Card Game: Kings in the Corner

CS 342 Design Document

I'm Dhrumil Patel, currently a junior in University Of Illinois At Chicago majoring in computer science with software engineering concentration, and my email is dpate85@uic.edu. This reports presents design of a Card Game called Kings in the Corner. This game is played by the user and one computer opponent. We used the following website http://www.pagat.com/domino/kingscorners.html as our official rules for the game.

The basic idea of the game is to test analytical skills. Just like any other standard deck, this game has 52 cards in a deck. Each card is assigned a unique Rank and Suite. So there will be a total of 13 Ranks and 4 Suits equalling 52 cards. The Suits are colored either Black or Red, and each card is listed with a specific rank and suit. The Rank is sorted in specific order, Ace being the low rank and King being the highest rank. The order of the Suit has no impact to the game. There are several requirements for the sub-system of the program. It truly gives a gist of object oriented programming. It also tests your Java skills.

The brief summary of the game is as follows; First of all, list all cards in players hand on one of eight lay-down piles. The piles must be ordered from high rank to low rank. Then, if the bottom card on the pile is Red Eight ,regardless of suit, the next card in the pile would need tone a Black Seven and then a Red Six, etc. Four of the lay-down piles must have Kings as the bottom card and other four can have any card as the bottom card. The lay-down piles are numbered 1 to 8. A particular algorithm is used for dealing the card:

- Shuffle 52 cards in the deck.
- Deal out seven cards to each player. A top card in the deck is given to the player who is not dealing. Then give new top card to dealer. Continue the process still both players have seven cards in their hands. The remaining 34 cards are kept in the draw pile.
- Put a card on each lay-down piles 1 through 4.

There are certain parts for completing the game. The winner of the game is decided via points at end of the game. A round is completed once a player has laid down all the cards. The algorithm for Computer's AI is also created. Detailed information about the game can be found in project pdf[1]. Now, the program takes commands such as Quit the program, Help, About, Draw a card from Draw pile, Lay a Card on a pile and Move one pile on top of another.

[1] https://www.cs.uic.edu/pub/CS342/AssignmentsS16/proj1.pdf

To achieve completeness and requirements of the game several methods and classes are used. Each class has its own purpose which supports the game as a whole. Brief information about the class that were used for making the game is in the table below.

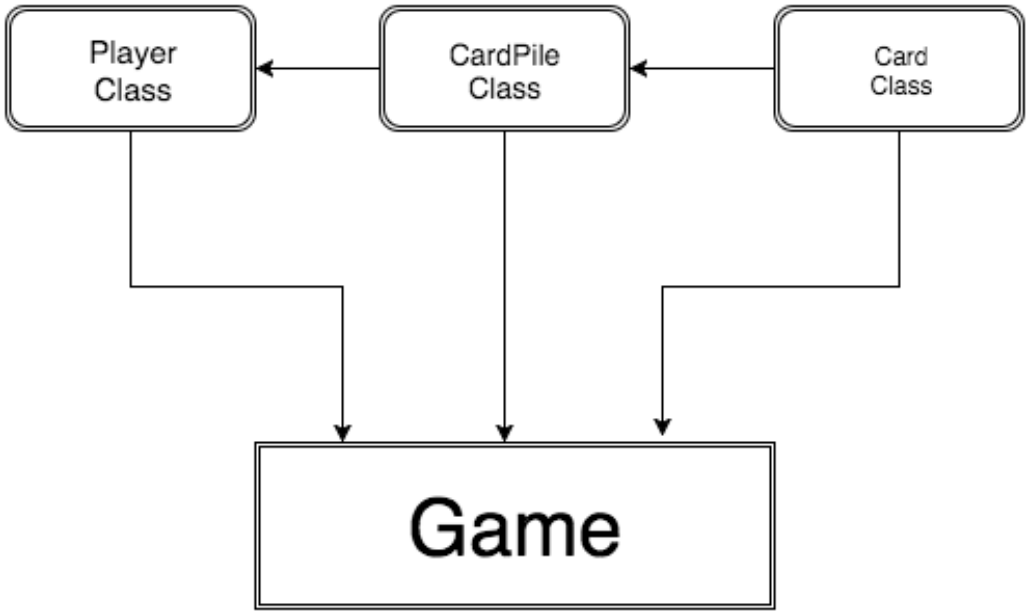
Class Name	Description
Card.java	Contains all the Rank and Suit information. Returns King, Queen, Jack, Ace as their respective integers number
CardPile.java	Builds the deck
Game.java	The main program which takes in the support of all other class. It includes the round and computer's AI algorithm.
Player.java	Implemented the scores, valid move, num of cards methods.

The Card class contains several functions in it. This class has rank, suit and isBlack private members. getCard(returns the string containing suit and rank in it, setCard sets the rank, suit and deck information. setRank and setSuit is quite self explanatory. print function will print the Rank and Suit as a string. This class is used by all other classes.

The CardPile class builds and contains the deck information. It has the Card(class) array, boolean anyCard and an int index as it's members. The index member is set to zero, it increases and decreases accordingly. The getSize method returns the size of deck. getTop returns the top card from Card deck. removeTop removes the top card, it also decreases the size. addCard adds the card and increases the index(size). isPlaceable method returns if the move is valid or not and print method prints the deck. The pile ranges from 1 through 8.

The player class uses Card class. It has the Card(class) array, size of deck, and score as its private members. This class keeps track of scores, number of cards in player's hand, checking for king in the hand, and other similar methods found in cardPile class such as removeTop, getTop, getSize, etc. addToHand adds the rank and suit information in the player's hand. calcScore will calculate the score after each round. remove method is self explanatory. This Class is used for Computer and User too.

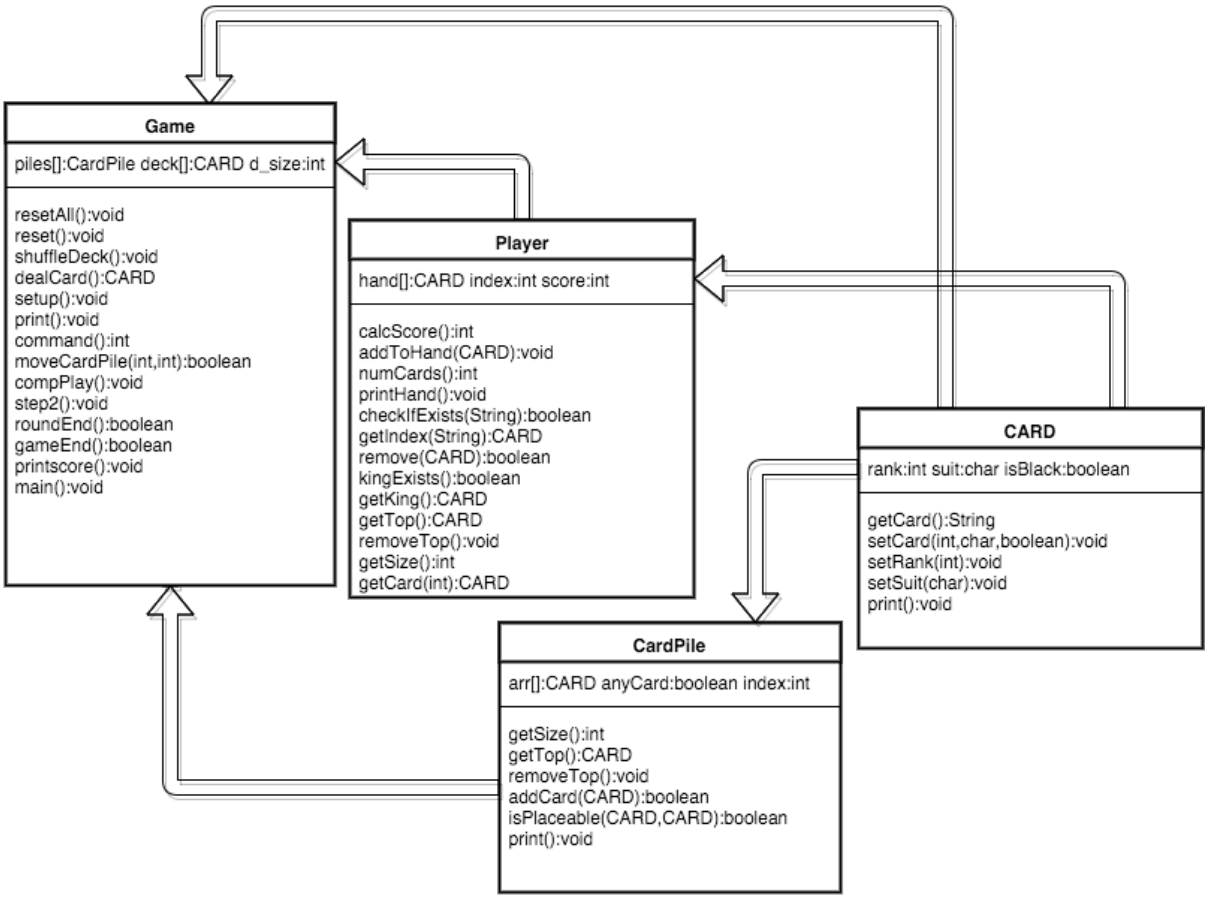
The classes above are utilized in the main class called Game. It contains several methods along with the main method. First of all, the resetAll method will reset the game and set everything to 0 including scores and deck. shuffleDeck sets the card at random. The methods for User commands are also included. For example, moveCardPile is used when the M command is used by the user. It also includes the compPlay which implements the computer's AI algorithm described in the pdf of project. The Dealing algorithm is also implemented in this class.



The above figure shows how each class and methods communicates with each other. The game takes in several commands from user. A brief summary of User Interaction is listed below:

- "Q" : Quits the program, The Program will exit by returning 0
- "H" : Help, itPrints the help menu i.e commands accepted, Prints the Help menu listing all the accepted commands along with its info and proper usage
- "A" : About (prints program and author info)
- "L <Card> <CardPile>" : Lay a card on a pile, This command utilizes method from Player class. It first checks if the CardPile exists. If it does, then it will add the card in given CardPile and remove the card from player's hand. If not then , it will print "Invalid Move"
- "D" : Draw a card from draw pile. Uses dealCard method to draw a card and adds it to user's hard via addToHand method (created in Player class). Returns the Card Drawn
- "M <CardPile1> <CardPile2>" : Move one CardPile on top of Another CardPile. A specific method ,moveCardPile, is created for this command. The moveCardPile returns a boolean. In the moveCardPile, the first thing it checks is the from variable. It would return false right away if the from was greater than four. Else if it's less then four, it will check the "to" variable by putting it in piles array. If the pile was her then false is returned. After move checking, it goes into a while loop with following condition: piles[from] size is > 0, and piles[to].addCard(piles[from].getTop()) is true, then piles[from].removeTop, increase move. if move remains zero, then return false else return true.
    - When this command is entered, it will check if the move "from, to" is valid. If it's not, a desired error will print informing user about it.

The figure below shows each method created in class along with its type and utilization. The function names work according to their name. The design was created keeping meaningful



variable and function names in mind.

This particular design has a lot of benefits. First of all, it's simple to read and understand the code. The variable names are meaningful. The function names are quite self explanatory too. Creating three different classes provides accessibility to the programmer for different scenarios. Programmer can change the algorithms in main function as desired. It's easier to add and remove functionality as the Card, Player and CardPile functions are already implemented. The other benefit is that if we want to change anything in Rank and Suit, only the Card class is required to change. Other classes will adapt the Rank and Suit change automatically. Same thing goes for CardPile and Player algorithm. The program might not run properly if major illogical changes are made in the three supporter — CardPile.java, Player.java, Card.java — classes. As every class is dependent on each other, the program will definitely not run logically. It would simply print out irrational result ,or in worst case scenario, it could crash.