PROJECT DOCUMENT

1. Personal information

- Title: Cassino
- Student's name: Hanh Nguyen Vu
- Student number: 101720784
- Degree program: Bachelor of Data Science
- Year of studies: 2023-2026
- Date: 03/04/2024

2. General description

Chosen level: Intermediate

This is a digital version of the Cassino card game. In this game, players compete to acquire points by strategically playing cards from their hands to take cards off the table. The game continues until no player has cards in his/her hand.

In my variation, the points are added up to the value of the cards the player captured. The game will start a new round if there exists a player that has over 16 points. The player next to the dealer will be the new dealer for the next round. They will play until the game ends, and after that the bonus points are added to the players' score according to the scoring rules. The last player to capture something will obtain the remaining cards on the table to his/her pile. (It has been approved by the supervisor).

Each round begins with the dealer shuffling the deck and dealing four cards to each player and four cards face-up on the table. Players take turns playing cards from their hands, either to capture cards from the table or to place cards on the table (to build or just when not wanting to do anything). The goal is to capture cards in combinations that match the value of the card played from their hand.

Special cards like Aces, Diamonds-10, and Spades-2 have enhanced values and can be strategically used to capture specific combinations of cards.

The program features a text-based user interface where players can see the game state, make their moves, and interact with the game. It also includes an algorithm to prevent illegal moves.

Additionally, the program supports saving and loading game states from text files, allowing players to continue their games later.

3. User interface

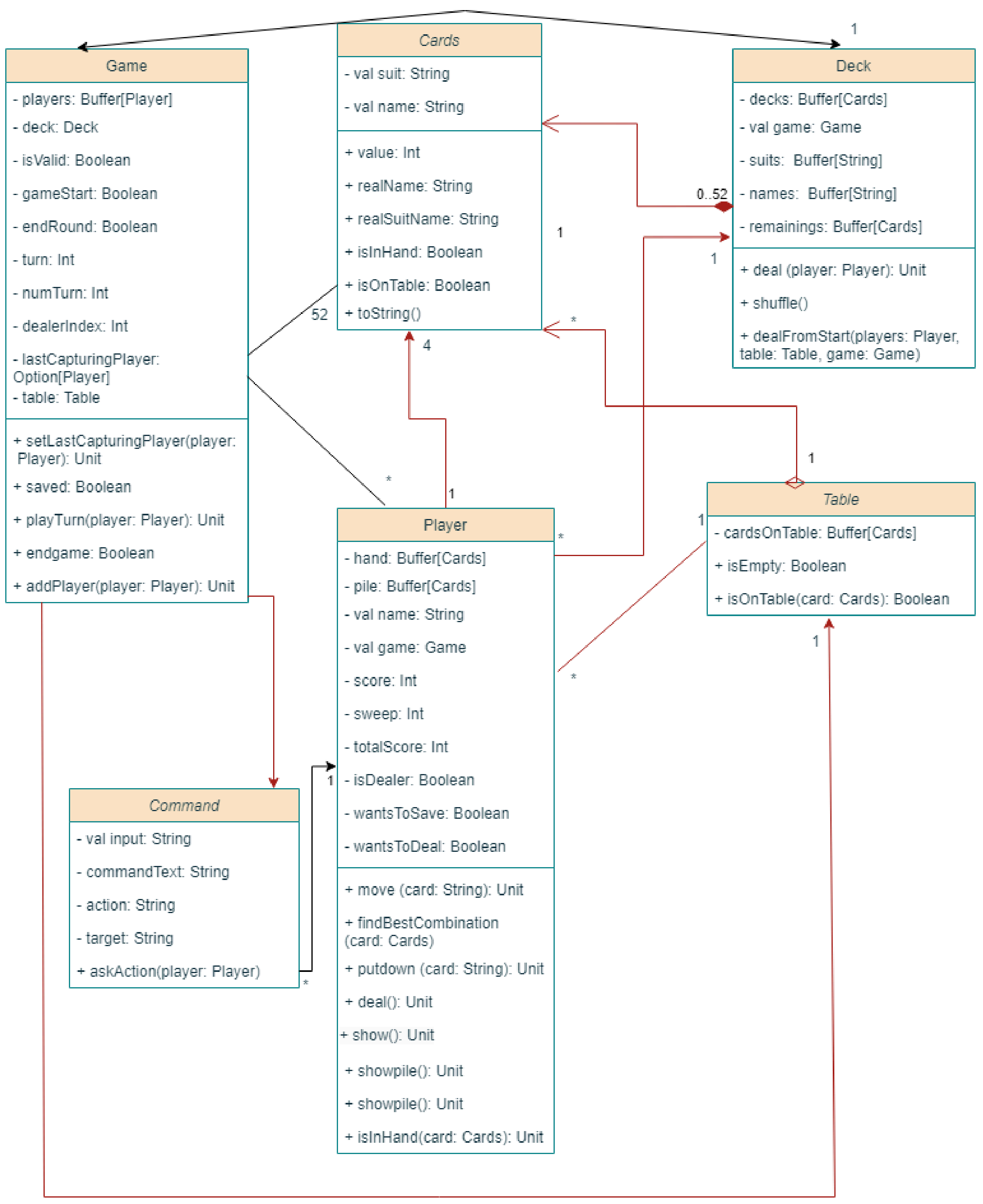
Launch the program by running the game object (textBased.scala to be specific).

The game will first ask if the user wants to load an existing game or play a new game. Press “load” if he wants to load, otherwise any input except from “load” can be pressed to start a new game.
After that, if a new game starts, it will ask for an integer input of the number of players. If it is smaller than 2 or not an integer, the game will ask the user again until the input is valid. Then the players can type their names, but their names cannot be identical. If identical, the game will ask them to put another name. After that, the game starts.

There are 5 commands that are considered valid:

- “play” : This is valid for all players except the current dealer. It is used to play cards (put down or capture something). (E.g. “play ace“) If the played card is not on the player’s hand, it will count as invalid and the player must enter another command. It will automatically avoid choosing a card of Spades (as the scoring counts the number of Spades) or choosing a valuable card with the same name if there are more than one card that shares the same name on the player’s hand and there exists no possible combinations.
  For example: 10 of Spades and 10 of Diamonds. It will choose 10 of Spades as 10 of Diamonds has value 16, which is beneficial for the player. For 2 of Spades (value 15) and 2 of Hearts (2), it will put down 2 of Hearts. For 10 of Clubs and 10 of Spades, it will put down 10 of Clubs as the game will count the Spades cards in players’ pile for bonus points.
- “hand”: For players to see their hand of cards in the game, including the dealer.
- “pile”: For players to see their pile of cards in the game, including the dealer.
- “save”: Save the game as a text file.
- “deal”: Only for the current dealer so that the game skips to the next player.

4. Program structure



- Main sub-parts:
    - Game logic (the classes): Handles the core logic of the card game, including managing players, decks, hands, turns, and scoring.
    - User interface (textBased.scala): Manages interactions between the user and the game, displaying game state, receiving user input, and providing feedback.
    - Game Saver/Loader: Responsible for saving and loading game states to and from files.

- Logic classes:
  - Game: Manages the overall game state, including players, decks, turns, and scoring.
  - Player: Represents a player in the game, handling their hand, pile, score, and actions.
  - Deck: Represents the deck of cards used in the game, handling shuffling, dealing, and remaining cards.
  - Table: Represents the table where cards are played, managing the cards on the table.
  - Cards: Represents individual cards with suits, values, and game context.
- User interface classes:
  - textBased: Manages the game progress, determines if the move is valid or not.
  - GameSaver, gameLoad: Handles the conversion of game state to and from strings for saving/loading for textBased.

At first the class Player, Deck and Cards cannot call class Game, but after working on the project I actually realized that they needed to call the game, otherwise there are no other ways to check if the table or any player of the game contains a card. Some boolean values that I need will not be able to work properly throughout the game.

5. Algorithms

Dealer:

The game rotates the dealer position after each round, starting from the first player. It does this by assigning a boolean variable in class Player and changing it once a new round starts.

Turns:

The game assigns turns to players by using the method that changes the index of the current player in the players buffer. If the command is "play" and it is valid, the turn will change to the next player in line. If they only call "hand" or "pile" or "save" then it will not change yet.

Playing algorithm:

When command "play" is called, the algorithm makes iterations (summing up values) by looping among the cards in the player's hand to determine which cards will be called according to his/her command.

Then it will iterate through all combinations that can be made from the cards on the table, choosing the combinations of the cards that add up to the played card's value and then selecting the combination that has the largest size. The cards in this combination will be added into the player's pile and their values are added to the score of the player. And also, this player is also considered as the last person to capture something (for the end game's scoring). If there is no combination can be found then it just merely puts the card down on the table and does not add anything into the pile.

When a player exceeds 16 then a new round will start, it will make the scores return to 0 and add the last round score to each player's total score. The game will clear the table, clear the players' hands, shuffle the remaining decks and deal to the players again.

There is a method that has a boolean value to check if the game ends or not (every player has run out of cards) and the end game's scoring will proceed if it is called.

Validating algorithm: The algorithm checks whether the combination of cards selected from the table forms a valid move based on the following criteria:

The sum of the values of the selected cards equals the value of the card played from the player's hand.

The player must have at least one card in their hand that matches the value of the card played.

The card they call to capture must be present on the table.

If the command is not valid, the game will give a reason for it - the player has to make another move. If the number of players or the name of the loaded file is not valid (the file does not exist or the game in the file has already been over), then it will return a feedback that shows the error. Validity can be stored as a false boolean variable and if it is valid, I will change it to "true".

It also does the same for checking if the current position of the player (dealer or player) is able to call the command.

Saving/loading:

The sweeps are counted throughout the game and only added to the total score once the game has ended, so it is also in the saving file, along with the table, the deck, last capturer and the turns and who is currently owning the turn. It also adds each player's score, total score, hand and pile. The gameLoad object will locate the lines of information, convert the text to number or cards according to the parameters' original types, change the parameters value into the game and return it to the interface. The "turns" part works as a sign to know if the game has ended or not.

Scoring:

It is based on various factors, such as the value of captured cards, sweeps, and special combinations. Players accumulate points based on the cards they capture and the additional scoring conditions. The game will then decide the player that has the highest scores throughout the rounds as the winner.

Interface: It makes sure that the table is shown in every turn and the player's hand is shown in their turn. It will also show the pile for them when they complete the command. If something is invalid, it will loop the command request until it is valid. It also offers to save the game when it is over, along with saving it when a player calls "save".

6. Data structures

Dynamic structures are best suited for this game to store information, as the game state is changed after every move from each player, so flexible types like dynamic structures are needed. For Deck, Table and Player, the card's information can be stored as a mutable buffer. The scores are stored inside a numerical variable in Player.

The game is saved as a text file, and the user interface, the game loader and game saver are objects. The rest are classes, used to act as a foundation for the game logic.
It could have used fixed structures, but these may be less flexible and require additional handling for changes in the game state.

7. **Files and Internet access**
The program deals with text files for saving and loading the game state. These text files store the current state of the game, including player information, scores, hands, piles, cards on the table, and deck.
Information about the game state, such as the current turn, the current player and last capturer is provided.
No internet access or specific protocols are required for the program to function. All data operations are performed locally on text files stored on the user's device.

8. **Testing**
The program was tested using a combination of manual testing through the command line interface and automated unit tests. I tested the game logic through the unit tests and the game state through the interface.
For the classes, separate tests were built to cover critical components of the program, such as the game logic, player actions, scoring mechanisms, and file handling.
Unit tests were written using the ScalaTest framework to verify the behavior of individual methods and functions.
For the interface, various scenarios were tested, including starting a new game, loading a saved game, playing turns, capturing cards, ending rounds, and saving the game state.Inputs were provided to test both valid and invalid commands, and error handling cases. Special attention was given to testing game mechanics such as dealing cards, capturing sets, starting a new round, calculating scores, and determining the winner.

9. **Known bugs and missing features**

Error handling for invalid input commands could be more flexible. Currently, the program handles the exceptions but may not cover all edge cases. Implementing more error handling would improve the experience and make the program more adaptive to unexpected inputs. I tried to cover as much as I could but maybe not all is covered.
Also, I have not implemented the tasks in the demanding level, such as the graphical interface, computer opponents and its tactics.

10. **3 best sides and 3 weaknesses**
    - Best sides:
        - Clear arrangement: The program is organized into well-defined classes and in separate packages so it is clear to see. The classes have separate files so they are not too massive.

    - Comprehensive user interface: The program provides a text-based user interface that guides users through the gameplay with clear prompts and instructions. Its feedback allows users to interact with the game smoothly.
    - Testing: I used a lot of unit tests and draft object files to test the classes and the objects. I also have tested various scenarios with the interface and tried to save one game a lot to observe even small changes in my file so as to check that everything works properly.
- Weaknesses:
    - Clarity: There may be areas where additional comments or documentation could improve clarity.
    - Some little-used variables: There are some variables that are used just once or twice, so maybe they can be reduced by changing some parts in my code.
    - Error handling: While the program includes error handling, the error messages provided to the user may sometimes be vague as I put in just two cases of specific exceptions to deal with, so their messages may lack specific information.

11. Deviations from the plan, realized process and schedule

- In the first two weeks, I have implemented the classes for the game logic, and a small part of the text-based interface.
- In the following two weeks from that, I have completed the classes testing, implemented the game interface and also added the functionality of saving and loading the game by the objects GameSaver and gameLoad. In short, I have finished the game at the intermediate level.
- In the following two weeks, I have tried to implement the graphical interface and implement a computer tactic.

I think the schedule goes accordingly except for the last week, and there is only one change. It is that I test the game's algorithm first and leave the graphical interface and the tactics last in line. At first I thought the classes would be separate and I tried to implement the game as an object, but when I actually started building it, I realized that the game has its own attributes that I could not make it as an object and use it as the interface directly.
The biggest difference for me is the adjustment in the order of implementation, focusing on testing and refining the game's algorithm before tackling the graphical interface and computer tactics. This adjustment is from the fact that I aimed to complete the intermediate level first before moving onto the next level.
From this, I have learned that I should plan my path more clearly and spend time thinking about it first so that I can avoid problems when actually implementing the game.

12. Final evaluation

I think my game works smoothly according to the rules of my variation and it shows that I can fulfill the requirements. Overall, the game works properly. The game can be improved by adding graphical features as well as computerized features like tactics or

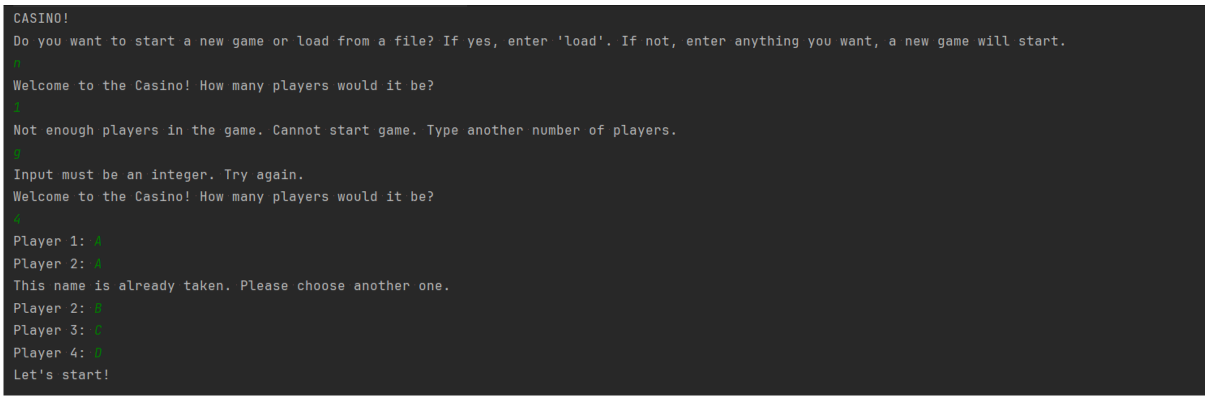
automatic opponents to make it suitable for less than 2 players. And I think my style can also be improved, but my choice of data structures will not change because using mutable buffers and variables are suitable for this game.
If I started the project again, I would have planned my way more clearly and implemented a graphical interface along with the algorithms.

13. References

Scala API Documentation
ScalaFX (https://www.scalafx.org/)
Wikihow - Complete Game Instructions for Casino (Cassino) https://www.wikihow.com/Play-Casino-(Card-Game)
Apart from this course's material, I also have learned from this link to save the game as a text file:
https://www.geeksforgeeks.org/stringbuilder-in-scala/

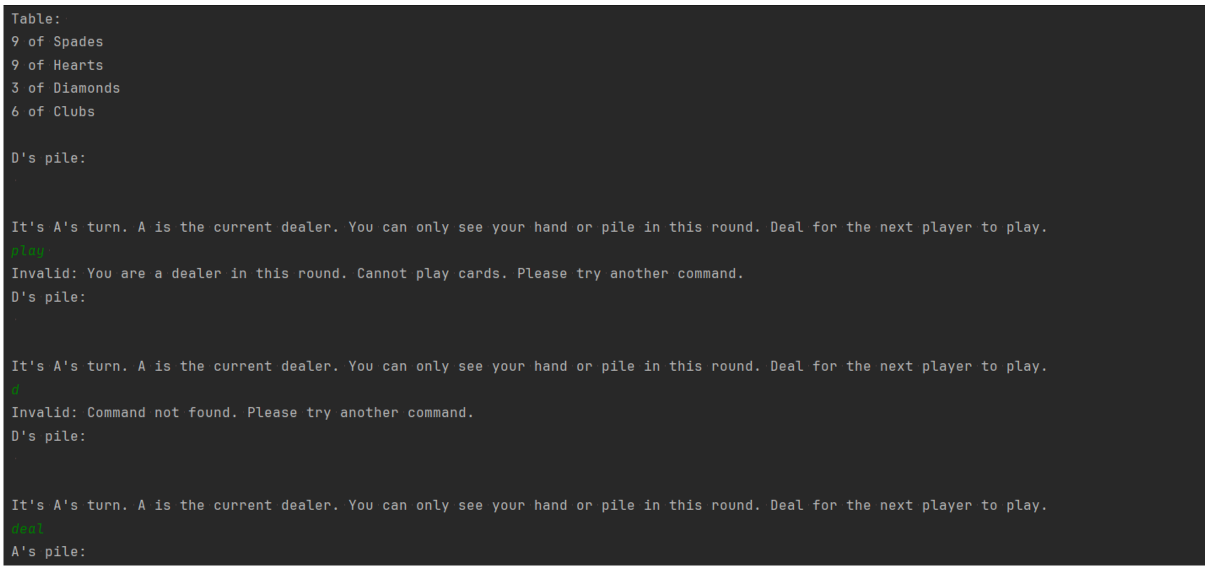
14. Appendixes

- **Illustration examples:**
  - When start a new game:

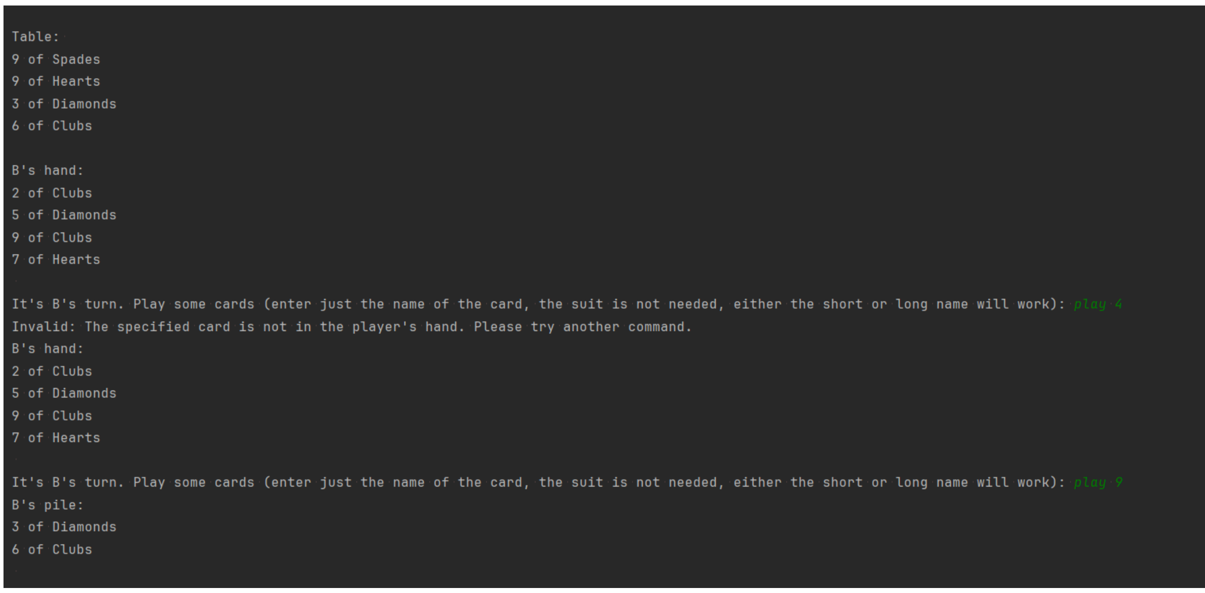


```
CASINO!
Do you want to start a new game or load from a file? If yes, enter 'load'. If not, enter anything you want, a new game will start.
n
Welcome to the Casino! How many players would it be?
1
Not enough players in the game. Cannot start game. Type another number of players.
g
Input must be an integer. Try again.
Welcome to the Casino! How many players would it be?
4
Player 1: A
Player 2: A
This name is already taken. Please choose another one.
Player 2: B
Player 3: C
Player 4: D
Let's start!
```

  - During the game:



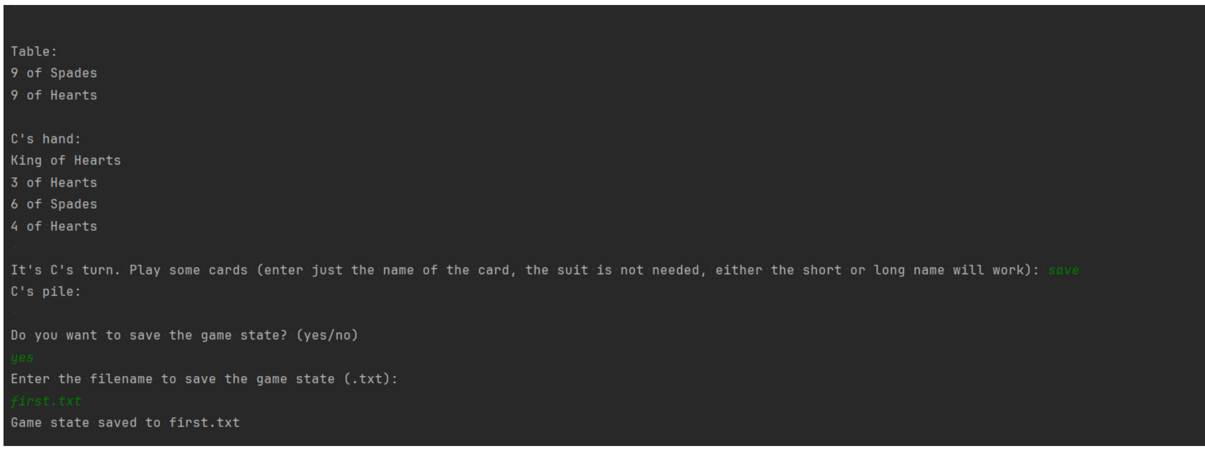
```
Table:
9 of Spades
9 of Hearts
3 of Diamonds
6 of Clubs

D's pile:


It's A's turn. A is the current dealer. You can only see your hand or pile in this round. Deal for the next player to play.
play
Invalid: You are a dealer in this round. Cannot play cards. Please try another command.
D's pile:


It's A's turn. A is the current dealer. You can only see your hand or pile in this round. Deal for the next player to play.
d
Invalid: Command not found. Please try another command.
D's pile:


It's A's turn. A is the current dealer. You can only see your hand or pile in this round. Deal for the next player to play.
deal
A's pile:
```



```
Table:
9 of Spades
9 of Hearts
3 of Diamonds
6 of Clubs

B's hand:
2 of Clubs
5 of Diamonds
9 of Clubs
7 of Hearts

It's B's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work): play 4
Invalid: The specified card is not in the player's hand. Please try another command.
B's hand:
2 of Clubs
5 of Diamonds
9 of Clubs
7 of Hearts

It's B's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work): play 9
B's pile:
3 of Diamonds
6 of Clubs
```

(The best combination to capture has the greatest number of cards).

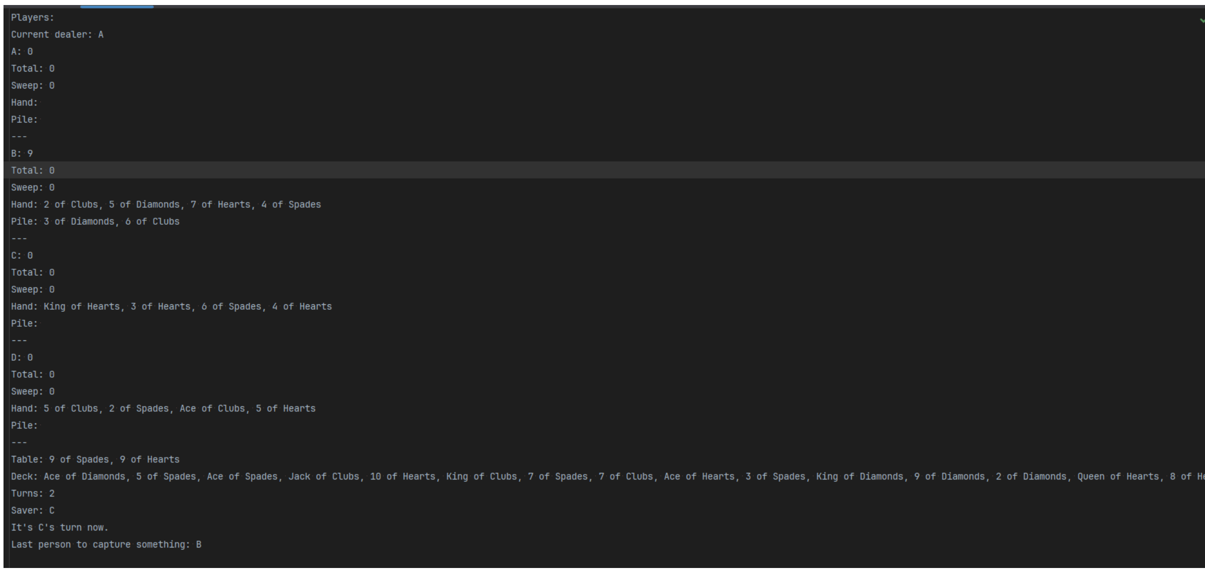
- When save the game:



```
Table:
9 of Spades
9 of Hearts

C's hand:
King of Hearts
3 of Hearts
6 of Spades
4 of Hearts

It's C's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work): save
C's pile:

Do you want to save the game state? (yes/no)
yes
Enter the filename to save the game state (.txt):
first.txt
Game state saved to first.txt
```

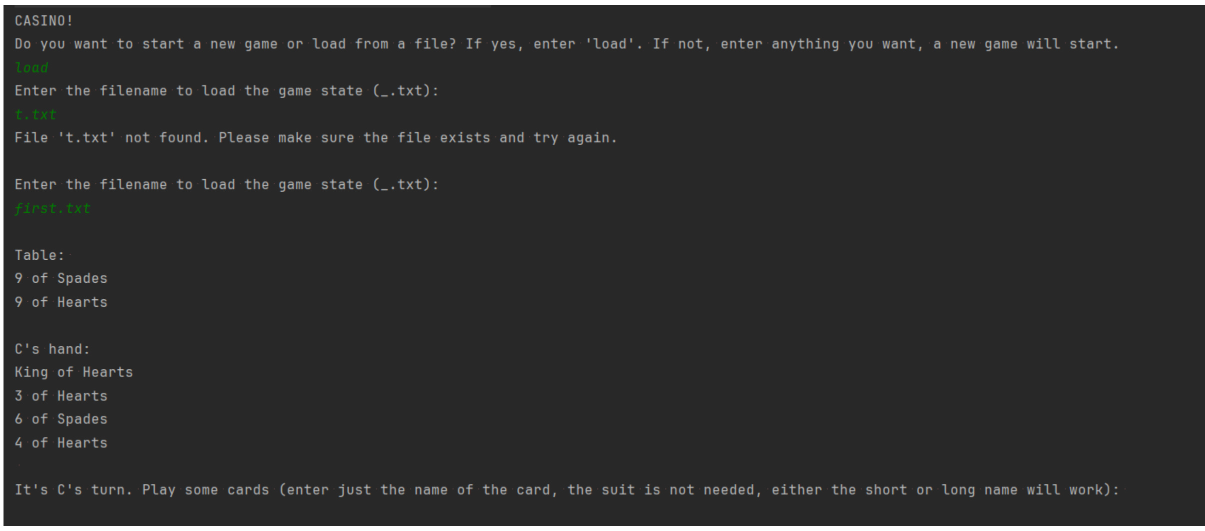
(The file can be with ".txt" or not with it, it still saves the game in the format of a text file, but if the file is named with ".txt" then when trying to load the game from the file again, you must type with".txt").

The file would then be:



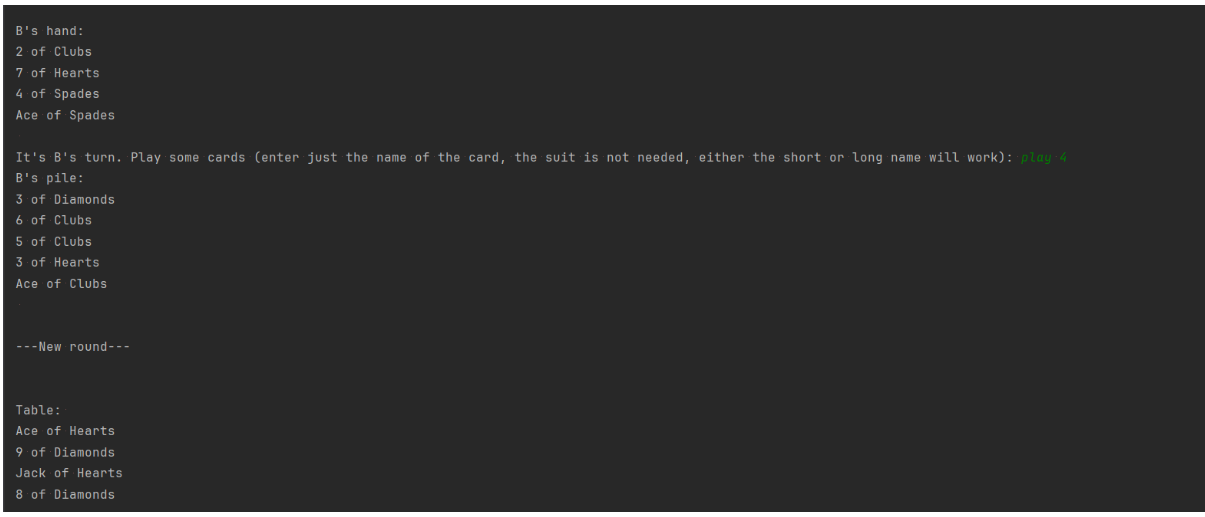
```
Players:
Current dealer: A
A: 0
Total: 0
Sweep: 0
Hand:
Pile:
---
B: 9
Total: 0
Sweep: 0
Hand: 2 of Clubs, 5 of Diamonds, 7 of Hearts, 4 of Spades
Pile: 3 of Diamonds, 6 of Clubs
---
C: 0
Total: 0
Sweep: 0
Hand: King of Hearts, 3 of Hearts, 6 of Spades, 4 of Hearts
Pile:
---
D: 0
Total: 0
Sweep: 0
Hand: 5 of Clubs, 2 of Spades, Ace of Clubs, 5 of Hearts
Pile:
---
Table: 9 of Spades, 9 of Hearts
Deck: Ace of Diamonds, 5 of Spades, Ace of Spades, Jack of Clubs, 10 of Hearts, King of Clubs, 7 of Spades, 7 of Clubs, Ace of Hearts, 3 of Spades, King of Diamonds, 9 of Diamonds, 2 of Diamonds, Queen of Hearts, 8 of He
Turns: 2
Saver: C
It's C's turn now.
Last person to capture something: B
```

- When load the game file:

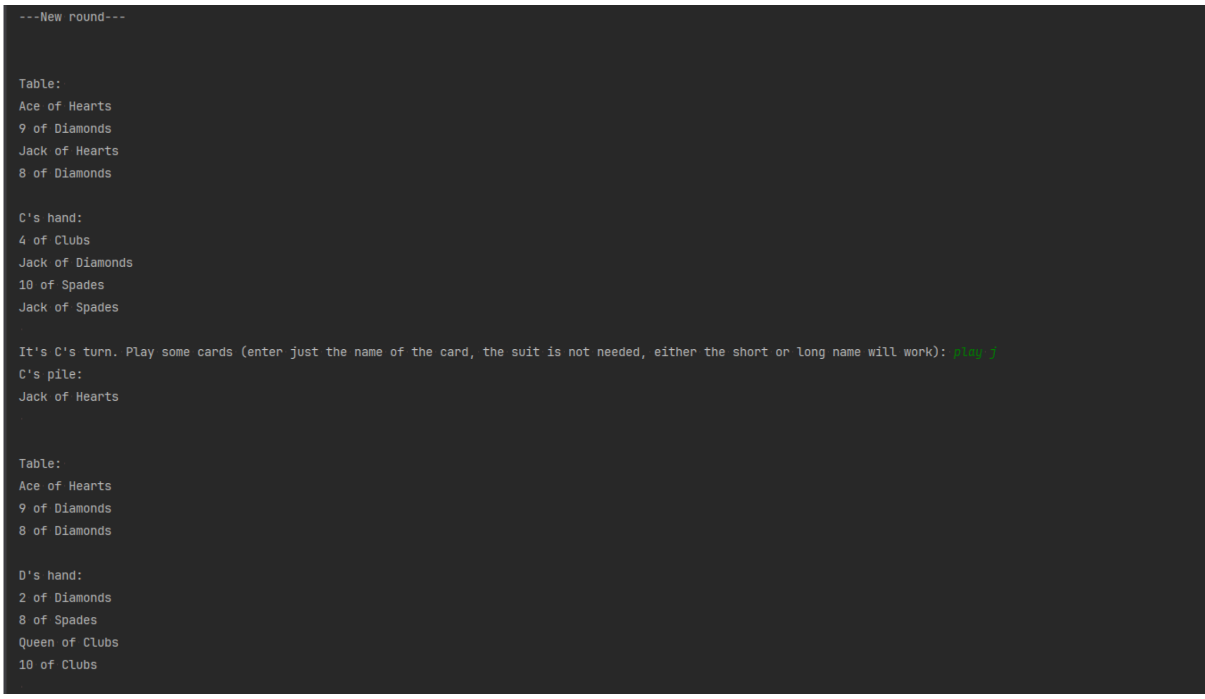


```
CASINO!
Do you want to start a new game or load from a file? If yes, enter 'load'. If not, enter anything you want, a new game will start.
load
Enter the filename to load the game state (_.txt):
t.txt
File 't.txt' not found. Please make sure the file exists and try again.

Enter the filename to load the game state (_.txt):
first.txt

Table:
9 of Spades
9 of Hearts

C's hand:
King of Hearts
3 of Hearts
6 of Spades
4 of Hearts

It's C's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work):
```

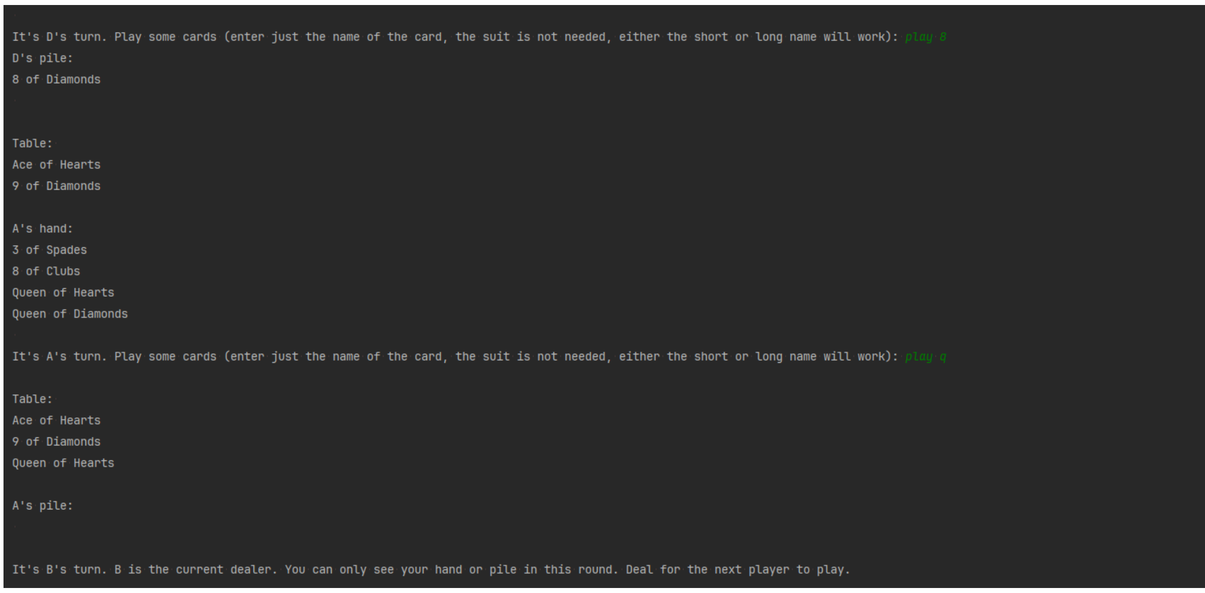
- When a new round starts:



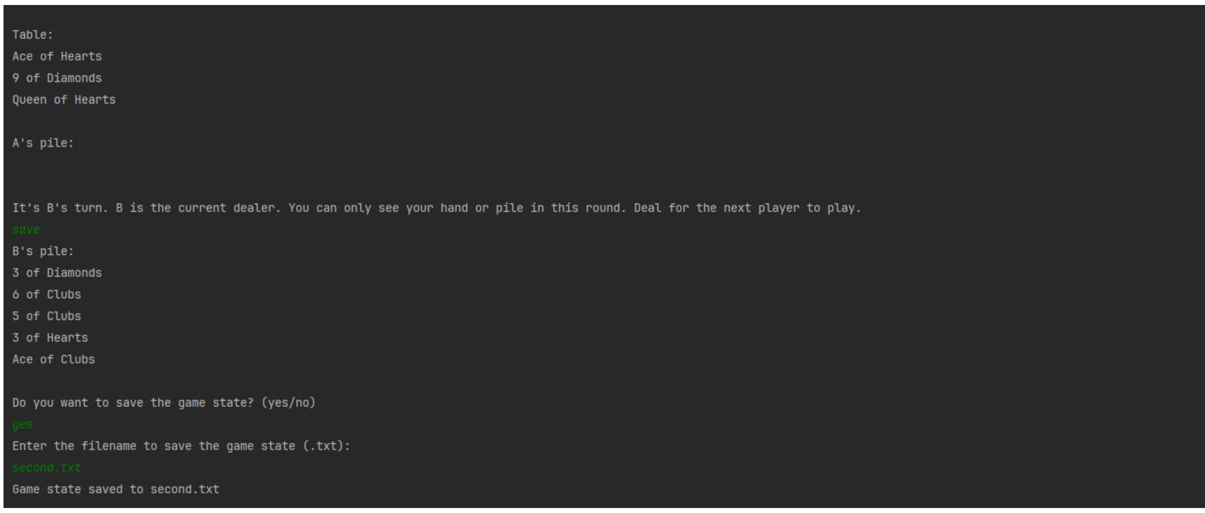
```
B's hand:
2 of Clubs
7 of Hearts
4 of Spades
Ace of Spades

It's B's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work): play 4
B's pile:
3 of Diamonds
6 of Clubs
5 of Clubs
3 of Hearts
Ace of Clubs


---New round---


Table:
Ace of Hearts
9 of Diamonds
Jack of Hearts
8 of Diamonds
```



```
---New round---


Table:
Ace of Hearts
9 of Diamonds
Jack of Hearts
8 of Diamonds

C's hand:
4 of Clubs
Jack of Diamonds
10 of Spades
Jack of Spades

It's C's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work): play j
C's pile:
Jack of Hearts


Table:
Ace of Hearts
9 of Diamonds
8 of Diamonds

D's hand:
2 of Diamonds
8 of Spades
Queen of Clubs
10 of Clubs
```

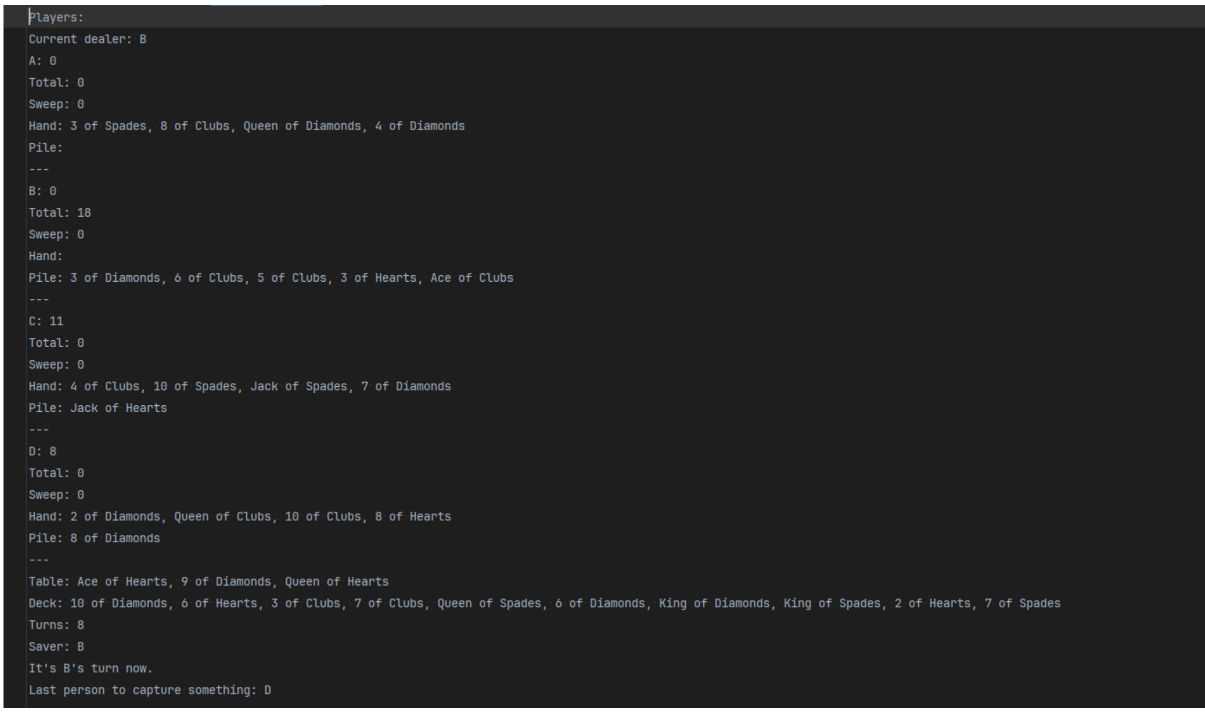


```
It's D's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work): play 8
D's pile:
8 of Diamonds


Table:
Ace of Hearts
9 of Diamonds

A's hand:
3 of Spades
8 of Clubs
Queen of Hearts
Queen of Diamonds

It's A's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work): play q

Table:
Ace of Hearts
9 of Diamonds
Queen of Hearts

A's pile:


It's B's turn. B is the current dealer. You can only see your hand or pile in this round. Deal for the next player to play.
```



```
Table:
Ace of Hearts
9 of Diamonds
Queen of Hearts

A's pile:


It's B's turn. B is the current dealer. You can only see your hand or pile in this round. Deal for the next player to play.
save
B's pile:
3 of Diamonds
6 of Clubs
5 of Clubs
3 of Hearts
Ace of Clubs

Do you want to save the game state? (yes/no)
yes
Enter the filename to save the game state (.txt):
second.txt
Game state saved to second.txt
```

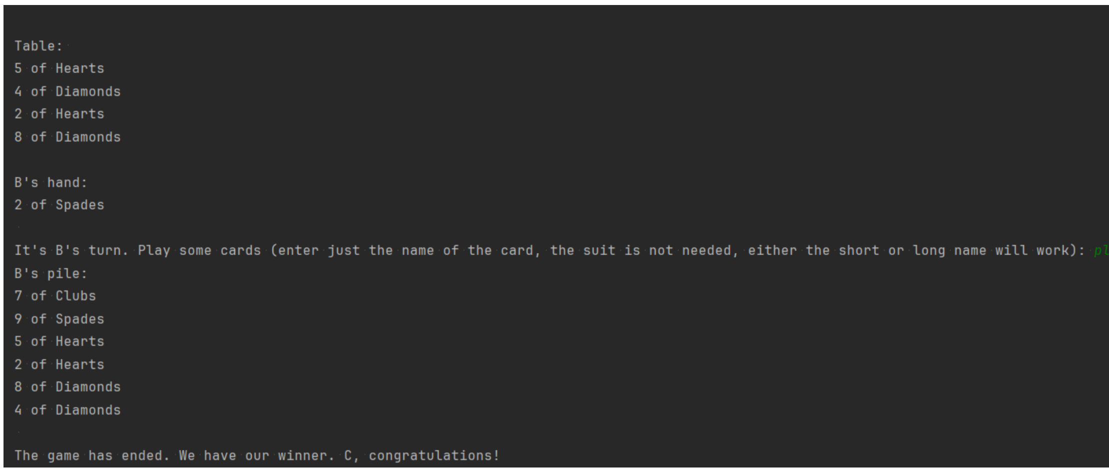
The game will then be saved for the second time, and the points for the last rounds will be added to the total score, not the current score.



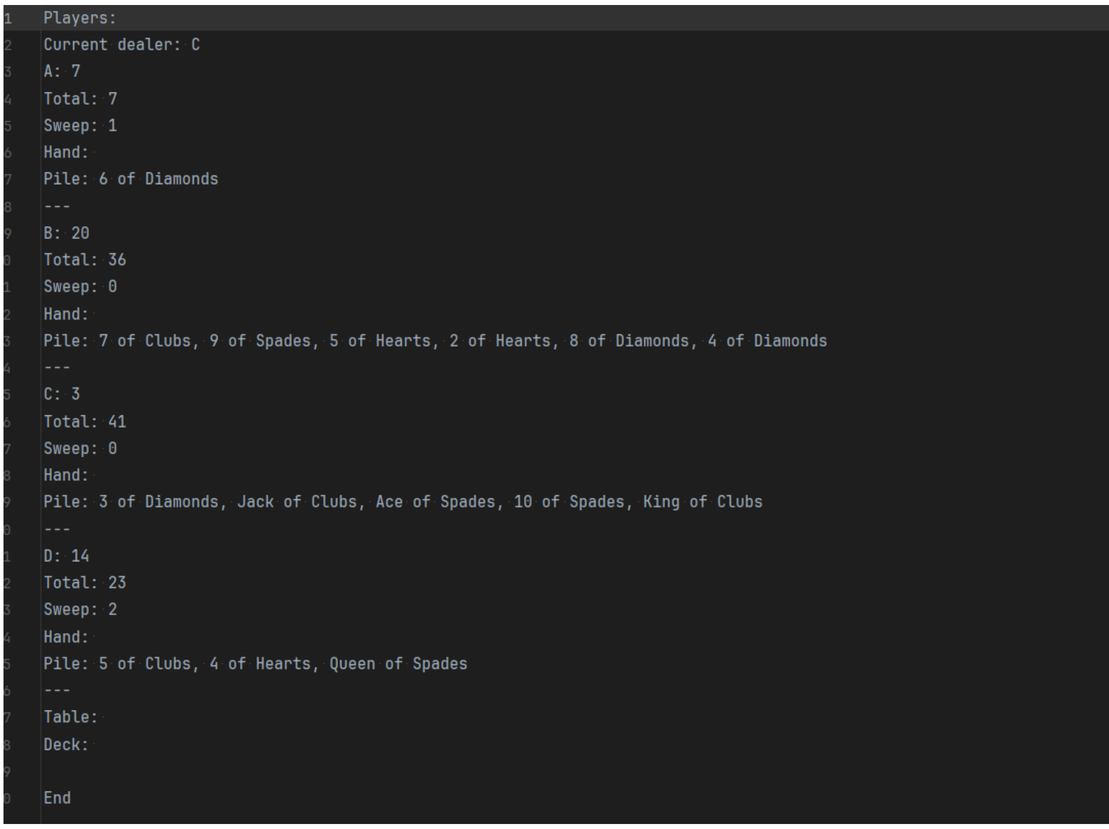
```
Players:
Current dealer: B
A: 0
Total: 0
Sweep: 0
Hand: 3 of Spades, 8 of Clubs, Queen of Diamonds, 4 of Diamonds
Pile:
---
B: 0
Total: 18
Sweep: 0
Hand:
Pile: 3 of Diamonds, 6 of Clubs, 5 of Clubs, 3 of Hearts, Ace of Clubs
---
C: 11
Total: 0
Sweep: 0
Hand: 4 of Clubs, 10 of Spades, Jack of Spades, 7 of Diamonds
Pile: Jack of Hearts
---
D: 8
Total: 0
Sweep: 0
Hand: 2 of Diamonds, Queen of Clubs, 10 of Clubs, 8 of Hearts
Pile: 8 of Diamonds
---
Table: Ace of Hearts, 9 of Diamonds, Queen of Hearts
Deck: 10 of Diamonds, 6 of Hearts, 3 of Clubs, 7 of Clubs, Queen of Spades, 6 of Diamonds, King of Diamonds, King of Spades, 2 of Hearts, 7 of Spades
Turns: 8
Saver: B
It's B's turn now.
Last person to capture something: D
```

- When the game ends: (I use another file to describe it more clearly):

  B captured 5,2 and 8 (as 2 of Spades is equal to 15), and because B is the last capturer, B then can obtain the remaining card on the table after that. You can see in the screenshot that 4 is the last one to be added to the pile.



```
Table:
5 of Hearts
4 of Diamonds
2 of Hearts
8 of Diamonds

B's hand:
2 of Spades

It's B's turn. Play some cards (enter just the name of the card, the suit is not needed, either the short or long name will work): pl
B's pile:
7 of Clubs
9 of Spades
5 of Hearts
2 of Hearts
8 of Diamonds
4 of Diamonds

The game has ended. We have our winner. C, congratulations!
```



```
Players:
Current dealer: C
A: 7
Total: 7
Sweep: 1
Hand:
Pile: 6 of Diamonds
---
B: 20
Total: 36
Sweep: 0
Hand:
Pile: 7 of Clubs, 9 of Spades, 5 of Hearts, 2 of Hearts, 8 of Diamonds, 4 of Diamonds
---
C: 3
Total: 41
Sweep: 0
Hand:
Pile: 3 of Diamonds, Jack of Clubs, Ace of Spades, 10 of Spades, King of Clubs
---
D: 14
Total: 23
Sweep: 2
Hand:
Pile: 5 of Clubs, 4 of Hearts, Queen of Spades
---
Table:
Deck:

End
```