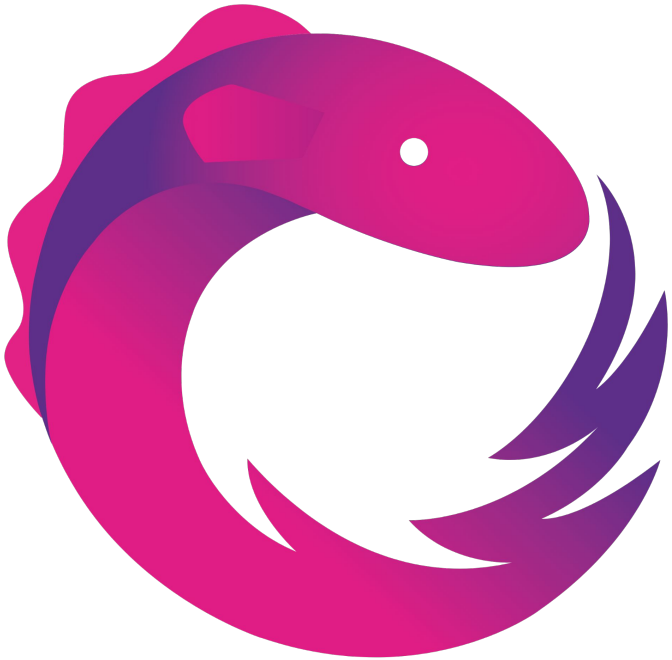
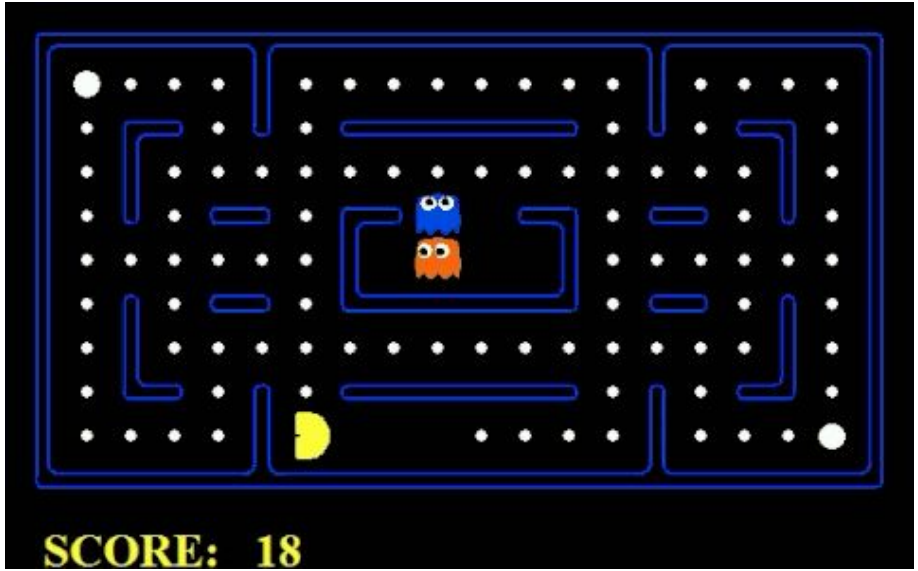
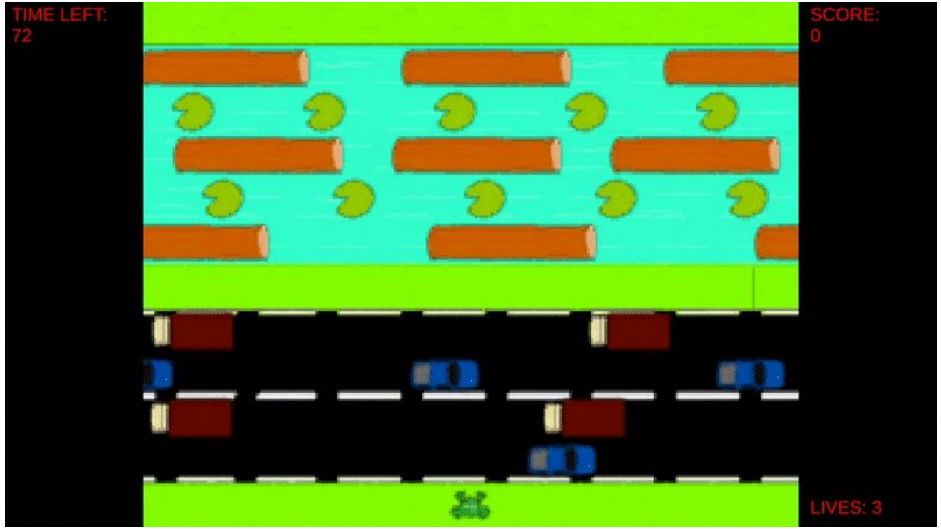


Programación Reactiva RxJS

Grupo 1:
Joaquín Cáceres
Julián García
Mathias Valdebenito

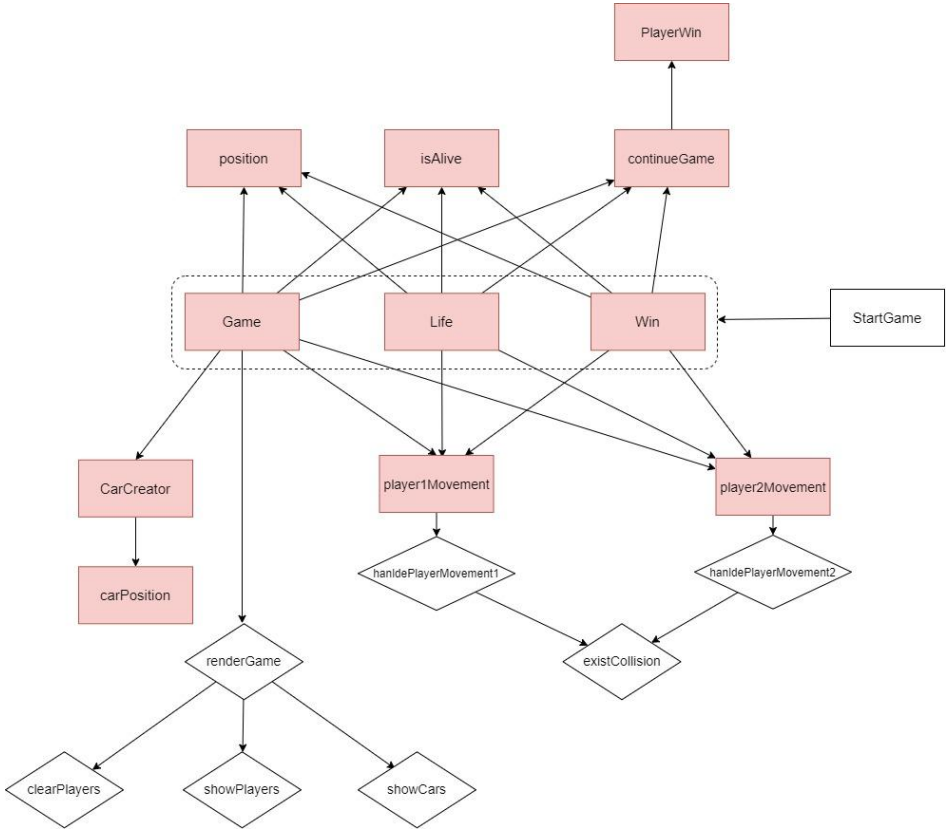
Juegos





01

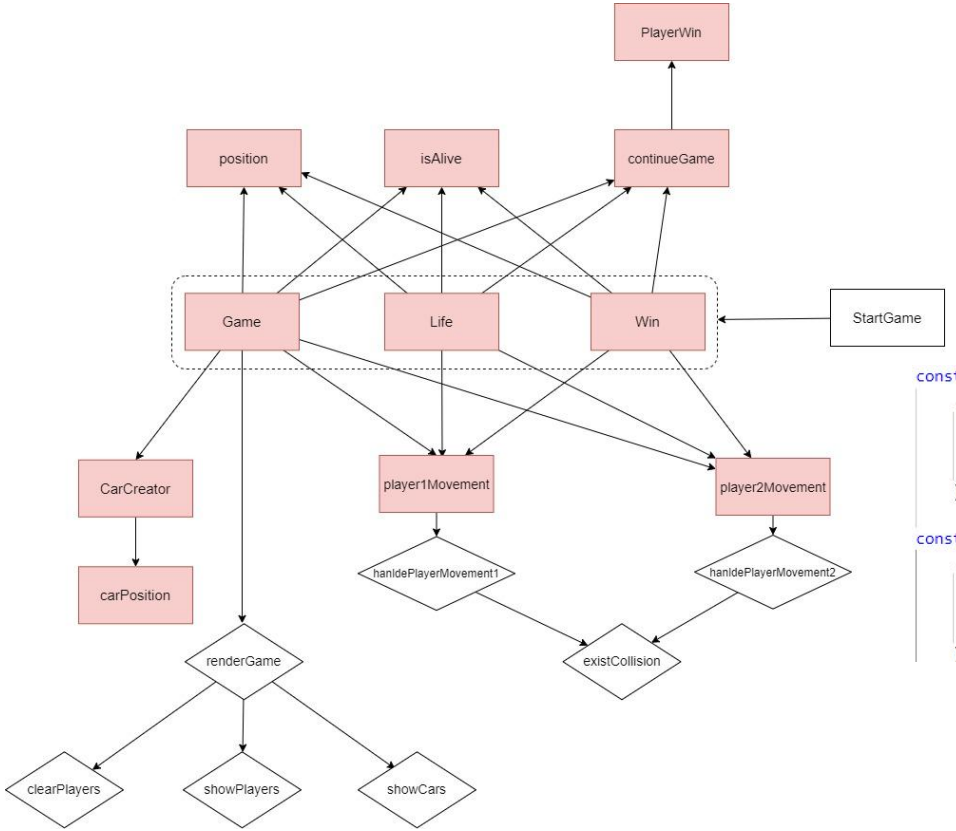
Demo juego: frogger



BehaviorSubject

```
const positions$ = new BehaviorSubject({
    X1: PLAYER1_STARTING_POSITION_X, Y1: PLAYER1_STARTING_POSITION_Y,
    X2: PLAYER2_STARTING_POSITION_X, Y2: PLAYER2_STARTING_POSITION_Y,
})

const carPositions$ = new BehaviorSubject([])

const IsAlive$ = new BehaviorSubject(true)
const PlayersWin$ = new BehaviorSubject(false)
```

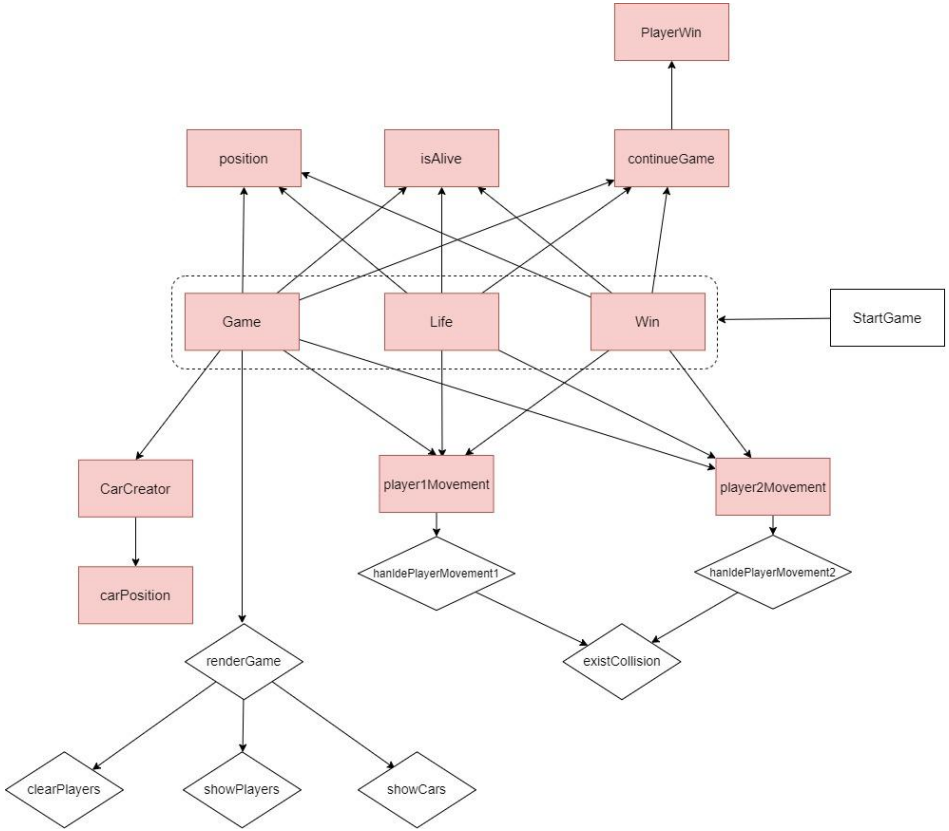
Uso de *behaviorSubject* para definir un observable de los jugadores, enemigos y estados del juego.



fromEvent

```
const player1Movement$ = fromEvent(document, 'keyup')
    .pipe(
        merge(fromEvent(document, 'keydown')),
        pluck('key'),
        scan(handlePlayer1Movement, {X1: PLAYER1_STARTING_POSITION_X, Y1: PLAYER1_STARTING_POSITION_Y}),
    )

const player2Movement$ = fromEvent(document, 'keyup')
    .pipe(
        merge(fromEvent(document, 'keydown')),
        pluck('key'),
        scan(handlePlayer2Movement, {X2: PLAYER2_STARTING_POSITION_X, Y2: PLAYER2_STARTING_POSITION_Y})
    )
```

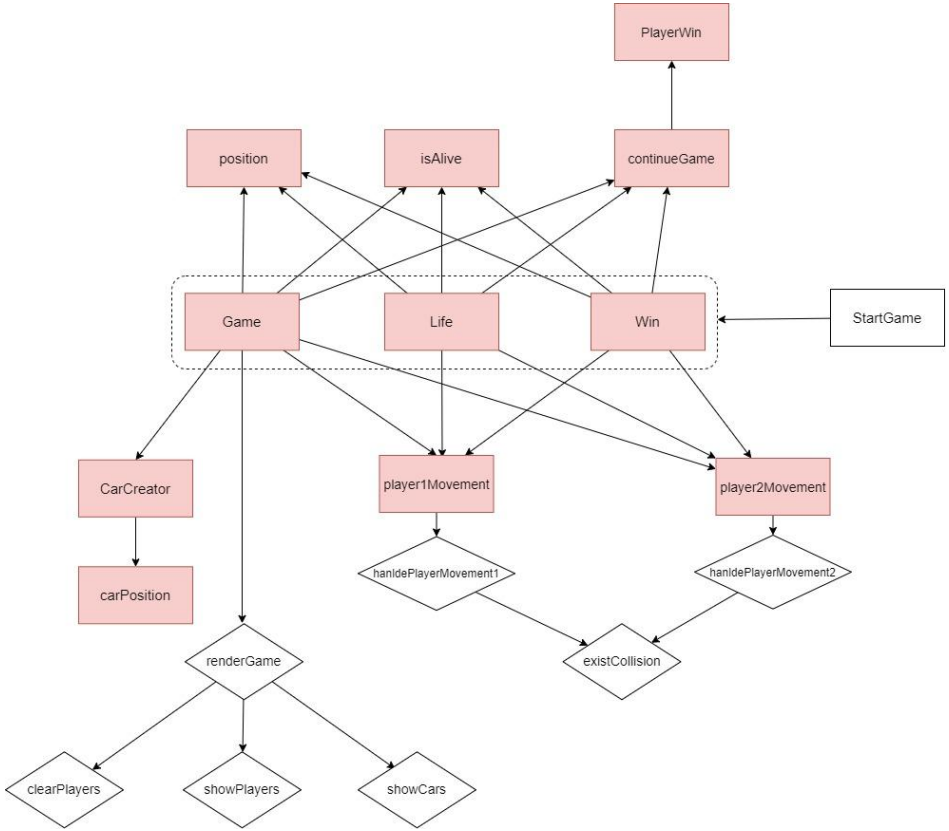
Uso de *fromEvent* para definir un stream del input de los jugadores.



interval

```
const CarCreator$ = interval(CAR_FREQUENCY).pipe(
    map(_ => createCar()),
    tap((car) => carPositions$.next([...carPositions$.getValue(), car])),
)

const Cars$ = CarCreator$.pipe(
    scan((cars, car) => [...cars, car], []),
    map(cars => {
        cars.forEach((car) => {
            car.x += CAR_SPEED
        })
        carPositions$.next(carPositions$.getValue().map((car) => {
            car.x += CAR_SPEED;
            return car
        }));
        return cars
    }),
    share()
)
```

Uso de *interval* para definir un stream que genera los enemigos.



CombineLatest

```
const Life$ = combineLatest(
    Cars$,
    player1Movement$,
    player2Movement$)
    .pipe(
    map(([cars, {X1, Y1}, {X2, Y2}]) => detectCarColission(cars, {X1, Y1}, {X2, Y2})),
    filter((bool) => bool),
    takeWhile(isAlive),
    takeWhile(continueGame)
)

const Win$ = combineLatest(
        player1Movement$,
        player2Movement$
        ).pipe(
            map(([{Y1}, {Y2}]) => (Y1 + PLAYER_HEIGTH >= MAP_BOTTOM + MAP_WIDTH)
            && (Y2 + PLAYER_HEIGTH >= MAP_BOTTOM + MAP_WIDTH) ),
            filter(bool => bool),
            takeWhile(isAlive),
            takeWhile(continueGame)
        )

const Game$ = combineLatest(player1Movement$, player2Movement$, Cars$,
    ({X1, Y1}, {X2, Y2}, cars) => ({X1, Y1, X2, Y2, cars}))
    .pipe(
    tap(({X1, Y1, X2, Y2}) => positions$.next({X1, Y1, X2, Y2})),
    sample(interval(50)),
    takeWhile(isAlive),
    takeWhile(continueGame)
)
```

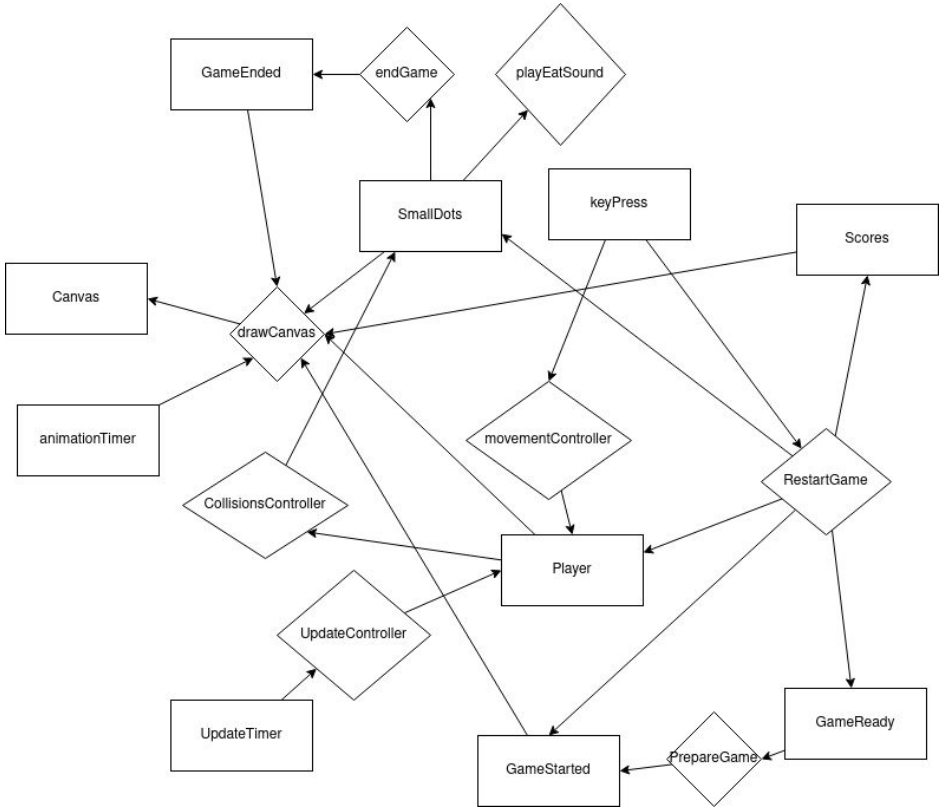
Observables que combinan todos los observables anteriores.

02

Demo juego: pacman

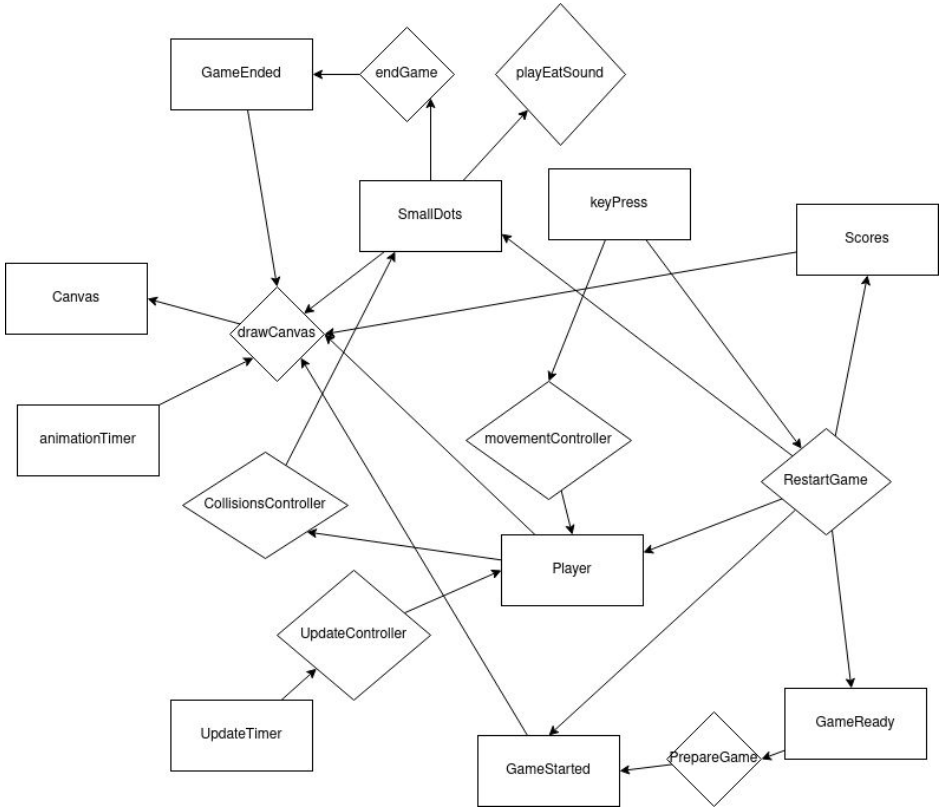
throttleTime

```
smallDots.pipe(
    rxjs.operators.throttleTime(PACMAN_SOUND_DURATION)
).subscribe(() => {
    if(gameReady.getValue() && gameStarted.getValue()){
      const audio = document.getElementById('pacmanSound');
      const newAudio = audio.cloneNode(true);
      newAudio.play()
    }
  });
```



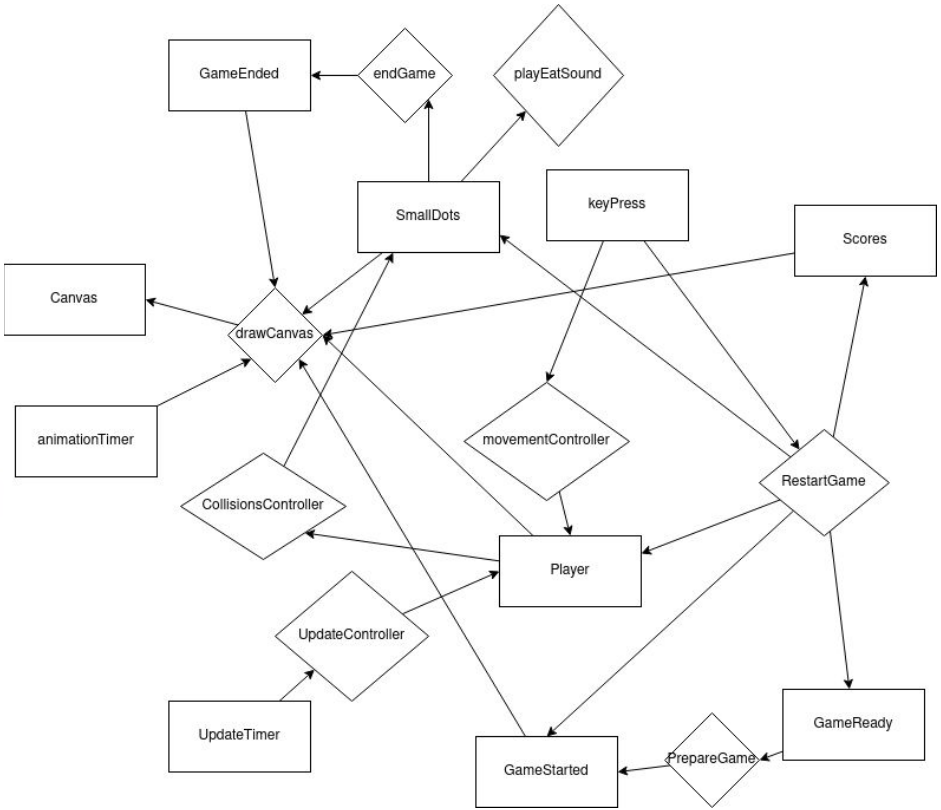
separar rendering y lógica

```
players.forEach(player => {
  player.subscribe(() => drawCanvas(ctx, map));
})

animationTime.subscribe(() => drawCanvas(ctx, map));

smallDots.subscribe(() => drawCanvas(ctx, map));
```



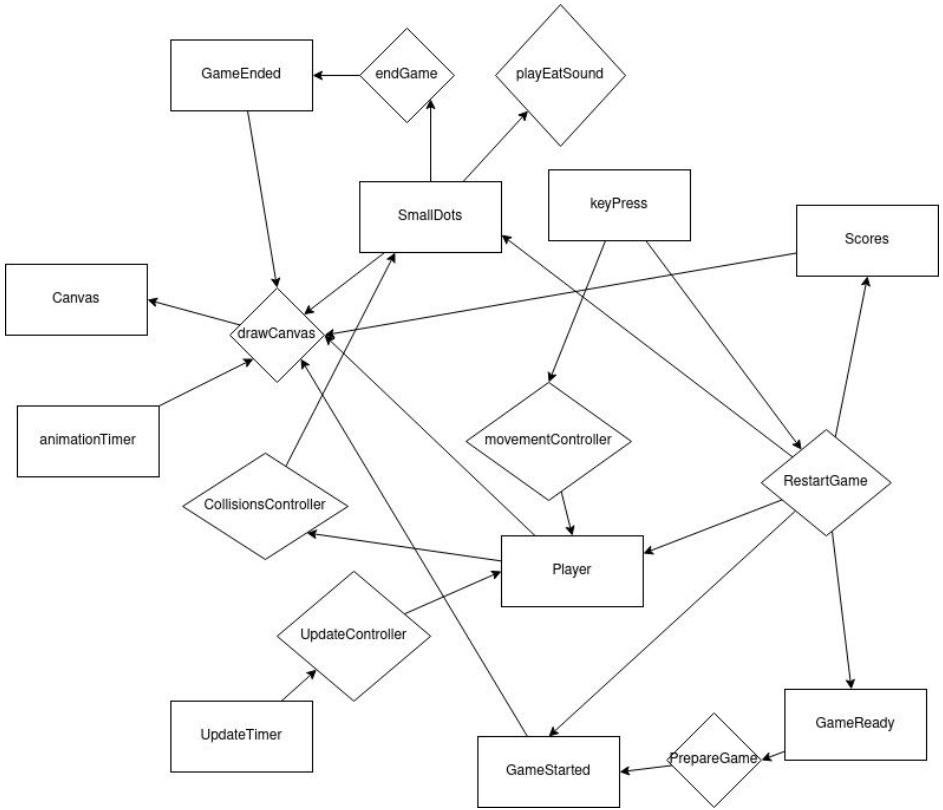
movement/update

```
function movementController(keyCode) {
  PLAYERS.forEach((PLAYER, i) => {
    switch(keyCode) {
      case PLAYER.UP:
        players[i].next({...players[i].getValue(), DIR: DIRECTIONS.UP})
        break;
      case PLAYER.DOWN:
        players[i].next({...players[i].getValue(), DIR: DIRECTIONS.DOWN})
        break;
      case PLAYER.LEFT:
        players[i].next({...players[i].getValue(), DIR: DIRECTIONS.LEFT})
        break;
      case PLAYER.RIGHT:
        players[i].next({...players[i].getValue(), DIR: DIRECTIONS.RIGHT})
        break;
    }
  });
  switch(keyCode) {
    case R:
      restartGame();
      break;
  }
}

keyPress.subscribe(movementController);
```



Pacman

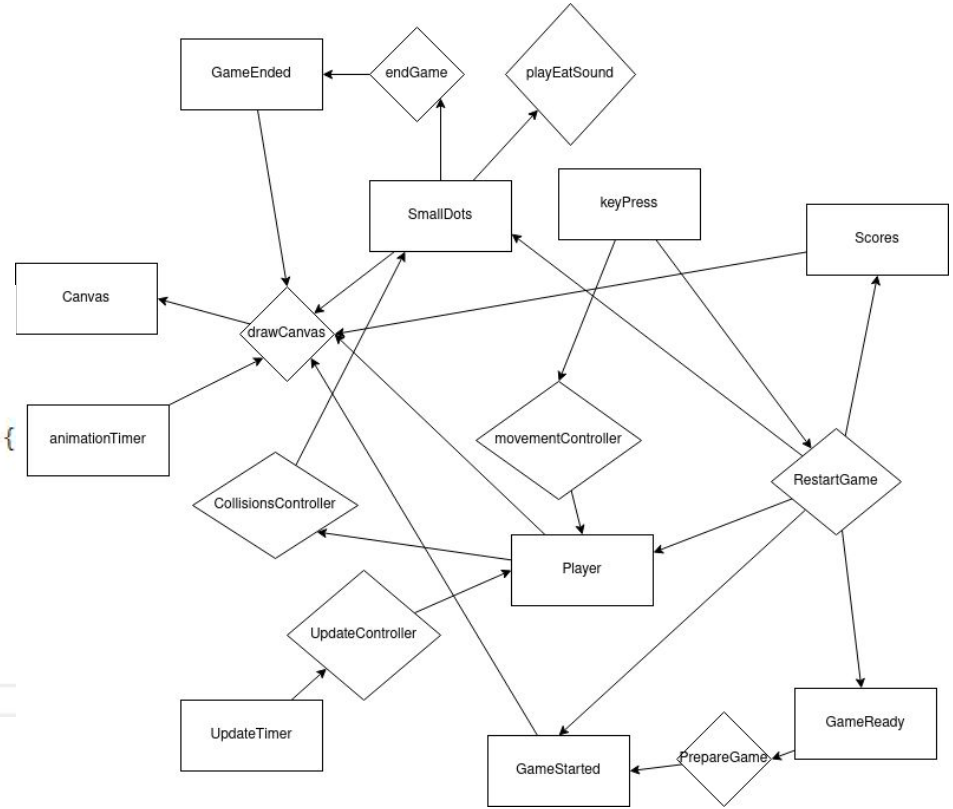
```
// actualizamos los jugadores      Julián, ayer • pacman …
if(!gameStarted.getValue()) {
  // Si aun no empieza no hacer nada
  return;
}
players.forEach((player, i) => {
  const { X, Y, DIR } = player.getValue();
  let newX = X, newY = Y;
  switch(DIR) {
    case DIRECTIONS.UP:
      newY -= Y_PLAYER_DELTA;
      if(map[Math.round(X)][Math.ceil(newY - 1)] === OBSTACLE_CHAR) {
        newY = Y;
      }
      break;
    case DIRECTIONS.DOWN:
      newY += X_PLAYER_DELTA;
      if(map[Math.round(X)][Math.floor(newY + 1)] === OBSTACLE_CHAR) {
        newY = Y;
      }
      break;
    case DIRECTIONS.LEFT:
      newX -= X_PLAYER_DELTA;
      if(map[Math.ceil(newX - 1)][Math.round(Y)] === OBSTACLE_CHAR) {
        newX = X;
      }
      break;
    case DIRECTIONS.RIGHT:
      newX += X_PLAYER_DELTA;
      if(map[Math.floor(newX + 1)][Math.round(Y)] === OBSTACLE_CHAR) {
        newX = X;
      }
      break;
  }
  player.next({
    X: newX,
    Y: newY,
    DIR,
  })
})
}
```



movement/update

colisiones

```
function collisionController() {
  for(let i=0; i<players.length; i++) {
    const { X, Y } = players[i].getValue();
    const positions = smallDots.getValue();
    if(positions[Math.round(X)][Math.round(Y)] === SMALL_DOT_CHAR) {
      positions[Math.round(X)][Math.round(Y)] = NOTHING_CHAR;
      smallDots.next(positions);
      scores[i].next(scores[i].getValue() + SMALL_DOT_SCORE);
      break;
    }

  }
}
```



Aprendizajes

- Ejecución en modo lazy de los streams es realmente útil cuando tenemos un pedazo de código que queremos se ejecute dado un determinado evento, evitando el uso de *if* y *while* para manejar el flujo de una aplicación.
- Creación de Observables a partir Observables ya existentes facilita el desarrollo, en especial en situaciones que se quieren verificar condiciones que involucran a más de un stream.
- El enfoque funcional que tiene la librería RxJS permite realizar diversas acciones sobre los Observables de una forma simple y clara.
- Workaround al uso de threads aprovechando los eventos de html.