CPE Group Assignment

See who's fastest

Chuheng Zhou cz2612, Jiacheng Xia jx2467, Bingrui Su bs4911

1. Introduction

The history of recreation dates back to early human civilization. For thousands of years, the ways humans entertain themselves continuously develop. Since the invention of poker in the 19th century, poker playing has become one of the most popular forms of entertainment. In various places, however, the way poker is played differs: blackjack, spider solitaire, Texas hold 'em, etc. For our project, we have developed an original poker game: see who's fastest.

The rule of this game is as follows: Firstly, only one deck of cards will be used, 54 cards including two jokers. Then, each player will be assigned 18 random cards, and the system will randomly select one player to start, say it's player two, then moving to player 3, to player 1, and it loops forever. The player starts by discarding cards. The acceptable card type to be discarded include single, double, triple, or quadruple (e.g. single 5, double 6, etc). Then, the next player has to discard cards that's bigger than the previous user. Order in this game is determined by two factors, value and quantity. When two cards equal in quantity, value is compared; otherwise, always the card with more quantity wins over. As a side note, double joker is considered the biggest of all cards in this game. The detailed ordering used in the game is given as below.

1<2<...<red joker<double 1<double 2<...<double k<triple 1<triple 2<...<triple k<quadruple 1<quadruple 2<...<quadruple k<double joker.

Whenever one player has no card bigger than the previous, that player may pass (or, the user could just pass even though he/she has bigger cards). When two players pass, then the player discarding the biggest card gets a free move, meaning he/she may start discarding any card he/she wants, so as to start over the loop again. It goes on until one player discards all cards, and that player is the winner.

Therefore, the objective of our project is to achieve this design through c++, so that the user of our program can play with either two other players, one player with one AI, two AIs, or simply watching three AIs playing against each other.

2. Project development

2.1. Pseudocode:

Declare global variable PLAYERNUM to be 3
Define Enumeration Suit with the following Values:
    Hearts has value 0
    Diamonds has value 1

Clubs has value 2
Spades has value 3
Joker has value 4

Create class card
Set the following to be private
Suit variable suit
Integer variable value

Set the following to be public
Constructor Card:
Set suit to Hearts (default value of 0)
Set value to 0

Constructor Card with parameters s (for suit) and v (for value):
If s is between 0 and 4 inclusive and v is between 0 and 14 inclusive:
Set suit to suit corresponding to integer s
Set value to v
Otherwise:
Print"err"
Set suit to Hearts (default value of 0)
Set value to -1

String function getSuitAsSymbol
If suit is Hearts, return “H”
If suit is Diamonds, return “D”
If suit is Clubs, return “C”
If suit is Spades, return “S”
If suit is Joker, return “Joker”
Default case: return “Unknown suit”

Integer function getValue
Return value

Bool Function Operator < (other Card):
If value of this card is not equal to value of other card:
Return True if value of this card is less than value of other card
Otherwise Return False
Otherwise

Return True if integer representation of suit of this card is less than that of other card

Otherwise return False

Bool Function Operator > (other Card):

If value of this card is not equal to value of other card:

Return True if value of this card is greater than value of other card

Otherwise Return False

Otherwise

Return True if integer representation of suit of this card is greater than that of other card

Otherwise return False

Create class Participant

Set the following to be protected

Create a vector named cards to store objects of type Card

Set numOfCards to 0 (an integer of the number of Card objects in the vector)

Declare a string named name (to store the name of participant)

Set the following to be public

Create void function SetCards with parameter value (a Card):

Add value to the end of the cards vector

Create integer function GetCardNum

Return numOfCards

Create string function GetName

Return name

Create boolean function Validate with input of integer array called input with 2 elements

If first element of input array is 2 and second element is 13:

Declare count1 to store the number of cards with a value of 13

Declare count2 to store the number of cards with a value of 14

For each card in the cards vector:

If the value of the card is 13:

Increment count1

If the value of the card is 14:

Increment count2

Return True if count1 is exactly 1 and count2 is exactly 1

Otherwise return False

Otherwise

Declare count to store the number of cards with a value equal to the second element of input
For each card in the cards vector:
If the value of the card matches the second element of input:
Increment count
Return True if count is greater than or equal to the first element of input
Otherwise return False

Create void function MakePlay with input of integer array called input with 2 elements
If first element of input array is 2 and second element is 13:
Remove all cards from the cards vector where the card's value is 13
Then remove all cards from the cards vector where the card's value is 14

Otherwise:
Initialize count to 0
Remove cards from the cards vector such that:
If the count is less than the first element of input and the card's value equals the second element of input:
Increment count
Remove the card
If the count reaches the first element of input, stop removing further cards

Define Function FreeMove:
This function performs a free move operation
Returns null (no specific card or move is returned)

Define Function Move with parameter prevCard (an array of pointers to integers, size of PLAYERNUM):
This function performs a move operation based on the previous moves of other players (given by prevCard)
Returns null (no specific move or card is returned)

Define Class Player as a subclass of Participant:

Set the following to be public

Constructor for Player with parameter i (an integer identifier):

Initialize the cards list to be empty

Set numOfCards to 54 divided by PLAYERNUM (allocating an equal share of cards to each player)

Set name to "Player " concatenated with the value of i plus 1 (to make the player numbering start from 1)

Function FreeMove:

Create a new string array named input with 2 elements

Create a new integer array named castedInput with 2 elements

Display the current player's cards

Loop indefinitely:

Prompt the player, indicating no one contests their move, to specify what they want to play

Accept two values into the input array

While the input is not valid:

Inform the player that the input was invalid

Prompt again for two values

Cast the first input value to an integer

Convert the second input value from a symbol to its corresponding value

Delete the input array to free memory

If the provided castedInput is valid according to game rules:

Execute the play using the castedInput values

Decrease numOfCards by the number indicated in castedInput[0]

Exit the loop

Otherwise

Inform the player that the input was invalid

Announce the move played using castedInput values

Return the castedInput array

Function Move with parameter prevCard (an array of integer pointers, size of PLAYERNUM):

Initialize integer variable i to 0
Loop from 0 until a non-null entry is found in prevCard:
Increment i
Declare prevEnter as the first non-null entry from prevCard

Create a new string array named input with 2 elements
Create a new integer array named castedInput with 2 elements

Print current player's name and "'s cards:"
Call function DisplayHand with cards

Repeat indefinitely:
Print "Previous player played ", prevEnter[0], " number of ", getSymbolUsingValue(prevEnter[1]), ". What do you want to play?"
Read user input into input[0] and input[1]

While input is not valid:
Print "Invalid Input"
Print "Previous player played ", prevEnter[0], " number of ", getSymbolUsingValue(prevEnter[1]), ". What do you want to play?"
Read user input into input[0] and input[1]

Convert input[0] to an integer and store in castedInput[0]
Convert input[1] from a symbol to its corresponding value and store in castedInput[1]

If castedInput[0] is -1:
Return nullptr

If the input is valid (Validate(castedInput)):
If castedInput is a valid and strategic move (comparing with prevEnter):
Call MakePlay with castedInput
Decrease numOfCards by the number indicated in castedInput[0]
Break from the loop
Otherwise

Print "Card Input Invalid"

Print "Played ", castedInput[0], " number of ", getSymbolUsingValue(castedInput[1])
Delete pointer prevEnter
Return castedInput

Define Class Computer as a subclass of Participant:

Constructor for Computer with parameter i (an integer identifier):
Initialize the cards list to be empty
Set numOfCards to 54 divided by PLAYERNUM (allocating an equal share of cards to each player)
Set name to "(AI)player " concatenated with the value of i plus 1 (to make the player numbering start from 1)

Function FreeMove (overridden):
Print "No one contests you. What do you want to play?"
Sort the cards
Create a new integer array named input with 2 elements
Set input[0] to 1 (indicating playing one card)
Set input[1] to the value of the first card in the sorted list
Call function MakePlay with input
Decrease numOfCards by 1
Announce "Played [number of cards] number of [card value]"
Return the input array

Function Move (overridden) with parameter prevCard (an array of integer pointers, size of PLAYERNUM):
Initialize integer pointers input, bJoker, and rJoker
Set bJoker to {1, 13} and rJoker to {1, 14} (special card values)
Sort the cards
Find the first non-null entry in prevCard and store it in prevEnter
Announce the move of the previous player

Check for playing a higher card with the same count:
Loop from the value of the previous card plus one to 14:
Set input to current loop values
If input is valid:
Call MakePlay with input

Decrease numOfCards by the played count
Announce the played move
Clean up and return input

Check for playing a greater number of cards:
Nested loops to iterate over possible counts and values:
Set input to current loop values
If input is valid:
Call MakePlay with input
Decrease numOfCards by the played count
Announce the played move
Clean up and return input

Check for playing double jokers:
If both jokers are valid:
Set input to {2, 13}
Call MakePlay with input
Decrease numOfCards by 2
Announce the played move
Clean up and return input

If no valid move is found:
Announce "Played nothing."
Return null

Main Function
Create an array named participants of size PLAYERNUM to store pointers to Participant objects
Create a vector named cards and fill it with a generated deck of cards
Define and initialize a boolean variable gameOn to true
Create an array named prevCard of size PLAYERNUM to store pointers to integers
Repeat the following for i from 0 to 2:
Set the i-th element of prevCard to null
Declare an integer variable named index
Declare a variable named winner of type Participant pointer
Define and initialize an integer variable choice1 to -1

Repeat the following while it is true
Clear the input buffer to remove any erroneous data
Ignore any remaining input until the end of the current line

```
            Print “Invalid input”, newline
            Print “Welcome to the game of ”, newline
            Print “1) Start Game”, newline
            Print “2) Help”, newline
            Print “3) Exit”, newline
            Read user input into choice1

        If choice1 equals 1,
            Set gameOn to be true

            Declare integer choice2
            Repeat the following three times (i going from 0 to 2)
                Print “Add a player (0: AI, 1:Player): ”
                Read user input into choice2
                If choice2 equals 1
                    Add new Player object
                Otherwise
                    Add new Computer object

            Repeat the following while gameOn is true
                cards = Shuffle(cards);
                Call function SplitCards with inputs cards and participants

                Seed the random number generator function
                Assign a random number from 0 to 2 to index
                Print name of player index % PLAYERNUM and “got the first move!”,
            newline
                player index % PLAYERNUM gets a free move;
                Index increase by 1

                Repeat the following while it is true
                    Print newline, name of player index % PLAYERNUM, and “‘s
turn!”, newline
                    If the prevCard[index % PLAYERNUM]th player’s previous play
is not covered
                        prevCard[index % PLAYERNUM]th player gets a free
move
                    Otherwise
                        prevCard[index % PLAYERNUM]th player gets a regular
move
```

If prevCard[index % PLAYERNUM]th player played cards
Repeat the following for 2 times, i going from 0 to 1
Cover the next player
If participants[index % PLAYERNUM]th card has no more cards
Assign participants[index % PLAYERNUM] to winner
Break
Index increases by 1
Print “Winner is: ” and winner’s name, newline
Assign false to gameOn
Repeat the following 3 times (i going from 0 to 2)
Delete ith element of dynamically allocated array participants
Otherwise if choice1 equals 2
Print “Press 1 in the main menu to enter game. The software will give the user 18 cards (from one deck of card) to each player (including two jokers), then a randomly selected player will start the game by discarding one or more cards.The acceptable form of card discarded in this game is only one single card, two or more cards with the same value(e.g. double 5, triple 9, quadruple k, double joker, etc.)Then, the next player(order determined by the loop of user, computer 1, and computer 2; if, for instance, computer 1 is the first player to discard a deck, the next one will be computer 2, then the user, etc.) shall discard cards that’s either 1. bigger in value(e.g. 1 < 2 < ... < k < black joker < red joker) or 2. bigger in quantity of the same card, and this process continues on for every next player.Notice that quantity is always considered before number(e.g.single 5 < double 4 < triple 3 < quadruple 2 < double joker) so that whenever one player discards one single card, another player can discard double cards regardless of the value of the card, and similar things hold for triple cards and quadruple cards.Double joker is considered the biggest card(although with only two cards, double joker is considered bigger than all quadruple cards and so on, this is an exception).For cards in the same quantity, the value of the card is compared(e.g. double 4 < double 6, triple j < triple k, quadruple 5 < quadruple 9, etc.) Overall, the whole pattern of every card is as follows\n1 < 2 < ... < red joker<double 1 < double 2 < ...<double k < triple 1 < triple 2 < ... < triple k < quadruple 1 < quadruple 2 < ... < quadruple k<double joker.\nIf the next player has no card bigger than the previous user, the player shall pass.After the other two players both pass, the player who discarded the biggest card may discard any new card and this process continues on and on(e.g. if one player discards double joker, that player gets to discard a new card by his / her / its preference since double joker is the biggest card, and the other two players can do nothing but pass).The game ends when one of the three players discards all cards, and that player is the winner.”, newline
Otherwise if choice1 equals 3
Print “Exiting game…”, newline
Break
Otherwise
Print “Invalid Choice”, newline

Delete the dynamically allocated array participants
Repeat the following 3 times (i going from 0 to 2)
Delete ith element of dynamically allocated array participants
Delete the dynamically allocated array prevCard
Return 0

Function GenerateDeck:
Create a vector named cards
space for 54 cards in the vector

Loop over suits from 0 to 3:
Loop over values from 0 to 12:
Add a new Card to cards with current suit and value

Add a Joker card to cards with suit 4 and value 13
Add another Joker card to cards with suit 4 and value 14

Return the cards vector

Function Shuffle with parameter deck (a vector of Card):
Create a random number generator seeded from hardware
Shuffle the cards in the deck using the random generator
Return the shuffled deck

Create void function SplitCards with inputs cards(vector of Card objects), array named participants of size PLAYERNUM to store pointers to Participant objects
Repeat the following while i, initialized as 0, increase by 1 when i is less than 18
Repeat the following while j, initialized as 0, increase by 1 when j is less than 3
Jth participant gets 54 / PLAYERNUM * j + ith element of cards array

Function DisplayHand with parameter cards (a vector of Card):
Sort the cards using SortCards function

Print a series of "----" for each card to create the top border of the card display
Print "------" to complete the line

Loop through each card in cards:
If card is a Black Joker:
Print "|B "
Otherwise if card is a Red Joker:

```
                Print "|R  "
            Otherwise
                If card value is "10":
                    Print "|" and card value and a single space
                Otherwise:
                    Print "|" and card value and two spaces
        Print "     |" to end the line

        Loop through each card in cards:
            If card is a Joker:
                Print "|J  "  (indicating Joker)
            Otherwise
                Print "|" and card suit symbol and two spaces
        Print "     |" to end the line

         Loop through each card in cards:
            If card is a Joker:
                Print "|O  "
            Otherwise
                Print "|   " (spaces for non-Joker cards)
        Print "     |" to end the line

        Repeat similar loops for additional lines of display for Jokers:
            For BJoker or RJoker, print "|K  ", "|E  ", and "|R  " respectively for different lines
            For non-Jokers, continue to print "|   " for each card
            Each time, finish the line with "     |"

        Print a series of "----" for each card to create the bottom border of the card display
        Print "------" to complete the line

Create string function getSymbolUsingValue with input of integer variable value
    If value is
        0, return “A”
        1, return “2”
        2, return “3”
        3, return “4”
        4, return “5”
        5, return “6”
        6, return “7”
        7, return “8”
```

8, return “9”
9, return “10”
10, return “J”
11, return “Q”
12, return “K”
13, return “BJoker”
14, return “RJoker”
By default, return “Unknown Value”

Create integer function getValueUsingSymbol using a constant string variable input of name symbol

If symbol is A, return 0
Otherwise if symbol is 2, return 1
Otherwise if symbol is 3, return 2
Otherwise if symbol is 4, return 3
Otherwise if symbol is 5, return 4
Otherwise if symbol is 6, return 5
Otherwise if symbol is 7, return 6
Otherwise if symbol is 8, return 7
Otherwise if symbol is 9, return 8
Otherwise if symbol is 10, return 9
Otherwise if symbol is J, return 10
Otherwise if symbol is Q, return 11
Otherwise if symbol is K, return 12
Otherwise if symbol is BJoker, return 13
Otherwise if symbol is RJoker, return 14
Otherwise, return -1

Create boolean function checkInput with input of a string array with two elements

Initialize integer start to 0

If the first character of input[0] is '+' or '-':
    Set start to 1
    If the length of input[0] is 1:
        Return false
For each index i from start to the length of input[0] - 1:
    If the character at index i in input[0] is not a digit:
        Return false

Return true

2.2. Object-Oriented Programming

For this project, Object Oriented Programming(OOP) is used so that the code is more organized and straightforward. The objects include cards, which is the imaginary medium we are playing the game with, and participants, which are the players playing the game. For the cards, it does not have many functions, as it is only the medium of the game, not a major element that contains actions in the game. Therefore it only has getters for its private variables and comparison operators overridden, as we will need a convenient way to compare the values of cards. As for participants, apart from getters and setters for protected variables, participants will also need to make plays, that is play cards and decrement cards from participants' hand. Also, the participants will need to verify whether they have the cards they input. Participants will also need to be able to make plays. Among participants, there are two types of such classes, which are computers and players. These two types are coded as classes inheriting the participant class, using all its functions and its protected variables. They also override the make-play functions of the participant class because the different types of participants have different logic when it comes to making a play.

2.3. Dynamically Allocated Memory

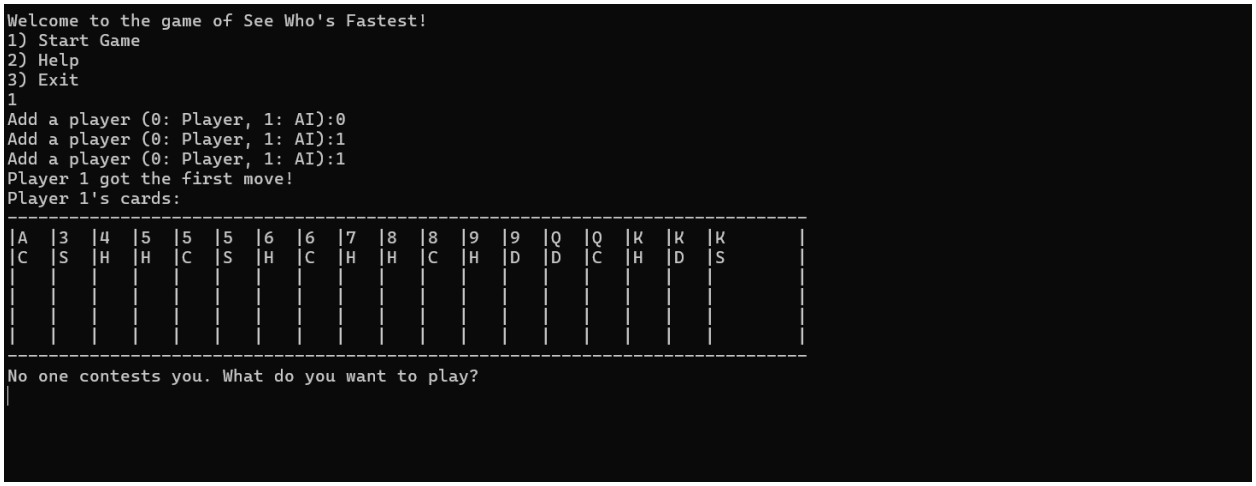
For the project, dynamically allocated memory is used for the card deck and the array of pointers to participants. The game is designed to be played with any number of players available, as long as it is a factor of 54(the total number of cards available). The memory allocated to them is then released when one game is finished.

3. Results and evaluations

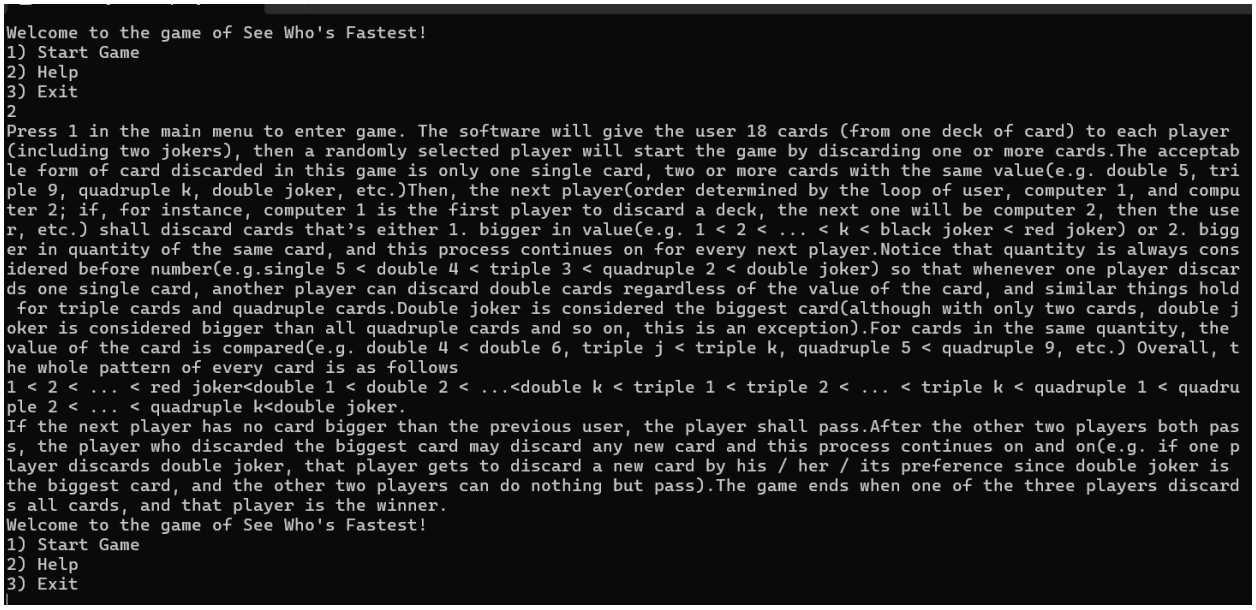
To test our program, there are various scenarios that we need to look into. Firstly, the initiation, execution and termination of the game should be tested to ensure the program can be run correctly.

Below is the Test of the Menu (including starting the game, showing help, and ending the game):

Start



Help:



Exit:

Secondly, in addition to the general process of the game, the key algorithm is the AI. However, because we did not implement dependency injection, the cards the AI has are unpredictable.

Therefore it is impossible to test all the cases. Therefore the best way we can test the AI algorithm is through equivalence partitioning of all cases based on our implementation methods, and testing one representative case for each partition. According to our algorithm, we separated the AI's output into 5 cases: The AI can make a play that has the same number of cards as the previous player but with higher value; The AI can make a play that has more cards than the previous player has regardless of value; The AI can only play double joker; the AI cannot play anything; and the AI got a free move. All these cases are tested below:

1. In general, AI chooses the card that is same in quantity and only "just" bigger than the previous card in value

Given: The previous player played single 2

Expected results: AI plays single 3, which is the smallest the AI can play, and is of larger value but same quantity.

Works because:

Algorithm:

Check for playing a higher card with the same count:

```
Loop from the value of the previous card plus one to 14:
    Set input to current loop values
    If input is valid:
        Call MakePlay with input
        Decrease numOfCards by the played count
        Announce the played move
        Clean up and return input
```

Rational:

Because the card number is set constant when looping, and the loop starts from a value that is one larger than the previous card discarded, and it goes up by one every iteration, so it makes sure that the card chosen will be the smallest larger value compared

to the previously discarded card.

```
|A  |A  |2  |3  |3  |4  |4  |5  |6  |7  |9  |9  |10 |10 |10 |10 |Q  |Q        |
|C  |S  |C  |H  |C  |D  |S  |C  |H  |D  |D  |C  |H  |D  |C  |S  |H  |C        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
--------------------------------------------------------------------------------
Previous player played 1 number of A. What do you want to play?
1 2
Played 1 number of 2

(AI)player 1's turn!
--------------------------------------------------------------------------------
|2  |3  |4  |5  |5  |6  |7  |7  |8  |8  |J  |J  |J  |Q  |K  |K  |B  |R        |
|S  |D  |H  |D  |S  |C  |H  |S  |D  |C  |D  |C  |S  |S  |H  |C  |J  |J        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |O  |O        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |K  |K        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |E  |E        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |R  |R        |
--------------------------------------------------------------------------------
Previous player played 1 number of 2. What do you want to play?
Played 1 number of 3
```

As can be seen, when the previous player discards a single 2, the AI discards a single 3

2. AI has no cards with same quantity but bigger value, thus having no choice but to discard cards that win in quantity

Given: The previous player played double J

Expected results: AI plays triple 5, which is the smallest the card AI can play, and is of larger quantity regardless of value

Works because:

Algorithm:

Check for playing a greater number of cards:

Nested loops to iterate over possible counts and values:
Set input to current loop values

If input is valid:

Call MakePlay with input
Decrease numOfCards by the played count
Announce the played move
Clean up and return input

Rational:

Because the card-number-set-constant case is run before this case, so it has been made sure that there is no same number of cards but higher value play for the AI. When looping, loop starts from a card value that is one larger than previous person entered, Then it checks from the smallest value, goes up by one every iteration, so it make sure that the card chosen will be the smallest larger play compared to the previously played, and it takes in one more or more cards than previously played by other players.

```
Player 3's turn!
Player 3's cards:
--------------------------------------------------------------------------------
|A  |2  |3  |3  |3  |3  |4  |4  |8  |9  |9  |10 |J  |J  |J  |Q  |Q  |K        |
|H  |C  |H  |D  |C  |S  |C  |S  |H  |H  |D  |D  |H  |C  |S  |D  |C  |C        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
--------------------------------------------------------------------------------
Previous player played 1 number of 4. What do you want to play?
2 J
Played 2 number of J

(AI)player 1's turn!
------------------------------------------------------------------------
|2  |4  |5  |5  |5  |6  |6  |6  |7  |8  |8  |9  |10 |J  |Q  |K  |B        |
|S  |D  |H  |D  |C  |H  |C  |S  |H  |D  |S  |C  |H  |D  |S  |S  |J        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |O        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |K        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |E        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |R        |
------------------------------------------------------------------------
Previous player played 2 number of J. What do you want to play?
1 0
Played 3 number of 5
```

As can be seen, the previous player discards double J, and the AI, having no double Q nor double K, has no choice but to discard triple 5 (which is also the smallest triple it has)

3. AI uses double joker

Given: The previous player played 4 numbers of 10

Expected results: AI plays two numbers of Joker, which is the only thing the AI can play

Works because:

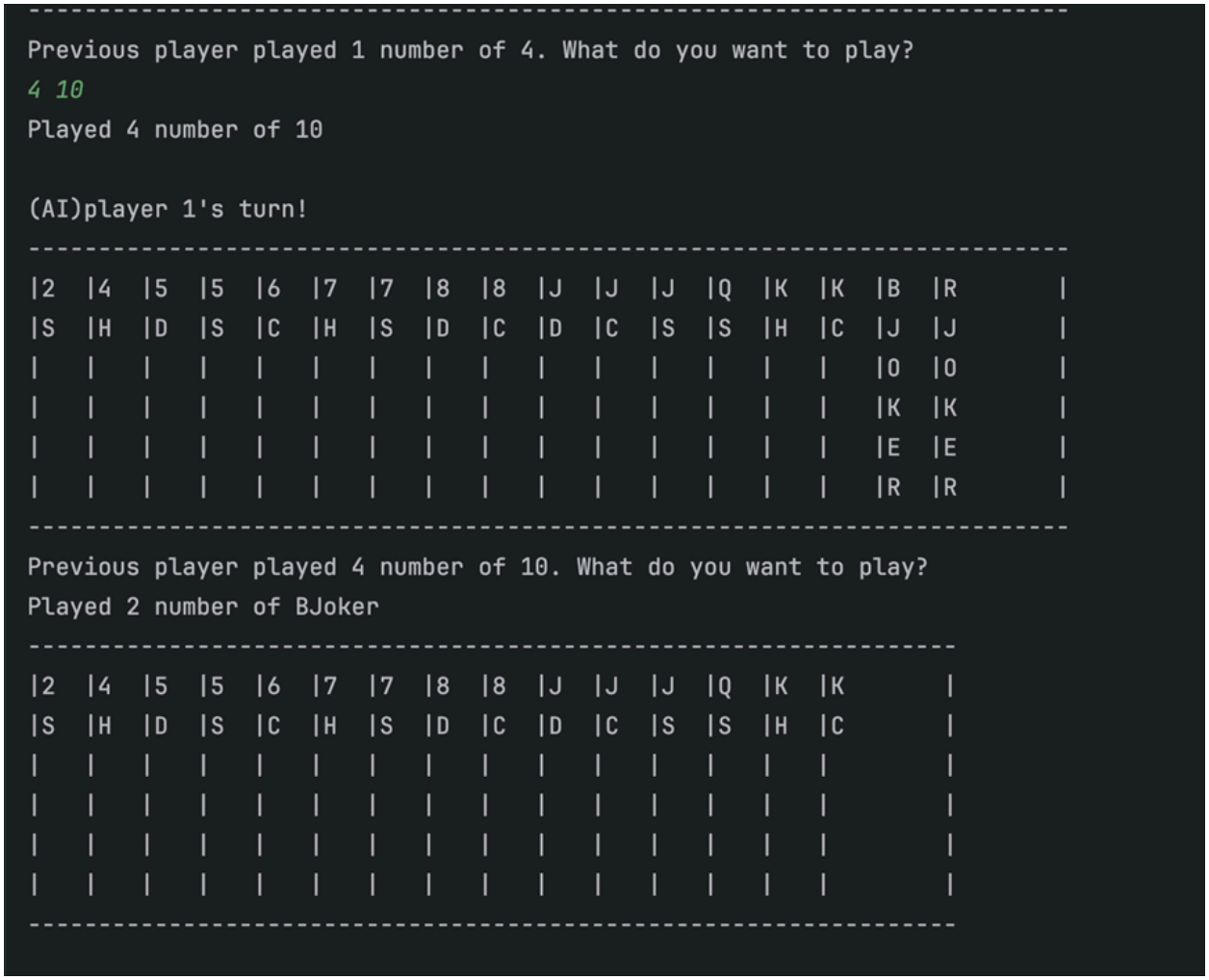
Algorithm:

Check for playing double jokers:

If both jokers are valid:
Set input to {2, 13}
Call MakePlay with input
Decrease numOfCards by 2
Announce the played move
Clean up and return input

Rational:

Because both other cases are run before this case, so it has been made sure that there is no normal set of cards playable by the AI. Then it checks the last resort, which is the highest play possible in the game, the Double Joker.



As can be seen, the previous player discards quadruple 10, and the AI, not having any quadruple cards, has no choice but to use the largest combination in the game, double Joker.

4. AI does not have any card that can surpass the previous played

Given: The previous player plays triple Q

Expected results: AI cannot play anything

Works because:

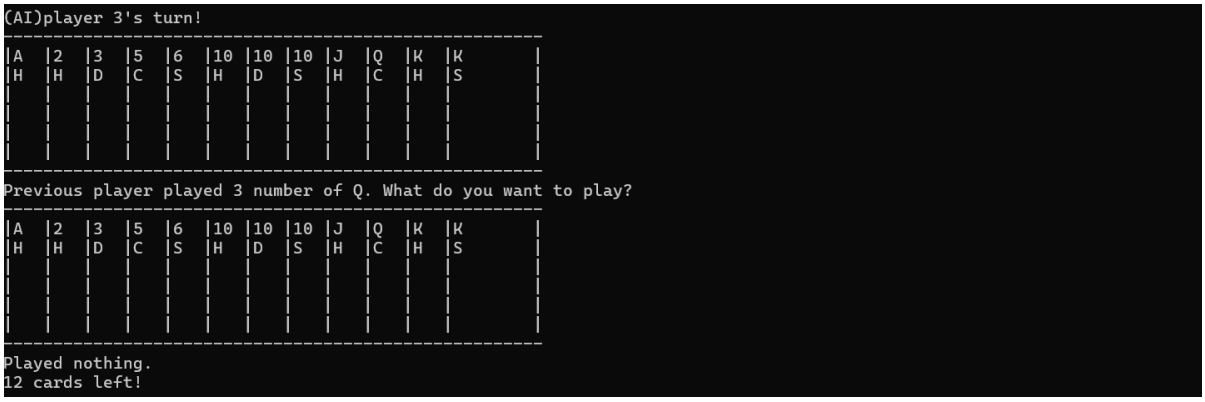
Algorithm:

If no valid move is found:

Announce "Played nothing."
Return null

Rational:

Because this is run after all the other cases have been failed, meaning that the AI has no capability to play anything that surpasses the previous play. Then the AI will pass.



As can be seen, the previous player played triple Q, AI has no card that's bigger than the previous cards, thus passing

5. AI has a free move

Given: The AI has a free move

Expected results: AI plays single 4, which is the smallest card the AI can play

Works because:

Algorithm:

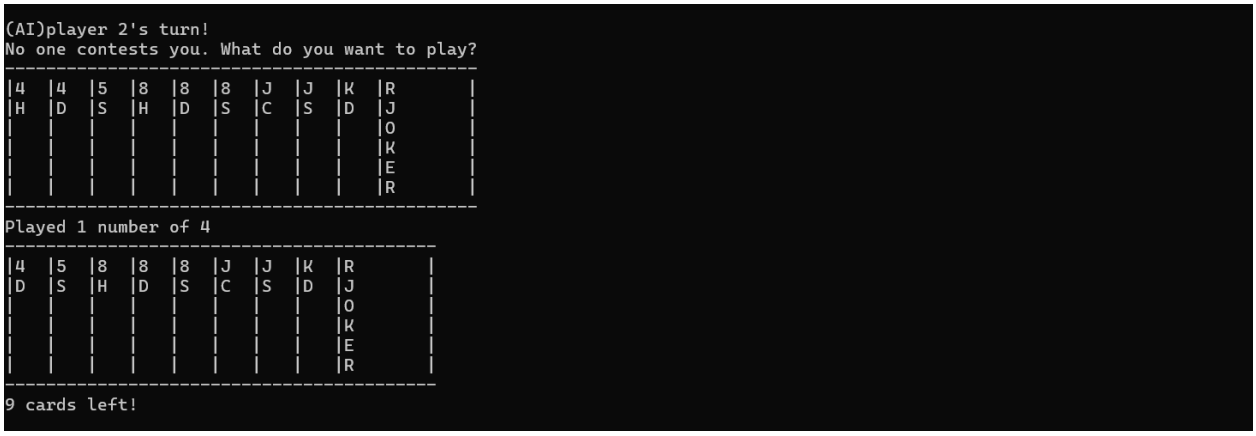
Sort the cards

Create a new integer array named input with 2 elements
Set input[0] to 1 (indicating playing one card)
Set input[1] to the value of the first card in the sorted list
Call function MakePlay with input
Decrease numOfCards by 1
Announce "Played [number of cards] number of [card value]"

Return the input array

Rational:

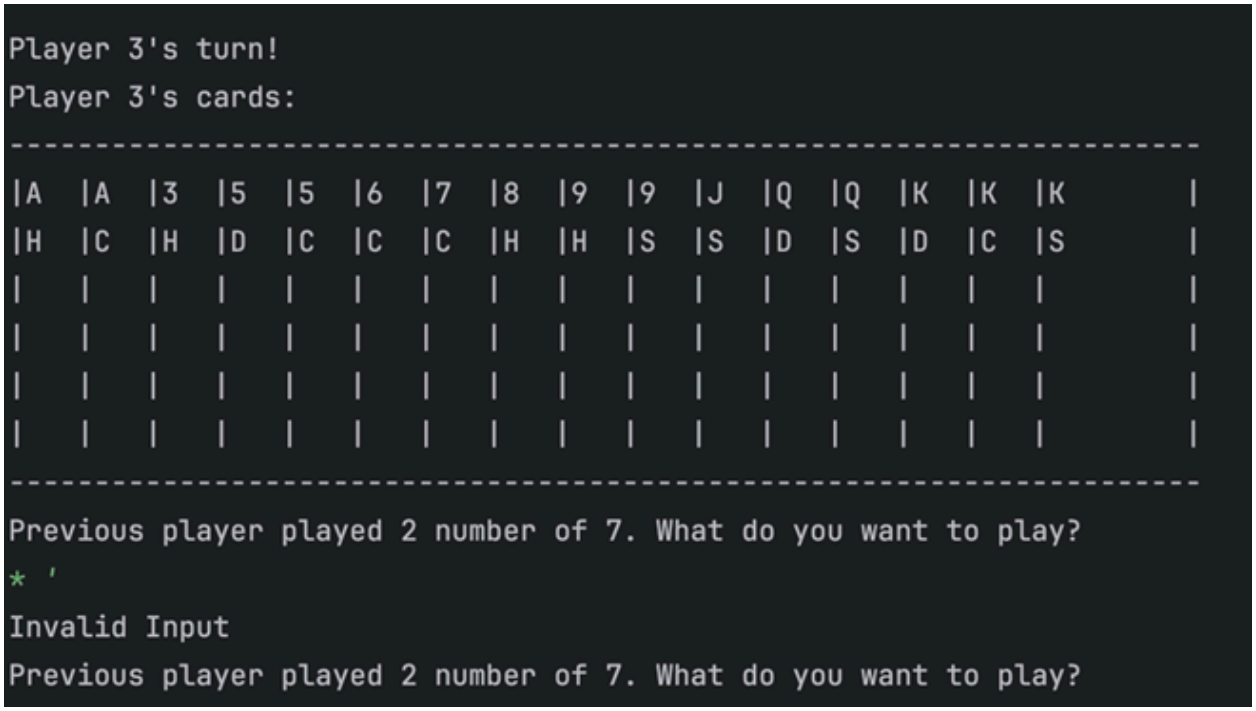
Because this first sorts the cards, then gets the first element of the vector to be played, it is sure that it plays the smallest card available to the AI.



As can be seen, when AI has a free move, it discards a single 4, which is the smallest card available to the AI.

Thirdly, the function of the player is tested by the following cases.

1. The user gets to discard cards. The acceptable form of input is “x y”, where x is any number between 1 to 4, as to indicate how many number of y the user wants to discard; and y is any number from 1-10 or ‘J’, ‘Q’, ‘K’, ‘BJOKER’ for black joker, ‘RJOKER’ for red Joker (e.g. “1 4” for single 4, ‘2 S’ for double joker, “3 J” for triple J, etc). Any input beyond this combination will be treated as invalid, thus returning an error message and asking the user to input again.



As can be seen, * ‘ is inputted, making no sense, and is required to input again.

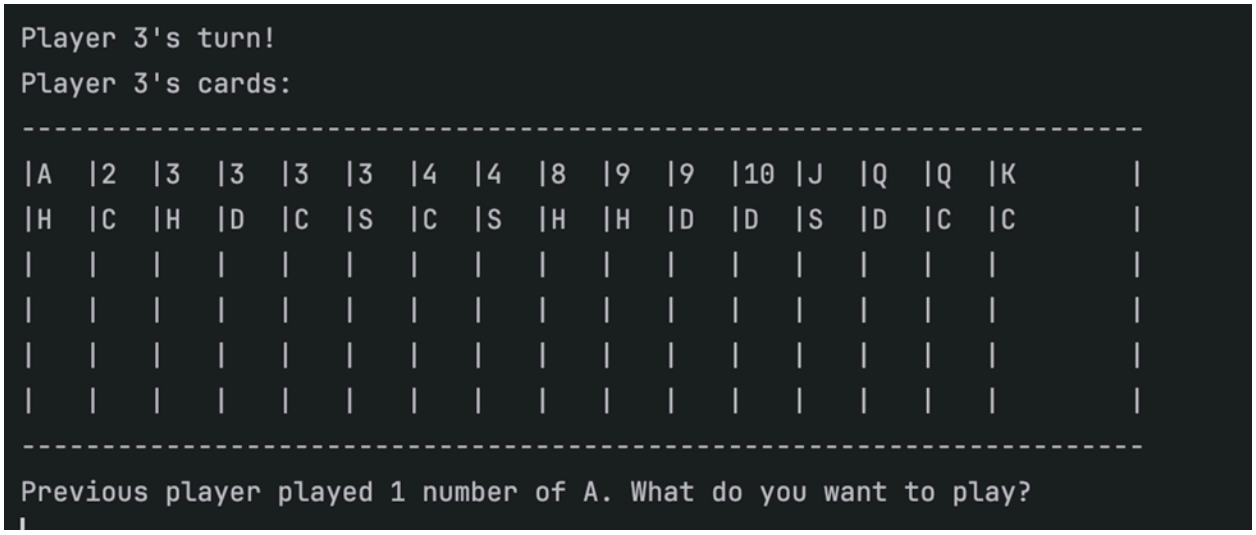
2. The user input is valid itself, but cannot be made due to not having the card or smaller than the previous one (e.g. “2 4” while not having double 4, “1 5” while the previously discarded card was a 9, etc.), an error message will be returned, asking the user to input again.

```
Player 1's turn!
Player 1's cards:
-------------------------------------------------------------------------
|A  |2  |3  |3  |6  |7  |7  |8  |8  |8  |9  |9  |J  |Q  |K  |R        |
|D  |S  |H  |C  |H  |D  |C  |H  |D  |S  |H  |C  |H  |S  |C  |J        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |O        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |K        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |E        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |R        |
-------------------------------------------------------------------------
Previous player played 2 number of 6. What do you want to play?
2 3
Card Input Invalid
Previous player played 2 number of 6. What do you want to play?
2 J
Card Input Invalid
Previous player played 2 number of 6. What do you want to play?
```

As can be seen, the user wants to input double 3 in front of a double 6, which is invalid because it’s smaller in value; also, the user wants to use double J, which is also invalid because the user doesn’t have double J at hand.

3. The player can pass by entering -1 -1.

```
Player 3's turn!
Player 3's cards:
-------------------------------------------------------------------------
|A  |2  |3  |3  |3  |3  |4  |4  |8  |9  |9  |10 |J  |Q  |Q  |K        |
|H  |C  |H  |D  |C  |S  |C  |S  |H  |H  |D  |D  |S  |D  |C  |C        |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |         |
-------------------------------------------------------------------------
Previous player played 3 number of 7. What do you want to play?
-1 -1
```



As can be seen, the user has no card bigger than triple 7, so the user chooses to pass by pressing -1, -1. Then, computer gets a free move and discards the smallest possible card, single Aces, while the user's card remains unchanged (showing that indeed it gets passed)

Lastly, the program should successfully identify the condition of winning.

As can be seen, after player 1 discards single J, which is the last card, player 1 wins the game.

Some challenges that we faced in the design of our program include the designing of the AI. Since the AI is always choosing the card that's "just" bigger than the previous, and in normal game playing, the user does not get to see the AI's card (otherwise this game would be meaningless). Therefore, in our testing, we have to develop a separate function that makes AI's card visible. After testing for many times, we are confident that our AI has our expected property, then we commented this function from our program.

A debugging problem that we had during the design of the program is that, since we have used many dynamically allocated arrays, we accidentally deleted one array while we still needed to use it. It took us a long time to debug this but we eventually found and fixed it.

4. Conclusion

The project successfully constructed all functions needed for our game. Our final result closely aligns with our proposal, achieving all results that we want to achieve. The program successfully asks the user for execution of the game, giving help, and termination of the game. Also, it creates options for the user to choose whether he/she wants to play with AI only, play with friends, or, just to watch three AI playing against each other. Moreover, for every game, the 18 cards displayed are different, leaving the player with a variety of choices and tactics. For the design of the AI, our program successfully achieves the result of having it discard cards that's "just" bigger by sorting algorithm, leaving playability of the game.

For future development, we can make the AI much smarter. For example, if a card single 4 is discarded and the AI has double 5 and single 6, then the AI will manually split the double 5 and discard a single 5, because a single 5 is considered the card "just" bigger than single 4. If it were a smarter AI, it would keep this double 5 and discard a single 6 instead. Also, we could incorporate more interesting rules like having straights and suits taken into consideration. Moreover, the game is largely based on terminal and command line tools. Therefore, if a UI is added, the game will be more interactive. Also, the search algorithm in the game is linear search, binary search might be used to improve the efficiency of the program.

5. Lessons learned

One lesson we have learned is that when the program includes randomness, we should use dependency injection to inject the random elements because the parts that use the random elements don't care where the data comes from. It only cares about whether the data could be accepted or not. Therefore, if we use dependency injection, we can decouple the deck assignment to the participants, making the code clearer. It is also good for testing. Using dependency injection, when testing, makes it easy to disconnect the random elements but use fixed sets of data to test more rigorously and conveniently. This way also allows the use of automated code to test all possible combinations if the code scope is manageable.

Another thing that we have learned is that the encoding for different environments is very different. Therefore we need to have some understanding of the encoding methods of different environments in order to achieve what we want to display in all environments. If we had more knowledge in encoding methods in UNIX environments, we could have used UNICODE to encode the suits instead of using letters.

6. Reference

We only used documentation for the standard library in C++.
https://en.cppreference.com/w/cpp/standard_library