Project 3 - Blackjack

Microcontrollers and Embedded Applications

CPE 316 – 01

Nathan Ho

Spring Quarter 2023

June 9th, 2023

California Polytechnic State University - San Luis Obispo

Professor Hummel

Behaviour Description

My program is a Blackjack simulator game where the user can play against a dealer. It operates on the STML476RGT3 MCU with a NUCLEO-L467RG board and the entirety of the game is displayed on a terminal interface with UART transmission. The board's built-in random number generator is used to shuffle a "deck" of cards in the game. The user is first given an option to play the game and can make choices by interacting with the connected keyboard. The cards are stored as predefined C structs to keep track of their values and whether or not they are a face card.

System Specification

Specification	**Parameters**
Number of Users	1
User's Starting Money	$1000
Betting Amounts	$100, $250, $500
Max Number of Cards in a Hand	11
First Round Dealer Hand	1 Card Face Down, 1 Card Face Up
First Round Player Hand	2 Cards Face Up
User Input	Computer Keyboard
Number of Possible Cards	52 Cards in Deck Array
Winning Hand Value	21 (or Hand < 21 and Hand > Dealer Score)
MCU Clock Frequency	24 MHz
RNG Clock Frequency	48 MHz
RNG Source	STML476RGT3 MCU Built-in RNG Peripheral
Terminal Baud Rate	115200
Terminal Interface	VT100 Serial Terminal

Figure 1: System Specification for Blackjack Simulator

System Schematic

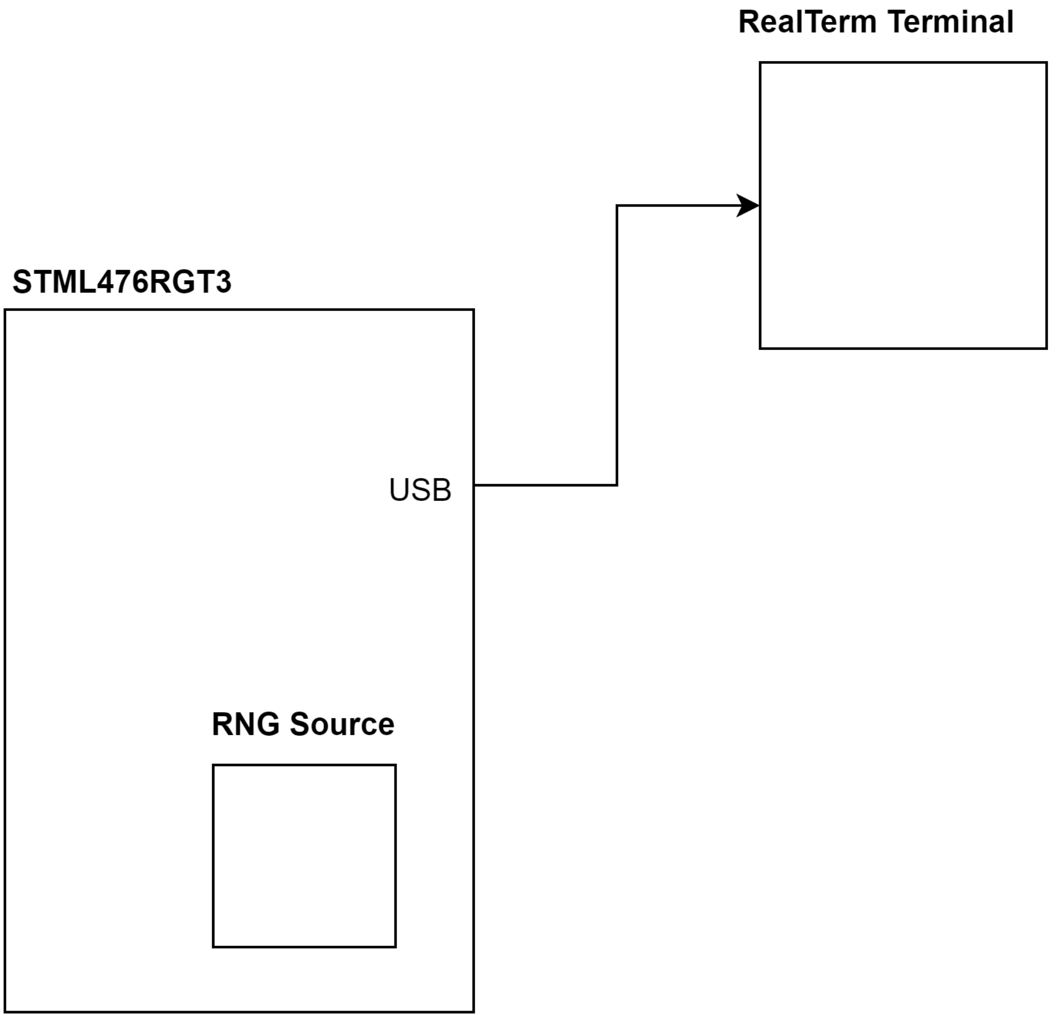

Figure 2: Blackjack Game System Schematic

Software Architecture

The bulk of this program is centered around a finite state machine that controls the entire flow of our Blackjack game. The FSM has 12 states: START_GAME, BETTING, DEAL_FIRST_CARDS, CALCULATE_HANDS, PLAYER_CHOICE, DEALER_FLIPS, DEALER_TURN, ROUND_DONE, CASH_OUT, RESET_HANDS, BUY_BACK_IN, and SHUFFLE_DECK.

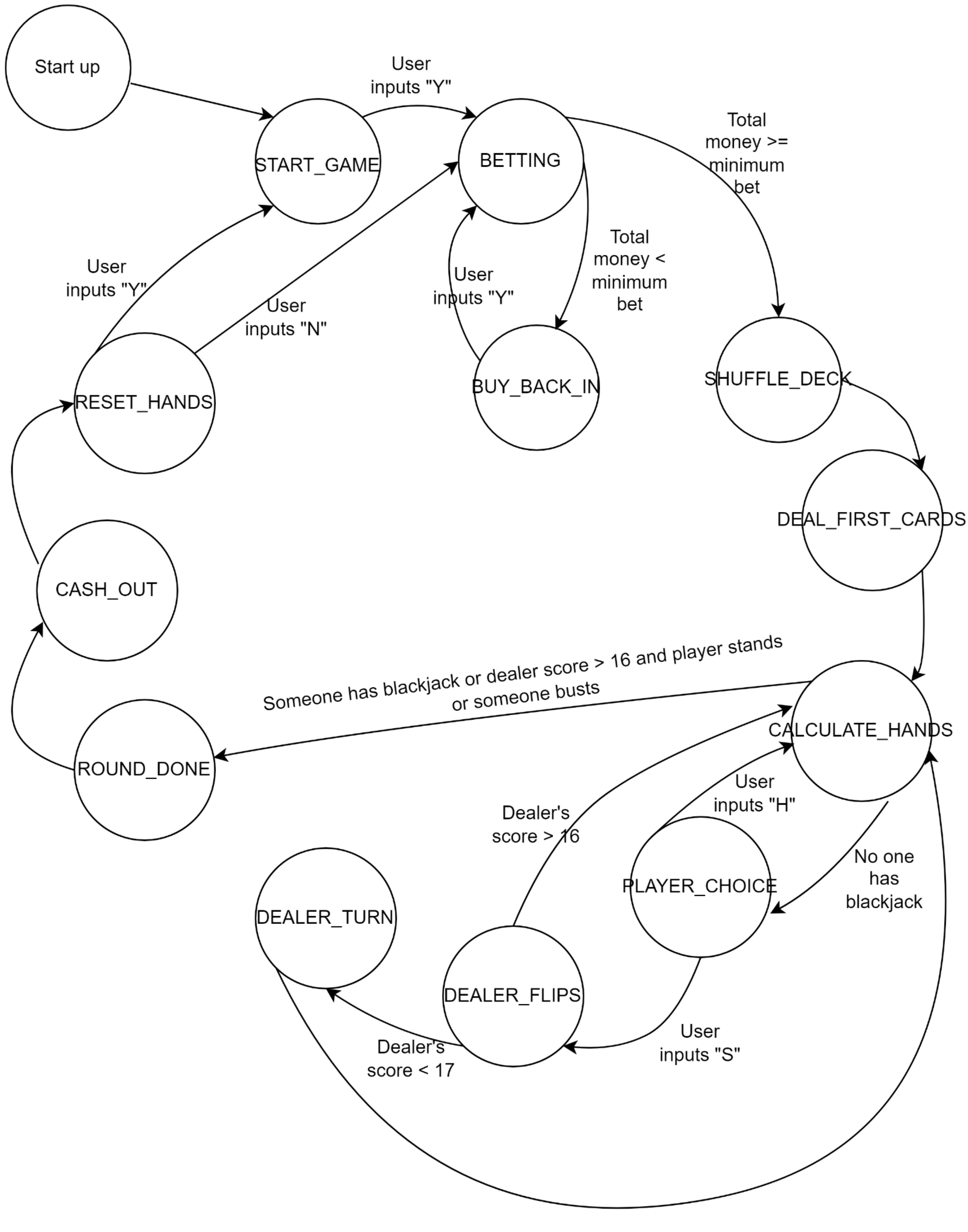

Figure 3: Finite State Machine of Blackjack Game

At startup, the user is given a choice to play the game in the START_GAME state. When the user inputs using the keyboard, the state is switched to the BETTING state where the user may choose the amount of money they are willing to bet. From there we go into the SHUFFLE_DECK state. In this state, the deck becomes initialized and then is shuffled. Whenever a round in the game has completed, we go into this state to shuffle the deck so that it is ready for the next round. The bulk of the game logic starts at the DEAL_FIRST_CARDS state where the initial cards are dealt/displayed to the terminal. The states CALCULATE_HANDS, PLAYER_CHOICE, DEALER_FLIPS, and DEALER_TURN deal with the interactive portions of the game as well as setting the win conditions. After every round, we switch to the ROUND_DONE and then the CASH_OUT states, where the user is prompted to exit the game or continue playing. The loop restarts at the BETTING state if the user decides to keep playing the game.

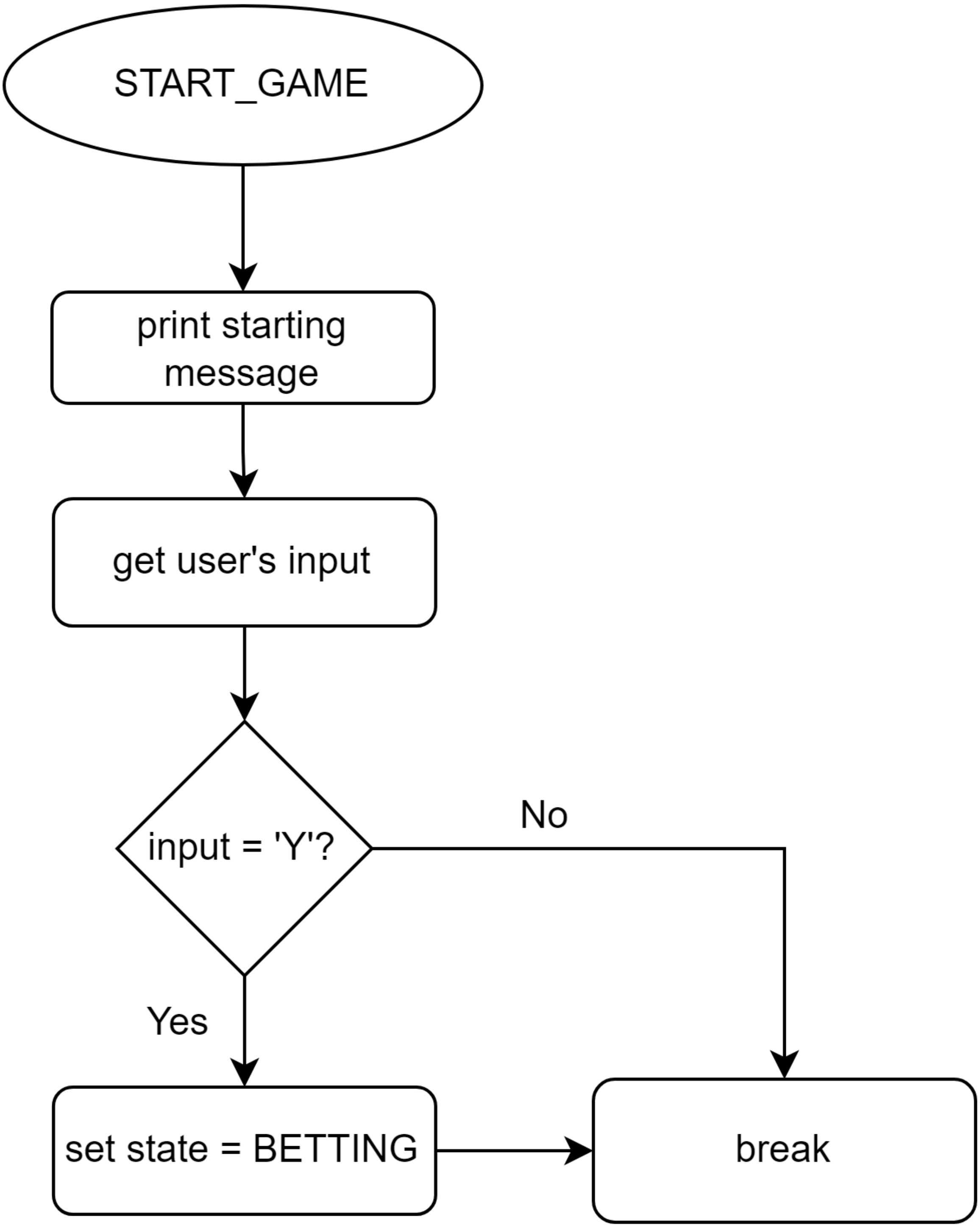

Figure 4: START_GAME State Flowchart

The START_GAME state outputs a starting game message to the terminal and then waits for the user to input using the keyboard. If the user enters in the character ‘Y’, the state is changed to the BETTING state. If any other character is pressed, no changes are made and the program waits until the user decides to play and press ‘Y’.

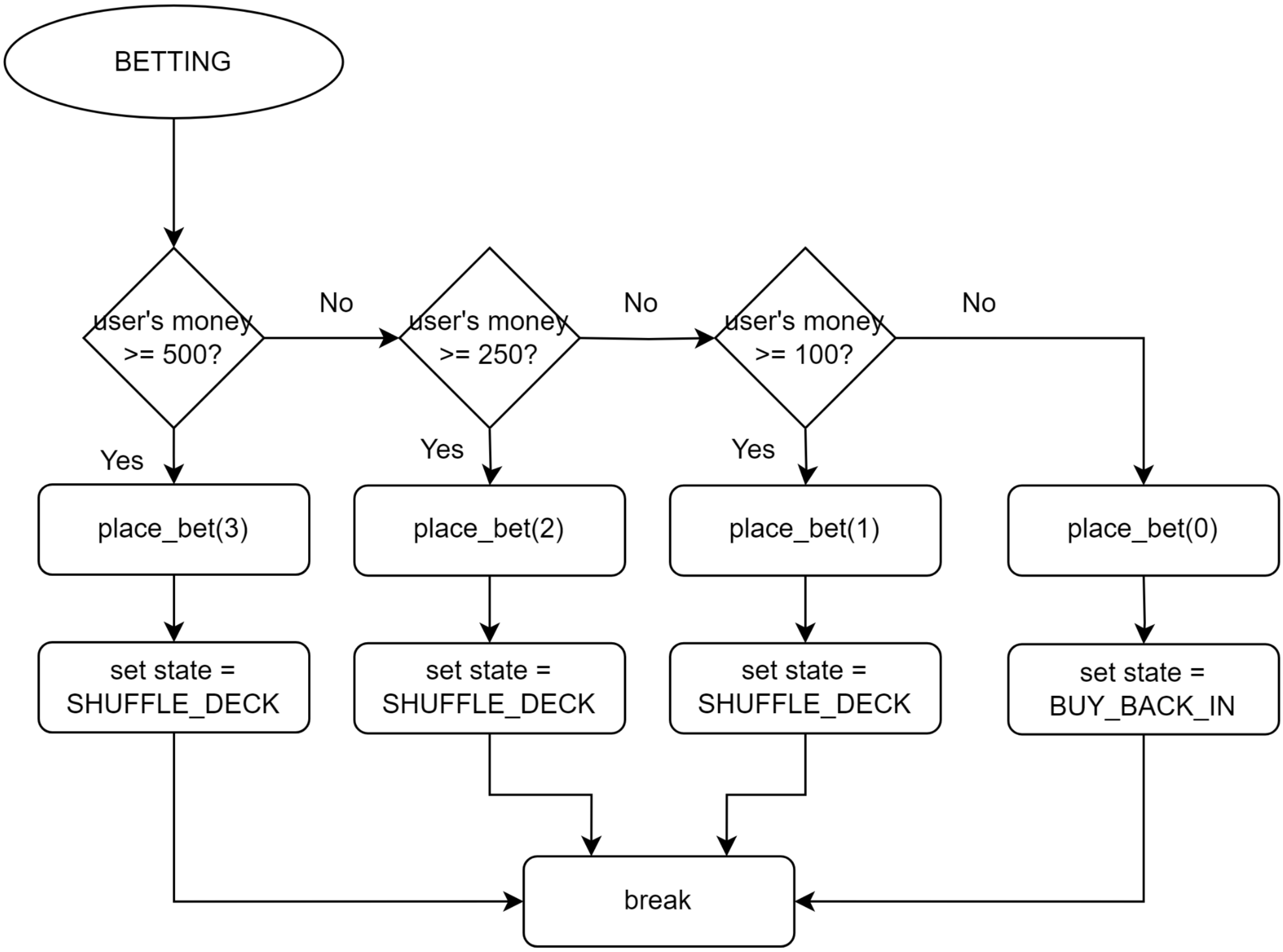

Figure 5: BETTING State Flowchart

In the BETTING state, different betting options are printed to the terminal depending on the total amount of money the user has. If the user has more than at least the minimum bet and they choose a betting option, the set is changed to the SHUFFLE_DECK state. If the user does not have more than the minimum bet, they are taken to the BUY_BACK_IN state.

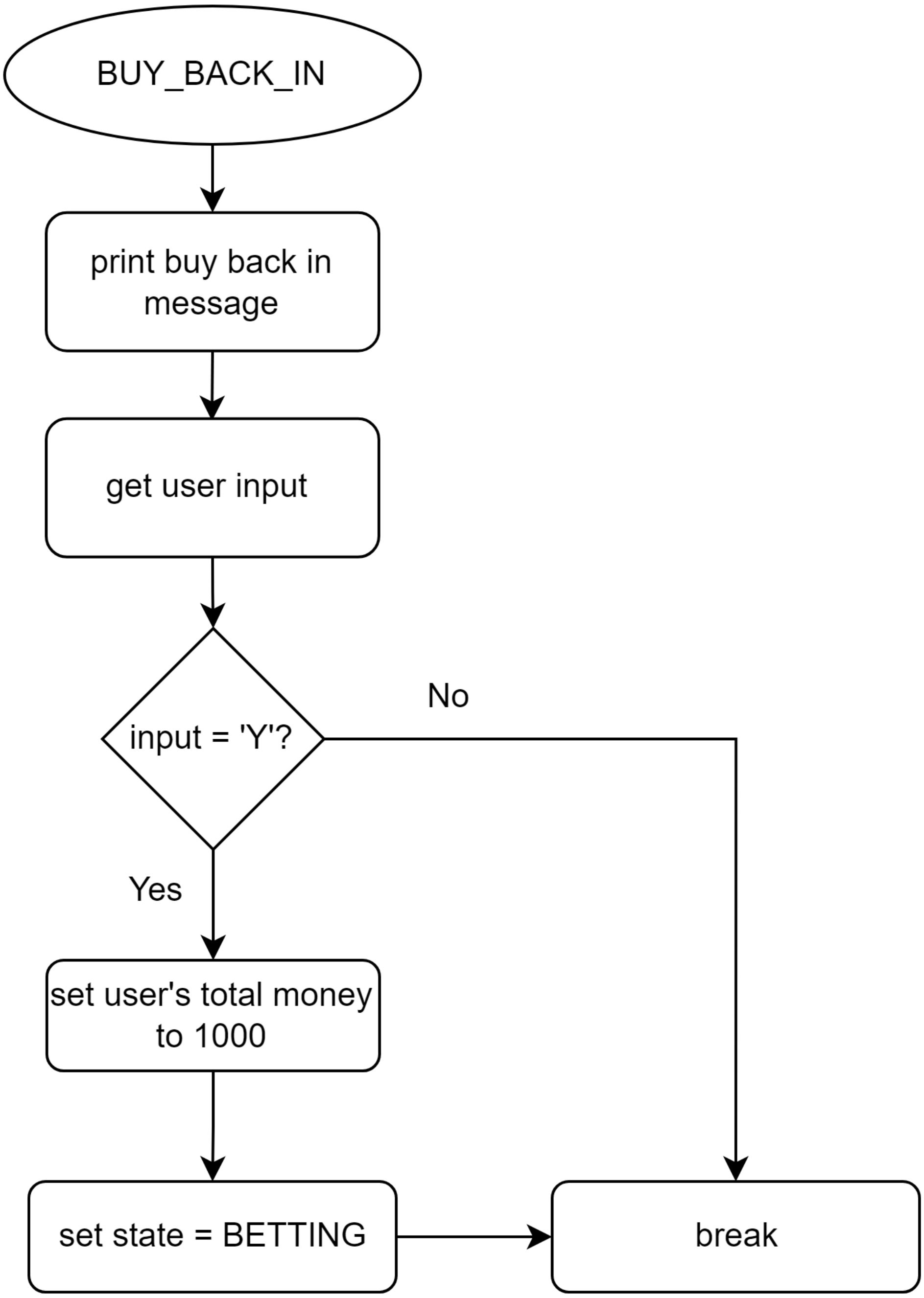

Figure 6: BUY_BACK_IN State Flowchart

In the BUY_BACK_IN state, a message is displayed to the terminal, indicating that the user does not have enough money to bet and prompting them to restart. Then the program again blocks for the user response. If the user enters ‘Y’, their total money is reset back to 1000 and the state is set back to the BETTING state.

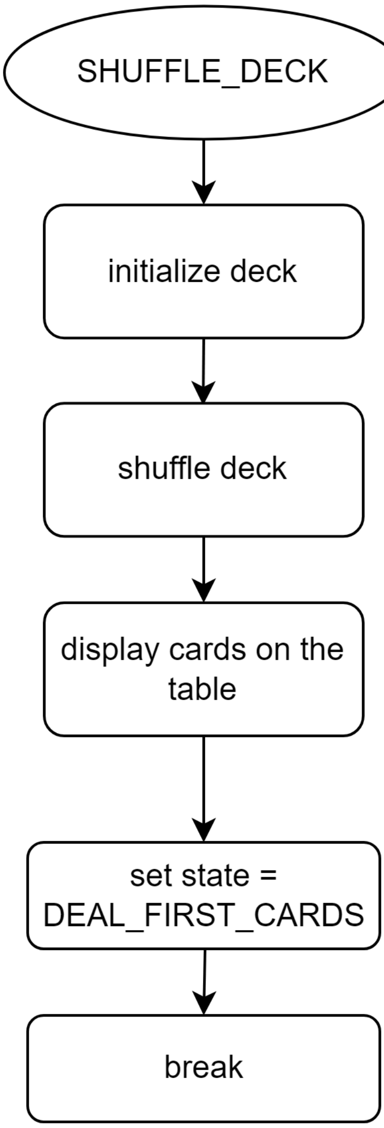

Figure 7: SHUFFLE_DECK State Flowchart

The SHUFFLE_DECK initializes the C array representing the deck of cards to have 52 cards. The deck is then “shuffled” by taking in a number produced by the RNG and swapping positions in the array at the indices of these produced numbers. We then update the table printed to the terminal and set the state to DEAL_FIRST_CARDS.

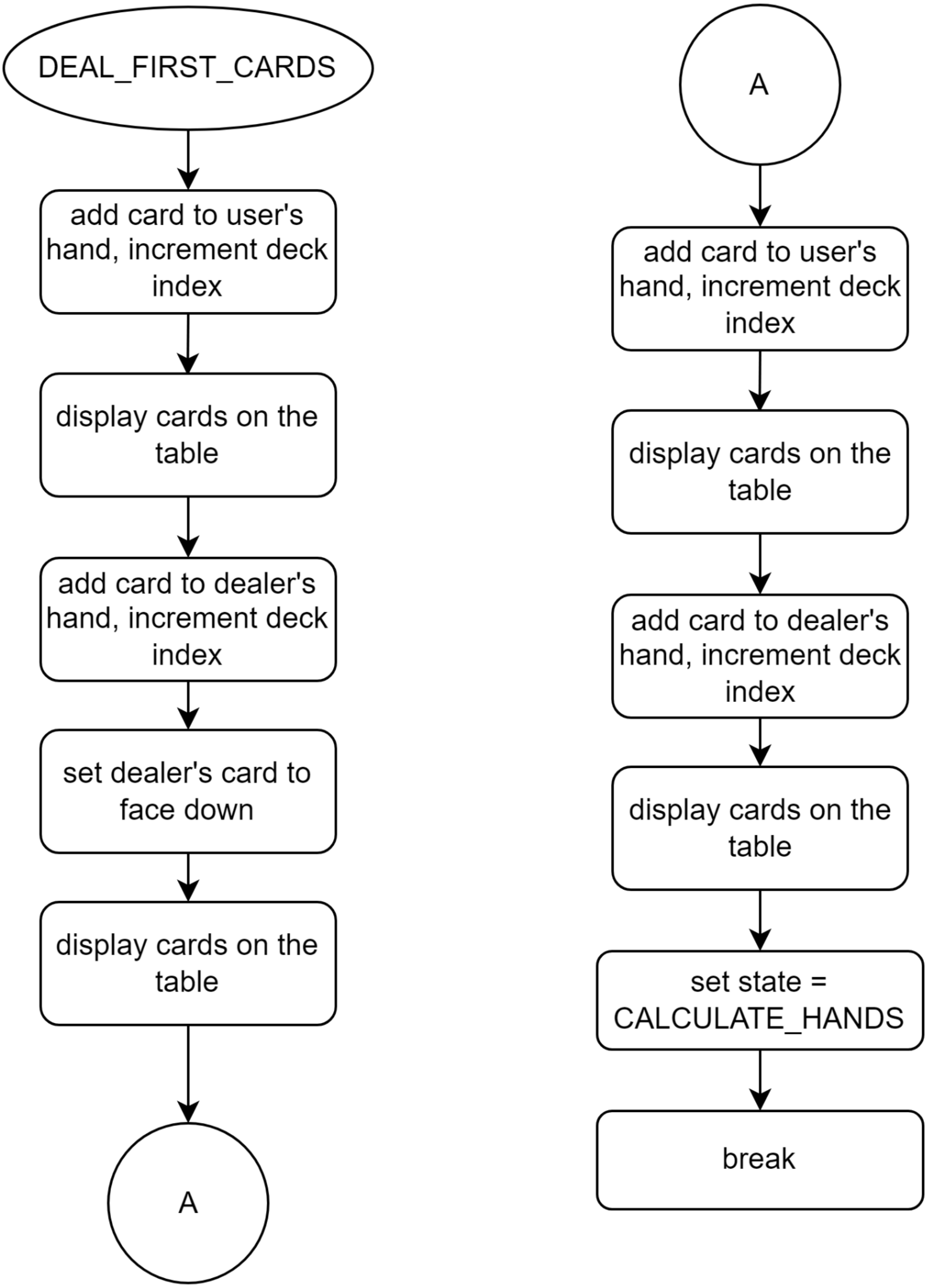

Figure 8: DEAL_FIRST_CARDS State Flowchart

The DEAL_FIRST_CARDS deals with starting the portion of the game where cards are being added to the player and the dealer's hand. A card is "drawn" from the deck by adding a card at an index to a hand and then incrementing that index by one. We then update the cards being printed to the terminal and the state is set to CALCULATE_HANDS.

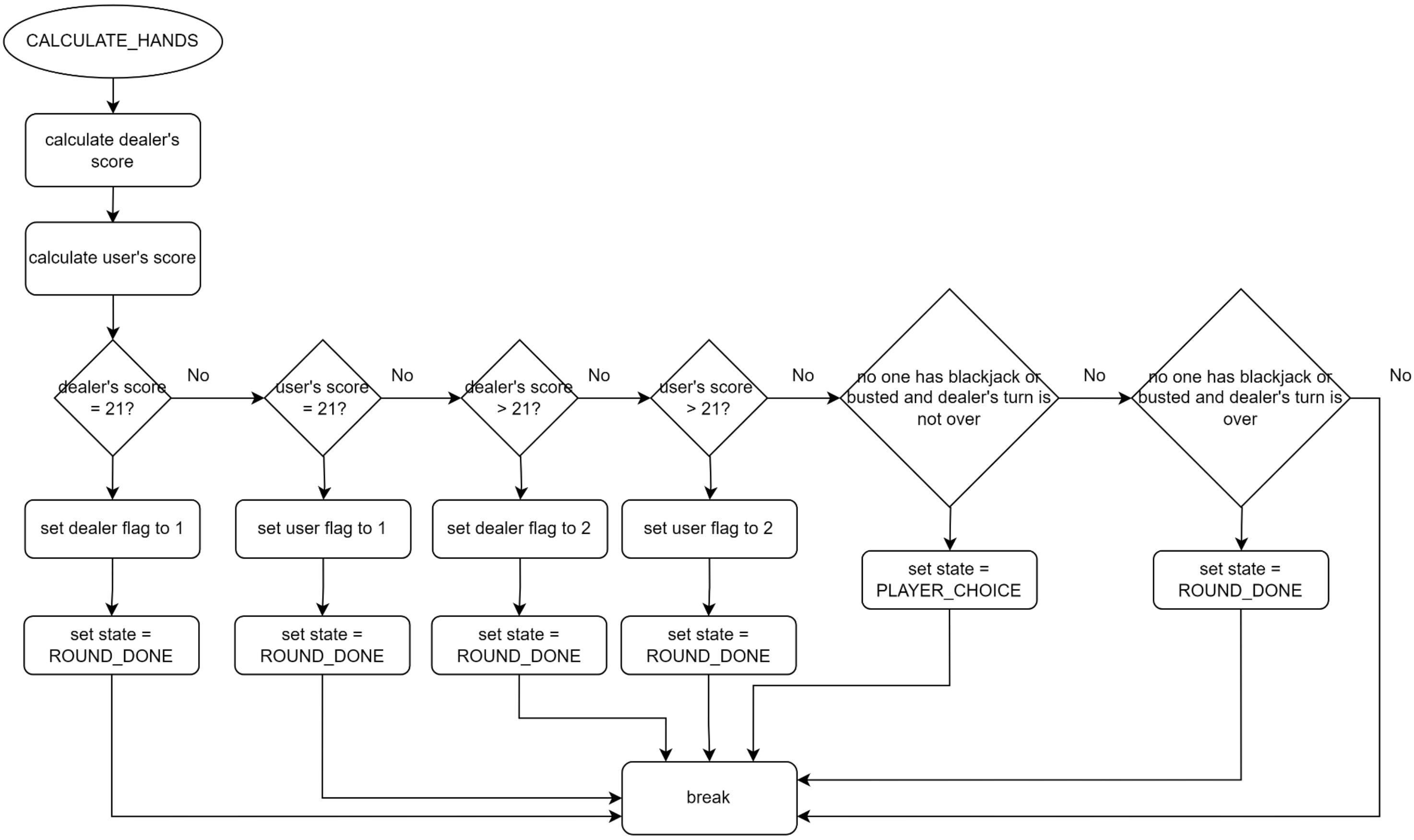

Figure 9: CALCULATE_HANDS State Flowchart

The CALCULATE_HANDS state is used for any time that we need to determine what scores the dealer and the user have. Here, we check if the dealer or the player has blackjack, they have more than 21, or if neither have more or equal to a score of 21. Global flags are utilized for the dealer and player and are set accordingly. If a win condition is fulfilled or the round is over, the state is set to ROUND_DONE. Otherwise, the state is set to PLAYER_CHOICE.

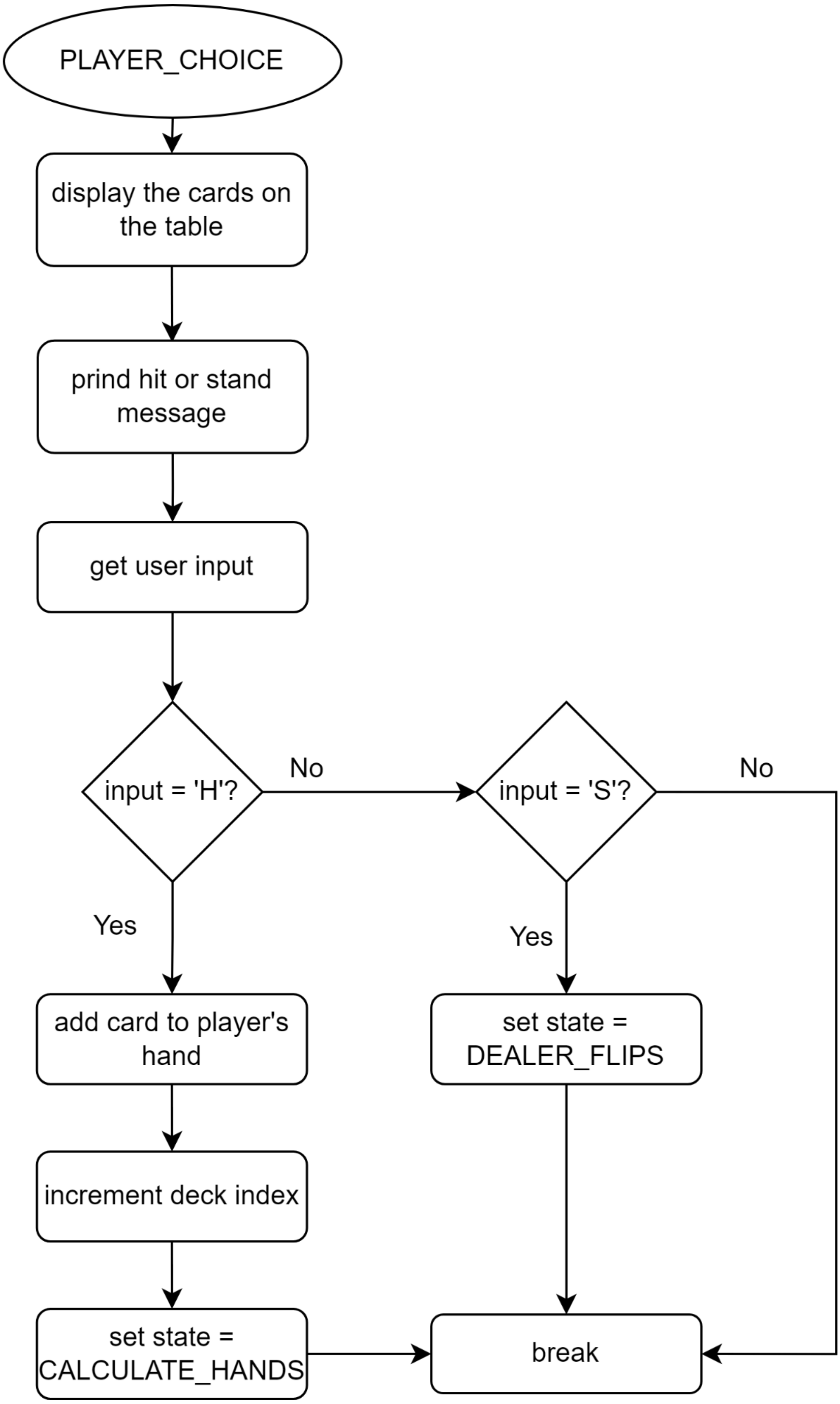

Figure 10: PLAYER_CHOICE State Flowchart

In the PLAYER_CHOICE state, the cards in each hand are printed to the terminal, as well as a prompt for the user. The user is given two options to play (hitting or standing). The program blocks until the user enters a valid character. If the user decides to hit, a card is added to their hand and the state is set back to the CALCULATE_HANDS state. If they choose not to, the state is changed to DEALER_FLIPS, which will start the dealer's turn for the round.

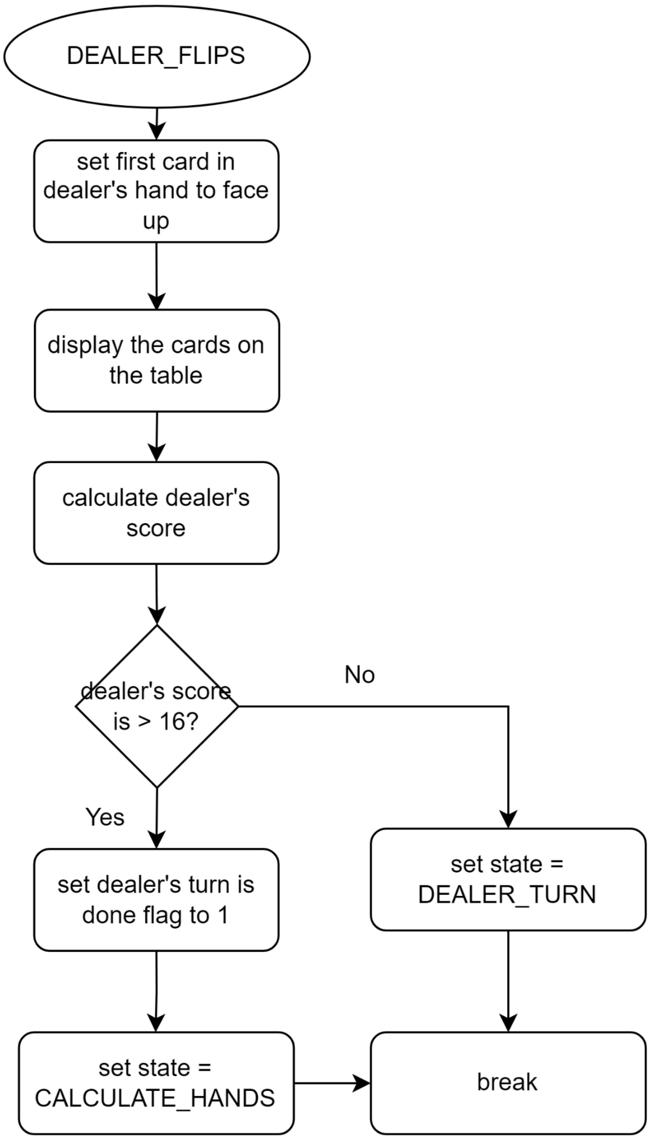

Figure 11: DEALER_FLIPS State Flowchart

In the DEALER_FLIPS state, the dealer's first card, which was originally set to be face down, is set to be face up. The cards are then updated in the terminal. The dealer's score is then calculated. If the dealer has a score greater than 16, then a flag is set indicating that the dealer's turn is over. If not, then the state is changed to DEALER_TURN.

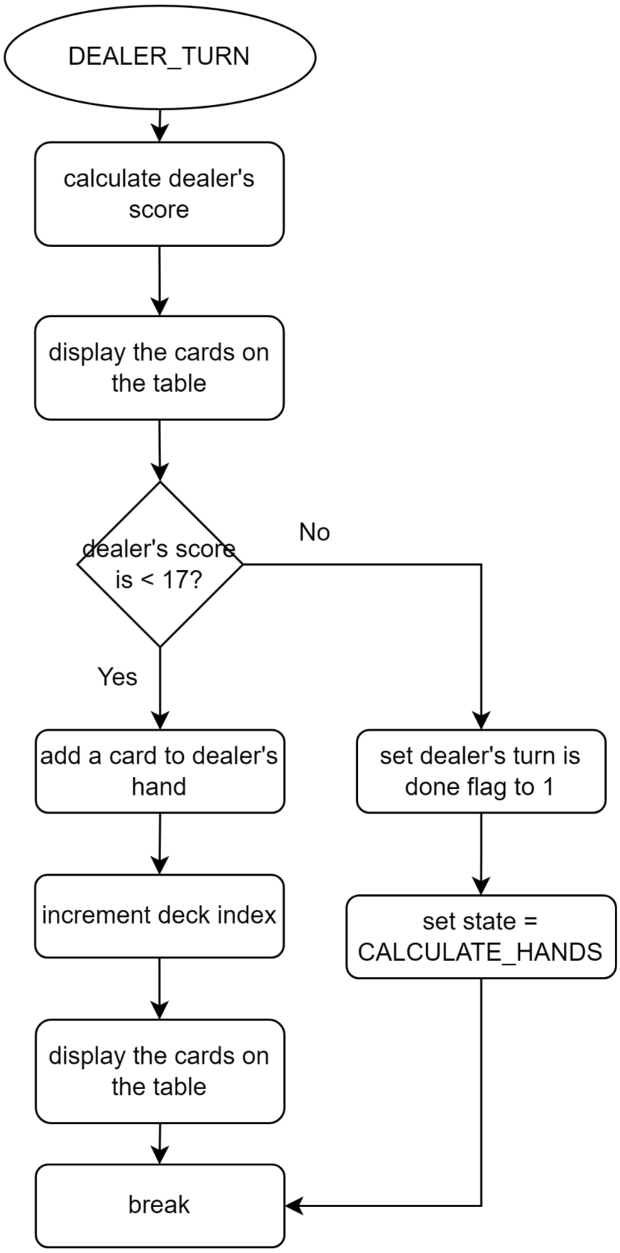

Figure 12: DEALER_TURN State Flowchart

In the DEALER_TURN state, the dealer's score is calculated. If it is not greater than 16, then cards will be added to the dealer's hand until it is. Once it is greater than 16, a flag is set indicating that the dealer has finished their turn and the state is set to the CALCULATE_HANDS state.

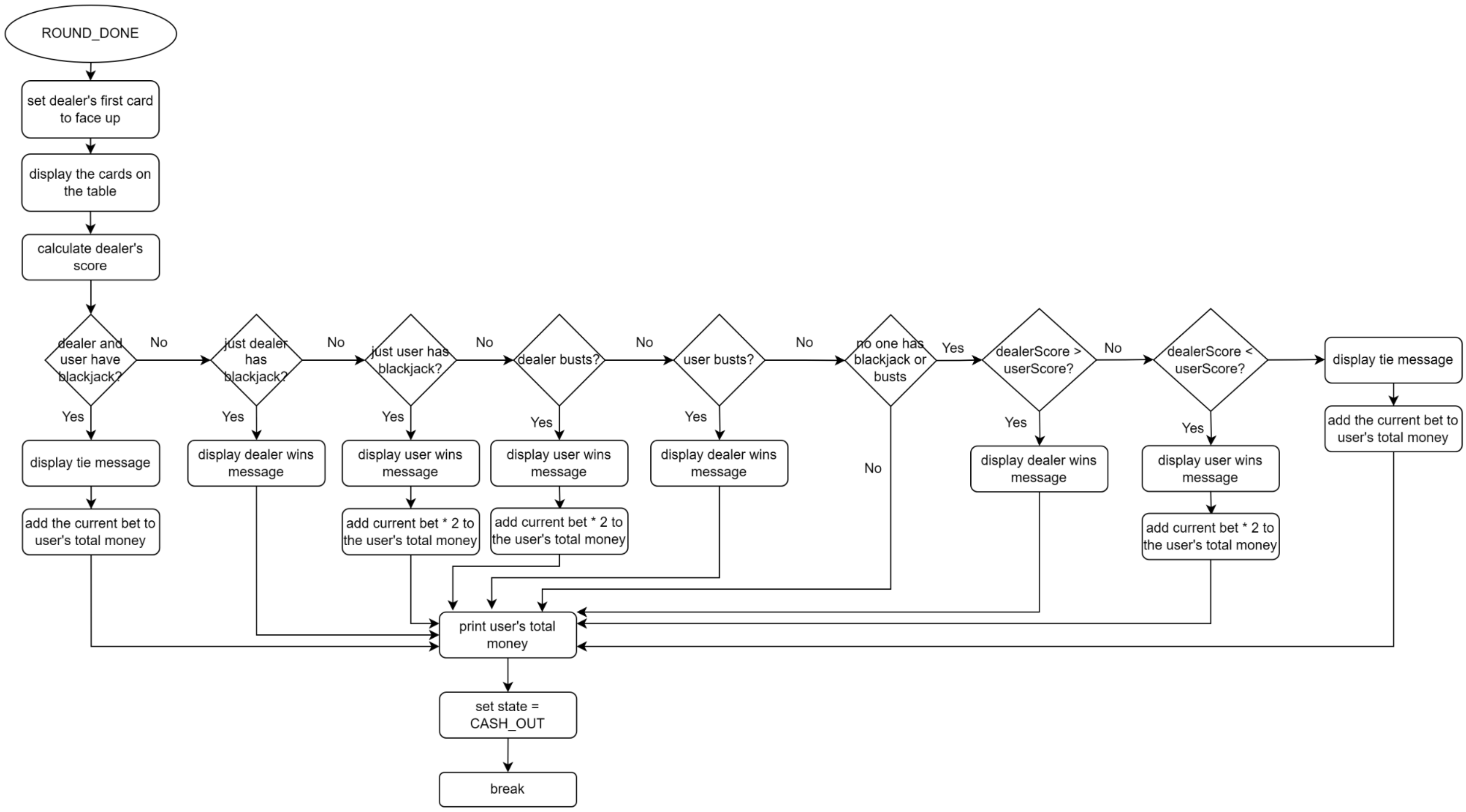

Figure 13: ROUND_DONE State Flowchart

The ROUND_DONE state deals with the win conditions calculated in the CALCULATE_HANDS state. This state determines who the winner of the round is and prints a corresponding message to the terminal. If the player wins, the amount of money that they selected in the BETTING state will be doubled and added to their total. If the player and dealer tie, then the player will receive back the same amount of money that they previously bet.

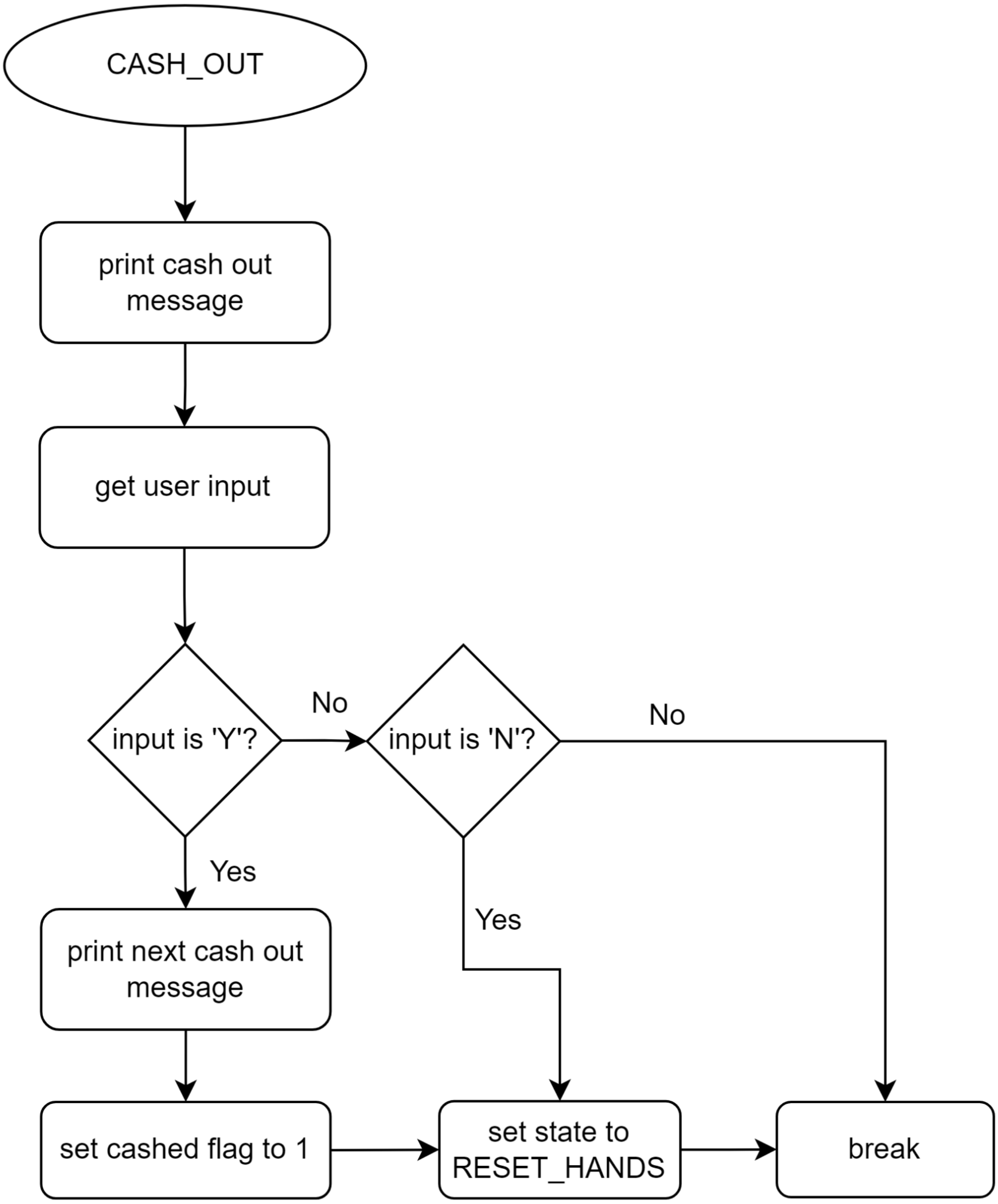

Figure 14: CASH_OUT State Flowchart

The CASH_OUT state prompts the user to either quit the game or to continue. The program blocks for the user's input. If the user chooses to quit the game, a flag is set to indicate this and a message is printed to the terminal. The state is then set to the RESET_HANDS state.

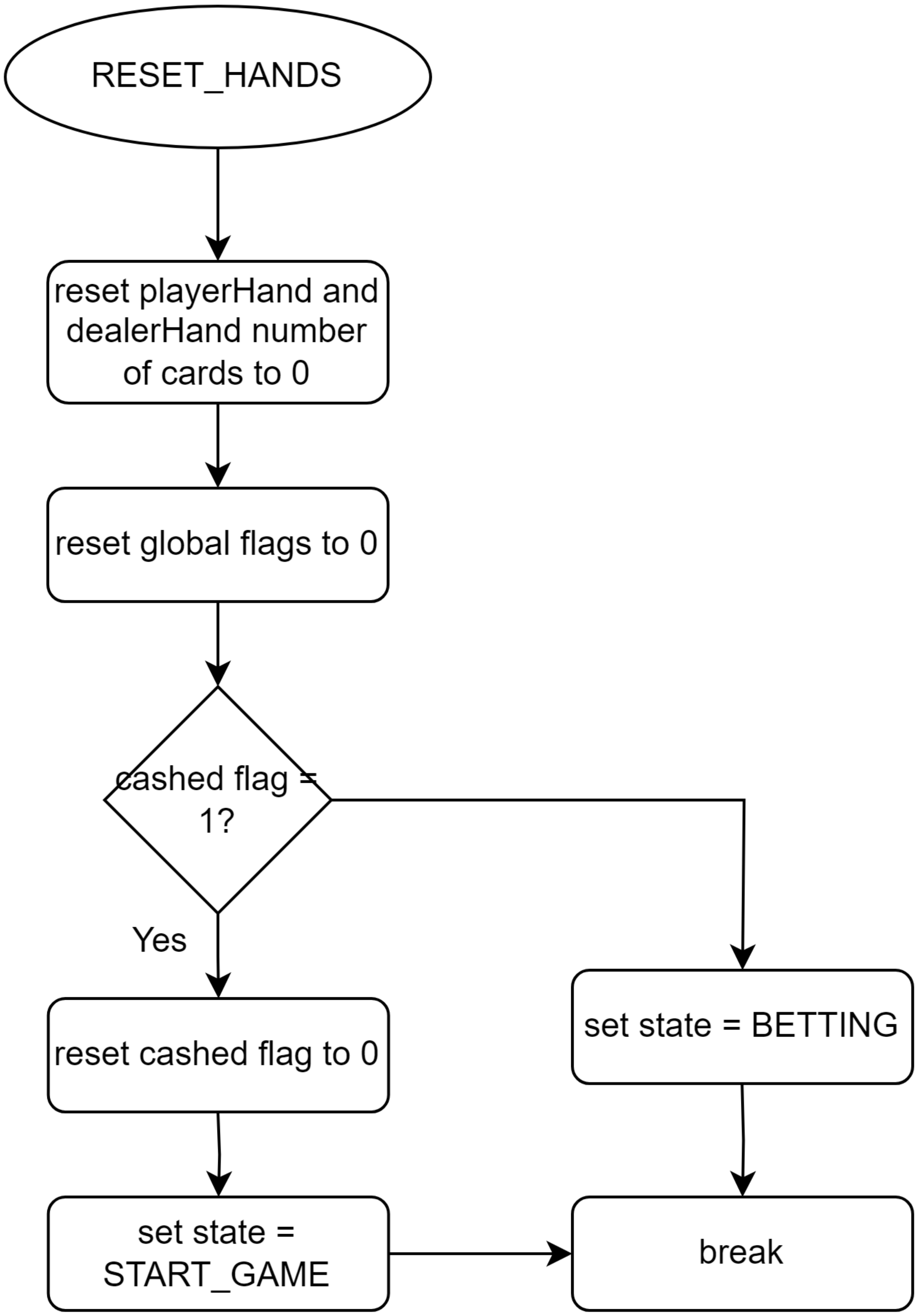

Figure 15: RESET_HANDS State Flowchart

The RESET_HANDS state resets flags to 0 and the dealer and user's hands. If the user chose to cash out in the previous state, the state will be set to the START_GAME state and the flag for this is reset. If not, they will be brought back to the BETTING state.

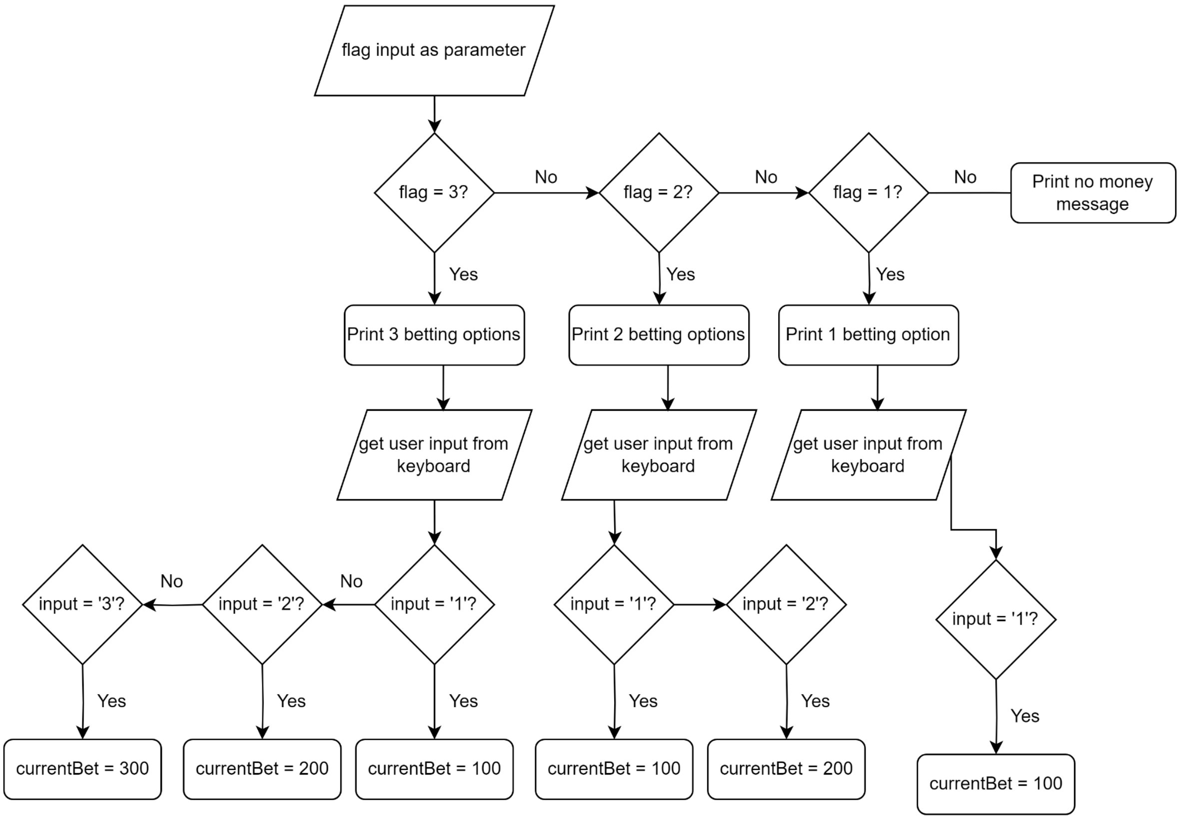

Figure 16: place_bet() Function Flowchart

This function controls the amount of betting options that will be printed to the terminal. This blocks for the user's input and sets the current bet for the round. This function is utilized in the BETTING state.

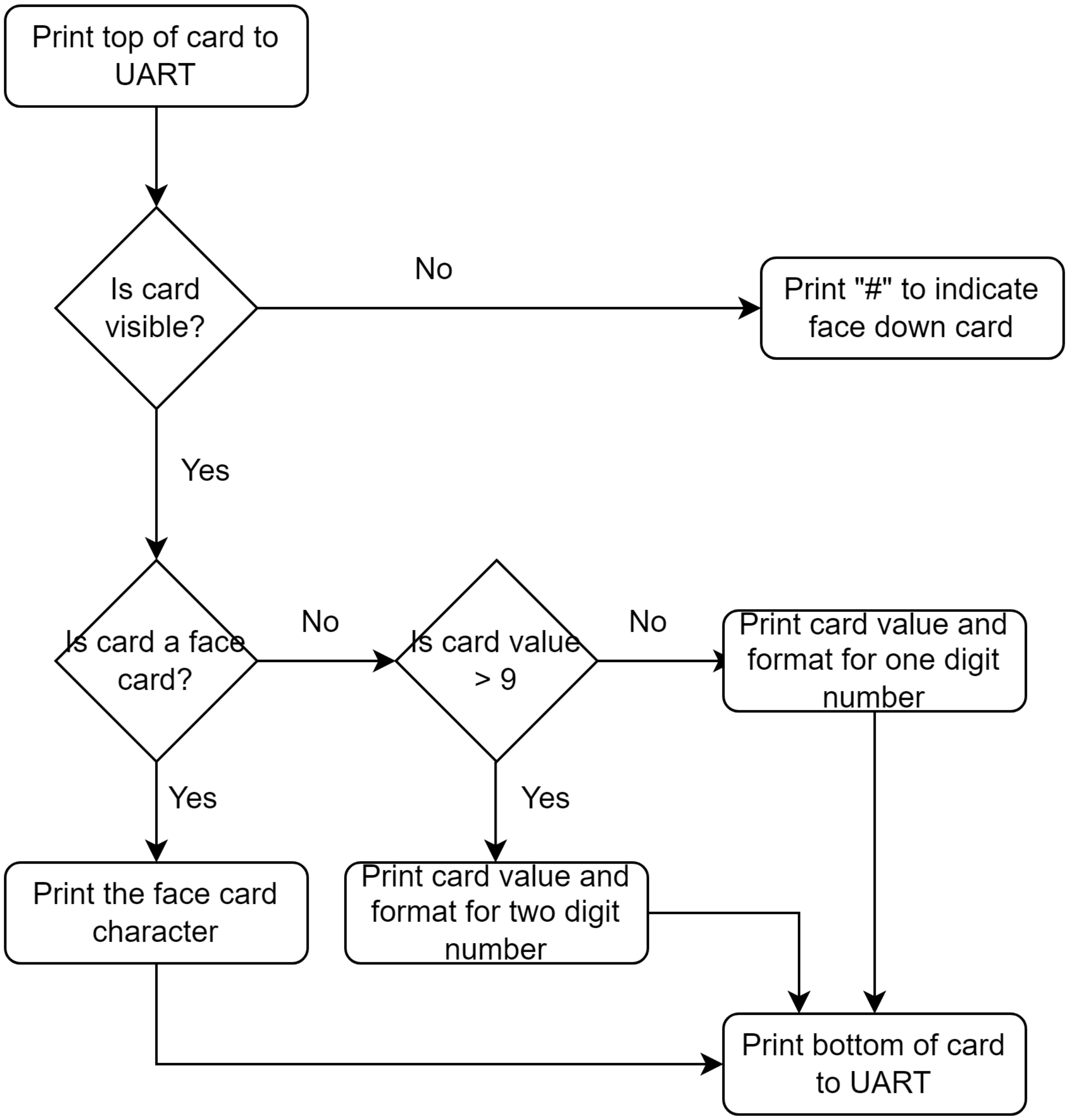

Figure 17: print_card() Function Flowchart

This function controls the formatting of printing “cards” to the terminal. If the card is a face card, the character for that face card is printed (‘J’, ‘Q’, ‘K’, ‘A’). If not then the value is printed. If the card is set as face down, then the card will be printed as multiple ‘#’ instead to indicate this on the terminal.

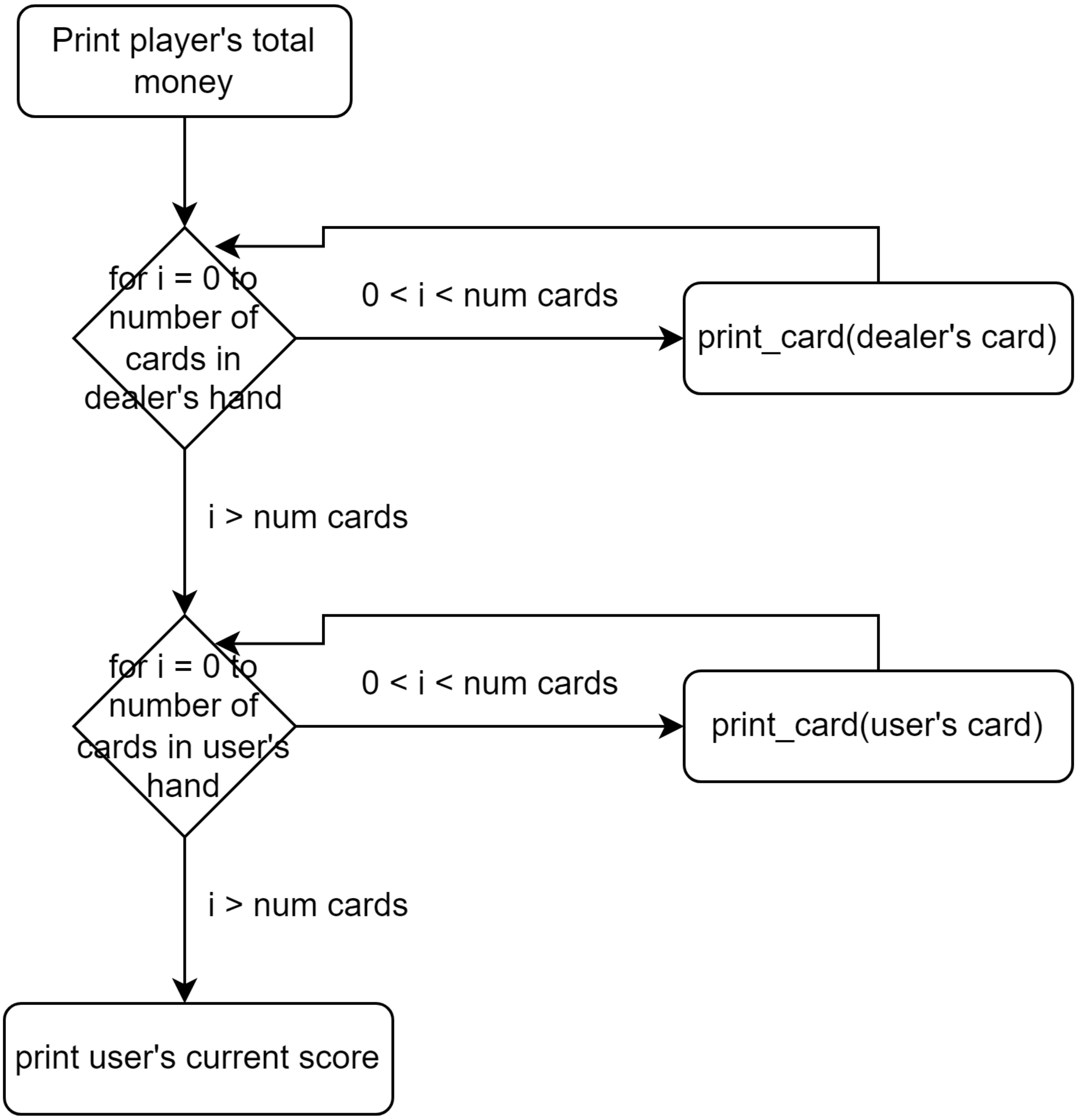

Figure 18: print_table() Function Flowchart

This function formats the entire view of the "table" that is printed to the terminal. This function utilizes the print_card() function to print each card in a specified location. VT100 escape codes are used for formatting the placements.

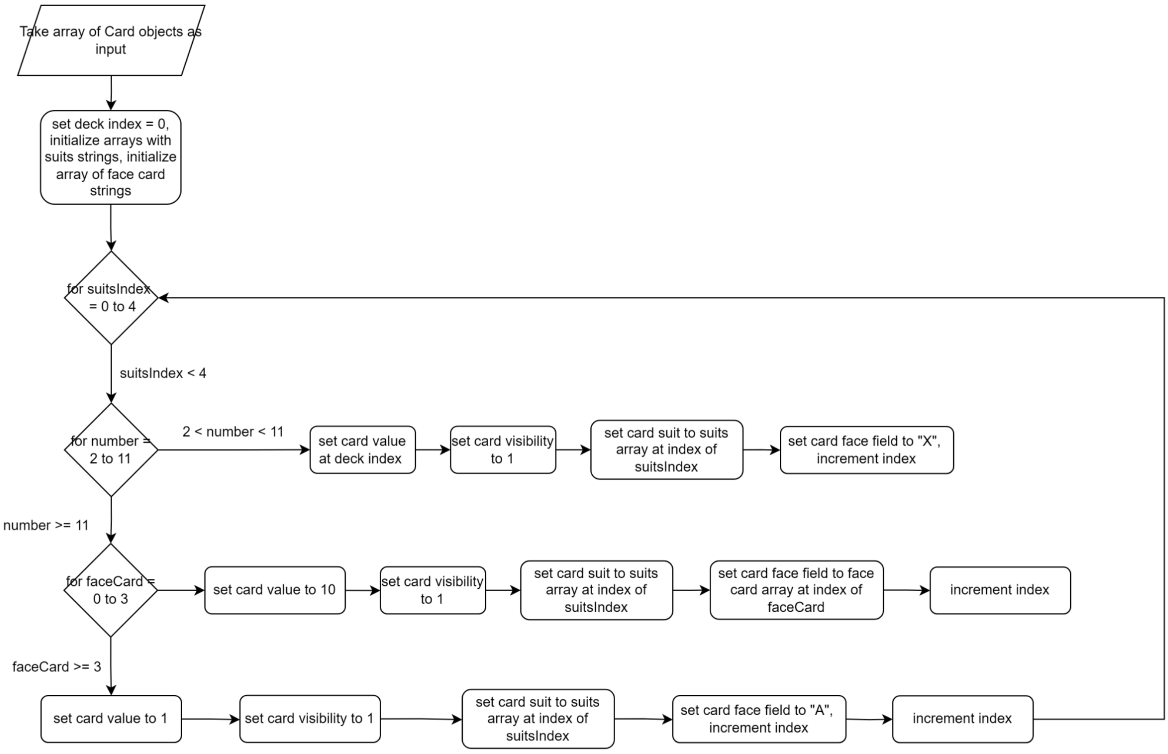

Figure 19: deck_init() Function Flowchart

This function is used in the SHUFFLE_CARDS state to initialize the deck of cards. Cards are initialized in the deck in order by value and suit.

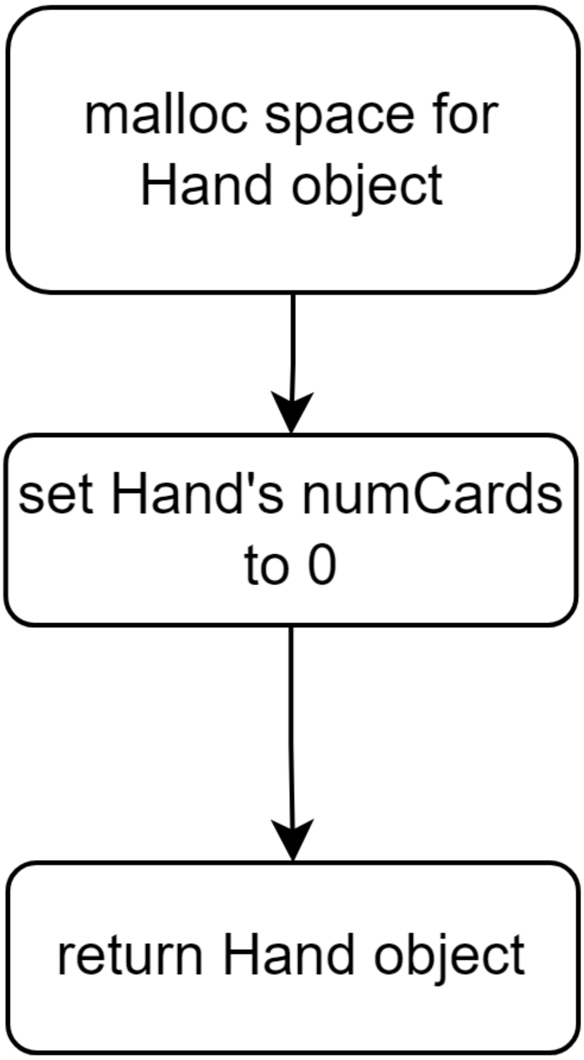

Figure 20: hand_init() Function Flowchart

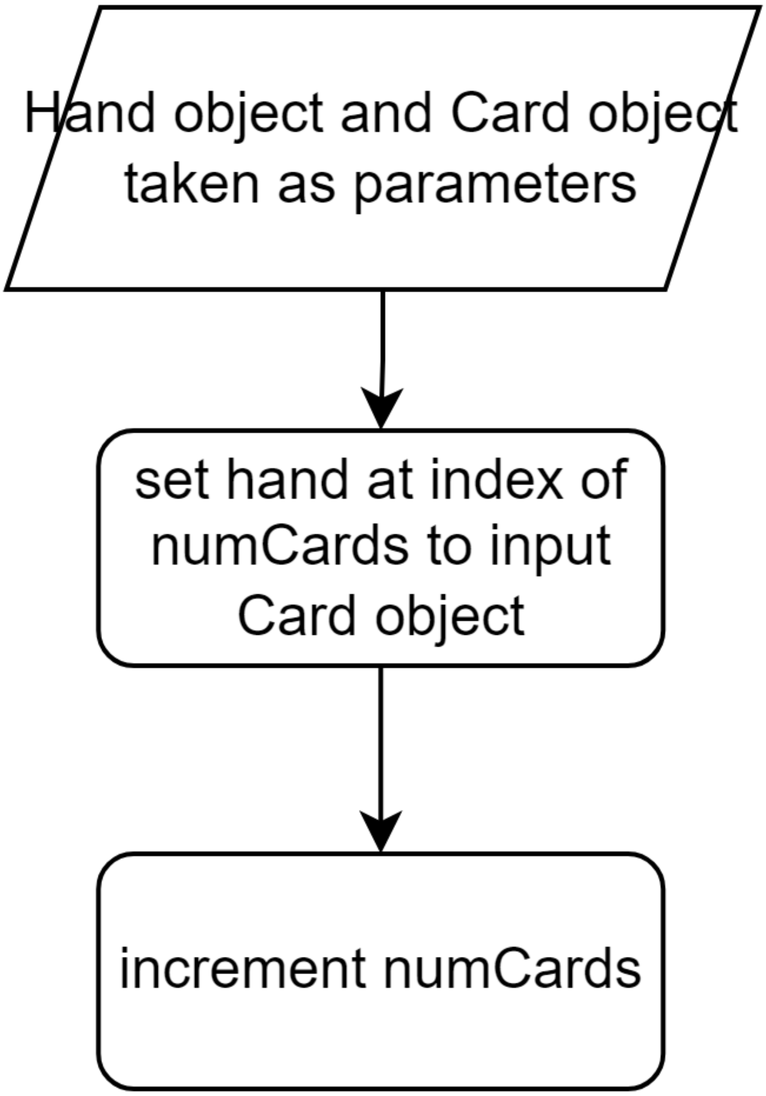

Figure 21: addCard() Function Flowchart

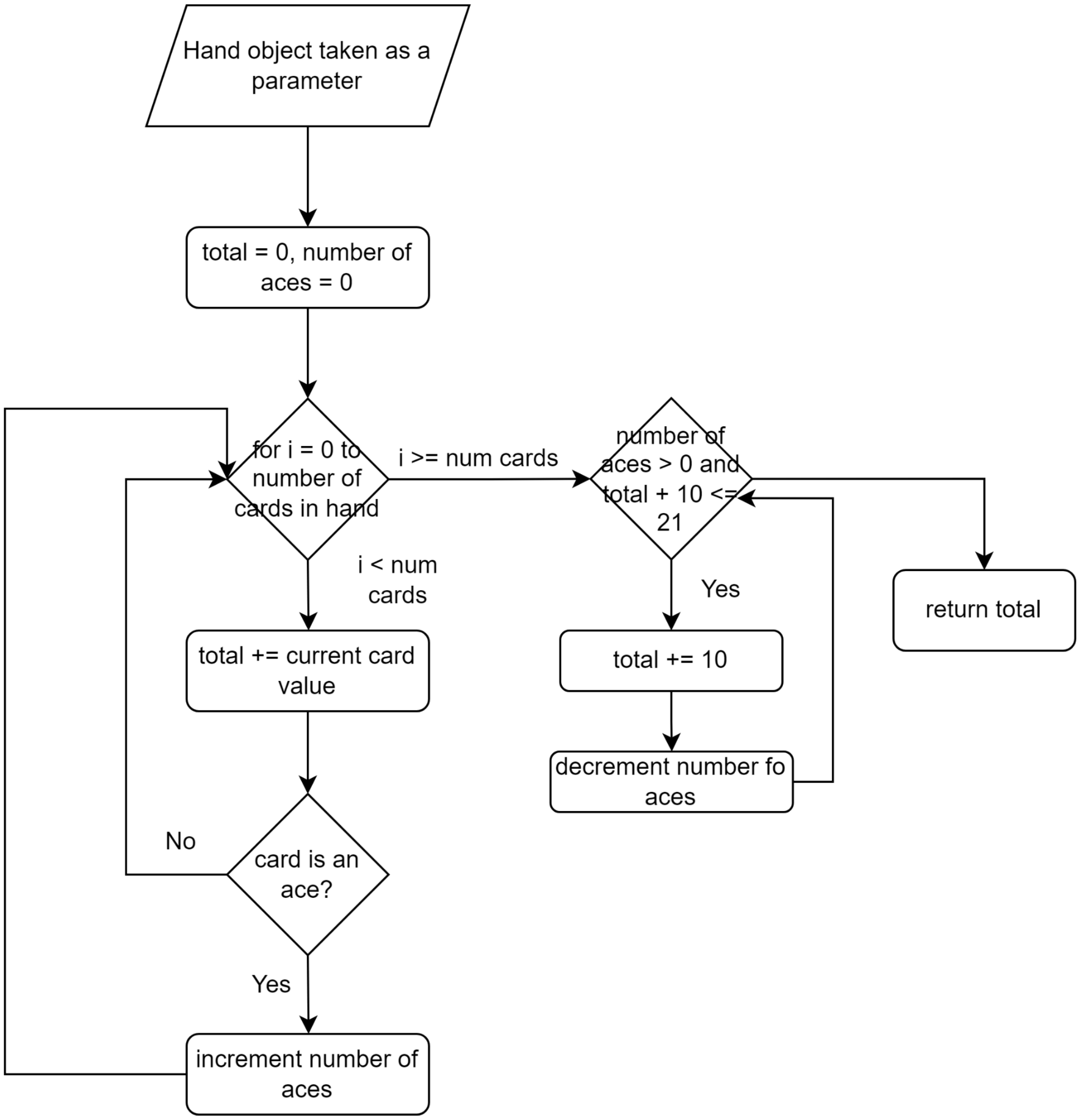

Figure 22: calculateHandValue() Function Flowchart

This function is used to calculate the input Hand object's total score value. This function also takes into account the number of aces in the hand. If the score may be less than or equal to 21 when the ace is valued at 11, then the score will be changed for this. If not, then the ace's value will stay at 1.

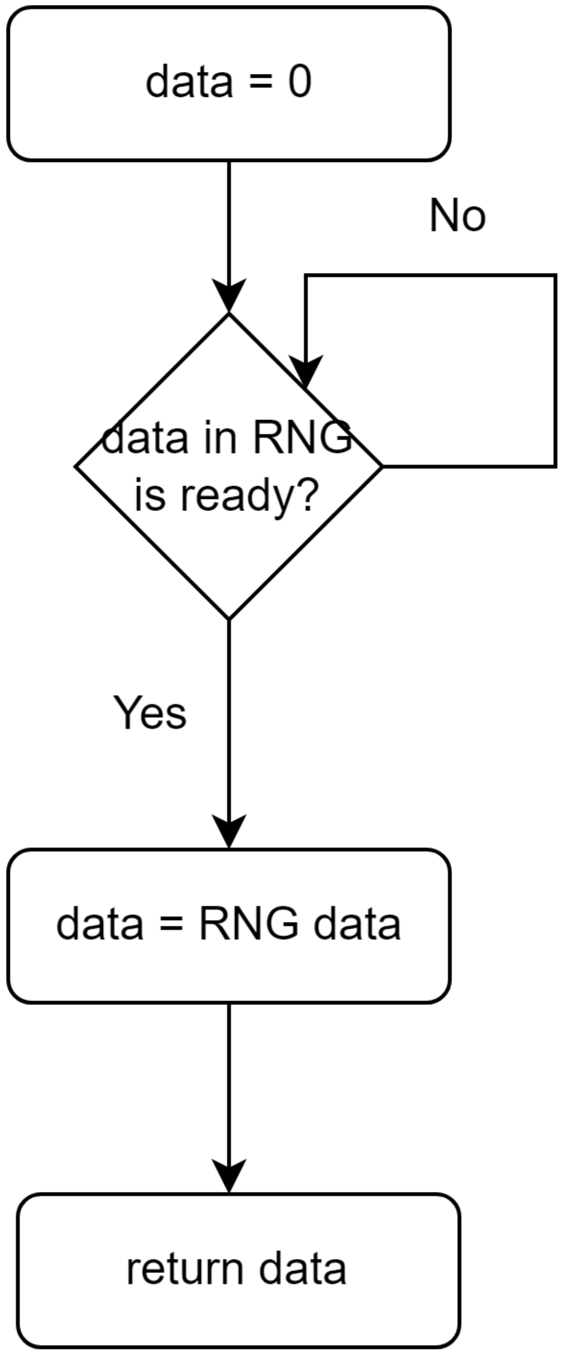

Figure 23: getRNG() Function Flowchart

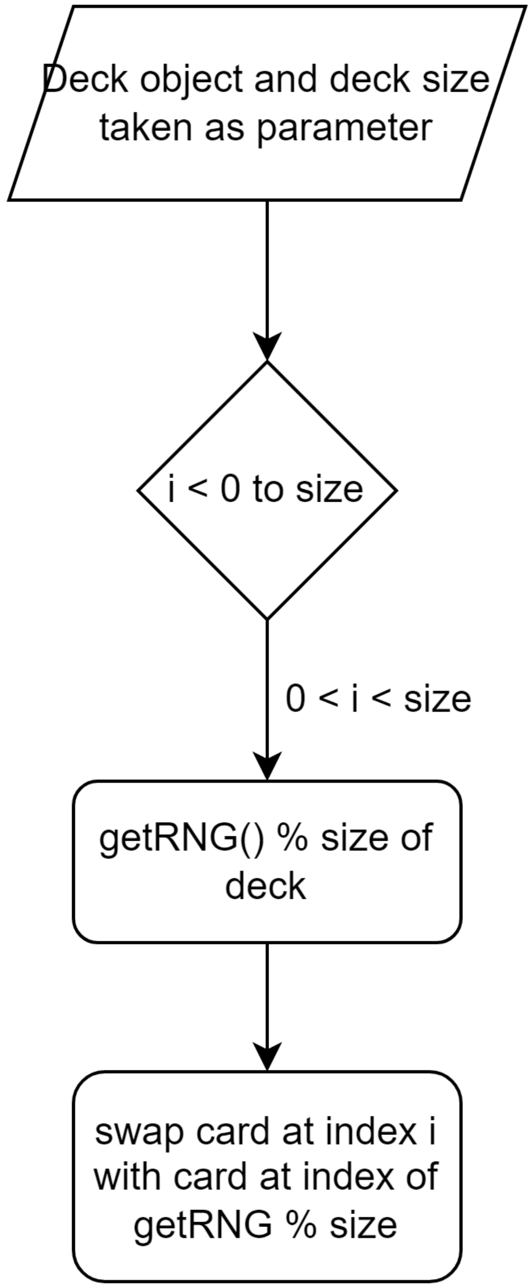

Figure 24: shuffleDeck() Function Flowchart

This function is used in the SHUFFLE_DECK state to swap the cards in the deck array. This function uses the RNG value that is generated by the built-in RNG, so that every call to this function always modifies the array differently.

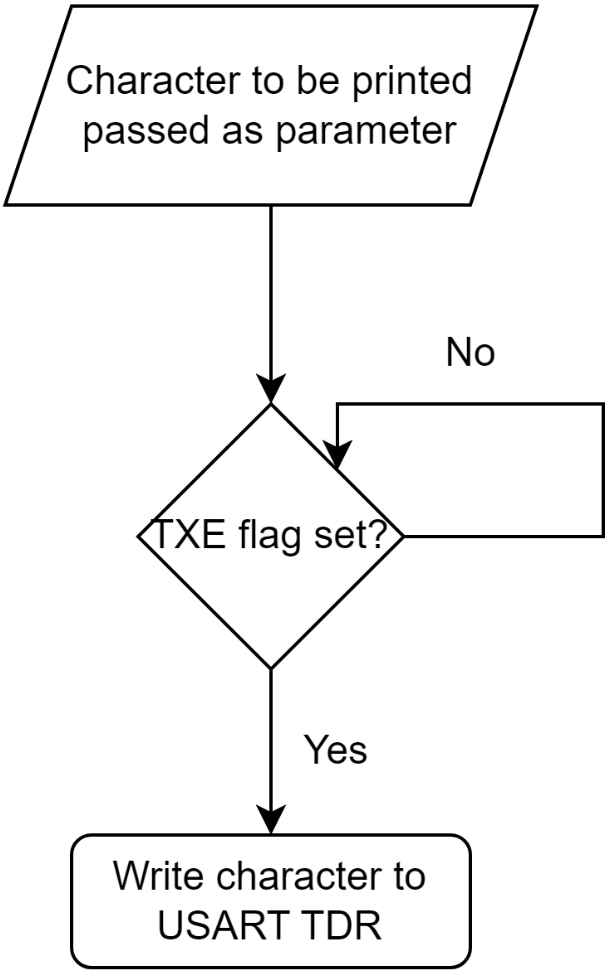

Figure 25: UART_print_char() Function Used in UART_print() Function

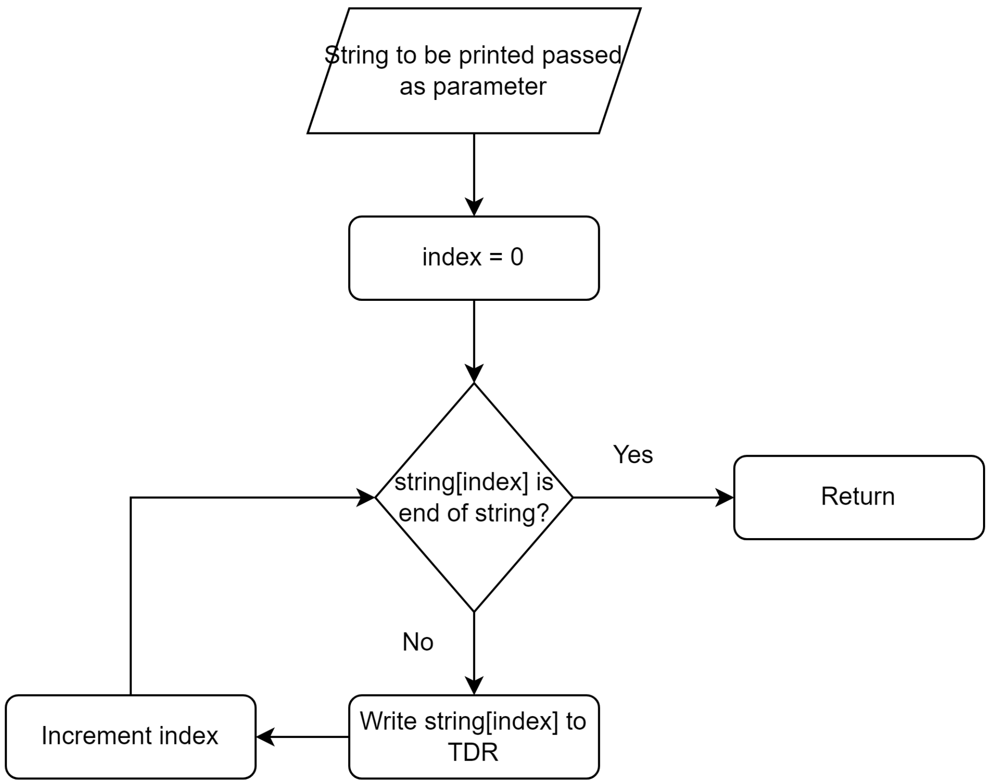

Figure 26: UART_print() Function Flowchart

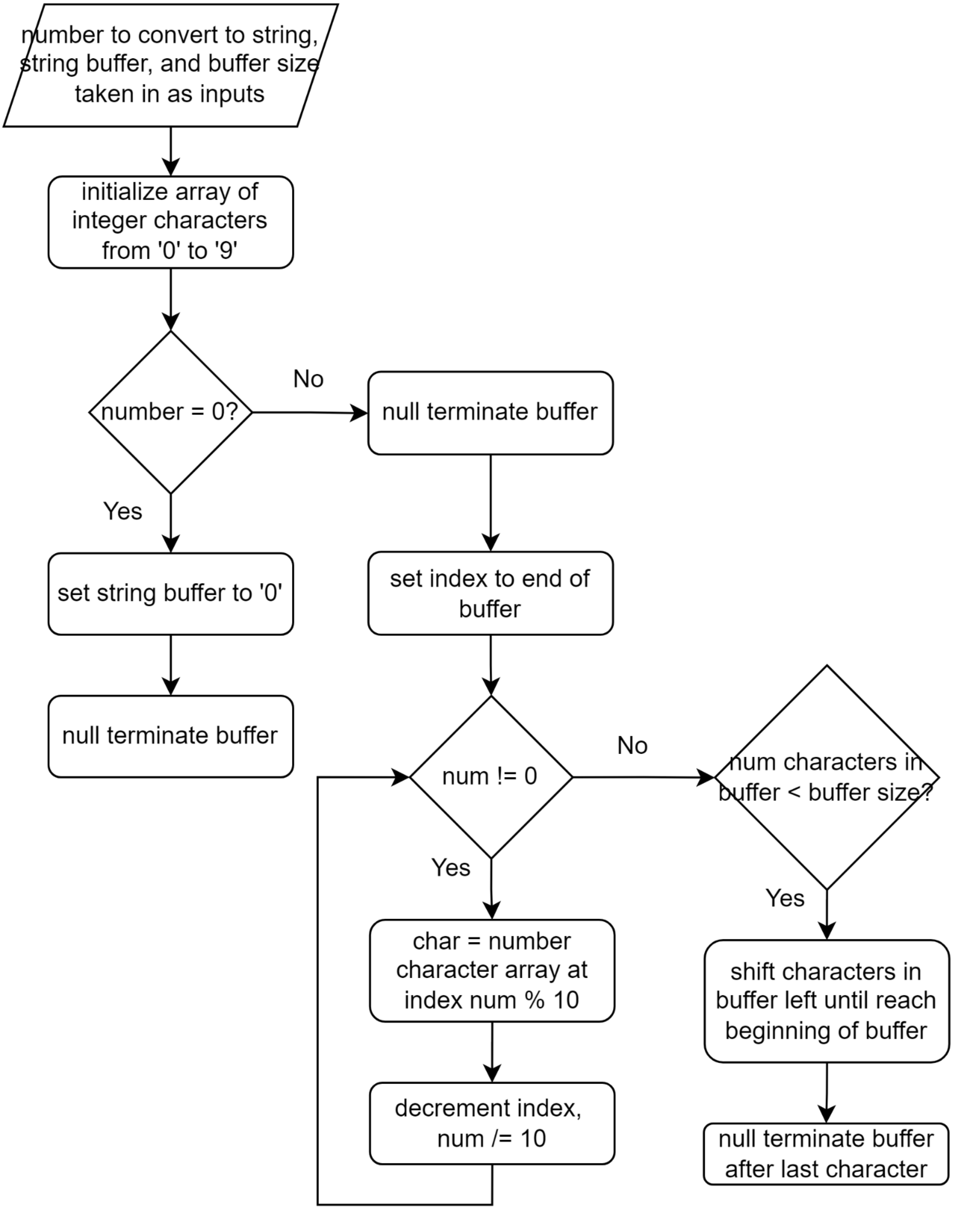

Figure 27: toString() Function Flowchart

This function is used to convert a number to a string so that it may be correctly printed to the terminal.

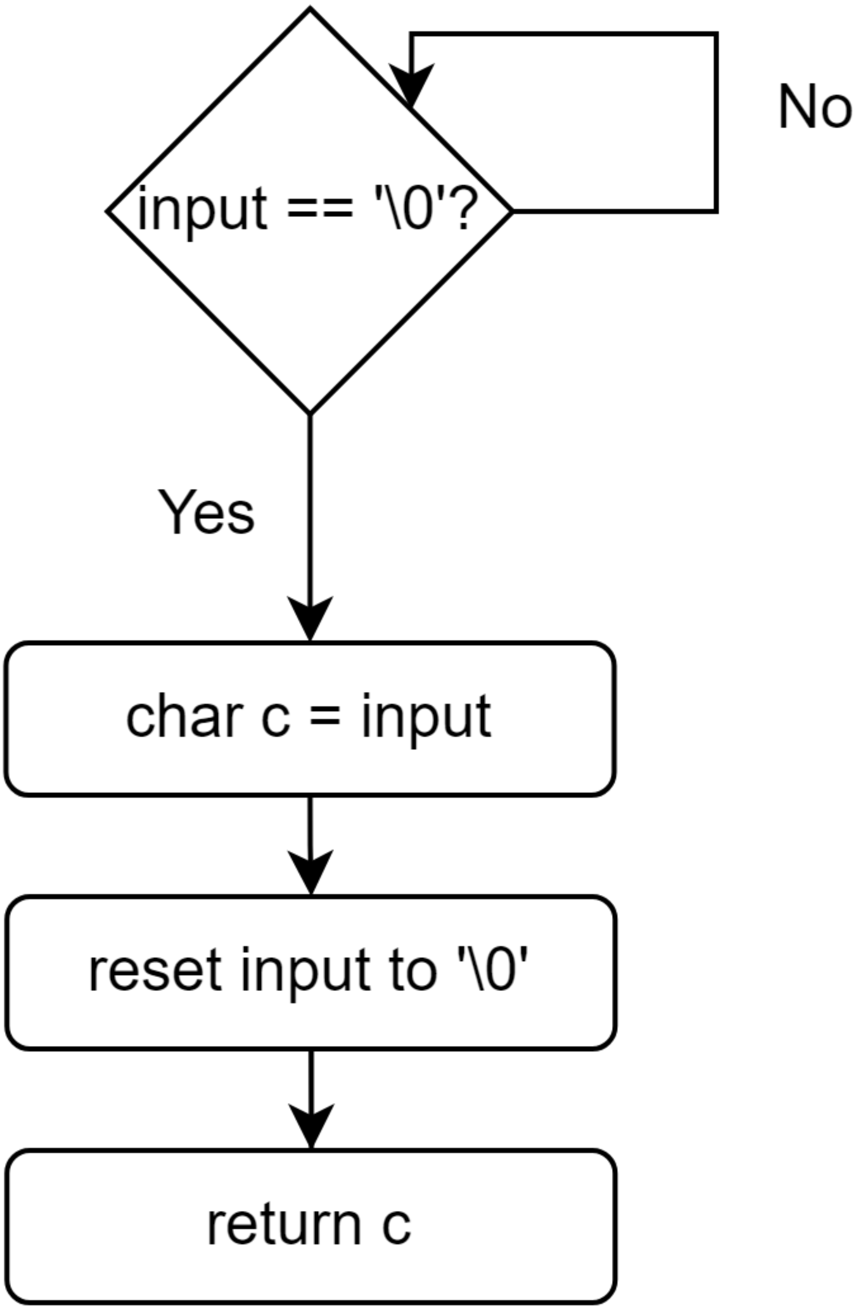

Figure 28: read_input() Function Flowchart

Battery Power Calculation

Estimate RNG value is generated once every 15 seconds.

15 seconds in sleep mode.

RNG independent clock domain current consumption: 2.2 µA/MHz

RNG AHB clock domain current consumption: 0.6 µA/MHz

RNG running at 48 MHz

$$RNG\ Current\ =\ (2.2\ \mu A/MHz)\ *\ 48\ MHz\ =\ 105.6\mu A$$

RNG takes 42 RNG clock cycles to generate number.

$$42\ cycles\ /\ 48000000\ Hz\ =\ 8.75\ x\ 10^{-7}\ s$$

$$I_{DD}\ =\ 2.77mA\ *\ 15s\ =\ 41.55\ mA\ *\ s$$

$$I_{RNG}\ =\ 8.8\ mA\ *\ 8.75\ x\ 10^{-7}s\ =\ 8\ \mu A\ *\ s$$

$$I_{AVG}\ =\ (8\mu As\ 41.55mAs)\ /\ 15\ s\ =\ 2.216x10^{-8}\ A$$

$$I_{TOTAL}\ =\ I_{AVG}\ +\ I_{RNGCLK}\ =\ 2.216\ x\ 10^{-8}\ +\ 105.6\ x\ 10^{-6}\ =\ 106\mu A$$

$$P_{In}\ x\ 0.93\ =\ P_{out}$$

$$4.5V\ *\ I_{batt}\ *\ 0.93\ =\ 3.3V\ *\ 106\mu A$$

$$I_{batt}\ =\ 83\mu A$$

<u>Battery Consumption</u>

Estimate 24000mAh

$$\frac{24000}{83\mu A}\ =\ 289157\ hours\ =\ 12048.2\ days$$

Appendix A: Main Source File

main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "deck.h"
#include "UART.h"

#define DELAY 2500000


/* Keep track of card in deck to draw */
uint8_t deckIndex = 0;

/* Keep track of winner */
/* 0: Nothing | 1: Blackjack | 2: Bust */
uint8_t dealerWin = 0;
uint8_t playerWin = 0;

uint16_t money = 1000;
uint16_t currentBet = 0;

/* Private variables ---------------------------------------------------------*/
RNG_HandleTypeDef hrng;

void SystemClock_Config(void);
static void MX_RNG_Init(void);
void print_table(Hand *dealerHand, Hand *playerHand);
void print_card(Card card);
```

```
void place_bet(int option);


int main(void)
{

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_RNG_Init();

  UART_init();
  RNG_init();



  //Create initial objects
  Hand *dealerHand;
  Hand *playerHand;
  Card deck[52];

  uint8_t dealerScore;
  uint8_t playerScore;

  uint8_t cashed = 0;

  typedef enum{
```

```
	START_GAME,
	BETTING,
	DEAL_FIRST_CARDS,
	CALCULATE_HANDS,
	PLAYER_CHOICE,
	DEALER_FLIPS,
	DEALER_TURN,
	ROUND_DONE,
	CASH_OUT,
	RESET_HANDS,
	BUY_BACK_IN,
	SHUFFLE_DECK
}state_var_type;

state_var_type state = START_GAME;

uint8_t dealerDone = 0;
char string[7];
string[0] = '\0';

//initialize Hand objects
dealerHand = hand_init();
playerHand = hand_init();



while (1)
{
	switch(state)
	{
```

```
case START_GAME:
	UART_print("\033[H");
	UART_print("\033[2J");
	UART_print("\033[15;15H");
	UART_print("Press Y to GAMBLE");

	//stay in this state until player clicks 'y'
	char start = read_input();
	if (start == 'y' || start == 'Y')
	{
		//start the game
		state = BETTING;
	}

	break;

case BETTING:
	//check if total money > bet
	if (money >= 500)
	{
		place_bet(3);
		state = SHUFFLE_DECK;
	}
	else if (money >= 250)
	{
		place_bet(2);
		state = SHUFFLE_DECK;
	}
	else if (money >= 100)
```

```
		{
			place_bet(1);
			state = SHUFFLE_DECK;
		}
		else
		{
			place_bet(0);
			state = BUY_BACK_IN;
		}
		break;

	case BUY_BACK_IN:
		UART_print("\033[H");
		UART_print("\033[2J");
		UART_print("\033[15;15H");
		UART_print("Buy Back In? [Y]");

		//stay in this state until player clicks 'y'
		char buyIn = read_input();
		if (buyIn == 'y' || buyIn == 'Y')
		{
			//start the game
			money = 1000;
			state = BETTING;
		}

		break;

	case SHUFFLE_DECK:
```

```
        //initialize and shuffle deck
        deck_init(deck);
        shuffleDeck(deck, 52);

        print_table(dealerHand, playerHand);
        for (int i = 0; i < DELAY; i++){}

        state = DEAL_FIRST_CARDS;
        break;

    case DEAL_FIRST_CARDS:
        //deal player's first card face up
        addCard(playerHand, deck[deckIndex]);
        deckIndex++;
        print_table(dealerHand, playerHand);
        for (int i = 0; i < DELAY; i++){}

        //deal dealer's first card face down
        addCard(dealerHand, deck[deckIndex]);
        deckIndex++;
        dealerHand->hand[0].visible = 0;
        print_table(dealerHand, playerHand);
        for (int i = 0; i < DELAY; i++){}

        //deal players second card faceup
        addCard(playerHand, deck[deckIndex]);
        deckIndex++;
        print_table(dealerHand, playerHand);
        for (int i = 0; i < DELAY; i++){}
```

```
        //play dealer's second card face up
        addCard(dealerHand, deck[deckIndex]);
        deckIndex++;
        print_table(dealerHand, playerHand);
        for (int i = 0; i < DELAY; i++){}

        state = CALCULATE_HANDS;
        break;

    case CALCULATE_HANDS:
        //check if the dealer/player has 21
        dealerScore = calculateHandValue(dealerHand);
        playerScore = calculateHandValue(playerHand);

        //set dealer wins flag
        if (dealerScore == 21)
        {
            dealerWin = 1;
            state = ROUND_DONE;
        }

        //set player wins flag
        if (playerScore == 21)
        {
            playerWin = 1;
            state = ROUND_DONE;
        }
```

```
        //check if dealer busts
        if (dealerScore > 21)
        {
              dealerWin = 2;
              state = ROUND_DONE;
        }

        //check if player busts
        if (playerScore > 21)
        {
              playerWin = 2;
              state = ROUND_DONE;
        }

        //if no one wins yet then move to the player's choice
        if ((dealerWin == 0) && (playerWin == 0) && (dealerDone == 0))
        {
              state = PLAYER_CHOICE;
        }

      if ((dealerWin == 0) && (playerWin == 0) && (dealerDone == 1))
      {
            state = ROUND_DONE;
      }
        break;

    case PLAYER_CHOICE:
        print_table(dealerHand, playerHand);
        for (int i = 0; i < DELAY; i++){}
```

```
        UART_print("\033[5;30H");
        UART_print("\033[0K");          //clear line from cursor to the right
        UART_print("Hit[H] or Stand[S]?");



        char choice = read_input();
        //for (int i = 0; i < DELAY; i++){}

        if (choice == 'h' || choice == 'H')
        {
                //deal player another card
                addCard(playerHand, deck[deckIndex]);
                deckIndex++;
                state = CALCULATE_HANDS;
        }
        else if (choice == 's' || choice == 'S')
        {
                state = DEALER_FLIPS;
        }
        break;

    case DEALER_FLIPS:
        //flip dealers face down card face up
        dealerHand->hand[0].visible = 1;
        print_table(dealerHand, playerHand);
        for (int i = 0; i < DELAY; i++){}
        //check if dealer hand is > 16
        dealerScore = calculateHandValue(dealerHand);
        if (dealerScore > 16)
```

```
        {
            //calculate to see who wins
            dealerDone = 1;
            state = CALCULATE_HANDS;
        }
        else
        {
            state = DEALER_TURN;
        }
        break;

    case DEALER_TURN:
        //add cards to dealer until dealers score is greater than 16
        dealerScore = calculateHandValue(dealerHand);
        print_table(dealerHand, playerHand);
        for (int i = 0; i < DELAY; i++){}

        if (dealerScore < 17)
        {
            //deal more cards to dealer
            addCard(dealerHand, deck[deckIndex]);
            deckIndex++;
            print_table(dealerHand, playerHand);
            for (int i = 0; i < DELAY; i++){}
        }
        else
        {
            dealerDone = 1;
            state = CALCULATE_HANDS;
```

```
			}
			break;

		case ROUND_DONE:
			dealerHand->hand[0].visible = 1;
			print_table(dealerHand, playerHand);
			//for (int i = 0; i < DELAY / 10; i++){}

			//print dealers score at end
			UART_print("\033[1;10H");
			string[0] = '\0';
			toString(calculateHandValue(dealerHand), string, 7);
			UART_print(string);

			for (int i = 0; i < DELAY; i++){}

			//dealer and player get blackjack
			if ((dealerWin == 1) && (playerWin == 1))
			{
				//tied
				UART_print("\033[5;30H");
				UART_print("\033[0K");		//clear line from cursor to the
right
				UART_print("Push: Player and Dealer Tied");
				money += currentBet;
			}
			//only dealer gets blackjack
			else if (dealerWin == 1)
			{
```

```
			//dealer wins
			UART_print("\033[5;30H");
			UART_print("\033[0K");		//clear line from cursor to the
right
			UART_print("The House Always Wins");
		}
		else if (playerWin == 1)
		{
			//player wins
			UART_print("\033[5;30H");
			UART_print("\033[0K");		//clear line from cursor to the
right
			UART_print("$$ Nice Hand $$");
			money += (currentBet * 2);
		}
		//dealer busts
		else if (dealerWin == 2)
		{
			//player wins
			UART_print("\033[5;30H");
			UART_print("\033[0K");		//clear line from cursor to the
right
			UART_print("$$ Nice Hand $$");
			money += (currentBet * 2);
		}
		//player busts
		else if (playerWin == 2)
		{
			//dealer wins
```

```
                UART_print("\033[5;30H");
                UART_print("\033[0K");          //clear line from cursor to the
right
                UART_print("The House Always Wins");
            }
            //check scores if neither busted or got blackjack
            else if ((dealerWin == 0) && (playerWin == 0))
            {
                if (dealerScore > playerScore)
                {
                    //dealer wins
                    UART_print("\033[5;30H");
                    UART_print("\033[0K");          //clear line from cursor to
the right
                    UART_print("The House Always Wins");
                }
                else if (dealerScore < playerScore)
                {
                    //player wins
                    UART_print("\033[5;30H");
                    UART_print("\033[0K");          //clear line from cursor to
the right
                    UART_print("$$ Nice Hand $$");
                    money += (currentBet * 2);
                }
                else
                {
                    //tie game
                    UART_print("\033[5;30H");
```

```
                        UART_print("\033[0K");          //clear line from cursor to
the right
                        UART_print("Push: Player and Dealer Tied");
                        money += currentBet;
                    }
                }

                //update player's money
                UART_print("\033[1;50H");
                UART_print("\033[0K");          //clear line from cursor to the right
                UART_print("$");
                string[0] = '\0';
                toString(money, string, 7);
                UART_print(string);
                string[0] = '\0';

                //delay to see ending result better
                for (int i = 0; i < DELAY * 3; i++){}



                state = CASH_OUT;
                break;

            case CASH_OUT:
                UART_print("\033[H");
                UART_print("\033[2J");
                UART_print("\033[15;15H");
                UART_print("CASH OUT? [Y] [N]");
                char cash = read_input();
```

```
        if (cash == 'y' || cash == 'Y')
        {
              UART_print("\033[H");
              UART_print("\033[2J");
              for (int i = 0; i < 30; i++)
              {
                    for (int j = 0; j < 30; j++)
                    {
                          UART_print("$");
                          for (int i = 0; i < DELAY / 100; i++){}
                    }
                  UART_print("\033[1B");       //move down 1
                  UART_print("\033[30D");           //move left 30 spaces
              }
              cashed = 1;
              state = RESET_HANDS;
        }
        else if (cash == 'n' || cash == 'N')
        {
              state = RESET_HANDS;
        }
        break;

  case RESET_HANDS:
        //reset globals and hands
        dealerHand->numCards = 0;
        playerHand->numCards = 0;
```

```
                dealerScore = 0;

                playerScore = 0;


                dealerWin = 0;

                playerWin = 0;


                dealerDone = 0;


                deckIndex = 0;


                if (cashed == 1)

                {

                      cashed = 0;

                      state = START_GAME;

                }

                else

                {

                      state = BETTING;

                }

                break;


          default:

                break;

     }




  }


}
```

```
void place_bet(int option)
{
	if (option == 3)
	{
		UART_print("\033[H");
		UART_print("\033[2J");
		UART_print("\033[15;10H");
		UART_print("Place bet: 100[1] 250[2] 500[3]");
		char bet = read_input();
		//subtract bet from current money
		if (bet == '1')
		{
			currentBet = 100;
			money -= currentBet;
		}
		if (bet == '2')
		{
			currentBet = 250;
			money -= currentBet;
		}
		if (bet == '3')
		{
			currentBet = 500;
			money -= currentBet;
		}
	}
	else if (option == 2)
	{
```

```
        UART_print("\033[H");
        UART_print("\033[2J");
        UART_print("\033[15;10H");
        UART_print("Place bet: 100[1] 250[2]");
        char bet = read_input();
        //subtract bet from current money
        if (bet == '1')
        {
              currentBet = 100;
              money -= currentBet;
        }
        if (bet == '2')
        {
              currentBet = 250;
              money -= currentBet;
        }
}
else if (option == 1)
{
        UART_print("\033[H");
        UART_print("\033[2J");
        UART_print("\033[15;10H");
        UART_print("Place bet: 100[1]");
        char bet = read_input();
        //subtract bet from current money
        if (bet == '1')
        {
              currentBet = 100;
              money -= currentBet;
```

```
		}
	}
	//if gets here, means the player does not have enough for minimum bet
	else
	{
		UART_print("\033[H");
		UART_print("\033[2J");
		UART_print("\033[15;10H");
		UART_print("Sorry you do not have enough money to bet");
		for (int i = 0; i < DELAY * 3; i++){}
	}

}

void print_table(Hand *dealerHand, Hand *playerHand)
{
	/* UART print codes */
	char *clearScreen = "\033[2J";
	//char *resetCurs = "\033[H";
	char string[7];
	string[0] = '\0';



	UART_print(clearScreen);

	//print player's money
	UART_print("\033[1;50H");
	UART_print("$");
	toString(money, string, 7);
```

```
UART_print(string);
string[0] = '\0';

UART_print("\033[H");
//UART_print(resetCurs);

//print dealer's cards
UART_print("Dealer");
UART_print("\033[1B");       //move down 1
UART_print("\033[6D");       //move left 6 spaces

for (int i = 0; i < dealerHand->numCards; i++)
{
      //print card then move cursor location
      print_card(dealerHand->hand[i]);
}

//print player's cards
UART_print("\033[1;20H");
UART_print("Player");
UART_print("\033[1B");       //move down 1
UART_print("\033[6D");       //move left 6 spaces

for (int i = 0; i < playerHand->numCards; i++)
{

      //print card then move cursor location
      print_card(playerHand->hand[i]);
}
```

```
	UART_print("\033[1;30H");
	toString(calculateHandValue(playerHand), string, 7);
	UART_print(string);



}

void print_card(Card card)
{
	char string[7];
	string[0] = '\0';

	UART_print(" ----- ");
	UART_print("\033[1B");		//move down 1
	UART_print("\033[7D");		//move left 7 spaces

	if (card.visible == 0)
	{
		UART_print("|#####|");
		UART_print("\033[1B");		//move down 1
		UART_print("\033[7D");		//move left 7 spaces
		UART_print("|#####|");
		UART_print("\033[1B");		//move down 1
		UART_print("\033[7D");		//move left 7 spaces
		UART_print("|#####|");

	}
```

```
//check if is a face card
else if (strcmp(card.face, "X"))
{
      UART_print("|");
      UART_print(card.face);
      UART_print("     |");

      UART_print("\033[1B");       //move down 1
      UART_print("\033[7D");       //move left 7 spaces
      UART_print("|      |");

      UART_print("\033[1B");       //move down 1
      UART_print("\033[7D");       //move left 7 spaces
      UART_print("|     ");
      UART_print(card.face);
      UART_print("|");


}
//check if 2 digits
else if (card.value > 9)
{
      toString(card.value, string, 7);
      UART_print("|");
      UART_print(string);
      UART_print("    |");

      UART_print("\033[1B");       //move down 1
      UART_print("\033[7D");       //move left 7 spaces
      UART_print("|      |");
```

```
        UART_print("\033[1B");       //move down 1
        UART_print("\033[7D");       //move left 7 spaces
        UART_print("|   ");
        UART_print(string);
        UART_print("|");
}
//single digit
else
{
        toString(card.value, string, 7);
        UART_print("|");
        UART_print(string);
        UART_print("    |");

        UART_print("\033[1B");       //move down 1
        UART_print("\033[7D");       //move left 7 spaces
        UART_print("|     |");

        UART_print("\033[1B");       //move down 1
        UART_print("\033[7D");       //move left 7 spaces
        UART_print("|    ");
        UART_print(string);
        UART_print("|");
}

//print bottom of the card
UART_print("\033[1B");       //move down 1
UART_print("\033[7D");       //move left 7 spaces
```

```
    UART_print(" ----- ");

    UART_print("\033[2B");      //move down 1
    UART_print("\033[7D");      //move left 7 spaces

}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
```

```
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_9;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief RNG Initialization Function
  * @param None
  * @retval None
  */
static void MX_RNG_Init(void)
{
```

```
  /* USER CODE BEGIN RNG_Init 0 */

  /* USER CODE END RNG_Init 0 */

  /* USER CODE BEGIN RNG_Init 1 */

  /* USER CODE END RNG_Init 1 */
  hrng.Instance = RNG;
  if (HAL_RNG_Init(&hrng) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RNG_Init 2 */

  /* USER CODE END RNG_Init 2 */

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
```

```c
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Appendix B: Deck Source and Header Files

deck.c

```c
/*
 * deck.c
 *
 *  Created on: Jun 3, 2023
 *      Author: natha
 */

#include "deck.h"
#include <stdlib.h>



//Configure the built in RNG
void RNG_init(void)
{
	//enable RNG clock
	RCC->AHB2ENR |= (RCC_AHB2ENR_RNGEN);

	//reset RNG
	RCC->AHB2RSTR |= RCC_AHB2RSTR_RNGRST;		//resets RNG
	RCC->AHB2RSTR &= ~(RCC_AHB2RSTR_RNGRST);



	//enable RNGEN for generation
	RNG->CR |= RNG_CR_RNGEN;

	//configure
	RNG->CR |= (1 << 5);		//CED
	RNG->CR &= ~(RNG_CR_IE);	//disable interrupt enable
```

```
}

//creates the 52 card deck (not shuffled)
void deck_init(Card deck[])
{
	uint8_t index = 0;
	uint8_t suitsIndex, number, faceCard;
	char suits[4][9] = {"Spades", "Clubs", "Diamonds", "Hearts"};
	char faces[3][2] = {"J", "Q", "K"};

	//put in each suit
	for (suitsIndex = 0; suitsIndex < 4; suitsIndex++)
	{
		//put in each number card
		for (number = 2; number < 11; number++)
		{
			deck[index].value = number;
			deck[index].visible = 1;
			strcpy(deck[index].suit, suits[suitsIndex]);
			strcpy(deck[index].face, "X");
			index++;
		}

		//put in face cards
		for (faceCard = 0; faceCard < 3; faceCard++)
		{
			deck[index].value = 10;
			deck[index].visible = 1;
			strcpy(deck[index].suit, suits[suitsIndex]);
```

```
                strcpy(deck[index].face, faces[faceCard]);
                index++;
            }

            //put in Ace
            deck[index].value = 1;
            deck[index].visible = 1;
            strcpy(deck[index].suit, suits[suitsIndex]);
            strcpy(deck[index].face, "A");
            index++;
        }

}

Hand* hand_init(void)
{
        Hand* hand = malloc(sizeof(Hand));
        hand->numCards = 0;
        return hand;
}

void addCard(Hand* hand, Card card)
{
        hand->hand[hand->numCards] = card;
        hand->numCards++;
}

//calculate the total value of target hand
uint8_t calculateHandValue(Hand* hand)
```

```
{
    uint8_t totalValue = 0;
    uint8_t numAces = 0;

    for (uint8_t i = 0; i < hand->numCards; i++)
    {
        totalValue += hand->hand[i].value;

        //check if card is an Ace
        if (hand->hand[i].value == 1)
        {
            numAces++;
        }

    }

    //change aces value depending on total
    while (numAces > 0 && totalValue + 10 <= 21)
    {
        totalValue += 10;
        numAces--;
    }

    return totalValue;
}

//get rng value from RNG data register
uint32_t getRNG(void)
{
```

```
	uint32_t data = 0;

	//wait until rng is ready
	while (!(RNG->SR & RNG_SR_DRDY)){}

	data = RNG->DR;
	return data;
}

//shuffle array using the rng value
void shuffleDeck(Card deck[], int size)
{
	for (int i = 0; i < size; i++)
	{
		int swap = getRNG() % (size - 1);
		Card temp = deck[i];
		deck[i] = deck[swap];
		deck[swap] = temp;
	}
}
```

----

deck.h

```
/*
 * deck.h
 *
 *  Created on: Jun 3, 2023
 *      Author: natha
```

```c
 */
#ifndef SRC_DECK_H_
#define SRC_DECK_H_

#include <stdint.h>
#include <string.h>
#include "stm32l4xx_hal.h"

typedef struct
{
      uint8_t value;
      uint8_t visible; //determine if facedown or faceup
      char face[2];    //letter value for face cards
      char suit[9];    //spades, diamonds, clubs, hearts

}Card;

typedef struct {
    Card hand[11];     //store the cards in the hand
    int numCards;
} Hand;

void RNG_init(void);
void deck_init(Card deck[]);
Hand* hand_init(void);
void addCard(Hand* hand, Card card);
uint8_t calculateHandValue(Hand* hand);
uint32_t getRNG(void);
void shuffleDeck(Card deck[], int size);
```

```
#endif /* SRC_DECK_H_ */
```

Appendix C: UART Source and Header Files

uart.c

```
#include "uart.h"
#include <stdio.h>

#define USARTDIV 0x00D0UL
#define ESC_CHAR 0x1B
#define MAX_IDX 9

char input = '\0';

void UART_init(void)
{
	/* enable clock for GPIOA and USART2*/
	RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);
	RCC->APB1ENR1|= (RCC_APB1ENR1_USART2EN);

	/* set to alternate function mode */
	GPIOA->MODER &= ~(GPIO_MODER_MODE2 | GPIO_MODER_MODE3);
	GPIOA->MODER |= (GPIO_MODER_MODE2_1 | GPIO_MODER_MODE3_1);

	/* enable alternate function registers */
	GPIOA->AFR[0] &= ~(GPIO_AFRL_AFSEL2_Msk | GPIO_AFRL_AFSEL3_Msk); // clear AFR
	GPIOA->AFR[0] |= ( (0x7UL << GPIO_AFRL_AFSEL2_Pos) | 0x7UL << GPIO_AFRL_AFSEL3_Pos);
	// set PA2, PA3
```

```
    /* set to high speed (11) */
    GPIOA->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED2 | GPIO_OSPEEDR_OSPEED3);
    GPIOA->OSPEEDR |= (GPIO_OSPEEDR_OSPEED2 | GPIO_OSPEEDR_OSPEED3);

    /* program the M bits in USART_CR1 to define the word length */
    USART2->CR1 &= ~(USART_CR1_M); // bit 28: set to 00 (8 data bits) 1 char is 8 bits
    USART2->CR1 &= ~(USART_CR1_UE); // disable UE to write BRR

    /* select the desired baud rate using the USART_BRR register */
    USART2->BRR = (USARTDIV); // set bits of BRR to USARTDIV

    /* program the number of stop bits in USART_CR2. */
    USART2->CR2 &= ~(USART_CR2_STOP); // setting stop bit to 1 (00)

    /*enable the USART by writing the UE bit in USART_CR1 register to 1*/
    USART2->CR1 |= (USART_CR1_UE); // setting bit 0 to 1

    /*set the TE bit in USART_CR1 to send an idle frame as first transmission*/
    USART2->CR1 |= (USART_CR1_TE); // setting transmission enable to ?

    /* enable interrupts */
    USART2->CR1 |= USART_CR1_RXNEIE;
    NVIC->ISER[1] = (1 << (USART2_IRQn & 0x1F));
    __enable_irq();

    USART2->CR1 |= (USART_CR1_RE); // setting reception enable to 1 ?
}
```

```
void UART_print_char(char string) // right now just printing one character

{


      /* check the TXE flag*/

      while (!(USART_ISR_TXE & USART2->ISR));// wait until it is ready to be written to,

if it is empty then write to it

      USART2->TDR = string; // write a char

}


void UART_print(char* string)

{

      int i = 0;

      while(string[i] != '\0')

      {

            UART_print_char(string[i]);

            i += 1;

      }

}



void USART_ESC_Code(char* code)

{

      /* printing an escape character 0x1B*/

      while (!(USART_ISR_TXE & USART2->ISR));

      USART2->TDR = ESC_CHAR;

      UART_print(code);

}
```

```
void toString(uint32_t value, char *str, int max_index)
{
	char nums[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
	if (value == 0)
	{
		str[0] = '0';
		str[1] = '\0';
	}
	else
	{
		uint32_t num = value;
		str[MAX_IDX - 1] = '\0'; // setting index 8 to null
		int idx = MAX_IDX - 2;   // start adding from the end of the array
		int size = 0 ;
		/* adding individual characters into string */
		while(num)
		{
			char toprint = nums[num % 10];
			str[idx] = toprint;
			idx -= 1;
			num /= 10;
			size += 1;
		}
		/* moving stuff so it can be printed */
		if(size < MAX_IDX - 1) // if the number is less than 8 digits
		{
			int gap = MAX_IDX - size - 1;
			for(int i = 0; i < size + 1; i++)
```

```c
                {
                        str[i] = str[i+ gap];
                }
                str[size] = '\0';
            }
        }
}

char read_input(void)
{
        //check if input has changed
     while (input == '\0'){}

     char c = input;
     input = '\0';
     return c;

}

void USART2_IRQHandler(void){
        if ((USART2->ISR & USART_ISR_RXNE) !=0)
        {
                input = USART2->RDR;
        }
}
```

----

uart.h

```c
#ifndef INC_UART_H_
#define INC_UART_H_
#include "stm32l4xx_hal.h"


void UART_init(void);
void UART_print_char(char string);
void UART_print(char* word);
void USART_ESC_Code(char* code);
void toString(uint32_t value, char *str, int max_index);
char read_input(void);


#endif /* INC_UART_H_ */
```
