VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

HO CHI MINH UNIVERSITY OF SCIENCE

FALCUTY OF INFORMATION TECHNOLOGY

----------o0o----------

MIDTERM PROJECT

PROGRAMMING TECHNIQUES

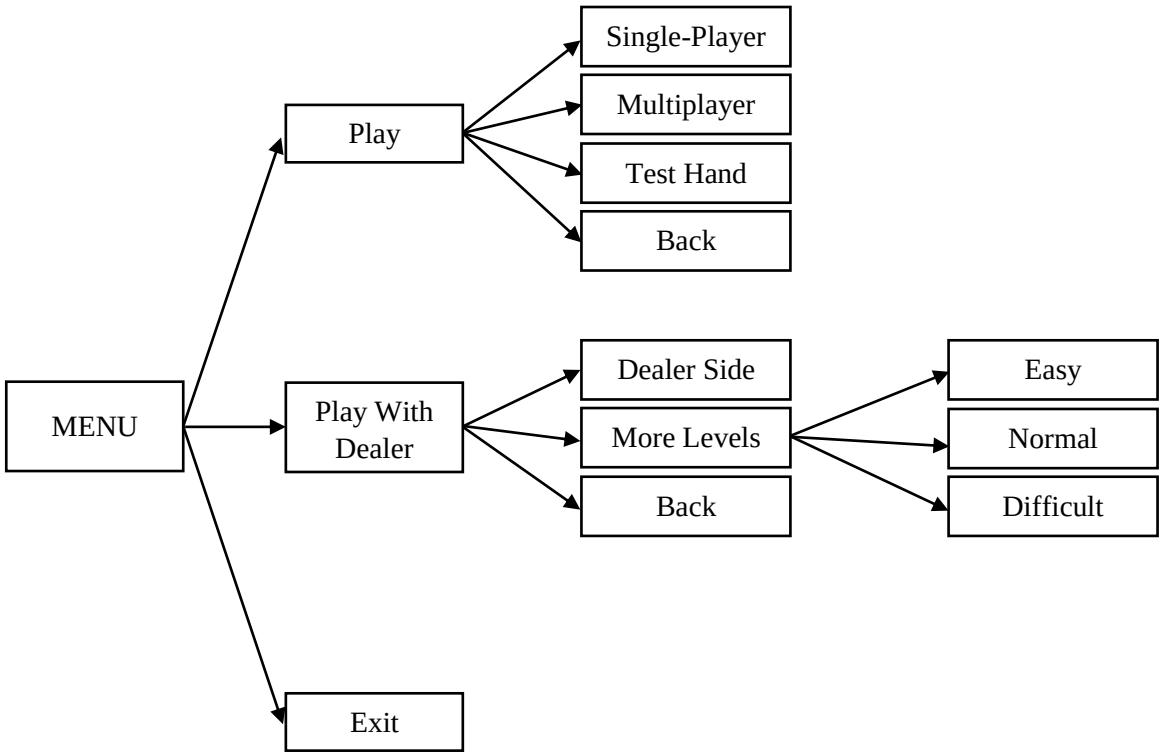
TOPIC: POINTERS

THEORY LECTURER:	**NGUYỄN THANH PHƯƠNG**
LAB INSTRUCTOR:	**NGUYỄN NGỌC THẢO** **BÙI HUY THÔNG**
FULL NAME:	**ĐỖ MINH TRÍ**
STUDENT ID:	**20127651**

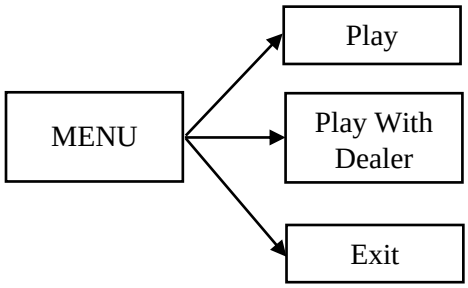
HO CHI MINH CITY, 4/2021

I. FLOWCHART

I.1. Function Flowchart



I.2. Interface Flowchart *(Interface images will be shown in the Appendix)*

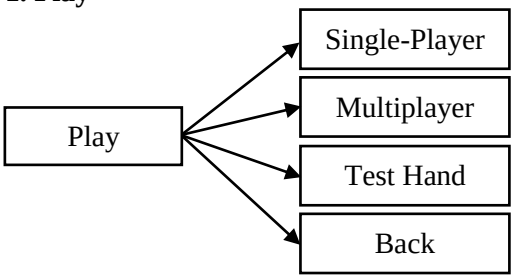


(1) Open the game without Dealer

(2) Open the game with Dealer

(3) End the program

1. Play



(1.1) Open the game for 1 Player

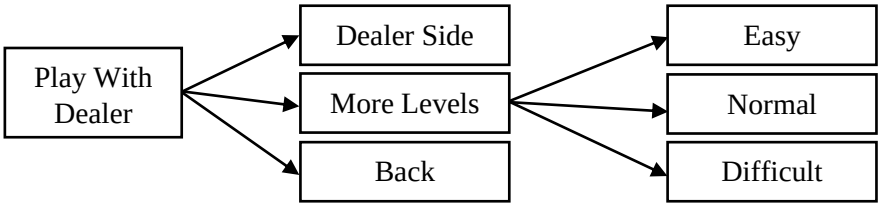
(1.2) Open the game for n Players

(1.3) Open the test hand game

(1.4) Return MENU

2. Play With Dealer

(2.1) Open the game for Dealer (2.2) Open the game for Player and Dealer



(2.2.1) Open easy level

(2.2.2) Open normal level

(2.2.3) Open difficult level

(2.3) Return MENU

II. FUNCTIONS

II.1. Provided Functions

a) Demonstration

- `void main();`
  - This function call `menu()`.

- `void menu();`
  - This function displays the menu interface.
  - Select *Play* to open `playWithoutDealer()`, *Play With Dealer* to open `playWithDealer()`, *Exit* to break the loop and then end the program.

- `void playWithDealer();`
  - Similar to `menu()`, select *Single-Player* to open `singlePlayer()`, *Multiplayer* to open `Multiplayer()`, *Test Hand* to open `testHand()`, *Back* to break the loop and return to the menu interface.

- `void playWithoutDealer();`
  - Similar to `menu()`, select *Dealer Side* to open `dealerSide()`, *More Levels* to open `M()`, *Back* to break the loop and return to the menu interface.

The input data of these functions are arrow characters entered from the keyboard and will change the color of the selected option with the `SetColor()` function.

b) Card shuffling & Dealing

- ❖ Define the constants `SUITS` = 4, `FACES` = 13.
- ❖ Initialize this string as a global variable:

```
wstring suits[SUITS] = { L"♥", L"♦", L"♣", L"♠" };
wstring faces[FACES] = { L"A", L"2", L"3", L"4", L"5", L"6", L"7", L"8",
                         L"9", L"10", L"J", L"Q", L"K" };
```

  - To visually display the card, I declare the library <io.h> and <fnctl.h>, use `wstring` instead of `char*` to print the characters "♠, ♣, ♦, ♥" in the Unicode table.

- `void shuffleCards(int deck[SUITS][FACES]);`
  - This function is used to shuffle the deck.
  - Loop the order from 1 to 52, use the `randomCard()` to randomly generate the position of the card, if the element at that position is equal to 0 then set the order for that position.

- `void printCardsShuffling(int deck[SUITS][FACES]);`
  - This function is used to print the cards in the deck of cards.
  - Loop the order from 1 to 52, and loop to reach each element in the `deck`. If any element is equal to the order of the card, print that element with row is suit and column is rank.
  - Old format: `void printCardsShuffling(int deck[][], char* suits[], char* faces[]);`
    Purpose for changing: Because the strings of suits and faces have been initilized, there is no need to pass into the function.

c) A poker game for 1 Player

- `int** dealingForHand(int deck[SUITS][FACES], int order, int numOfPlayers);`
  - This function is used to deal cards from the deck.
  - Loop to reach each element in the `deck` array to find elements whose values are equal to the order of the player's cards, then add them to player's `hand`. `Hand` is a two-dimensional array with 5 rows for 5 cards and 2 columns, the first column for suit, the second column for rank. After dealing, sorting player's `hand` by using `sortHand()` function and return that `hand`.
  - Old format: `int** dealingForHand(int deck[SUITS][FACES]);`

Purpose for changing: Pass the variables `order` and `numOfPlayers` to deal cards following to the player's order.
The order of the next card = The order of the previous card + The number of players
Ex: For 4 players, the second player receives the [2, 6, 10, 14, 18] cards.

- `void printHand(int** hand);`
  - This function is used to print cards in the hand.
  - Convert each card from the `hand` array to a card shape by using `addHandLine()` function, then print cards in the `hand` array.
  - Old format: `void printHand(int** hand, char* suits[], char* faces[]);`
    Purpose for changing: Because the strings of suits and faces have been initilized, there is no need to pass into the function.

- `int** createHandTest(int deck[SUITS][FACES], int a[]);`
  - This function generates a test case.
  - Similar to the `dealingForHand()` function but the order of cards is the elements of the `a` array.

- `void testHand();`
  - This program is used to test to draw cards.

- `int isFourOfAKind(int** hand);`
  - This function checks whether a hand contains *Four of a kind.*
  - Using the variable `count` is the number of cards per rank in the `hand` array. Loop through 13 ranks, if any rank has `count` = 4 then return 1, else return 0.

- `int isFullHouse(int** hand);`
  - This function checks whether a hand contains *Full house*.
  - If there are both *Three of a kind* and *Pair* then return 1, else return 0.

- `int isFlush(int** hand);`
  - This function checks whether a hand contains *Flush.*
  - Using the variable `count` is the number of cards per suit in the `hand` array. Loop through 4 suits, if any suit has `count` = 5 then return 1, else return 0.

- `int isStraight(int** hand);`
  - This function checks whether a hand contains *Straight*.
  - After dealing, the cards in hand are sorted in ascending order. So only consider if each element increments 1 unit apart then return 1, else return 0.

- `int isStraightFlush(int** hand);`
  - This function checks whether a hand contains *Straight Flush.*
  - If there are both *Straight* and *Flush* then return 1, else return 0.

- `int isThreeOfAKind(int** hand);`
  - This function checks whether a hand contains *Three of a kind.*
  - Similar to *Four of a kind*, but `count` = 3.

- `int isTwoPairs(int** hand);`
  - This function checks whether a hand contains *Two pairs.*
  - Using the variable `numOfPair` is the number of pairs and the variable `count` is the number of cards per rank in the `hand` array. Loop through 13 ranks, if any rank has `count` = 2 then `numOfPair` increases 1. When the loop ends, if `numOfPair` = 2 then return 1, else return 0.

- `int isPair(int** hand);`
  - This function checks whether a hand contains *Pair*.
  - Similar to *Two Pairs*, but `numOfPair` = 1.

- `int getHighestCard(int** hand);`
  - This function return the value of the highest card.
  - After dealing, the cards in hand are sorted in ascending order. So just return the 5th card.

- `void singlePlayer();`
  - This program is the game for 1 player.

d) A poker game for n Players

- `int*** dealingForHands(int deck[SUITS][FACES], int numOfPlayers);`
  - This function distributes cards to n players.
  - Allocate a 3-star pointer hands that is a list containing two-dimensional arrays, each two-dimensional array is a hand. Loop the `dealingForHands()` function n times to deal cards with n players.
  - Old format: `int*** dealingForHands(int deck[SUITS][FACES], int n);`
    Purpose for changing: Rename the variable `n` to `numOfPlayers` for ease to use.

- `int getStatusOfHand(int** hand);`
  - This function returns the hand-ranking of five cards (8 if there is *Straight Flush*, 0 if they do not fall into any hand-ranking category).
  - Check through each case from largest to smallest, if it falls into any case then return that case number.

- `int* rankingHands(int*** hands, int numOfPlayers);`
  - This function ranks n players in one turn and returns an array of n elements such that the player ith is in the rank a[i].
  - Allocate a pointer rank with n elements, then loop through each element. The value of each element is determined by using the `getStatusOfHand()` function.
  - Old format: `int* rankingHands(int*** hands, int n);`
    Purpose for changing: Rename the variable `n` to `numOfPlayers` for ease to use.

- `int* evaluateHands(int* Score, int* rank, int numOfPlayers);`
  - This function records the person with the highest score.
  - Find the highest score in the `rank` array, then loop through the `rank` array, if `rank[i]` is equal to the highest score then add points for `Score[i]`, `Score` is the array that records all players' scores in all round.

- `void Multiplayer();`
  - This program is the game for n players.

e) A poker game for Dealer

- `void drawingNewCards(int deck[SUITS][FACES], int** hand);`
  - This function requires to enter the option to draw cards: (1) Draw at random; (2) Draw cards for a better situation.
  - Enter the value of the variable selection. If selection = 1 then call `randomReplace()`, selection = 2 then call `betterReplace()`. If selection is different from the above two values, the player will be required to enter again.

- `void randomReplace(int deck[SUITS][FACES], int** hand);`
  - This function is used to randomly replace cards.
  - The number of cards and the cards to be replaced will be randomly assigned and replaced via the `replaceCard`() function. After replacing, sorting the cards in `hand` by using the

`sortHand()` function and print out the replaced cards information via the `printChangedCards()` function.

- `void betterReplace(int deck[SUITS][FACES], int** hand);`
  - This function is used to replace cards for a better situation.
  - Randomly replace the cards, compare the hand-ranking via the `getStatusOfHand()` function if the next hand is greater than the previous hand then stop the loop and print out the hand after drawing, otherwise, replace randomly until there is. better situation. If after looping 100.000 times but there is no better situation then stop loop and announce that better situation can't be drawn.

- `void replaceCard(int deck[SUITS][FACES], int** hand, int cardPosition);`
  - This function replaces the card in hand with the card on the top of the deck.
  - Find the number of the card at the top of the deck using the `findTopCard()` function. Then loop through the deck to find the element at position [i] [j] whose value is equal to the number of the card on the top of the deck, then update the i, j values for the card need replacing on hand.

- `void dealerSide();`
  - This program is the game for dealer.

f) A poker game Player vs Dealer

- `void askDealerToDraw(int deck[SUITS][FACES], int** hand, bool& Ans);`
  - This function will random the dealer 's decision about drawing.

- `void askPlayerToDraw(int deck[SUITS][FACES], int** hand, bool& Ans);`
  - This function requires the player to enter the number of cards to draw and the order of the cards to be replaced.

- `void selectReplace(int deck[SUITS][FACES], int** hand);`
  - This function replaces the number of cards according to the player's selection.

This game only player can draw cards, so it is easy level.

g) Game Levels

- `void easyLevel();`
  - This program lets a player and the dealer compete with each other, but only the player can draw cards.

- `void normalLevel();`
  - This program lets a player and the dealer compete with each other, both the player and dealer can draw cards.

- `void difficultLevel();`
  - This program lets a player and the dealer compete with each other, but only the dealer can draw cards.

- `void moreLevels();`
  - This function is used to choose game levels.

II.2. Self-Written Functions

Functions that allocate memory

- `int** memAlloc_2D(int row, int col);`
  - Allocate two-dimensional dynamic array.

- `int*** memAlloc_ListOf2D(int num2DArr, int row, int col);`
  - Allocate a list contains two-dimensional dynamic arrays.

Functions that free up memory

- `void memFree_2D(int** a, int row);`
  - Free up two-dimensional dynamic array memory.
- `void memFree_ListOf2D(int*** a, int num2DArr, int row);`
  - Free up the memory of the list containing the two-dimensional dynamic arrays

Functions that input data

- `void inputNumOfGames(int& numOfGames);`
  - Enter the number of rounds to play.
- `void inputNumOfPlayers(int& numOfPlayers);`
  - Enter the number of players joining the game.
- `void inputNumOfReplacedCards(int& numOfReplacedCards);`
  - Enter the number of cards to be replaced.

Functions that output data

- `void printStatus(int** hand);`
  - Print the status of hand.
- `void printRankingHands(int* rank, int numOfPlayers);`
  - Print the hand-ranking of players.
- `void printHighestRank(int* rank, int numOfPlayers);`
  - Print who has highest hand-ranking.
- `void printChangedCards(int numOfReplacedCards, int cardPosition[]);`
  - Print the changed cards information.
- `void printResult(int** dealerHand, int** playerHand);`
  - Print the winner.

Functions support for other functions

- `void randomCard(int& x, int& y);`
  - Randomly generate 1 card (x is the suit and y is the rank of the card).
- `void resetArray(int* a, int lenA);`
  - Set the value of the elements in the array to 0.
- `void resetArray_2D(int** a, int row, int col);`
  - Set the value of the elements in a two-dimensional array to 0.
- `int findMax(int* arr, int n);`
  - Find the largest number in the array.
- `int findMax_2D(int arr[SUITS][FACES], int row, int col);`
  - Find the largest number in a two-dimensional array.
- `int findTopCard(int deck[SUITS][FACES]);`
  - Find the card at the top of the deck (find the smallest nonzero number in a two-dimensional array).
- `void copyHandAToHandB(int** A, int** B);`
  - Copy data from hand A to hand B.
- `void copyDeckAToDeckB(int A[SUITS][FACES], int B[SUITS][FACES]);`
  - Copy data from deck A to deck B.
- `void addHandLine(wstring handLine[], int suit, int face);`
  - Visualize the card, convert to a card shape.
- `bool checkDrawn(int drawnCard[], int cardPosition);`
  - Check and return true if the card has not been drawn yet.
- `void Swap(int& a, int& b);`
  - Swap 2 numbers.

- `void sortHand(int** hand);`
  - Arrange cards on hand with suit and face in ascending order.
- `void pokerArt();`
  - Print the words "POKER" in the menu interface.
- `void endArt();`
  - Print the words “SEE YOU AGAIN” when exiting the program.

NOTE: *- The functions are stored in the file Function.cpp, Function.h*
- The game items are stored in Menu.cpp, Menu.h

III. EXTENTION

1. Menu selection

Select by using arrow keys instead of inputting data, referenced from [2].
Functions referenced from [1]:

- `void GoTo(int posX, int posY);`
  - Move cursor position to position (x, y).
- `void ShowCur(bool CursorVisibility);`
  - Hide/show the cursor position.

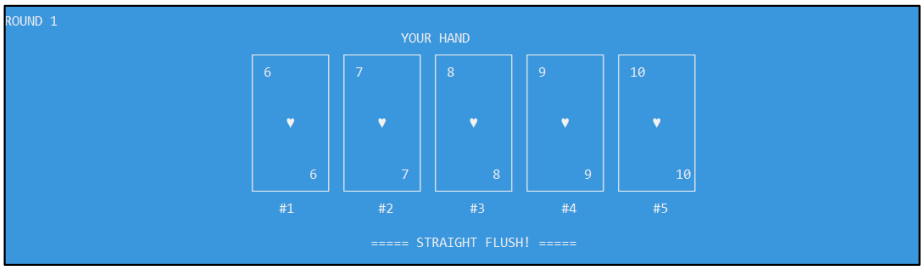
2. Console background and font color

Function referenced from [1]:

- `void SetColor(int backgound_color, int text_color);`
  - Set the background and text color.

3. Print the card shape

Declare the library <io.h> and <fnctl.h>, use `wstring` instead of `char*` to print the characters "♠, ♣, ♦, ♥" in the Unicode table. The card shape is referenced from [3].



IV. REFERENCE

[1] https://codelearn.io/sharing/windowsh-ham-dinh-dang-noi-dung-console

[2] https://www.youtube.com/watch?v=OegTA6QrbDw

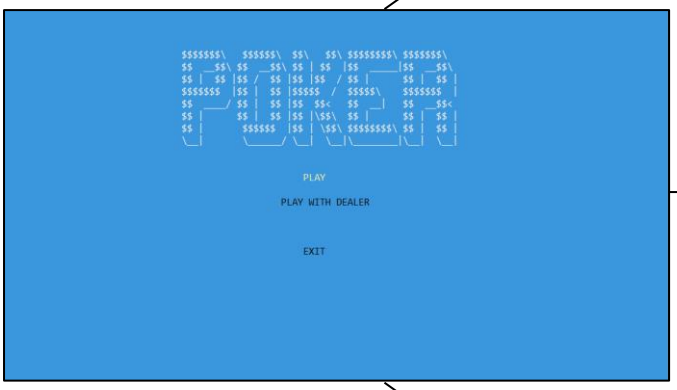
[3] https://codereview.stackexchange.com/questions/82103/ascii-fication-of-playing-cards

[4] https://patorjk.com/software/taag

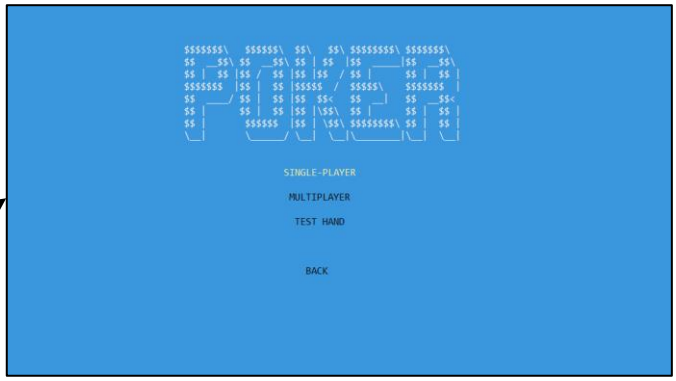
V. APPENDIX

(Game interface images)

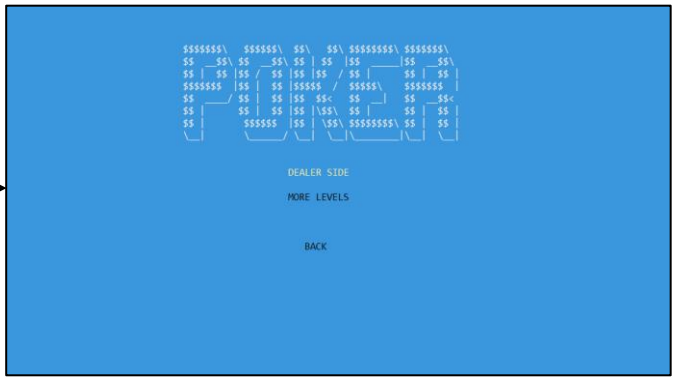
MENU



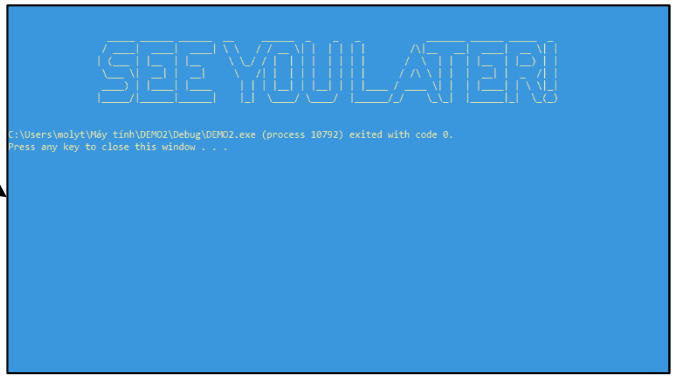
(1) PLAY



(2) PLAY WITH DEALER



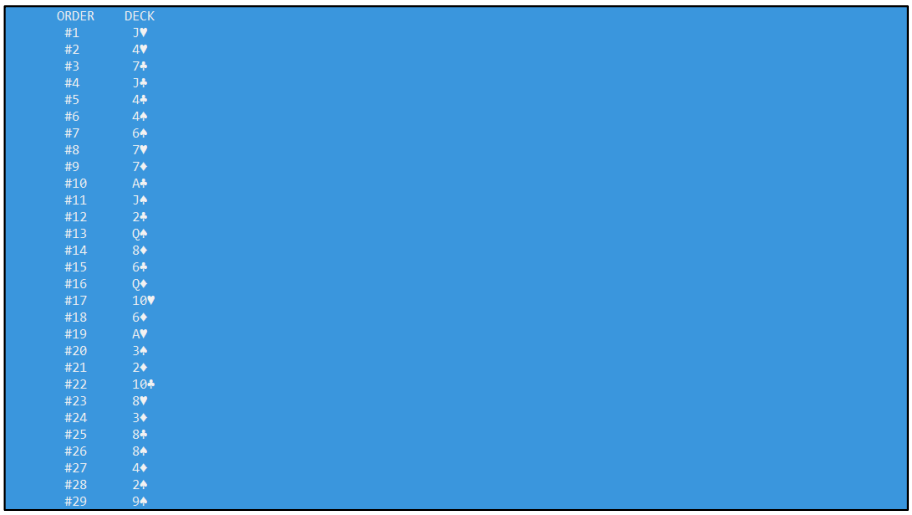
(3) EXIT



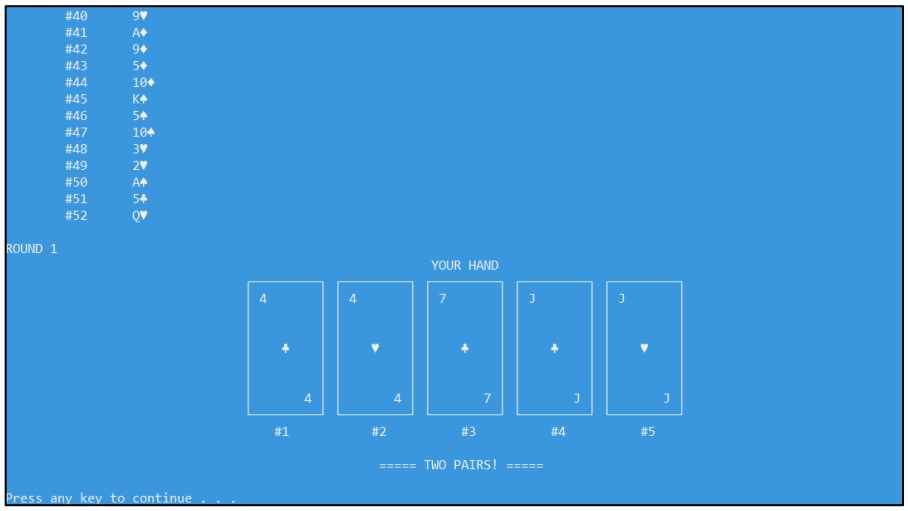
1. Play

1.1. Single-Player

Enter the number of games



Print the deck

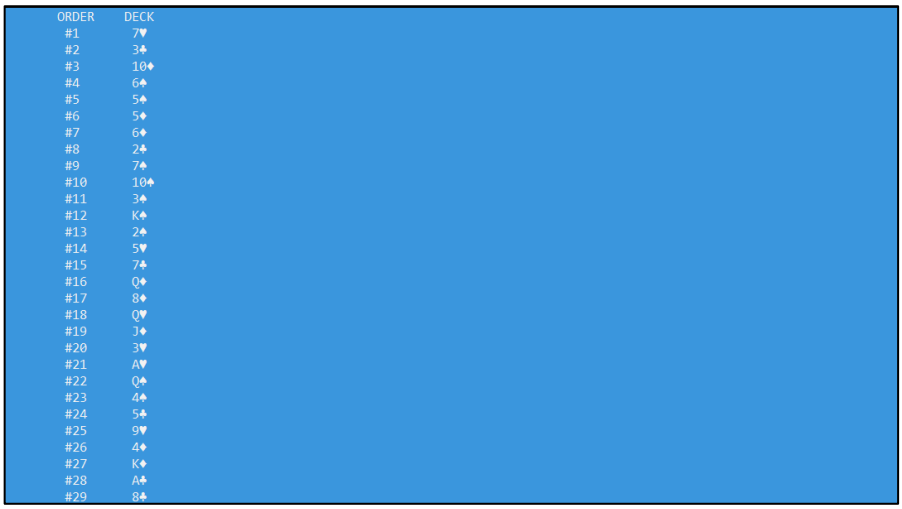


Print player's hand

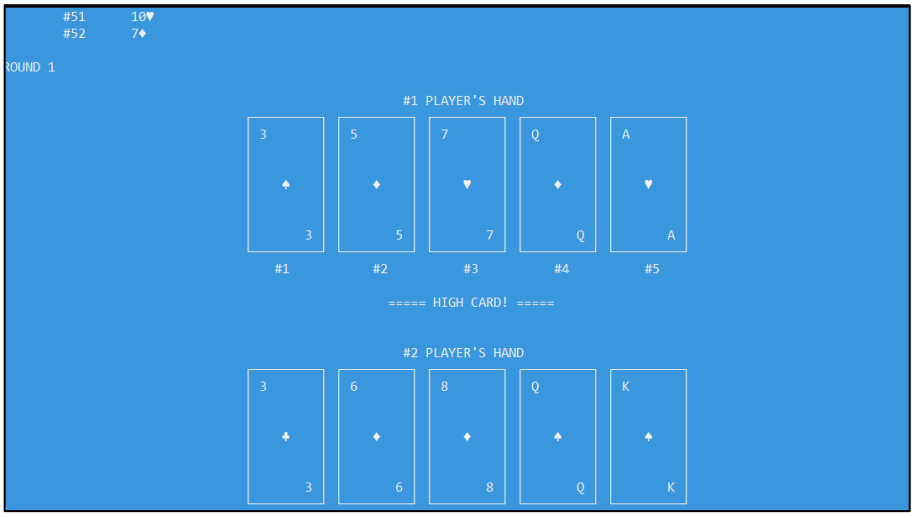
1.2. Multiplayer

Enter the number of games

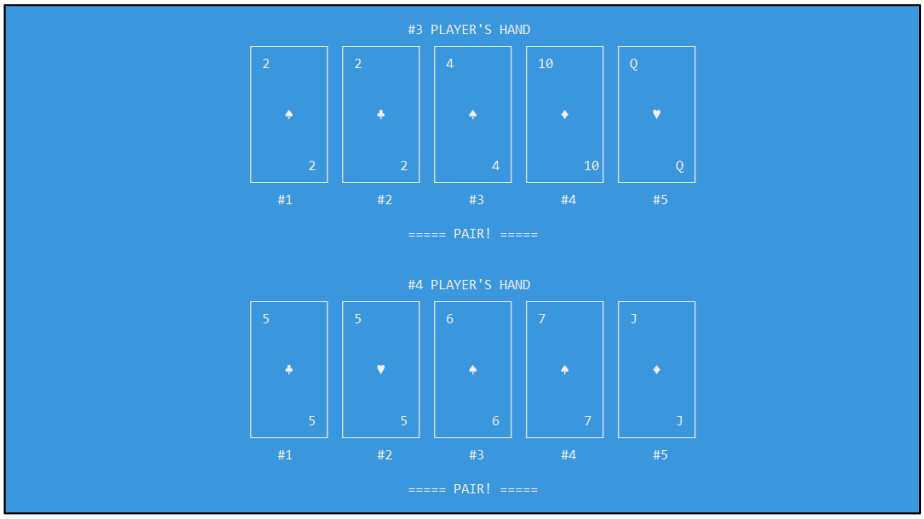
Enter the number of players



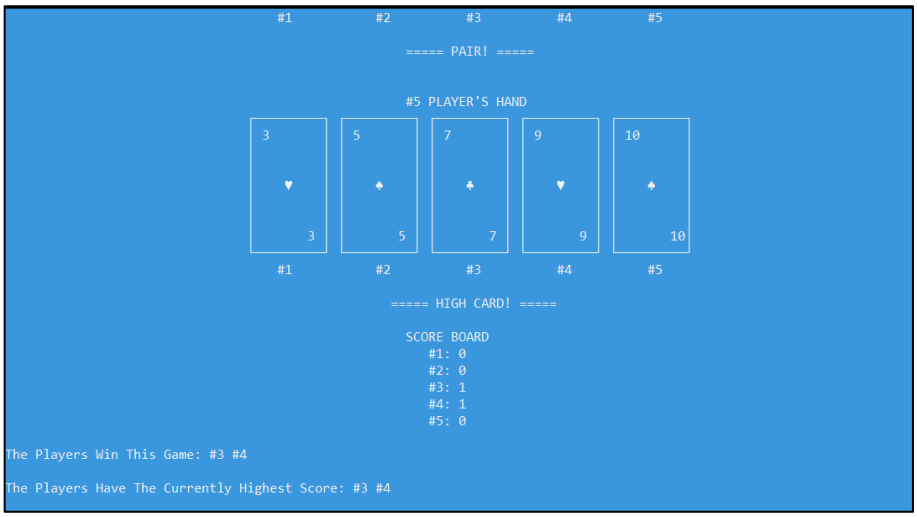
Print the deck



Print all player's cards

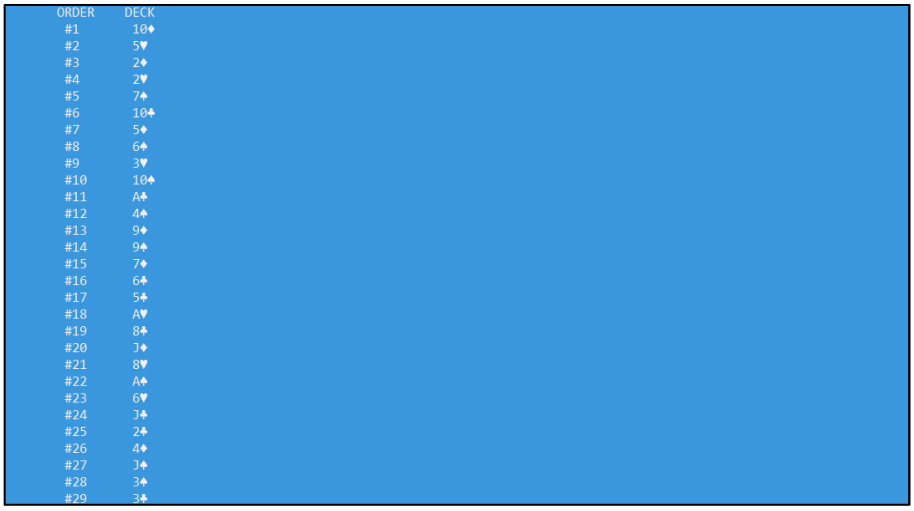


Print all player's cards

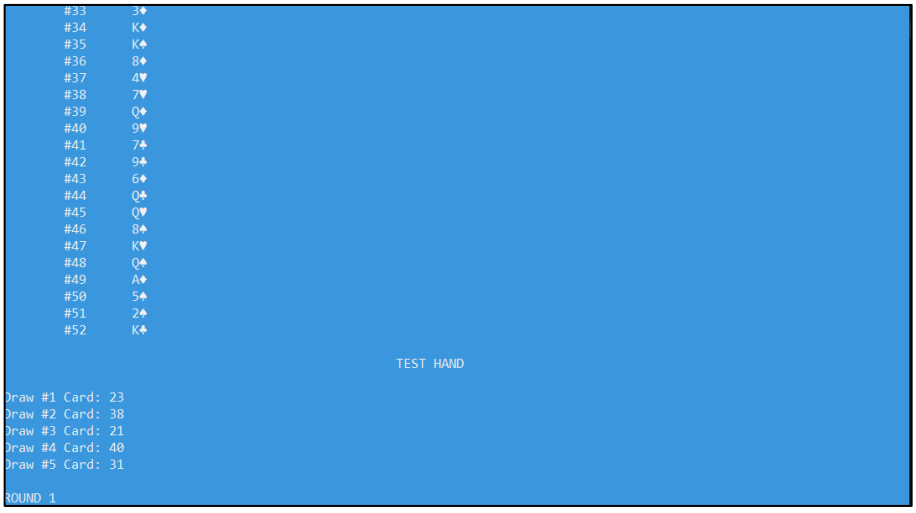


Print hand-ranking and the winner

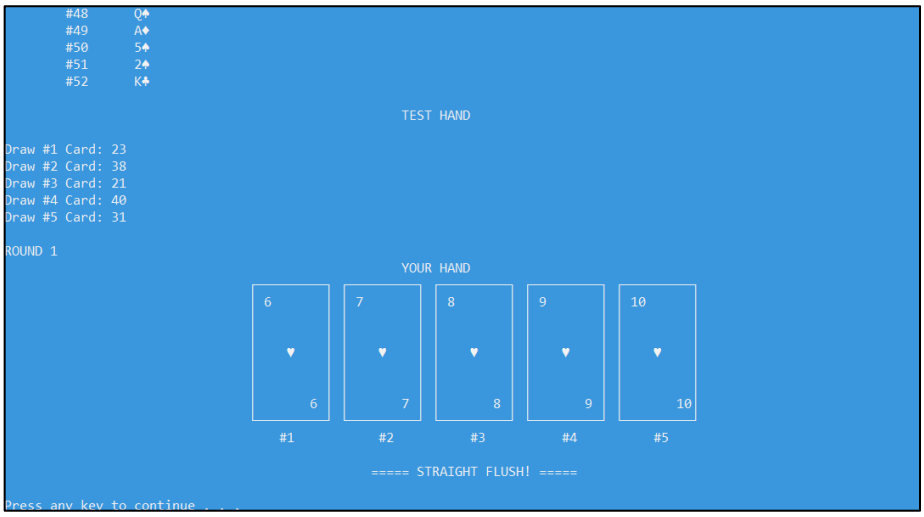
1.3. Test Hand



Print the deck



Enter the order of the cards to be drawn

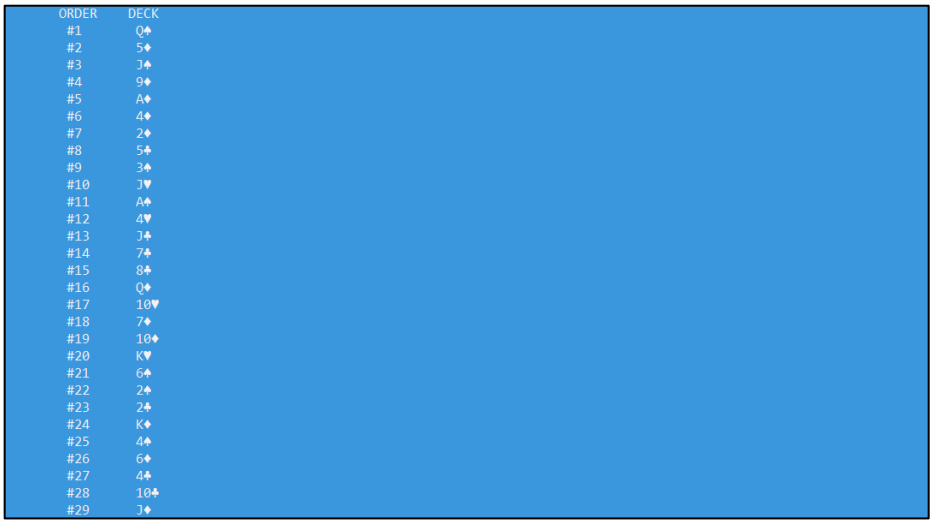


Print player's hand

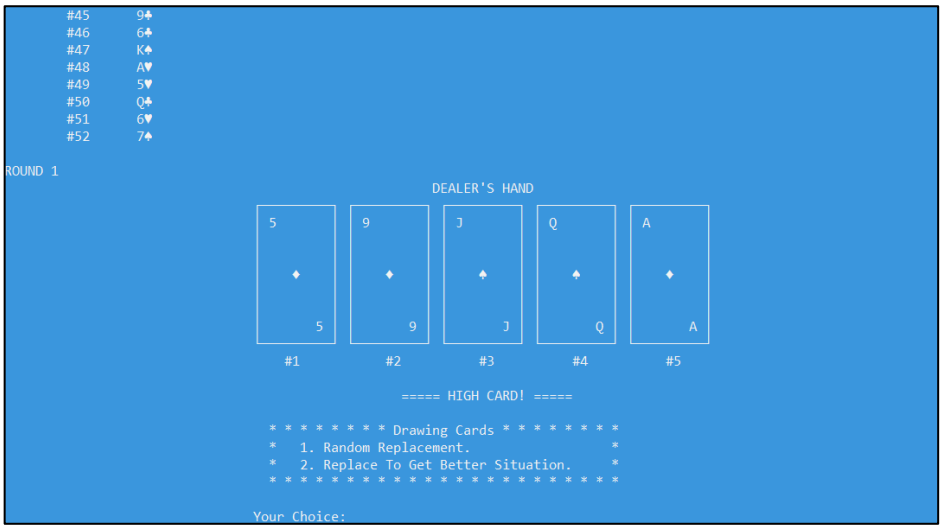
2. Play With Dealer

2.1. Dealer Side

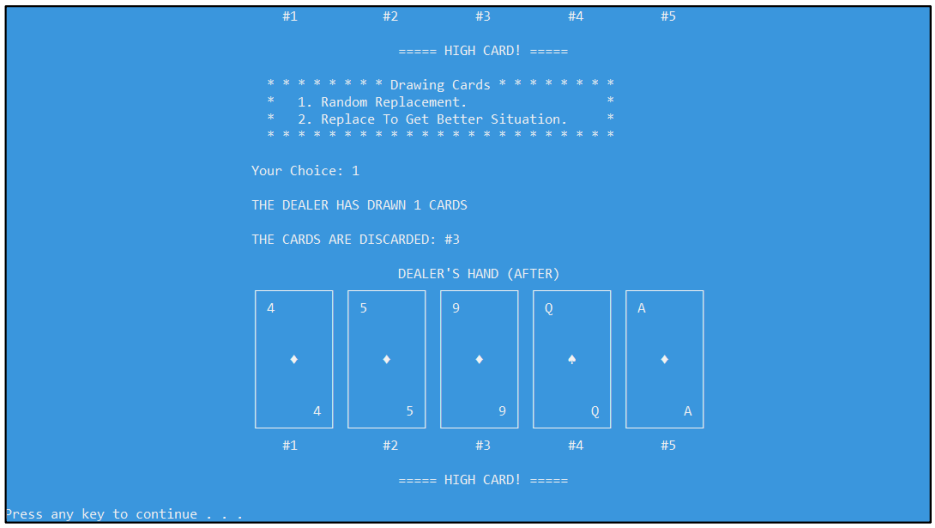
Enter the number of games



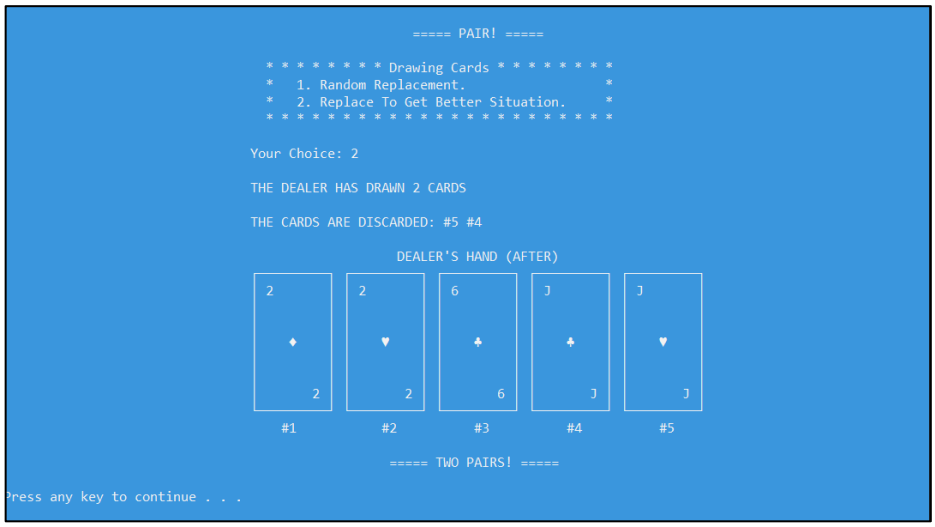
Print the deck



Print dealer's hand and drawing options



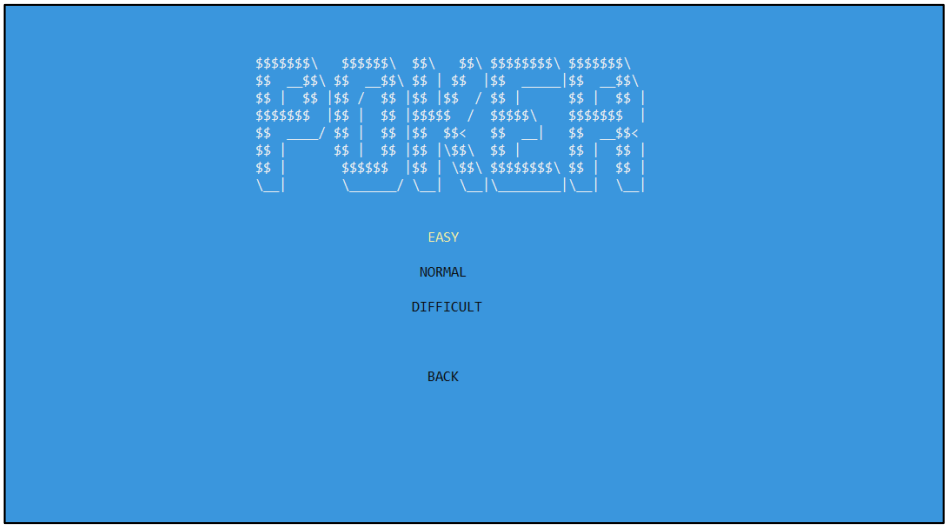
(1) Random replacement



(2) Replace to get better situation (from Pair to Two Pairs)

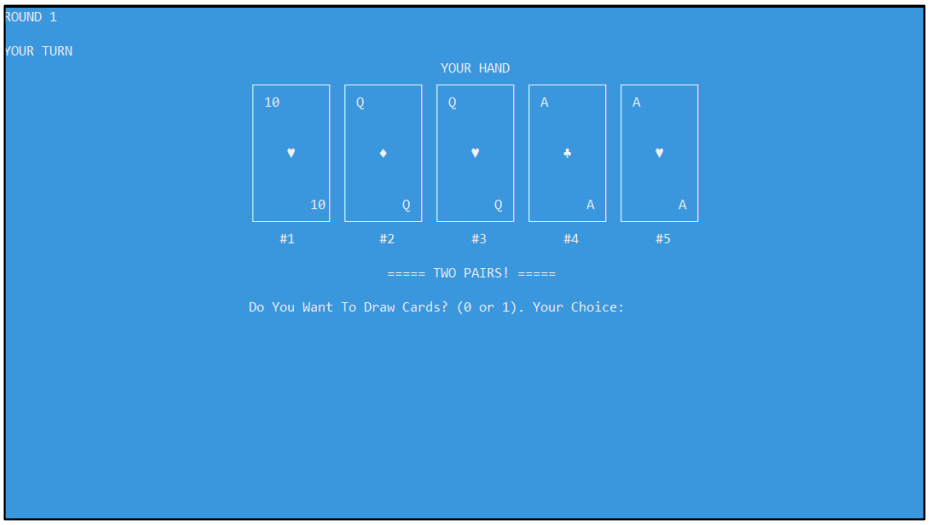
2.2. More Levels

2.2.1. Easy *(Only the player can draw cards)*

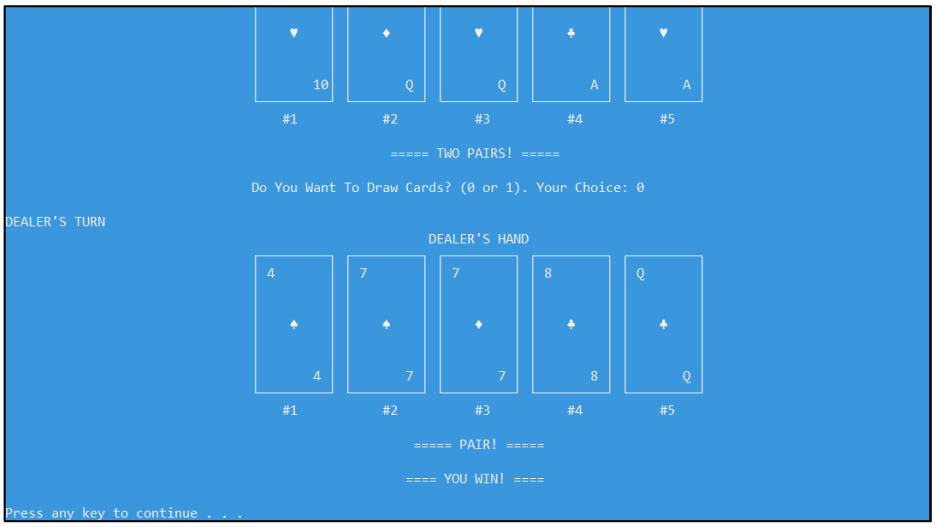


Choose Easy level

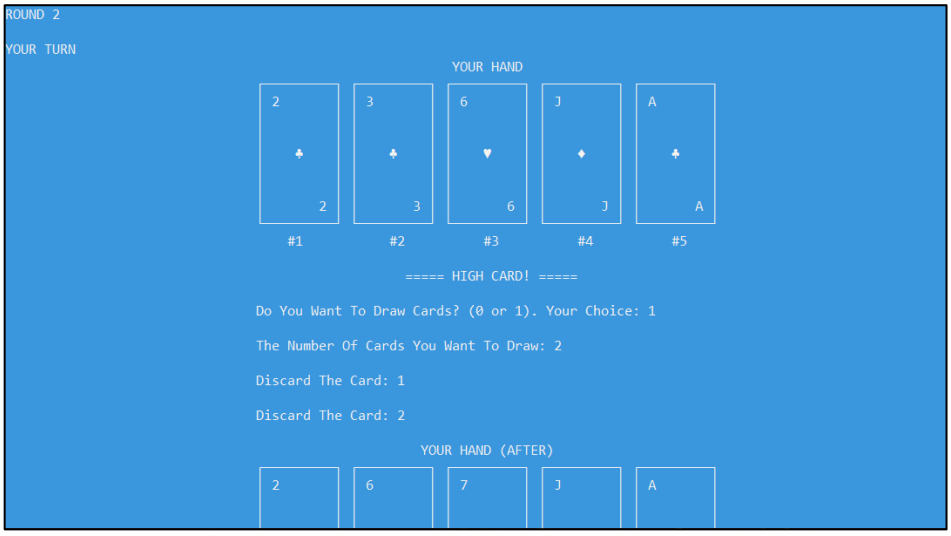
Enter the number of games



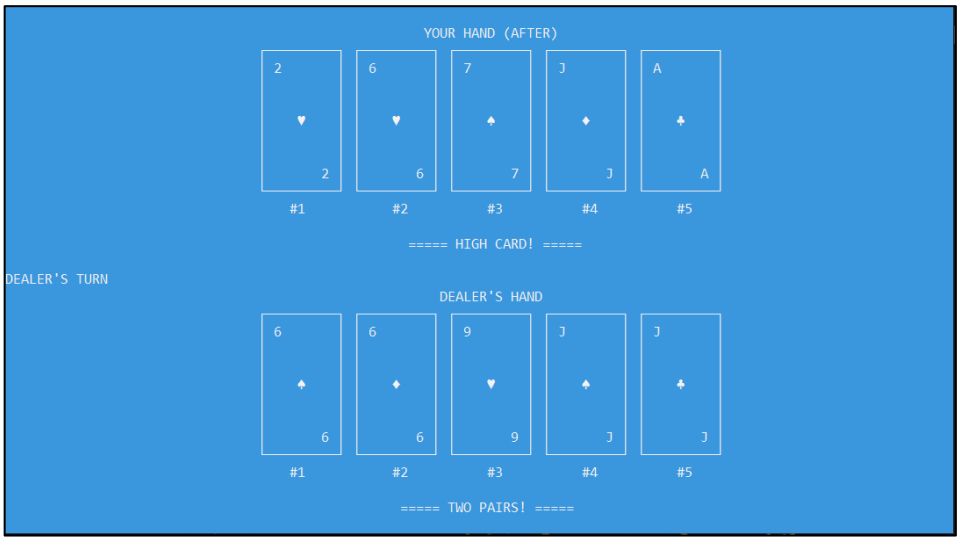
Ask the player to draw cards



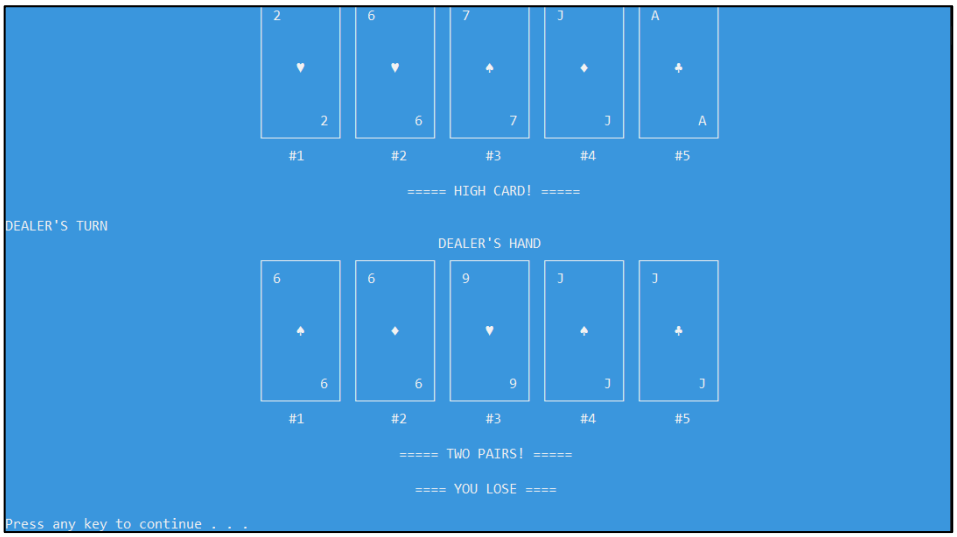
Choose not to draw, then print the result



Choose to draw

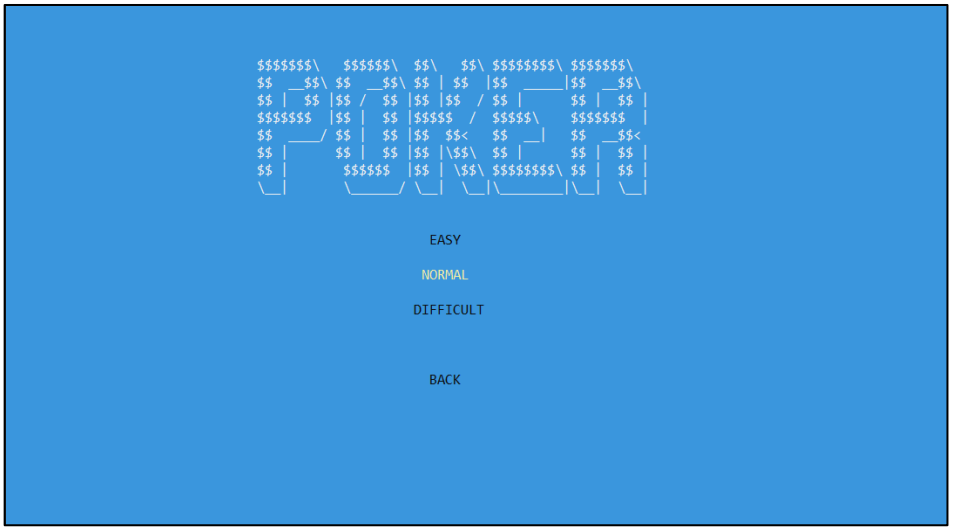


Print player's hand after drawing and dealer's hand



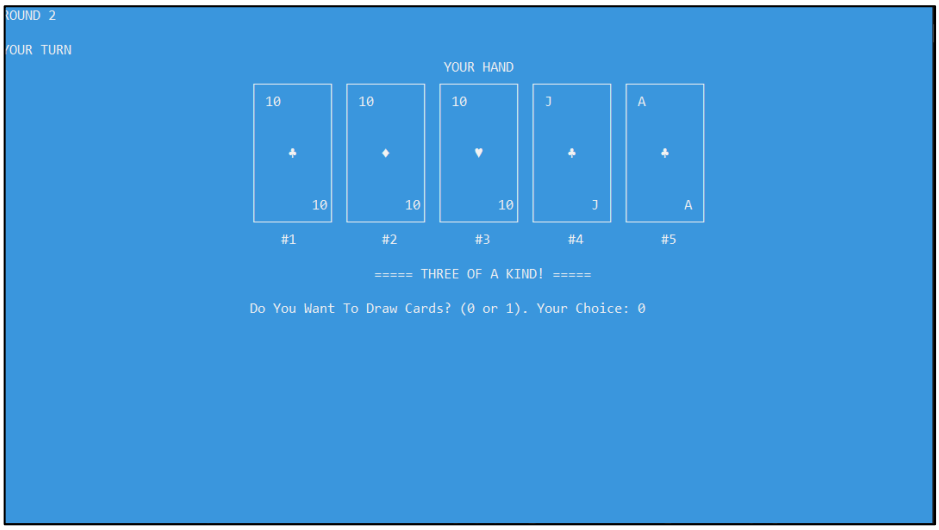
Print the result and go to the next round

2.2.2. **Normal** *(Both player and dealer can draw cards)*

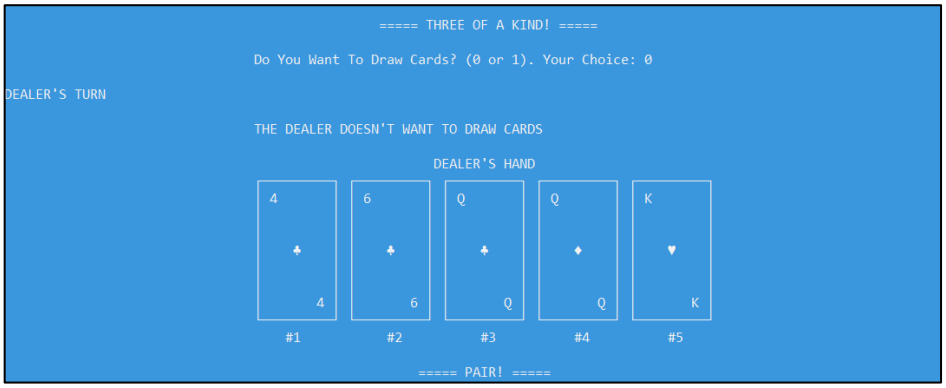


Choose Normal level

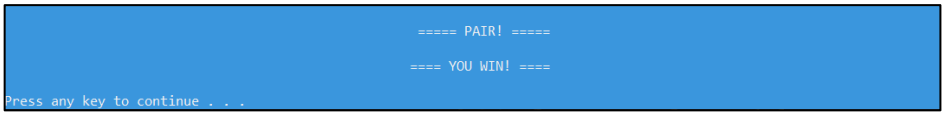
Enter the number of games



Ask the player to draw

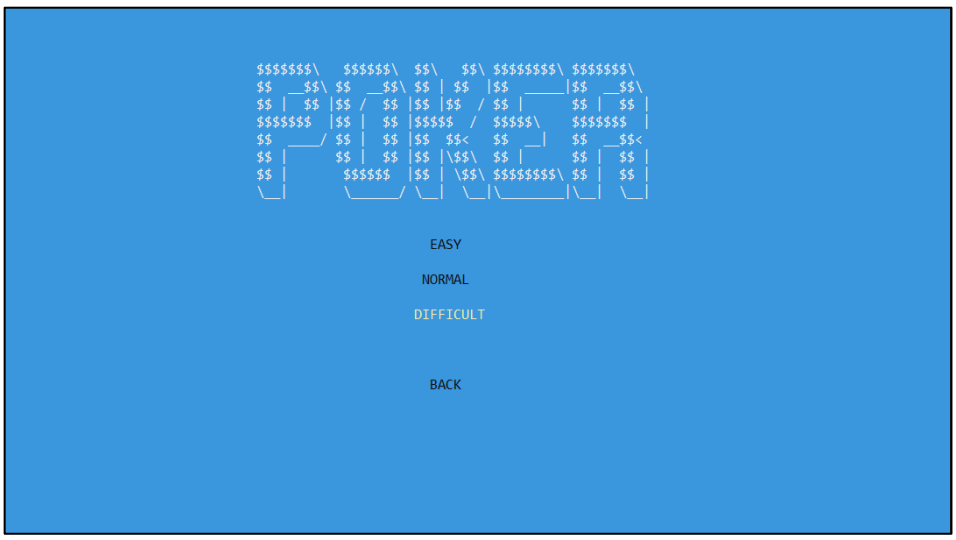


The dealer can decide to draw or not, then print the dealer's hand



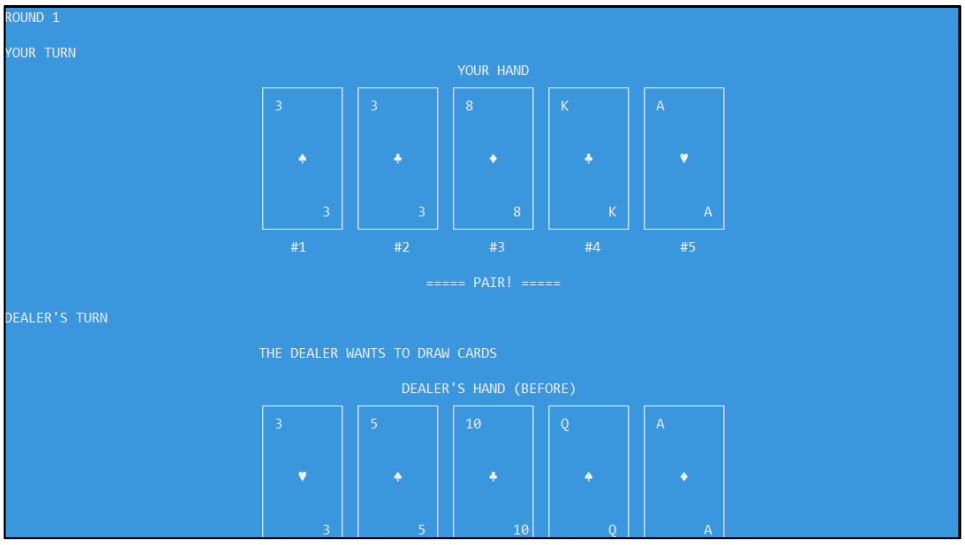
Print the result and go to the next round

2.2.3. **Difficult** *(Only the dealer can draw cards)*

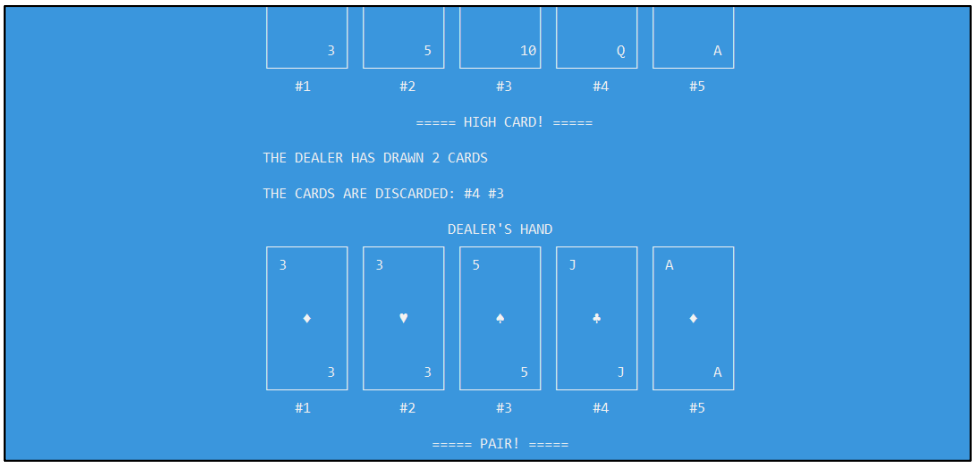


Choose Difficult level

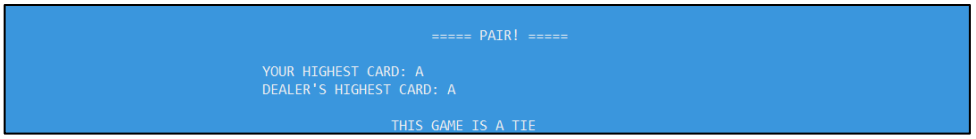
Enter the number of games



Print player's hand



Ask the dealer to draw, then print the dealer's hand after drawing



Print the result and go to the next round