



Tarea 2

Programación Reactiva

Julio Andrade
Gerardo Crot
M^a Josefa Espinoza



Tabla de contenidos

01

Reglas del juego

02

Demo

03

**Principales
implementaciones**

04

**Problemas
encontrados**



01

Reglas del juego

Reglas del juego



- Los jugadores se mueven en línea recta y uno les indica las direcciones
- Si se choca con un fantasma se restan 2 puntos y te devuelve a la parte inicial
- Si se agarra una moneda se obtiene un punto
- Monedas aparecen aleatoriamente en el mapa de una en una
- Gana el que llega a 10 puntos





02

Demo

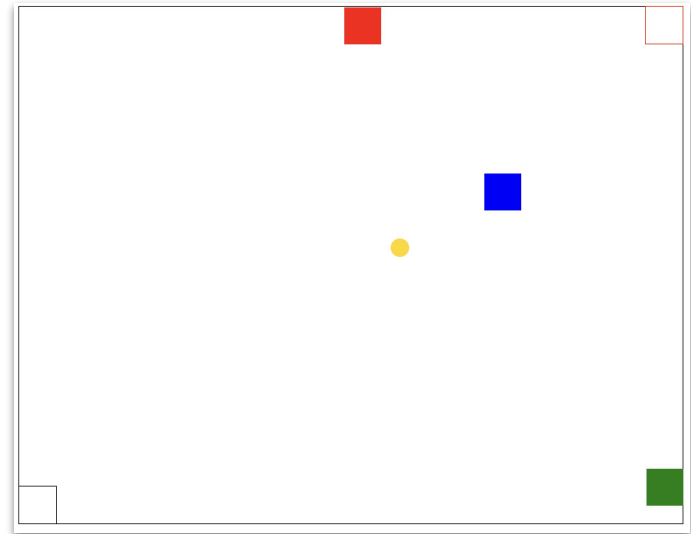


03

Principales implementaciones

Aparición de los personajes

Al inicio debíamos generar los personajes y guardarlos en variables. Se les hizo objetos y los agregamos con sus observables. Creamos observables por tipos de datos para manejar los objetos y arrays



```

src/characters.js

let player1 = {
  id: 1,
  size: 50,
  div: $("#square1"),
  direction: "ArrowDown",
  player: true,
  points: 0,
};
player1.initPosition = {
  top: 8,
  left: 8,
}

let player2 = {
  id: 2,
  size: 50,
  div: $("#square2"),
  direction: "w",
  player: true,
  points: 0,
};
player2.initPosition = {
  top: 700 + 8 - player2.size,
  left: 900 + 8 - player2.size,
}

// NPC
const basetNPC = {
  id: 0,
  size: 50,
  direction: "",
  color: "",
  player: false,
};

let NPCs = [
  { ...basetNPC, id: 1, color: "red" },
  { ...basetNPC, id: 2, color: "green" },
  { ...basetNPC, id: 3, color: "orange" },
];

```

```

src/characters.js

Rx.Observable.of(player1)
  .subscribe(() => {
    player1.div.css({
      top: player1.initPosition.top,
      left: player1.initPosition.left,
      width: player1.size + "px",
      height: player1.size + "px",
    });
    generatePoints(player1);
  });

Rx.Observable.of(player2)
  .subscribe(() => {
    player2.div.css({
      top: player2.initPosition.top,
      left: player2.initPosition.left,
      width: player2.size + "px",
      height: player2.size + "px",
    });
    generatePoints(player2);
  });

// set NPCs
Rx.Observable.from(NPCs)
  .map(npc => {
    npc.div = generateNPCDiv(npc.id);
    return npc
  })
  .subscribe((npc) => {
    npc.div.css({
      width: npc.size + "px",
      height: npc.size + "px",
      position: "absolute",
      margin: 0,
      backgroundColor: npc.color,
    });
  });

```


Dirección de los personajes

Uso de observables para presionar y levantar una tecla al que se le hizo merge agrupándolos por tecla

```
src/rx.js

let keyDownObservable = Rx.Observable.fromEvent($(document), "keydown");
let keyUpsObservable = Rx.Observable.fromEvent(document, "keyup");

let keyPresses = keyDownObservable
  .merge(keyUpsObservable)
  .groupBy((e) => e.keyCode)
  .distinctUntilChanged(null, (e) => e.type + (e.key || e.which))
  .mergeAll();

// subscriptions
let keyPressesSubscription = keyPresses.subscribe((e) => {
  const player1Key = "Arrow";
  const player2Key = ["w", "a", "s", "d"];
  if (player1Key === e.key.slice(0, 5)) {
    player1.direction = e.key;
  }
  if (player2Key.includes(e.key)) {
    player2.direction = e.key;
  }
});
```

Movimiento de los personajes

Se usó el observable interval para ir moviendo cada cierta cantidad de pixeles a los personajes

```
src/nx.js

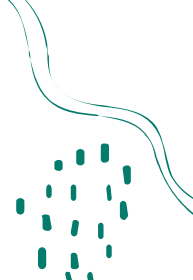
// Movements with intervals
let playersMove = Rx.Observable.interval(35).subscribe(() => {
  movePlayer1(player1);
  movePlayer2(player2);
  for (let i = 0; i < NPCs.length; i++) {
    moveNPC(NPCs[i]);
  }
});

let npcDirection = Rx.Observable.interval(8*100)
  .subscribe(() => {
    const moves = ['up', 'right', 'left', 'down']
    for (let i = 0; i < NPCs.length; i++){
      let move = moves[Math.floor(Math.random()*moves.length)];
      NPCs[i].direction = move;
    }
  });
```



Manejo de colisiones


Dos observables de intervalos para los fantasmas y las monedas
Revisamos las colisiones según las posiciones de los personajes



```
src/rx.js

// players collisions with intervals
let playersCollisionSubscription = Rx.Observable.interval(3*100).subscribe(() => {
  checkNPCCollision(player1, player2, NPCs);
});

let coinCollisionSubscription = Rx.Observable.interval(100).subscribe(() => {
  coin = checkCoinCollision(player1, player2, coin);
});
```



src/utils.js

```
const checkCollision = (player, gameObject) => {
  let playerPosY = player.div.position().top;
  let playerPosX = player.div.position().left;
  const condition = (element) => {
    let gameObjectPosX = element.div.position().left;
    let gameObjectPosY = element.div.position().top;
    let directions = {
      right: ['d', 'ArrowRight'],
      left: ['a', 'ArrowLeft'],
      up: ['w', 'ArrowUp'],
      down: ['s', 'ArrowDown']
    }
    if (
      //right collision
      playerPosX < gameObjectPosX &&
      playerPosX + player.size >= gameObjectPosX &&
      distance(playerPosX, playerPosY, gameObjectPosX, gameObjectPosY) <
        player.size * Math.sqrt(2) &&
      directions.right.includes(player.direction)
    ) {
      return true;
    };
    if (
      //left collision
      playerPosX > gameObjectPosX &&
      playerPosX <= gameObjectPosX + gameObject.size &&
      distance(playerPosX, playerPosY, gameObjectPosX, gameObjectPosY) <
        player.size * Math.sqrt(2) &&
      directions.left.includes(player.direction)
    ) {
      return true;
    };
  };
};
```

src/utils.js

```
if (
  //up collision
  playerPosY > gameObjectPosY &&
  playerPosY <= gameObjectPosY + gameObject.size &&
  distance(playerPosX, playerPosY, gameObjectPosX, gameObjectPosY) <
    player.size * Math.sqrt(2) &&
  directions.up.includes(player.direction)
) {
  return true;
};
if (
  //down collision
  playerPosY < gameObjectPosY &&
  playerPosY + player.size >= gameObjectPosY &&
  distance(playerPosX, playerPosY, gameObjectPosX, gameObjectPosY) <
    player.size * Math.sqrt(2) &&
  directions.down.includes(player.direction)
) {
  return true;
};
return false;
};
return condition(gameObject);
};
```



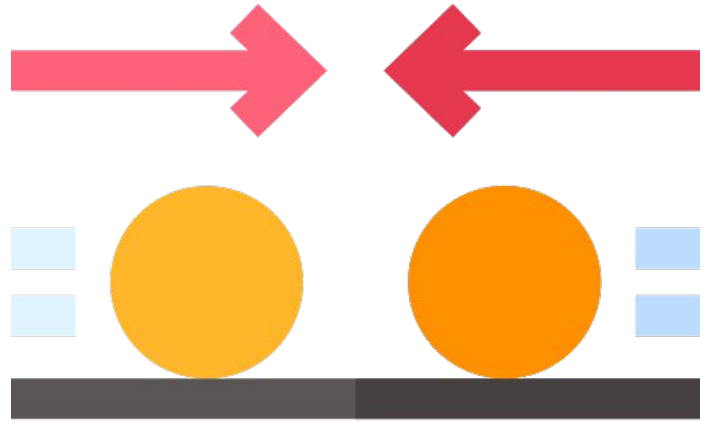
04

Problemas encontrados



Detección de colisiones

El proceso de chequeo es iterativo y considerando también la velocidad de los movimientos podrían ocurrir colisiones no identificadas. Por lo que fue necesario calibrar los intervalos de tiempo y las situaciones borde.



Cómo observar cambios

Una de nuestras intenciones (y como no conocemos tanto de Rxjs), era intentar crear un Observable que viera los cambios en los puntajes, para el cual no encontramos una forma simple

```
src/utlis.js

const checkCoinCollision = (player1, player2, coin) => {
  if (checkCollision(player1, coin)) {
    coin.div.remove();
    coin = generateCoin();
    player1.points += 1;
    pipe(generatePoints(player1), checkWinner(player1));
  };
  if (checkCollision(player2, coin)) {
    coin.div.remove();
    coin = generateCoin();
    player2.points += 1;
    pipe(generatePoints(player2), checkWinner(player2));
  };
  return coin;
};
```

Tarea 2

Programación Reactiva

