



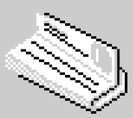


Blackjack Terminal App T1A3

By Jordan Benjamin

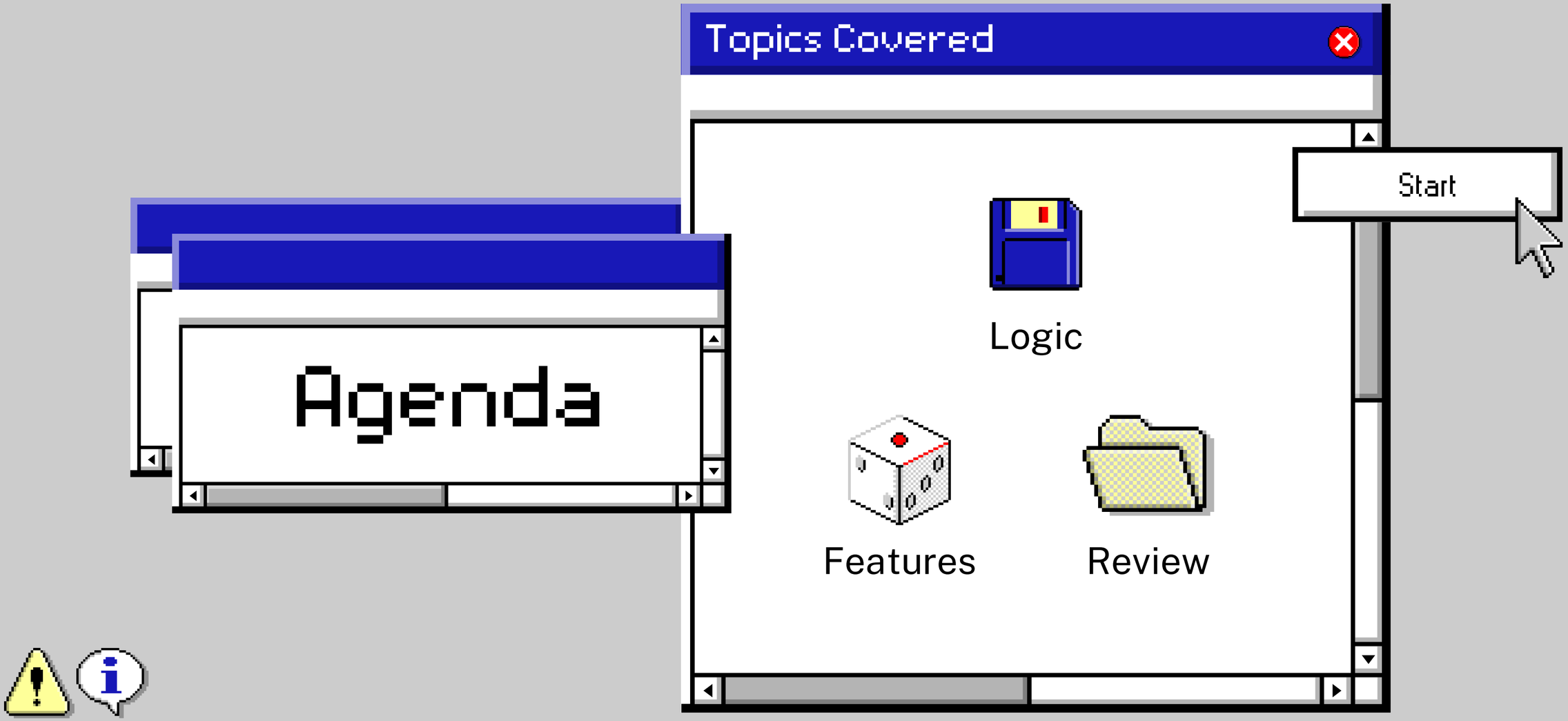




11:11PM

nda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Agenda • Ager



The Premise

Blackjack fun right on the terminal, the premise is simple, try and get to 21 as close as possible without going over. This application will be a stripped down version of blackjack, with house rules made specifically for this version of the game.

Logic

Back to Agenda Page

Program Requirements

Part 1

1. The user will recieve a starting hand of 2 random cards, but the dealer only gets dealt 1 card as only 1 is revealed.

2. Figure out if the user has a Blackjack which is a picture card like a King, Queen, Jack (value of 10) or a number 10 card, along with an Ace card (value of 11) which equals to total value of 21, this means its a Blackjack and the user automatically wins.

3. Figure out if the dealer (computer) has a Blackjack, if so, the dealer automatically wins.

4. Figure out if both the user and dealer (computer) has a Blackjack, if so, then the dealer wins even though user has a blackjack.

5. Calculate the user's scores based on the values of their card.

Program Requirements

Part 2

6. Calculate the dealer's (computer) scores based on the values of their card

7. If both the user and dealer (computer) draw an Ace, automatically count this Ace's value as 11, but if the total of the hand (current cards they have) goes over 21, then the Ace's value is 1 instead.

8. Only reveal dealers (computer) first card to the user

9. The game will end automatically when the user's card totals to a score of 21 or if either the user or dealer (computer) gets a Blackjack

10. Prompt the user if they want to draw another card or in the case of Blacjack lingo, another 'hit'.

Program Requirements

Part 3

11. Prompt the user if they want skip a turn or in the case of Blacjack lingo, a 'stand' or 'standing'.

12. When a user stands and no longer wants to draw a card, they finish their turn and the dealer's (computer) turn starts. The dealer will keep drawing cards unless their total score goes over 16.

13. Compare users score or dealers (computer) and find out if it's a win, loss, or draw.

14. Display the users and dealers (computer) final hand along with their scores at the end of the game

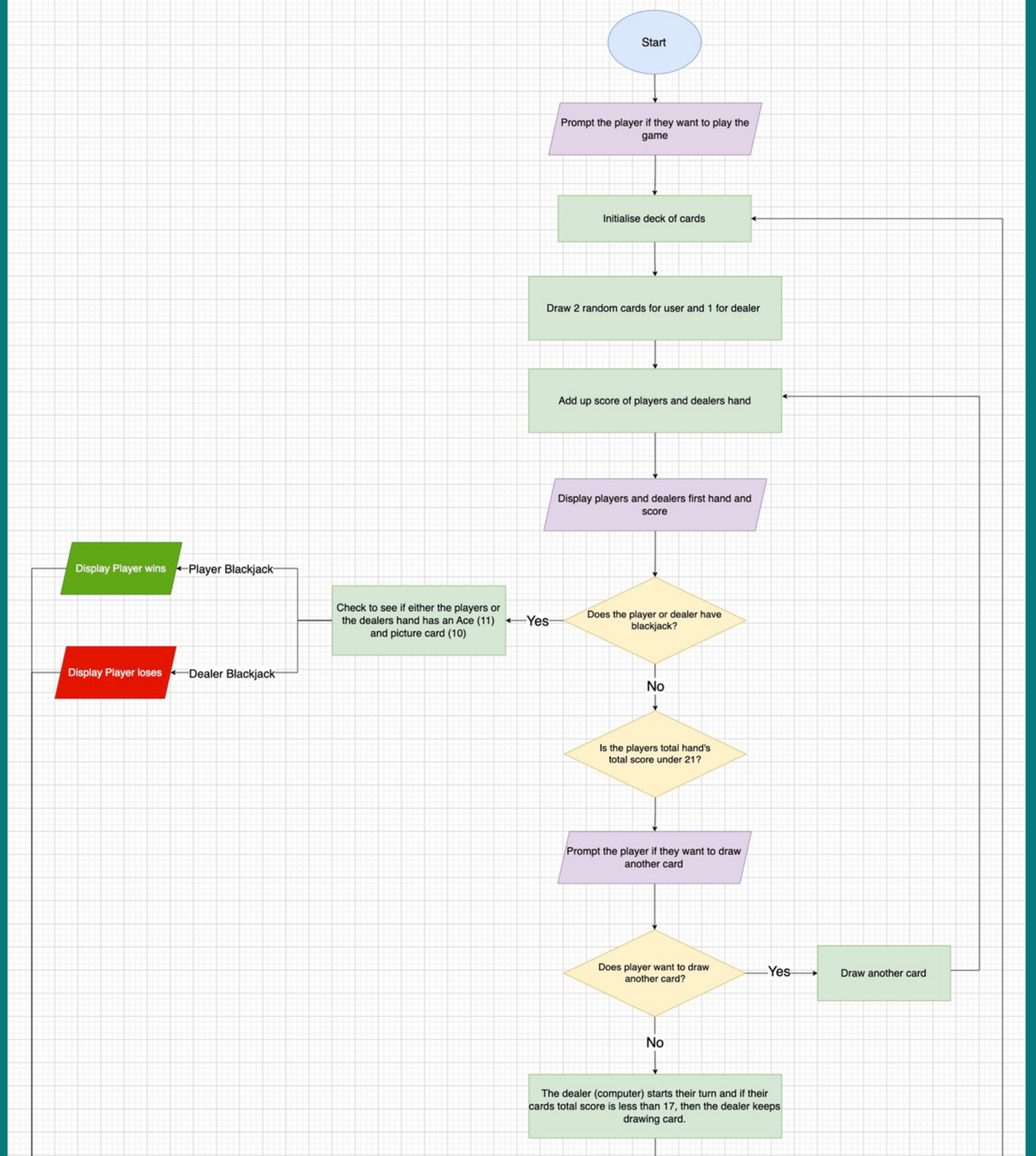
15. When the game ends, ask the user if they'd like to play again. Clear the terminal for a fresh slate.

Flowchart Part 1

Gameplay logic flow

I created a flowchart in order to lay out the steps and main logic of the game, again, I wanted to focus and prioritise the main features that makes up the play of the game which the flowchart presents those steps.

The flow cart flows this way

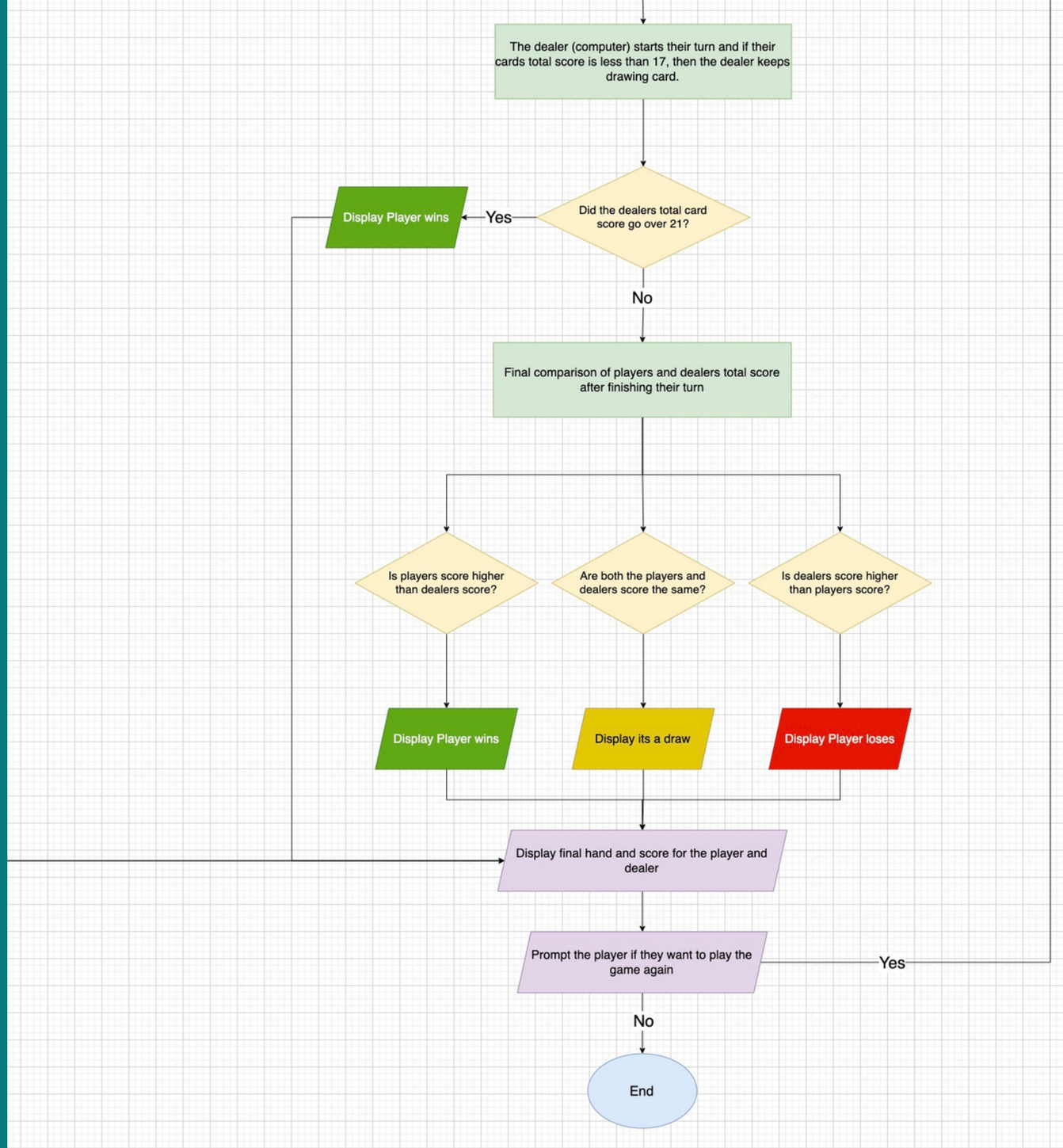


Flowchart Part 2

Gameplay logic flow

This is the second part of the flowchart

The flow cart flows this way



Features

Back to Agenda Page

Deck of cards

```python
# Initialises dictionary of cards
cards_dict = {
    'A': 11,
    '2': 2,
    '3': 3,
    '4': 4,
    '5': 5,
    '6': 6,
    '7': 7,
    '8': 8,
    '9': 9,
    '10': 10,
    'J': 10,
    'Q': 10,
    'K': 10,
}

# Initialises list of suits
suits = ['♦', '♥', '♠', '♣']
```

Pairing of cards with their respective score using a dictionary + 4 suits in a list

```python
# Initialises empty deck list
deck = []
```

Global deck variable

```python
def init_deck():
    '''This function initialises deck of cards which adds each card to its own respective suits adding up to a total of 52 deck of cards, additionally shuffle the deck of cards'''

    for card in cards_dict:
        for suit in suits:
            cards = suit + card
            deck.append(cards)
    np.random.shuffle(deck)
```

Using a nested for loop to achieve the pairing of the card and the suit. The deck will be shuffled before the start of every game to maximise the randomness of a game like Blackjack.

Deck of cards

```python
def draw_card():
    '''This function draws a card from the deck, checks condition if deck of cards is empty then initialise the
    deck, otherwise pop the card from the deck and return that card'''

    if len(deck) == 0:
        init_deck()
    card = deck.pop(0)
    return card
```

Draws a card from the deck and also removes it, simulating a real deck of cards

```python
def deal_card(n, hand):
    '''This function adds drawn card for both player and dealer'''

    for i in range(n):
        card = draw_card()
        hand.append(card)
```

Parameters are used to determine number of cards drawn and appends to hand

Calculate Score

Handles all the logic to determine scores of cards in each hand.

- Removes suits from card to get the card value
- For any picture cards, convert to its value
- Logic of handling instances of Aces in a hand


```python
def calculate_score(player_or_dealer_hand):
    '''This function calculates the scores of the cards from a hand'''

    score = 0
    cards = []
    # Remove suits and put it into a cards list for calculation
    for card in player_or_dealer_hand:
        cards.append(card[1:])
    # Converts any picture cards to a score of 10 and Ace to 11 using the cards dictionary
    for card in cards:
        score += cards_dict[card]
    # If Ace card/s are found and score is over 21, count Ace as 1 instead of 11
    aces = cards.count('A')
    for i in range(aces):
        if score > 21 and i == 1:
            score -= 10
        elif score > 21 and i > 1:
            score -= 1
        else:
            break

    # After some bug fixing, this if/elif statements are required to check the hand for counts of aces in a card
    if score > 21 and aces == 2:
        score -= 10
    elif score > 21 and aces == 3:
        score -= 20
    elif score > 21 and aces == 4:
        score -= 30

    return score
```

Player

```
# Initialise empty player hand list, requires this to be global as it is accessed various times throughout the program
player_hand = []
```

Global player_hand variable

This feature involves how the user interacts and plays the game, mainly the logic behind their turn where they can choose to 'hit' for another card or 'pass' to skip their turn

Conditions are checked to calculate the winning/losing potential of their hand

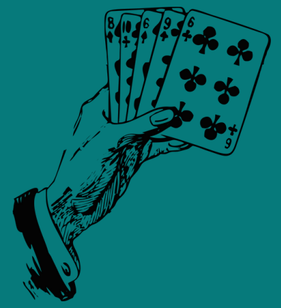
```
def player_turn():
    '''This function involves logic of players turns and play'''

    # Calculates player score from initial dealt cards
    player_score = calculate_score(player_hand)
    # Checks if score is 21, if so players turn automatically ends
    if player_score == 21:
        return

    while True:
        hit_or_pass = input(f"\nDo you want to hit for another card or pass to skip your turn? Type {attr(1)}{fg(2)}'h'{attr('reset')} or {attr(1)}{fg(9)}'p'{attr('reset')}: ").lower().strip(' ')

        try:
            if hit_or_pass == 'h':
                deal_card(1, player_hand)
                # The code below repeats itself because the score has to be re-calculated for every new card drawn in the loop to check updated score
                player_score = calculate_score(player_hand)
                print(f"\n{attr(1)}{fg(21)}Your current hand:{attr('reset')} {player_hand}, {attr(1)}{fg(21)}current score:{attr('reset')} {player_score}")
                if player_score == 21:
                    return
                elif player_score > 21:
                    return
            elif hit_or_pass == 'p':
                break
            else:
                raise ValueError(f"\n{attr(1)}{fg(9)}Invalid Input! Please enter 'h' to hit for another card or 'p' to pass and skip a turn:{attr('reset')}")
        except ValueError as InvalidInput:
            print(InvalidInput)
```

Dealer



```
# Initialise empty dealer hand list, requires this to be global as it is accessed various times throughout the program
dealer_hand = []
```

Global dealer_hand variable

This feature involves how the dealer (computer) interacts and plays the game, mainly the logic behind their turn, while their score is less than 17, they keep drawing cards until a condition is met.

```
def dealer_turn():
    '''This function involves logic of dealers turns and play'''

    # The codes below to calculate score is required as it is used for conditional statements to check
    dealer_score = calculate_score(dealer_hand)
    player_score = calculate_score(player_hand)
    # Dealer doesn't have to draw another card if certain conditions are met: Player bust, dealer score higher, or dealer has BJ
    if player_score > 21 or dealer_score > player_score or dealer_score == 21:
        return
    # If dealers hand score is under 17, keep drawing cards
    while dealer_score < 17:
        card = draw_card()
        dealer_hand.append(card)
        # This code is required as the conditions checks the updated hand/score
        dealer_score = calculate_score(dealer_hand)
        # Conditions for dealer
        if dealer_score == 21: # If dealer BJ, turn ends
            return
        elif dealer_score > 21: # If dealer bust, turn ends
            return
    return
```

Determine winner

Handles all the logic to determine winner or loser

- Using if/elif statements to check each winning and losing condition for the player or dealer.
- Displays the appropriate outcome of the game

```python
def determine_winner():
    '''This function involves the logic for determining conditions of winner and displaying the final score and outcome'''

    # Sum up final player and dealer score
    player_score, dealer_score = calculate_score(player_hand), calculate_score(dealer_hand)
    # Display both players and dealers final hand and their final score
    print(f"\n{attr(1)}{fg(21)}Your final hand:{attr('reset')} {player_hand}, {attr(1)}{fg(21)}final score:{attr('reset')} {player_score}")
    print(f"{attr(1)}{fg(21)}Dealers final hand:{attr('reset')} {dealer_hand}, {attr(1)}{fg(21)}final score:{attr('reset')} {dealer_score}\n")
    # Determine condition of winner/loser based on Natural Blackjack, which means that they have a 21 from the first cards dealt, return their outcome
    if player_score == 21 and len(player_hand) == 2:
        result = "Instant win! With a natural Blackjack 🤠\n"
    elif dealer_score == 21 and len(dealer_hand) == 2:
        result = "You lose.. Dealer wins with a natural Blacjack 🤠\n"
    # Determine condition of winner/loser based on normal Blackjack, which means that they have a 21 from cards drawn throughout the game, return their outcome
    elif player_score == 21:
        result = "You won with a Blackjack! 😎\n"
    elif dealer_score == 21:
        result = "You lose.. Dealer wins with a Blackjack 😤\n"
    # Determine all other possible winning/losing conditions, return their outcome
    elif player_score > 21:
        result = "Bust! You lost.. 😩\n"
    elif dealer_score > 21:
        result = "Dealer bust! Lucky you, you win! 😅\n"
    elif player_score > dealer_score:
        result = "You're a winner! You have a higher score than the Dealer 🥳\n"
    elif player_score < dealer_score:
        result = "You lost... The dealer has a higher score 🙃\n"
    else:
        result = "It's a draw! 😲\n"

    print(result)
    return result
```

Main Function block

The play_blackjack function serves as the main flow of the game, it includes all other parts of the program all added to create a cohesive whole.

Main guard of the program

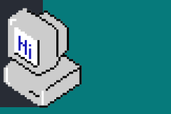
```python
if __name__ == '__main__':
    welcome()
    create_csv_file()
    init_deck()
    main_menu()
```

```python
def play_blackjack():
    '''This functions serves as the main function body of the program that includes other parts of the program.
    All combined together to serves as the main flow which runs to play the Blackjack program'''

    try:
        # Starts the game on a clean slate, clearing the player and dealers hand and clears screen
        clean_slate()
        print(main_logo)
        # Initially deals 2 cards for player to start the game and 1 card for dealer
        init_deal()
        # Prints first hand of player and dealer along with their score
        print_first_hand_and_score()
        # Start of players turn
        player_turn()
        # Start of dealers turn
        dealer_turn()
        # Determines winner and displays outcome of the game
        winner = determine_winner()
        # Calculates score for final hand
        player_score, dealer_score = calculate_score(player_hand), calculate_score(dealer_hand)
        # Writes game results to CSV
        write_game_history(player_hand, player_score, dealer_hand, dealer_score, winner)

        while True:
            try:
                play = input(f"Continue playing a game of Blackjack? Type {fg(2)}{attr('bold')}'y'{attr
                ('reset')} to continue or {fg(196)}{attr('bold')}'b'{attr('reset')} to go back to main menu: ").
                lower().strip(' ')

                match play:
                    case 'y':
                        # Recursively starts the game for another round
                        play_blackjack()
                        break
                    case 'b':
                        # Exits loop and function to go back to main menu
                        clear()
                        break
                    case _:
                        clear()
                        raise ValueError(f"{fg(9)}{attr(1)}\nInvalid Input! Please enter 'y' or 'b':{attr
                        ('reset')}\n")
            except ValueError as InvalidInput:
                print(InvalidInput)
                enter_to_continue()
                clear()
    except KeyboardInterrupt:
        exit_game_to_menu()
```



Main Menu

The main_menu function serves as the main hub of interaction for the user to navigate throughout the program.

It is similar to the play_blackjack function where it is comprised of different parts of menu components.

```
def main_menu():
    '''This function serves as the main menu which introduces the user to the program and so the user can
    interact and navigate throughout the program'''

    try:
        clear()
        user_choice = ''

        while user_choice != '6':

            print(main_logo)
            print(f"{attr(1)}Welcome to {fg(196)}Blackjack ♠{attr('reset')}  {attr(1)}Please choose from the
            following options:{attr('reset')}\n")
            user_choice = menu()

            try:
                match user_choice:
                    case '1':
                        # Starts the game or to go for another round after navigating menu
                        # clean_slate()
                        play_blackjack()
                    case '2':
                        # This will get you to view the instructions for the game, leaving the menu
                        instructions()
                    case '3':
                        # This will get you to view the house rules of the game, leaving the menu
                        rules()
                    case '4':
                        # This will get you to view the history and score for the game, leaving the menu
                        history()
                    case '5':
                        # This will let the user decide if they want to wipe out the game history
                        wipe_history()
                    case '6':
                        # Exits the program entirely
                        clear()
                        print(thanks_logo)
                        break
                    case _:
                        clear()
                        raise ValueError(f"{fg(196)}{attr(1)}Invalid Input! Please enter a numbered option from
                        the menu (1-6):{attr('reset')}\n")
            except ValueError as InvalidInput:
                print("\n" + str(InvalidInput))
                enter_to_continue()
                clear()
    except KeyboardInterrupt:
        exit_game()
```

Menu Components

Part 1

Displays menu selection for user to choose

```python
def menu():
    '''The menu function displays all the list of menu items for the user to select from'''

    print(f"{attr(1)}{fg(196)}1. Start playing Blackjack")
    print(f"{attr(1)}{fg(27)}2. Read instructions")
    print(f"{attr(1)}{fg(27)}3. View house rules")
    print(f"{attr(1)}{fg(27)}4. Check game history")
    print(f"{attr(1)}{fg(3)}5. Wipe game history")
    print(f"{attr(1)}{fg(196)}6. Exit Blackjack{attr('reset')}\n")

    # Prompt user for their selection and return selection
    choice = input(f"{attr(1)}Choose a numbered option from the menu: {attr('reset')}").strip(' ')
    return choice
```

```python
def back_to_menu():
    '''A function that allows the user to navigate back to the main menu'''

    while True:
        back = input(f"Press {fg(9)}{attr(1)}'b'{attr('reset')} to go back to the Main Menu: ").lower().strip('
        ')

        try:
            match back:
                    case 'b':
                        clear()
                        return
                    case _:
                        raise ValueError(f"\n{attr(1)}{fg(9)}Invalid Input! Please type 'b':{attr('reset')}\n")
        except ValueError as InvalidInput:
            print(InvalidInput)
```

A function so the user can go back to the main menu

Menu Components

Part 2

Displays to the user so they are able to view the instructions of the Blackjack program

```
def instructions():
    '''This function allows the user to view the instructions for the game'''

    clear()
    print(instruction_logo)

    print(f"""
    1. The premise of the game is simple, get as close to 21 as possible without going over.
    2. As the player you will be dealt 2 cards, the dealer will only reveal and present 1 card.
    3. As the player, you can decide to 'hit' for another card or 'pass' to skip a turn.
    4. If you get a Natural Blackjack (An Ace and a picture card) you instantly win the game.
    5. If the dealer gets a Natural Blackjack, you instantly lose the game.
    6. The game will end if either the players or dealers final hand score is over 21.
    7. The game will end if either player or dealers final hand score is less than each others.
    8. You are able to view the game history from the menu to see previous wins/losses.
    9. You are also able to wipe the game history to start fresh if you wish!

    Most importantly have fun, since theres no money involved, no pressure! 😉

    {attr(1)}{fg(9)}NOTE: You can quit the game at any time using the command ctrl + c. Depending on where
          you are in the game, you will either be brought back to the main menu or the game
          will quit.{attr('reset')}
    """)

    back_to_menu()
```

Menu Components

Part 3

Displays to the user so they are able to view the house rules of the Blackjack program

```
def rules():
    '''This functions allows the user to view the house rules of the game'''

    clear()
    print(rules_logo)
    print("""
    The deck size is the original 52 cards including suits.
    The Jack/Queen/King all count as 10.
    The Ace can count as 11 or 1.
    There are no jokers.
    The deck of cards will be shuffled before starting a game.
    Cards are removed from the deck as they are drawn.
    The dealer is the computer.
    """)

    back_to_menu()
```

Menu Components

Part 4

```
def history():
    '''This functions allows the user to view the history and score of games played, called from
    view_game_history() function'''

    clear()
    print(history_logo)
    view_game_history()
    print("\n")
    back_to_menu()
```

Comprised of the view_game_history function itself which allows users to view the games history

Comprised of the wipe_game_history function itself which allows users to wipe the games history for a fresh start

```
def wipe_history():
    '''This function allows the user to decide to wipe and delete the game history, called from the
    wipe_game_history() function'''

    clear()
    wipe_game_history()
    back_to_menu()
```

Game History

```
game_history = 'game_history.csv'
```

Saved CSV filename as variable

create_csv_file

```
def create_csv_file():
    '''This function creates the CSV file to store game history and scores.'''

    try:
        if not os.path.isfile(game_history):
            with open(game_history, 'a', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(['Game', 'Player Hand', 'P. Score', 'Dealer Hand', 'D. Score', 'Result'])
    except FileNotFoundError:
        print("Game history file not Found.")
    except PermissionError:
        print("Permission denied. Please make sure you have permission to create a file in the specified
        directory.")
```

Allows automation to create a CSV file to store and record the games history

count_games_played

```
def count_games_played():
    '''This function will count the total number of games played by checking the number of rows written in the
    CSV file'''

    try:
        with open(game_history, 'r') as file:
            reader = csv.reader(file)
            return sum(1 for row in reader)
    except FileNotFoundError:
        return 1
```

Counts the number of games played by checking the number of rows written in the CSV file

Game History

write_game_history

```python
def write_game_history(player_hand, player_score ,dealer_hand, dealer_score ,winner):
    '''This function will write and record the outcome and history of each game played to the CSV file'''

    game_num = count_games_played()

    try:
        with open(game_history, 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([game_num, player_hand, player_score, dealer_hand, dealer_score, winner])
    except IOError:
        print("An error occurred while writing to the file.")
```

Allows for the recording of each games played written to the CSV file

view_game_history

```python
def view_game_history():
    '''This function will allow the user to view the game history which is recorded and written in the CSV, it
    also implements prettytable for a better viewing experience'''

    try:
        with open(game_history, 'r') as file:
            reader = csv.reader(file)
            headers = next(reader)
            table = PrettyTable(headers)
            for row in reader:
                table.add_row(row)
            print(table)
    except FileNotFoundError:
        print("Game history file not found. Please re-start the game and play again to create a game history.")
```

Allows the user to view the games history which is record on the CSV file, used prettytable for a better viewing experience

Game History

Code

The user has the choice to completely wipe out the games history if they wish, wiping it will also instantly create a new game_history file ready to go as a blank slate.

The user is greeted with a confirmation message to continue or to cancel wiping the games history.

```
def wipe_game_history():
    '''This function allows the user to wipe out the game history from the CSV file for a fresh start'''

    while True:
        print(wipe_logo)
        choice = input(f"Are you sure you want to wipe out the game history? This action cannot be undone. Type
        {attr(1)}{fg(2)}'yes'{attr('reset')} to continue or {attr(1)}{fg(9)}'no'{attr('reset')} to cancel: ").
        lower().strip(' ')
        print("\n")

        try:
            if choice == 'yes':
                try:
                    clear()
                    print(wipe_logo)
                    os.remove(game_history)
                    if not os.path.isfile(game_history):
                        create_csv_file()
                    print("The game history has been sucessfully wiped out.\n")
                    break
                except FileNotFoundError:
                    print("Game history file not found. Please re-start the game and play again to create a game
                    history.\n")
                    break
            elif choice.lower() == 'no':
                clear()
                print(wipe_logo)
                print("Game history file wipe out has been cancelled.\n")
                break
            else:
                raise ValueError(f"{attr(1)}{fg(9)}Invalid input! Please type 'yes' to confirm or 'no' to cancel
                the wipe out.{attr('reset')}")
        except ValueError as InvalidInput:
            print(InvalidInput)
            print("\n")
            enter_to_continue()
            clear()
            continue
    return
```

Game History

CSV

This is sample game history which is recorder on the CSV file.

```
Game,Player Hand,P. Score,Dealer Hand,D. Score,Result
1,"['♠A', '♥10']",21,"['♥8', '♠8', '♦K']",26,"Instant win! With a natural Blackjack 🤠
"
2,"['♠Q', '♥2', '♦6']",18,"['♦8', '♠2', '♦10']",20,"You lost... The dealer has a higher score 🙃
"
3,"['♠9', '♦3', '♠6']",18,"['♥K', '♥6', '♠3']",19,"You lost... The dealer has a higher score 🙃
"
4,"['♠J', '♥7']",17,"['♥3', '♦J', '♦7']",20,"You lost... The dealer has a higher score 🙃
"
5,"['♥A', '♦Q']",21,"['♠5', '♣7', '♣4', '♠10']",26,"Instant win! With a natural Blackjack 🤠
"
6,"['♥5', '♥4', '♥Q']",19,"['♦2', '♣A', '♣6']",19,"It's a draw! 😢
"
7,"['♥9', '♣J']",19,"['♠7', '♣9', '♥J']",26,"Dealer bust! Lucky you, you win! 😅
"
```

Result

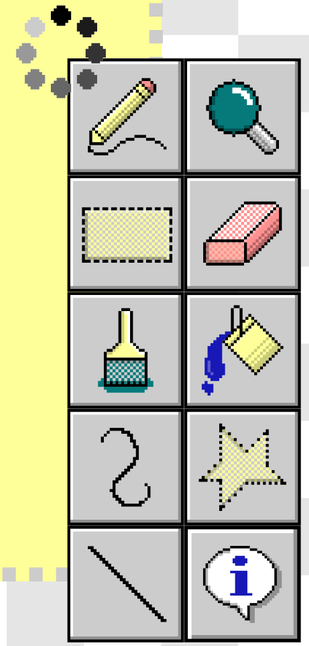
Game	Player Hand	P. Score	Dealer Hand	D. Score	Result
1	['♠A', '♥10']	21	['♥8', '♠8', '♦K']	26	Instant win! With a natural Blackjack 🤠
2	['♠Q', '♥2', '♦6']	18	['♦8', '♠2', '♦10']	20	You lost... The dealer has a higher score 🙃
3	['♠9', '♦3', '♠6']	18	['♥K', '♥6', '♠3']	19	You lost... The dealer has a higher score 🙃
4	['♠J', '♥7']	17	['♥3', '♦J', '♦7']	20	You lost... The dealer has a higher score 🙃
5	['♥A', '♦Q']	21	['♠5', '♣7', '♣4', '♠10']	26	Instant win! With a natural Blackjack 🤠
6	['♥5', '♥4', '♥Q']	19	['♦2', '♣A', '♣6']	19	It's a draw! 😢
7	['♥9', '♣J']	19	['♠7', '♣9', '♥J']	26	Dealer bust! Lucky you, you win! 😅

Press 'b' to go back to the Main Menu:

The result of the games history where the user is able to view from reading the CSV file and converting it to a better viewing experience using prettytable.

Review

Back to Agenda Page



Project Review

Ethical issues

- Accessibility of text size in terminal
- Copyright and IP issues with Blackjack
- Accessibility of possibly adding different language options for the program

Challenges

- Creating relevant tests for the program (pytest)
- Planning took the longest amount of time
- Figuring out logic of the game and how to translate that into code

Favorite parts

- For some reason I had so much fun debugging!
- Creating a smooth experience for user to navigate the program
- Overall experience was great and learned a lot and now I can say I made a game application!

11:11PM


Thank you!
T1A3 - Jordan Benjamin