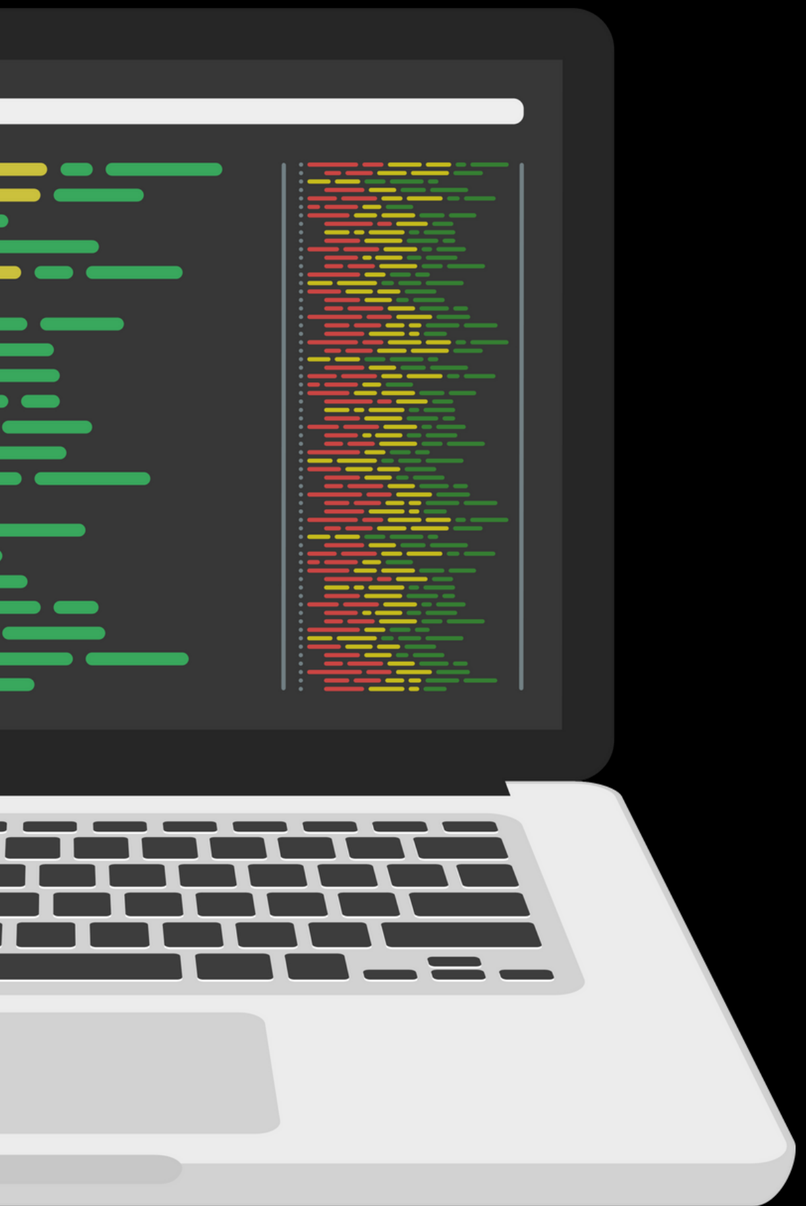


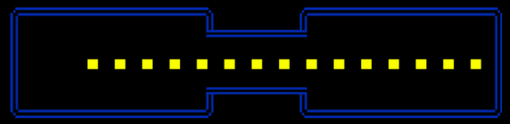
TAREA 2

Diseño avanzado de aplicaciones web

Grupo 3

NUESTRO PAC-MAN

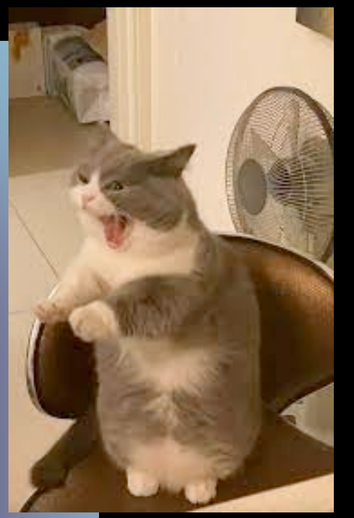
- 2 Jugadores fijos.
- Un jugador utiliza las teclas: → ↑ ← ↓ El otro jugador utiliza las teclas: **a w s d**
- Sólo un nivel (se gana  o se pierde )
- El juego es cooperativo, por lo que el puntaje se acumula entre ambos jugadores.
- Si uno de los jugadores muere, el juego se acaba.



¿POR QUE PROGRAMACIÓN REACTIVA?

- Permite emitir y/o recibir data de forma continua en cualquier momento.
- Permite una mayor capacidad de respuesta logrando una mejor performance a las solicitudes de los usuarios.
- Mejora la calidad y eficiencia del desarrollo a través de un código más limpio gracias a la gran variedad de operadores disponibles.

```
const keyUp = rxjs.fromEvent(document, 'keyup').pipe(
  rxjs.filter((e) => e?.keyCode in keyMap),
  rxjs.takeUntil(keyUpSubject)
).subscribe((e) => {
  currentPlayerSubject.next(p1Movements.includes(e.keyCode) ? player1 : player2);
  movePacman(e, squares, currentPlayerSubject.value, squaresSubject);
});
```



1er Problema: ¿Como de-suscribir un evento dentro de la función de suscripción?

Solución: Subject + takeUntil

La suscripción se mantiene hasta que el Subject gatilla un evento, y este puede ser desde cualquier función o scope.

1

```
const keyUpSubject = new rxjs.Subject();
```

2

```
const keyUp = rxjs.fromEvent(document, 'keyup').pipe(
    rxjs.filter((e) => e?.keyCode in keyMap),
    rxjs.takeUntil(keyUpSubject)
  ).subscribe((e) => {
```

3

```
keyUpSubject.next();
```

```
1  const keyUp = rxjs.fromEvent(document, 'keyup').pipe(
2    rxjs.filter((e) => e?.keyCode in keyMap),
3    rxjs.takeUntil(keyUpSubject)
4  ).subscribe((e) => {
5    currentPlayerSubject.next(p1Movements.includes(e.keyCode) ? player1 : player2);
6    movePacman(e, squares, currentPlayerSubject.value, squaresSubject);
7  });
```

2do Problema: ¿Como actualizar quien era el jugador que se está moviendo?

Solución: BehaviorSubject

Extiende de Subject en donde además de emitir eventos en tiempo real a subscriptores, este almacena el valor actualizado de la variable.

```
1  const currentPlayerSubject = new rxjs.BehaviorSubject(player1);
```

Con BehaviourSubject podemos:

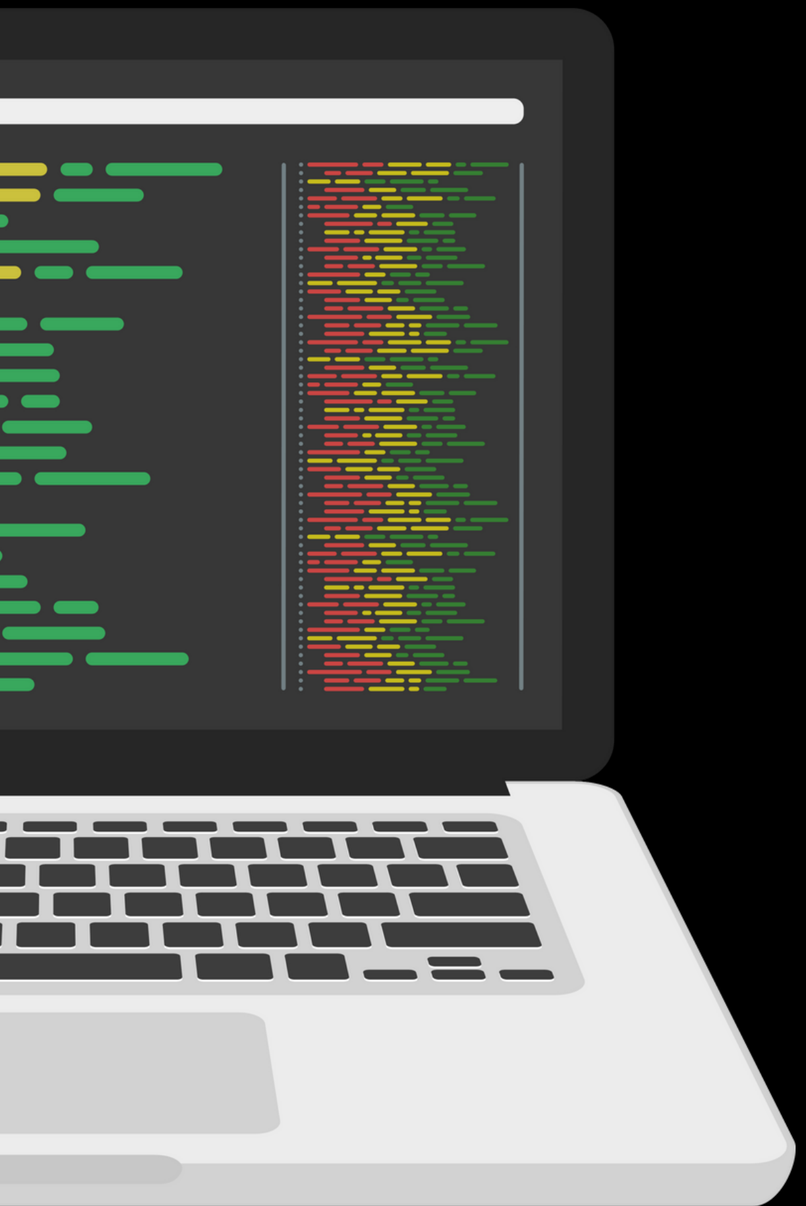
1. Actualizar el valor con un evento
2. Utilizarlo en contextos fuera del subscribe

```
1  currentPlayerSubject.next(p1Movements.includes(e?.keyCode) ? player1 : player2);
```

```
1  rxjs.map(newSquares => ({ squares: newSquares, currentPlayer: currentPlayerSubject.value })),
```

```
squaresSubject.pipe(
  rxjs.map(newSquares => ({ squares: newSquares, currentPlayer: currentPlayerSubject.value })),
).subscribe(({ squares, currentPlayer }) => {
  runGameFunctions(squares, currentPlayer);
});
```

Concatenación de estados en la gatillación de eventos.

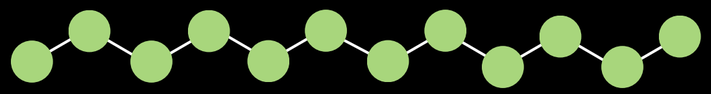


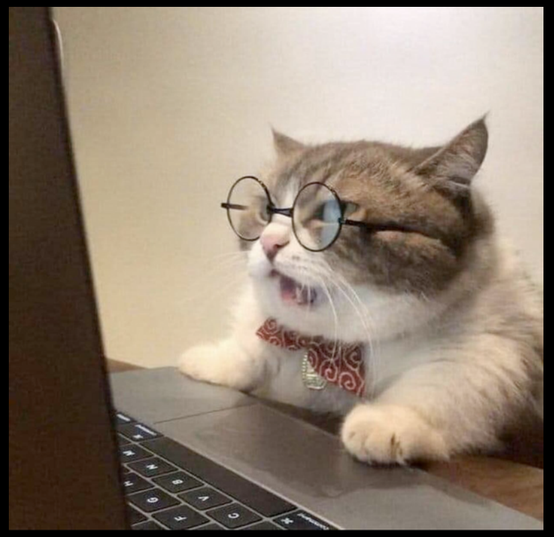
DEMO

Grupo 3

APRENDIZAJES

- Código más sencillo y simple al utilizar reactividad.
- Completo control por el usuario para usar reactividad segun sus necesidades.
- RxJS tiene gran variedad de funciones para manejar stream de datos. Agradecidos con los Subject.
- No forzamos el uso de funciones en todo el código. Por ejemplo al modelar Pacman o los Ghosts.





Grupo 3