GRUPO 10

PROGRAMACIÓN REACTIVA

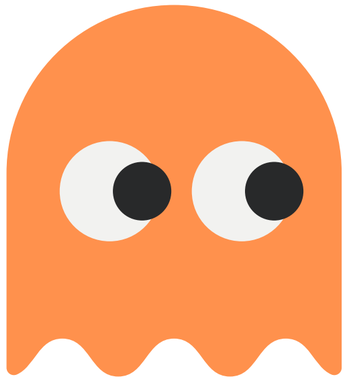
DISEÑO AVANZADO DE APLICACIONES WEB

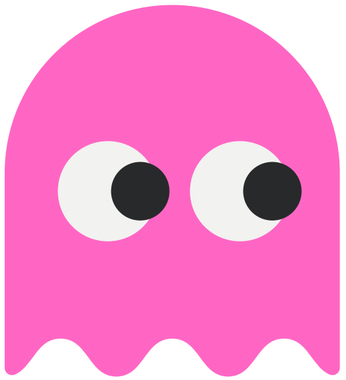
RAFAELA KARA - CRISTOBAL MUÑOZ - FLAVIO TARSETTI

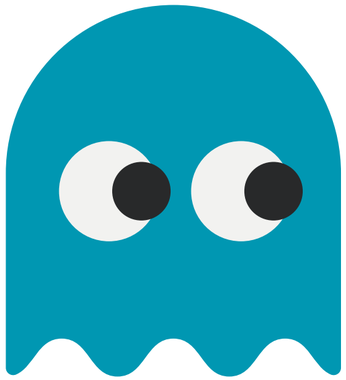
ALCANCE DE LA TAREA

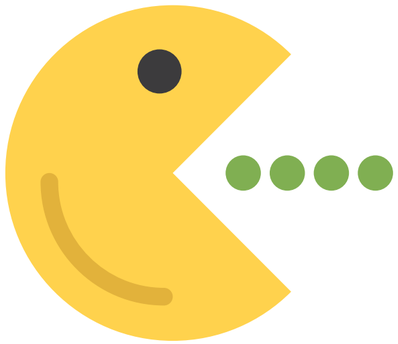
- Multijugador
- Fantasmas
- Game Over
- Mapas

- Vidas
- Frutas
- Portal
- Comer fantasmas
- Puntaje









OBSERVABLES IMPLEMENTADOS

JUGADOR

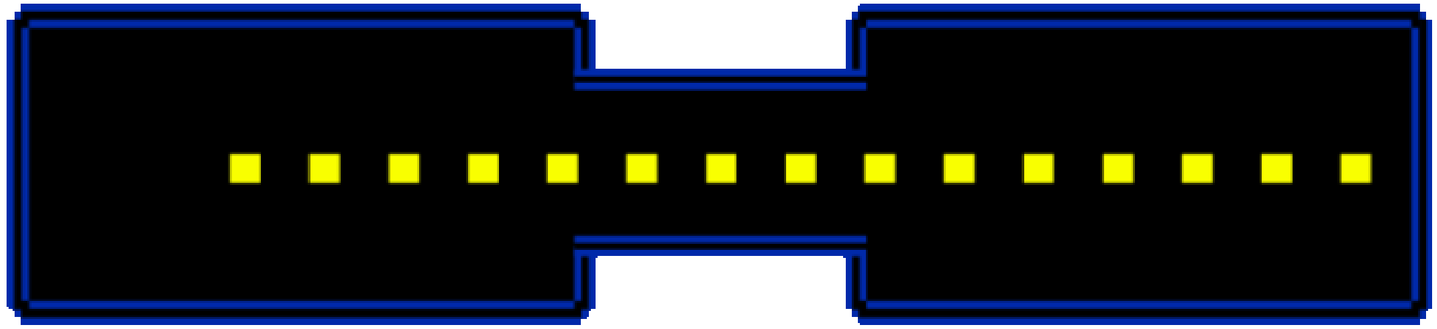
- Direcciones individuales
- Movimiento completo del jugador
- Colisiones con fantasmas

FANTASMAS

- Movimiento aleatorio

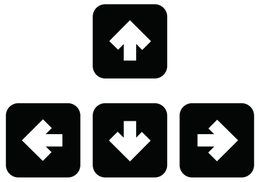
SESIÓN DEL JUEGO

- Verificación del término de la partida



JUGADOR

DIRECCIONES INDIVIDUALES



```
function DirectionObservable(dirKey, nextMoveX, nextMoveY) {
        return fromEvent(document, "keydown")
            .pipe(
                filter(event => event.code == dirKey),
                filter(_ => dirKey !== player.currDir),
                filter(_ => checkNoCollision(player.x + nextMoveX, player.y + nextMoveY)),
                tap(_ => {
                    const intervalObservable = interval(PLAYER_INTERVAL_LENGTH)
                        .pipe(
                            tap(_ => {
                                if (checkNoCollision(player.x + nextMoveX, player.y + nextMoveY)) {
                                    player.x += nextMoveX
                                    player.y += nextMoveY
                                    GAME.draw()
                                }
                                checkDotCollection(player.x, player.y)
                            })
                        ).subscribe()
                    player.activeSubscriptions.push(intervalObservable)
                    player.currDir = dirKey
                })
            )
    }
```

1. Recibimos input del teclado
2. Filtramos por tecla en la dirección que queremos
3. Verificamos si hubo un cambio de dirección
4. Verificamos que no haya colisión
5. Generamos loop infinito para que el jugador se mueva en la misma dirección

JUGADOR

MOVIMIENTO COMPLETO

1. Asociamos observable a cada tecla de dirección y su desplazamiento

2. Unimos cuatro observables en uno solo

3. Interrumpimos todos los movimientos previos

```
1
1  const UpObservable = DirectionObservable(upKey, 0, -MOVEMENT)
2  const DownObservable = DirectionObservable(downKey, 0, MOVEMENT)
3  const LeftObservable = DirectionObservable(leftKey, -MOVEMENT, 0)
4  const RightObservable = DirectionObservable(rightKey, MOVEMENT, 0)
```

```
1  const MovementObservable = merge(UpObservable, DownObservable, LeftObservable, RightObservable) 2
2          .pipe(
3              tap(_ ⇒ {
4                  const lastId = player.activeSubscriptions.pop()
5  3               player.activeSubscriptions.forEach((intervalObservable) ⇒ intervalObservable.unsubscribe())
6                  player.activeSubscriptions = [lastId]
7              }),
8          )
9          .subscribe()
```

JUGADOR

COLISIONES CON FANTASMAS

```
1  player.ghostSubscription = interval(PLAYER_INTERVAL_LENGTH)
2      .pipe(
3          filter(_ ⇒ GAME.ghosts.some(
4              ghost ⇒ checkCollisionPlayerGhosts(player.x, player.y, ghost.x, ghost.y)
5          )),
6          tap(_ ⇒ {
7              GAME.players = GAME.players.filter(p ⇒ p.name =/= player.name)
8              player.activeSubscriptions.forEach((intervalObservable) ⇒ intervalObservable.unsubscribe())
9              MovementObservable.unsubscribe()
10         })
11     )
12     .subscribe()
```

1 Generamos loop infinito en el tiempo

2 Verificamos si jugador colisiona con algún fantasma

3 Eliminamos al jugador de la sesión

4 Terminamos todas sus subscripciones activas

FANTASMAS

MOVIMIENTO ALEATORIO

1. Generamos loop infinito en el tiempo

2. Si el fantasma está en una intersección, elegimos una dirección al azar

3. Verificamos si nueva dirección provocará colisión con pared

```
1  ghost.subscription = interval(GHOST_INTERVAL_LENGTH)                    1
2          .pipe(
3              map(_ ⇒ {
4                  if (checkIntersection(ghost.x, ghost.y)) {              2
5                      return GAME.directions[Math.floor(Math.random() * GAME.directions.length)]
6                  }
7                  return ghost.direction
8              }),
9              map(dir ⇒ {
10                 let noCollision = false
11                 if (dir == 'U') {                                       3
12                     noCollision = checkNoCollision(ghost.x, ghost.y - MOVEMENT)
13                 } else if (dir == 'D') {
14                     noCollision = checkNoCollision(ghost.x, ghost.y + MOVEMENT)
15                 } else if (dir == 'L') {
16                     noCollision = checkNoCollision(ghost.x - MOVEMENT, ghost.y)
17                 } else if (dir == 'R') {
18                     noCollision = checkNoCollision(ghost.x + MOVEMENT, ghost.y)
19                 }
20                 if (noCollision) {
21                     ghost.direction = dir
22                 }
23                 return noCollision
24             }),
```

FANTASMAS

MOVIMIENTO ALEATORIO (CONTINUACIÓN)

```
            filter(x ⇒ x), 4
            tap(_ ⇒ {
5           if (ghost.direction == 'U') {
                ghost.y -= MOVEMENT
            } else if (ghost.direction == 'D') {
                ghost.y += MOVEMENT
            } else if (ghost.direction == 'L') {
                ghost.x -= MOVEMENT
            } else if (ghost.direction == 'R') {
                ghost.x += MOVEMENT
            }
            GAME.draw()
        })
    )
    .subscribe()
```

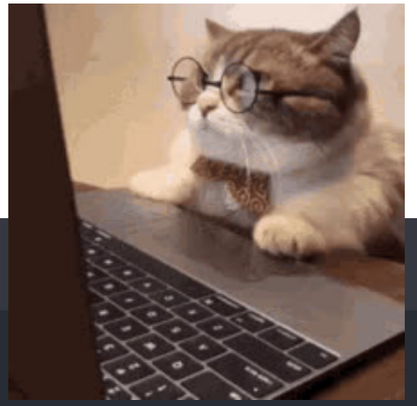
4. **Filtramos si nueva dirección es válida**

5. **Actualizamos posición del fantasma en el tablero**

SESIÓN DEL JUEGO

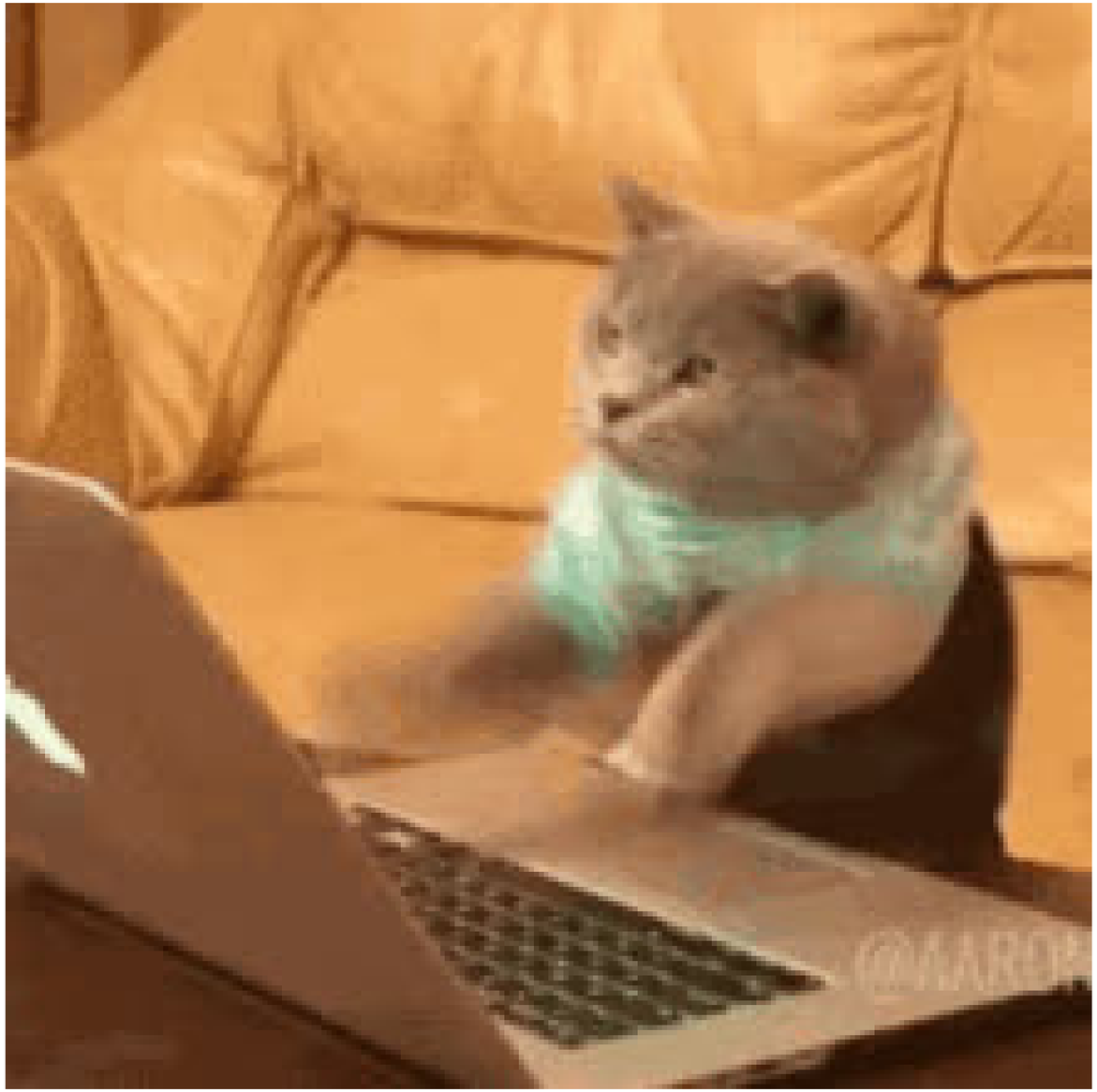
1. Generamos loop infinito en el tiempo
2. Filtramos si sesión ha comenzado y si quedan puntos o jugadores en tablero
3. Detenemos loop infinito
4. Detenemos todas las suscripciones activas
5. Mostramos pantalla de término

VERIFICACIÓN TÉRMINO DE PARTIDA



```
interval(PLAYER_INTERVAL_LENGTH) 1
        .pipe(
        2 filter(_ ⇒ GAME.startedSession),
          filter(_ ⇒ GAME.players.length == 0 || GAME.dots.length == 0),
          take(1), 3
          tap(_ ⇒ {
              GAME.ghosts.forEach(ghost ⇒ ghost.subscription.unsubscribe())
              GAME.players.forEach(player ⇒ {
          4       player.ghostSubscription.unsubscribe()
                  player.activeSubscriptions.forEach(sub ⇒ sub.unsubscribe())
              })
              ctx.fillStyle = "black"
              ctx.fillRect(0, 0, board.width, board.height)
              ctx.font = "40px Arial";
              ctx.fillStyle = "white"
              ctx.textAlign = "center";
              if (GAME.players.length == 0) {
          5       ctx.fillText("GAME OVER", board.width/2, board.height/2);
              } else if (GAME.dots.length == 0) {
                  ctx.fillText("YOU WIN", board.width/2, board.height/2);
              }
          }),
        ).subscribe()
```

DIFICULTADES



- **Entender los operadores de RxJS**
- **Determinar en qué partes del juego ocuparlos**
- **Fluidez y precisión del movimiento de los jugadores**
- **Eficiencia del juego y complejidad de las funciones**

DEMO