Tarea 1
Programación Funcional

Julio Andrade
Gerardo Crot
Mª Josefa Espinoza

Tabla de contenidos

01
Funcionalidades del juego

Cálculo de puntajes

Se usó la librería lodash con encadenamiento para aprovechar sus opciones de manejo de arrays

Cambios de turnos

Se implementó un iterador para modificar el turno de a qué jugador le toca

calculator.js

```
export function* iterator2(i, n_players){
  while(true){
    yield ((i++)%n_players);
  }
}
```

initGame.js

```
const enterPlayers = async (players_array, n_players) => {
  if (n_players === 0) {
    // Here is stored the last players_array state
    console.log("¡Que comienze el juego!");
    const iterator = iterator2(0, players_array.length);
    return game(players_array, iterator);
  }

  await beginDialog(1, players_array, n_players);
};
```

Parseo de inputs

Todas las variables eran inmutables y entregamos nuevas versiones de estas si las teníamos que modificar

Parsers.js

```
export const playParser = (play) => {
  let play_array = play.replace(/^\[|\]$/, "").split(',')
  if (play_array.length === 2){

    return _.map(play_array, (number) => parseInt(number));
  };
  return play_array[0];
}
```

Obtención de inputs

Se usó la librería readline para generar asíncronamente consolas para obtener el input y continuar con el flujo del juego

initGame.js

```
export const beginDialog = async (id, players_array, n_players) => {
  //codigo para dialogo por consola
  // ID = 0 -> pedir numero de jugadores
  // ID = 1 -> pedir nombre jugadores
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });

  if (id === 0) {
    rl.question("Ingresa el número de jugadores:\n", (number) => {
      if (!checkTwoOrMorePlayers(number)) {
        console.log("Deben ser al menos dos jugadores, ¡Hasta la próxima!");
        return rl.close();
      }
      rl.close();
      enterPlayers(players_array, parseInt(number));
    });
  }

  if (id === 1) {
    rl.question("Ingresar nombre jugador:\n", (player_name) => {
      addPlayers(players_array, player_name);
      console.log("¡Hola " + player_name + "! tu puntaje inicial es 501\n");
      rl.close();
      enterPlayers(players_array, n_players - 1);
    });
  }
};
```

02
Ventajas del paradigma

Ventajas ofrecidas por el paradigma

Separación de trabajos

Más facilidades para dividirnos la tarea y sus funcionalidades

Modularización más simple

Se simplificó el esfuerzo de modularización y de reorganización del código

03
Problemas encontrados

Asincronía de los inputs

Al usar dialogs para obtener los inputs la asincronía nos impidió manejar los datos como queríamos inicialmente

initGame.js

```
export const beginDialog = async (id, players_array, n_players) => {
  //codigo para dialogo por consola
  // ID = 0 -> pedir numero de jugadores
  // ID = 1 -> pedir nombre jugadores
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });

  if (id === 0) {
    rl.question("Ingresa el número de jugadores:\n", (number) => {
      if (!checkTwoOrMorePlayers(number)) {
        console.log("Deben ser al menos dos jugadores, ¡Hasta la próxima!");
        return rl.close();
      }
      rl.close();
      enterPlayers(players_array, parseInt(number));
    });
  }

  if (id === 1) {
    rl.question("Ingresar nombre jugador:\n", (player_name) => {
      addPlayers(players_array, player_name);
      console.log("¡Hola " + player_name + "! tu puntaje inicial es 501\n");
      rl.close();
      enterPlayers(players_array, n_players - 1);
    });
  }
};
```

Asincronía de los inputs

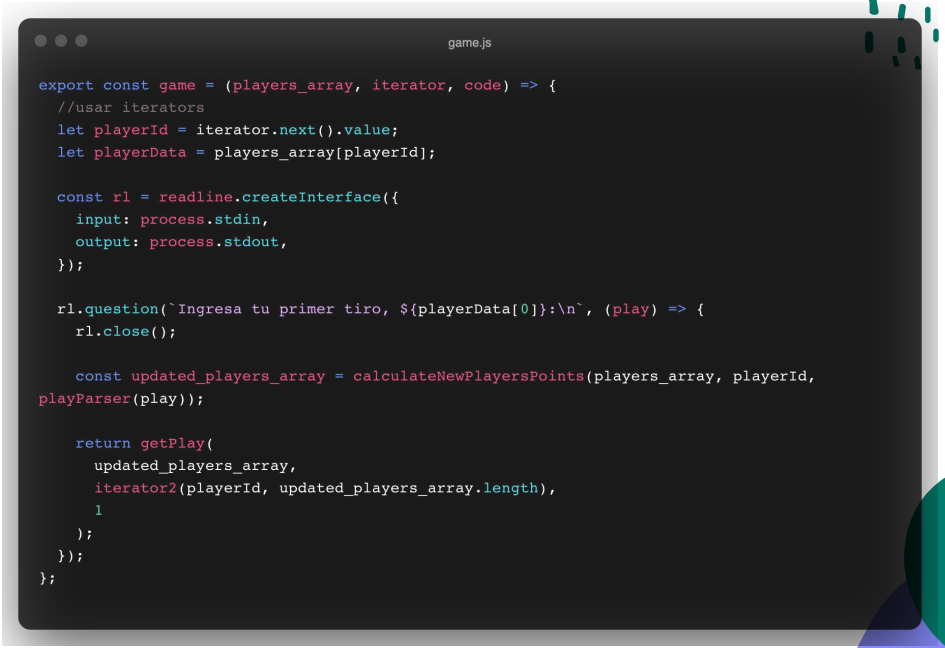
Terminamos muchas veces usando recursiones para manejar estas asincronías

game.js

```
const getPlay = (players_array, iterator, plays) => {
  let playerId = iterator.next().value;
  let playerData = players_array[playerId];
  const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });

  rl.question(`Ingresa tu siguiente tiro, ${playerData[0]}:\n`, (play) => {
    rl.close();

    const updated_players_array = calculateNewPlayersPoints(
      players_array,
      playerId,
      playParser(play)
    );

    if (plays + 1 === 3){
      showGameLogs(updated_players_array);

      if (checkWinner(updated_players_array)){
        return true;
      }

      return game(
        updated_players_array,
        iterator2(playerId + 1, updated_players_array.length)
      );
    }

    return getPlay(
      updated_players_array,
      iterator2(playerId, updated_players_array.length),
      plays + 1
    );
  });
};
```

Pérdidas de datos

Al pasar variables y constantes entre funciones definidas previamente se perdian sus datos y teníamos que rehacerlas cada cierto tiempo



04

Demo