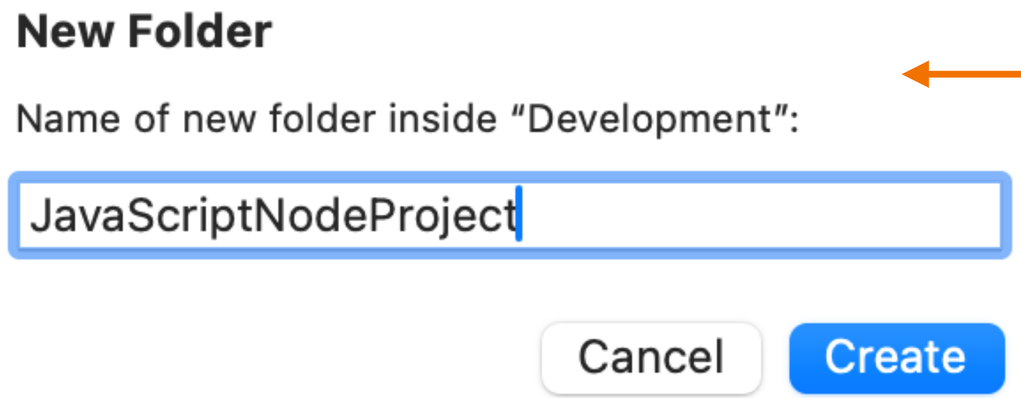
Node & JavaScript Project - How to build a simple slot game simulator!

Arya Trivedi - http://aryablogs.com

Installation (make sure you have Node and Visual Code installed)

https://nodejs.org/en/learn/getting-started/how-to-install-nodejs



First create an empty project using folder or VS Code

Then do the command “npm init” from the terminal of the VS Code within the same folder, this create package.json file.

```
JavaScriptNodeProject % npm init
```

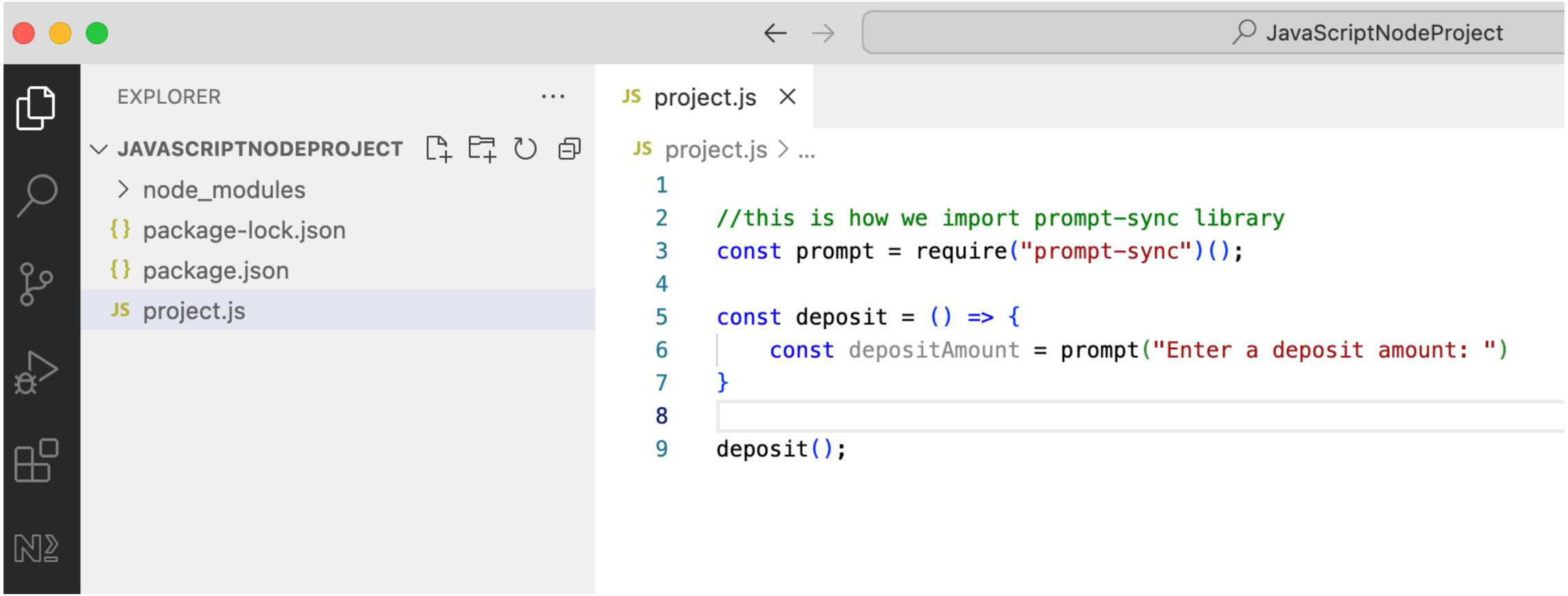
package.json

```
{
  "name": "javascriptnodeproject",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "prompt-sync": "^4.2.0"
  }
}
```

```
JavaScriptNodeProject % npm i prompt-sync

added 3 packages, and audited 4 packages in 1s
```

Next we want to install prompt-sync library, this helps to take inputs from the user. This is also done from the terminal folder.



Next create a file called "project.js" in your main directory as shown, and write the above code. The first line will import prompt-sync library. Function deposit (in EE6 format), just asks user to enter an amount. Finally we call the function, this is to test our setup.

```
JavaScriptNodeProject % node project.js
Enter a deposit amount: 200
```

To run the code, just fire the command "node project.js" to press enter, you will be asked to enter an amount, nothing else.

project.js

```
1
2   //this is how we import prompt-sync library
3   const prompt = require("prompt-sync")();
4
5   const deposit = () => {
6       const depositAmount = prompt("Enter a deposit amount: ");
7       //input is in string, convert into float
8       const numberDepositAmount = parseFloat(depositAmount);
9
10      //NaN -= Not a number, check if number entered is valid
11      if(isNaN(numberDepositAmount) || numberDepositAmount <= 0){
12          console.log("Invalid deposit, Try again!")
13      }
14  }
15
16  deposit();
17
18
```

Next, we want to make sure the amount entered is integer or decimal. And its value is more than 0. Rerun the program and test with a word or negative number as shown below.

```
Enter a deposit amount: test
Invalid deposit, Try again!

Enter a deposit amount: -10
Invalid deposit, Try again!
```

JS project.js

JS project.js > ...

```
1
2   //this is how we import prompt-sync library
3   const prompt = require("prompt-sync")();
4
5   //function to take deposit from the user
6   const deposit = () => {
7
8       while(true){
9           const depositAmount = prompt("Enter a deposit amount: ");
10          //input is in string, convert into float
11          const numberDepositAmount = parseFloat(depositAmount);
12
13          //NaN -= Not a number, check if number entered is valid
14          if(isNaN(numberDepositAmount) || numberDepositAmount <= 0){
15              console.log("Invalid deposit, Try again!")
16          } else{
17              return depositAmount; //if valid, retun deposit amount
18          }
19      }
20  }
21
22  const depositAmount = deposit();
23  console.log("Amount deposited : " + depositAmount)
```

Let's have a while loop, which will keep asking for deposit till we have a valid number.

If the data is correct, return the deposit data.

Finally capture the return and print in the console.

```
Enter a deposit amount: 100
Amount deposited : 100
```

```
//function to take number of lines user wants to play (between 1-3)
//the game requires how many lines player wants to match when the
//wheel is spinned
const getNumberofLines = () => {

    while(true){
        const lines = prompt("Enter number of lines to bet between 1 and 3 : ");
        //input is in string, convert into float
        const numberOfLines = parseFloat(lines);

        //number entered is valid, its not less than 0 and not higher than 3
        if(isNaN(numberOfLines) || numberOfLines <= 0 || numberOfLines > 3){
            console.log("Invalid number of lines, Try again!")
        } else{
            return numberOfLines; //if valid, retun deposit amount
        }
    }
}

const depositAmount = deposit();
console.log("Amount deposited : " + depositAmount)
const numberOfLines = getNumberofLines();
console.log("Number of Lines : " + numberOfLines)
```

Next we need a function using which player can enter number of lines they want to match in the slot machine.

Number of lines that can match has to between 1 and 3.

So very similar to deposit method, this method, getNumberOfLines, also check for validity of the input as a number and between 1 and 3.

```
Enter a deposit amount: 100
Amount deposited : 100
Enter number of lines to bet between 1 and 3 : 0
Invalid number of lines, Try again!
Enter number of lines to bet between 1 and 3 : adasdsa
Invalid number of lines, Try again!
Enter number of lines to bet between 1 and 3 : 4
Invalid number of lines, Try again!
Enter number of lines to bet between 1 and 3 : 3
Number of Lines : 3
```

```
//the function gets a bet amount, this has to be less
//than the balance amount
const getBet = (balance) => {

    while(true){
        const bet = prompt("Enter your bet amount : ");
        const numberOfBet = parseFloat(bet);

        //number entered is valid, its not less than 0
        //and not higher than balance amount
        if(isNaN(numberOfBet) || numberOfBet <= 0 || numberOfBet > balance){
            console.log("Invalid bet, Try again!");
        } else{
            return numberOfBet; //if valid, retun bet amount
        }
    }
}

//deposit amount can change, so it cannot change
let depositAmount = deposit();
const numberOfLines = getNumberofLines();
const bet = getBet(depositAmount);
```

Next we need a function using which player can enter a bet amount. The bet amount has to be less than the balance.

For this similar to other two methods, we take bet input number that should be less than the balance player has, this balance is passed as argument to the getBet function.

Also, change the depositAmount to let from const, because deposit amount or balance will change as player will play the game.

```
Enter a deposit amount: 100
Enter number of lines to bet between 1 and 3 : 3
Enter your bet amount : sdasdsa
Invalid bet, Try again!
Enter your bet amount : -10
Invalid bet, Try again!
Enter your bet amount : 0
Invalid bet, Try again!
Enter your bet amount : 200
Invalid bet, Try again!
Enter your bet amount : 100
```

```
//the function gets a bet amount, this has to be less
//than the balance amount. Also, we need to take of
//number of lines for each bet.
const getBet = (balance, lines) => {

    while(true){
        const bet = prompt("Enter your bet amount per line : ");
        const numberOfBet = parseFloat(bet);

        //number entered is valid, its not less than 0
        //and not higher than balance amount
        if(isNaN(numberOfBet) || numberOfBet <= 0 || numberOfBet > balance/lines){
            console.log("Invalid bet, Try again!");
        } else{
            return numberOfBet; //if valid, retun bet amount
        }
    }
}

//deposit amount can change, so it cannot change
let depositAmount = deposit();
const numberOfLines = getNumberofLines();
const bet = getBet(depositAmount, numberOfLines);
```

We have to modify getBet function to take care of bets per number of lines person has bet on.

For example, say person has initial 90$, and person wants to bet on 3 lines, so that means person cannot bet more than 30$ per line (3 * 30=90)

To take care of this use case, pass the number of lines as argument to the getBet function.

Also, divide the balance / lines and check its not greater than number of bets.

```
//global variables
const ROWS = 3; //num of rows
const COLS = 3; //num of cols

//map of symbols and how many of them in our spin wheel
const SYMBOLS_COUNT = {
    "A" : 2, //only 2 rows of A, highest value or most rare
    "B" : 4,
    "C" : 6,
    "D" : 8 //8 rows of D, lowest value or most common
}
//map of symbols and value/worth of each of them
const SYMBOLS_VALUES = {
    "A" : 5, //if wheel row gets all A's, bet win is multiplied by 5
    "B" : 4, //if wheel row gets all A's, bet win is multiplied by 4
    "C" : 3,
    "D" : 2
}
```

We need bunch of global variables, which we can declare in top of our file, these variables define the size of the wheel, count of each type and value associated with each count.

We are using alphabets A,B,C,D .. these are same as when you play in a slot machine you get three of a kind of same pictures.

Since we using a console to demonstrate this, for simplicity we are using four alphabets.

```
const spin = () => {
    const symbols = []; //array are constant reference types in JavaScript
    for (const symbol of Object.entries(SYMBOLS_COUNT)){
        console.log(symbol)
    }
}
spin();
```

```
A 2
B 4
C 6
D 8
```

Next we want to spin our wheel. For this, first thing is we need to define a constant array where we will add our global symbols. Note, in JavaScript, arrays are basically reference types. Which means you can define them as constants i.e. name cannot change, but you can add values inside.

We can modify the above code as show below as we can get a list of symbol and count.

```
const spin = () => {
    const symbols = []; //array are constant reference types in JavaScript
    for (const [symbol, count] of Object.entries(SYMBOLS_COUNT)){
        console.log(symbol, count)
    }
}
spin();
```

```
const spin = () => {
    const symbols = []; //array are constant reference types in JavaScript
    for (const [symbol, count] of Object.entries(SYMBOLS_COUNT)){
        for (let i=0; i < count; i++){
            symbols.push(symbol); //push to append to an array
        }
    }
    console.log(symbols)
}
spin();
```

```
[
  'A', 'A', 'B', 'B', 'B',
  'B', 'C', 'C', 'C', 'C',
  'C', 'C', 'D', 'D', 'D',
  'D', 'D', 'D', 'D', 'D'
]
```

Next we want to able to append symbols into our empty array. For this, lets have another for loop, this will iterate through each row, and add symbols to the array.

This is probably the most difficult function of this program.

We need to spin the wheel so we get random set of ROWS and COLS with the alphabets between A, B, C, D

```
const spin = () => {
    const symbols = []; //array are constant reference types in JavaScript
    //started by generating all possible answers and stored in symbols array
    for (const [symbol, count] of Object.entries(SYMBOLS_COUNT)){
        for (let i=0; i < count; i++){
            symbols.push(symbol); //push to append to an array
        }
    }
    // const nested array which will store random symbols
    const reels = [];  // temp array to store all reels
    for(let i = 0; i < COLS; i++){ //columns
        reels.push([]); //this will create a nested array for length of COLS, i.e. [[],[],[]]
        //a copy of symbols array from above to work upon
        const reelSymbols = [...symbols]; //copy
        for(let j=0; j < ROWS; j++) { //rows
            //Math.random() will give us a value between 0 & 1, floating pt. value
            //we multiply that index with whatever length of our symbol is to get max possible number
            //math.floor will round it to nearest whole number round down, less than
            //end of array - 1 (floor is round down )
            const randomIndex = Math.floor(Math.random() * reelSymbols.length);
            const selectedSymbol = reelSymbols[randomIndex];
            reels[i].push(selectedSymbol);
            //slice and remove one element, randomIndex is index we are
            //removing position, so we don't select again while we
            //generating the wheel
            reelSymbols.splice(randomIndex,1); //remove 1 element to avoid duplicates
        }
    }
    return reels;
}

const reels = spin();
console.log(reels);
```

```
const reels = spin();

// the spin output will be like [[A,B,C],[D,D,C],[A,B,A]] (random)
// this will rows are as shown below
// [A, D, A]
// [B, D, B]
// [C, C, A]
// above is called transposing a matrix or 2d array

const transpose = (reels) => {
    const rows = [];
    for(let i = 0; i < ROWS; i++){
        rows.push([]);
        for(let j=0; j < COLS; j++){
            rows[i].push(reels[j][i]);
        }
    }
    return rows;
}

console.log("reels : ", reels);
console.log("transpose : ", transpose(reels));
```

```
reels :  [ [ 'B', 'B', 'D' ], [ 'A', 'D', 'D' ], [ 'C', 'A', 'C' ] ]
transpose :  [ [ 'B', 'A', 'C' ], [ 'B', 'D', 'A' ], [ 'D', 'D', 'C' ] ]
```

Next we need to write a transpose function which will basically convert the array of arrays into rows and columns.

Notice how element[col][row] approach is used to build a row.

Each reel is generated possible set of symbols. These reels are transposed into array of elements.

```
// build a piped out like { "A" | "B" | "C"}
const printRows = (rows) => { //pass array rows
    for(const row of rows){
        let rowString = ""; //empty string
        for(const [i, symbol] of row.entries()){
            rowString += symbol; //append
            if(i != row.length - 1){ //till end of array
                rowString += " | "; // add a pipe to non last element
            }
        }
        console.log(rowString) // this will print every individual to row string
    }
}

//deposit amount can change, so it cannot change
let depositAmount = deposit();
const numberOfLines = getNumberofLines();
const bet = getBet(depositAmount, numberOfLines);
const reels = spin();
const rows = transpose(reels);
printRows(rows);
```

Next we need a method to properly print the rows and cols of the array.

Here we iterate through the rows we get from transpose function. For each row, we create an sting and append the symbols. Between symbols we are adding | to make output look nicer.

```
Enter a deposit amount: 100
Enter number of lines to bet between 1 and 3 : 1
Enter your bet amount per line : 1
B | C | C
B | A | D
A | D | D
```

```
// function to find players winnings
const getWinnings = (rows, bet, lines) => {
    let winnings = 0; //initialize
    for (let row = 0; row < lines; row++){
        const symbols = rows[row];
        let allSame = true;
        for (const symbol of symbols){
            if(symbol != symbols[0]){ //if first symbol matches, all matches
                allSame = false;
                break;
            }
        }
        if(allSame){//player won
            //get the value associated with that specific symbol
            winnings += bet * SYMBOLS_VALUES[symbols[0]];
        }
    }
    return winnings;
}

//deposit amount can change, so it cannot change
let depositAmount = deposit();
const numberOfLines = getNumberofLines();
const bet = getBet(depositAmount, numberOfLines);
const reels = spin();
const rows = transpose(reels);
printRows(rows);
const winnings = getWinnings(rows,bet,numberOfLines);
console.log("You won, $" + winnings.toString());
```

Next we need a method to get winnings value. For this, we rows, bet amount and number of lines players has betted on.

Iterate through the spinning row and if any of the line is all three same, get the value associated for that symbol and add to the winnings.

In the end return the total winnings.

Below shown one such winnings.

```
Enter a deposit amount: 100
Enter number of lines to bet between 1 and 3 : 3
Enter your bet amount per line : 10
D | D | C
C | C | A
D | D | D
20
You won, $20
```

```
//finally we can have a game function
//which iterate though till player is broke
//or wants to stop the game.
const game = () => {

    //deposit amount can change, so it cannot change
    let depositAmount = deposit();

    while(true){
        console.log("Current Balance $" + depositAmount);
        const numberOfLines = getNumberofLines();
        const bet = getBet(depositAmount, numberOfLines);
        depositAmount -= bet * numberOfLines;
        const reels = spin();
        const rows = transpose(reels);
        printRows(rows);
        const winnings = getWinnings(rows,bet,numberOfLines);
        depositAmount += winnings;
        console.log("You won, $" + winnings.toString());

        if(depositAmount <= 0){
            console.log("You ran out of money!");
            break;
        }
        const playAgain = prompt("Do you want to play again(y/n)? ");
        if(playAgain != "y") break;
    }
    console.log("Final Balance $" + depositAmount);
}

//finally call the game function.
game();
```

Finally, we need to make the game iterative, for this, we build game function, which will check if the balance player is valid to play the game.

The program will continuously play, till the user quits or the balance becomes zero!

```
Enter a deposit amount: 100
Current Balance $100
Enter number of lines to bet between 1 and 3 : 1
Enter your bet amount per line : 10
D | C | A
D | B | D
D | D | D
You won, $0
Do you want to play again(y/n)? y
Current Balance $90
Enter number of lines to bet between 1 and 3 : 2
Enter your bet amount per line : 15
B | C | D
D | D | D
D | B | D
You won, $30
Do you want to play again(y/n)? y
Current Balance $90
Enter number of lines to bet between 1 and 3 : 3
Enter your bet amount per line : 5
D | C | C
B | D | D
C | D | A
You won, $0
Do you want to play again(y/n)? n
Final Balance $75
```

Full Code

project.js

project.js > SYMBOLS_COUNT

```
/*
    Author : Arya Trivedi
    Date : 04/29/2024
    Program : Slot Machine Simulator using JavaScript.

    Description: The program depicts a slot machine. The UI is prompt based, shows
    how to take input from user. The user will start with an amount they want to bet
    upon. Number of lines they want to bet and amount per line.

    The program will then spin the wheel based upon number of rowxcolumns. For simplicity
    we are keeping 3x3 grid. You can change the rows to as many as you like.

    The grid will be transposed and print a random result. If the symbols match on a
    line, the player wins. Else player looses and the balance is accordingly adjusted.

    The user can play again or quit the game. If the deposited amount becomes zero,
    the program will quite automatically saying all the initial balance is lost.
*/

//this is how we import prompt-sync library
const prompt = require("prompt-sync")();

//global variables
const ROWS = 3; //num of rows
const COLS = 3; //num of cols

```

Full Code

```
//map of symbols and how many of them in our spin wheel
const SYMBOLS_COUNT = {
    "A" : 2, //only 2 rows of A, highest value or most rare
    "B" : 4,
    "C" : 6,
    "D" : 8 //8 rows of D, lowest value or most common
}
//map of symbols and value/worth of each of them
const SYMBOLS_VALUES = {
    "A" : 5, //if wheel row gets all A's, bet win is multiplied by 5
    "B" : 4, //if wheel row gets all A's, bet win is multiplied by 4
    "C" : 3,
    "D" : 2
}

//function to take deposit from the user
const deposit = () => {

    while(true){
        const depositAmount = prompt("Enter a deposit amount: ");
        //input is in string, convert into float
        const numberDepositAmount = parseFloat(depositAmount);

        //NaN -= Not a number, check if number entered is valid
        if(isNaN(numberDepositAmount) || numberDepositAmount <= 0){
            console.log("Invalid deposit, Try again!");
        } else{
            return depositAmount; //if valid, retun deposit amount
        }
    }
}

//function to take number of lines user wants to play (between 1-3)
//the game requires how many lines player wants to match when the
//wheel is spinned
const getNumberofLines = () => {

    while(true){
        const lines = prompt("Enter number of lines to bet between 1 and 3 : ");
        //input is in string, convert into float
        const numberOfLines = parseFloat(lines);

        //number entered is valid, its not less than 0 and not higher than 3
        if(isNaN(numberOfLines) || numberOfLines <= 0 || numberOfLines > 3){
            console.log("Invalid number of lines, Try again!");
        } else{
            return numberOfLines; //if valid, retun deposit amount
        }
    }
}

//the function gets a bet amount, this has to be less
//than the balance amount. Also, we need to take of
//number of lines for each bet.
const getBet = (balance, lines) => {

    while(true){
        const bet = prompt("Enter your bet amount per line : ");
        const numberOfBet = parseFloat(bet);

        //number entered is valid, its not less than 0
        //and not higher than balance amount
        if(isNaN(numberOfBet) || numberOfBet <= 0 || numberOfBet > balance/lines){
            console.log("Invalid bet, Try again!");
        } else{
            return numberOfBet; //if valid, retun bet amount
        }
    }
}

```

Full Code

```
// the funcction will simulate spinning of a wheel.

const spin = () => {
    const symbols = []; //array are constant reference types in JavaScript
    //started by generating all possible answers and stored in symbols array
    for (const [symbol, count] of Object.entries(SYMBOLS_COUNT)){
        for (let i=0; i < count; i++){
            symbols.push(symbol); //push to append to an array
        }
    }
    // const nested array which will store random symbols
    const reels = [];  // temp array to store all reels
    for(let i = 0; i < COLS; i++){ //columns
        reels.push([]); //this will create a nested array for length of COLS, i.e. [[],[],[]]
        //a copy of symbols array from above to work upon
        const reelSymbols = [...symbols]; //copy
        for(let j=0; j < ROWS; j++) { //rows
            //Math.random() will give us a value between 0 & 1, floating pt. value
            //we multiply that index with whatever length of our symbol is to get max possible number
            //math.floor will round it to nearest whole number round down, less than
            //end of array - 1 (floor is round down )
            const randomIndex = Math.floor(Math.random() * reelSymbols.length);
            const selectedSymbol = reelSymbols[randomIndex];
            reels[i].push(selectedSymbol);
            //slice and remove one element, randomIndex is index we are
            //removing position, so we don't select again while we
            //generating the wheel
            reelSymbols.splice(randomIndex,1); //remove 1 element to avoid duplicates
        }
    }
    return reels;
}
```

Full Code

```
128
129   // the spin output will be like [[A,B,C],[D,D,C],[A,B,A]] (random)
130   // this will rows are as shown below
131   // [A, D, A]
132   // [B, D, B]
133   // [C, C, A]
134   // above is called transposing a matrix or 2d array
135   const transpose = (reels) => {
136       const rows = [];
137       for(let i = 0; i < ROWS; i++){
138           rows.push([]);
139           for(let j=0; j < COLS; j++){
140               rows[i].push(reels[j][i]);
141           }
142       }
143       return rows;
144   }
145
146   // build a piped out like { "A" | "B" | "C"}
147   const printRows = (rows) => { //pass array rows
148       for(const row of rows){
149           let rowString = ""; //empty string
150           for(const [i, symbol] of row.entries()){
151               rowString += symbol; //append
152               if(i != row.length - 1){ //till end of array
153                   rowString += " | "; // add a pipe to non last element
154               }
155           }
156           console.log(rowString) // this will print every individual to row string
157       }
158   }
159
160   // function to find players winnings
161   const getWinnings = (rows, bet, lines) => {
162       let winnings = 0; //initialize
163       for (let row = 0; row < lines; row++){
164           const symbols = rows[row];
165           let allSame = true;
166           for (const symbol of symbols){
167               if(symbol != symbols[0]){ //if first symbol matches, all matches
168                   allSame = false;
169                   break;
170               }
171           }
172           if(allSame){//player won
173               //get the value associated with that specific symbol
174               winnings += bet * SYMBOLS_VALUES[symbols[0]];
175           }
176       }
177       return winnings;
178   }
```

Sample Running

```
JavaScriptNodeProject % node project.js
Enter a deposit amount: 100
Current Balance $100
Enter number of lines to bet between 1 and 3 : 2
Enter your bet amount per line : 5
D | D | D
C | C | D
A | C | D
You won, $10
Do you want to play again(y/n)? y
Current Balance $100
Enter number of lines to bet between 1 and 3 : 3
Enter your bet amount per line : 10
D | C | D
C | D | D
D | A | D
You won, $0
Do you want to play again(y/n)? y
Current Balance $70
Enter number of lines to bet between 1 and 3 : 1
Enter your bet amount per line : 1
B | D | C
C | A | D
D | D | C
You won, $0
Do you want to play again(y/n)? nn
Final Balance $69
```

Thank you !!!