SE 116 PROJECT REPORT

1. Introduction :

1.1 Purpose and Objectives of the Project:

The purpose of the lecture “Introduction to Programming II” Project is to design an advanced “Pisti” game named “Misti” and provide players with a better experience. To achieve this goal, we will add many features in addition to the standard rules of the “Misti” game. These may include new card points and difficulty levels. Additionally, we may add different game modes to enhance players' game experiences.

2. Design

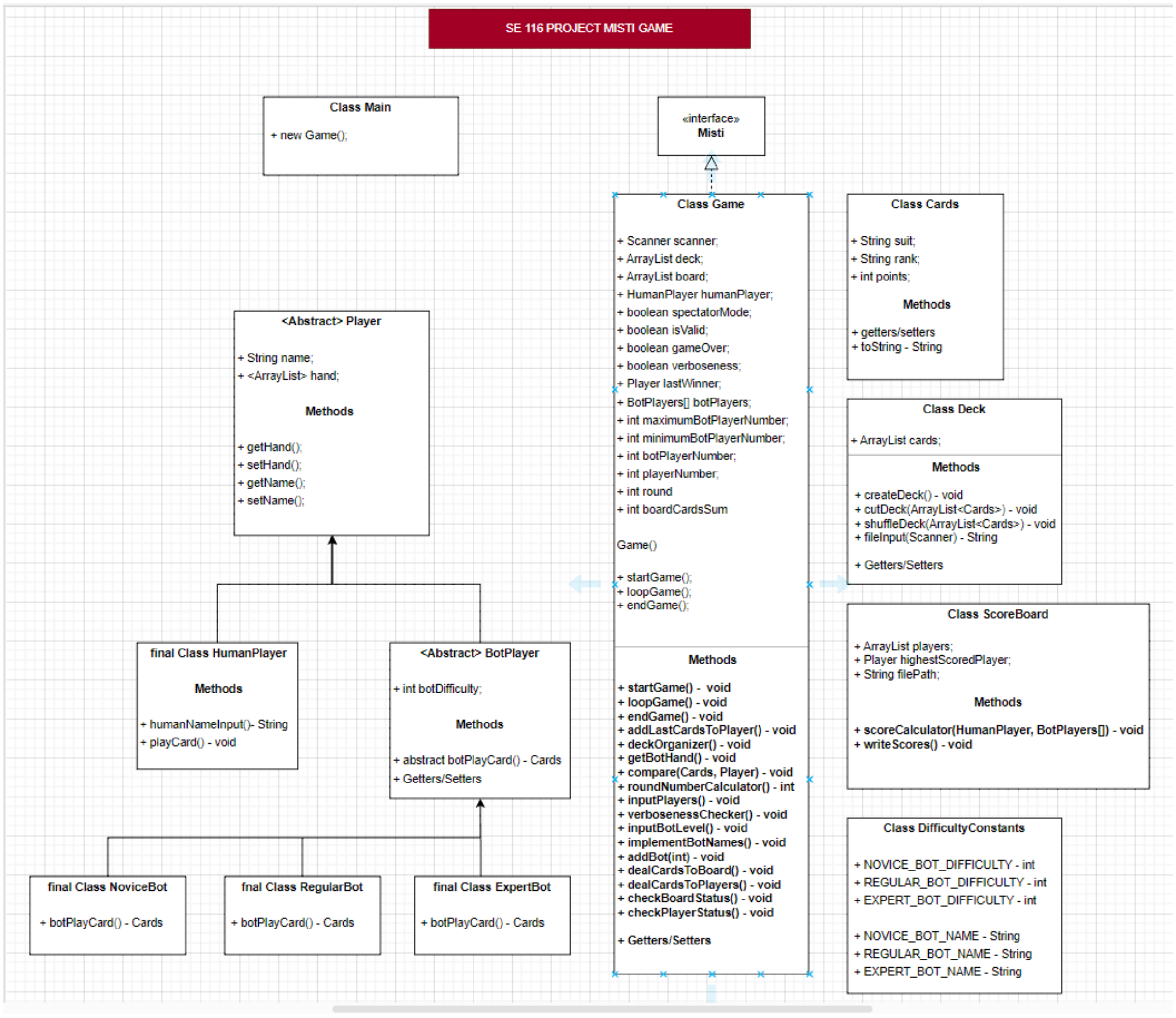
2.1 Programming language and tools used:

This Project was developed with Java programming language and via IDE Eclipse.

2.2 Data Structures, algorithms, and design patterns:

This Project was developed using subjects OOP Design Principles (Abstraction, Encapsulation, Modularity), Classes, Java Collections Framework, Extending Classes (Inheritance), Polymorphism, Interfaces and Abstract Classes, Exceptions, Buffers, Text Processing

2.3 UML Diagram



3. Implementation :

3.1 Detailed explanation of each functional requirement:

Functional requirements of the Project:

1. The program must be able to create a deck of cards.

2. The program must be able to shuffle and cut the deck.

3. The program must be able to read game parameters from the command line and display appropriate error messages for invalid input. Parameters include the number of players, point file name, name and expertise level of each player, and verboseness level.

4. The program must be able to move cards from the deck to the players and the boards.

5. The program must print to console a log of hands dealt to each player, and all the moves for each round and each hand in verbose mode, and just the score for each round in succinct mode.

6. The program must be able to calculate each player's score.

7. The program must be able to store a "high score list" on a file that stores the top 10 scores and the names, and expertise levels of the players who scored them.

8. The program must include a Novice player. This player should obey all the rules of the game, but have a very rudimentary strategy, and just randomly choose one of the available cards to play at their turns.

9. The program must include a Regular player. This player should obey all the rules of the game but also choose a good card to play, based on which card is on the board, and what cards are in the hand, taking into consideration the point value of each card. For example, consider when the board has a 3 on the top, and the next player also has a 3: In this situation, the next player must play the 3 if the total value of cards to be taken is positive, otherwise, a different card must be chosen.

10. The program must include an Expert player. This player should obey all the rules of the game but also choose a good card to play, based on what cards are on the board, what cards are in their hand, and tracking what cards have already been played, taking into consideration the point value of each card.

Functional Requirement 1:

The Cards class represents a single card with its suit, rank, and point value. The Deck class represents a deck of cards. It creates a new deck of cards with the standard 52 cards using a nested loop that iterates through each suit and rank combination.

When creating the deck of cards, each card is assigned a default point value of Integer.MAX_VALUE, which represents an unassigned value. This is because the point values for the cards are later read from a file and updated accordingly.

If a card's point value is still set to Integer.MAX_VALUE after reading the file, it means that the card was not included in the file and thus does not have a specific point value assigned to it. In this case, the default point value is used, which is initially set to 1.

Therefore, the default point value serves as a fallback option for cards that are not explicitly assigned a point value in the file. This ensures that every card in the deck has a point value assigned to it, even if it is a default value.

The class also prompts the user to input a file name that contains the point values of the cards. The process of reading the file line by line involves iterating through the contents of the file and examining each line to determine if it contains the default point value. This is done by checking if the line starts with "**". If this condition is met, the substring method is used to extract the value from the line after the second character. The trim method is then applied to remove any unnecessary whitespace characters from the extracted value. Otherwise, each line contains the point value for a specific suit or rank, and the corresponding card in the deck is updated with this value.

Functional Requirement 2:

This requirement is handled by two methods, cutDeck, and shuffleDeck, which operate on an ArrayList of Cards. The cutDeck method randomly selects a cut index within the range of the ArrayList and creates two new ArrayLists, bottom and top, that respectively contain the bottom and top halves of the original ArrayList. This technique is handled by using an ArrayList Class method, sublist. The sublist method provides us with to convenience of extracting the list from the start to the cut index which represents the top half and the cut index to the end which represents the bottom half just like the substring method of String Class. The original ArrayList is then cleared and re-populated with the top half followed by the bottom half, effectively cutting and re-ordering the deck.

The shuffleDeck method shuffles the ArrayList in random order. This is done by calling the static shuffle method of the Collections class, which shuffles the ArrayList using the default random number generator.

Functional Requirement 3:

The program is designed to read game parameters from the command line, and it provides appropriate error messages for invalid input using exception handling. Upon starting the program, the user is prompted to enter the name of the point file. Subsequently, the user is given the option to play or spectate the game, or to exit the program. If the user chooses to play, they are asked to input their name, which is then returned in upper case using the StringBuilder class. The user is then welcomed into the game. The program then prompts the user to select the number of bot players in the game, as well as their respective difficulty levels. Each input is handled with try-catch blocks to handle exceptions that may occur if the user inputs an unexpected value.

If the user inputs a string instead of an expected integer value or an integer value that is not within the expected range, the program uses a switch case and the String class method "matches" to handle the issue. The scope of valid inputs is determined by whether the user is in the game or spectating. For example, if the user is in the game, the scope is limited to integers between 1 and 3, while if the user is spectating, the scope is between 2 and 4. Therefore the maximum number of players allowed in the game is four.

Finally, the program prompts the user to select whether they would like to turn on or off the verboseness mode. Verboseness mode allows the user to see the opponent's cards and the point values of cards that would normally be hidden. If an exception occurs during any of the input prompts, the program enters a loop, prompting the user to enter a valid input until a valid value is entered.

Functional Requirement 4:

The StartGame method uses the dealsCardsToBoard method to deal four cards to the board and start the game. In the LoopGame method, each player is dealt four cards until the end of the rounds, which is calculated differently based on the number of players. If there are four players, there will be three rounds, if there are three players, the number of rounds is four and if there are only two players, there will be six rounds. In each round, the players take turns playing a card.

Functional Requirement 5:

The program utilizes three methods, namely checkBoardStatus, checkPlayerStatus, and getBotHand, to fulfill the stipulated requirements. These methods are responsible for printing out the cards on the board, in the player's hand, and the bot's hand, respectively. In addition, the program incorporates a feature that allows for the handling of verboseness mode, which is achieved by initializing a boolean value. If verboseness mode is enabled, the program prints out the cards in the opponent's hand and their corresponding point values. On the other hand, if verboseness mode is disabled, the program operates in a succinct mode, wherein no additional information is printed beyond the essential card details. These functions collectively serve to enhance the user experience by providing relevant information while also accommodating different user preferences.

Functional Requirement 6:

This requirement is handled with the score calculator method, under this Class players ArrayList is initialized to gather all players in one list. Subsequently, by iterating each player’s collected cards players’ scores are set with the sum of their previous score and the value of the card that was collected. After that players’ misti scores were updated with the five times the misti score was calculated right after the Misti situation under compare method. Misti scores are added to scores after. Finally with iterating all player’s cards the player who earns the most point is assigned to the highest score player.

Functional Requirement 7:

The method then finds the player with the highest score and stores it in a variable called highestScorePlayer. The players’ ArrayList is cleared and the highestScorePlayer is added to it. The method then reads a file at a given filePath location. The file contains the names and scores of the top 10 players in the game. The scores of the players in the players ArrayList are compared to the scores of the players in the file. If the highestScoredPlayer has a higher score than the player in the 10th position in the file, the highestScoredPlayer is added to the file, and the player in the 10th position is removed. The player’s ArrayList is sorted in descending order of scores and then written to the file, replacing the previous contents of the file. The method displays the names and scores of the players that were written to the file.

Functional Requirement 8:

The requirement is handled with the method named "botPlayCard" which is responsible for simulating the play of a card by a Novice Bot in a card game. The bot follows all the rules of the game but lacks any sophisticated strategy. Instead, it simply selects a random card from its hand and plays it, without taking into consideration the value or rank of the cards already on the board. As a result, if the Novice Bot player manages to achieve a high score, it can be attributed to luck rather than any skillful playing. The implementation of this Novice Bot can serve as a useful baseline for future enhancements to the bot's strategy and decision-making process, as well as for comparisons with other more advanced bots in the game.

Functional Requirement 9:

The requirement is handled with the method named “botPlayCard” which is responsible for simulating the play of a card by a Regular Bot in a card game. The method returns the card that the bot has played. The method starts by initializing some boolean flags and an integer variable to keep track of the maximum potential point value of the cards on the board. Then, it checks if the game board is not empty. If it is not empty, the bot iterates over all the cards in its hand and checks if any of them has the same rank as the last card played on the board. If there is such a card, the bot checks if the sum of the point values of that card and the cards on the board is greater than the current maximum potential point value. If it is, the bot selects that card as the one to play and updates the appropriate boolean flags and the maximum potential point value. If the size of the board is one, it also sets the "canMakePisti" flag to true. If there is no card in the bot's hand that has the same rank as the last card played on the board, the bot checks if it has a Jack card, and the "canMakePisti" flag is not already set to true. If it has a Jack card and the sum of the point values of that card and the cards on the board is greater than the current maximum potential point value, the bot selects that card as the one to play and updates the appropriate boolean flags and the maximum potential point value. If neither of the above conditions is met, the bot selects the card with the lowest point value that does not have the same rank as the last card played on the board and is not a Jack card. If there is only one card in the bot's hand, it selects that card. Finally, the bot calls the "handOrganizer" method to remove the played card from its hand and returns the played card.

Functional Requirement 10:

The requirement is handled with the method named “botPlayCard” which is responsible for simulating the play of a card by an Expert Bot in a card game. It initializes several variables that will be used later: canTakeBoard, boardCardsMaxPotentialPoint, canMakePisti, and playedCard.

It creates an ArrayList called mostPlayedCards and an array mostThrownCardsNumber of the same size as the bot player's hand. The array will store how many times each card has been thrown by all players (bots and human) in the game.

It then loops through the cards in the bot player's hand, and for each card, it loops through all bot players and the human player to see if they have collected any cards with the same rank. If they have, it increments the mostThrownCardsNumber for that card by 1. Checks if any cards have been played on the board. Using this way, Expert Bot keeps track of the cards that have been played in-game without having the issue of duplicate cards. It checks if the card in the bot player's hand has the same rank as any of the cards that have been played on the board. If it does, it increments the mostThrownCardsNumber for that card by 1 taking into consideration the index of the card in hand. Prints out the card and the number of times it has been thrown (if verbosity is enabled). The method then loops through mostThrownCardsNumber to find any cards that have been thrown 3 times. If there are any, it adds them to mostPlayedCards.

If there are cards on the board, the method loops through the bot player's hand to find any card that has the same rank as the last card played on the board. If it finds any, it checks if playing that card will give the bot player the maximum potential point on the board. In this way, the program checks whether the sum of the cards on the board and the card that will be played are greater than zero or not. Expert Bot also checks all cards playable and selects the card which has the highest point value. It sets then played card to that card and sets canTakeBoard to true. If the board has only one card and the bot player plays a card with the same rank card as the top card, canMakePisti is set to true to prevent the bot from playing jack and missing the making misti chance. If there are no cards on the board, the method checks if there are any cards that have been thrown 3 times or more. This way bot prevents other players to make misti. If there are cards that have been thrown 3 times, it sets mostPlayedCards to the cards with the highest number of throws. If there is more than one card in mostPlayedCards, it sets the played card to the lowest point card in the list. If playedCard is not null and is not a jack, the method organizes the bot player's hand and returns playedCard. If canTakeBoard is true, the method organizes the bot player's hand and returns played card. If none of the above conditions are met, the method finds the card with the lowest point in its hand (excluding jacks). If there is only one card in the hand, it sets played card to that card. If there is more than one card in the hand, it sets played card to the card that both has been thrown the most times by all players and the lowest point card. The method then organizes the bot player's hand and returns the played card.

4. Challenges and problems encountered. Solutions to the challenges and changes made during this process

- Misunderstanding of the implementation of the cards file and making it all cards instead of a points file.
- A better understanding of what file we are excepted to create by examining the description well.

- A challenge to give their hands of cards and collected cards lists to each player will add accordingly to the number of players.
- Instead of initializing lists with references above, create the list objects when we create players and assign hands to them. We add one hand list to the Player class and solve this issue.

- Assign all bot players' names to their bot level regardless of how many bot players are in the game and create confusion about same-level bot players' names.
- To solve this problem, we create a method that sets unique names for bots based on their difficulty levels. It initializes counters for Novice, Regular, and Expert Bots, then loops over an array of BotPlayers objects and sets each bot's name based on its difficulty level and the appropriate counter value. (Example: REGULAR BOT 1, REGULAR BOT 2, REGULAR BOT 3 etc..)

- Creating two overloaded compare methods because the player's reference is not Player yet it's subclasses as bot player or human player.
- Player reference is right to use as a parameter. Therefore one compare method works correctly.

- In case of Regular or Expert Bot players has more than one playable card (it is obligatory to board and played card points are positive) they should not play the lowest point one between them.
- The variable "boardCardsMaxPotentialPoint" keeps track of the highest potential point value a player can score by playing a card from their hand on the board. It is used in a for loop to compare the sum of the current card's point value and the total point value of the cards on the board with the current value of "boardCardsMaxPotentialPoint". If the sum is greater, "boardCardsMaxPotentialPoint" is updated.

- Game-play of Regular or Expert Bots initially separates the situations of the same rank card or jack card. But this led to confusion and a mess of codes.
- The Bot plays the card with the highest point value if it is possible to take cards from the board according to the Misti game rules regardless of whether the card is a jack or the same rank card. Because in this advanced misti game board total points might decrease and keeping Jack to play later is not indeed sensible. Therefore Regular and Expert Bots should collect the cards with the highest point card.

- There was a chance for Regular or Expert bot plays Jack when they have matched card because the sum of cards will be collected are higher points even though there was an opportunity to make misti.
- To solve this issue we add one if statement to check whether the board's length is 1 or not and set canMakePisti to true if it is. After the code does not step into the if statement where the Jack situation checks.

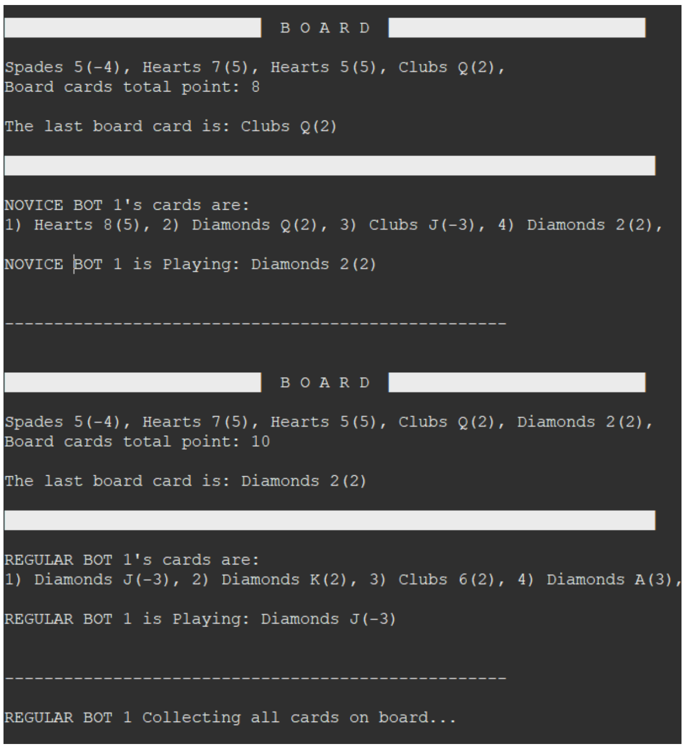
- When we improved the code, Bot Players cannot play to collect cards and not play the matched or Jack card accidentally because they are the lowest point. This caused to null pointer exception.
- Add the if statement and the one which ensures played card does not match with the rank also make sure that the board is not empty and the other if statement only checks the situation of the played card being a Jack accidentally.

- Expert Bot might have played the worse card because they have more than one same number from mostThrownCards Array and they should have to play the lowest point between them.
- A method that solves this problem takes the "mostPlayedCards" ArrayList and returns the card with the lowest point value. It initializes two variables, "maxCardPoint" and "lowestMostPlayedCard", loops through the ArrayList, and updates "lowestMostPlayedCard" to the current card if its point value is less than the current value of "maxCardPoint".

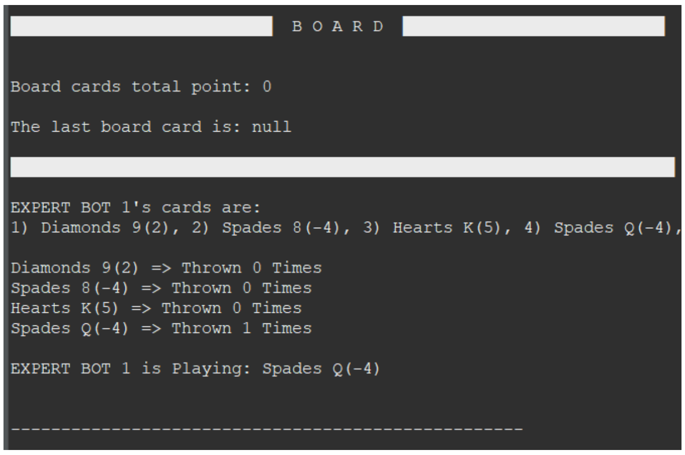
- The Expert Bot might have played the worse card because they have more than one lowest point card with the same point and should play the most played card between them.
- A method that solves this problem selects the card from a player's hand that has the same point value as a given played card, has been thrown the most by other players, is not a jack, and does not have the same rank as the top card on the game board. It returns the selected card.

- First, the played cards are added to a thrown cards list and the Expert Bot kept track of played cards by checking that list, which caused the copying card issue.
- To not copy of cards, instead of creating a thrown cards list Expert Bot keeps track of the collected cards of each player and cards on the board. This way Expert Bot checks the thrown cards as well.

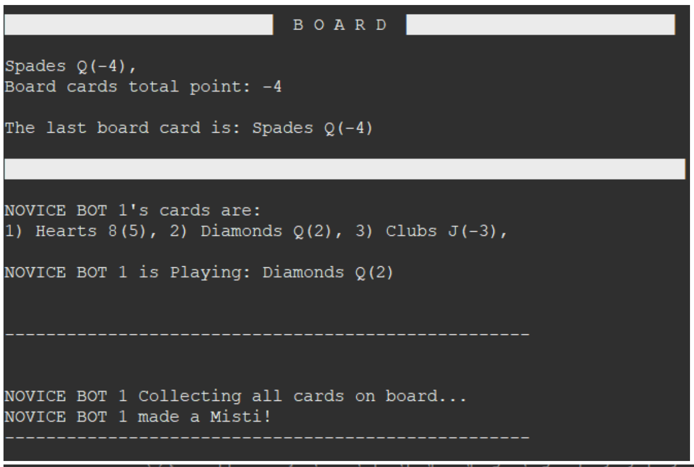
5.1 One-round sample run



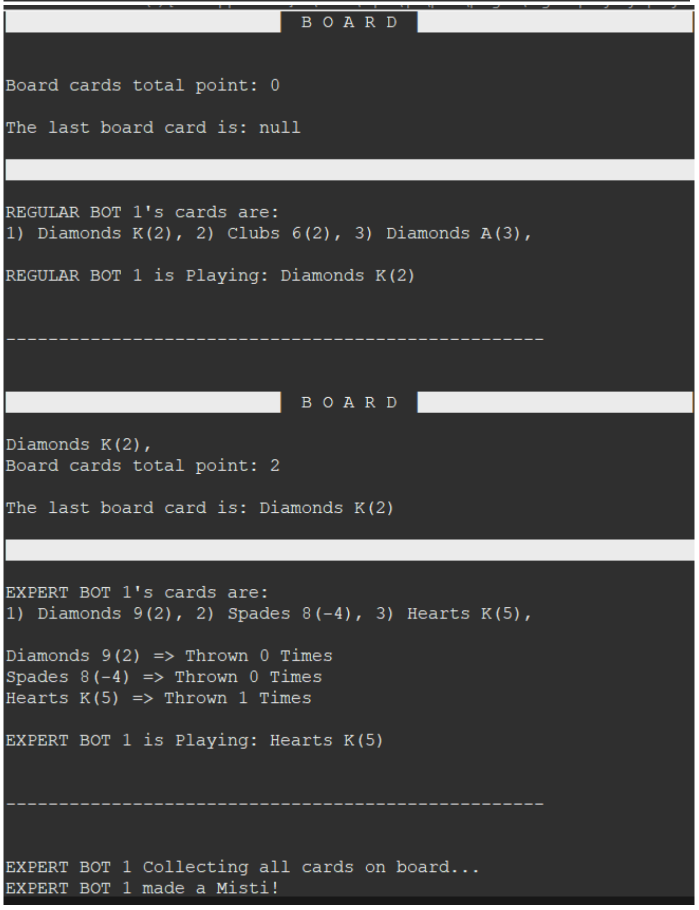
```
                 B O A R D

Spades 5(-4), Hearts 7(5), Hearts 5(5), Clubs Q(2),
Board cards total point: 8

The last board card is: Clubs Q(2)


NOVICE BOT 1's cards are:
1) Hearts 8(5), 2) Diamonds Q(2), 3) Clubs J(-3), 4) Diamonds 2(2),

NOVICE BOT 1 is Playing: Diamonds 2(2)


---------------------------------------------------


                 B O A R D

Spades 5(-4), Hearts 7(5), Hearts 5(5), Clubs Q(2), Diamonds 2(2),
Board cards total point: 10

The last board card is: Diamonds 2(2)


REGULAR BOT 1's cards are:
1) Diamonds J(-3), 2) Diamonds K(2), 3) Clubs 6(2), 4) Diamonds A(3),

REGULAR BOT 1 is Playing: Diamonds J(-3)


---------------------------------------------------

REGULAR BOT 1 Collecting all cards on board...
```



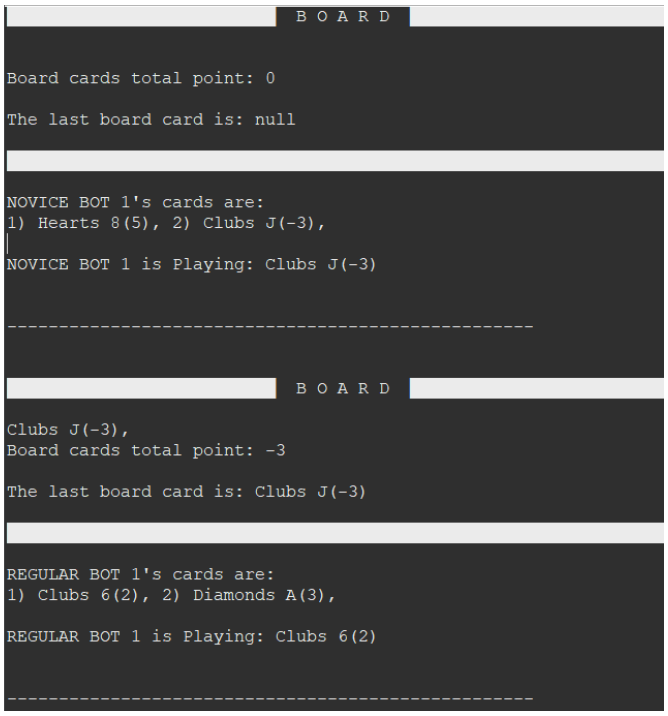
```
                 B O A R D


Board cards total point: 0

The last board card is: null


EXPERT BOT 1's cards are:
1) Diamonds 9(2), 2) Spades 8(-4), 3) Hearts K(5), 4) Spades Q(-4),

Diamonds 9(2) => Thrown 0 Times
Spades 8(-4) => Thrown 0 Times
Hearts K(5) => Thrown 0 Times
Spades Q(-4) => Thrown 1 Times

EXPERT BOT 1 is Playing: Spades Q(-4)


---------------------------------------------------
```



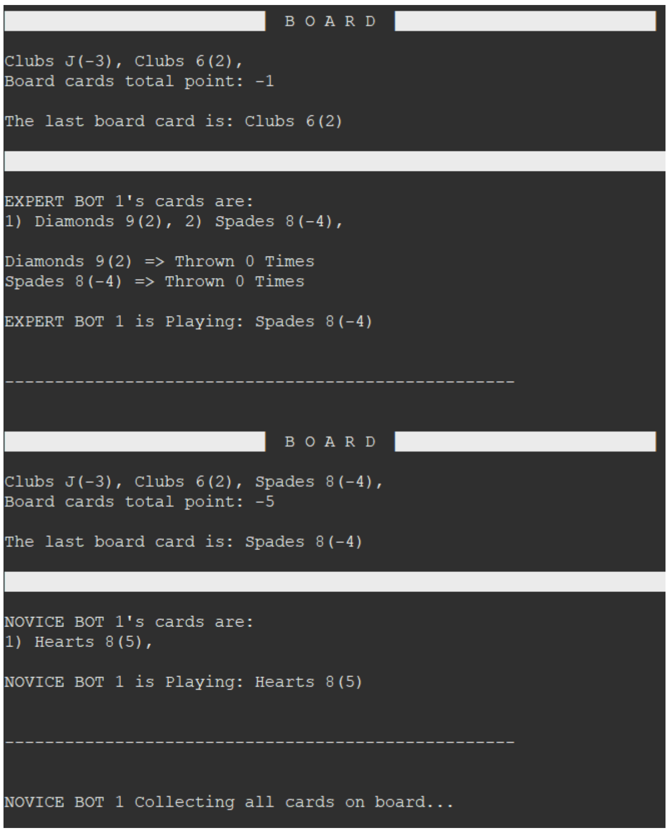
```
              B O A R D

Spades Q(-4),
Board cards total point: -4

The last board card is: Spades Q(-4)


NOVICE BOT 1's cards are:
1) Hearts 8(5), 2) Diamonds Q(2), 3) Clubs J(-3),

NOVICE BOT 1 is Playing: Diamonds Q(2)


---------------------------------------------------


NOVICE BOT 1 Collecting all cards on board...
NOVICE BOT 1 made a Misti!
---------------------------------------------------
```



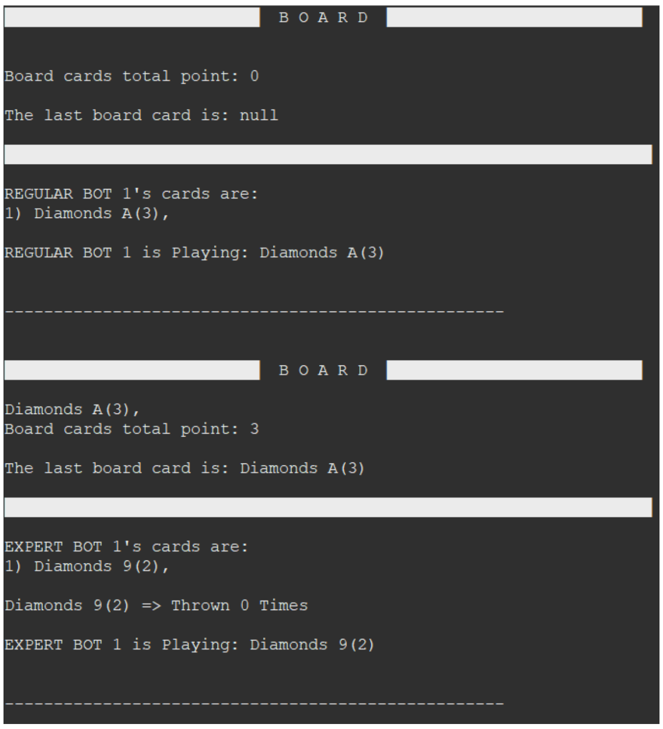
```
              B O A R D


Board cards total point: 0

The last board card is: null


REGULAR BOT 1's cards are:
1) Diamonds K(2), 2) Clubs 6(2), 3) Diamonds A(3),

REGULAR BOT 1 is Playing: Diamonds K(2)


---------------------------------------------------


              B O A R D

Diamonds K(2),
Board cards total point: 2

The last board card is: Diamonds K(2)


EXPERT BOT 1's cards are:
1) Diamonds 9(2), 2) Spades 8(-4), 3) Hearts K(5),

Diamonds 9(2) => Thrown 0 Times
Spades 8(-4) => Thrown 0 Times
Hearts K(5) => Thrown 1 Times

EXPERT BOT 1 is Playing: Hearts K(5)


---------------------------------------------------


EXPERT BOT 1 Collecting all cards on board...
EXPERT BOT 1 made a Misti!
```



```
BOARD

Board cards total point: 0

The last board card is: null


NOVICE BOT 1's cards are:
1) Hearts 8(5), 2) Clubs J(-3),

NOVICE BOT 1 is Playing: Clubs J(-3)


---------------------------------------------------


BOARD

Clubs J(-3),
Board cards total point: -3

The last board card is: Clubs J(-3)


REGULAR BOT 1's cards are:
1) Clubs 6(2), 2) Diamonds A(3),

REGULAR BOT 1 is Playing: Clubs 6(2)


---------------------------------------------------
```

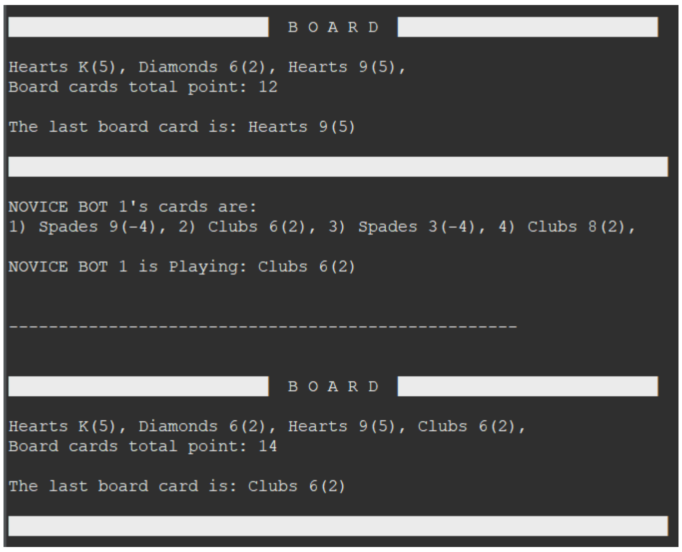


```
BOARD

Clubs J(-3), Clubs 6(2),
Board cards total point: -1

The last board card is: Clubs 6(2)


EXPERT BOT 1's cards are:
1) Diamonds 9(2), 2) Spades 8(-4),

Diamonds 9(2) => Thrown 0 Times
Spades 8(-4) => Thrown 0 Times

EXPERT BOT 1 is Playing: Spades 8(-4)


---------------------------------------------------


BOARD

Clubs J(-3), Clubs 6(2), Spades 8(-4),
Board cards total point: -5

The last board card is: Spades 8(-4)


NOVICE BOT 1's cards are:
1) Hearts 8(5),

NOVICE BOT 1 is Playing: Hearts 8(5)


---------------------------------------------------


NOVICE BOT 1 Collecting all cards on board...
```



```
                         B O A R D

Board cards total point: 0

The last board card is: null


REGULAR BOT 1's cards are:
1) Diamonds A(3),

REGULAR BOT 1 is Playing: Diamonds A(3)


---------------------------------------------------

                         B O A R D

Diamonds A(3),
Board cards total point: 3

The last board card is: Diamonds A(3)


EXPERT BOT 1's cards are:
1) Diamonds 9(2),

Diamonds 9(2) => Thrown 0 Times

EXPERT BOT 1 is Playing: Diamonds 9(2)


---------------------------------------------------
```

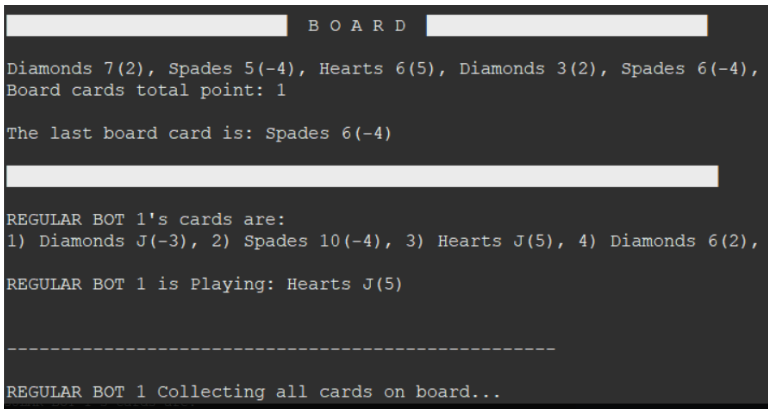
5.2 Novice Bot sample run



```
                         B O A R D

Hearts K(5), Diamonds 6(2), Hearts 9(5),
Board cards total point: 12

The last board card is: Hearts 9(5)


NOVICE BOT 1's cards are:
1) Spades 9(-4), 2) Clubs 6(2), 3) Spades 3(-4), 4) Clubs 8(2),

NOVICE BOT 1 is Playing: Clubs 6(2)


---------------------------------------------------

                         B O A R D

Hearts K(5), Diamonds 6(2), Hearts 9(5), Clubs 6(2),
Board cards total point: 14

The last board card is: Clubs 6(2)
```

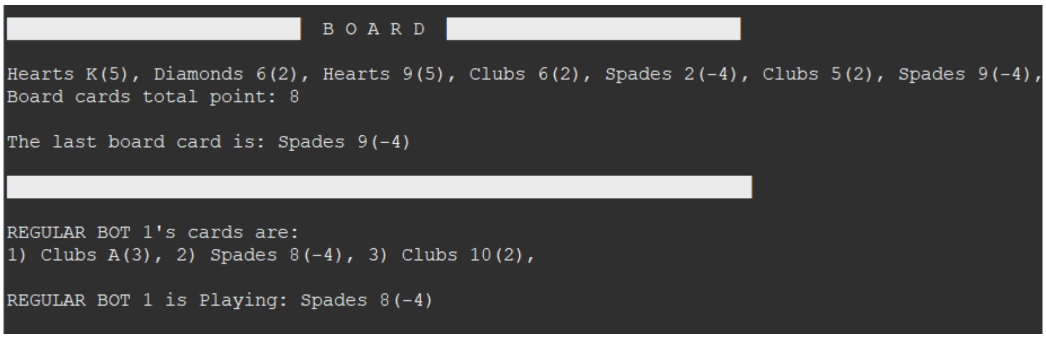
Plays randomly and cannot play the card that matches.

5.3 Regular Bot sample run



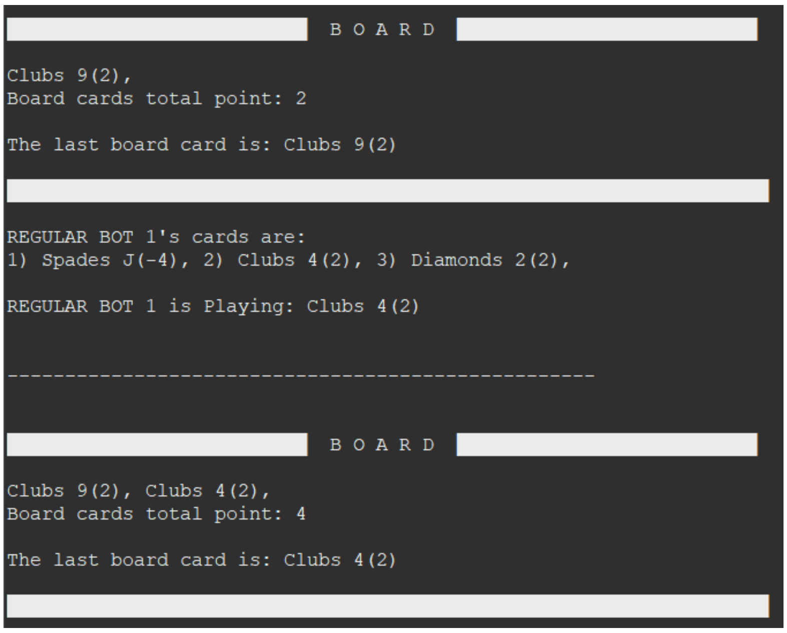
```
                            B O A R D

Diamonds 7(2), Spades 5(-4), Hearts 6(5), Diamonds 3(2), Spades 6(-4),
Board cards total point: 1

The last board card is: Spades 6(-4)


REGULAR BOT 1's cards are:
1) Diamonds J(-3), 2) Spades 10(-4), 3) Hearts J(5), 4) Diamonds 6(2),

REGULAR BOT 1 is Playing: Hearts J(5)


---------------------------------------------------

REGULAR BOT 1 Collecting all cards on board...
```

Board's and played card's total points are positive. Hence plays Jack and collects the board.



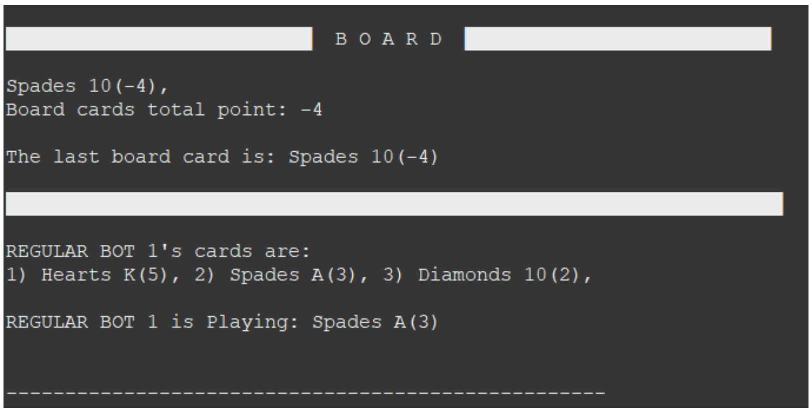
```
                            B O A R D

Hearts K(5), Diamonds 6(2), Hearts 9(5), Clubs 6(2), Spades 2(-4), Clubs 5(2), Spades 9(-4),
Board cards total point: 8

The last board card is: Spades 9(-4)


REGULAR BOT 1's cards are:
1) Clubs A(3), 2) Spades 8(-4), 3) Clubs 10(2),

REGULAR BOT 1 is Playing: Spades 8(-4)
```

Cannot collect the board. Therefore plays the lowest point card.



```
                              B O A R D

Clubs 9(2),
Board cards total point: 2

The last board card is: Clubs 9(2)


REGULAR BOT 1's cards are:
1) Spades J(-4), 2) Clubs 4(2), 3) Diamonds 2(2),

REGULAR BOT 1 is Playing: Clubs 4(2)


--------------------------------------------------


                              B O A R D

Clubs 9(2), Clubs 4(2),
Board cards total point: 4

The last board card is: Clubs 4(2)

```

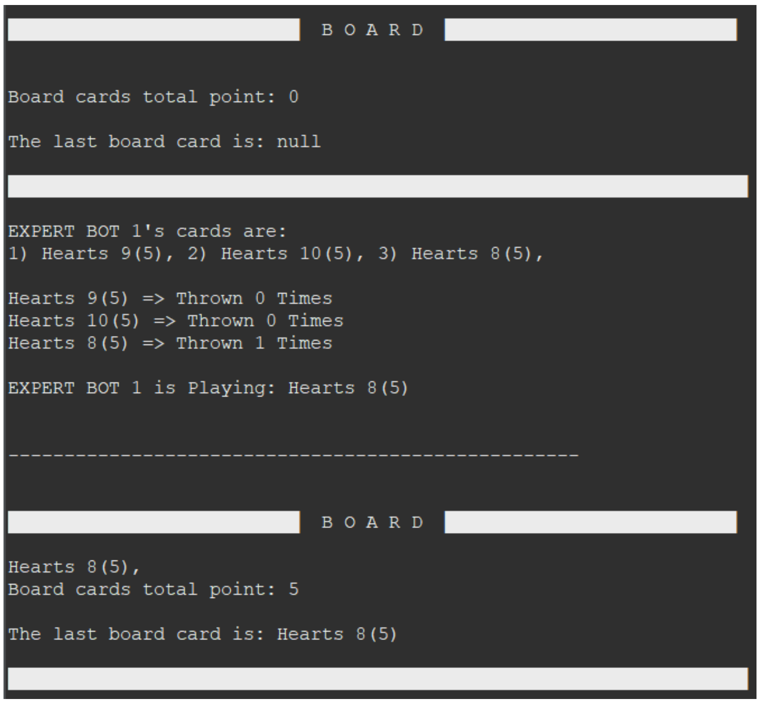
Can play the jack but does not because the board’s and Jack’s total points are negative.



```
                              B O A R D

Spades 10(-4),
Board cards total point: -4

The last board card is: Spades 10(-4)


REGULAR BOT 1's cards are:
1) Hearts K(5), 2) Spades A(3), 3) Diamonds 10(2),

REGULAR BOT 1 is Playing: Spades A(3)


---------------------------------------------------
```

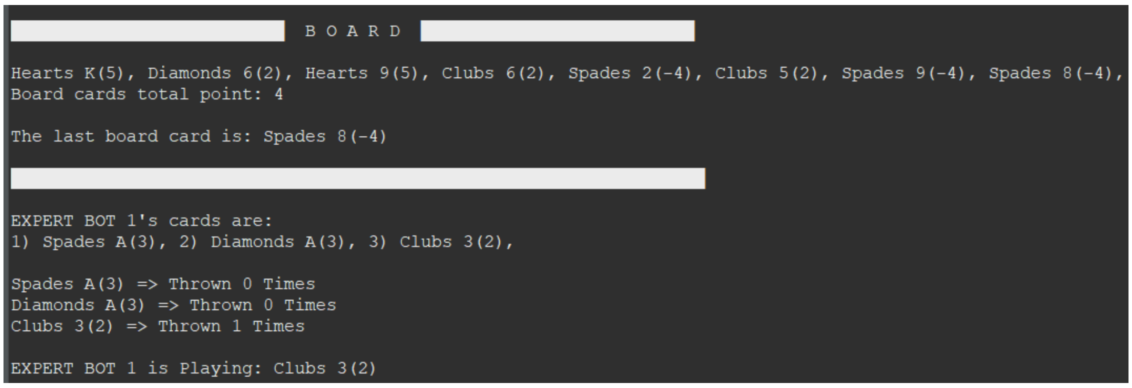
Does not play the matched card and make misti because there will be no positive points earned.

5.4 Expert Bot sample run



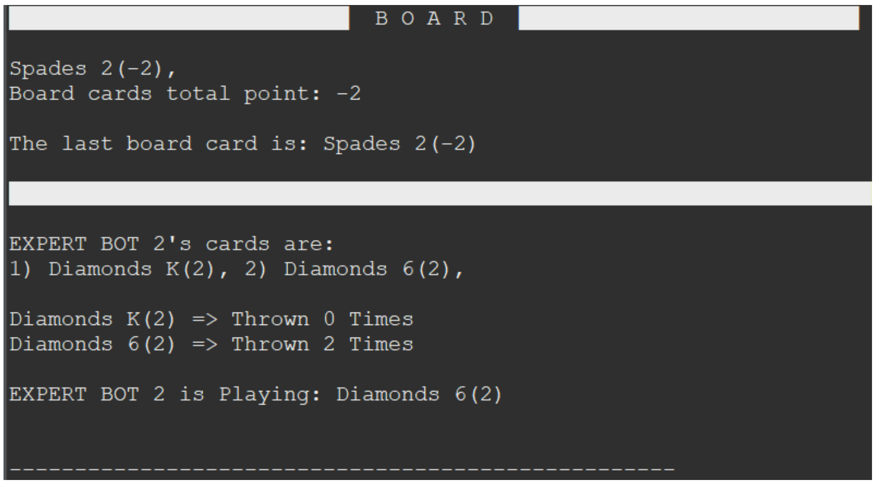
```
                              B O A R D

Board cards total point: 0

The last board card is: null


EXPERT BOT 1's cards are:
1) Hearts 9(5), 2) Hearts 10(5), 3) Hearts 8(5),

Hearts 9(5) => Thrown 0 Times
Hearts 10(5) => Thrown 0 Times
Hearts 8(5) => Thrown 1 Times

EXPERT BOT 1 is Playing: Hearts 8(5)


---------------------------------------------------


                              B O A R D

Hearts 8(5),
Board cards total point: 5

The last board card is: Hearts 8(5)
```

To prevent other players from making misti plays the card that is thrown the most times in case their points are equal.



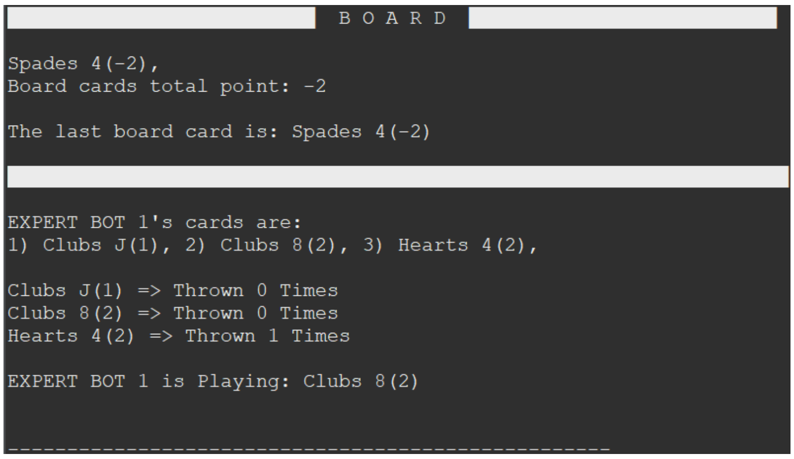
```
                              B O A R D

Hearts K(5), Diamonds 6(2), Hearts 9(5), Clubs 6(2), Spades 2(-4), Clubs 5(2), Spades 9(-4), Spades 8(-4),
Board cards total point: 4

The last board card is: Spades 8(-4)


EXPERT BOT 1's cards are:
1) Spades A(3), 2) Diamonds A(3), 3) Clubs 3(2),

Spades A(3) => Thrown 0 Times
Diamonds A(3) => Thrown 0 Times
Clubs 3(2) => Thrown 1 Times

EXPERT BOT 1 is Playing: Clubs 3(2)
```

Plays the card that both thrown the most and has the lowest point, in case they cannot collect the board.



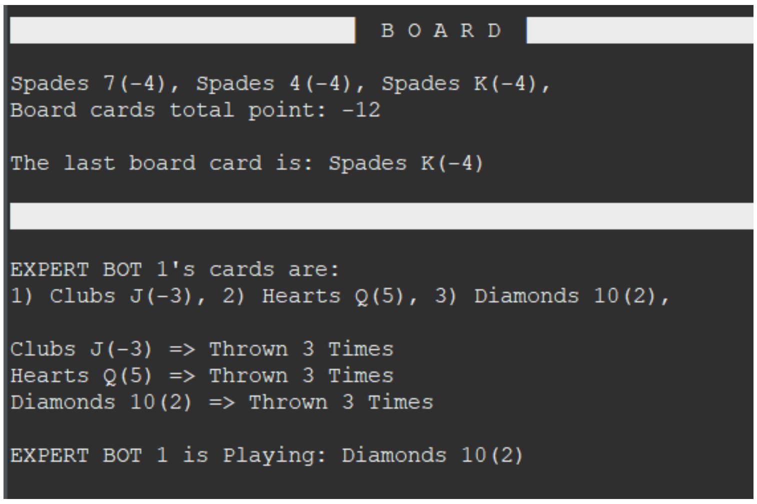
```
B O A R D

Spades 2(-2),
Board cards total point: -2

The last board card is: Spades 2(-2)


EXPERT BOT 2's cards are:
1) Diamonds K(2), 2) Diamonds 6(2),

Diamonds K(2) => Thrown 0 Times
Diamonds 6(2) => Thrown 2 Times

EXPERT BOT 2 is Playing: Diamonds 6(2)


---------------------------------------------------
```

Plays the card that both thrown the most and has the lowest point between other same lowest point cards, in case they cannot collect the board.



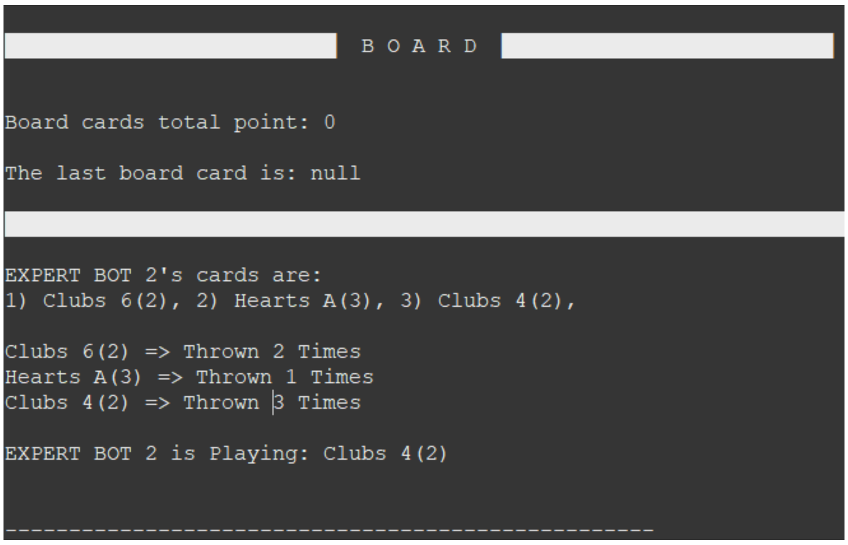
```
B O A R D

Spades 4(-2),
Board cards total point: -2

The last board card is: Spades 4(-2)


EXPERT BOT 1's cards are:
1) Clubs J(1), 2) Clubs 8(2), 3) Hearts 4(2),

Clubs J(1) => Thrown 0 Times
Clubs 8(2) => Thrown 0 Times
Hearts 4(2) => Thrown 1 Times

EXPERT BOT 1 is Playing: Clubs 8(2)


---------------------------------------------------
```

Does not play the same rank or Jack card because there will be no positive points earned. Plays the other card.



```
B O A R D

Spades 7(-4), Spades 4(-4), Spades K(-4),
Board cards total point: -12

The last board card is: Spades K(-4)


EXPERT BOT 1's cards are:
1) Clubs J(-3), 2) Hearts Q(5), 3) Diamonds 10(2),

Clubs J(-3) => Thrown 3 Times
Hearts Q(5) => Thrown 3 Times
Diamonds 10(2) => Thrown 3 Times

EXPERT BOT 1 is Playing: Diamonds 10(2)
```

Can play the jack but does not because the board’s and Jack’s total points are negative.



```
B O A R D


Board cards total point: 0

The last board card is: null


EXPERT BOT 2's cards are:
1) Clubs 6(2), 2) Hearts A(3), 3) Clubs 4(2),

Clubs 6(2) => Thrown 2 Times
Hearts A(3) => Thrown 1 Times
Clubs 4(2) => Thrown 3 Times

EXPERT BOT 2 is Playing: Clubs 4(2)


---------------------------------------------------
```

If the board is empty plays the three times thrownS card to prevent the others from making Misti.