Team 8

# Maintenance Guide

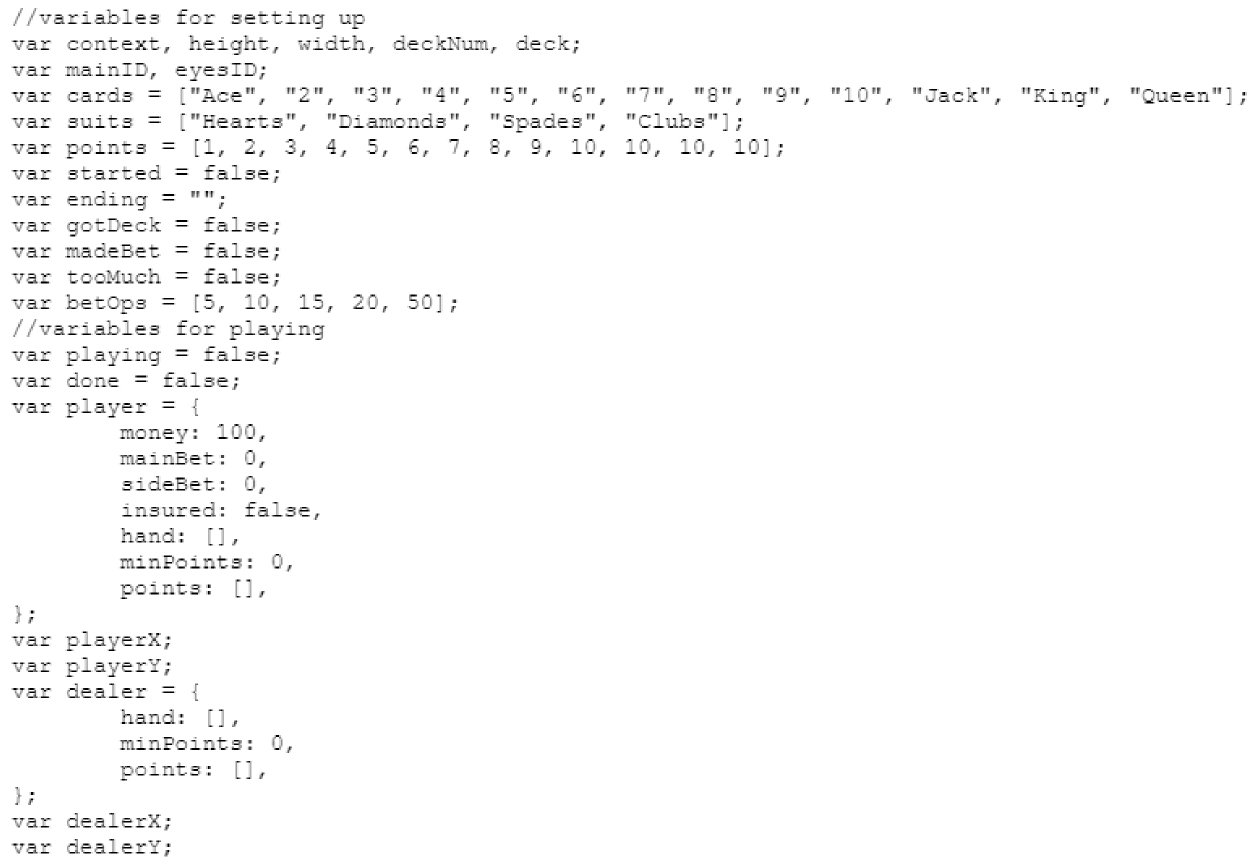
## Introduction

This guide is intended as a guide for the continued upkeep and maintenance of the project created by Team 8, Blackjack or Bust. The guide's purpose is to explain in as much detail as possible how to update the code we came up with in the future without breaking the code or the product as a whole.

The product is intended as a game of blackjack and it is the aim of this guide to maintain the integrity of the game of blackjack while also allowing for upgrades and personal touches of other developers.

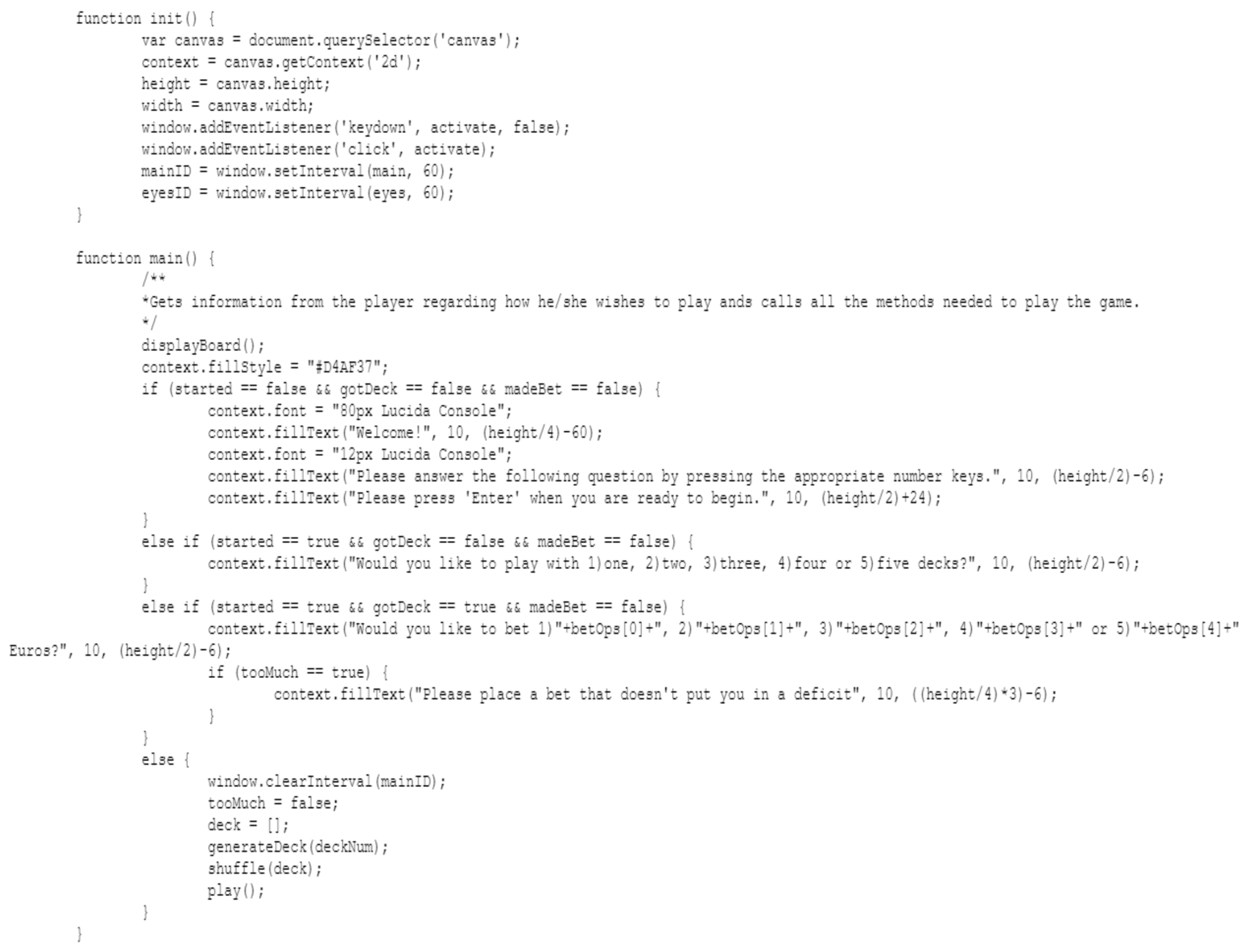
## Part 1: Javascript

The Javascript used for this product is responsible for the main functionality of the game itself. There will be some room for change in the Javascript to allow for developers to add their own spin on blackjack but as a whole it should really remain as is.



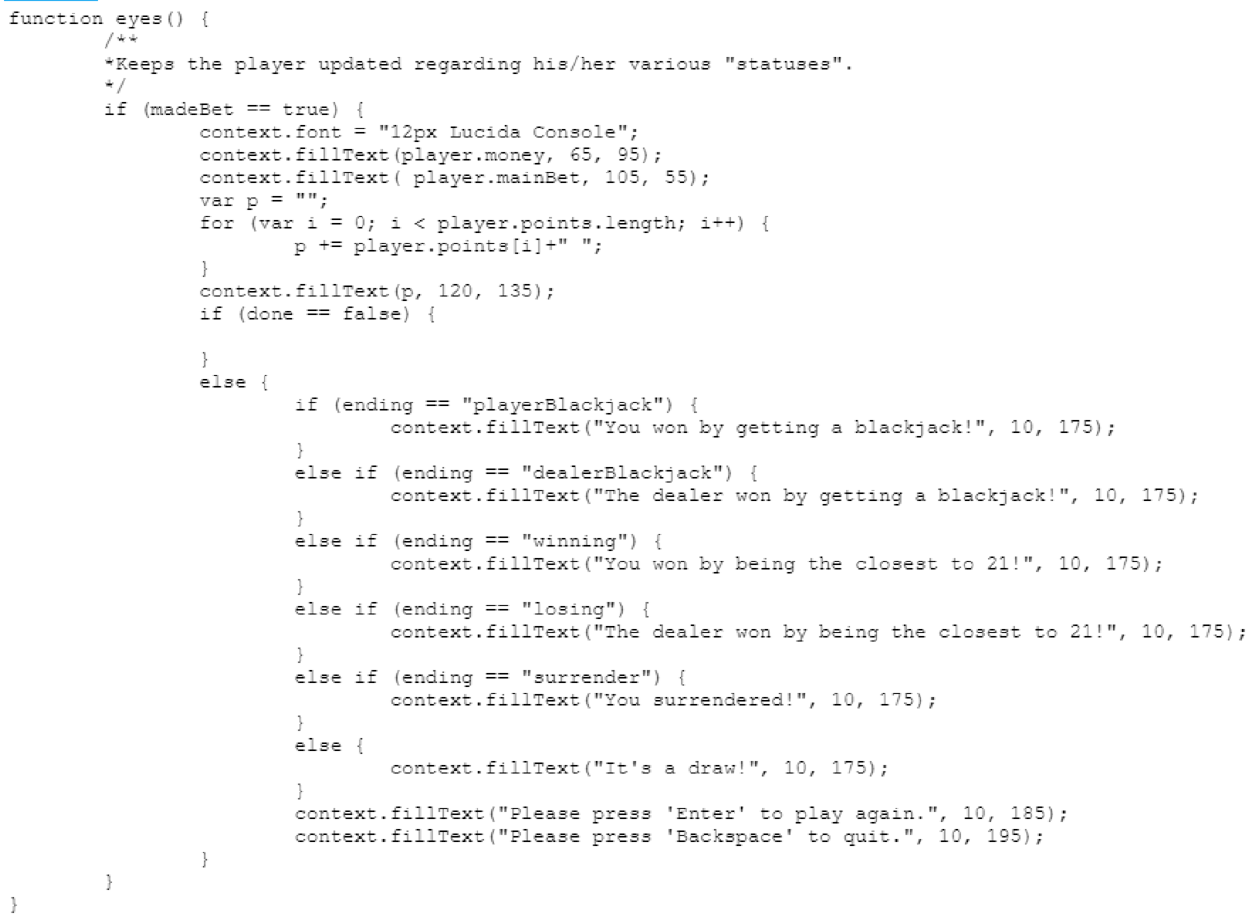
```
//variables for setting up
var context, height, width, deckNum, deck;
var mainID, eyesID;
var cards = ["Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "King", "Queen"];
var suits = ["Hearts", "Diamonds", "Spades", "Clubs"];
var points = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10];
var started = false;
var ending = "";
var gotDeck = false;
var madeBet = false;
var tooMuch = false;
var betOps = [5, 10, 15, 20, 50];
//variables for playing
var playing = false;
var done = false;
var player = {
        money: 100,
        mainBet: 0,
        sideBet: 0,
        insured: false,
        hand: [],
        minPoints: 0,
        points: [],
};
var playerX;
var playerY;
var dealer = {
        hand: [],
        minPoints: 0,
        points: [],
};
var dealerX;
var dealerY;
```

As we can see from the screenshot, this portion of the Javascript declares all variables used. Most of these should remain untouched, however the player’s starting money can be altered quite easily to whatever the developer likes. The *betOps* variable can also be altered in order to change the increments by which you can bet. As decks of cards are standard over the world the *cards* and *suits* variables should not be changed under any circumstances.



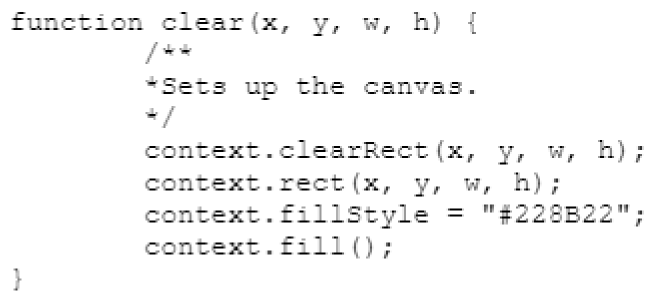
```
	function init() {
		var canvas = document.querySelector('canvas');
		context = canvas.getContext('2d');
		height = canvas.height;
		width = canvas.width;
		window.addEventListener('keydown', activate, false);
		window.addEventListener('click', activate);
		mainID = window.setInterval(main, 60);
		eyesID = window.setInterval(eyes, 60);
	}

	function main() {
		/**
		*Gets information from the player regarding how he/she wishes to play ands calls all the methods needed to play the game.
		*/
		displayBoard();
		context.fillStyle = "#D4AF37";
		if (started == false && gotDeck == false && madeBet == false) {
			context.font = "80px Lucida Console";
			context.fillText("Welcome!", 10, (height/4)-60);
			context.font = "12px Lucida Console";
			context.fillText("Please answer the following question by pressing the appropriate number keys.", 10, (height/2)-6);
			context.fillText("Please press 'Enter' when you are ready to begin.", 10, (height/2)+24);
		}
		else if (started == true && gotDeck == false && madeBet == false) {
			context.fillText("Would you like to play with 1)one, 2)two, 3)three, 4)four or 5)five decks?", 10, (height/2)-6);
		}
		else if (started == true && gotDeck == true && madeBet == false) {
			context.fillText("Would you like to bet 1)"+betOps[0]+", 2)"+betOps[1]+", 3)"+betOps[2]+", 4)"+betOps[3]+" or 5)"+betOps[4]+"
 Euros?", 10, (height/2)-6);
			if (tooMuch == true) {
				context.fillText("Please place a bet that doesn't put you in a deficit", 10, ((height/4)*3)-6);
			}
		}
		else {
			window.clearInterval(mainID);
			tooMuch = false;
			deck = [];
			generateDeck(deckNum);
			shuffle(deck);
			play();
		}
	}
```

In the above screenshot the *init()* and *main()* functions can be seen. These functions again should largely remain unchanged, particularly *init()*. However some slight personalisation changes can be made. For example, the *context.font* and *context.fillStyle* can be altered to whatever the developer wishes as they are mainly aesthetic changes. For the first *else if* statement in the *main()* function the developer could also alter the amount of decks the player can use, however this will involve changes later in the code as well.



```
function eyes() {
        /**
        *Keeps the player updated regarding his/her various "statuses".
        */
        if (madeBet == true) {
                context.font = "12px Lucida Console";
                context.fillText(player.money, 65, 95);
                context.fillText( player.mainBet, 105, 55);
                var p = "";
                for (var i = 0; i < player.points.length; i++) {
                        p += player.points[i]+" ";
                }
                context.fillText(p, 120, 135);
                if (done == false) {

                }
                else {
                        if (ending == "playerBlackjack") {
                                context.fillText("You won by getting a blackjack!", 10, 175);
                        }
                        else if (ending == "dealerBlackjack") {
                                context.fillText("The dealer won by getting a blackjack!", 10, 175);
                        }
                        else if (ending == "winning") {
                                context.fillText("You won by being the closest to 21!", 10, 175);
                        }
                        else if (ending == "losing") {
                                context.fillText("The dealer won by being the closest to 21!", 10, 175);
                        }
                        else if (ending == "surrender") {
                                context.fillText("You surrendered!", 10, 175);
                        }
                        else {
                                context.fillText("It's a draw!", 10, 175);
                        }
                        context.fillText("Please press 'Enter' to play again.", 10, 185);
                        context.fillText("Please press 'Backspace' to quit.", 10, 195);
                }
        }
}
```

The function shown in the above screenshot, *eyes()*, can be altered but again, it will be mainly changes to the appearance of the game rather than the actual functionality. The main thing that can be changed is the messages that appear depending on the outcome of the game and, again, the font styles.



```
function clear(x, y, w, h) {
        /**
        *Sets up the canvas.
        */
        context.clearRect(x, y, w, h);
        context.rect(x, y, w, h);
        context.fillStyle = "#228B22";
        context.fill();
}
```

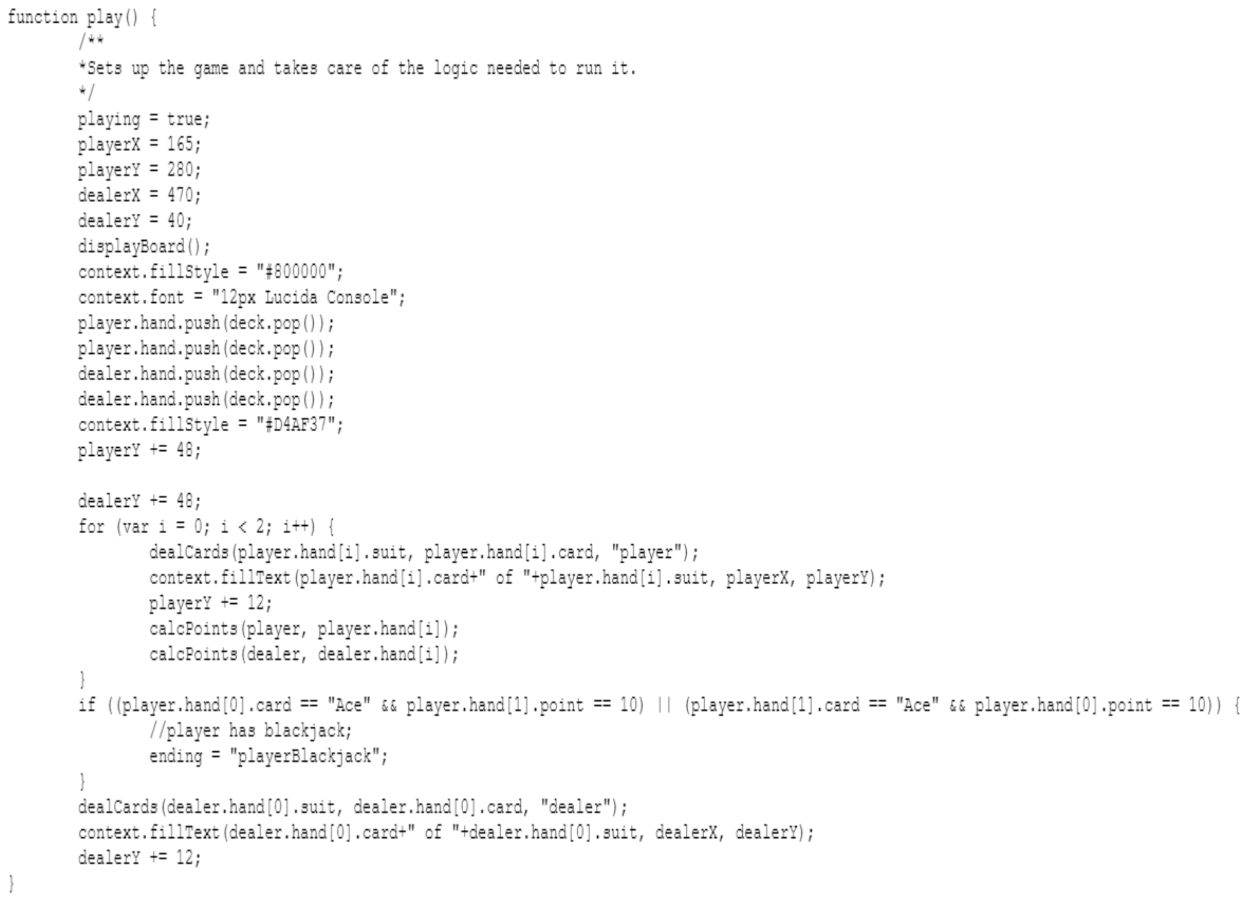
This function is used to set up the canvas, so this can be changed to the developers personal preference of canvas appearance.

```
function placeBet(array, index) {
        /**
        *Checks if the player can place a bet and either places the bet or tells the player he/she can't place that bet.
        */
        if (player.money >= array[index]) {
                player.mainBet = array[index];
                player.money -= array[index];
                if (array == betOps) {
                        madeBet = true;
                }
        }
        else {
                tooMuch = true;
        }
}
```

This function is what allows or disallows the player to place a bet depending on how much money they have. This function should not be changed and remain constant over any iterations of the game.

```
function generateDeck(numOfDecks) {
        /**
        *Creates the 52 card deck(s) needed to play the game.
        */
        for (var i = 1; i <= numOfDecks; i++) {
                for (var j = 0; j < suits.length; j++) {
                        for (var k = 0; k < cards.length; k++) {
                                var c = {
                                        card: cards[k],
                                        suit: suits[j],
                                        point: points[k],
                                };
                                deck.push(c);
                        }
                }
        }
}

function shuffle(array) {
        /**
        *Shuffles the deck(s) 3 times.
        */
        var currIndex, randIndex, tempValue;
        for (var i = 0; i < 3; i++) {
                currIndex = array.length;
                while (0 !== currIndex) {
                        currIndex -= 1;
                        randIndex = Math.floor(Math.random() * (currIndex+1));
                        tempValue = array[currIndex];
                        array[currIndex] = array[randIndex];
                        array[randIndex] = tempValue;
                }
        }
        return;
}
```

These two functions deal with deck manipulation. The first *generateDeck()* is used to choose how many decks the player wishes to use based on user input. Our version has a max number of 5 decks, however this can be changed by altering the *main()* function from earlier in the program. The *shuffle()* function shuffles the decks selected. For our product we decided to shuffle the decks three times however any developer can change the amount of times to shuffle if they so wish.



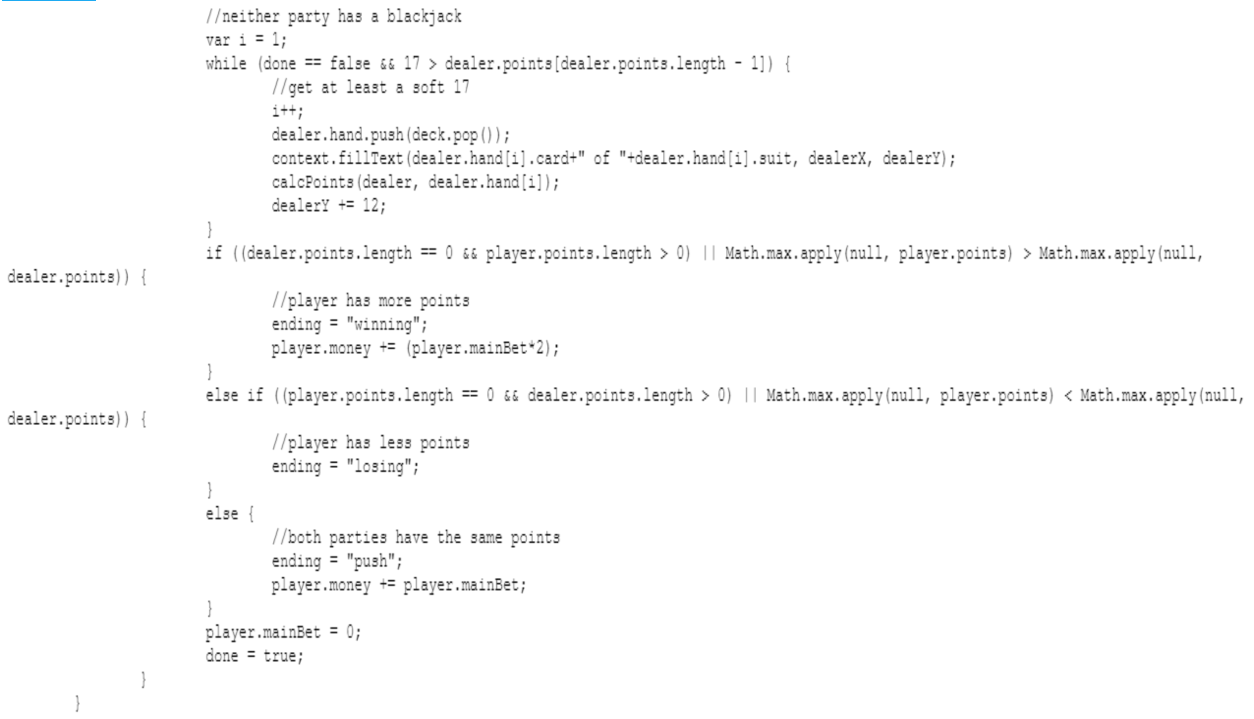
```
function play() {
        /**
        *Sets up the game and takes care of the logic needed to run it.
        */
        playing = true;
        playerX = 165;
        playerY = 280;
        dealerX = 470;
        dealerY = 40;
        displayBoard();
        context.fillStyle = "#800000";
        context.font = "12px Lucida Console";
        player.hand.push(deck.pop());
        player.hand.push(deck.pop());
        dealer.hand.push(deck.pop());
        dealer.hand.push(deck.pop());
        context.fillStyle = "#D4AF37";
        playerY += 48;

        dealerY += 48;
        for (var i = 0; i < 2; i++) {
                dealCards(player.hand[i].suit, player.hand[i].card, "player");
                context.fillText(player.hand[i].card+" of "+player.hand[i].suit, playerX, playerY);
                playerY += 12;
                calcPoints(player, player.hand[i]);
                calcPoints(dealer, dealer.hand[i]);
        }
        if ((player.hand[0].card == "Ace" && player.hand[1].point == 10) || (player.hand[1].card == "Ace" && player.hand[0].point == 10)) {
                //player has blackjack;
                ending = "playerBlackjack";
        }
        dealCards(dealer.hand[0].suit, dealer.hand[0].card, "dealer");
        context.fillText(dealer.hand[0].card+" of "+dealer.hand[0].suit, dealerX, dealerY);
        dealerY += 12;
}
```

The above function sets up the game of blackjack according to the ruleset. It sets positions for the player and the dealer which can be changed if a different sized canvas has been used. As with other functions the stylistic choices can be changed, however the logic used in this function should remain as is as it deals with the rules of blackjack which are fairly universal.

```
function calcPoints(person, card) {
        /**
        *Calculates the points of the player and the dealer.
        */
        person.minPoints += card.point;
        if (person.points.length == 0) {
                person.points.push(person.minPoints);
        }
        else {
                for (var i = 0; i < person.points.length; i++) {
                        person.points[i] += card.point;
                }
        }
        if (card.card == "Ace") {
                var temp;
                for (var j = 0; j < person.points.length; j++) {
                        if (j == person.points.length - 1) {
                                temp = (person.points[j]+10);
                        }
                }
                person.points.push(temp);
        }
        remvPoints(person);
}

function remvPoints(person) {
        /**
        *Removes points from the player and the dealer that are over 21.
        */
        for (var i = person.points.length - 1; i >= 0; i--) {
                if (person.points[i] > 21) {
                        person.points.pop(i);
                }
        }
        if (person.points.length == 0) {
                playing = false;
                if (person == player) {
                        ending = "losing";
                        done = true;
                        getDealerToPlay();
                }
                else {
                        ending = "winning";
                        done = true;
                }
        }
}
```

These functions involve calculating the points of the player and dealer. These functions should also remain untouched as the logic used in creating them is sound and adheres to the rules of blackjack.

```
	function getDealerToPlay() {
		/**
		*Makes the dealer play blackjack, stopping at or above a soft 17.
		*/
		dealCards(dealer.hand[1].suit, dealer.hand[1].card);
		context.fillText(dealer.hand[1].card+" of "+dealer.hand[1].suit, dealerX, dealerY);
		dealerY += 12;
		if (dealer.hand[0].card == "Ace") {
			if (player.insured == true) {
				//player is insured...
				player.insured = false;
				if (dealer.hand[1].point == 10){
					//...and dealer has a blackjack
					player.money += (player.mainBet + (player.sideBet*2));
					player.mainBet = 0;
					ending = "dealerBlackjack";
					done = true;
				}
				else {
					//...dealer doesn't have a blackjack
				}
				player.sideBet = 0;
			}
		}
		if ((dealer.hand[0].card == "Ace" && dealer.hand[1].point == 10) || (dealer.hand[1].card == "Ace" && dealer.hand[0].point == 10) && done
== false) {
			//dealer has a blackjack...
			if (ending == "playerBlackjack") {
				//...and player too; push
				player.money += player.mainBet;
			}
			else {
				//...but player doesn't
				ending = "dealerBlackjack";
			}
			player.mainBet = 0;
			done = true;
		}
		else if (ending == "playerBlackjack") {
			//player has a blackjack but dealer doesn't
			player.money += ((player.mainBet*2) + (player.mainBet/2));
			player.mainBet = 0;
			done = true;
		}
		else {
```

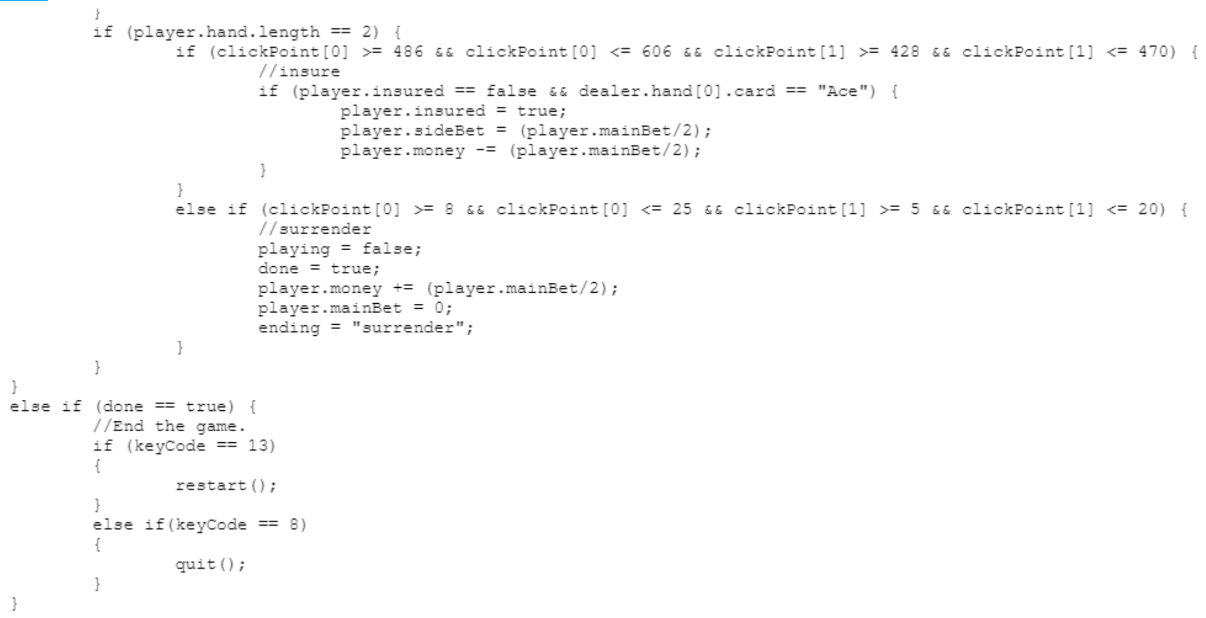


```
                    //neither party has a blackjack
                    var i = 1;
                    while (done == false && 17 > dealer.points[dealer.points.length - 1]) {
                            //get at least a soft 17
                            i++;
                            dealer.hand.push(deck.pop());
                            context.fillText(dealer.hand[i].card+" of "+dealer.hand[i].suit, dealerX, dealerY);
                            calcPoints(dealer, dealer.hand[i]);
                            dealerY += 12;
                    }
                    if ((dealer.points.length == 0 && player.points.length > 0) || Math.max.apply(null, player.points) > Math.max.apply(null,
dealer.points)) {
                            //player has more points
                            ending = "winning";
                            player.money += (player.mainBet*2);
                    }
                    else if ((player.points.length == 0 && dealer.points.length > 0) || Math.max.apply(null, player.points) < Math.max.apply(null,
dealer.points)) {
                            //player has less points
                            ending = "losing";
                    }
                    else {
                            //both parties have the same points
                            ending = "push";
                            player.money += player.mainBet;
                    }
                    player.mainBet = 0;
                    done = true;
            }
    }
```

The above two screenshots detail the function *getDealerToPlay()*. This function essentially is what controls the AI dealer and makes them play the game of blackjack. This is one of the more important functions of the entire game and as a result it is not recommended to change this function. Of course in any updated versions of the product if any variable names are changed they should be updated throughout the code, otherwise the function should be left as is.

```
function activate(event) {
        /**
        *Takes key strokes and mouse-clicks as input.
        */



        var rect = event.target.getBoundingClientRect();
        var cursorX = event.clientX - rect.left; //x position within the element.
        var cursorY = event.clientY - rect.top;  //y position within the element.


        var clickPoint = [cursorX, cursorY];
        //window.alert(clickPoint);

        var keyCode = event.keyCode;
        if (started == false) {
                //Start the game.
                if (keyCode == 13) {
                        started = true;
                }
        }
        else if (gotDeck == false) {
                //Set the number of decks used.
                if (keyCode == 49 || keyCode == 97) {
                        deckNum = 1;
                        gotDeck = true;
                }
                else if (keyCode == 50 || keyCode == 98) {
                        deckNum = 2;
                        gotDeck = true;
                }
                else if (keyCode == 51 || keyCode == 99) {
                        deckNum = 3;
                        gotDeck = true;
                }
                else if (keyCode == 52 || keyCode == 100) {
                        deckNum = 4;
                        gotDeck = true;
                }
                else if (keyCode == 53 || keyCode == 101) {
                        deckNum = 5;
                        gotDeck = true;
                }
        }
```

```
		,
		else if (madeBet == false) {
			//Set the amount betted.
			if (keyCode == 49 || keyCode == 97) {
				placeBet(betOps, 0);
			}
			else if (keyCode == 50 || keyCode == 98) {
				placeBet(betOps, 1);
			}
			else if (keyCode == 51 || keyCode == 99) {
				placeBet(betOps, 2);
			}
			else if (keyCode == 52 || keyCode == 100) {
				placeBet(betOps, 3);
			}
			else if (keyCode == 53 || keyCode == 101) {
				placeBet(betOps, 4);
			}
		}
		else if (playing == true) {
			//Play the game
			if (clickPoint[0] >= 40 && clickPoint[0] <= 158 && clickPoint[1] >= 428 && clickPoint[1] <= 470) {
				//hit
				ending = "";
				player.hand.push(deck.pop());
				context.fillText(player.hand[player.hand.length - 1].card+" of "+player.hand[player.hand.length - 1].suit, playerX,
playerY);

				dealCards(player.hand[player.hand.length - 1].suit, player.hand[player.hand.length - 1].card, "player")
				playerY += 12;
				calcPoints(player, player.hand[player.hand.length - 1]);
			}
			else if (clickPoint[0] >= 188 && clickPoint[0] <= 308 && clickPoint[1] >= 428 && clickPoint[1] <= 470) {
				//stick
				playing = false;
				getDealerToPlay();
			}
			else if (clickPoint[0] >= 339 && clickPoint[0] <= 460 && clickPoint[1] >= 428 && clickPoint[1] <= 470) {
				//double down
				if (player.money >= player.mainBet) {
					player.money -= player.mainBet;
					player.mainBet *= 2;
					player.hand.push(deck.pop());
					context.fillText(player.hand[player.hand.length - 1].card+" of "+player.hand[player.hand.length - 1].suit,
playerX, playerY);

					playerY += 12;
					calcPoints(player, player.hand[player.hand.length - 1]);
					playing = false;
					getDealerToPlay();
				}
```



```
		}
		if (player.hand.length == 2) {
			if (clickPoint[0] >= 486 && clickPoint[0] <= 606 && clickPoint[1] >= 428 && clickPoint[1] <= 470) {
				//insure
				if (player.insured == false && dealer.hand[0].card == "Ace") {
					player.insured = true;
					player.sideBet = (player.mainBet/2);
					player.money -= (player.mainBet/2);
				}
			}
			else if (clickPoint[0] >= 8 && clickPoint[0] <= 25 && clickPoint[1] >= 5 && clickPoint[1] <= 20) {
				//surrender
				playing = false;
				done = true;
				player.money += (player.mainBet/2);
				player.mainBet = 0;
				ending = "surrender";
			}
		}
	}
	else if (done == true) {
		//End the game.
		if (keyCode == 13)
		{
			restart();
		}
		else if(keyCode == 8)
		{
			quit();
		}
	}
```

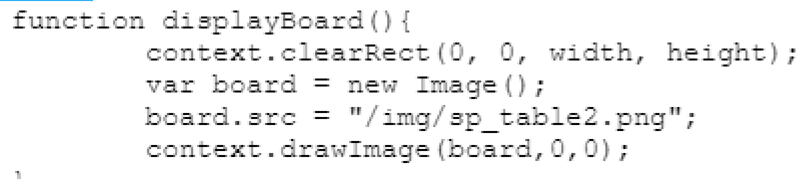
These three screenshots detail the longest function of our product, *activate()*. This function is used to register keystrokes and mouse clicks which are how the player interacts with the game. This function, like the previous one, is extremely important to the functionality of the game and again it would not be recommended to make any changes to the logic of the game. However if you wish to use different keyboard commands in any updated versions of the game you should be able to easily alter the code to register the desired keystrokes rather than the ones we have coded in. No canvas measurements are hardcoded into this section of code so the portion responsible for registering clicks will work no matter the size of the canvas and so there is no need to alter it.

```
function restart() {
	/**
	*Either resets and restarts the game or prompts the user to refresh the page is he/she wishes to play again.
	*/
	window.clearInterval(eyesID);
	clear(0, 0, width, height);
	log();
	if (player.money >= Math.min.apply(null, betOps)) {
		window.removeEventListener('keydown', activate, false);
		//Reset/Restart
		gotDeck = false;
		madeBet = false;
		tooMuch = false;
		player.hand = [];
		player.minPoints = 0;
		player.points = [];
		dealer.hand = [];
		dealer.minPoints = 0;
		dealer.points = [];
		ending = "";
		playing = false;
		done = false;
		init();
	}
	else {
		//Prompt
		context.fillStyle = "#D4AF37";
		context.font = "80px Lucida Console";
		context.fillText("Game Over!", 10, (height/4)-40);
		context.font = "12px Lucida Console";
		context.fillText("You do not have enough money to continue!", 10, (height/2)-6);
		context.fillText("Please press 'Backspace' to go to the home page or reload the page to reset everything!", 10, (height/2)+24);
	}
}
```

The function shown above is responsible for resetting the entire game if the player wishes to restart or prompts the player on what to do if they run out of money As with other functions the main changes that can be made here are in regards to stylistic choices or what messages are to be displayed. Other than those options these functions are quite basic and need not be altered in any major way.

```
function quit() {
	/**
	*Quits the user out of the game.
	*/
	window.removeEventListener('keydown', activate, false);
	window.clearInterval(eyesID);
	window.location.href = "index.php";
}
```

This function is quite basic as all it does is remove the player from the game and return them to the home screen and as such there should not be any need to alter it.



```
function displayBoard(){
        context.clearRect(0, 0, width, height);
        var board = new Image();
        board.src = "/img/sp_table2.png";
        context.drawImage(board,0,0);
}
```

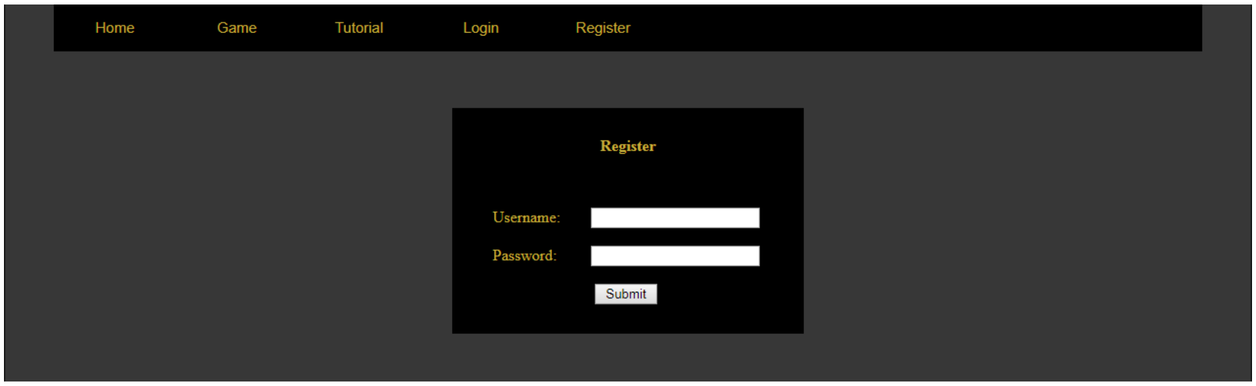
This is the function used for displaying the image we used for the game board. This function can be altered to use an image of the developers choice in any future iterations of this product.

```
function dealCards(suit, card, sender){
        //card across 84
        //card up 123

        var dest = (0,0)
        if (sender == "player"){
                dest = (273,240);
        }
        else if (sender == "dealer"){
                dest = (273, 4);
        }


        var playCard = new Image(84, 123);
        var backCard = new Image(84, 123);
        backCard.src = "/img/CardBack.png";
        holder = "/img/" + suit + "/" + card + ".png";
        playCard.src = holder;
        //window.alert(holder);

        var pos = (475, 10);

        var move = setInterval(function(){

                if(pos[0] > dest[0]){
                        pos[0] -= 10;
                }
                else if(pos[1] < dest[1]){
                        pos[1] += 10;
                }
                else if(pos[1] > dest[1]){
                        pos[1] -=1;
                }

                context.drawImage(backCard, pos[0], pos[1] , 84, 123);

                if(pos[0] == dest[0] && pos[1] == dest[1]){
                        clearInterval(move);
                }

        }, 10);

        backCard.src = holder;
}
```

This is the function responsible for displaying the images of each card in the deck. As with the function for displaying the board, any images can be used for the cards and that is realistically all that would need to be changed in this function

## Part 2: CSS

The CSS is responsible for the appearance of all the pages on our website. The CSS portion of this product is probably the most adaptable to personal taste as it doesn't directly interact with any functionality of the game, it just changes the aesthetics of the website. Below is a screenshot of the appearance of our iteration of the website just as a demonstration of the CSS.



Again, this is just the style we have chosen for our particular version, the CSS can be changed to display the website any way future developers wish.

## Part 3: SQL

The SQL is probably the smallest part of our product, yet it plays an important role. It is used to maintain a database of users which will allow users to log in at any time they wish.

```
CREATE TABLE users(
        UserId int(11) NOT NULL auto_increment,
        GameOpponent varchar(50) NOT NULL,
        GameId int(11) NOT NULL,
        GameColor varchar(11) NOT NULL,
        UserName varchar(50) NOT NULL,
        UserPassword varchar(50) NOT NULL ,
        MoveString varchar(100) NOT NULL,
        latestMove varchar(100) NOT NULL,
        PRIMARY KEY (UserId)
);

CREATE TABLE chats(
        ChatId int(11) NOT NULL auto_increment,
        ChatUserId int(11) NOT NULL,
        chatGameId int(11) NOT NULL,
        ChatText varchar(50) NOT NULL,
        PRIMARY KEY (ChatId)
);
```

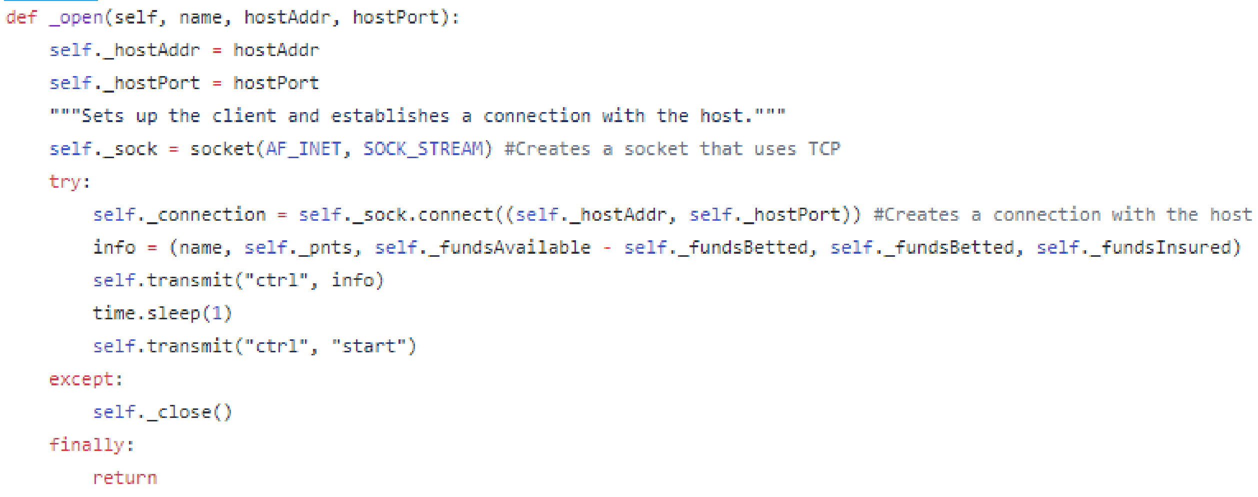
As can be seen above, we chose to only allow users to enter an ID of up 11 characters. This can be altered in order to allow users to have longer or shorter ID’s. It is recommended to leave the rest of the specification alone, particularly the *PRIMARY KEY* attribute.

## Part 4: Python

The Python we used for this product is mainly used to set up the server and client connections. It is also responsible for sending the player actions to the server so that the server can respond appropriately by changing game states.

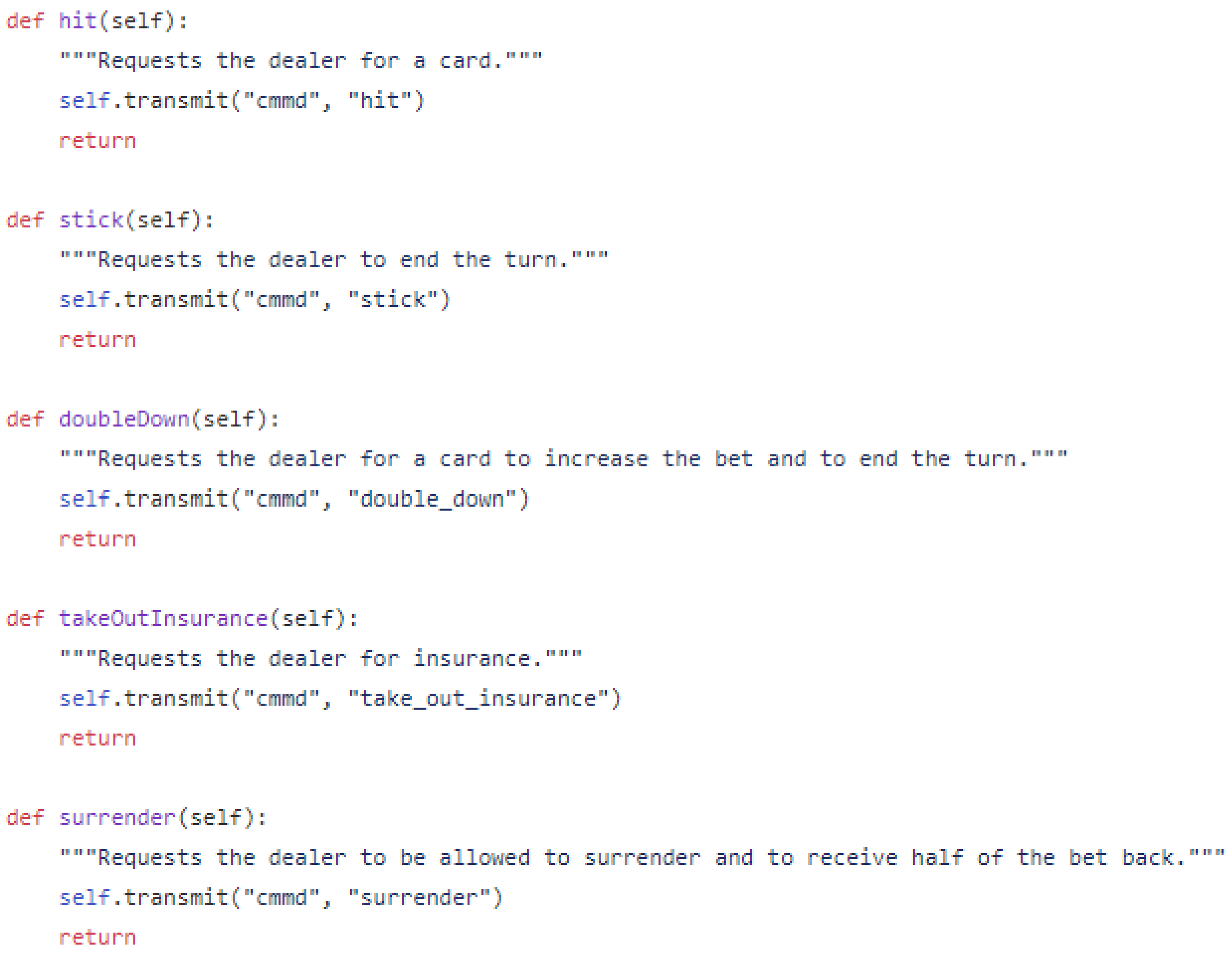
```python
#!/usr/bin/env python3

from socket import *
import time
from tkinter import *

class Client(object):

    def __init__(self):
        """Initialises the client."""
        self._addr = gethostbyname(gethostname())
        self._hostAddr = None
        self._hostPort = None
        self._pnts = []
        self._fundsAvailable = 500
        self._fundsBetted = 10
        self._fundsInsured = 0
        self._sock = None
```

This portion of code is used to import any relevant material and to initialise the client. There are small changes that can be made here such as changing the available funds or changing the variable names. Other than that this is a fairly standard way of setting up a client and should remain unaltered.



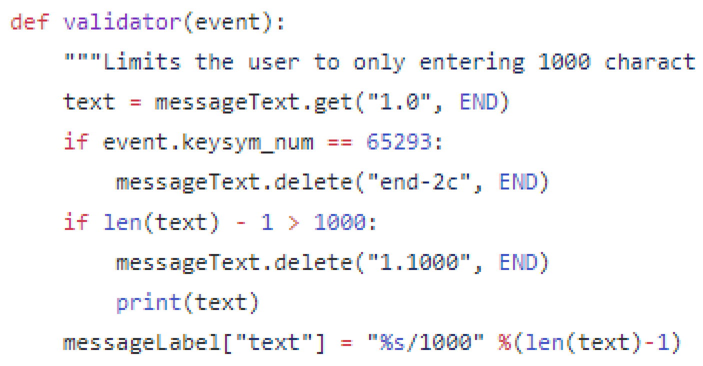
```python
def _open(self, name, hostAddr, hostPort):
    self._hostAddr = hostAddr
    self._hostPort = hostPort
    """Sets up the client and establishes a connection with the host."""
    self._sock = socket(AF_INET, SOCK_STREAM) #Creates a socket that uses TCP
    try:
        self._connection = self._sock.connect((self._hostAddr, self._hostPort)) #Creates a connection with the host
        info = (name, self._pnts, self._fundsAvailable - self._fundsBetted, self._fundsBetted, self._fundsInsured)
        self.transmit("ctrl", info)
        time.sleep(1)
        self.transmit("ctrl", "start")
    except:
        self._close()
    finally:
        return
```

This function sets up the client by assigning it the relevant info from the __*init*__ function and creating a socket that uses a TCP connection. Again this is a standard client set up in Python and has no reason to be altered in the future.



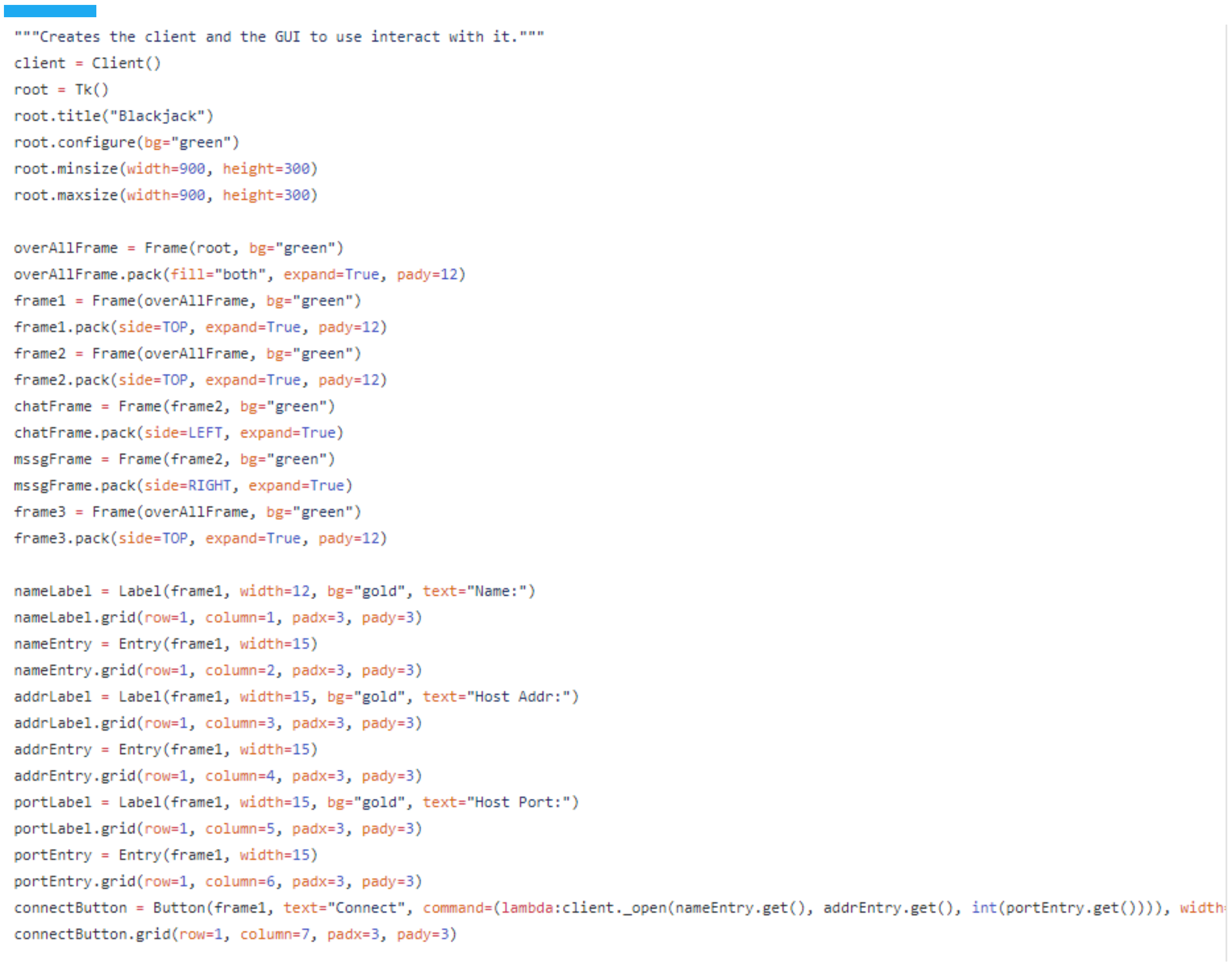
```python
def hit(self):
    """Requests the dealer for a card."""
    self.transmit("cmmd", "hit")
    return

def stick(self):
    """Requests the dealer to end the turn."""
    self.transmit("cmmd", "stick")
    return

def doubleDown(self):
    """Requests the dealer for a card to increase the bet and to end the turn."""
    self.transmit("cmmd", "double_down")
    return

def takeOutInsurance(self):
    """Requests the dealer for insurance."""
    self.transmit("cmmd", "take_out_insurance")
    return

def surrender(self):
    """Requests the dealer to be allowed to surrender and to receive half of the bet back."""
    self.transmit("cmmd", "surrender")
    return
```

This portion of the code is used for transmitting the player’s choice of action to the server. These functions are for the main actions in the game of blackjack and as such will remain the same for any iteration of the game because the rules of blackjack do not change.

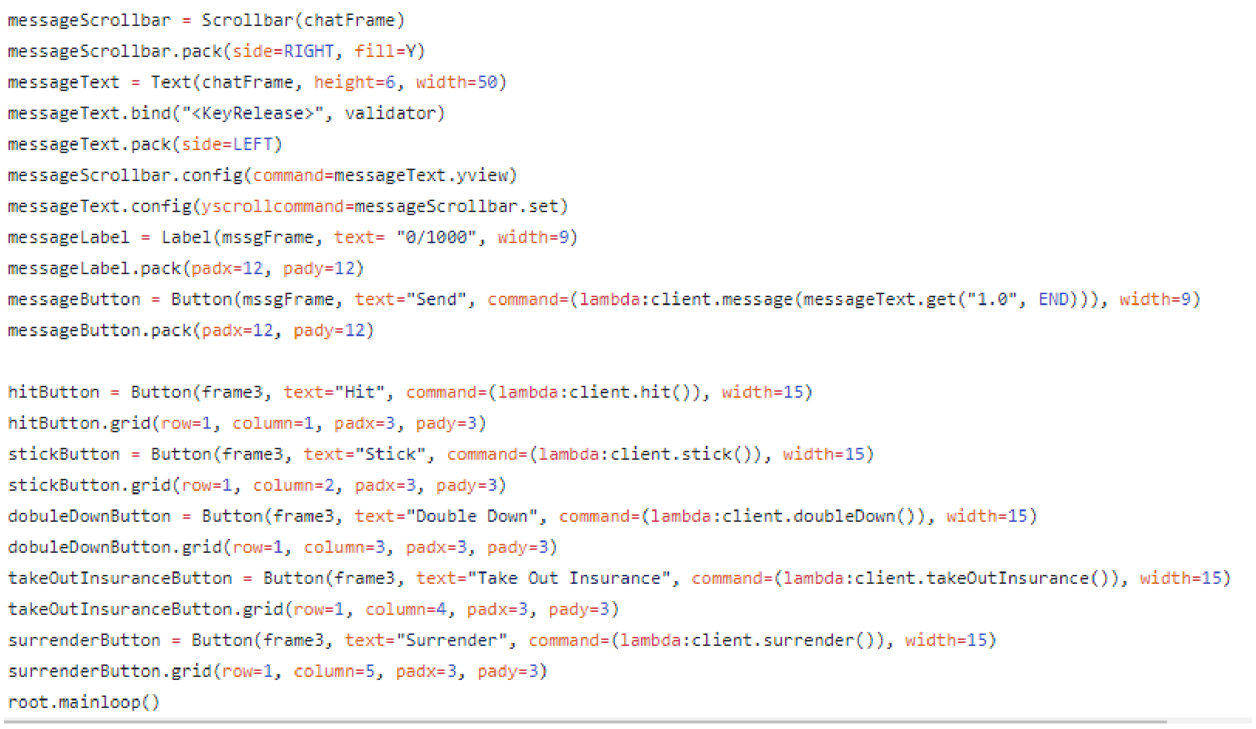


```python
def validator(event):
    """Limits the user to only entering 1000 charact
    text = messageText.get("1.0", END)
    if event.keysym_num == 65293:
        messageText.delete("end-2c", END)
    if len(text) - 1 > 1000:
        messageText.delete("1.1000", END)
        print(text)
    messageLabel["text"] = "%s/1000" %(len(text)-1)
```

The above function is used to ensure that a player can only enter up to 1,000 characters into the *messageText* and *messageLabel* fields to accurately keep track of the number of entered characters. This is the only thing that really has any reason to be altered in this function if the developer wishes to allow less or more characters to be entered into these fields.

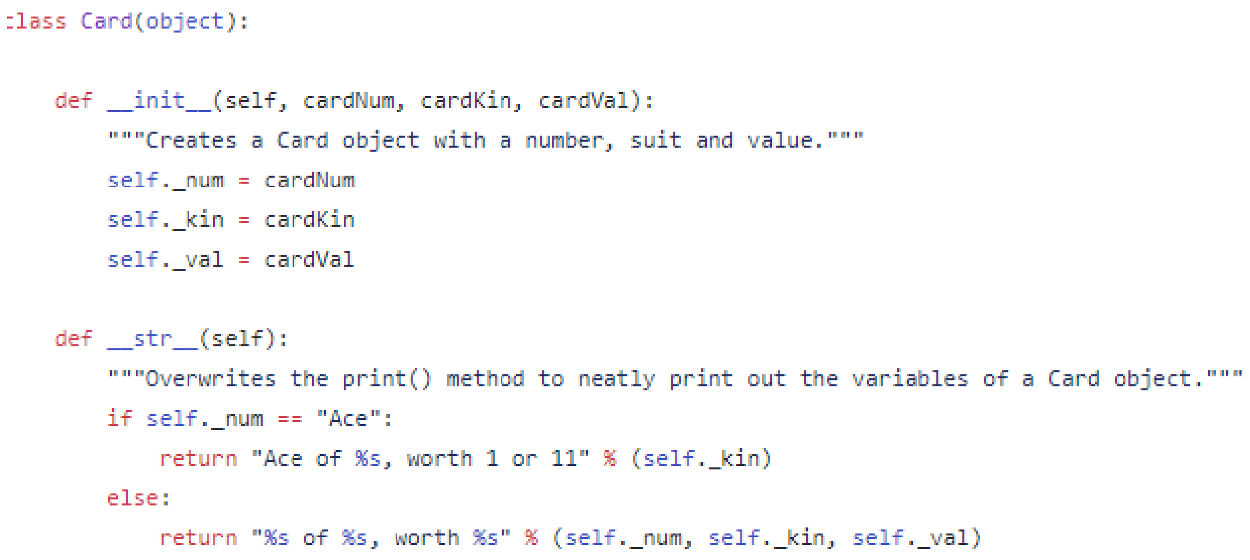


```
"""Creates the client and the GUI to use interact with it."""
client = Client()
root = Tk()
root.title("Blackjack")
root.configure(bg="green")
root.minsize(width=900, height=300)
root.maxsize(width=900, height=300)

overAllFrame = Frame(root, bg="green")
overAllFrame.pack(fill="both", expand=True, pady=12)
frame1 = Frame(overAllFrame, bg="green")
frame1.pack(side=TOP, expand=True, pady=12)
frame2 = Frame(overAllFrame, bg="green")
frame2.pack(side=TOP, expand=True, pady=12)
chatFrame = Frame(frame2, bg="green")
chatFrame.pack(side=LEFT, expand=True)
mssgFrame = Frame(frame2, bg="green")
mssgFrame.pack(side=RIGHT, expand=True)
frame3 = Frame(overAllFrame, bg="green")
frame3.pack(side=TOP, expand=True, pady=12)

nameLabel = Label(frame1, width=12, bg="gold", text="Name:")
nameLabel.grid(row=1, column=1, padx=3, pady=3)
nameEntry = Entry(frame1, width=15)
nameEntry.grid(row=1, column=2, padx=3, pady=3)
addrLabel = Label(frame1, width=15, bg="gold", text="Host Addr:")
addrLabel.grid(row=1, column=3, padx=3, pady=3)
addrEntry = Entry(frame1, width=15)
addrEntry.grid(row=1, column=4, padx=3, pady=3)
portLabel = Label(frame1, width=15, bg="gold", text="Host Port:")
portLabel.grid(row=1, column=5, padx=3, pady=3)
portEntry = Entry(frame1, width=15)
portEntry.grid(row=1, column=6, padx=3, pady=3)
connectButton = Button(frame1, text="Connect", command=(lambda:client._open(nameEntry.get(), addrEntry.get(), int(portEntry.get()))), width
connectButton.grid(row=1, column=7, padx=3, pady=3)
```



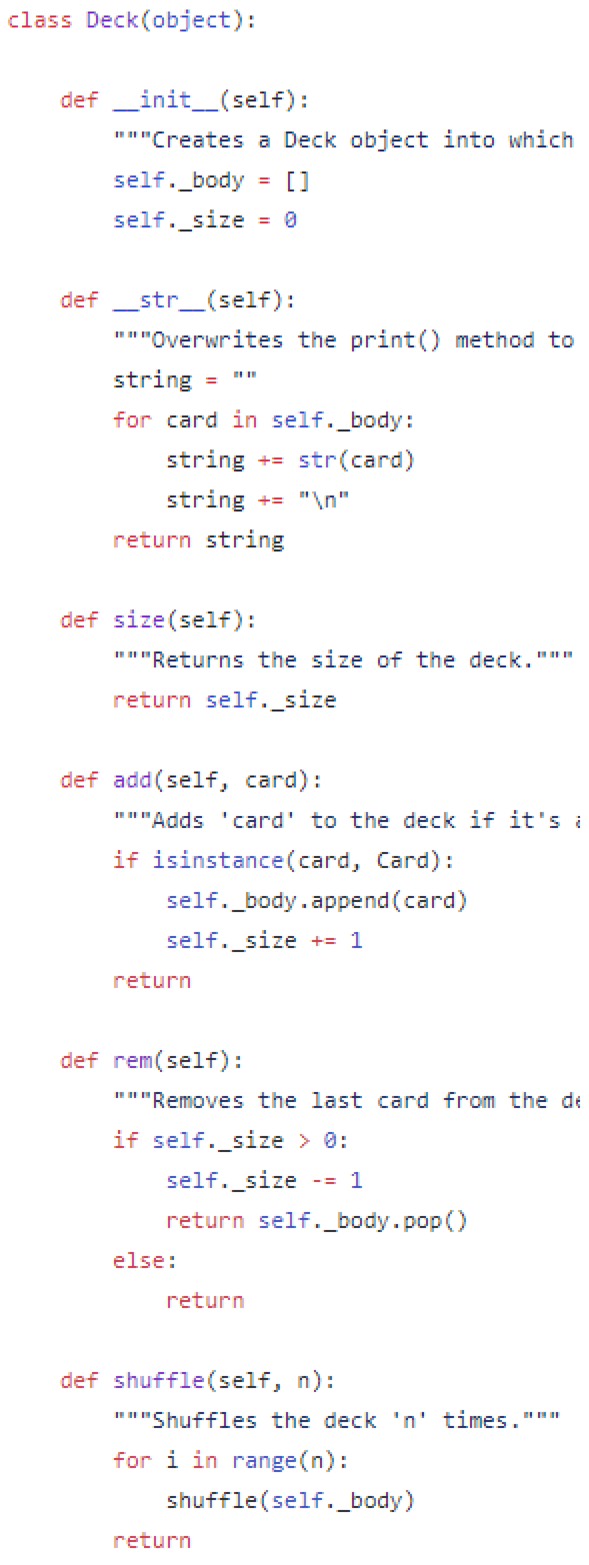
```
messageScrollbar = Scrollbar(chatFrame)
messageScrollbar.pack(side=RIGHT, fill=Y)
messageText = Text(chatFrame, height=6, width=50)
messageText.bind("<KeyRelease>", validator)
messageText.pack(side=LEFT)
messageScrollbar.config(command=messageText.yview)
messageText.config(yscrollcommand=messageScrollbar.set)
messageLabel = Label(mssgFrame, text= "0/1000", width=9)
messageLabel.pack(padx=12, pady=12)
messageButton = Button(mssgFrame, text="Send", command=(lambda:client.message(messageText.get("1.0", END))), width=9)
messageButton.pack(padx=12, pady=12)

hitButton = Button(frame3, text="Hit", command=(lambda:client.hit()), width=15)
hitButton.grid(row=1, column=1, padx=3, pady=3)
stickButton = Button(frame3, text="Stick", command=(lambda:client.stick()), width=15)
stickButton.grid(row=1, column=2, padx=3, pady=3)
dobuleDownButton = Button(frame3, text="Double Down", command=(lambda:client.doubleDown()), width=15)
dobuleDownButton.grid(row=1, column=3, padx=3, pady=3)
takeOutInsuranceButton = Button(frame3, text="Take Out Insurance", command=(lambda:client.takeOutInsurance()), width=15)
takeOutInsuranceButton.grid(row=1, column=4, padx=3, pady=3)
surrenderButton = Button(frame3, text="Surrender", command=(lambda:client.surrender()), width=15)
surrenderButton.grid(row=1, column=5, padx=3, pady=3)
root.mainloop()
```

These two portions of code are responsible for creating a GUI to interact with the client, as such it can be altered any way the developer wishes. This allows high levels of customisation of the GUI itself and allows a lot of personalisation. However, all variable names should be consistent in order to maintain integrity of the code, as always.



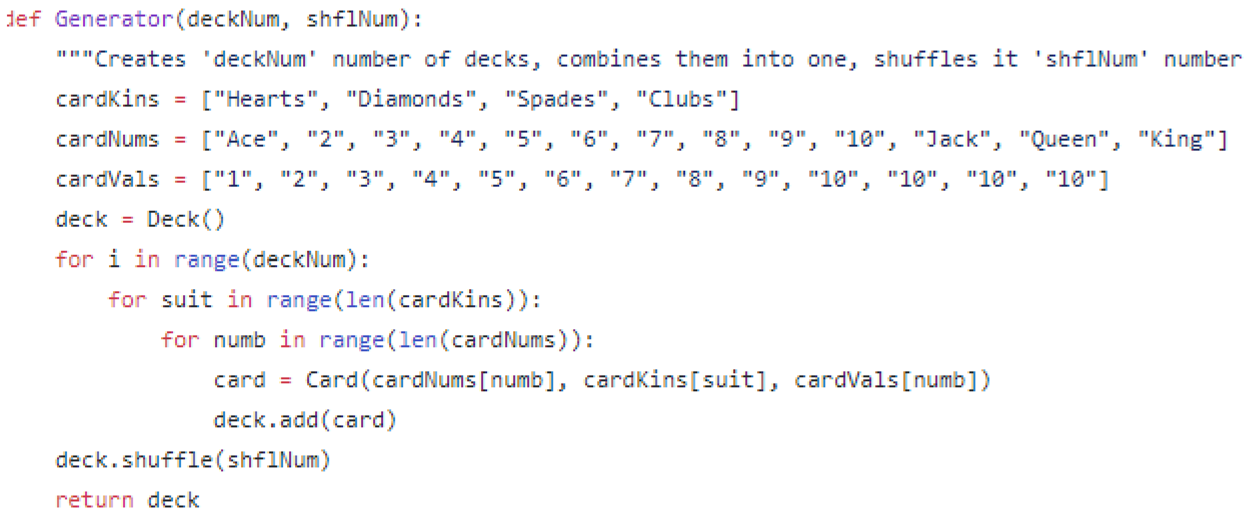
```
lass Card(object):

    def __init__(self, cardNum, cardKin, cardVal):
        """Creates a Card object with a number, suit and value."""
        self._num = cardNum
        self._kin = cardKin
        self._val = cardVal

    def __str__(self):
        """Overwrites the print() method to neatly print out the variables of a Card object."""
        if self._num == "Ace":
            return "Ace of %s, worth 1 or 11" % (self._kin)
        else:
            return "%s of %s, worth %s" % (self._num, self._kin, self._val)
```

This section of Pyrhon code is used to declare a class for a card in the deck and overwrite the *print()* method in order to print out the value of a given card. These functions should remain unchanged as the values of cards are consistent across all games of blackjack so there would never be a reason to change anything here.



```
class Deck(object):

    def __init__(self):
        """Creates a Deck object into which
        self._body = []
        self._size = 0

    def __str__(self):
        """Overwrites the print() method to
        string = ""
        for card in self._body:
            string += str(card)
            string += "\n"
        return string

    def size(self):
        """Returns the size of the deck."""
        return self._size

    def add(self, card):
        """Adds 'card' to the deck if it's
        if isinstance(card, Card):
            self._body.append(card)
            self._size += 1
        return

    def rem(self):
        """Removes the last card from the de
        if self._size > 0:
            self._size -= 1
            return self._body.pop()
        else:
            return

    def shuffle(self, n):
        """Shuffles the deck 'n' times."""
        for i in range(n):
            shuffle(self._body)
        return
```

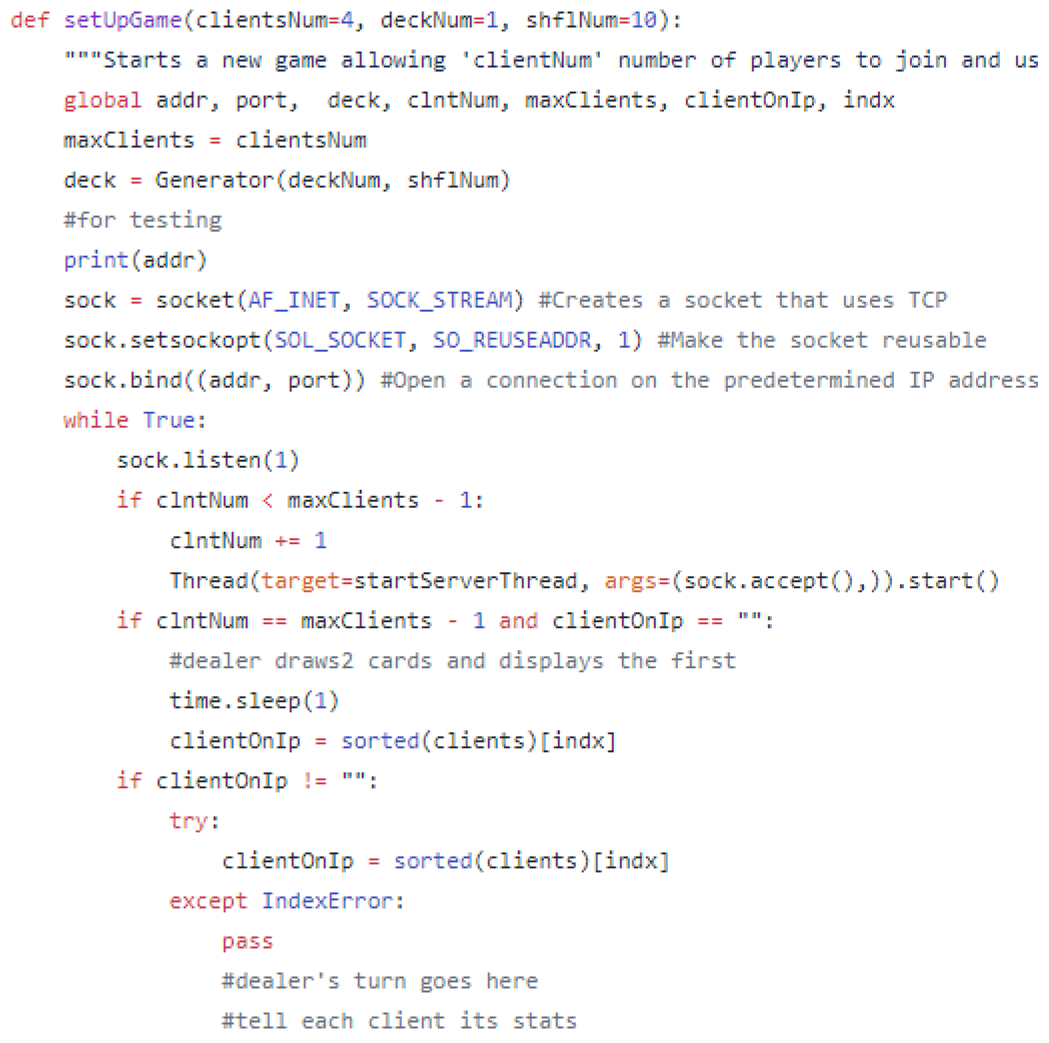
This portion of code is responsible for declaring a class to deal with the entire deck of cards, and declares some commands relating to the deck. Again these functions deal with most, if not all, things a user could want to do with a deck so there is no need to change anything. However if there was an action a developer wanted to add it would be fairly straightforward to add into the class.



```
def Generator(deckNum, shflNum):
    """Creates 'deckNum' number of decks, combines them into one, shuffles it 'shflNum' number
    cardKins = ["Hearts", "Diamonds", "Spades", "Clubs"]
    cardNums = ["Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King"]
    cardVals = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "10", "10", "10"]
    deck = Deck()
    for i in range(deckNum):
        for suit in range(len(cardKins)):
            for numb in range(len(cardNums)):
                card = Card(cardNums[numb], cardKins[suit], cardVals[numb])
                deck.add(card)
    deck.shuffle(shflNum)
    return deck
```

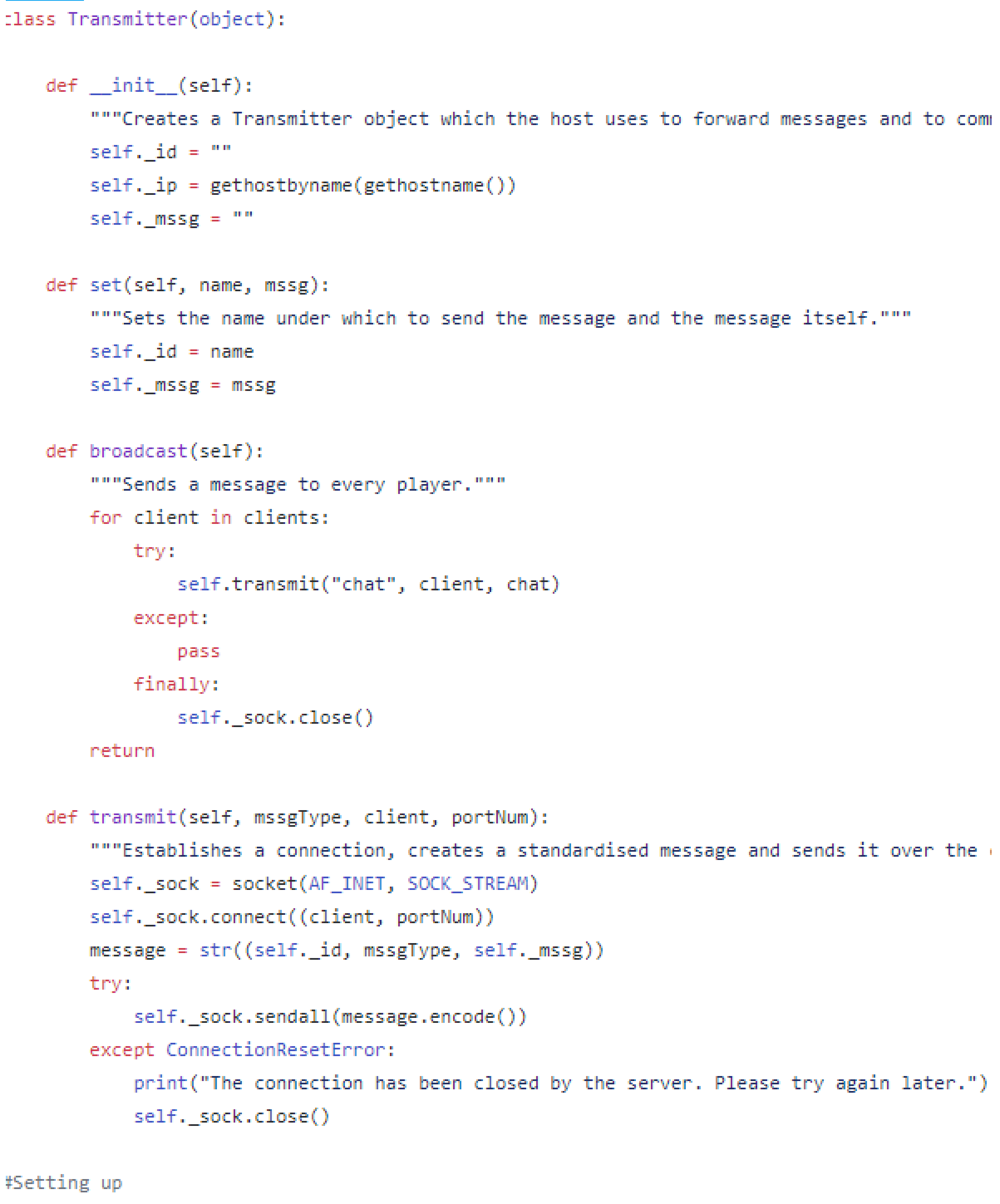
This function is used to select the number of decks a user wishes to have and how many times to shuffle those decks. The function itself has no need to be altered so the only thing that would be changed from iteration to iteration would be the number of decks or the number of times the decks get shuffled.

The host server portion of the Python code should remain largely unchanged as it is a standard set up for a host server using the Python language, however there are small portions that can be altered which will be detailed below.



```python
def setUpGame(clientsNum=4, deckNum=1, shflNum=10):
    """Starts a new game allowing 'clientNum' number of players to join and us
    global addr, port,  deck, clntNum, maxClients, clientOnIp, indx
    maxClients = clientsNum
    deck = Generator(deckNum, shflNum)
    #for testing
    print(addr)
    sock = socket(AF_INET, SOCK_STREAM) #Creates a socket that uses TCP
    sock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1) #Make the socket reusable
    sock.bind((addr, port)) #Open a connection on the predetermined IP address
    while True:
        sock.listen(1)
        if clntNum < maxClients - 1:
            clntNum += 1
            Thread(target=startServerThread, args=(sock.accept(),)).start()
        if clntNum == maxClients - 1 and clientOnIp == "":
            #dealer draws2 cards and displays the first
            time.sleep(1)
            clientOnIp = sorted(clients)[indx]
        if clientOnIp != "":
            try:
                clientOnIp = sorted(clients)[indx]
            except IndexError:
                pass
                #dealer's turn goes here
                #tell each client its stats
```

This portion of the host server setup if altered will allow you to change the number of clients that can connect. As can be seen we hard coded 4 clients in, however a developer can choose however many they so wish. The same goes for the deck number and shuffle number.



```
class Transmitter(object):

    def __init__(self):
        """Creates a Transmitter object which the host uses to forward messages and to com
        self._id = ""
        self._ip = gethostbyname(gethostname())
        self._mssg = ""

    def set(self, name, mssg):
        """Sets the name under which to send the message and the message itself."""
        self._id = name
        self._mssg = mssg

    def broadcast(self):
        """Sends a message to every player."""
        for client in clients:
            try:
                self.transmit("chat", client, chat)
            except:
                pass
            finally:
                self._sock.close()
        return

    def transmit(self, mssgType, client, portNum):
        """Establishes a connection, creates a standardised message and sends it over the
        self._sock = socket(AF_INET, SOCK_STREAM)
        self._sock.connect((client, portNum))
        message = str((self._id, mssgType, self._mssg))
        try:
            self._sock.sendall(message.encode())
        except ConnectionResetError:
            print("The connection has been closed by the server. Please try again later.")
            self._sock.close()

#Setting up
```

This class is used to set up a transmitter the host can use to communicate with the players. The main alterations that can be made here are the messages and their names. Everything else is standardised and has no reason to be changed.

## Part 5: PHP

The PHP code included in our project can largely be left as it is as it mainly focuses on the login and logout systems and registration. However some small alterations can be made.

For example, if someone wishes to add HTML content to any of the pages they must do so in the HTML_start() method in the basic.php file. If a new file is created it must have the format :

```
import basics.php
HTML_start($title)
/*
*your HTML content goes here
*/
HTML_end()
```

To change the canvas size the basic.php file also must be edited as the canvas size is stored in this file.

If any of the SQL tables are altered for some reason then the queries in the login page and the variables in the home page must be altered to coincide with the changes made.

Other than that the PHP code is all very standard and the team sees no reason for any of it to be altered