26 January 2020

Assignment 2­ – Design

**Understanding the Problem/Problem Analysis:**

Write a program that allows one player to play a game of Crazy Eights against the computer using a standard 52 card deck with one card for each rank and suit.

* Set up a deck of 52 cards (LAB)
* Play a game of Crazy Eights:
  + The deck of cards is shuffled, randomizing the order of the cards. (LAB)
  + Each player is dealt 7 cards.
  + The remaining cards are placed face-down (with rank and suit hidden) in a stack on the table and becomes the "stock", with the top card turned over and displayed in a separate "pile".
  + One of the players (player A) begins the game by playing a card from their hand of the same suit or rank as the top card on the pile, which becomes the new top.
  + Player B then plays a card from their hand of the same suit or rank as the top card, which becomes the new top.
  + If a player does not have any cards of required rank or suit, they must draw from the deck (stock) and add cards to their hand until they draw a card that can be played on the pile, or until the deck runs out of cards.
  + The game continues with the players alternating turns until player gets rid of all their cards, or until all cards are drawn and no one can play. If neither player has zero cards at the end of the game, the player with the least number of cards wins.
  + Eights of any suit "wild" and may be played on any turn. When an eight is played, the player must specify a suit for the next player to match.
* Include the following gameplay features:
  + Print the current state of the game (e.g. cards held by the human player and card of on the top of the pile) after each turn (keep computer player's cards hidden).
  + On the human player's turn, prompt the user for a card to play. When they enter a card, either put the card on top of the pile or have them draw a card from the deck if they can't play. If the card is an 8, prompt the user to declare a suit.
* Once the game is over, announce the winner and ask the user if they want to play again.
* No memory leaks.

*User inputs/requirements, program outputs:*

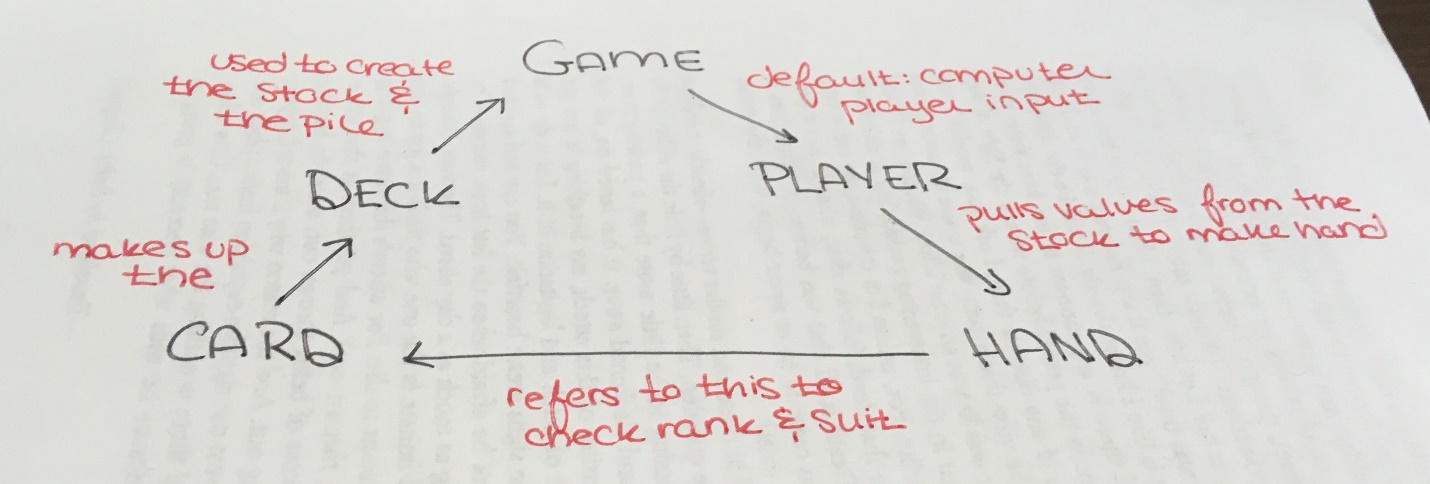
* Input:
  + Player choice of card
  + When 8: choice of suit
  + If the player wishes to play again
* Requirements:
* Output:
  + Print player hand
  + Print card on top of the pile

*Tasks and subtasks in this problem:*

* CARD
  + Create card (assign it a suit and a rank)
  + Print the card
* DECK
  + Create the deck (52 cards of unique rank and suit)
  + Shuffle the deck
* Deal each of the players 7 cards
  + Create a hand for each player and add 7 cards to their hands
  + Subtract cards from the total deck
  + Pull one last card from the deck to start the pile
  + Print the player hand and the pile
  + Play the game
    - When the player plays an 8 prompt for what suit they want to call
    - Verify that the player is putting down a valid card, i.e. a card of the same suit / rank or an 8
    - Note that any rank can be played off of an 8 but it must be the same suit
    - COMPUTER:
      * Verify that it is putting down a card of valid suit / rank
      * When placing an 8 choose a suit (randomly?)
    - If a player puts down an 8 prompt for a suit; if the computer puts down an 8 randomly generate a suit
* When game is complete prompt user if they wish to play again

**Program Design:**

*Overall big picture of this program (flowchart or pseudocode):*



Do while player wishes to continue:

GAME

Uses player.h

OBJECTS: Deck full\_deck, Player computer, Player user

Upon initiating the game

Create players (which create hand); one user (prompt for name) and one computer

Fill the hand with 7 cards (alternate dealing) do while

Flip the first card of the deck to begin the pile

Do while for turns until the game ends: begin with player and then computer

Check that player inputs are values corresponding to cards (number them)

Use atoi to check integer values

If the player selects a card that is not valid (wrong suit/rank) spit out error

If rank != rank and suit != suit … prompt for good value

8:

Get value for player suit (verify that value is between 1 and 4)

Computer: randomly choose a suit (value between 1 and 4)

Have the computer run through its hand and check if suit then rank match pile

If not, check if the computer has an 8 in the hand

Begin drawing; continue drawing until a card matches (same for player)

Check if deck is empty

If it is, end the game

Print the game board: pile and player hand

Reference print card and hand

PLAYER

Uses hand.h

OBJECTS: Hand

Add and remove cards from hand

Check hand for cards with specific suit or rank

HAND

Uses deck.h or card.h

Dynamic array of cards

Functions to add and remove cards

Functions to check suit/rank

DECK (from Lab)

Uses card.h

Creates a deck (52 cards of unique rank and suit; dynamic array)

Shuffles the cards

Prints the cards (calls print card function)

Remove card to deal to player – can you delete elements of a static array?

CARD (from Lab)

Get card rank and suit (integers)

Print card (convert integer values into printable values)

*Required Code:*

rank and suit are represented with int values

must also map those values to a representation of the suit or rank (ex: string)

class Card {  
 private:  
    int rank;  // Should be in the range 0-12.  
    int suit;  // Should be in the range 0-3.  
 public:  
  // must have constructors, destructor, accessor methods, and mutator methods  
  
};

the source of all the cards

Cards initially start in a Deck object and are then be transferred to players' hands

Important method: remove a card and return it so it can be placed in a player's hand (deal a card from the deck to the player)

class Deck {  
 private:  
    Card cards[52];  
    int n\_cards;  // Number of cards remaining in the deck.  
 public:  
  // must have constructors, destructor, accessor methods, and mutator methods  
  
};

hold the cards in one player's hand

number of cards a player holds may change -> dynamic array

need functions to add and remove cards

other useful methods might check for a given suit or rank.

class Hand {  
 private:  
    Card\* cards;  
    int n\_cards;  // Number of cards in the hand.  
 public:  
 // must have constructors, destructor, accessor methods, and mutator methods  
};

represents a single player

each Player has a Hand and an array keeping track of the cards the player lays down

methods to consider:

add / remove cards from hand

check hand for cards with a specific suit or rank

figure out what suit they want to ask for if they play an eight

Note that Player class must represent both the human player and the computer player;

write class methods accordingly and add extra data members needed

class Player {  
 private:  
    Hand hand;  
    string name;  
 public:  
  // must have constructors, destructor, accessor methods, and mutator methods  
  
};

represents the state of an entire game

contains objects representing the deck of cards and both players

useful to check whether the game is over and that execute a player's turn

class Game {  
 private:  
    Deck cards;  
    Player players[2];  
 public:  
 // must have constructors, destructor, accessor methods, and mutator methods  
}

*What data is created:*

* A deck of cards (52 unique cards; static array)
* Two players, one computer and one user
  + Hands for each player
  + The hands are the only dynamic arrays

*When user input is read:*

* Upon program initiation

*Decisions that need to be made in this program:*

* How smart is the computer?
  + There must be some sort of check to see which card it can play
  + Run through the array and check if any suit matches the pile
  + Then check if any rank matches
  + If not, begin drawing
  + Check if each new card matches
  + When playing an 8 randomly select a suit from 1 to 4

*Repeated tasks:*

* Drawing cards
* Prompt user to place a card OR
* Parse through computer hand to find matching rank/suit
* Checking rank/suit of the cards (compare to the pile)
* Draw from the stock
* Printing the board
  + Printing card values
* Check how many cards are left in the stock

*Bad input to handle:*

* See error handling

**Program Testing:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Value** | **Expected** | **Actual meet expected** |
| Want to play again? [1] yes [2] no |  | 1 or 2 | 1 starts the game over, 2 ends the program |
|  | 3 |  | No; use if else statements and reprompt for valid integer |
|  | a |  | No; use atoi to deal with string input |
|  |  |  |  |
| Select a suit:  [1] hearts [2] spades [3] clubs [4] diamonds |  | 1, 2, 3, 4 | 1 is a heart, 2 is a spade, 3 is a club, 4 is a diamond |
|  | 5 |  | No; use if else statements and reprompt for valid integer |
|  | b |  | No; use atoi to deal with string input |
|  |  |  |  |
| Select which card you wish to play; list card suit and types with corresponding array value, ex: [0] 4 of hearts [1] ace of spades … |  | Number between 0 and size of array | Use a for loop to print the elements of the array of cards in the user’s hand; this number will be no larger than 52 (technically no larger than 44) |
|  | 100 |  | Have this value bounded by the size of the array; use if else to reprompt for a valid integer |
|  | c |  | No; use atoi to deal with string input |