Módulo 6 - Algoritmos III - Práctica

Dado trucado

Este ejercicio es un caso práctico sobre como hacer un generador aleatorio pero aplicando distintas probabilidades a los resultados.

Alternativa rápida

Vamos a partir del caso general de generador con múltiples opciones. Si quisieramos hacer un generador para simular la tirada de un dado, aplicando la fórmula general tendríamos:

```
var rollDice = () => Math.floor(Math.random() * 6) + 1;
```

Representándolo gráficamente tendríamos:

```
0     1     2     3     4     5     6
[-----|-----|-----|-----|-----|-----)
   0     1     2     3     4     5     Math.floor()
   1     2     3     4     5     6     + 1
```

Todas las tiradas con la misma probabilidad, 1/6. ¿Y si quisieramos favorecer más a unos números que a otros? Por ejemplo, aumentemos la probabilidad de obtener un 6 hasta el 50%. Así, tendremos un 50% de probabilidades de obtener un 6, y el otro 50% repartido al resto de números. Para ello vamos a construirnos un array con las posibles tiradas, pero repetiendo aquellas que nos interesan favorecer:

```
var results = [1, 2, 3, 4, 5, 6, 6, 6, 6, 6];
var rollDice = () => results[Math.floor(Math.random() * 10)];
```

Gráficamente tendríamos:

```
0     1     2     3     4     5     6     7     8     9    10
[-----|-----|-----|-----|-----|-----|-----|-----|-----|-----)
   1     2     3     4     5     6     6     6     6     6
```

Alternativa precisa

Sin embargo, si queremos tener un control total y exacto de las probabilidades debemos enfocar el problema al revés. Es decir:

1. Codificamos las distintas probabilidades que queremos para cada número del dado en un array de 6 posiciones. **La probabilidad total debe sumar 100**.

```
[5, 5, 5, 10, 10, 65]; // Num 1 = 5%, Num 2 = 5%, ... Num 6 = 65%
```

2. A partir de el obtenemos un array con las probabilidades acumuladas. Esto es, este array representará los intervalos que asignamos a cada número del dado. Dichos intervalos son proporcionales a su probabilidad.

```
       0  5 10 15   25   35                        100
Interv: [--|--|--|----|----|-------------------------)
Probs:   5% 5% 5%  10%  10%           65%
Dado:    1  2  3    4    5             6
```

3. Finalmente, para simular una tirada obtenemos un número aleatorio entre 0 y 100 y lo mapeamos a los intervalos de probabilidad, de donde a su vez obtendremos el número del dado.

```
var diceProbs = [5, 5, 5, 10, 10, 65]; // DEBEN SUMAR 100

var calcAccumulatedProbs = prob => {
  var accProb = [];
  for (var i = 0, acc = 0; i < prob.length; i++) {
    acc += prob[i];
    accProb.push(acc);
  }
  return accProb;
};

var accProbs = calcAccumulatedProbs(diceProbs);
console.log(accProbs); // [5, 10, 15, 25, 35, 100]

var rollDice = () => {
  var rnd = Math.random() * 100; // [0, 100);
  for (var i = 0; i < accProbs.length; i++) {
    if (rnd <= accProbs[i]) return i + 1;
  }
};

setInterval(() => {
  console.log(`${Date.now()} |`, rollDice());
}, 400);
```

Triángulo de Billar

Este es un ejercicio práctico en el que combinaremos distintos apartados dados en la teoría.

Se trata generar en orden aleatorio las bolas de billar (numeradas del 1 al 15) y posteriormente dibujar por consola el triángulo de salida de la mesa, acomodando las bolas de menor a mayor.

* Primero, crearemos un array vacío donde iremos almacenando las bolas con la que trabajaremos:

```
var balls = [];
```

* Ahora crearemos nuestra función generadora de bolas desordenadas:

```
var generateUnsortedBalls = array => {
  while (array.length < 15) {
    var newBall = Math.floor(Math.random() * 15) + 1;
    if (array.indexOf(newBall) === -1) {
      array.push(newBall);
    }
  }

  return array;
};
```

Vamos a probar que efectivamente nos devuelve un array de 15 elementos desordenados.

```
console.log("**** Array de bolas desordenadas: ", generateUnsortedBalls(balls));
```

* Seguidamente, vamos a crear una función que imprima un triángulo de billar por consola.

```
var printTriangle = array => {
  for (var row = 1, index = 0; index < array.length; row++) {
    var show = "";
    for (var ballToShow = 1; ballToShow <= row; ballToShow++) {
      show += ` ${array[index++]}`;
    }
    console.log(show);
  }
};
```

Vamos a ver cómo imprime nuestro array desordenado.

```
printTriangle(balls);

/**
 13
 12 7
 5 14 10
 2 8 11 3
 15 6 1 9 4
*/
```

* Posteriormente debemos ordenarlas para poder montar nuestro triángulo de billar. Yo usaré el algoritmo de burbuja pero puede ser cualquier otro de ordenación.

```
var swap = (array, a, b) => {
  var temp = array[a];
  array[a] = array[b];
  array[b] = temp;
};

var bubbleSort = array => {
  var size = array.length;
  for (var temp = 1; temp < size; temp++) {
    for (var left = 0; left < size - temp; left++) {
      var right = left + 1;
      if (array[left] > array[right]) {
        swap(array, left, right);
      }
    }
  }

  return array;
};
```

Y comprobamos que funciona correctamente

```
printTriangle(bubbleSort(balls));
```