Javascript Funcional

Diseño Avanzado de Aplicaciones Web
IIC3585-1

Diego González
Melisa Rodríguez
Max Schudeck

Profesor Jaime Navon Cohen

Propiedades de JS Funcional

Pasar Funciones

Puedes asignar funciones a **variables** y pasarlas como **parámetro** a otras funciones o devolverlo como **función**.

Clausura

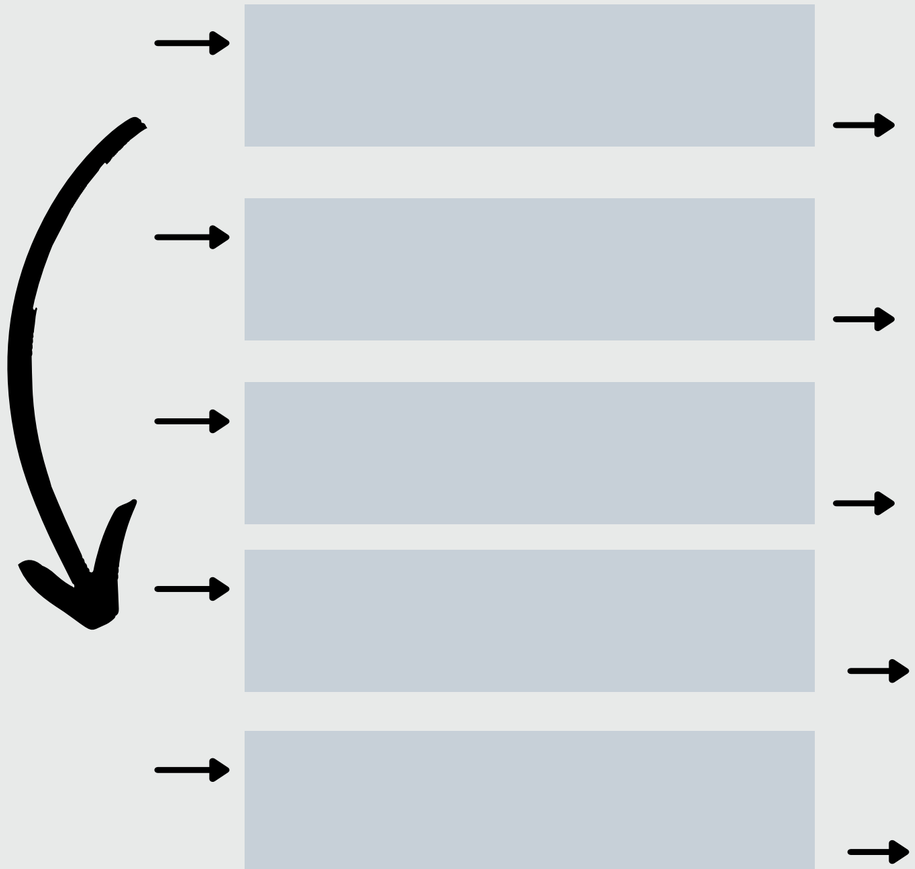
Es posible hacer referencia a variables desde **fuera del alcance** de la función dentro del cuerpo de la función anónima.

Inmutabilidad

Para disminuir errores, las propiedades de un objeto no serán modificadas, se podrá crear una **nueva** copia con el cambio.

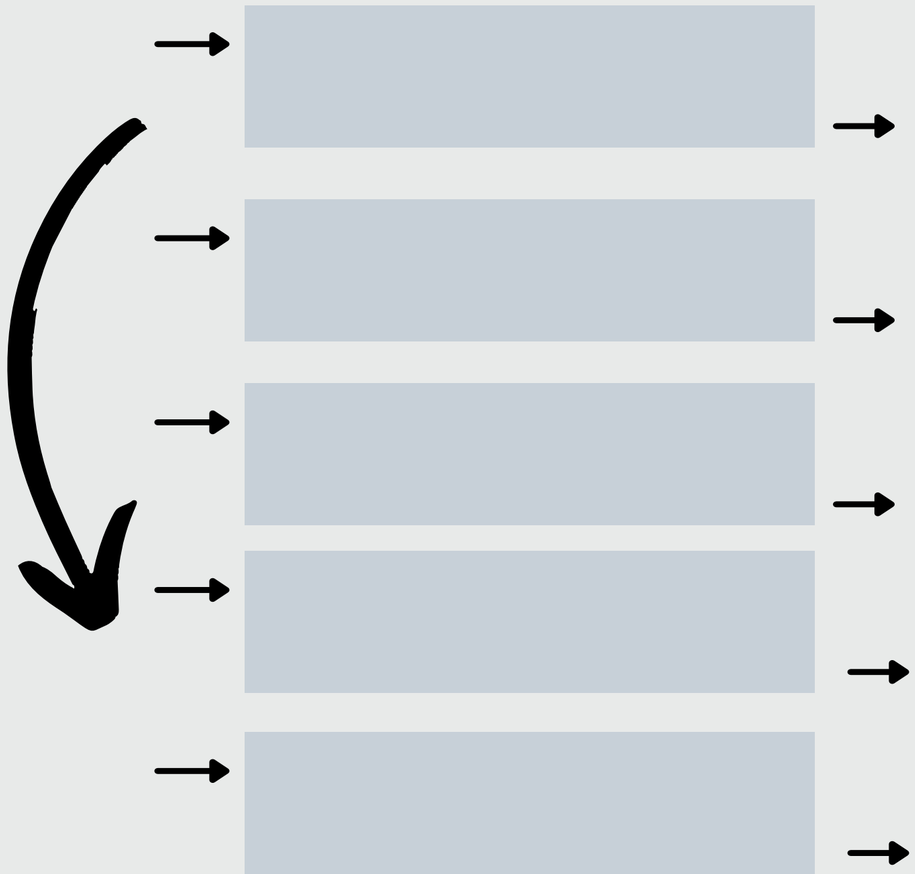
Javascript Funcional

no necesita ser tan específico como al escribir código imperativo

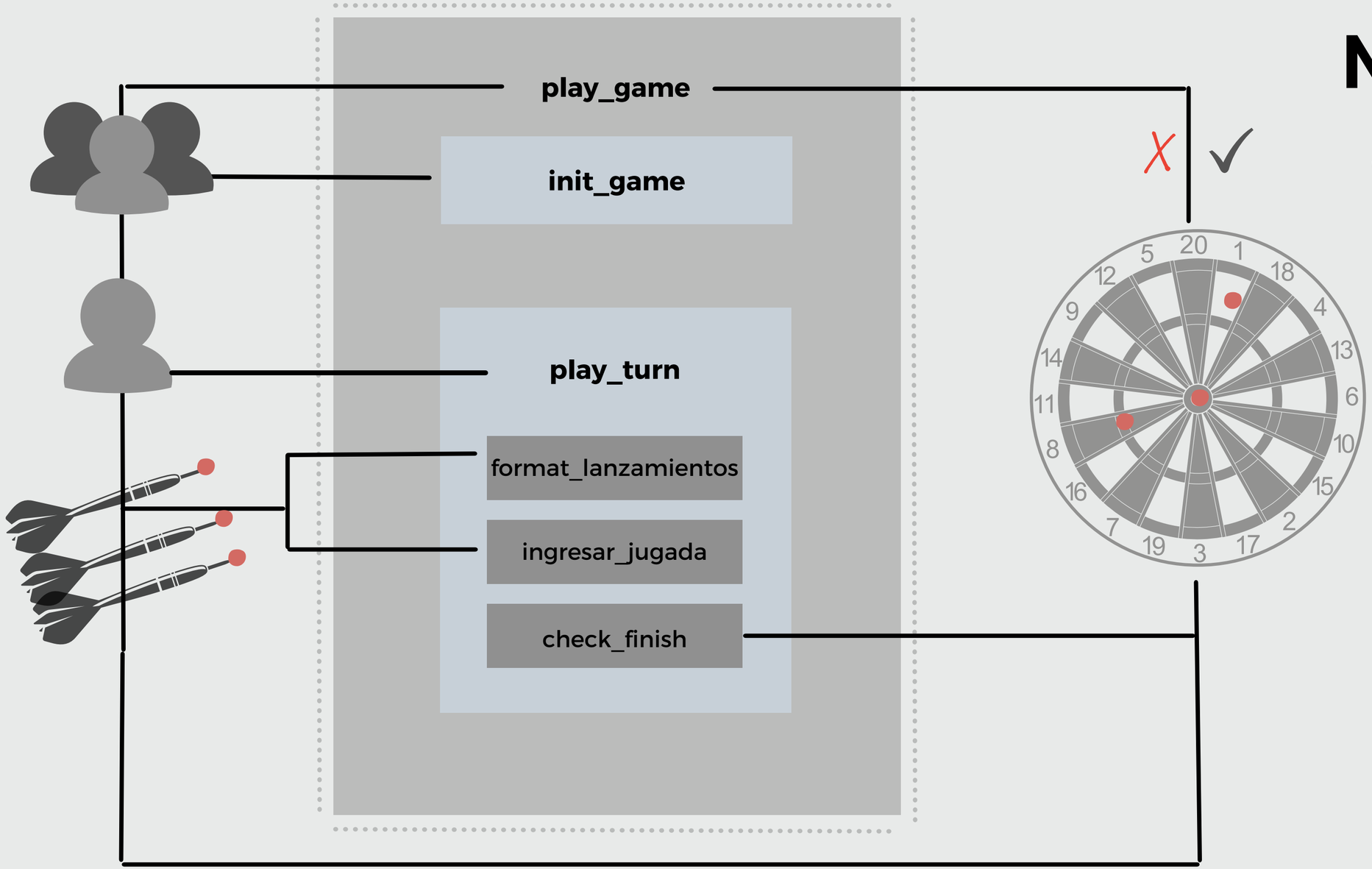


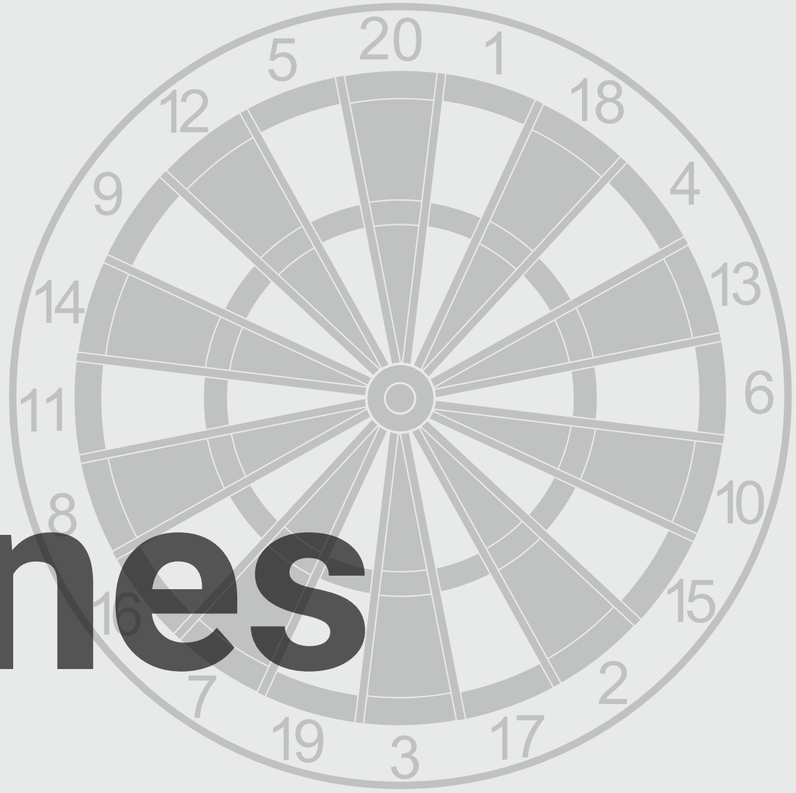
Javascript Funcional

- Más **Comprensible**
- Más **autodescriptivo**
- Menos propenso a **errores**



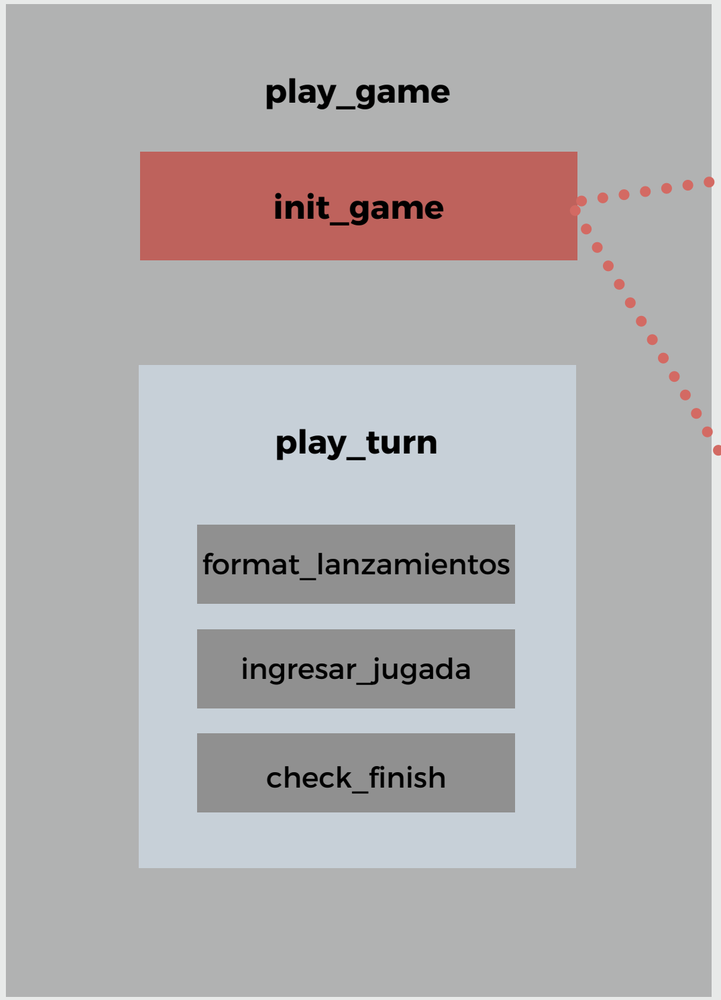
Nuestro Juego





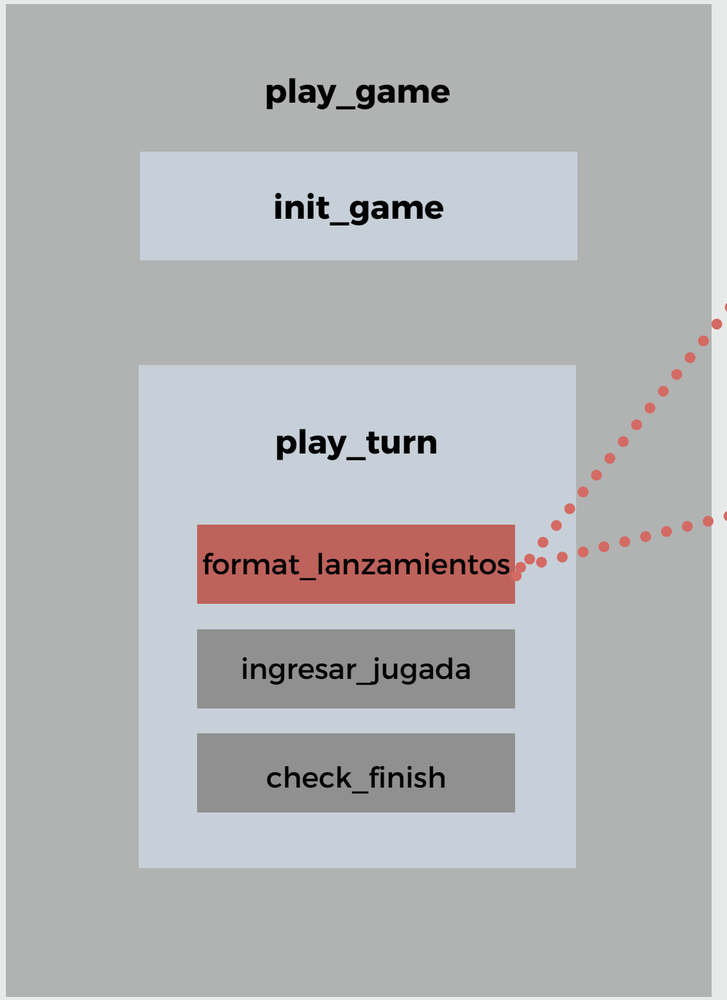
Funciones

init_game



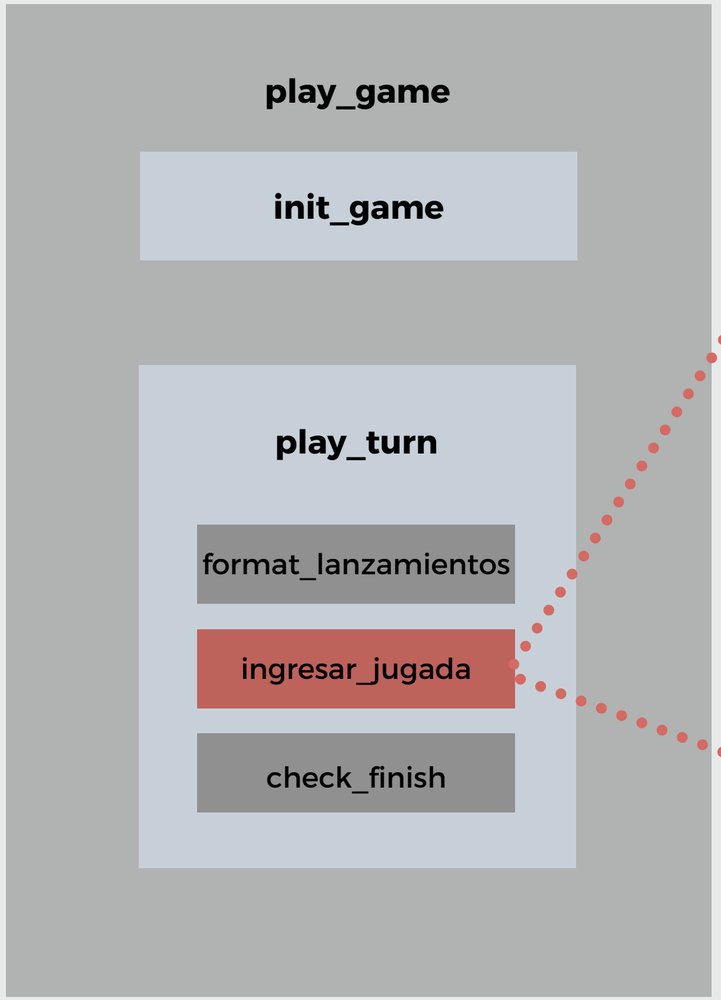

```
//INICIALIZAR EL JUEGO --> array de jugadores (usarmos args para que funcione con +2 jugadores)
function init_game(...args) {
  console.log(`Juego inicializado con los jugadores ${args}\n`)
  const array_nombre = [...args[0]];
  // Puntajes iniciales
  const array_puntaje = array_nombre.map((element) => {
    return [element, 501];
  });
  return array_puntaje;
}
```

format_lanzamientos



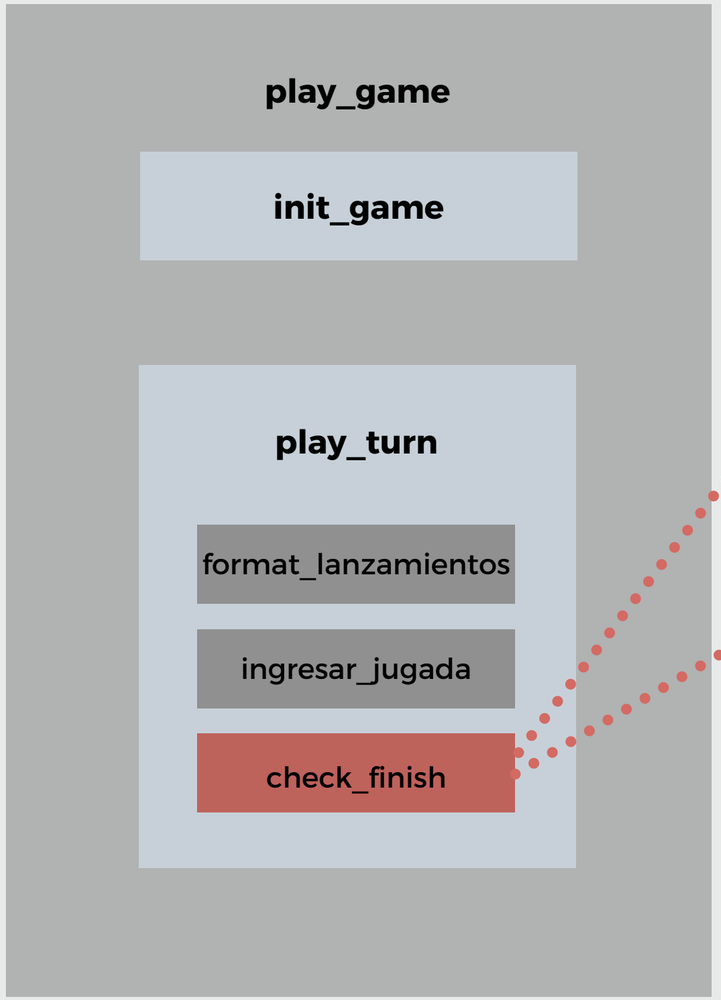

```
// Funcion que formatea el string de jugada ingresado por el usuario
const format_lanzamientos = (lanzamientos) => {
  return lanzamientos.split(', ').map(elem => elem.replace(/[^a-zA-Z0-9, ]/g, ''));
} // OUTPUT: Lanzamientos en forma de array
```

ingresar_jugada



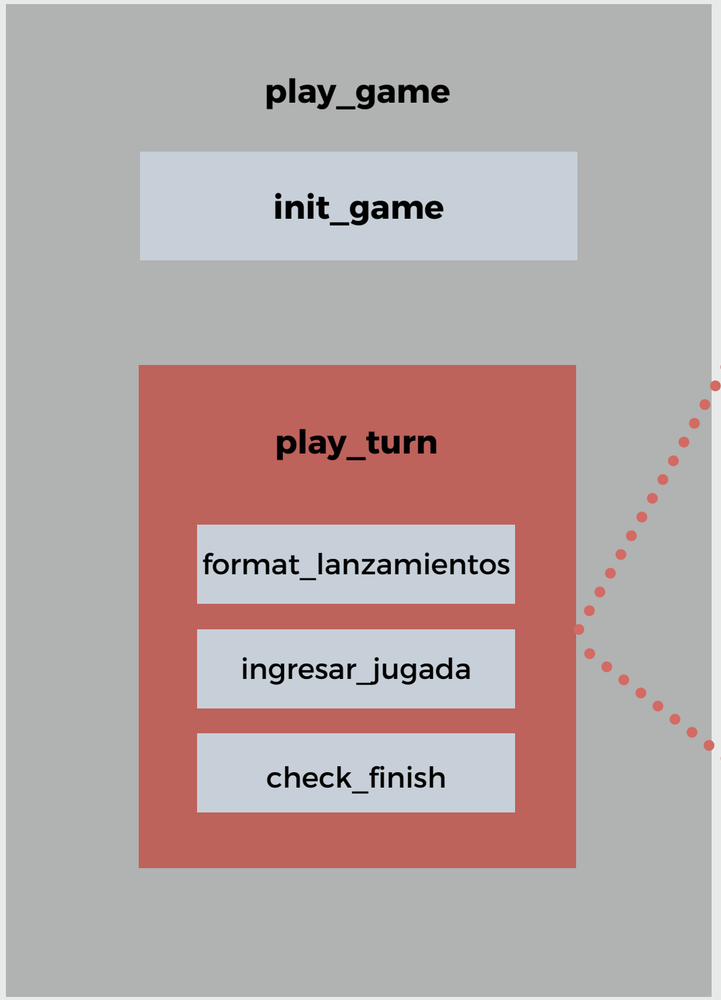

```
// Funcion que calcula puntajes despues de ingresar una jugada válida
const ingresar_jugada = (game, jugador, lanzamientos) => {
  const puntaje = game[jugador][1]
  const bulls = {'DB': 50, 'SB':25}
  const nuevo_puntaje = puntaje - lanzamientos.reduce((prev, curr) => {
    if (bulls[curr]) return prev + bulls[curr]
    const [mult, puntaje] = curr.split(',').map(elem => + elem)
    return prev + (mult * puntaje)
  },0 )
  game[jugador][1] = nuevo_puntaje < 0 ? 0: nuevo_puntaje

} //OUTPUT: Actualiza el puntaje del jugador
```

check_finish



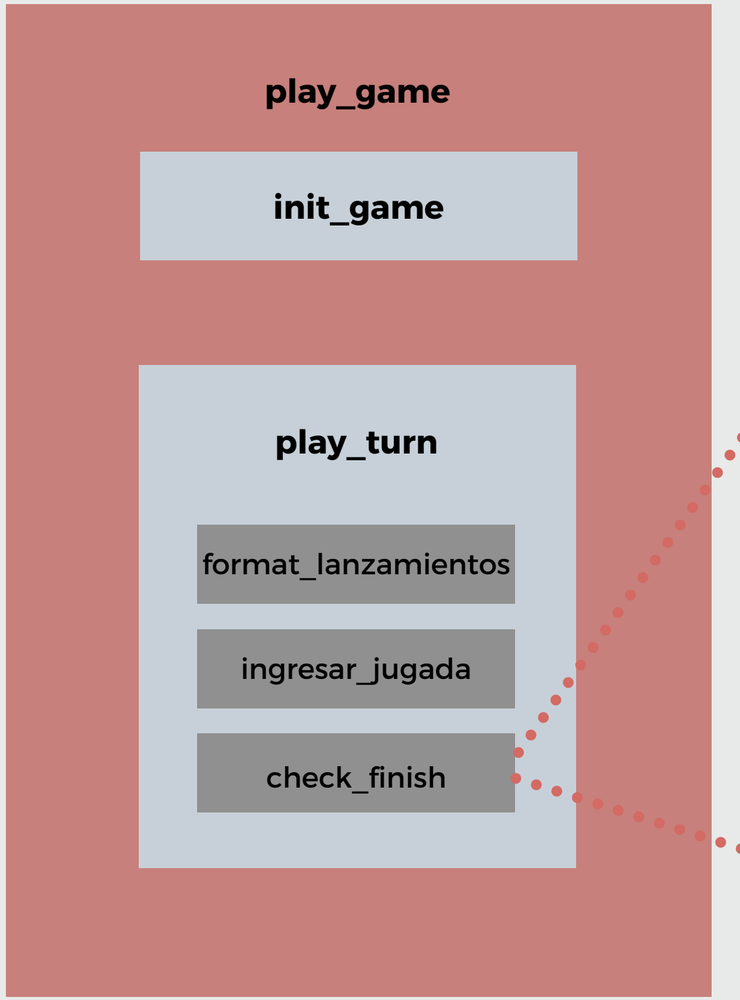

```
const check_finish = (jugador, game) => {
  return game[jugador][1] === 0
} // OUTPUT : bool
```

play_turn




```
// Funcion que maneja flujo de turnos
const play_turn = (game, turn, finish) => {
  if (finish) return;
  const input = prompt(`Ingrese los lanzamientos de ${game[turn][0]}:`)
  const lanzamientos = format_lanzamientos(input)
  const ingresar_jugada_curry = _.curry(ingresar_jugada)(game)
  ingresar_jugada_curry(turn, lanzamientos);
  console.log(`${game[turn][0]} queda con ${game[turn][1]} puntos`)
  finish = check_finish(turn, game)
  console.log(finish ? `Felicidades ${game[turn][0]}`: '')

  turn = (turn + 1) % game.length
  play_turn(game, turn, finish)
}
```

play_game




```
const play_game = (...args) => {
  // Variables de estado del juego
  let game;
  let turn = 0
  let finish = false;
  game = init_game(args);
  const play_turn_curry = _.curry(play_turn)(game)
  play_turn_curry(turn, finish)

}

play_game('Jaime', 'Ema', "Doc")
```

Características Destacables

Transformando Arrays con map
Toma una matriz y la transforma en otra matriz mediante la aplicación dada.

Reduce
transforma obedeciendo a una función, el valor del acumulador actual y el elemento de una matriz al siguiente

Usando recursión en vez de loop
Cambio de uso de While a uso de recursividad para realizar los turnos.

Lodash - Curry
Currying es un proceso de transformación de una función multiparámetro a una función de un solo parámetro

Transformando Arrays con map

```
function init_game(...args) {
  console.log(`Juego inicializado con los jugadores ${args}\n`)
  const array_nombre = [...args[0]];
  // Puntajes iniciales
  const array_puntaje = array_nombre.map((element) => {
    return [element, 501];
  });
  return array_puntaje;
}
```

```
// Funcion que formatea el string de jugada ingresado por el usuario
const format_lanzamientos = (lanzamientos) => {
  return lanzamientos.split(', ').map(elem => elem.replace(/[^a-zA-Z0-9, ]/g, ''));
} // OUTPUT: Lanzamientos en forma de array
```

Reduce

```
// Funcion que calcula puntajes despues de ingresar una jugada válida
const ingresar_jugada = (game, jugador, lanzamientos) => {
  const puntaje = game[jugador][1]
  const bulls = {'DB': 50, 'SB':25}
  const nuevo_puntaje = puntaje - lanzamientos.reduce((prev, curr) => {
    if (bulls[curr]) return prev + bulls[curr]
    const [mult, puntaje] = curr.split(',').map(elem => + elem)
    return prev + (mult * puntaje)
  },0 )
  game[jugador][1] = nuevo_puntaje < 0 ? 0: nuevo_puntaje

} //OUTPUT: Actualiza el puntaje del jugador
```

Usando recursión en vez de loop

Loop

```
const play_turn = (game, turn, finish) => {
  while (!finish) {
    const input = prompt(`Ingrese los lanzamientos de ${game[turn][0]}:`)
    lanzamientos = format_lanzamientos(input)
    ingresar_jugada(turn, game[turn][1], lanzamientos);
    console.log(`${game[turn][0]} queda con ${game[turn][1]} puntos`)
    finish = check_finish(turn)
    console.log(finish ? `Felicidades ${game[turn][0]}`: '')
    turn = (turn +1)%2
  }
}
```

Usando recursión en vez de loop

Recursión

```
const play_turn = (game, turn, finish) => {
  if (finish) return;
  const input = prompt(`Ingrese los lanzamientos de ${game[turn][0]}:`)
  const lanzamientos = format_lanzamientos(input)
  ingresar_jugada(turn, game[turn][1], lanzamientos, game);
  console.log(`${game[turn][0]} queda con ${game[turn][1]} puntos`)
  finish = check_finish(turn, game)
  console.log(finish ? `Felicidades ${game[turn][0]}`: '')
  turn = (turn +1)%2
  play_turn(game, turn, finish)
}
```

Lodash - Curry

```
// Funcion que maneja flujo de turnos
const play_turn = (game, turn, finish) => {
  if (finish) return;
  const input = prompt(`Ingrese los lanzamientos de ${game[turn][0]}:`)
  const lanzamientos = format_lanzamientos(input)
-   ingresar_jugada(turn, game[turn][1], lanzamientos, game);

  console.log(`${game[turn][0]} queda con ${game[turn][1]} puntos`)
  finish = check_finish(turn, game)
  console.log(finish ? `Felicidades ${game[turn][0]}`: '')

-   if (turn == (game.length - 1)){
-     turn = 0
-   } else {
-     turn = turn + 1
-   }
  play_turn(game, turn, finish)
}
```

```
// Funcion que maneja flujo de turnos
const play_turn = (game, turn, finish) => {
  if (finish) return;
  const input = prompt(`Ingrese los lanzamientos de ${game[turn][0]}:`)
  const lanzamientos = format_lanzamientos(input)
+   const ingresar_jugada_curry = _.curry(ingresar_jugada)(game)
+   ingresar_jugada_curry(turn, lanzamientos);
  console.log(`${game[turn][0]} queda con ${game[turn][1]} puntos`)
  finish = check_finish(turn, game)
  console.log(finish ? `Felicidades ${game[turn][0]}`: '')

+   turn = (turn + 1)%game.length

  play_turn(game, turn, finish)
}
```

Lodash - Curry

```
// Funcion que maneja flujo de turnos
const play_turn = (game, turn, finish) => {
  if (finish) return;
  const input = prompt(`Ingrese los lanzamientos de ${game[turn][0]}:`)
  const lanzamientos = format_lanzamientos(input)
- ingresar_jugada(turn, game[turn][1], lanzamientos, game);
+ const ingresar_jugada_curry = _.curry(ingresar_jugada)(game)
+ ingresar_jugada_curry(turn, lanzamientos);

  finish = check_finish(turn, game)
  console.log(finish ? `Felicidades ${game[turn][0]}`: '')

-   if (turn == (game.length - 1)){
-     turn = 0
-   } else {
-     turn = turn + 1
-   }
+   turn = (turn + 1)%game.length
  play_turn(game, turn, finish)
}
```

Javascript Funcional

Diseño Avanzado de Aplicaciones Web
IIC3585-1

Diego González
Melisa Rodríguez
Max Schudeck

Profesor Jaime Navon Cohen