Programación Reactiva con RxJS

Benjamín Vicente
José Madriaza
Jose Antonio Castro

Programación Reactiva

Programación Reactiva

- Paradigma **declarativo** para trabajar con **flujos** de datos
- La propagación de datos es “automática”

https://en.wikipedia.org/wiki/Reactive_programming

Programación Reactiva

```
let a = 10
let b ← a + 1
a = 20
Assert.AreEqual(b, 21)
```

```
let a = 10
$: b = a + 1
a = 20
Assert.AreEqual(b, 21)
```

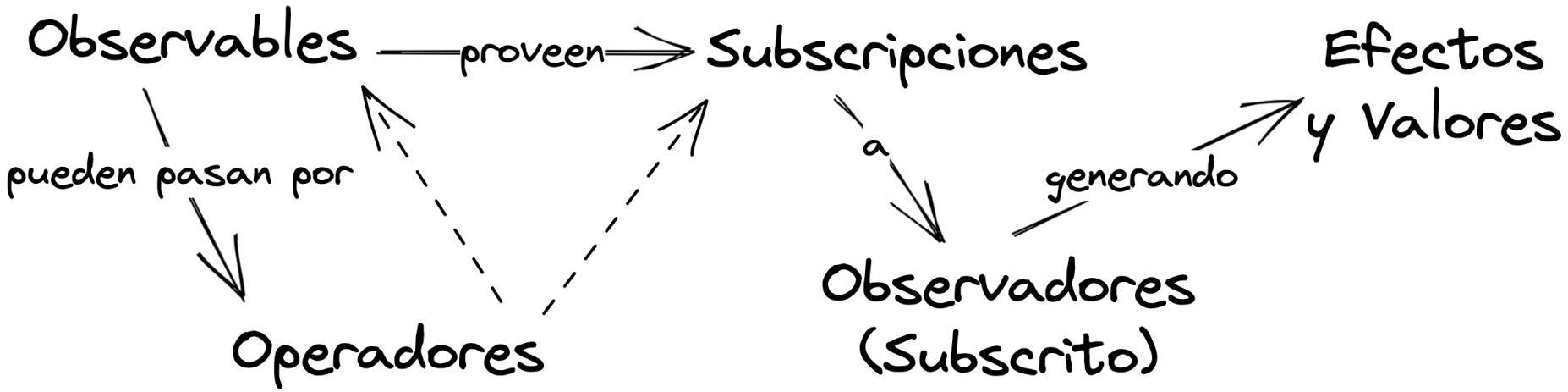
```
let a = ref(10)
let b = computed(a.value + 1)
a.value = 20
Assert.AreEqual(b.value, 21)
```

Se define la **relación** entre a y b

https://paulstovell.com/reactive-programming

https://en.wikipedia.org/wiki/Reactive_programming

Programación Reactiva



ReactiveX y RxJS

ReactiveX

- API **funcional** para programación asíncrona
- Considerada **menos opinada** que otros estilos
- Implementada en varios lenguajes de programación

https://reactivex.io/intro.html#:~:text=Observables%20Are%20Less%20Opinionated

RxJS

- Implementación de ReactiveX en JavaScript
- Provee la habilidad de producir **funciones puras**

https://rxjs.dev/guide/overview#purity

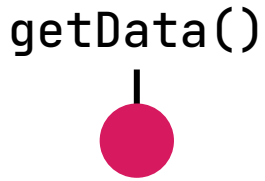
Programación ReactiveX se trabaja con `Observables`

	single items	multiple items
synchronous	`T getData()`	`Iterable<T> getData()`
asynchronous	`Future<T> getData()`	`Observable<T> getData()`

https://reactivex.io/intro

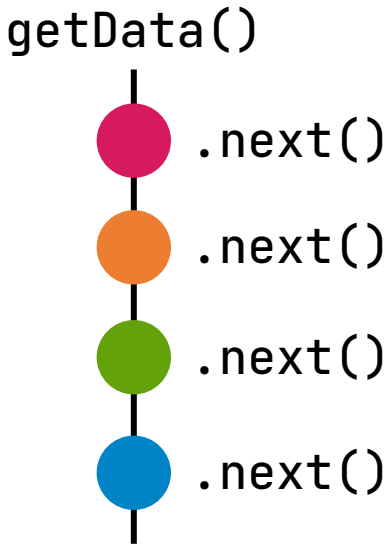
	single items	multiple items
synchronous	`T getData()`	`Iterable<T> getData()`
asynchronous	`Future<T> getData()`	`Observable<T> getData()`

Síncrono + Único



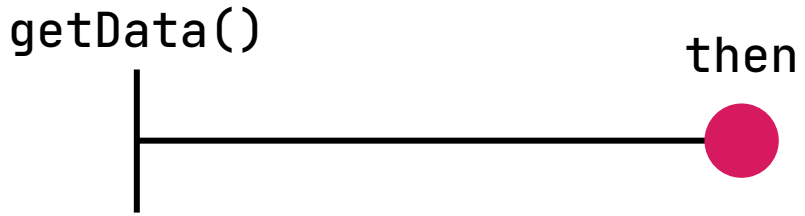
	single items	multiple items
synchronous	`T getData()`	`Iterable<T> getData()`
asynchronous	`Future<T> getData()`	`Observable<T> getData()`

Síncrono + Múltiple



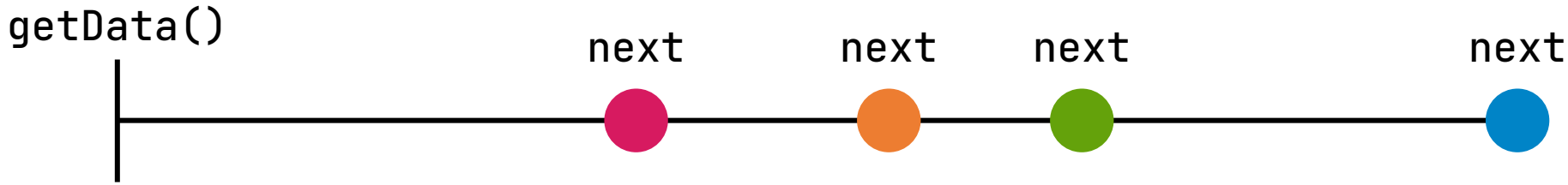
	single items	multiple items
synchronous	`T getData()`	`Iterable<T> getData()`
asynchronous	`Future<T> getData()`	`Observable<T> getData()`

Asíncrono + Único



	single items	multiple items
synchronous	`T getData()`	`Iterable<T> getData()`
asynchronous	`Future<T> getData()`	`Observable<T> getData()`

Asíncrono + Múltiple



Estos diagramas se les llama “Marble Diagrams”

usar al
tener vs llegar
los datos

Iterable

```
getDataFromLocalMemory()
  .skip(10)
  .take(5)
  .map({ s -> return s + " transformed" })
  .forEach({ println "next => " + it })
```

Observable

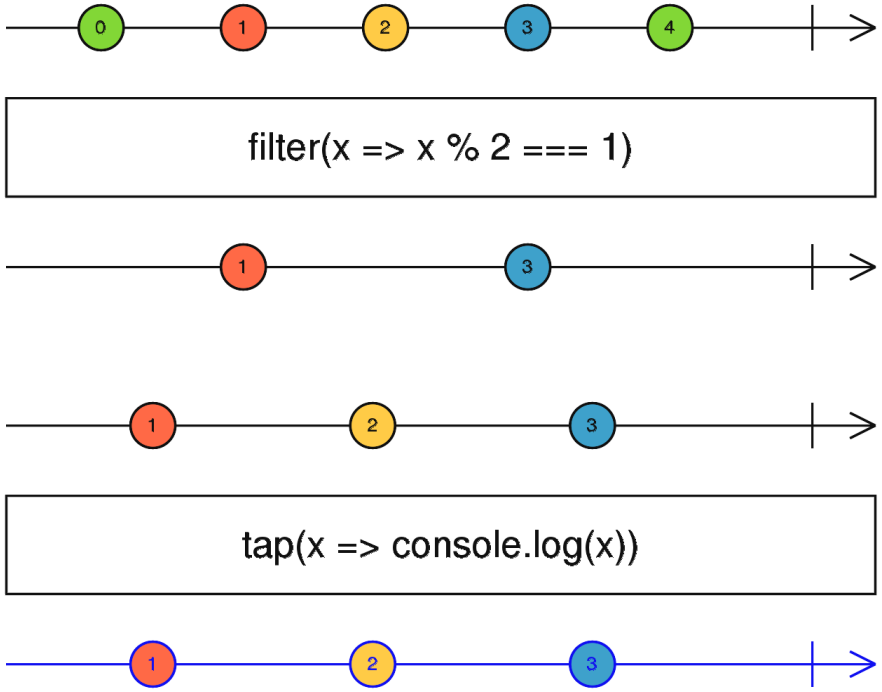
```
getDataFromNetwork()
  .skip(10)
  .take(5)
  .map({ s -> return s + " transformed" })
  .subscribe({ println "onNext => " + it })
```

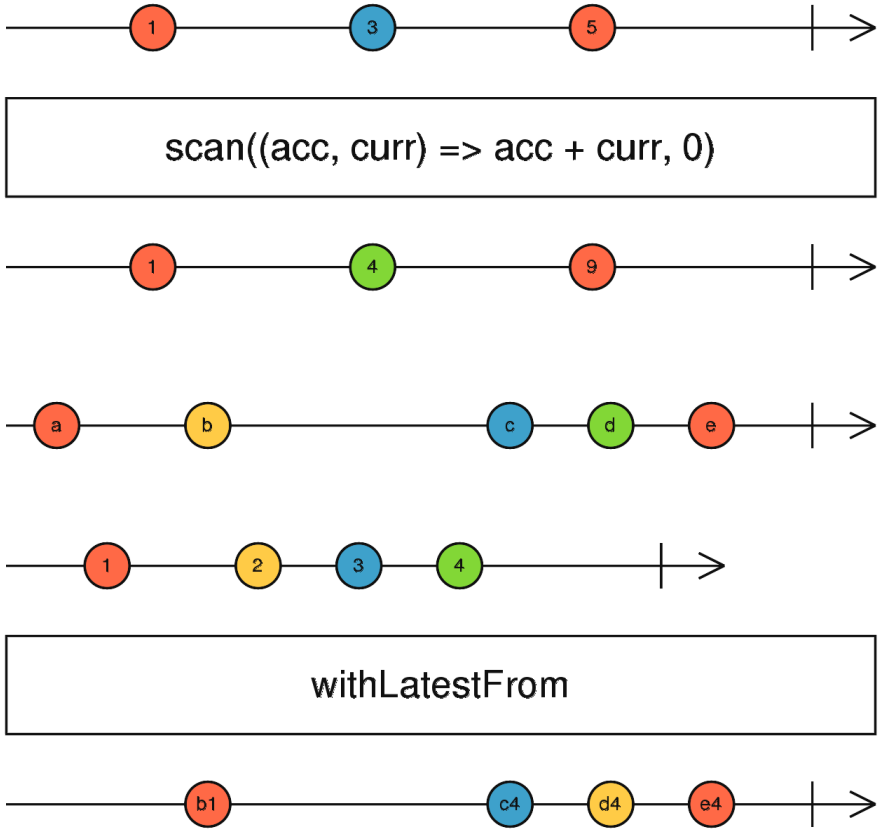
Operators

- Funciones que retornan Observables
- Hay 2 tipos: Creation y Pipablle

https://reactivex.io/documentation/operators

Ejemplos de Pipable Operators





https://rxjs.dev/api
https://rxmarbles.com

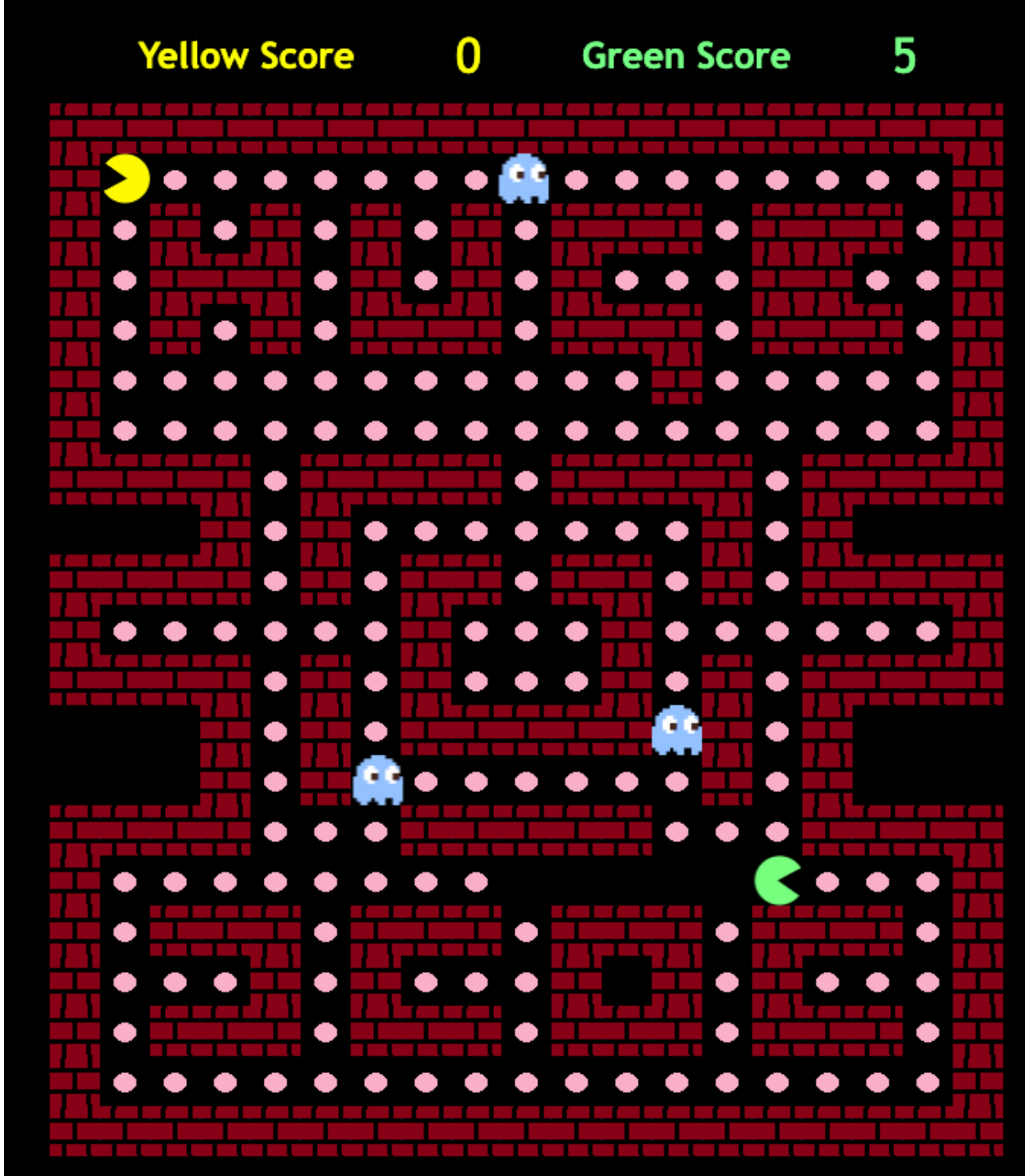
Implementación del problema

Problema a resolver

- Variación a PacMan reactiva y funcional utilizando RxJS
- Juego para al menos 2 jugadores y con malos a evitar

https://docs.google.com/document/d/1-Ofho-I0H5zkKQ_d4nxb-ot9VVEfuuLGjEW699NFbQw

Demo



Solución

```
const btn = document.getElementById("game-button");

fromEvent(btn, "click").subscribe(() => {
  btn.style.display = "none";

  const directionsObservable = fromEvent(window, "keydown").pipe(
    filter(onlyArrowKeys),
    tap(preventDefault),
    map(asUserAndDirection),
    startWith(initialDirections),
    scan(updateDirection)
  );

  interval(350)
    .pipe(
      takeCurrentState(directionsObservable),
      startWith(initGameState(defaultMap)),
      scan(updateGameState(defaultMap)),
      takeWhile(gameState => !gameState.finished, true),
    ).subscribe({
      next: renderToDom({ map: defaultMap }),
      complete: () => {
        btn.style.display = "block";
        btn.textContent = "Press To Restart!";
      }
    })
});
```

```
// Componente de PacMan
function createPacManComponent(mapElement, index = 0) {
  // Setup
  const pacManSubject = new Subject();

  const pacManElement = document.createElement("div");
  const pacManImg = document.createElement("div");
  pacManElement.appendChild(pacManImg);
  mapElement.appendChild(pacManElement);

  const scoreValue = document.querySelector(`#score-p${index}`);

  const pacManSubscription = pacManSubject.subscribe(({ position, direction, alive, score }) => {
    // Update
    const { x, y } = position;
    pacManElement.style.transform = entityTransform({ x, y });
    pacManImg.style.transform = `rotate(${directionToRotation[direction]}deg)`;
    setTimeout(() => scoreValue.innerHTML = score, 150);

    if (!alive) {
      pacManSubscription.unsubscribe();
      setTimeout(() => pacManElement.classList.remove("pacman"), 150);
    }
  });

  return pacManSubject;
}
```

```
gameState.players.forEach((player, index) =>
    playersPacsSubjects[index].next(player);
```

```css
.pacman, .ghost {
  position: absolute;
  width: 33px;
  height: 33px;
  /* Movimiento continuo */
  transition: transform 0.35s linear;
}
```