Five Crowns (C++) Manual

Five Crowns is a card game that can be played by two or more players. This document contains the details of the Five Crown game which has been designed and implemented in C++.

How to Run the Program

In order to run the Five Crowns game, go to the "*pabhash/bin*" folder and double click on the executable file to run the game. You are all set. A command-line interface window will pop up where you will play the game.

Bug Report

No bugs have been found in the features that have been implemented in the program.

Feature Report

- Additional Features
    - In terms of the computer's strategy, the computer is able to consider potential books or runs, set them aside and drop other remaining cards in hand. Also, the computer will drop the greatest card among the remaining cards in order to minimize the total points in its hand.
- Missing Features
    - All the recommended and fundamental features of the game have been correctly implemented. Hence, there are no missing features.

Description of Classes

Following the principles of Object Oriented Programming, this project contains multiple classes which provide us with diverse functionalities that all come together to create a multi-player card game. Each class and their functionalities have been described below.

1. Card Class

This class serves as the blueprint of all the card objects that are used in the five crowns game. This class gives us the functionality to create a card object with their rank and suit values and also create joker cards.

2. Deck Class

The Deck class serves as the blueprint for a typical 58 card deck that we use in the five crowns game. The Deck class creates objects of Card class and holds them as a deck. A deck contains cards from 3 to King for each of the five suits and also includes 3 jokers. In a five crowns game, we use two objects of the Deck class to play the game.

3. Hand Class

This class represents the collection of cards in player's hand. It contains a vector of cards in player's hand and provides us the functionality to add card, drop card, print and clear player's hand.

4. Round Class

This class provides us with the main interface of the game where player can play each round, switching turns, arranging cards into runs and books and going out when possible. This class is responsible to bring together most of the key functions of the games like initializing the draw and discard piles, play player's turn and evaluate the points of each player when a player goes out. This class also allows the user to save the round state using the Serializer class object.

5. Game Class

This is the uppermost class in the class design hierarchy as it controls the main menu when the game starts and starts round based on the user's input. This class gives the player an option to either start a new game or load a previously saved game. If starting new game, it will ask the number of players and players' information. Then, it will create a round object to initiate a new game. If loading a previously saved game, it uses the Serializer class object to load the previously saved round state from the save file. At the end of 11 rounds, this class determines the winner of game (one with the lowest points) and print the winner on the screen.

6. Player Class

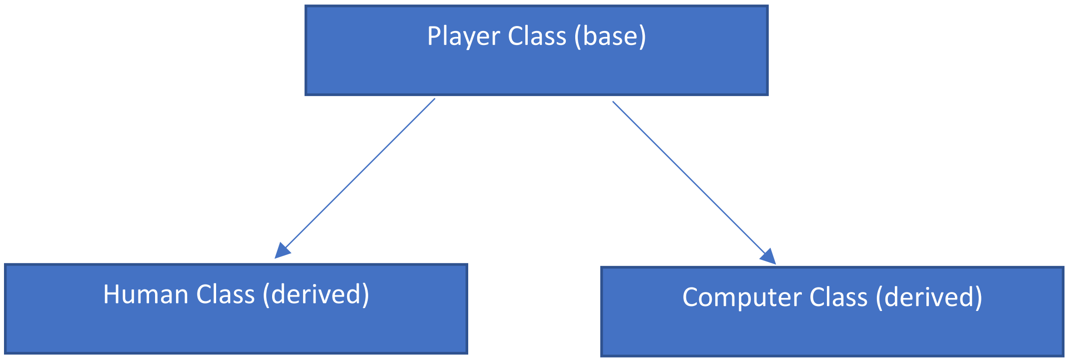
This is the base class for Human and Computer classes which each represent a human player and computer player respectively. This class contains functions that are in common between both human and computer players and also contains virtual functions that behave different based on the type of derived class object that calls the functions. This class is a generic representation of the players playing the game.

7. Human Class

This class serves as the representation of a human player and their functionalities. It is derived from the Player class and contains functions that are specific to human players. This class allows the human player to play their turn (pick card, discard card, go out or ask for help). This class is functions mainly based on human input since he human player is controlled by a human user of the program.

8. Computer Class

This class serves as the blueprint for a computer player and their functionalities. Like human class, it is also derived from the Player class and inherits a lot of general properties and functionalities. Similarly, it also uses the HandAnalyzer object to obtain the AI strategy to play the game and win the game. The computer automatically plays its turn using the strategy and prints its rationale for every move.



Inheritance diagram for derived classes of Player Class

9. Serializer Class

This class functions as the save and load client used to save a particular round state into a text file and load a saved round state from a text file. This process of saving and loading, called serialization is one of the key features of this game. This class contains multiple functionalities that allow it to write the round data to a text file and also record statements from the text file and accordingly obtain round data by parsing the text file.

10. HandAnalyzer Class

This class is responsible for the AI strategy that the computer uses to play the game and to provide help to the human player. This class provides functionalities to take a player's hand and find all possible run/books as well as remaining cards from the current cards in player's hand. It uses quite powerful recursive algorithm to obtain the best combination that yields the least number of remaining cards.

11. Combination Class

This class serves as the blue print for a combination (run/book/half run/half book). We use this in order to encapsulate and hold combinations in players' hand. It is the base class for the Run, Book, HalfRun and HalfBook classes.

12. **Run Class**

This class is a blueprint for a typical complete run object (no jokers/wildcards used). We can use this class object to hold the cards in player's hand that are in a run. This class is derived from Combination Class.

13. **Book Class**

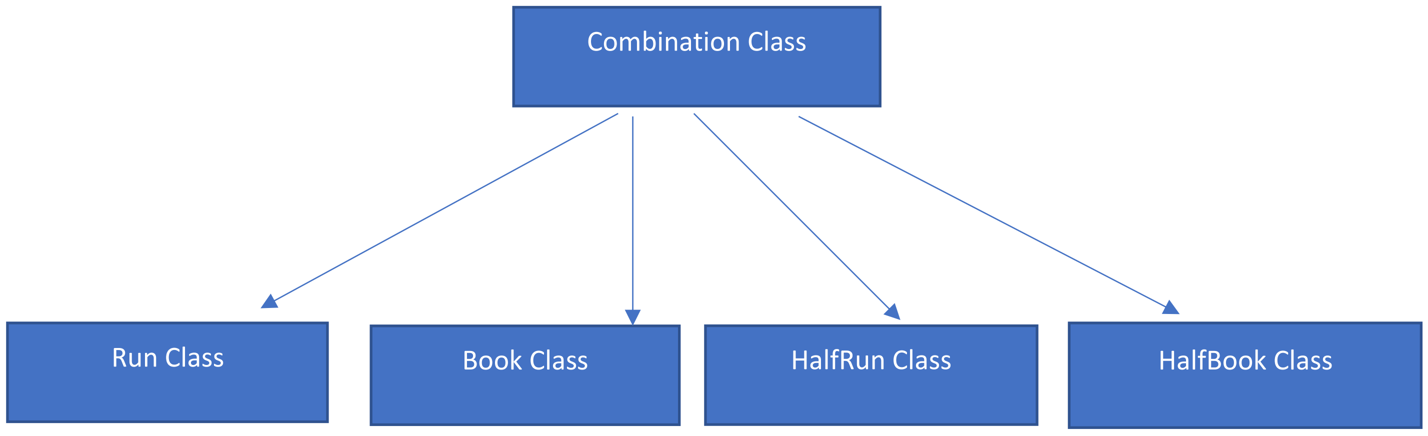
This class is a blueprint for a typical complete book object (no jokers/wildcards used). We can use this class object to hold the cards in player's hand that are in a book. This class is also derived from Combination Class.

14. **HalfRun Class**

This class is a blue print for all the incomplete run objects (that might require a wildcard/joker to make it a complete run. We use this class to identify potential runs that can be complete by adding wildcard/joker or by adding one more card that would complete a sequence. This class is also derived from Combination Class.

15. **HalfBook Class**

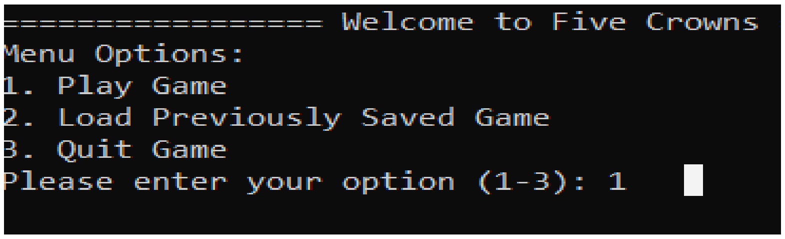
This class is a blue print for all the incomplete book objects (that might require a wildcard/joker to make it a complete book. We use this class to identify potential runs that can be complete by adding wildcard/joker or by adding one more card that would complete a sequence. This class is also derived from Combination Class.



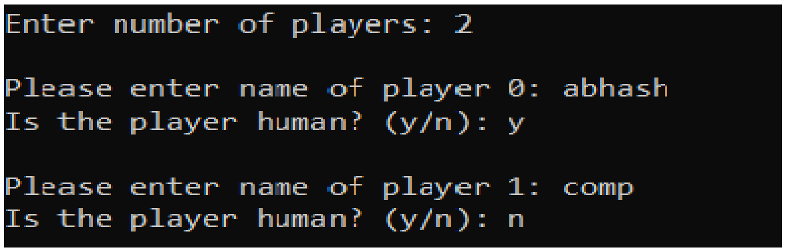
Inheritance diagram for derived classes of the Combination Class

Screenshots

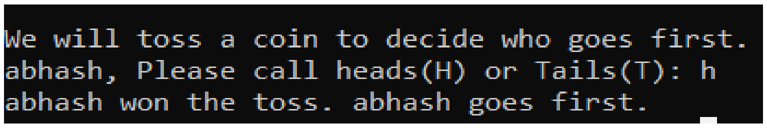
1. All Inputs



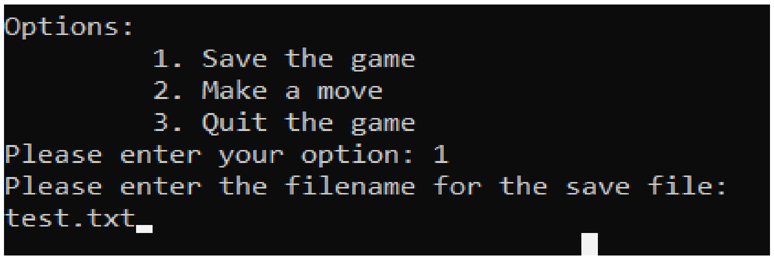
```
================= Welcome to Five Crowns
Menu Options:
1. Play Game
2. Load Previously Saved Game
3. Quit Game
Please enter your option (1-3): 1
```



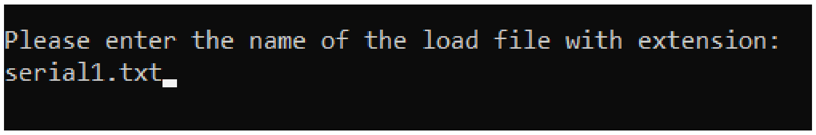
```
Enter number of players: 2

Please enter name of player 0: abhash
Is the player human? (y/n): y

Please enter name of player 1: comp
Is the player human? (y/n): n
```



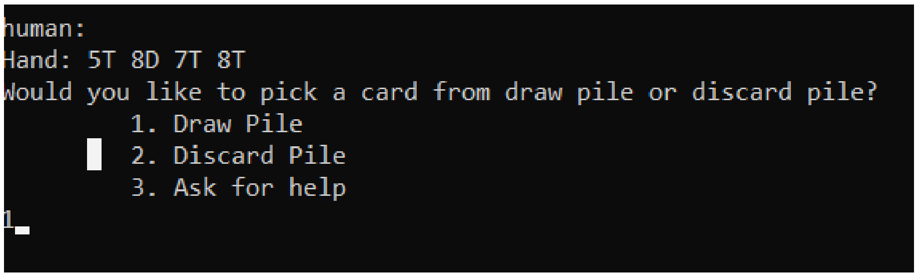
```
We will toss a coin to decide who goes first.
abhash, Please call heads(H) or Tails(T): h
abhash won the toss. abhash goes first.
```



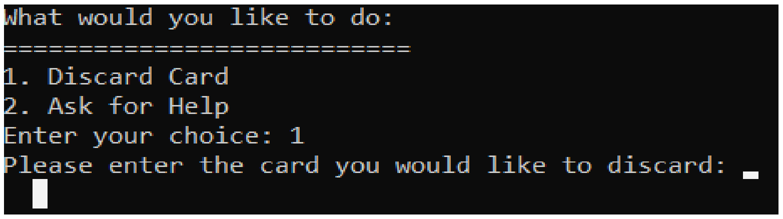
```
Options:
         1. Save the game
         2. Make a move
         3. Quit the game
Please enter your option: 1
Please enter the filename for the save file:
test.txt
```



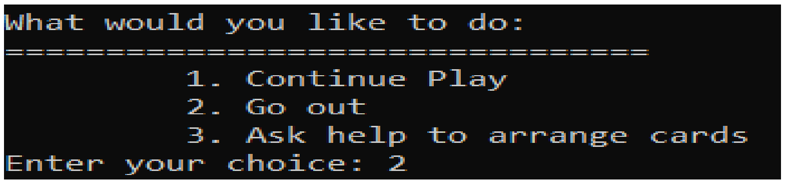
```
Please enter the name of the load file with extension:
serial1.txt
```



```
human:
Hand: 5T 8D 7T 8T
Would you like to pick a card from draw pile or discard pile?
        1. Draw Pile
        2. Discard Pile
        3. Ask for help
1
```

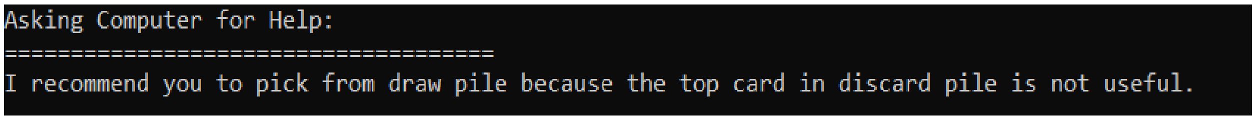


```
What would you like to do:
===========================
1. Discard Card
2. Ask for Help
Enter your choice: 1
Please enter the card you would like to discard:
```

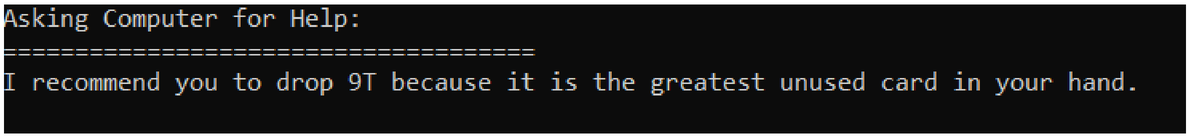


```
What would you like to do:
==================================
          1. Continue Play
          2. Go out
          3. Ask help to arrange cards
Enter your choice: 2
```

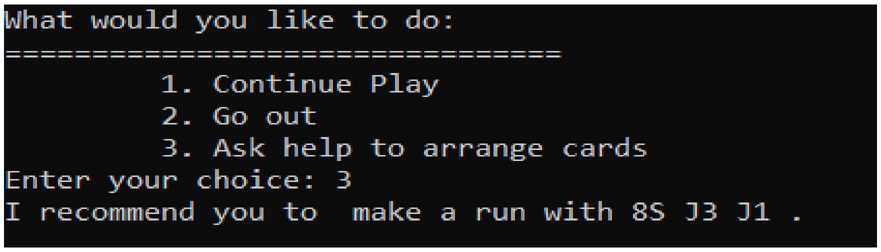
2. Computer Providing Help to Human Player:



```
Asking Computer for Help:
=====================================
I recommend you to pick from draw pile because the top card in discard pile is not useful.
```

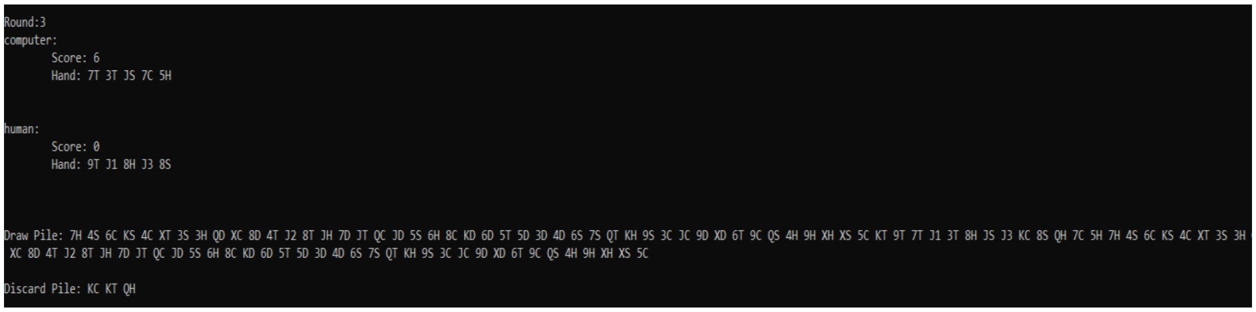


```
Asking Computer for Help:
========================================
I recommend you to drop 9T because it is the greatest unused card in your hand.
```



```
What would you like to do:
=================================
         1. Continue Play
         2. Go out
         3. Ask help to arrange cards
Enter your choice: 3
I recommend you to  make a run with 8S J3 J1 .
```

3. All Components Printed on the screen:



```
Round:3
computer:
        Score: 6
        Hand: 7T 3T JS 7C 5H


human:
        Score: 0
        Hand: 9T J1 8H J3 8S



Draw Pile: 7H 4S 6C KS 4C XT 3S 3H QD XC 8D 4T J2 8T JH 7D JT QC JD 5S 6H 8C KD 6D 5T 5D 3D 4D 6S 7S QT KH 9S 3C JC 9D XD 6T 9C QS 4H 9H XH XS 5C KT 9T 7T J1 3T 8H JS J3 KC 8S QH 7C 5H 7H 4S 6C KS 4C XT 3S 3H
 XC 8D 4T J2 8T JH 7D JT QC JD 5S 6H 8C KD 6D 5T 5D 3D 4D 6S 7S QT KH 9S 3C JC 9D XD 6T 9C QS 4H 9H XH XS 5C

Discard Pile: KC KT QH
```

Project Logs

September 8, 2019:

- Broke down the project into sub problems and parts using top-down programming design principle. Identified classes that are required for the completion of basic structure of the game. (2.5 hrs)
- Created a card class, set up its member variables and functions and found the way to represent each card using enumerators. (2 hrs)
  Total: 4.5 hrs

September 9, 2019:

- Extended the card class to handle jokers and added more utility functions as required. (2 hrs)
- Created a deck class and defined the necessary member functions and variables. (1.5 hrs)
  Total 3.5 hrs

September 10, 2019

- Extended the deck class, added initialize and shuffle functions (1.5 hrs)
- Created a hand class to represent player's cards in hand and set up basic member variables and functions (1.5 hrs)
- Carefully thought about the design of the game to see if I was going in the right direction. Thought of how to integrate all the classes together. (2 hrs).
  Total: 5 hrs

September 11, 2019

- Solved the linker error problem in Card Class caused due to invalid inline function declaration (1 hr)
- Extended the Hand Class by adding utility functions (1 hr)
  Total: 2 hours

September 13, 2019

- Created a Round Class that will be the interface for the player to play a round. (1 hr)
- Added necessary member functions and member variables. (30 mins)
- Realized the need to create Player classes so created player, human and computer classes. (30 mins)
- Completed the inheritance structure of the Player, human and computer classes. (30 mins)
- Declared virtual functions in the base class and implemented them in respective derived class. (1 hr)
  Total: 3.5 hrs

September 14, 2019

- Extended the play functions in Human and Player class. Used random number generator to determine computer’s moves and designed user interface for human’s moves. (2 hrs)
- Integrated the play functions in the round class and tested the functions to see if it worked correctly. Several semantic errors occurred. Had to debug using VS debugger stepping through every line of code to solve the problem. (2.5 hrs)
  Total: 4.5 hrs

September 15, 2019

- Had design and implementation issues: how to update drawPile and discardPile after player picks a card. (1 hr)
- Decided to make the round class static (all member variables and functions are static). (30 mins)
- Tested the code to see if it worked. It did work as expected. (30 mins)
  Total: 2 hrs

September 17, 2019

- Realized that making Round class static was a wrong move. Decided to pass drawPile and discardPile by value to the play functions so player can update the piles. Rectified the mistakes and made all necessary changes to the classes. (2 hrs)

- Tested the code to see if it worked. Few syntax errors and a minor semantic error that was easily fixed using the debugger. (1.5 hrs)
  Total: 3.5 hrs

September 18, 2019

- The basic game components are working well. Need to think of and design the strategy for the computer and serialization next.
- Brainstormed possible ideas for computer strategy (considered sorting the cards by ranks and suits). (3 hrs)
  Total: 3 hrs

September 20, 2019

- Couldn't figure out the strategy to use for computer player. Considered using multimaps instead to easily sort the cards by ranks and values. (1.5 hrs)
- Tried implementing it but it didn't work as expected. (1.5 hrs)
  Total: 3 hours

September 22, 2019

- Created new class to analyze player's hand (HandAnalyzer) (1 hr)
- Decided to use the frequency table to determine the frequency of each card in hand. (1 hr)
  Total: 2 hrs

September 23, 2019

- Created the Serializer class that would help save the game and load the game from a text file. (1.5 hrs)
- Extended saveRound() function to write all the key round states into the save file. (1.5 hrs)
  Total: 3 hrs

September 24, 2019

- Completed the loadRound() function by creating necessary utility function to read the statements from the save file and read each component of the round one by one. (3 hrs)
- Finalized the Serializer class and testing all its functionalities. (1.5 hr)

Total: 4.5 hrs

September 25, 2019

- Implemented the frequency table idea in the HandAnalyzer class and designed algorithms to extract runs and books from the frequency table. (3.5 hrs)
- Created Combination, Run and Book class to extract runs/books. (1 hour)
- Worked on the extractRuns and extractBooks functions to get the complete runs/books in a player's hand. (1.5 hrs)
- Realized that we need to consider incomplete runs/books with jokers so thought about how we could handle that. (1 hr)
  Total: 7 hours

September 26, 2019

- Created halfRuns/halfBooks classes to account for the incomplete runs/books. (1 hr)
- Worked on extractHalfRuns/extractHalfBooks to identify potential cases where halfRuns/halfBooks can be formed using jokers/wildcards. (2 hrs)
- Designed the recursive function that would take all the possible combinations and find the best set of combinations that would result in the least remaining cards. (2 hrs)
- Tested all the new functions and classes to check if they worked, severe semantic errors were found. Using the debugger, stepped through each function to locate the error and solve the problem. (1.5 hr)
  Total: 6.5 hrs

September 27, 2019

- Integrated all the required functions of the HandAnalyzer class into an interface function called analyzeHand() that will simply take the player's hand as a parameter, do all the calculations and update the best combination and remaining cards. (1.5 hrs)
- Checked if it the AI strategy worked correctly, it tested fine. (1 hr)
- So, integrated the AI strategy into the computer's pick/discard/go out strategies and also in help functions for human players. (2.5 hrs)
- Also, integrated the AI strategy to check if a player can go out. (30 mins)
  Total: 5.5 hrs

September 28, 2019

- Solved the syntax and semantic errors that popped up after integrating new code into the classes. (30 mins)
- Designed how the last round is going to be played after a player goes out. (1 hr)
- Created the game class which will allow the user to start a new round, load a saved game and even print out the winners of the game after the final round ends. (1.5 hrs)
  Total: 3 hrs

September 29, 2019

- Some errors found in the Serializer class while reading the save file. Fixed the errors that were identified. (1 hr)
- Worked on the documentation of the source code. (1.5 hrs)
- Tested the code again to identify potential bugs. Several minor bugs were found using the debugger and they were solved as well. (1.5 hrs)
  Total: 4 hrs

September 30, 2019

- While testing the code, I realized the strategy had a flaw. It would allow the player to drop the greatest card even if it could make a potential run/book. So, extended the HandAnalyzer class to consider potential runs/books before dropping the card. (2.5 hrs)

October 1, 2019 – Demo Day

- The code was working except it couldn't handle wildcards for some reason. Tried debugging the code to identify the reason. No luck. (3 hrs)

October 2, 2019

- Set the wildcards when the draw pile is initialized so that we can track the problem in the very beginning. It helped and the program started working normally. (1.5 hrs)
- Also, updated the help function to provide help in each stage of a human player's turn. (1.5 hrs)
  Total: 3 hrs

October 3, 2019

- Worked on source code documentation. (3.5 hrs)

October 4, 2019

- Completed the source code documentation and worked on the project manual. (3 hrs)

END OF MANUAL