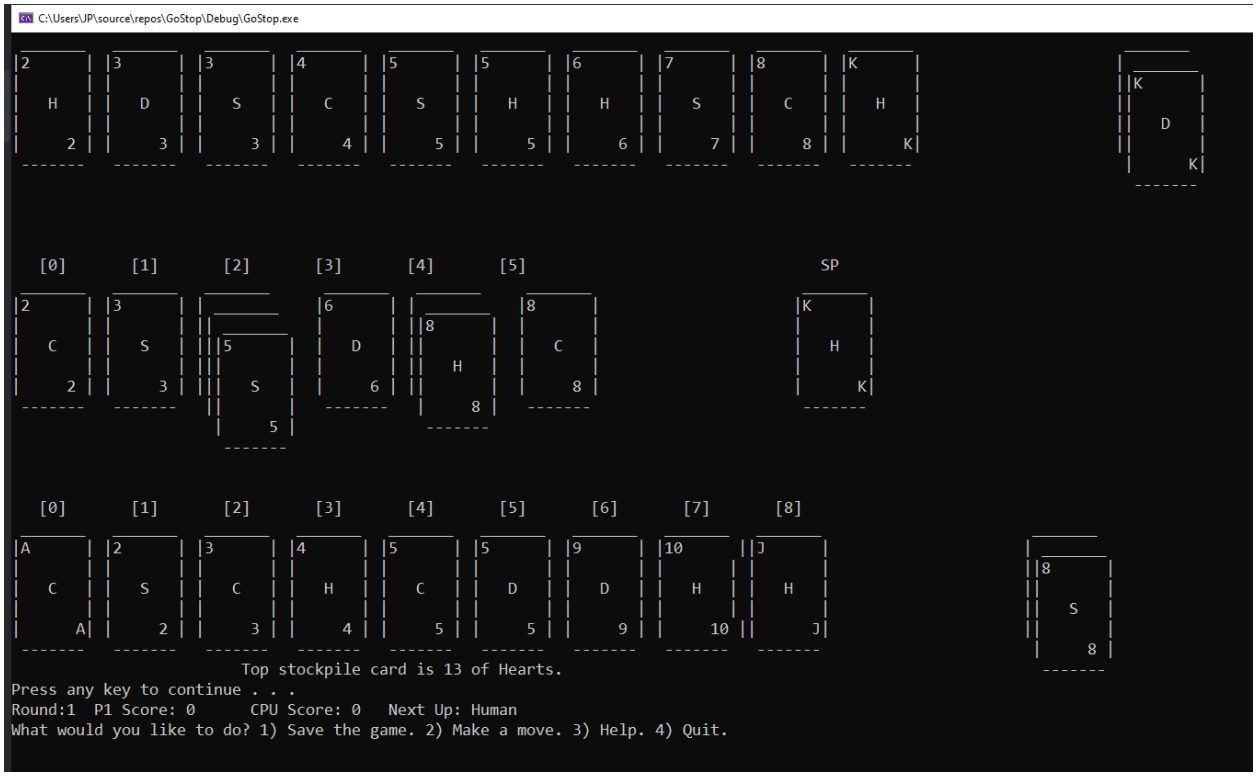
Inputting 2 on a human turn prompts for a card index to play.



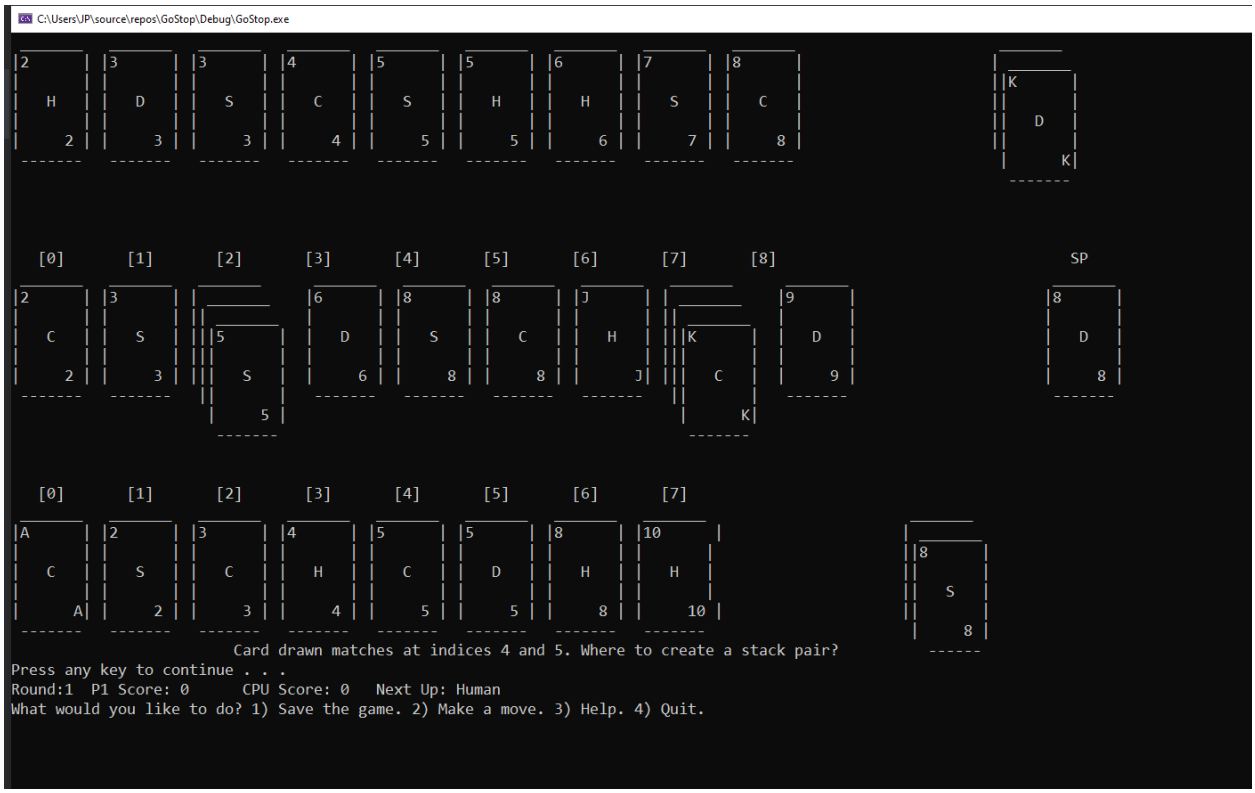
Choosing the index of a card with multiple spots to match will prompt the user to indicate which index they would like to match with.



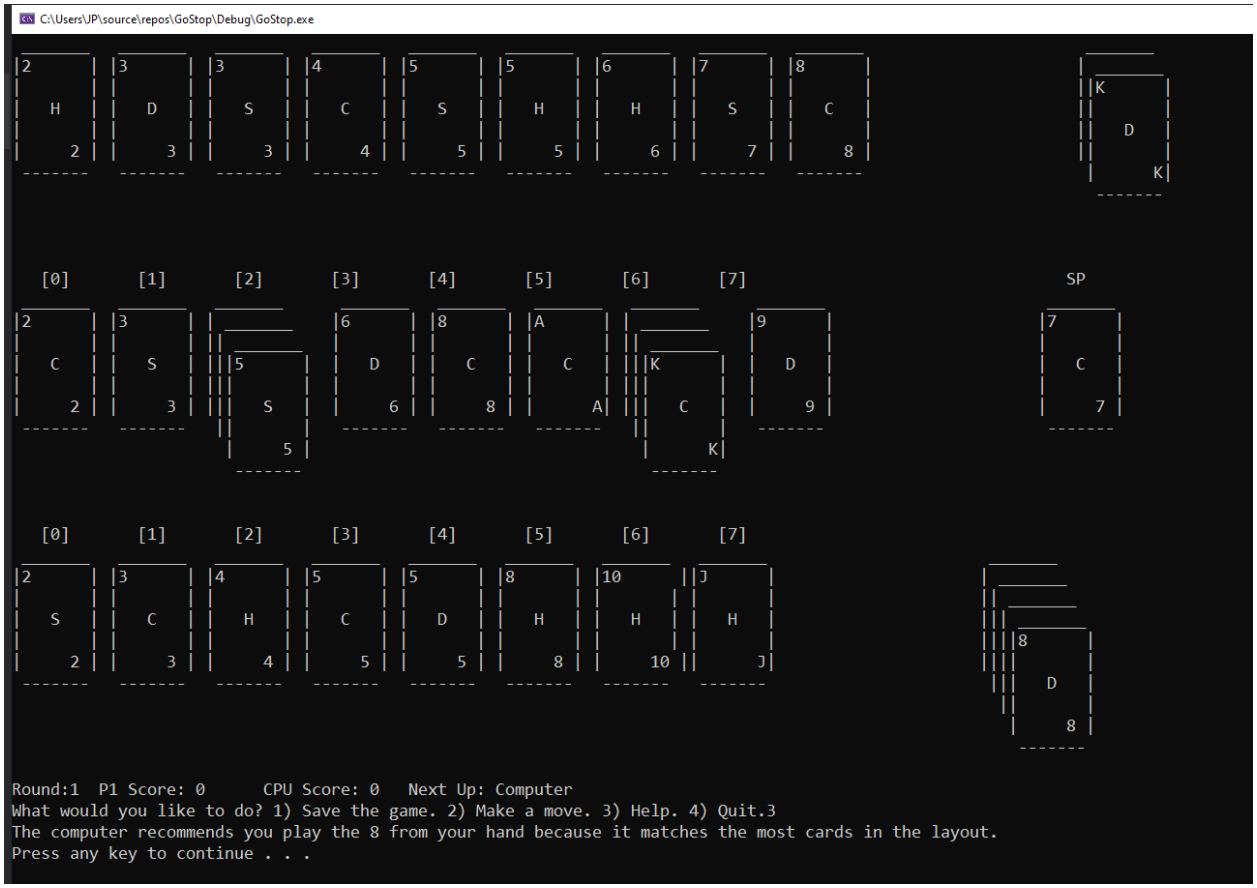
The selected index is matched appropriately.



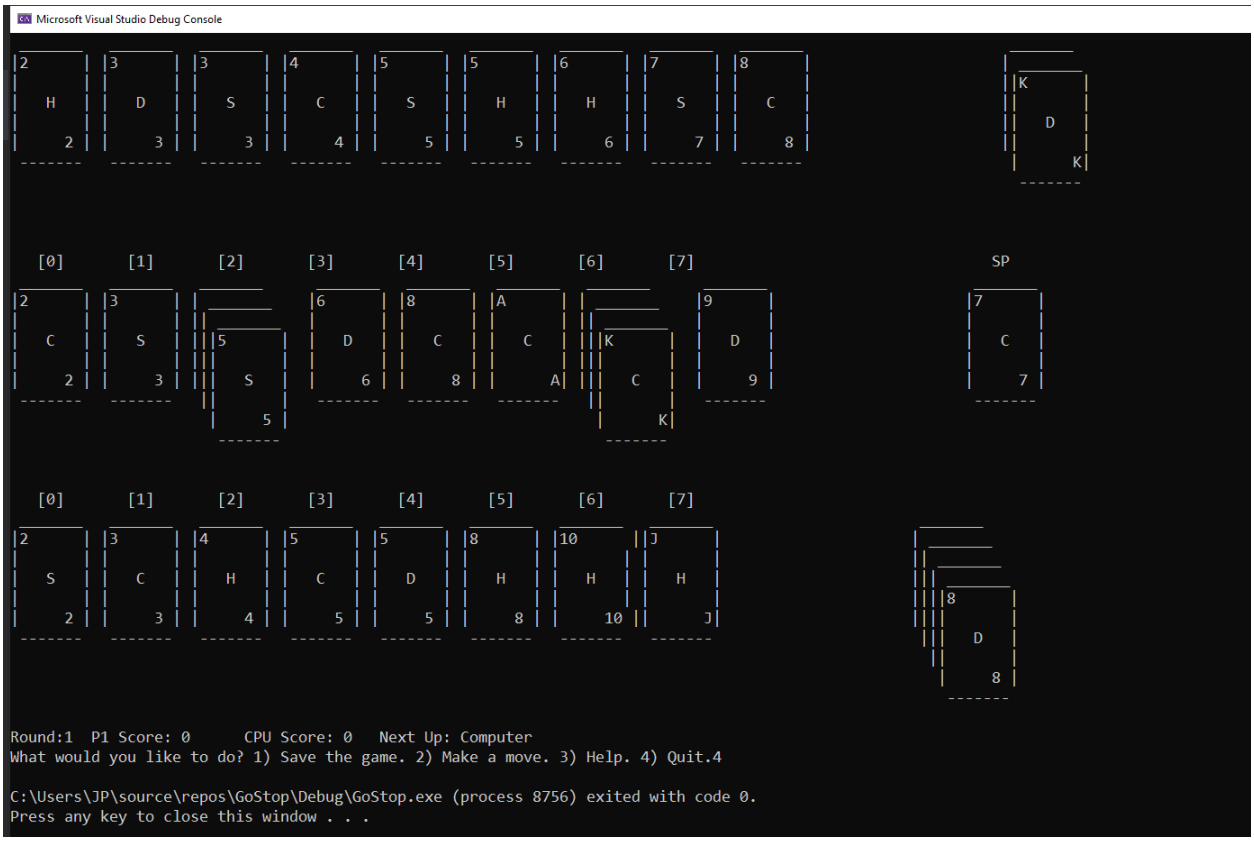
The player is informed what the stockpile card is. If it matches no locations or only one location in the layout, it is played automatically after pressing any button. If it matches multiple indices, the user will be prompted which index to match the stockpile card at.



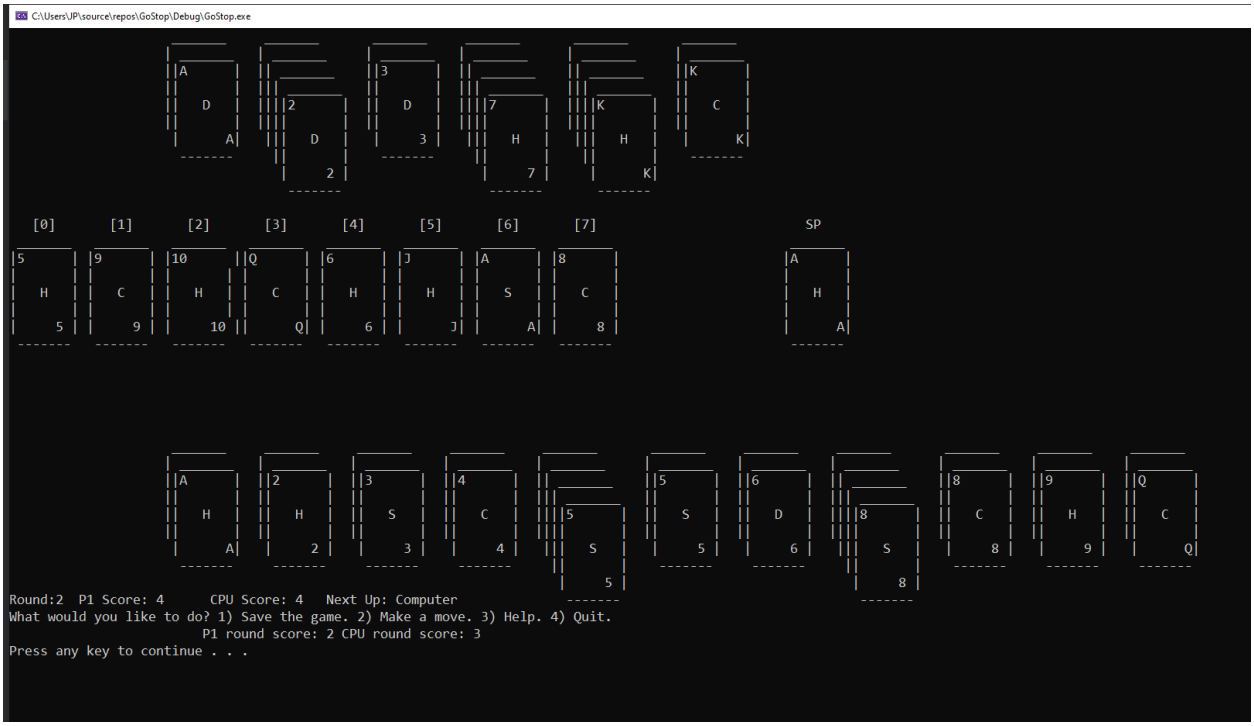
Inputting 3 on either players turn gives the human player a hint on what the computer thinks he or she should play. Although this is redundant for the computer turn because the board state will almost certainly be different by the next player turn, it can allow the player to learn about correct play in more board states.



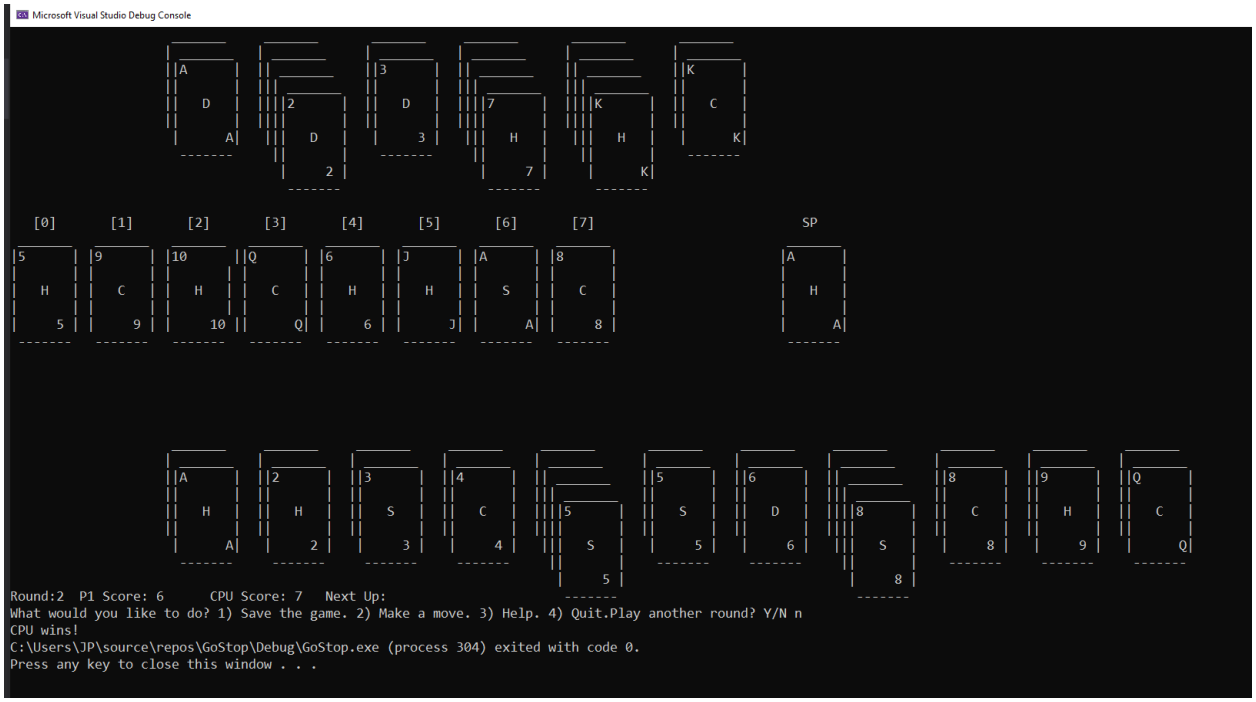
Pressing 4 on either players turn will quit the game.



The end of a round informs the human player of the round scores and waits for a keystroke.



The player is prompted for a new round. If they answer no, a winner will be declared and the game will quit.



If they answer yes, a new game will be set up. Round number and running scores will be incremented accordingly.

On the computer turn, the player is informed of the computer's intention before it moves, and waits for a keystroke before the action is recorded.

LOG

Jan 25 2020:
Planned initial class hierarchy and created card class (1 hr).
created Gameboard and Player class and defined constructors and data members for each (30 minutes).
Added createdeck(), drawcard(), and printcard() functions to Gameboard class (1 hr)
Added PrintPile(), startRound(), and dealCards() functions to Gameboard class (1 hr)

jan 27 2020:
Reworked card class to no longer be a template class, deemed unnecessary and causing typename issues. (15 minutes)
created client, human, and comp classes, defining members and constructors for each (1 hour)

created getter and setter functions for various data members across the classes (card value/suit, player hand, etc.) (45 minutes)
created main() and began working on standard gameplay loop, including various Client functions to facilitate gameplay, such as offerTurn() (1.5 hours)

==========

Feb 5 2020:
added function definition comments for the gameboard class (30 minutes)
added function definition comments for the client class (15 minutes)

Feb 6 2020:
added firstMove function and changed certain Gameboard data structures to be vector<vector<card>> instead of just vector<card> (2 hours)
firstMove function handles cases H0-H3 of playing a card from the hand as well as input validation for selection
added secondMove function to handle cases depending on return value (h value) of firstMove (40 minutes)