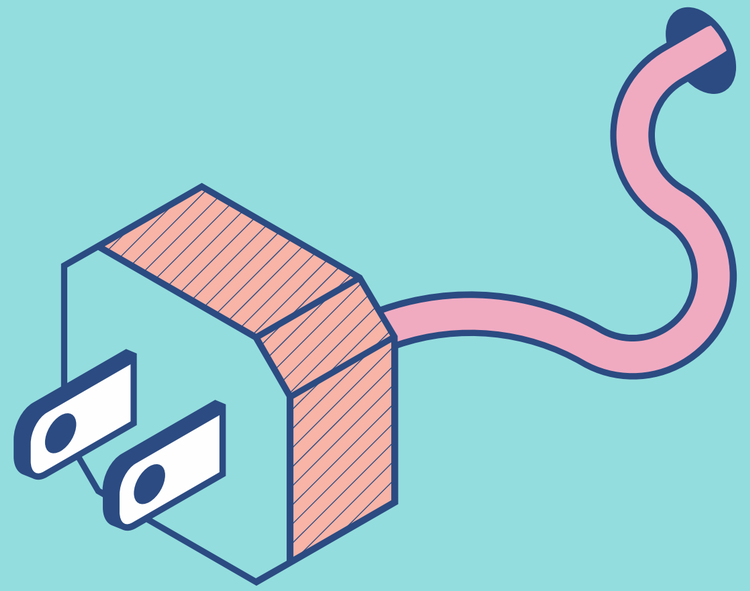
IIIC3548 - DAAW

Tarea 2 - Programacion Reactiva

Grupo 7: Matías Cadile, Matías Soto y José Luco

RxJS Library

Asynchronous data streams



Event-based reactions

Elementos centrales

1 OBSERVABLE

Representa la idea de una colección invocable de valores o eventos futuros.

2 OBSERVER

Es una colección de callbacks que escucha los valores entregados por el Observable.

3 SUBSCRIPTION

Representa la ejecución de un Observable, es usada para cancelar la ejecución.

Elementos centrales

4 OPERATORS

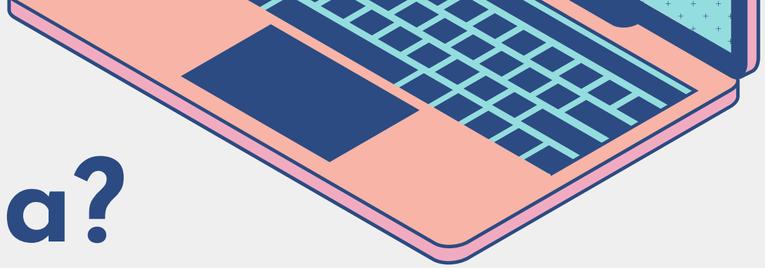
Son funciones que facilitan la programación funcional. Permite tratar las colecciones con operaciones como map, filter, concat, reduce, etc.

5 SUBJECT

Equivalente a un EventEmitter. Permite difundir un valor o evento a múltiples Observadores.

6 SCHEDULERS

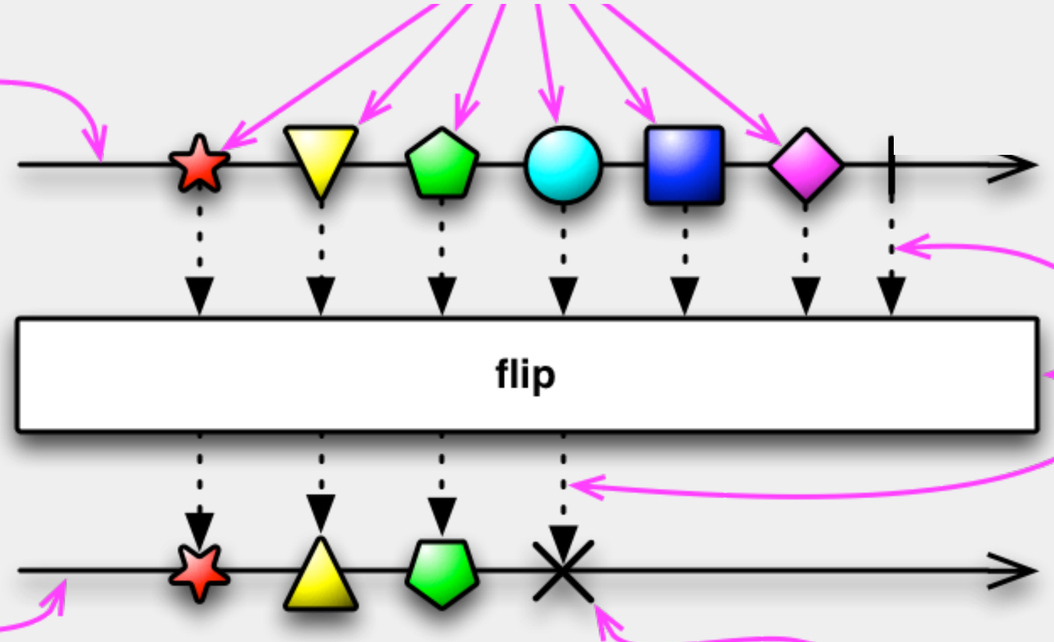
Despachadores centralizados para controlar la concurrencia, coordinar cómputo



¿Cómo funciona?

Flujo del *Observable* *(Data Stream)*

Emisiones del *Observable*

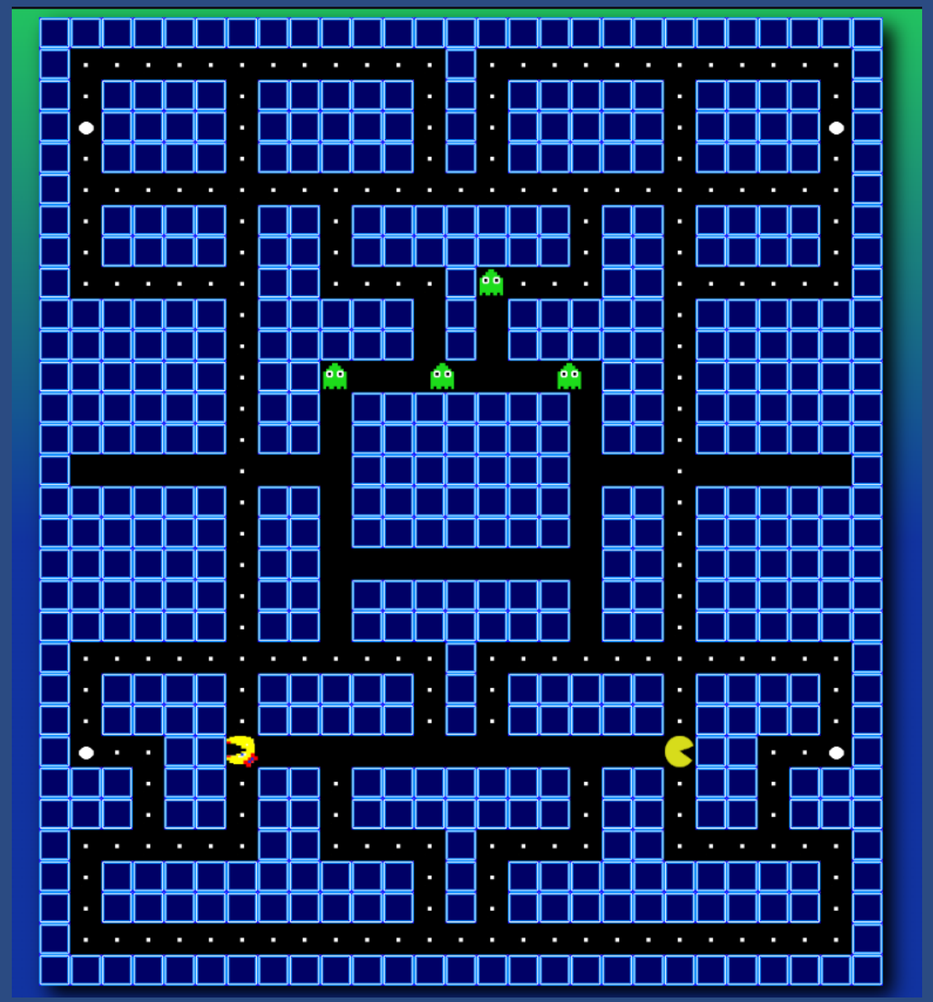


"Transformación" a través de un operador

Observable creado como salida de la transformación

Si este O*bservable* arroja error, el flujo de trasnformaciones termina

Tarea 2

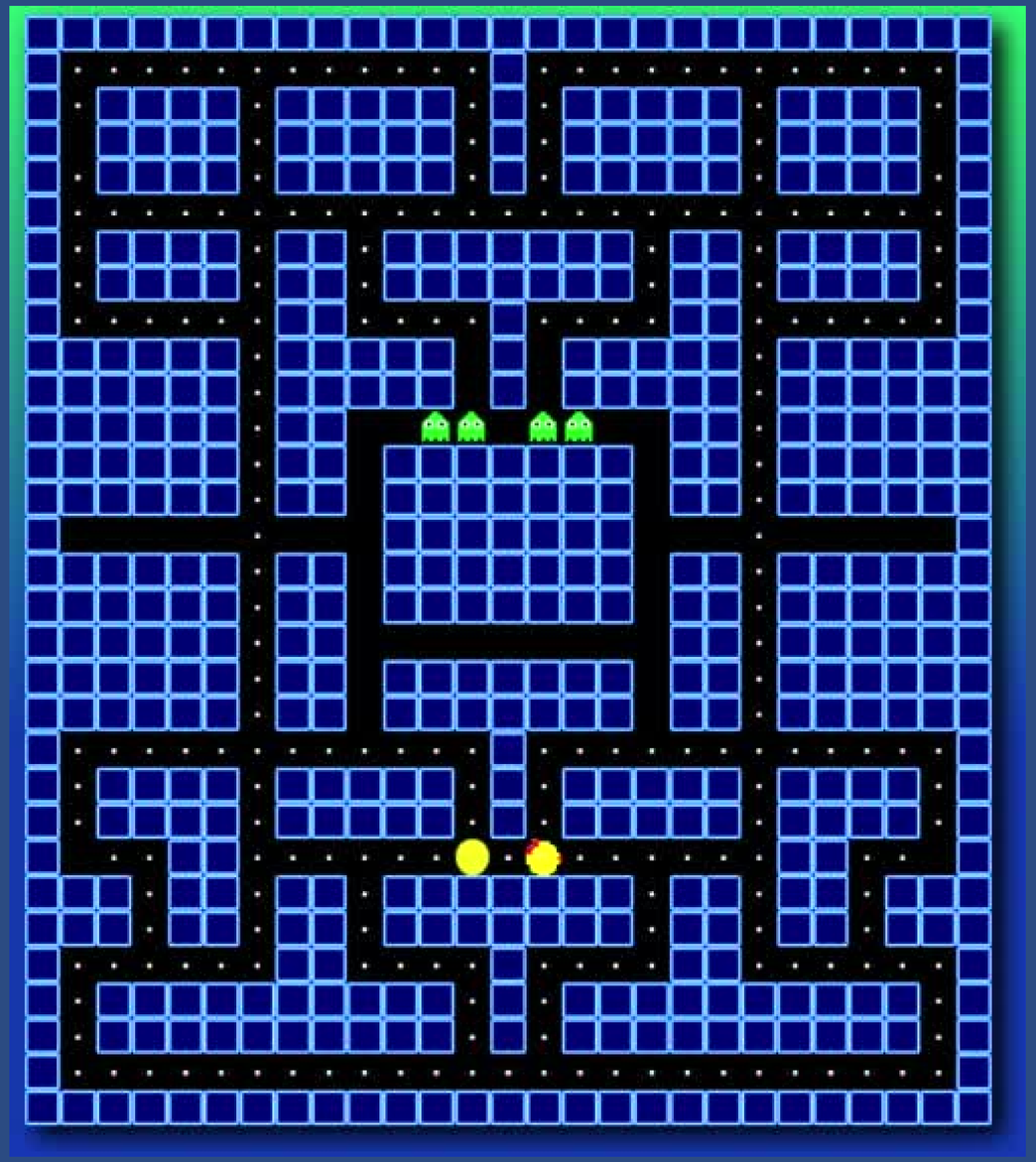


Pacman usando la librería RxJs.

Aproximación

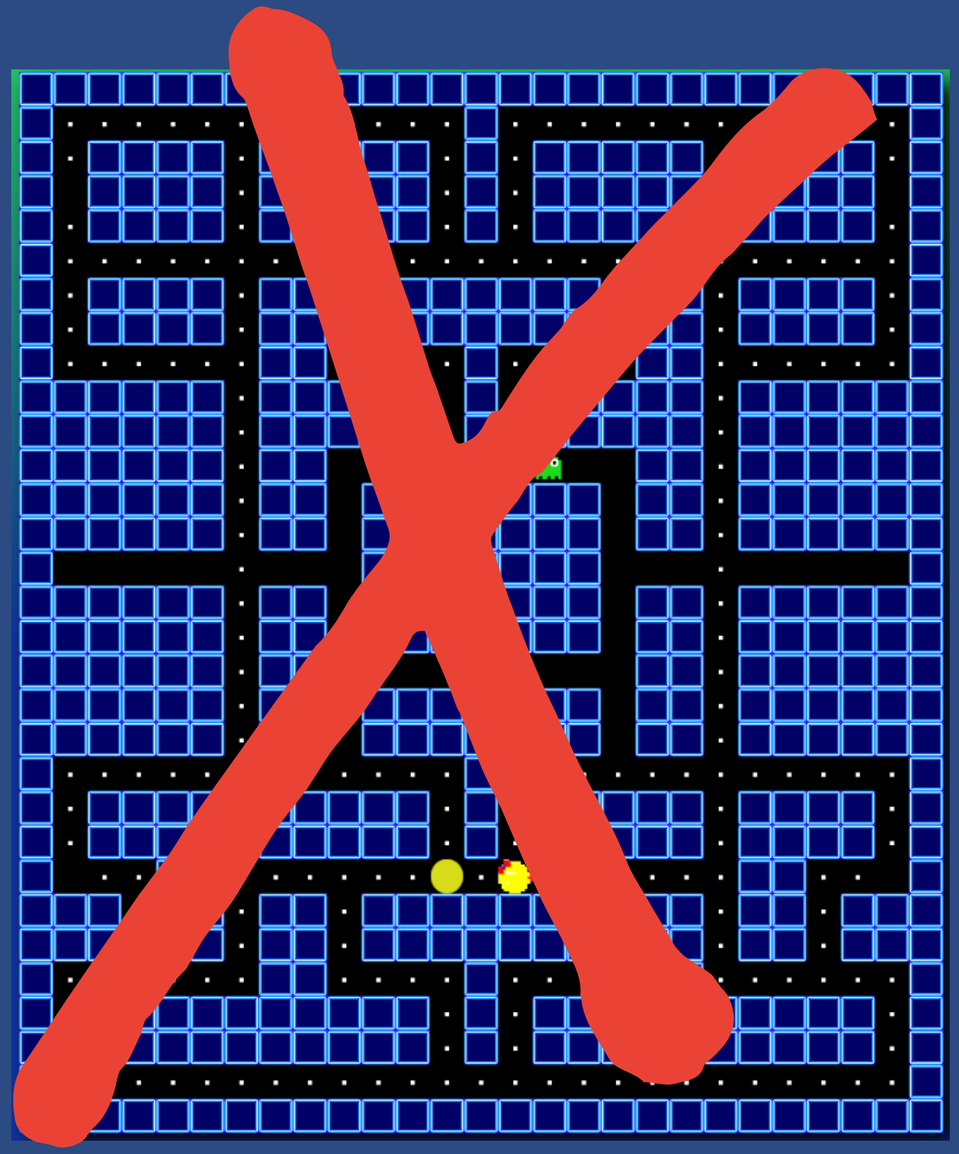
1

Realizar el programa en lenguaje orientado a objetos y luego adapatar la solucion a una con RxJS Library



Aproximación 1

Cuando se tiene un programa más extenso no es viable esta aproximación.



Aproximación 2

Identificar cuales son los observable, fuentes de los data stream y emepzar a operar sobre estos

Aproximación 2

Logramos encaminarnos a una solución funcional al problema. Ej: Programamos un frontEvent que capturaba un evento y luego creaba un observable en base a el como fue gatillado .

```
const input = fromEvent(document, "keydown").pipe(
  map((e) => {
    var movement = KEY_TO_SPEED[e.key];
    if (movement !== undefined) {
      switch (movement.player) {
        case 1:
          //PLAYER 1
          stateSubjectOne.next(movement.speed);
          break;
        case 2:
          // PLAYER 2
          stateSubjectTwo.next(movement.speed);
          break;
      }
    }
    return e;
  })
);
```

Aproximación 2

Creamos un Scheduler que controlaba cuando se comienza a enviar las notificaciones y cuando estas regresaban

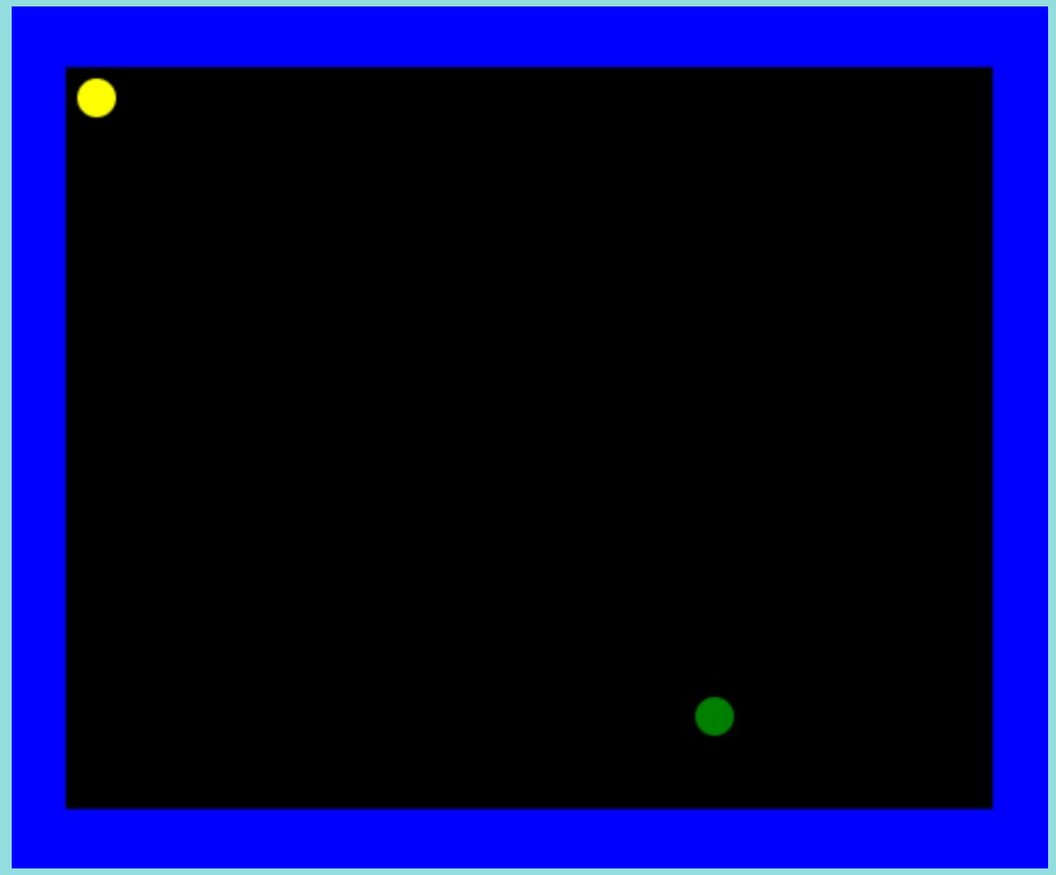
```
/* Ticker */

const TICKER_INTERVAL = 17;

const ticker$ = Rx.Observable.interval(
  TICKER_INTERVAL,
  Rx.Scheduler.requestAnimationFrame
)
  .map(() => ({
    time: Date.now(),
    deltaTime: null,
  }))
  .scan((previous, current) => ({
    time: current.time,
    deltaTime: (current.time - previous.time) / 1000,
  }));
```

Implementaciones realiazadas

- Implementación de un Scheduler
- Combinación de observables (merge, combineLatest)
- Encadenamiento de operadores con Pipe
- Implementación de Subjects como observables para el manejo de estados



Conclusiones

El manejo de estados y de eventos como IO, se simplifica bastante con la programación funcional reactiva porque además de abstraerte de ciertos procesos, ayuda a tener un código más modularizado y puro, permitiendo el manejo e idenficación de errores más expedita