NEA

[**Analysis** 1](#_Toc63507783)

[Stakeholders 1](#_Toc63507784)

[Evaluation 1](#_Toc63507785)

[Research 1](#_Toc63507786)

[Hardware/Software requirements 1](#_Toc63507787)

[**Design** 3](#_Toc63507788)

[Decomposition 3](#_Toc63507789)

[Algorithms 4](#_Toc63507790)

[Key Variables, Data structures and Validation 5](#_Toc63507791)

[Test Plans 6](#_Toc63507792)

[**Development and Testing** 7](#_Toc63507793)

[Iterations 7](#_Toc63507794)

[**Iteration 1 (08/01/2021) -** **Repl.it - Card Jitsu Iteration 1** 7](#_Toc63507795)

[**Iteration 2 (08/01/2021) - Repl.it - Card Jitsu Iteration 2** 7](#_Toc63507796)

[**Iteration 3 (05/02/2021) - Repl.it - Card Jitsu Iteration 3** 7](#_Toc63507797)

[Success Criteria 7](#_Toc63507798)

## **Analysis**

### Stakeholders

Loise - Commissioner - Wants a card game with the correct criteria and rules

Competitive scene - Admins, Regular Users

Other competitors - Regular Users Any potential competitors who will want to play the game with one another

### Evaluation

Admins

* Need to be able to create verified accounts and verify users

Users

* Able to enter details
* Able to understand the game and the outcomes of actions visually
* Able to have user interaction when playing the game
* Play the game fairly
* Be unable to break the game or use exploits

The Programme

* Authentication
  + Able to authenticate verified players
* Game
  + Shuffle cards
  + Allow for user interaction
  + Calculates the winner and displays this
  + Needs to be enjoyable and actions need to be visually represented
* Databasing
  + Stores the top 5 names and amount of cards In external file
  + List the cards held by the winning player

### Research

### Hardware/Software requirements

* Basic computer requirements - Requirements are excessively low and computers can easily run the software needs as HTML, CSS and JS are made to run almost universally, with very minor and few exceptions

Programme

* HTML, CSS and JS - In order for the code to run based off the commands it has contained inside of it

Administrators

* HTML, CSS and JS - In order for the clients to access it, interact and be able to change settings within the HTML, use the client side of the application and have access to the server side of the application (via scoring and account management) it needs to have the basic functionality of these and also needs to be able to have values and settings changed within the server side.

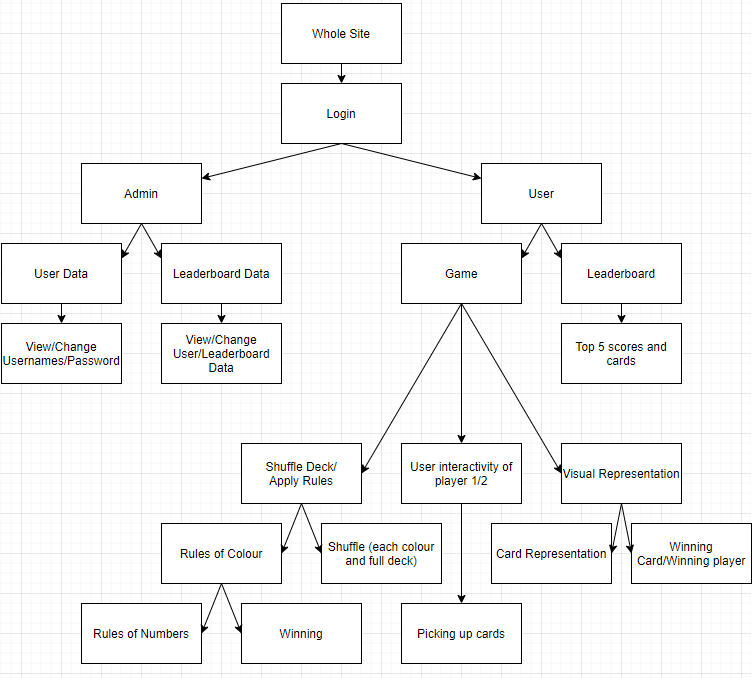
Users

* HTML, CSS and JS - In order for the end user to access it and interact with the HTML, use the client side of the application and have restricted access to the server side of the application (via scoring and account management) it needs to have the basic functionality of these.

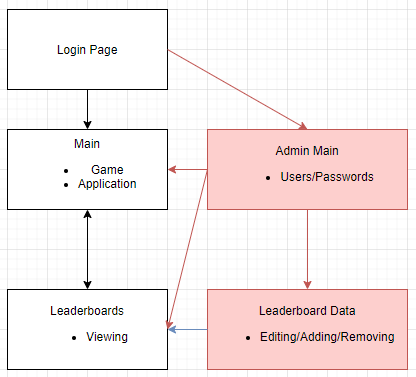
## **Design**

### Decomposition

Structure chart



Site Map



### Algorithms

**System Login**

(User and Password are textboxes, Users is the data for logins as a 2D array)

For i to Users.length

If User == Users[i][0]

If Password = Users[i][1]

Output(“Access Gained”)  
 Break

Endif

Endif

Next i

**Admin - User Data - Login Changes**

For i to Users.length

If User == Users[i][0]

Users[i][0] = Input(“Username”)

User[i][1] = Input (“Password”)

Endif

Next i

**Admin - Leaderboard Data - Leaderboard Changes**

**User - Game - Shuffle Decks**

YellowDeck[] = [“y0, y1, y2”] (etc. until “y9”)

For (i to YellowDeck.length)

YellowDeck[i].shuffle(math.random)

Next i

(same for RedDeck[] and BlackDeck[])

WholeDeck = [YellowDeck[], RedDeck[], BlackDeck[]]

For (i to WholeDeck.length)

WholeDeck[i].shuffle(math.random)

Next i

**User - Game - Rules**

Select Case Player1Card.Substring(0)

Case “r”

If (Player2Card.Substring(0) = “r”)

Neutral();

Elseif (Player2Card.Substring(0) = “y”)

P2Win();

Else

P1Win();

Endif

End Case

**User - Game - Rules - Winning**

Function Neutral()

If Player1Card.Substring(1) > Player2Card.Substring(1)

P1Win()

Else

P2Win()

Endif

End Function

Function P1Win()

Player1Cards[].push [Player1Card, Player2Card] (change Array push location for P2)

End Function

**User - Game - Visual Representation**

Function Refresh()

P1Image = (Player1Card + “.png”)

P2Image = (Player2Card + “.png”)

If WholeDeck.length =< 0

DeckImage = Null

Output (ScoreStr) (Planning to have scores as 1 large string for simplicity rather than incorporating all the variables)

Endif

End Function

**User - Game - User Interactivity**

TurnValue = 0 (TurnValue 0 is player 1, 1 is player 2, 2 is applying the rules)

(DeckPressed is button)

If DeckPressed && TurnValue == 0 (Same for P2 variables)

Player1Card = WholeDeck[0]

WholeDeck.remove[0]

TurnValue++

Endif

If TurnValue == 2

ApplyRules()

TurnValue = 0

Endif

### Key Variables, Data structures and Validation

|  |  |  |  |
| --- | --- | --- | --- |
| Variable/Data Structure | Description/Function | Justification | Validation Nessesary |
| Individual Card arrays -  “BlackCards[]”/”“RedCards[]”/“RedCards[]” | Stores the individual suits for the cards | Allows for a more robust shuffling technique as each suit is shuffled then the entire deck, also keeping the data of the cards (as CardArray data is removed) | Static (No validation necessary) |
| Whole Card CardArray | Stores all of the cards and removes them when they are used | Allows for a “physical” use of a deck where cards can be taken and removed | No values are changed but are reordered (No validation necessary) |
| Player1Card/Player2Card | Stores the card in the players hand | Allows for the card comparisons to be made and also takes the card that is “taken” from CardArray | Values are only changed by the program (No validation necessary) |
| Player1Cards[]/Player2Cards[] | Stores the won cards from the players | Allows for the score and cards in hand to be kept track of and also completes the “physical” side of the deck | Values are only changed by the program (No validation necessary) |

### Test Plans

|  |  |  |
| --- | --- | --- |
| Test | Description | Expected Outcomes |
| Login Page | Allows for an input of a username and password, having differentiated access as an admin and a general user | Login systems work robustly, no invalid logins accepted and User and Admin Logins provide different accesses |
| Game (User) | Allows for the card game to be played with another player | The game has functionality, valid rules, interaction and displays an end winner |
| Leaderboard (User) | Allows for the leaderboard to be seen and look at the top 5 players | Displays the leaderboard values (User and cards in hand) |
| Login Data (Admin) | Able to change data about the User login details and adding/removing users | Able to access and change, add and remove login details of users (user and password) |
| Leaderboard Data (Admin) | Able to change data about the Leaderboard data and adding/removing entries | Able to change add and remove leaderboard data (user and cards held) |

## **Development and Testing**

### Success Criteria

Admins

* Need to be able to create verified accounts and verify users

Users

* Able to enter details
* Able to understand the game and the outcomes of actions visually
* Able to have user interaction when playing the game
* Play the game fairly
* Be unable to break the game or use exploits

The Game Program

* Authentication
  + Able to authenticate verified players
* Game
  + Shuffle 30 cards
  + Allow for user interaction
  + Calculates the winner and displays this
  + Needs to be enjoyable and actions need to be visually represented
* Databasing
  + Stores the top 5 names and amount of cards In external file
  + List the cards held by the winning player

Note (visual clarity) :- Any information shown in the console is to be deemed not visible to the end user

### Iterations

#### **Iteration 1 (08/01/2021) -** [**Repl.it - Card Jitsu Iteration 1**](https://repl.it/@SamHarding/Card-Jitsu-Iteration-1)

**HTML – Unchanged from default**

**JS –**

//Login

//Game

//Declares Cards

let blackCards = ["b0","b1","b2","b3","b4","b5","b6","b7","b8","b9"]

let redCards = ["r0","r1","r2","r3","r4","r5","r6","r7","r8","r9"]

let yellowCards = ["y0","y1","y2","y3","y4","y5","y6","y7","y8","y9"]

//Shuffle cards

blackCards = blackCards.sort(() => Math.random() - 0.5);

redCards = redCards.sort(() => Math.random() - 0.5);

yellowCards = yellowCards.sort(() => Math.random() - 0.5);

//Declares card collection - and shuffles

let cards2D = [blackCards, redCards, yellowCards];

cards2D = cards2D.flat();

cards2D = cards2D.sort(() => Math.random() - 0.5); //Shuffles again for randomness

//Variables for the "game"

let player1Card = "";

let player1Cards = [];

let player2Card = "";

let player2Cards = [];

do{

player1Card = cards2D[0];

cards2D.shift();

player2Card = cards2D[0];

cards2D.shift();

switch(player1Card.substring(0,1)){

case "r":

if(player2Card.substring(0,1) === "r"){ //Red (Same)

console.log("neutral - down to numbers");

neutral()

}else if (player2Card.substring(0,1) === "y"){ //Yellow

console.log("player2 wins");

player2Win();

}else{ //Must be black

console.log("player1 wins")

player1Win();

}

break;

case "y":

if(player2Card.substring(0,1) === "r"){ //Red

console.log("player1 wins");

player1Win();

}else if (player2Card.substring(0,1) === "y"){ //Yellow (Same)

console.log("neutral - down to numbers");

neutral()

}else{ //Must be black

console.log("player2 wins")

player2Win();

}

break;

case "b":

if(player2Card.substring(0,1) === "r"){ //Red

console.log("player2 wins");

player2Win();

}else if (player2Card.substring(0,1) === "y"){ //Yellow

console.log("player1 wins");

player1Win();

}else{ //Must be black (Same)

console.log("neutral - down to numbers")

neutral()

}

break;

}

} while (cards2D.length > 0)

console.log("\n\n \_-\_-\_-\_ \n" + player1Cards.length + " <-> " + player2Cards.length)

if (player1Cards.length > player2Cards.length){

console.log("Player 1 Wins");

}else{

console.log("Player 2 Wins");

}

//Functions

function player1Win(){

player1Cards.push(player1Card)

player1Cards.push(player2Card)

}

function player2Win(){

player2Cards.push(player1Card)

player2Cards.push(player2Card)

}

function neutral(){

if(player1Card.substring(1,2) > player2Card.substring(1,2)){

console.log("\_ \_ \_ player1 wins");

player1Win();

}else{

console.log("\_ \_ \_ player2 wins");

player2Win();

}

}

**Notes -** The program uses console exclusively to communicate with the players and there is

no user interaction, with the game part of the project working barebones. It also stores the cards each player has in their collected pile

**Pseudocode -**

Cards = [“b1”, “r1”, “y1”, “b2”, “r2”, “y2” *(etc)* ]

Cards = Cards.shuffle()

player1Card = “”

player1Cards = []

player2Card = “”

player2Cards = []

Do

player1Card = Cards[0]

Cards[0].remove()

player2Card = Cards[0]

Cards[0].remove()

Select (player1Card.substring(0))

Case “r”

If (player2Card.substring(0) = “r”)

*Apply the correct rules and add cards to player1/2Cards[]*

Else if (player2Card.substring(0) = “y”)

*Apply the correct rules and add cards to player1/2Cards[]*

Else

*Apply the correct rules and add cards to player1/2Cards[]*

Case “y”

*Does similar for the other case statements depending on card*

Case “b”

*Does similar for the other case statements depending on card*

End select

While Cards.Length > 0

If (player1Cards.length > player2Cards.length)

Output “player 1 wins”

Else

Output “player 2 wins”

Endif

**Test Table – Iteration 1 –** The game has the basic functions but does not meet the other criteria

|  |  |
| --- | --- |
| Shuffles 30 Cards | Yes |
| User Interactivity | No interactivity |
| Calculates the winner and **Displays Cards** | No physical winner shown to players (shown in console) |
| Needs to be enjoyable and visually represented | No |
| Stores top 5 names and number of cards in an external file. | No |
| Lists the cards held by the winning player | No |

#### **Iteration 2 (08/01/2021) -** [**Repl.it - Card Jitsu Iteration 2**](https://repl.it/@SamHarding/Card-Jitsu-Iteration-2#index.html)

**HTML –**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width">

<title>repl.it</title>

<link href="style.css" rel="stylesheet" type="text/css" />

</head>

<body>

<h1>Card Game Main</h1>

<button id="deck">Deck</button> <br><br>

<p id="score">Player1 Score \_\_\_ Player2 Score \_\_\_</p>

<p>Game basics</p>

<script src="script.js">

</script>

</body>

</html>

**JS –**

//Login

//Game

let game = 0;

//HTML interactivity

//Declares Cards

let blackCards = ["b0", "b1", "b2", "b3", "b4", "b5", "b6", "b7", "b8", "b9"]

let redCards = ["r0", "r1", "r2", "r3", "r4", "r5", "r6", "r7", "r8", "r9"]

let yellowCards = ["y0", "y1", "y2", "y3", "y4", "y5", "y6", "y7", "y8", "y9"]

let player1Card = "";

let player1Cards = [];

let player2Card = "";

let player2Cards = [];

let pass = 0;

let cards2D = [];

document.getElementById("deck").addEventListener("click", function () {

switch (pass) {

case 0:

ShuffleDeck();

pass++;

console.log("case0");

break;

case 1: //Player 1 action

pass++;

player1Card = cards2D.shift(); //Shifts the array and gets rid of the last card in the deck

console.log("case1");

break;

case 2: //Player 2 action

pass++;

player2Card = cards2D.shift(); //Shifts the array and gets rid of the last card in the deck

console.log("case2");

break;

case 3: //Rule Application

RuleApply(); //Apply the rules

if (cards2D.length > 0) { //repeats the game if no cards left

pass = 1 //Sends progression backwards

}else{

pass = 4 //Sends progression forwards

}

console.log("case3");

break;

case 4:

pass++;

console.log("\n\n \_-\_-\_-\_ \n" + player1Cards.length + " <-> " + player2Cards.length)

if (player1Cards.length > player2Cards.length) {

console.log("Player 1 Wins");

} else {

console.log("Player 2 Wins");

}

console.log("case4");

break;

case 5:

if(confirm("do you want to replay?")){

alert("reloading the game");

location.reload();

}else{

alert("you can replay at any time by pressing the button");

}

break;

}

});

if (pass == 4){

alert('uh oh');

console.log("\n\n \_-\_-\_-\_ \n" + player1Cards.length + " <-> " + player2Cards.length)

if (player1Cards.length > player2Cards.length) {

console.log("Player 1 Wins");

} else {

console.log("Player 2 Wins");

}

console.log("case4");

}

//Shuffle cards

function ShuffleDeck() {

blackCards = blackCards.sort(() => Math.random() - 0.5);

redCards = redCards.sort(() => Math.random() - 0.5);

yellowCards = yellowCards.sort(() => Math.random() - 0.5);

//Declares card collection - and shuffles

cards2D = [blackCards, redCards, yellowCards];

cards2D = cards2D.flat();

cards2D = cards2D.sort(() => Math.random() - 0.5); //Shuffles again for randomness

}

function RuleApply() {

switch (player1Card.substring(0, 1)) {

case "r":

if (player2Card.substring(0, 1) === "r") { //Red (Same)

console.log("neutral - down to numbers");

neutral()

} else if (player2Card.substring(0, 1) === "y") { //Yellow

console.log("player2 wins");

player2Win();

} else { //Must be black

console.log("player1 wins");

player1Win();

}

break;

case "y":

if (player2Card.substring(0, 1) === "r") { //Red

console.log("player1 wins");

player1Win();

} else if (player2Card.substring(0, 1) === "y") { //Yellow (Same)

console.log("neutral - down to numbers");

neutral()

} else { //Must be black

console.log("player2 wins");

player2Win();

}

break;

case "b":

if (player2Card.substring(0, 1) === "r") { //Red

console.log("player2 wins");

player2Win();

} else if (player2Card.substring(0, 1) === "y") { //Yellow

console.log("player1 wins");

player1Win();

} else { //Must be black (Same)

console.log("neutral - down to numbers");

neutral();

}

break;

}

}

//Functions

function player1Win() {

player1Cards.push(player1Card)

player1Cards.push(player2Card)

}

function player2Win() {

player2Cards.push(player1Card)

player2Cards.push(player2Card)

}

function neutral() {

if (player1Card.substring(1, 2) > player2Card.substring(1, 2)) {

console.log("\_ \_ \_ player1 wins");

player1Win();

} else {

console.log("\_ \_ \_ player2 wins");

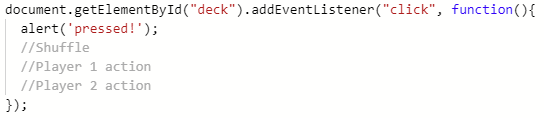
player2Win();

}

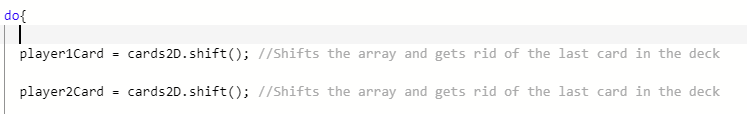
}

**Notes -** Starting to use HTML aspects within the JS code in order for the game to be physically played.

**Event listeners –** With my event listeners I wanted to make sure their functions are clear and concise. For my first event listener, on the “Deck” button, I wanted to have 3 main functions, for different contexts and times. I plan to implement them all into 1 event listener and to activate them when given the correct contexts. Given I already have the basic functionality for the game, without interactive functionality, I can easily implement the code into the correct places.

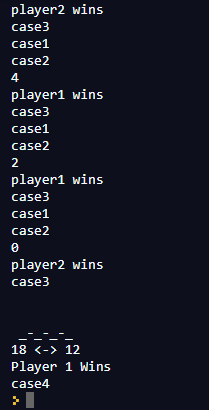
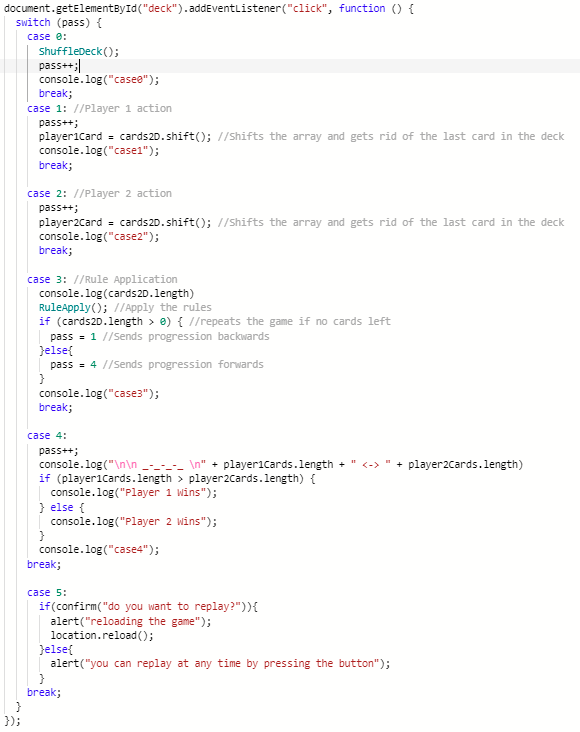


Shuffle – I plan to apply a procedure using the shuffling cards JS code



Player ½ action – I plan to apply a waiting function to the pre-existing do while loop (the start shown above) that runs the game as functionality within the 1 button can be kept this way. By applying the functionality of buttons to these 2 parts, the button can act as the player taking the card.

**Post analysis of this method -** After experimentation with this (Plan was using While loops and progression variables but they did not stop progression and instead crashed the code) I have decided instead am going to use a variable called “pass” which will control the code progression, with all of the “game code” within the event listener button Deck. Given all the interactivity is exclusive to this 1 button the functionality is not lost. This also means I need to restructure my code as I need to remove the do while loop and instead use the pass variable to make an artificial do while loop. (the new method and console testing shown below)



**Case/Pass 0 (Initiation of the game) –** Creates the deck and shuffles it.

**Case/Pass 1+2 (Player1+2 Interaction)** – Picks up the players cards and removes them from the array, done to create interaction

**Case/Pass 3 (Rules Applied) –** Applies the Rules in the function. The code for this has not changed outside the fact it is now not within the do while loop and is instead in the artificial loop I have created

**Case/Pass 4 (End results) –** Presents the end results in console form for this iteration, planning to use the HTML side to display the results in later iterations

**Case/Pass 5 (Reloading) –** Reloads the game page so that the game can be played again, being easier and more practical to do so.

**Test Table – Iteration 2 –** Better interactivity but information and databasing not used

|  |  |
| --- | --- |
| Shuffles 30 Cards | Yes |
| User Interactivity | Yes – Use of 1 central button to do all of the interactivity |
| Calculates the winner and **Displays cards** | No physical winner shown to players (shown in console) |
| Needs to be enjoyable and visually represented | UI but with no information shown on. |
| Stores top 5 names and number of cards in an external file. | No |
| Lists the cards held by the winning player | No |

#### **Iteration 3 (05/02/2021) -** [**Repl.it - Card Jitsu Iteration 3**](https://repl.it/@SamHarding/Card-Jitsu-Iteration-3#script.js)

**HTML –**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width">

<title>repl.it</title>

<link href="style.css" rel="stylesheet" type="text/css" />

</head>

<body>

<h1>Card Game Main</h1>

<button id="deck">Deck</button> <br><br>

<p id="cardhand">Player 1 Card \_\_\_ Player 2 card \_\_\_</p>

<p id="score">Player1 Score \_\_\_ Player2 Score \_\_\_</p>

<p>Game basics</p>

<script src="script.js">

</script>

</body>

</html>

**JS –**

//Login

//Variables

//Declares Cards

let blackCards = ["b0", "b1", "b2", "b3", "b4", "b5", "b6", "b7", "b8", "b9"]

let redCards = ["r0", "r1", "r2", "r3", "r4", "r5", "r6", "r7", "r8", "r9"]

let yellowCards = ["y0", "y1", "y2", "y3", "y4", "y5", "y6", "y7", "y8", "y9"]

let player1Card = "";

let player1Cards = [];

let player2Card = "";

let player2Cards = [];

//Progression Variables

let pass = 0;

let cards2D = [];

//HTML interactivity

let player1Cardtxt = "";

let player2Cardtxt = "";

document.getElementById("deck").addEventListener("click", function () {

switch (pass) {

case 0:

ShuffleDeck();

pass++;

console.log("case0");

break;

case 1: //Player 1 action

pass++;

player1Card = cards2D.shift(); //Shifts the array and gets rid of the last card in the deck

player1Cardtxt = player1Card;

document.getElementById("cardhand").innerHTML = ("Player 1 Card " + player1Card + " Player 2 Card \_\_\_");

console.log("case1");

break;

case 2: //Player 2 action

pass++;

player2Card = cards2D.shift(); //Shifts the array and gets rid of the last card in the deck

player2Cardtxt = player2Card;

document.getElementById("cardhand").innerHTML = ("Player 1 Card " + player1Card + " Player 2 Card " + player2Card);

console.log("case2");

break;

case 3: //Rule Application

RuleApply(); //Apply the rules

player1Cardtxt = "";

player2Cardtxt = "";

document.getElementById("cardhand").innerHTML = ("Player 1 Card \_\_\_ Player 2 Card \_\_\_");

document.getElementById("score").innerHTML = ("Player1 Score " + player1Cards.length + " Player2 Score " + player2Cards.length)

if (cards2D.length > 0) { //repeats the game if no cards left

pass = 1 //Sends progression backwards

}else{

pass = 4 //Sends prgression forwards

}

console.log("case3");

break;

case 4:

pass++;

console.log("\n\n \_-\_-\_-\_ \n" + player1Cards.length + " <-> " + player2Cards.length)

if (player1Cards.length > player2Cards.length) {

console.log("Player 1 Wins");

} else {

console.log("Player 2 Wins");

}

console.log("case4");

break;

case 5:

if(confirm("do you want to replay?")){

alert("reloading the game");

location.reload();

}else{

alert("you can replay at any time by pressing the button");

}

break;

}

});

if (pass == 4){

alert('uh oh');

console.log("\n\n \_-\_-\_-\_ \n" + player1Cards.length + " <-> " + player2Cards.length)

if (player1Cards.length > player2Cards.length) {

console.log("Player 1 Wins");

} else {

console.log("Player 2 Wins");

}

console.log("case4");

}

//Shuffle cards

function ShuffleDeck() {

blackCards = blackCards.sort(() => Math.random() - 0.5);

redCards = redCards.sort(() => Math.random() - 0.5);

yellowCards = yellowCards.sort(() => Math.random() - 0.5);

//Declares card collection - and shuffles

cards2D = [blackCards, redCards, yellowCards];

cards2D = cards2D.flat();

cards2D = cards2D.sort(() => Math.random() - 0.5); //Shuffles again for randomness

}

function RuleApply() {

switch (player1Card.substring(0, 1)) {

case "r":

if (player2Card.substring(0, 1) === "r") { //Red (Same)

console.log("neutral - down to numbers");

neutral()

} else if (player2Card.substring(0, 1) === "y") { //Yellow

console.log("player2 wins");

player2Win();

} else { //Must be black

console.log("player1 wins");

player1Win();

}

break;

case "y":

if (player2Card.substring(0, 1) === "r") { //Red

console.log("player1 wins");

player1Win();

} else if (player2Card.substring(0, 1) === "y") { //Yellow (Same)

console.log("neutral - down to numbers");

neutral()

} else { //Must be black

console.log("player2 wins");

player2Win();

}

break;

case "b":

if (player2Card.substring(0, 1) === "r") { //Red

console.log("player2 wins");

player2Win();

} else if (player2Card.substring(0, 1) === "y") { //Yellow

console.log("player1 wins");

player1Win();

} else { //Must be black (Same)

console.log("neutral - down to numbers");

neutral();

}

break;

}

}

//Functions

function player1Win() {

player1Cards.push(player1Card)

player1Cards.push(player2Card)

}

function player2Win() {

player2Cards.push(player1Card)

player2Cards.push(player2Card)

}

function neutral() {

if (player1Card.substring(1, 2) > player2Card.substring(1, 2)) {

console.log("\_ \_ \_ player1 wins");

player1Win();

} else {

console.log("\_ \_ \_ player2 wins");

player2Win();

}

}

**Notes –** Moving all aspects of the functionality away from the console and making sure the end user understands what is happening in the game, also meaning the game functionality is transparent.

**Test Table – Iteration 3 –** The game has the basic functions and displays basic UI (Barebones)

|  |  |
| --- | --- |
| Shuffles 30 Cards | Yes |
| User Interactivity | No interactivity |
| Calculates the winner and **Displays cards** | Winner shown to players but not cards |
| Needs to be enjoyable and visually represented | Visually represented but needs work to add realism into the game |
| Stores top 5 names and number of cards in an external file. | No |
| Lists the cards held by the winning player | No |

#### **Testing Table (Game Program)**

The uniform testing table for the game program is below

|  |  |
| --- | --- |
| Shuffles 30 Cards |  |
| User Interactivity |  |
| Calculates the winner and **Displays cards** |  |
| Needs to be enjoyable and visually represented |  |
| Stores top 5 names and number of cards in an external file. |  |
| Lists the cards held by the winning player |  |