

# Algoritmos y Estructura de Datos 2

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

## Exorcismo Extremo TP2

Integrante	LU	Correo electrónico
Rosinov, Gaston Einan	37/18	<code>grosinov@gmail.com</code>
Schuster, Martin Ariel	208/18	<code>m.a.schuster98@gmail.com</code>
Panichelli, Manuel	72/18	<code>panicmanu@gmail.com</code>

### Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

## Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Módulo Juego</b>	<b>4</b>
<b>3. Módulo Mapa</b>	<b>23</b>
<b>4. Módulo Dirección</b>	<b>26</b>
<b>5. Módulo Acción</b>	<b>28</b>
<b>6. Diccionario Trie (<math>\alpha</math>)</b>	<b>32</b>

## 1. Introducción

A lo largo del TP, hacemos los siguientes reemplazos sintácticos:

- *jugador* es un *string*
- *fantasma* es un *vector(evento)*
- *pos* es  $\langle x : \text{nat}, y : \text{nat} \rangle$
- *evento* es  $\langle pos : \mathbf{pos}, dir : \mathbf{dir}, dispara? : \mathbf{bool} \rangle$
- *pasoDisparos* es  $\langle pasoDispFan : \text{nat}, pasoDispPJ : \text{nat} \rangle$

Suponemos que el módulo *vector* tiene una función

**Vectorizar:**  $\text{lista}(\alpha) \rightarrow \text{vector}(\alpha)$

esta función puede ser fácilmente implementada iterando y tomando referencia de los elementos de la lista y agregándolos atrás del vector.

La complejidad de esta función será  $\mathcal{O}(n)$  siempre y cuando  $\alpha$  se copie en  $\mathcal{O}(1)$ .

## 2. Módulo Juego

### Interfaz

**generos:** juego.

**se explica con:** JUEGO.

Exorcismo Extremo Operaciones básicas de Juego

// Generador

• INICIAR(in  $m$ : mapa, in  $pjs$ : conj(jugador), in  $eventosFan$ : vector(evento))  $\rightarrow res$  : juego

Pre  $\equiv \{\neg vacio(pjs) \wedge \neg vacio(f) \wedge (\forall e : evento)(esta?(e, eventosFan) \Rightarrow_L e.pos \in libres(m))\}$

Post  $\equiv \{res =_{obs} nuevo.Juego(m, pjs, eventosFan)\}$

Complejidad:  $\Theta(m^2 + \#pjs * |pjMasLargo| + loc.Jugadores + long(eventosFan)^2)$

Descripción: crea un nuevo juego con el mapa dado, un conjunto de jugadores, y los eventos de un fantasma.

// Operaciones pedidas

• INFOACTUALPJVIVOS(in  $j$ : Juego)  $\rightarrow res$  : conj(Tupla(jugador, pos, dir))

Pre  $\equiv \{true\}$

Post  $\equiv \{res =_{obs} infoActualPjVivos(j)\}$

Complejidad:  $\Theta(1)$

Descripción: devuelve un conjunto referencias a identidad, posicion y direccion actual de los jugadores vivos.

• INFOACTUALFANVIVOS(in  $j$ : Juego)  $\rightarrow res$  : conj(Tupla(pos, dir))

Pre  $\equiv \{true\}$

Post  $\equiv \{res =_{obs} infoActualFanVivos(j)\}$

Complejidad:  $\Theta(1)$

Descripción: devuelve un conjunto referencias a la información de los fantasmas que están vivos.

• INFOACTUALFANESPECIAL(in  $j$ : Juego)  $\rightarrow res$  : Tupla(pos, dir)

Pre  $\equiv \{true\}$

Post  $\equiv \{res =_{obs} infoActualFanEspecial(j)\}$

Complejidad:  $\Theta(1)$

Descripción: devuelve la posicion y direccion del fantasma especial.

• INFOACTUALFANVIVOSQUEDISP(in  $j$ : Juego)  $\rightarrow res$  : conj(Tupla(pos, dir))

Pre  $\equiv \{true\}$

Post  $\equiv \{res =_{obs} infoActualFanVivosQueDisp(j)\}$

Complejidad:  $\mathcal{O}(\#fv)$

Descripción: devuelve un conjunto con la información de los fantasmas que están vivos y disparan en el ultimo paso ejecutado en el juego.

• VIVO?(in  $j$ : juego, in  $pj$ : string)  $\rightarrow res$  : bool

Pre  $\equiv \{pj \in jugadores(j)\}$

Post  $\equiv \{res =_{obs} jugadorVivo(pj, j)\}$

Complejidad:  $\Theta(|pj|)$

Descripción: devuelve si un jugador está vivo

• EJECUTARACCION(in/out  $j$ : juego, in  $a$ : accion, in  $pj$ : jugador)

Pre  $\equiv \{j =_{obs} j_0 \wedge_L pj \in jugadores(j) \wedge_L jugadorVivo(pj, j) \wedge \neg esPasar(a)\}$

Post  $\equiv \{j =_{obs} step(j_0, a, pj)\}$

Complejidad:  $\mathcal{O}(|pj| + \#fv * m + \#jv)$

Descripción: actualiza con la acción  $a$  del jugador  $pj$ .

• PASARTIEMPO(in/out  $j$ : juego)

Pre  $\equiv \{true\}$

Post  $\equiv \{j =_{obs} pasar(j_0)\}$

Complejidad:  $\mathcal{O}(\#fv * m + \#jv)$

**Descripción:** actualiza sin acción de jugador.

• POSOCUPADASPORDISPAROSFAN(**in**  $j$ : juego)  $\rightarrow res$  : conj(posicion)

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} alcanceDisparosFantasmas(fantasmas(j), j)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve un conjunto de las posiciones afectadas por disparos de fantasmas en el último paso.

// Observadores del TAD

HABITACION(**in**  $j$ : juego)  $\rightarrow res$  : mapa

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} habitacion(j)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve la habitacion del juego

FANTASMAS(**in**  $j$ : juego)  $\rightarrow res$  : conj(fantasma)

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} fantasmas(j)\}$

**Complejidad:**  $\mathcal{O}(\#f * long(eventoMasLargo))$

**Descripción:** devuelve por referencia un conjunto de todos los fantasmas del juego

FANTASMAESPECIAL(**in**  $j$ : juego)  $\rightarrow res$  : fantasma

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} fantasmaEspecial(j)\}$

**Complejidad:**  $\Theta(\#f)$

**Descripción:** devuelve por referencia un conjunto de todos los fantasmas del juego

JUGADORES(**in**  $j$ : juego)  $\rightarrow res$  : conj(jugadores)

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} jugadores(j)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve por referencia un conjunto de todos los jugadores del juego

ACCIONES(**in**  $pj$ : jugador, **in**  $j$ : juego)  $\rightarrow res$  : secu(evt)

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} acciones(pj, j)\}$

**Complejidad:**  $\Theta(|pj|)$

**Descripción:** devuelve por referencia un conjunto de todos los jugadores del juego

Exorcismo ExtremoEspecificación de las operaciones auxiliares utilizadas en la interfaz

**TAD** Juego Extendido( $\alpha$ )

**otras operaciones**

infoActualPjVivos : Juego  $\rightarrow$  conj(Tupla(jugador, posicion, direccion))

infoActualPjVivosAux : Juego  $\times$  conj(jugador)  $\rightarrow$  conj(Tupla(jugador, posicion, direccion))

infoActualFanVivos : Juego  $\rightarrow$  conj(Tupla(posicion, direccion))

infoActualFanVivosAux : Juego  $\times$  conj(fantasma)  $\rightarrow$  conj(Tupla(posicion, direccion))

infoActualFanEspecial : Juego  $\rightarrow$  Tupla(posicion, direccion)

infoActualFanVivosQueDisp : Juego  $\rightarrow$  conj(Tupla(posicion, direccion))

infoActualFanVivosQueDisAux : Juego  $\times$  conj(fantasmas)  $\rightarrow$  conj(Tupla(posicion, direccion))

**axiomas**

infoActualPjVivos( $j$ )  $\equiv$  infoActualPjVivosAux( $j$ , jugadores( $j$ ))

```

infoActualPjVivosAux(j, js) ≡ if  $\emptyset?(js)$  then  $\emptyset$  else
    if jugadorVivo(dameUno(js), j) then
        Ag( $\langle posJugador(dameUno(js), j), dirJugador(dameUno(js), j) \rangle$ ,
            infoActualPjVivosAux(j, sinUno(js)))
    else
        infoActualPjVivosAux(j, sinUno(js))
    fi fi

infoActualFanVivos(j) ≡ infoActualPjVivosAux(j, jugadores(j))
infoActualFanVivosAux(j, fs) ≡ if  $\emptyset?(fs)$  then  $\emptyset$  else
    if fantasmaVivo(dameUno(fs), j) then
        Ag( $\langle posFantasma(dameUno(fs), j), dirFantasma(dameUno(fs), j) \rangle$ ,
            infoActualFanVivosAux(j, sinUno(fs)))
    else
        infoActualFanVivosAux(j, sinUno(fs))
    fi fi

infoActualFanEspecial(j) ≡  $\langle posFantasma(fantasmaEspecial(j), j), dirFantasma(fantasmaEspecial(j), j) \rangle$ 
infoActualFanVivosQueDisp(j) ≡ infoActualFanVivosAux(j, infoActualFanVivosQueDisAux(j, fantasmas(j)))
infoActualFanVivosQueDispAux(j, fs) ≡ if  $\emptyset?(fs)$  then  $\emptyset$  else
    if fantasmaVivo(dameUno(fs), j)  $\wedge$  disparando(dameUno(fs),
        step(j)) then
        Ag(dameUno(fs), infoActualFanVivosQueDispAux(j, sinUno(fs)))
    else
        infoActualFanVivosQueDispAux(j, sinUno(fs))
    fi fi

```

**Fin TAD**

## Representación

Exorcismo ExtremoRepresentación de Juego

juego se representa con estr

```

donde j es tupla(// General
    paso: nat,
    ronda: nat,
    mapa: Mapa,

    // Disparos
    mapaDisparos: arreglo(arreglo(pasoDisparos)),
    disparosFanUltimoPaso: conj(posicion),

    // Jugadores
    infoJugadores: diccTrie(string, infoPJ),
    infoActualJugadoresVivos: conj(infoActualPJ),
    infoJugadoresVivos: conj(puntero(infoPJ)),

    // Fantasmas
    infoFantasmas: conj(infoFan),
    infoActualFantasmasVivos: conj(infoActualFan),
    infoFantasmasVivos: conj(itConj(infoFan)),
    infoFantasmaEspecial: itConj(infoActualFan) )

donde infoPJ es tupla(eventos: lista(evento),
    vivo?: bool,
    infoActual: itConj(infoActualPJ) )

```

donde **infoActualPJ** es **tupla**(*identidad*: **string**,  
                                   *posicion*: **pos**,  
                                   *direccion*: **dir** )  
 donde **infoFan** es **tupla**(*eventos*: **vector**(evento),  
                               *vivo?*: **bool**,  
                               *infoActual*: **itConj**(**infoActualFan**) )  
 donde **infoActualFan** es **tupla**(*posicion*: **pos**,  
                                   *direccion*: **dir** )

Rep : **estr**  $\rightarrow$  **bool**

Rep(*e*)  $\equiv$  **true**  $\iff$

$(\forall t : \text{tupla})(t \in e.\text{mapaDisparos} \Rightarrow_L \Pi_1(p), \Pi_2(p) \leq e.\text{paso}) \wedge$

$(\forall p : \text{pos})(p \in e.\text{disparosFanUltimoPaso} \iff \text{mapaDisp}[pos.x][pos.y].\text{pasoDispFan} = \text{paso}) \wedge$

$(\forall ip : \text{infoPJ})(ip \in \text{valores}(e.\text{infoJugadores}) \Rightarrow_L (\text{long}(ip.\text{eventos}) \leq \text{paso} \wedge (ip.\text{vivo?} \Rightarrow_L$   
 $\text{long}(ip.\text{eventos} = \text{paso}) \wedge (ip.\text{vivo?} \iff \text{no estuvo en las posiciones afectadas por los disparos de los}$   
 $\text{fantasmas en los pasos anteriores.}) \wedge \text{todas las posiciones de los eventos son posiciones libres en e.mapa}$   
 $\wedge \text{no hay saltos de posiciones entre 2 eventos consecutivos.} \wedge \text{el cambio de posición entre un evento de}$   
 $\text{if.eventos y su consecutivo debe ser el correspondiente al cambio de dirección entre ellos, considerando}$   
 $\text{también la libertad de la posición en e.mapa})) \wedge$

$(\forall if : \text{infoFan})(if \in e.\text{infoFantasmas} \Rightarrow_L (e.\text{vivo?} \iff \text{no estuvo en las pos afectadas por}$   
 $\text{los disparos de los jugadorse en los pasos anteriores.}) \wedge \text{todas las posiciones de if.eventos son posiciones}$   
 $\text{libres en e.mapa} \wedge \text{no hay saltos de posiciones entre 2 eventos consecutivos en if.eventos} \wedge \text{el cambio de}$   
 $\text{posición entre un evento de if.eventos y su consecutivo debe ser el correspondiente al cambio de dirección}$   
 $\text{entre ellos, considerando también la libertad de la posición en e.mapa}) \wedge$

$(\forall iaf : \text{infoActualFan})(iaf \in e.\text{infoActualFantasmasVivos} \Rightarrow_L \text{la posición y dirección son las}$   
 $\text{del evento correspondiente a la posición de la secuencia definida por paso mód long(eventos del fantasma)})$   
 $\wedge$

$\#e.\text{infoFantasmas} = \text{ronda} + 1 \wedge$

Los tamaños de *e.mapa* y *e.mapaDisparos* son **exactamente** iguales.  $\wedge$

$\neg(\exists p : \text{pos})(\text{Es válida y está ocupada en e.mapa y tiene disparos en e.mapaDisparos}) \wedge$

Una posición en *e.mapaDisparos* tiene un disparo de un personaje en un paso en particular  $\iff$  un  
 personaje disparó ese paso y no hubo disparos posteriores a ese paso efectuados por personajes.  $\wedge$

Una posición en *e.mapaDisparos* tiene un disparo de un fantasma en un paso en particular  $\iff$   
 un fantasma disparó ese paso y no hubo disparos posteriores a ese paso efectuados por fantasmas.  $\wedge$

Si un personaje o fantasma dispara desde una *pos* hacia una *dir*, en el mapa disparos debe haber  
 una línea recta desde esa pos hacia esa dir hasta el fin del mapa o una posición ocupada con valores  
 mayores o iguales al paso en el que se efectuó  $\wedge$

Una *infoPJ* se corresponde con su *infoActual*  $\iff$  (está vivo  $\wedge$  la posición y la dirección son  
 iguales a las de su último evento  $\wedge$  su identidad es la clave que obtiene dicha *infoPJ* en *e.infoJugadores*)  $\wedge$   
 $(\forall iapj : \text{infoActualPJ})(iapj \in e.\text{infoActualJugadoresVivos} \iff \text{su correspondiente infoPJ está vivo})$   
 $\wedge$

$(\forall ip : \text{puntero}(\text{infoPJ}))(ip \in e.\text{infoJugadoresVivos} \iff ip \rightarrow \text{vivo?}) \wedge$

$\#e.\text{infoActualJugadoresVivos} = \#e.\text{infoJugadoresVivos} \wedge$

$(\forall if : \text{infoFan})(if \in e.\text{infoFantasmas} \Rightarrow_L (\exists n : \text{nat})(\text{long}(if.\text{eventos}) = n * 2 + 5 \wedge_L (\text{los últi-}$   
 $\text{mos } n \text{ eventos de if.eventos son los primeros } n \text{ invertidos } \wedge \text{los eventos entre } n \text{ y } n+5 \text{ son pasar}))) \wedge$

Una *infoFan* se corresponde con su *infoActual*  $\iff$  (está vivo  $\wedge$  la posición y la dirección son iguales a  
 las de su último evento efectuado)  $\wedge$

$(\forall iaf : \text{infoActualFan})(iaf \in e.\text{infoActualFanVivos} \iff \text{su correspondiente infoFan está vivo}) \wedge$

$(\forall if : \text{infoFan})(if \in e.\text{infoFantasmasVivos} \iff if.\text{vivo?}) \wedge$

$\#e.\text{infoActualFantasmasVivos} = \#e.\text{infoFantasmasVivos} \wedge$

El fantasma especial siempre está vivo.

$Abs : \text{estr } e \longrightarrow \text{Juego}$   $\{\text{Rep}(e)\}$   
 $Abs(e) =_{\text{obs}} j : \text{Juego} \mid e.mapa =_{\text{obs}} habitacion(j) \wedge$   
 $claves(e.infoJugadores) =_{\text{obs}} jugadores(j) \wedge_L$   
 $(\forall pj : jugador)(pj \in jugadores(j) \Rightarrow_L acciones(pj, j) =_{\text{obs}} obtener(pj, e.infoJugadores).eventos) \wedge$   
 $fantasmas(j) =_{\text{obs}} tomarSubsec(infoFantasmas.eventos, 0, long(fantasmas(j))) \wedge$   
 $(\exists if : infoFan)(if \in e.infoFantasmas \wedge if.infoActual =_{\text{obs}} e.infoFantasmaEspecial \wedge_L$   
 $if.eventos =_{\text{obs}} fantasmaEspecial(j))$

## Algoritmos

Exorcismo Extremo Algoritmos del módulo

---

```

• iIniciar(in m : mapa, in pjs : conj(jugador), in eventosFan : vector(evento)) → res : estr
1: // Inicializo la estructura
2: res : {
3:   // Inicializo contadores
4:   paso : 0, ▷ Θ(1)
5:   ronda : 0, ▷ Θ(1)
6:
7:   // Seteo el mapa
8:   mapa : m, ▷ Θ(Tam(m)2)
9:
10:  // Inicializo el mapa de disparos con el mismo tamaño que el mapa
11:  mapaDisparos : arreglo(arreglo(tupla(nat, nat))[Tam(m)] [Tam(m)], ▷ Θ(Tam(m)2)
12:  disparosFanUltimoPaso : Vacio(), ▷ Θ(1)
13:
14:  /a/ Inicializo estructuras de jugadores y fantasmas como vacías
15:  infoActualJugadoresVivos : Vacio(), ▷ Θ(1)
16:  infoJugadoresVivos : Vacio(), ▷ Θ(1)
17:  infoJugadores : Vacia(), ▷ Θ(1)
18:  infoFantasmas : Vacio(), ▷ Θ(1)
19:  infoActualFantasmasVivos : Vacio(), ▷ Θ(1)
20:  infoFantasmasVivos : Vacia(), ▷ Θ(1)
21:  infoFantasmaEspecial : CrearIt(Vacio()) ▷ Θ(1)
22: }
23:
24: // Inicializo los jugadores
25: iIniciarJugadores(res, m, pjs) ▷ Θ(#pjs * |pjMasLargo| + locJugadores)
26:
27: // Creo el nuevo fantasma
28: iNuevoFanEspecial(res, eventosFan) ▷ Θ(long(eventosFan)2)

```

Complejidad:  $\Theta(m^2 + \#pjs * |pjMasLargo| + locJugadores + long(eventosFan)^2)$

Justificación: Copiar y generar iteradores, tuplas y conjuntos es  $\Theta(1)$ .

---



---

```

iIniciarJugadores(in/out  $j$ : estr, in  $m$ : mapa, in  $pjs$ : conj(jugador)) ▷ Función privada
1: // Suponemos la existencia de la función
2: //  $dict(jugador, tupla(pos, dir))$   $localizarJugadores(m, pjs)$ 
3:
4: // Obtengo las posiciones y direcciones de jugadores
5:  $localPJs \leftarrow localizarJugadores(m, pjs)$  ▷  $\Theta(locJugadores)$ 
6:
7: // Lleno las estructuras de jugadores
8: for ( $pj, localizacion$  :  $localPJs$ ) do ▷  $\mathcal{O}(\#pjs * (|pjMasLargo| + copy(infoMasGrande)))$ 
9:   // Creo la infoActual y la agrego a su conjunto
10:   $infoActual \leftarrow \langle identidad : pj, posicion : localizacion.pos, direccion : localizacion.dir \rangle$  ▷  $\Theta(copy(pj)) = \Theta(|pj|)$ 
11:   $itInfoActual \leftarrow AgregarRapido(j.infoActualJugadoresVivos, infoActual)$  ▷  $\Theta(copy(infoActual))$ 
12:
13:  // Creo la infoPJ con la actual
14:   $info \leftarrow iNuevaInfoPJ(localizacion, itInfoActual)$  ▷  $\Theta(1)$ 
15:  // La agrego al trie y me guardo el puntero a la info guardada
16:   $infoPtr \leftarrow \&Definir(j.infoJugadores, pj, info)$  ▷  $\Theta(|pj| + copy(info))$ 
17:
18:  // Agrego al conjunto de jugadores vivos el puntero a la info del PJ
19:   $AgregarRapido(j.infoJugadoresVivos, infoPtr)$  ▷  $\Theta(1)$ 
20: end for

```

Pre: pjs no es vacío

Post: se inicializan las estructuras de los jugadores

Complejidad:  $\mathcal{O}(\#pjs * |pjMasLargo| + locJugadores)$

Justificación: Copiar y generar iteradores, tuplas y conjuntos es  $\Theta(1)$ . Definir es  $\Theta(|pj|)$  ya que copiar la tupla de info es  $\Theta(|pj|)$  (porque hay que copiar el nombre) y definir también. Luego, definir  $\#pjs$  es  $\mathcal{O}(\#pjs * |pjMasLargo|)$ . Finalmente, la complejidad de todo el algoritmo es  $\mathcal{O}(\#pjs * |pjMasLargo| + locJugadores)$ .

---



---

```

iNuevaInfoPJ(in  $localizacion$ : tupla(pos, dir), in  $itInfoActual$ : itConj(infoActualPJ))  $\rightarrow res$ : infoPJ ▷ Función privada

```

```

1: // Armo la infoPJ
2:  $res \leftarrow \{$  ▷  $\Theta(1)$ 
3:    $eventos : iCrearEventosConLocalizacion(localizacion)$ 
4:    $vivo? : true$ 
5:    $infoActual : itInfoActual$ 
6:  $\}$ 

```

Pre: el iterador es válido

Post: se genera la info pj con el iterador y los

Complejidad:  $\Theta(1)$

Justificación: Copiar y generar iteradores, tuplas y conjuntos es  $\Theta(1)$ .

---

---

**iCrearEventosConLocalizacion**(in *localizacion* : tupla(pos, dir)) → *res* : lista(evento) ▷ Función privada

```

1: // Creo el evento
2: evento ← ⟨ ▷ Θ(1)
3:   pos : localizacion.pos,
4:   dir : localizacion.dir,
5:   disparo? : false
6: ⟩
7:
8: // Creo una lista con él
9: evts ← Vacía() ▷ Θ(1)
10: AgregarAtras(evts, evento) ▷ Θ(copy(evento))
11:
12: // La devuelvo
13: res ← evts

Pre: true
Post: se genera una lista con un solo evento que contiene la info de la localizacion
Complejidad: Θ(1)
Justificación: Copiar y generar tuplas, listas y eventos es Θ(1).
```

---



---

**iNuevoFanEspecial**(in/out *j* : estr, in *eventosFan* : vector(evento)) ▷ Función privada

```

1: // Creo la infoActual y la agrego a su conjunto
2: infoActualFan ← ⟨posicion : eventosFan[0].pos, direccion : eventosFan[0].dir⟩ ▷ Θ(1)
3: itInfoActualFan ← AgregarRapido(infoActualFan, j.infoActualFantasmasVivos) ▷ Θ(copy(infoActual))
4:
5: // Hago que el fantasma especial sea este
6: j.infoFantasmaEspecial ← itInfoActualFan ▷ Θ(1)
7:
8: // Le doy forma al vector de eventos
9: nuevosEventosFan ← Inversa(eventosFan) ▷ Θ(Longitud(eventosFan)2)
10:
11: // Creo la infoFan con la actual
12: infoFan ← ⟨eventos : nuevosEventosFan, vivo? : true, infoActual : itInfoActualFan⟩ ▷ Θ(1)
13: // La agrego al conjunto de información de fantasmas y me guardo su iterador
14: itInfoFan ← AgregarRapido(infoFan, j.infoFantasmas) ▷ Θ(copy(infoFan))
15:
16: // Agrego al conjunto de fantasmas vivos el interador a la info del Fan
17: AgregarRapido(itInfoFan, j.infoFantasmasVivos) ▷ Θ(copy(itInfoFan))

Pre: eventosFan no es vacío
Post: Se agrega un nuevo fantasma (el especial) a todas las estructuras de forma correcta
Complejidad: Θ(long(eventosFan)2)
Justificación: Copiar y generar iteradores, tuplas y conjuntos es Θ(1).
```

---



---

• **iInfoActualPjVivos**(in *j* : estr) → *res* : conj(infoActualPJ)

```

1: res ← j.infoActualJugadoresVivos

Complejidad: Θ(1)
```

---



---

• **iInfoActualFanVivos**(in *j* : estr) → *res* : conj(infoActualFan)

```

1: res ← j.infoActualFantasmasVivos

Complejidad: Θ(1)
```

---

---

• **iInfoActualFanEspecial**(in  $j$ : **estr**)  $\rightarrow res$ : infoActualFan

1:  $res \leftarrow Siguiente(j.infoFantasmaEspecial)$

Complejidad:  $\Theta(1)$

---



---

• **iInfoActualFanVivosQueDisp**(in  $j$ : **estr**)  $\rightarrow res$ : conj(infoActualFan)

1:  $fantasmasVivosQueDisp \leftarrow Vacio()$

$\triangleright \Theta(1)$

2:

3: // Recorro los fantasmas vivos y agrego a res los que estan vivos y disparando

4: **for** itInfoFan :  $j.infoFantasmasVivos$  **do**

$\triangleright \Theta(\#fv)$

5:    $infoFan \leftarrow Siguiente(itInfoFan)$

6:

7:   **if**  $iEventoActualFan(infoFan, j.paso).dispara?$

8:    **then**  $AgregarRapido(fantasmasVivosQueDisp, Siguiente(infoFan.infoActual))$   $\triangleright \Theta(copy(infoActual))$

9:    **end if**

10: **end for**

11:

12:  $res \leftarrow fantasmasVivosQueDisp$

Complejidad:  $\Theta(\#fv)$

Justificacion: Copiar una infoActual es  $\Theta(1)$ , ya que copiar tuplas y bools es  $\Theta(1)$ .

---



---

• **iVivo?**(in  $j$ : **estr**, in  $pj$ : **string**)  $\rightarrow res$ : bool

1:  $res \leftarrow Significado(j.infoJugadores, pj).vivo?$

$\triangleright \Theta(|pj|)$

Complejidad:  $\Theta(|pj|)$

---

---

• **iEjecutarAccion**(in/out  $j$ : estr, in  $a$ : accion, in  $pj$ : jugador)  $\rightarrow res$ :estr

```

1: // Incremento el paso
2:  $j.paso \leftarrow j.paso + 1$   $\triangleright \Theta(1)$ 
3:
4: // Actualizo la información del jugador con la nueva acción,
5: // y me guardo una referencia a su info modificada
6:  $infoPJ \leftarrow iActualizarPJ(pj, a)$   $\triangleright \Theta(|pj|)$ 
7:  $evtPJ \leftarrow iEventoActualPJ(infoPJ)$   $\triangleright \Theta(1)$ 
8:
9: // Reinicio los disparos de fantasmas
10:  $iReiniciarDisparosFan(j)$   $\triangleright \mathcal{O}(1)$ 
11:
12: // Modifico el mapa de disparos (solo si dispara)
13:  $iActualizarMapaDisparosConPJ(j, evtPJ)$   $\triangleright \mathcal{O}(m)$ 
14:
15: // Veo que fantasmas mueren, guardandome si murió el fantasma especial
16:  $murioFanEspecial \leftarrow iChequearMuerteFantasmas(j)$   $\triangleright \Theta(\#fv)$ 
17:
18: // Si murió el fantasma especial, cambio de ronda
19: if  $murioFanEspecial$  then
20:    $iNuevaRonda(j, infoPJ) \triangleright \mathcal{O}(m^2 + \#f + locJugadores + \#j * (|pjMasLargo| + Longitud(evMasLarg)))$ 
21: else
22:   // Sigo en la misma ronda
23:   // Actualizo las acciones de los fantasmas,
24:   // actualizando el mapa de disparos si disparan.
25:    $iActualizarFantasmas(j)$   $\triangleright \mathcal{O}(\#fv * m)$ 
26:
27:   // Veo que jugadores mueren
28:    $iChequearMuerteJugadores(j)$   $\triangleright \Theta(\#jv)$ 
29:
30:   // Agrego los 'pasar' faltantes
31:    $iAgregarPasarFaltantes(j)$   $\triangleright \Theta(\#jv)$ 
32: end if

```

Complejidad sin cambiar:  $\mathcal{O}(|pj| + m + \#fv + \#fv * m + \#jv) \in \mathcal{O}(|pj| + \#fv * m + \#jv)$

Complejidad cambiando:  $\mathcal{O}(|pj| + m^2 + \#f + locJugadores + \#j * (|pjMasLargo| + Longitud(eventoMasLargo)))$

---

---

**iActualizarPJ**(in  $pj$ : jugador, in  $a$ : accion)  $\rightarrow res$ : infoPJ ▷ Funcion privada

```

1: // Busco la información del PJ enviando infoJugadores como referencia modificable
2:  $infoPJ \leftarrow Significado(j.infoJugadores, pj)$  ▷  $\Theta(|pj|)$ 
3:
4: // Genero un evento con la acción y el evento anterior (el actual)
5:  $evtPJ \leftarrow Aplicar(a, j, iEventoActualPJ(infoPJ))$  ▷  $\Theta(1)$ 
6:
7: // Agrego el evento al jugador
8:  $AgregarAtras(infoPJ.eventos, evtPJ)$  ▷  $\Theta(copy(evtPJ))$ 
9:
10: // Obtengo su información actual
11:  $itInfoActual \leftarrow infoPJ.infoActual$  ▷  $\Theta(1)$ 
12:
13: // La actualizo
14:  $Siguiente(itInfoActual).posicion \leftarrow evt.pos$  ▷  $\Theta(1)$ 
15:  $Siguiente(itInfoActual).direccion \leftarrow evt.dir$  ▷  $\Theta(1)$ 
16:
17: // Devuelvo la info del pj
18:  $res \leftarrow infoPJ$ 

```

Pre:  $pj$  está definido en  $j.infoJugadores$   
Post: Se actualiza la info del personaje y se devuelve una referencia a ella  
Complejidad:  $\Theta(|pj|)$   
Justificación: Copiar tuplas, naturales y operaciones del iterador son  $\Theta(1)$ .

---



---

**iEventoActualPJ**(in  $info$ : infoPJ)  $\rightarrow res$ : evento ▷ Funcion privada

```

1:  $res \leftarrow Ultimo(info.eventos)$  ▷  $\Theta(1)$ 

```

Pre: true  
Post: Se devuelve el último evento de  $info.eventos$   
Complejidad:  $\Theta(1)$   
Justificación: La operación Último sobre una lista es  $\Theta(1)$

---



---

**iReiniciarDisparosFan**(in/out  $j$ : estr) ▷ Funcion privada

```

1: // Vacío la lista de disparos del ultimo paso
2: // Al asignarle vacío, se libera la memoria que ocupaba anteriormente.
3:  $j.disparosFanUltimoPaso \leftarrow Vacio()$  ▷  $\mathcal{O}(1)$ 

```

Pre: true  
Post: La lista de disparos de fantasmas es vacía  
Complejidad:  $\mathcal{O}(1)$   
Justificación: Liberar la memoria de un conjunto de tuplas de naturales es  $\mathcal{O}(1)$ .

---



---

**iActualizarMapaDisparosConPJ**(in/out  $j$ : estr, in  $evtPJ$ : evento) ▷ Funcion privada

```

1: // Si dispara, agrego las posiciones afectadas por su disparo al mapa de disparos.
2: if  $evtPJ.dispara?$ 
3: then  $iAgregarDisparo(j, evtPJ.pos, evtPJ.dir, false)$  ▷  $\mathcal{O}(m)$ 
4: end if

```

Pre: La posicion del evento es valida en el mapa del juego.  
Post: Si el evento es disparar, se llena el mapa con los disparos correctos.  
Complejidad:  $\mathcal{O}(m)$

---

---

```

iAgregarDisparo(in/out j: estr, in pos: pos, in dir: dir, in esFan: bool) ▷ Funcion privada
1: // Copio pos para no modificar el original
2: posCopy ← copy(pos) ▷  $\Theta(1)$ 
3:
4: // Parado desde posCopy en mapaDisparos, recorro hacia dir
5: // hasta que me choco con un obstaculo o la pared.
6: while Valida?(j.mapa, posCopy)  $\wedge$  Libre(j.mapa, posCopy) do ▷  $\mathcal{O}(Tam(j.mapa))$ 
7:   // Me guardo una referencia al pasoDisp correcto
8:   if esFan
9:     then pasoDisp ← mapaDisparos[pos.x][pos.y].pasoDispFan ▷  $\Theta(1)$ 
10:    else pasoDisp ← mapaDisparos[pos.x][pos.y].pasoDispPJ ▷  $\Theta(1)$ 
11:    end if
12:
13:    // Si no pasé ya por está posición con otro
14:    // (i.e si en el mapa de disparos no está ya el paso actual)
15:    if pasoDisp ≠ j.paso then
16:      // Le pongo el paso actual al paso en el que hubo un disparo
17:      pasoDisp ← j.paso ▷  $\Theta(1)$ 
18:
19:      // Si es un fantasma, agrego la posición al conjunto de disparos de fantasmas
20:      if esFan
21:        then AgregarRapido(j.disparosFanUltimoPaso, posCopy) ▷  $\Theta(copy(posCopy))$ 
22:        end if
23:    end if
24:
25:    // Avanzo la posición en esa dirección
26:    pos ← Avanzar(posCopy, dir) ▷  $\Theta(1)$ 
27: end while

```

---

Pre: La pos es valida en el mapa del juego.

Post: Si el evento es disparar, se llena el mapa con los disparos correctos.

Complejidad:  $\mathcal{O}(m)$

Justificación: Copiar naturales, indexar y tomar referencia son  $\Theta(1)$ . Luego, como hacemos operaciones que son  $\Theta(1)$  y lo hacemos a lo sumo  $Tam(j.mapa)$  veces, tenemos  $\mathcal{O}(Tam(j.mapa))$ .

---

---

```

iChequearMuerteFantasmas(in/out  $j$  : estr)  $\rightarrow res$  : bool ▷ Funcion privada
1: // Me guardo si el fantasma especial muere
2:  $muereFanEspecial \leftarrow false$ 
3:
4: // Recorro los fantasmas vivos con un iterador
5:  $itFanVivos \leftarrow CrearIt(j.infoFantasmasVivos)$ 
6:
7: while HaySiguiente(itFanVivos) do ▷  $\Theta(\#fv)$ 
8:   // Obtengo su info
9:    $infoFan \leftarrow Siguiente(Siguiente(itFanVivos))$  ▷  $\Theta(1)$ 
10:
11:   // Obtengo su evento actual
12:    $eventoActual \leftarrow iEventoActualFan(infoFan, j.paso)$  ▷  $\Theta(1)$ 
13:
14:   if  $iFanAfectadoPorDisparo(j, eventoActual.pos)$  ▷  $\Theta(1)$ 
15:   then  $muereFanEspecial \leftarrow iMuereFan(j, itFanVivos)$  ▷  $\Theta(1)$ 
16:   end if
17:
18:   // Avanzo el iterador
19:    $Avanzar(itFanVivos)$  ▷  $\Theta(1)$ 
20: end while
21:
22: // Retorno si murio el fan especial
23:  $res \leftarrow muereFanEspecial$ 

Pre: true
Post: Mata a los fantasmas que están en posiciones en las que hay disparos en este paso
Complejidad:  $\Theta(\#fv)$ 

```

---



---

```

iFanAfectadoPorDisparo(in  $j$  : estr, in  $pos$  : pos)  $\rightarrow res$  : bool ▷ Funcion privada
1: // El Fan estará afectado si en la posición en la que está hay un disparo de un PJ en el paso actual
2: // Indexo por su posición en el mapa de disparos
3:  $pasoDispPJ \leftarrow j.mapaDisparos[pos.x][pos.y].pasoDispPJ$  ▷  $\Theta(1)$ 
4:
5: // Estará afectado si el paso del disparo del PJ es igual al actual
6:  $afectado? \leftarrow (pasoDispPJ == j.paso)$  ▷  $\Theta(1)$ 
7:  $res \leftarrow afectado?$ 

Pre: pos es valida en el mapa
Post: res es true sii en pos hay un disparo de un jugador en este paso
Complejidad:  $\Theta(1)$ 

```

---

---

**iMuereFan**(in/out  $j$ : **estr**, in/out  $itFanVivos$ : **itConj**(**itConj**(**infoFan**)))  $\rightarrow res$ : **bool** ▷ Funcion privada

```

1: // Obtengo la info
2:  $infoFan \leftarrow Siguiente(Siguiente(itFanVivos))$  ▷  $\Theta(1)$ 
3:
4: // Lo seteo como muerto
5:  $infoFan.vivo? \leftarrow false$  ▷  $\Theta(1)$ 
6:
7: // Obtengo la info actual
8:  $itInfoActual \leftarrow infoFan.infoActual$  ▷  $\Theta(1)$ 
9:
10: // Veo si es el fantasma especial
11:  $eraFanEspecial \leftarrow (itInfoActual == j.infoFantasmaEspecial)$  ▷  $\Theta(1)$ 
12:
13: // Lo borro de infoActualFantasmasVivos
14:  $EliminarSiguiente(itInfoActual)$  ▷  $\Theta(1)$ 
15:
16: // Lo borro de infoFantasmasVivos
17:  $EliminarSiguiente(itFanVivos)$  ▷  $\Theta(1)$ 
18:
19: // Retorno si era el fantasma especial
20:  $res \leftarrow eraFanEspecial$ 

Pre: el iterador es valido
Post: Saca al fantasma de las estructuras que solo tienen vivos
Complejidad:  $\Theta(1)$ 

```

---



---

**iNuevaRonda**(in/out  $j$ : **estr**, in  $pjMatoFanEspecial$ : **infoPJ**) ▷ Funcion privada

```

1: // Incremento la ronda
2:  $j.ronda \leftarrow j.ronda + 1$  ▷  $\Theta(1)$ 
3:
4: // Reinicio el paso
5:  $j.paso \leftarrow 0$  ▷  $\Theta(1)$ 
6:
7: // Reinicio el mapa de disparos y los disparos de los fantasmas
8:  $iReiniciarMapaDisparos(j)$  ▷  $\Theta(m^2)$ 
9:  $iReiniciarDisparosFan(j)$  ▷  $\mathcal{O}(1)$ 
10:
11: // Reinicio los fantasmas
12:  $iReiniciarFantasmas(j)$  ▷  $\Theta(\#f)$ 
13:
14: // Agrego el nuevo fan especial luego de convertir la lista en un vector
15:  $nuevoFan \leftarrow Vectorizar(pjMatoFanEspecial.eventos)$ 
16:  $iNuevoFanEspecial(j, nuevoFan)$  ▷  $\Theta(long(nuevoFan))$ 
17:
18: // Reinicio los jugadores
19:  $iReiniciarJugadores(j)$  ▷  $\mathcal{O}(locJugadores + \#j * (|pjMasLargo| + Longitud(eventoMasLargo)))$ 

Pre: el pj es uno de los del juego
Post: se agrega un nuevo fantasma especial correspondiente al jugador que lo mató.  $\wedge$  Se reinician todas las estructuras
Complejidad:  $\mathcal{O}(m^2 + \#f + locJugadores + \#j * (|pjMasLargo| + Longitud(eventoMasLargo)))$ 

```

---



---



---

<b>iReiniciarMapaDisparos(in/out j : estr)</b>	▷ Funcion privada
1: // Recorro todas las posiciones y las pongo en 0	
2: <b>for</b> $i \leftarrow 0 \dots Tam(j.mapaDisparos)$	▷ $\Theta(m^2)$
3: <b>for</b> $j \leftarrow 0 \dots Tam(j.mapaDisparos)$	▷ $\Theta(m)$
4: $j.mapaDisparos[i][j].pasoDispFan \leftarrow 0$	▷ $\Theta(1)$
5: $j.mapaDisparos[i][j].pasoDispPJ \leftarrow 0$	▷ $\Theta(1)$
6: <b>end for</b>	
7: <b>end for</b>	
<u>Pre:</u> true	
<u>Post:</u> el mapa de disparos no tiene mas disparos	
<u>Complejidad:</u> $\Theta(m^2)$	

---



---



---

<b>iReiniciarFantasmas(in/out j : estr)</b>	▷ Funcion privada
1: // Vacío la información de los fantasmas vivos, liberando la memoria que ocupaba.	
2: $infoFantasmasVivos \leftarrow Vacio()$	▷ $\Theta(1)$
3: $infoActualFantasmasVivos \leftarrow Vacio()$	▷ $\Theta(1)$
4:	
5: // Recorro <i>infoFantasmas</i> con un iterador	
6: $itInfoFan \leftarrow CrearIt(j.infoFantasmas)$	▷ $\Theta(1)$
7: <b>while</b> HaySiguiente(itInfoFan) <b>do</b>	▷ $\Theta(\#f)$
8:     // Obtengo su info	
9: $info \leftarrow Siguiente(itInfoFan)$	▷ $\Theta(1)$
10:	
11:     // Lo seteo como vivo	
12: $info.vivo? \leftarrow true$	▷ $\Theta(1)$
13:	
14:     // Creo su <i>infoActual</i> y la agrego a <i>infoActualFantasmasVivos</i> , guardandome su iterador	
15: $infoActualFan \leftarrow \langle posicion : info.eventos[0].pos, direccion : info.eventos[0].dir \rangle$	▷ $\Theta(1)$
16: $itInfoActualFan \leftarrow AgregarRapido(j.infoActualFantasmasVivos, infoActualFan)$	▷
$\Theta(copy(infoActualFan))$	
17:	
18:     // Le seteo el iterador a la info actual	
19: $info.infoActual \leftarrow itInfoActualFan$	▷ $\Theta(1)$
20:	
21:     // Agrego un iterador a su info a <i>infoFantasmasVivos</i>	
22: $AgregarRapido(j.infoFantasmasVivos, itInfoFan)$	▷ $\Theta(copy(itInfoFan))$
23: <b>end while</b>	
24:	
<u>Pre:</u> true	
<u>Post:</u> todos los fantasmas están en las estructuras de vivos	
<u>Complejidad:</u> $\Theta(\#f)$	
<u>Justificación:</u> Copiar iteradores, tuplas, naturales, etc. es $\Theta(1)$	

---

---



---

<b>iReiniciarJugadores(in/out <math>j</math>: estr)</b>	▷ Funcion privada
1: // Vacío las estructuras de vivos	
2: $j.infoActualJugadoresVivos \leftarrow Vacio()$	▷ $\mathcal{O}(1)$
3: $j.infoJugadoresVivos \leftarrow Vacio()$	▷ $\mathcal{O}(1)$
4:	
5: // Obtengo sus localizaciones	
6: $localPJs \leftarrow localizarJugadores(j.mapa, Claves(j.infoJugadores))$	▷ $\Theta(locJugadores)$
7:	
8: // Por cada clave y valor de las localizaciones,	
9: <b>for</b> $pj, localizacion : localPJs$ <b>do</b>	▷ $\mathcal{O}(\#j * ( pjMasLargo  + Longitud(eventoMasLargo)))$
10:   // Obtengo $infoPJ$ del trie por referencia modificable (enviandole infoJugadores como modificable)	
11: $infoPJ \leftarrow Significado(j.infoJugadores, pj)$	▷ $\Theta( pj )$
12:	
13:   // Vacío los eventos, así liberando la memoria que ocupaban	
14: $infoPJ.eventos \leftarrow Vacio()$	▷ $\Theta(1)$
15:	
16:   // Creo una lista de eventos con la localización y se la seteo	
17: $infoPJ.eventos \leftarrow iCrearEventosConLocalizacion(localizacion)$	▷ $\Theta(1)$
18:	
19:   // Lo seteo como vivo	
20: $infoPJ.vivo? \leftarrow true$	▷ $\Theta(1)$
21:	
22:   // Creo una info actual	
23: $infoActualPJ \leftarrow \langle identidad : pj, posicion : localizacion.pos, direccion : localizacion.dir \rangle$	▷ $\Theta( pj )$
24:	
25:   // La agrego a $infoActualJugadoresVivos$ y me guardo itInfoActual	
26: $itInfoActual \leftarrow AgregarRapido(j.infoActualJugadoresVivos, infoActualPJ)$	▷ $\Theta(1)$
27:	
28:   // Le seteo infoActual a infoPJ con el iterador	
29: $infoPJ.infoActual \leftarrow itInfoActual$	▷ $\Theta(1)$
30:	
31:   // Agrego un puntero a la infoPJ a $infoJugadoresVivos$	
32: $AgregarRapido(j.infoJugadoresVivos, \&infoPJ)$	▷ $\Theta(1)$
33: <b>end for</b>	
<u>Pre:</u> true	
<u>Post:</u> todos los fantasmas están en las estructuras de vivos	
<u>Complejidad:</u> $\mathcal{O}(locJugadores + \#j * ( pjMasLargo  + Longitud(eventoMasLargo)))$	
<u>Justificacion:</u> Copiar tuplas, iteradores y direcciones es $\Theta(1)$	

---

---

```

iActualizarFantasmas(in/out j : estr) ▷ Función privada
1: // Recorro los fantasmas vivos
2: for (itInfoFan : j.infoFantasmasVivos) do ▷  $\Theta(\#fv)$ 
3:   // Obtengo la información del fantasma
4:   infoFan  $\leftarrow$  Siguiente(itInfoFan) ▷  $\Theta(1)$ 
5:
6:   // Actualizo su información actual, obteniendo el evento actual
7:   eventoActual  $\leftarrow$  iActualizarFan(infoFan, j.paso) ▷  $\Theta(1)$ 
8:
9:   // Si dispara, agrego su disparo a los del paso
10:  if eventoActual.dispara?
11:  then iAgregarDisparo(j, eventoActual.pos, eventoActual.dir, true) ▷  $\mathcal{O}(m)$ 
12:  end if
13: end for

```

Pre: truePost: actualiza las acciones de los fantasmas, actualizando el mapa de disparos si disparan.Complejidad:  $\mathcal{O}(\#fv * m)$ Justificación: Tomar referencia del elemento al que apunta un iterador y copiar un evento es  $\Theta(1)$ .

---

```

iActualizarFan(in/out info : infoFan, in paso : nat)  $\rightarrow$  res : evento ▷ Funcion privada
1: // Obtengo el evento actual
2: eventoActual  $\leftarrow$  iEventoActualFan(infoFan, paso) ▷  $\Theta(1)$ 
3:
4: // Obtengo el iterador a la info actual
5: itInfoActual  $\leftarrow$  info.infoActual ▷  $\Theta(1)$ 
6:
7: // La actualizo con el eventoActual
8: Siguiente(itInfoActual).posicion  $\leftarrow$  eventoActual.pos ▷  $\Theta(1)$ 
9: Siguiente(itInfoActual).direccion  $\leftarrow$  eventoActual.dir ▷  $\Theta(1)$ 
10:
11: // Devuelvo el evento actual
12: res  $\leftarrow$  eventoActual

```

Pre: truePost: actualiza la info actual del fantasmaComplejidad:  $\Theta(1)$ Justificación: Actualizar el iterador y generar el eventoActual es  $\Theta(1)$ .

---

```

iEventoActualFan(in info : infoFan, in paso : nat)  $\rightarrow$  res : evento ▷ Funcion privada
1: idx  $\leftarrow$  mod(j.paso, Longitud(info.eventos)) ▷  $\Theta(1)$ 
2: res  $\leftarrow$  info.eventos[idx] ▷  $\Theta(1)$ 

```

Pre: truePost: devuelve el evento que corresponde al paso actualComplejidad:  $\Theta(1)$ Justificación: La operaciones matemáticas y la indexación en un vector es  $\Theta(1)$ .

---

```

iChequearMuerteJugadores(in/out  $j$ : estr) ▷ Funcion privada
1: // Recorro los jugadores vivos con un iterador
2:  $itPJVivos \leftarrow CrearIt(j.infoJugadoresVivos)$  ▷  $\Theta(1)$ 
3:
4: while HaySiguiente( $itPJVivos$ ) do ▷  $\Theta(\#jv)$ 
5:   // Obtengo su evento actual
6:    $ptrInfoPJ \leftarrow Siguiente(itPJVivos)$  ▷  $\Theta(1)$ 
7:    $eventoActual \leftarrow iEventoActualPJ(*ptrInfoPJ)$  ▷  $\Theta(1)$ 
8:
9:   if  $iPJAfectadoPorDisparo?(j, eventoActual.pos)$  ▷  $\Theta(1)$ 
10:  then  $iMuerePJ(j, itPJVivos)$  ▷  $\Theta(1)$ 
11:  end if
12:
13:   // Avanzo el iterador
14:    $Avanzar(itPJVivos)$  ▷  $\Theta(1)$ 
15: end while

Pre: true
Post: quita al jugador de las estr de vivos y lo deja como muerto si está en una pos en la que hay un disparo de un fan
Complejidad:  $\Theta(\#jv)$ 
Justificación: Crear punteros, iteradores y tuplas es  $\Theta(1)$ .
16:

```

---



---

```

iPJAfectadoPorDisparo?(in  $j$ : estr, in  $pos$ : pos)  $\rightarrow res$ : bool ▷ Funcion privada
1: // El PJ estará afectado si en la posición en la que está hay un disparo de un fantasma
2: // Indexo por su posición en el mapa de disparos para obtener el paso en el que hubo un disparo del fantasma
3:  $pasoDispFan \leftarrow j.mapaDisparos[pos.x][pos.y].pasoDispFan$  ▷  $\Theta(1)$ 
4:
5: // Estará afectado si el paso del disparo del fantasma es igual al actual
6:  $afectado? \leftarrow (pasoDispFan == j.paso)$  ▷  $\Theta(1)$ 
7:  $res \leftarrow afectado?$ 

Pre: la pos es válida en el mapa del juego
Post: El PJ estará afectado si en la posición en la que está hay un disparo de un fantasma
Complejidad:  $\Theta(1)$ 
Justificación: La indexación en arreglos es  $\Theta(1)$ 

```

---



---

```

iMuerePJ(in/out  $j$ : estr, in/out  $itPJVivos$ : itConj(puntero(infoPJ))) ▷ Funcion privada
1: // Obtengo una referencia modificable a su información
2:  $infoPJ \leftarrow *Siguiente(itPJVivos)$  ▷  $\Theta(1)$ 
3:
4: // Lo seteo como muerto
5:  $infoPJ.vivo? \leftarrow false$  ▷  $\Theta(1)$ 
6:
7: // Lo borro del conjunto infoActualJugadoresVivos
8:  $EliminarSiguiente(infoPJ.infoActual)$  ▷  $\Theta(1)$ 
9:
10: // Lo borro del conjunto infoJugadoresVivos
11:  $EliminarSiguiente(itPJVivos)$  ▷  $\Theta(1)$ 

Pre: el iterador es válido
Post: modifica el booleano y quita su info de las estr de vivos
Complejidad:  $\Theta(1)$ 
Justificación: Las operaciones del iterador son  $\Theta(1)$ 

```

---

---

**iAgregarPasarFaltantes(in/out  $j$ : estr)** ▷ Funcion privada

```

1: // Recorro a todos los jugadores vivos, y le agrego un pasar a los jugadores que no se movieron
2: for ptrInfoPJ : j.infoJugadoresVivos do ▷  $\Theta(\#jv)$ 
3:   // Si su lista de acciones no es de la misma longitud que el paso actual + 1,
4:   // entonces es necesario agregarle una acción pasar
5:   if Longitud(infoPJ.eventos)  $\neq$  j.paso + 1 then
6:     // Genero el evento pasar ▷  $\Theta(1)$ 
7:     eventoPasar  $\leftarrow$  {
8:       pos : Ultimo(infoPJ.eventos).pos,
9:       dir : Ultimo(infoPJ.eventos).dir,
10:      disparo? : false
11:     }
12:     // Lo agrego al final
13:     AgregarAtras(infoPJ.eventos, eventoPasar) ▷  $\Theta(\text{copy}(\text{eventoPasar}))$ 
14:   end if
15: end for

Pre: true
Post: agrega 'pasar' a los jugadores que no hayan realizado una acción
Complejidad:  $\Theta(\#jv)$ 
Justificacion: Copiar un evento es  $\Theta(1)$ 

```

---



---

**• iPasarTiempo(in/out  $j$ : estr)**

```

1: // Incremento el paso
2:  $j.paso \leftarrow j.paso + 1$  ▷  $\Theta(1)$ 
3:
4: // Reinicio los disparos de fantasmas
5: iReiniciarDisparosFan( $j$ ) ▷  $\mathcal{O}(1)$ 
6:
7: // Actualizo las acciones de los fantasmas,
8: // actualizando el mapa de disparos si disparan.
9: iActualizarFantasmas( $j$ ) ▷  $\mathcal{O}(\#fv * m)$ 
10:
11: // Veo que jugadores mueren
12: iChequearMuerteJugadores( $j$ ) ▷  $\Theta(\#jv)$ 
13:
14: // Agrego los 'pasar' faltantes
15: iAgregarPasarFaltantes( $j$ ) ▷  $\Theta(\#jv)$ 

Complejidad:  $\mathcal{O}(\#fv * m + \#jv)$ 

```

---



---

**• iPosOcupadasPorDisparosFan(in  $j$ : estr)  $\rightarrow$  res : conj(posicion)**

```

1: res  $\leftarrow j.disparosFanUltimoPaso$ 

Complejidad:  $\Theta(1)$ 

```

---



---

**iHabitacion(in  $j$ : estr)  $\rightarrow$  res : mapa**

```

1: res  $\leftarrow j.mapa$ 

Complejidad:  $\Theta(1)$ 

```

---

---



---

**iFantasmas**(**in**  $j$  : **estr**)  $\rightarrow res$  : conj(vector(evento))

```

1:  $res \leftarrow Vacio()$   $\triangleright \Theta(1)$ 
2: for ( $infoFan$  :  $j.infoFantasmas$ ) do  $\triangleright \mathcal{O}(\#f * Longitud(eventoMasLargo))$ 
3:    $AgregarRapido(res, infoFan.eventos)$   $\triangleright \Theta(copy(infoFan.eventos)) = \Theta(Longitud(infoFan.eventos))$ 
4: end for

```

Complejidad:  $\mathcal{O}(\#f * Longitud(eventoMasLargo))$

---



---



---

**iFantasmaEspecial**(**in**  $j$  : **estr**)  $\rightarrow res$  : vector(evento)

```

1: for ( $infoFan$  :  $j.infoFantasmas$ ) do  $\triangleright \Theta(\#f)$ 
2:   if ( $infoFan.infoActual == j.infoFantasmaEspecial$ )
3:     then  $res \leftarrow infoFan.eventos$ 
4:   end if
5: end for

```

Complejidad:  $\Theta(\#f)$

---



---



---

**iJugadores**(**in**  $j$  : **estr**)  $\rightarrow res$  : conj(string)

```

1:  $res \leftarrow Claves(j.infoJugadores)$   $\triangleright \Theta(1)$ 

```

Complejidad:  $\Theta(1)$

---



---



---

**iAcciones**(**in**  $pj$  : string, **in**  $j$  : **estr**)  $\rightarrow res$  : lista(evento)

```

1:  $res \leftarrow Significado(j.infoJugadores, pj).eventos$   $\triangleright \Theta(|pj|)$ 

```

Complejidad:  $\Theta(|pj|)$

---

### 3. Módulo Mapa

El módulo Mapa provee una habitación en la que se puede ocupar y consultar por una posición en  $\Theta(1)$ .

#### Interfaz

**generos:** mapa.

**se explica con:** HABITACIÓN.

Exorcismo ExtremoOperaciones básicas del mapa **NUEVOMAPA**(**in**  $n : \text{nat}$ )  $\rightarrow res : \text{mapa}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{nuevaHab}(n)\}$

**Complejidad:**  $\Theta(n^2)$

**Descripción:** genera un mapa de tamaño  $n \times n$ .

**OCUPAR**(**in/out**  $m : \text{mapa}$ , **in**  $p : \text{pos}$ )

**Pre**  $\equiv \{m =_{\text{obs}} m_0 \wedge p \in \text{casilleros}(m) \wedge_{\text{L}} \text{libre}(m, p) \wedge \text{alcanzan}(\text{libres}(m) - p, \text{libres}(m) - p, m)\}$

**Post**  $\equiv \{m =_{\text{obs}} \text{ocupar}(c, m_0)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** ocupa una posición del mapa siempre y cuando éste no deje de ser conexo.

**TAM**(**in**  $m : \text{mapa}$ )  $\rightarrow res : \text{nat}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{tam}(m)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve el tamaño del mapa.

**LIBRE**(**in**  $m : \text{mapa}$ , **in**  $p : \text{pos}$ )  $\rightarrow res : \text{bool}$

**Pre**  $\equiv \{p \in \text{casilleros}(m)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{libre}(p, m)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve si una posición está ocupada.

**VALIDA?**(**in**  $m : \text{mapa}$ , **in**  $p : \text{pos}$ )  $\rightarrow res : \text{bool}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} 0 \leq \Pi_1(pos) < \text{tam}(m) \wedge 0 \leq \Pi_2(pos) < \text{tam}(m)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve si una posición es válida en el mapa.

**COPIAR**(**in**  $m : \text{mapa}$ )  $\rightarrow res : \text{mapa}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} m\}$

**Complejidad:**  $\Theta(m^2)$

**Descripción:** devuelve si una posición está ocupada.

**Aliasing:** se devuelve una copia del elemento ingresado por referencia.

#### Representación

Exorcismo ExtremoRepresentación del mapa

El objetivo de este módulo es implementar una matriz de tamaño  $n$  con vectores de booleanos que indican si una posición está ocupada. La estructura de representación, su invariante de representación y su función de abstracción son las siguientes.

**mapa se representa con map**

donde **map** es **tupla**(*tamano*: **nat**, *casilleros*: **vec**(**vec**(**bool**)),)

**Rep** : **mapa**  $\rightarrow$  **bool**

$\text{Rep}(\text{map}) \equiv \text{true} \iff$  La longitud de  $\text{map.casilleros}$  es igual a  $\text{tamano} \wedge$   
 La longitud del vector  $\text{m.casilleros}$  es igual a la de todo otro vector dentro de el)  $\wedge$   
 Toda posición libre debe ser alcanzable por todo el resto de las posiciones libres a través de un camino  
 de posiciones libres (conexo).

$\text{Abs} : \text{mapa } \text{map} \longrightarrow \text{hab} \quad \{\text{Rep}(\text{map})\}$   
 $\text{Abs}(\text{map}) =_{\text{obs}} \text{h} : \text{hab} \mid m.\text{tamano} =_{\text{obs}} \text{tam}(\text{h}) \wedge_{\text{L}}$   
 $(\forall t: \text{tupla}(\text{nat}, \text{nat})) (0 \leq \Pi_1(t), \Pi_2(t) < \text{map.tamano} - 1 \Rightarrow_{\text{L}}$   
 $\text{libre}(\text{h}, t) =_{\text{obs}} \text{map.casilleros}[\Pi_1(t)][\Pi_2(t)])$

## Algoritmos

Exorcismo Extremo Algoritmos del módulo

---

**iTam**(in  $m : \text{map}$ )  $\rightarrow res : \text{nat}$   
 1:  $res \leftarrow m.\text{tamano}$   $\triangleright \Theta(1)$   
Complejidad:  $\Theta(1)$

---



---

**iOcupar**(in/out  $m : \text{map}$ , in  $p : \text{pos}$ )  
 1:  $m[\Pi_1(p)][\Pi_2(p)] \leftarrow \text{true}$   $\triangleright \Theta(1)$   
Complejidad:  $\Theta(1)$   
Justificación: El acceso a una posición de un vector y su modificación es  $\Theta(1)$

---



---

**iLibre**(in  $m : \text{map}$ , in  $p : \text{pos}$ )  $\rightarrow res : \text{bool}$   
 1:  $res \leftarrow \neg m[\Pi_1(p)][\Pi_2(p)]$   $\triangleright \Theta(1)$   
Complejidad:  $\Theta(1)$   
Justificación: El acceso a una posición de un vector es  $\Theta(1)$

---



---

**iNuevoMapa**(in  $n : \text{nat}$ )  $\rightarrow res : \text{map}$   
 1: // Inicializo el tamaño, el vector y el mapa.  
 2:  $res \leftarrow \langle \text{tamano} : n, \text{casilleros} : \text{Vacía}() \rangle$   $\triangleright \Theta(1)$   
 3:  
 4: // Genero un vector de booleanos en falso con n posiciones.  
 5:  $i \leftarrow 0$   $\triangleright \Theta(1)$   
 6: **while**  $i < n$  **do**  $\triangleright \mathcal{O}(n^2)$   
 7:    $v.\text{AgregarAtras}(\text{false})$   $\triangleright \mathcal{O}(n)$   
 8:    $i \leftarrow i + 1$   
 9: **end while**  
 10:  
 11: // Genero la matriz de n x n posiciones haciendo n copias del vector de booleanos antes creado.  
 12:  $i \leftarrow 0$   
 13: **while**  $i < n$  **do**  $\triangleright \mathcal{O}(n^2)$   
 14:    $res.\text{AgregarAtras}(v.\text{Copiar}())$   $\triangleright \mathcal{O}(n)$   
 15:    $i \leftarrow i + 1$   $\triangleright \Theta(1)$   
 16: **end while**

Complejidad:  $\mathcal{O}(n^2)$

Justificación: Copiar un vector de n booleanos es  $\mathcal{O}(n * \text{copy}(\text{bool}))$  y copiar un bool es  $\Theta(1)$ . Luego, agregar n veces la copia del vector es  $\mathcal{O}(n^2)$ , puesto que AgregarAtras es  $\mathcal{O}(n)$  y copiarlo es  $\mathcal{O}(n)$  por lo antes visto. Luego la complejidad de la operación de la línea 10 es  $\mathcal{O}(n)$  y, por lo tanto, todo el while es  $\mathcal{O}(n^2)$ .

---



---



---

**iValida?**(**in**  $m : \text{map}$  **in**  $p : \text{pos}$ )  $\rightarrow res : \text{nat}$

1:  $res \leftarrow 0 \leq pos.x, pos.y < m.tamano$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---



---

**iCopiar**(**in**  $m : \text{map}$ )  $\rightarrow res : \text{map}$

1:  $res \leftarrow \langle tamaño : m.tamano, casilleros : Copiar(m.casilleros) \rangle$

$\triangleright \Theta(m.tamano^2)$

Complejidad:  $\Theta(1)$

Justificación: Copiar un vector es  $\Theta\left(\sum_{i=1}^{\ell} copy(v[i])\right)$ , donde  $\ell = \text{long}(v)$ . Luego, copiar un vector de vectores de booleanos de tamaño  $n$  es  $\Theta(m.tamano^2)$ .

---

## 4. Módulo Dirección

El módulo Dirección provee una dirección y una función que permite invertir las mismas.

### Interfaz

**generos:** dir.

**se explica con:** DIRECCIÓN.

Exorcismo ExtremoOperaciones básicas de Dirección

ARRIBA()  $\rightarrow res : dir$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} \uparrow\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera la dirección arriba.

ABAJO()  $\rightarrow res : dir$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} \downarrow\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera la dirección abajo.

IZQUIERDA()  $\rightarrow res : dir$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} \leftarrow\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera la dirección izquierda.

DERECHA()  $\rightarrow res : dir$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} \rightarrow\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera la dirección derecha.

INVERTIR(in/out  $d : dir$ )

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} invertir(d)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** invierte la dirección.

### Representación

El objetivo de este módulo es implementar una dirección utilizando strings. La estructura de representación, su invariante de representación y su función de abstracción son las siguientes.

Exorcismo ExtremoRepresentación de Dirección

**dir se representa con string**

$Rep : dir \rightarrow bool$

$Rep(d) \equiv true \iff$

$d =_{obs} "arriba" \vee$

$d =_{obs} "abajo" \vee$

$d =_{obs} "izquierda" \vee$

$d =_{obs} "derecha"$

$Abs : dir \rightarrow dir$

$\{Rep(d)\}$

$Abs(d) =_{obs} d_{tad} : dir \mid (d =_{obs} "arriba" \wedge d_{tad} =_{obs} \uparrow) \vee$   
 $(d =_{obs} "abajo" \wedge d_{tad} =_{obs} \downarrow) \vee$   
 $(d =_{obs} "izquierda" \wedge d_{tad} =_{obs} \leftarrow) \vee$   
 $(d =_{obs} "derecha" \wedge d_{tad} =_{obs} \rightarrow)$

## Algoritmos Exorcismo Extremo Algoritmos del módulo

---

---

**iArriba()**  $\rightarrow res : \text{dir}$

1:  $res \leftarrow \text{"arriba"}$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

---

**iAbajo()**  $\rightarrow res : \text{dir}$

1:  $res \leftarrow \text{"abajo"}$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

---

**iIzquierda()**  $\rightarrow res : \text{dir}$

1:  $res \leftarrow \text{"izquierda"}$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

---

**iDerecha()**  $\rightarrow res : \text{dir}$

1:  $res \leftarrow \text{"derecha"}$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

---

**iInvertir(in/out  $d : \text{dir}$ )**

1:  $\text{switch}(d)$

2:  $\text{case "arriba" :}$

3:      $d \leftarrow \text{"abajo"}$

4:  $\text{case "abajo" :}$

5:      $d \leftarrow \text{"arriba"}$

6:  $\text{case "izquierda" :}$

7:      $d \leftarrow \text{"derecha"}$

8:  $\text{case "derecha" :}$

9:      $d \leftarrow \text{"izquierda"}$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---

## 5. Módulo Acción

El módulo Acción provee una acción y una funciones que permiten operar con acciones y eventos.

### Interfaz

**generos:** accion.

**se explica con:** ACCIÓN.

Exorcismo ExtremoOperaciones básicas de Acción

**MOVER**(in  $d$ : dir)  $\rightarrow res$  : accion

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} mover(d)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera una acción de mover en la dirección especificada.

**PASAR**()  $\rightarrow res$  : accion

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} pasar\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera la acción de pasar.

**DISPARAR**()  $\rightarrow res$  : accion

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} disparar\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera la acción de disparar.

**APLICAR**(in  $a$ : acción, in  $j$ : juego, in  $e$ : evento)  $\rightarrow res$  : evento

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} aplicar(a, j, e)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera el evento a partir de la acción a realizar.

**INVERTIR**(in  $e$ : evento)  $\rightarrow res$  : evento

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} invertir(e)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** invierte un evento.

**INVERSA**(in/out  $es$ : vector(evento))  $\rightarrow res$  : vector(evento)

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} inversaReal(es)\}$

**Complejidad:**  $\Theta(long(es)^2)$

**Descripción:** genera una secuencia que contiene a la inicial, le suma 5 pasos de espera y le agrega la secuencia original invertida.

**AVANZAR**(in  $p$ : pos, in  $d$ : dir)  $\rightarrow res$  : pos

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{d =_{obs} \uparrow \wedge res =_{obs} \langle \Pi_1(pos) + 1, \Pi_2(pos) \rangle \vee$

$d =_{obs} \downarrow \wedge res =_{obs} \langle \Pi_1(pos) - 1, \Pi_2(pos) \rangle \vee$

$d =_{obs} \leftarrow \wedge res =_{obs} \langle \Pi_1(pos), \Pi_2(pos) - 1 \rangle \vee$

$d =_{obs} \rightarrow \wedge res =_{obs} \langle \Pi_1(pos) + 1, \Pi_2(pos) + 1 \rangle\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve la posición correspondiente a moverse en la dirección indicada.

Exorcismo ExtremoEspecificación de las operaciones auxiliares utilizadas en la interfaz

**TAD** Acción Extendida( $\alpha$ )

**extiende** ACCIÓN

**otras operaciones**

$\text{inversaReal} : \text{secu}(\text{evento}) \rightarrow \text{secu}(\text{evento})$   
 $\text{pasar} : \text{evento} \rightarrow \text{evento}$

#### axiomas

$\text{inversasReal}(es) \equiv es \ \& \ (\text{pasar}(\text{ult}(es)) \bullet \text{pasar}(\text{ult}(es)) \bullet \text{pasar}(\text{ult}(es)) \bullet \text{pasar}(\text{ult}(es)) \bullet \text{pasar}(\text{ult}(es)) \bullet \dots)$   
 $\quad \quad \quad \& \ \text{inversa}(es)$   
 $\text{pasar}(\text{evt}) \equiv \langle \Pi_1(\text{evt}), \Pi_2(\text{evt}), \text{false} \rangle$

Fin TAD

## Representación

Exorcismo ExtremoRepresentación de Acción

El objetivo de este módulo es implementar una acción utilizando una tupla de string y dirección. La estructura de representación, su invariante de representación y su función de abstracción son las siguientes.

acción se representa con a

donde a es  $\text{tupla}(\text{acción: string}, \text{dir: dir})$

$\text{Rep} : \text{acción} \rightarrow \text{bool}$

$\text{Rep}(a) \equiv \text{true} \iff$

$\text{a.acción} =_{\text{obs}} \text{"disparar"} \vee$

$\text{a.acción} =_{\text{obs}} \text{"pasar"} \vee$

$\text{a.acción} =_{\text{obs}} \text{"mover"}$

$\text{Abs} : \text{acción } a \rightarrow \text{acción}$

$\{\text{Rep}(a)\}$

$\text{Abs}(a) =_{\text{obs}} \text{a}_{\text{tad}} : \text{acción} \mid (\text{a.acción} =_{\text{obs}} \text{"disparar"} \wedge \text{esDisparar}(\text{a}_{\text{tad}})) \vee$   
 $(\text{a.acción} =_{\text{obs}} \text{"pasar"} \wedge \text{esPasar}(\text{a}_{\text{tad}})) \vee$   
 $((\text{a.acción} =_{\text{obs}} \text{"mover"} \wedge \text{esMover}(\text{a}_{\text{tad}})) \wedge_{\text{L}} \text{a.dir} =_{\text{obs}} \text{direccion}(\text{a}_{\text{tad}}))$

## Algoritmos

Para las acciones que no tienen dirección, les definimos la dirección *Arriba()*.

Esto no importa ya que la dirección es ignorada en general para esas acciones.

Exorcismo ExtremoAlgoritmos del módulo

---

**iPasar()**  $\rightarrow res : \text{acción}$

1:  $res \leftarrow \langle \text{acción} : \text{"pasar"}, \text{dir} : \text{Arriba}() \rangle$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**iDisparar()**  $\rightarrow res : \text{acción}$

1:  $res \leftarrow \langle \text{acción} : \text{"disparar"}, \text{dir} : \text{Arriba}() \rangle$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**iMover(in d: dir)**  $\rightarrow res : \text{acción}$

1:  $res \leftarrow \langle \text{acción} : \text{"mover"}, \text{dir} : d \rangle$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---



---

**iInvertir(in e: evento)**  $\rightarrow res : \text{evento}$

1:  $res \leftarrow \langle \text{pos} : e.\text{pos}, \text{dir} : \text{Invertir}(e.\text{dir}), \text{disparo?} : e.\text{disparo?} \rangle$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

---

---

**iAvanzar**(in  $p$ : pos, in  $d$ : dir)  $\rightarrow res$ : pos

```

1: switch( $d$ )  $\triangleright \Theta(1)$ 
2: case Arriba():
3:    $res \leftarrow \langle pos.x + 1, pos.y \rangle$ 
4: case Abajo():
5:    $res \leftarrow \langle pos.x - 1, pos.y \rangle$ 
6: case Izquierda():
7:    $res \leftarrow \langle pos.x, pos.y - 1 \rangle$ 
8: case Derecha():
9:    $res \leftarrow \langle pos.x, pos.y + 1 \rangle$ 

```

Complejidad:  $\Theta(1)$

---



---

**iInversa**(in/out  $es$ : vector(evento))

```

1: // El resultado deseado es el siguiente
2:
3: // Copio el vector de entrada
4:  $esCopy \leftarrow copy(es)$   $\triangleright \Theta(long(es))$ 
5:
6: // Me guardo la longitud original
7:  $longOriginal \leftarrow Longitud(es)$ 
8:
9: // Creo un evento que sea pasar y lo agrego 5 veces
10:  $eventoPasar \leftarrow \langle pos : Ultimo(esCopy).pos, dir : Ultimo(esCopy).dir, disparo? : false \rangle$   $\triangleright \Theta(1)$ 
11: for ( $i = 0, \dots, 4$ ) do  $\triangleright \mathcal{O}(long(es) * 5)$ 
12:    $AgregarAtras(esCopy, eventoPasar)$   $\triangleright \mathcal{O}(long(es))$ 
13: end for
14:
15: // Recorro los eventos de la secuencia original de atrás para adelante,
16: // invirtiendolos y agregándolos al final
17: for ( $i = longOriginal - 1, \dots, 0$ ) do  $\triangleright \mathcal{O}(long(es)^2)$ 
18:    $AgregarAtras(esCopy, invertir(esCopy[i]))$   $\triangleright \mathcal{O}(long(es))$ 
19: end for
20:
21: // Devuelvo los eventos
22:  $res \leftarrow esCopy$ 

```

Complejidad:  $\mathcal{O}(long(es)^2)$

Justificación: Crear una tupla y acceder al vector es  $\Theta(1)$ .  $\mathcal{O}(long(es) * 5) + \mathcal{O}(long(es)^2) = \mathcal{O}(long(es)^2)$ .

---

---

**iAplicar**(in  $a$ : acción, in  $j$ : juego, in  $e$ : evento)  $\rightarrow res$ : evento

```

1: if ( $a.accion = disparar$ )  $\triangleright \Theta(1)$ 
2:   then  $res \leftarrow \langle pos : e.pos, dir : e.dir, disparo? : true \rangle$   $\triangleright \Theta(1)$ 
3: end if
4:
5: if ( $a.accion = pasar$ )  $\triangleright \Theta(1)$ 
6:   then  $res \leftarrow \langle pos : e.pos, dir : e.dir, disparo? : false \rangle$   $\triangleright \Theta(1)$ 
7: end if
8:
9: if ( $a.accion = mover$ ) then  $\triangleright \Theta(1)$ 
10:    $prox \leftarrow Avanzar(e.pos, a.dir)$ 
11:   if ( $Valida?(j.mapa, prox) \wedge_L Libre(j.mapa, prox)$ )  $\triangleright \Theta(1)$ 
12:     then  $res \leftarrow \langle pos : prox, dir : a.dir, disparo? : false \rangle$   $\triangleright \Theta(1)$ 
13:     else  $res \leftarrow \langle pos : e.pos, dir : a.dir, disparo? : false \rangle$ 
14:   end if
15: end if

```

Complejidad:  $\Theta(1)$ 
Justificación: Crear una tupla, comparar sus elementos y las operaciones del mapa son  $\Theta(1)$ .

---

## 6. Diccionario Trie ( $\alpha$ )

El módulo Diccionario Trie provee un diccionario básico montado sobre un trie. Solo se definen e implementan las operaciones que serán utilizadas.

### Interfaz

**parámetros formales**

**géneros**  $\alpha$   
**función**  $\text{COPIAR}(\text{in } s : \alpha) \rightarrow res : \alpha$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} s\}$   
**Complejidad:**  $\Theta(\text{copy}(s))$   
**Descripción:** función de copia de  $\alpha$

**se explica con:**  $\text{DICCIONARIO}(\text{string}, \alpha)$ .

**géneros:**  $\text{diccTrie}(\text{string}, \alpha)$ .

Exorcismo ExtremoOperaciones básicas de diccionario

$\text{VACÍO}() \rightarrow res : \text{diccTrie}(\text{string}, \alpha)$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{vacío}\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** genera un diccionario vacío.

$\text{DEFINIR}(\text{in/out } d : \text{diccTrie}(\text{string}, \alpha), \text{in } k : \text{string}, \text{in } s : \alpha) \rightarrow res : \alpha$   
**Pre**  $\equiv \{d =_{\text{obs}} d_0\}$   
**Post**  $\equiv \{d =_{\text{obs}} \text{definir}(d, k, s)\}$   
**Complejidad:**  $\Theta(|k| + \text{copy}(s))$   
**Descripción:** define la clave  $k$  con el significado  $s$  en el diccionario.  
**Aliasing:** los elemnetos  $k$  y  $s$  se definen por copia.

$\text{DEFINIDO?}(\text{in } d : \text{diccTrie}(\text{string}, \alpha), \text{in } k : \text{string}) \rightarrow res : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{def?}(d, k)\}$   
**Complejidad:**  $\mathcal{O}(|k|)$   
**Descripción:** devuelve **true** si  $k$  está definido en el diccionario.

$\text{SIGNIFICADO}(\text{in } d : \text{diccTrie}(\text{string}, \alpha), \text{in } k : \text{string}) \rightarrow res : \alpha$   
**Pre**  $\equiv \{\text{def?}(d, k)\}$   
**Post**  $\equiv \{\text{alias}(res =_{\text{obs}} \text{obtener}(d, k))\}$   
**Complejidad:**  $\Theta(|k|)$   
**Descripción:** devuelve el significado de la clave  $k$  en  $d$ .  
**Aliasing:**  $res$  es modificable si y sólo si  $d$  es modificable.

$\text{CLAVES}(\text{in } d : \text{diccTrie}(\text{string}, \alpha)) \rightarrow res : \text{conj}(\text{string})$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{claves}(d)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** devuelve las claves definidas en el diccionario

### Representación

Exorcismo ExtremoRepresentación del diccionario

$\text{diccTrie}(\text{string}, \alpha)$  se representa con  $\text{dic}$

donde  $\text{dic}$  es  $\text{tupla}(\text{raiz: puntero(nodo)},$   
 $\text{claves: conj(string)})$



donde **nodo** es  $\text{tupla}(\text{significado: puntero}(\alpha),$   
 $\text{siguientes: arreglo}(\text{puntero}(\text{nodo})) [256] )$

$\text{Rep} : \text{dic} \rightarrow \text{bool}$

$\text{Rep}(d) \equiv \text{true} \iff$

(Los nodos del diccionario (excepto la raiz) tienen un unico padre. Es decir, no hay dos Nodos en la estructura que tengan punteros iguales en los siguientes del Nodo.  $\wedge$

La raiz no tiene padre. Es decir, no hay un camino de hijos por el cual se llegue a dicho Nodo.  $\wedge$

Todas las hojas tienen un significado distinto de NULL.  $\wedge$

Un  $s$  string pertenece a  $d.\text{claves}$   $\iff \text{estáDefinido}(s, d.\text{claves})$ )

// La primer condicion implica que no hay ciclos ni Nodos con hijos de menor nivel

$\text{Abs} : \text{dic } e \rightarrow \text{dicc}(\text{string}, \alpha)$

$\{\text{Rep}(e)\}$

$\text{Abs}(e) =_{\text{obs}} d : \text{dicc}(\text{string}, \alpha) \mid$

$(\forall :: s \text{ string})(\text{def?}(s, d) =_{\text{obs}} \text{estáDefinido}(e.\text{raiz}, s)) \wedge$

$(\forall :: s \text{ string})(\text{def?}(s, d) \Rightarrow_L \text{obtener}(s, d) =_{\text{obs}} \text{significado}(e.\text{raiz}, s)) \wedge$

$\text{claves}(d) =_{\text{obs}} e.\text{claves}$

$\text{estáDefinido}(r, s) \equiv \text{if } \text{vacía?}(s)$

**then**  $r \rightarrow \text{significado} \neq \text{NULL}$

**else**  $r \rightarrow \text{siguientes}[\text{int}(\text{prim}(s))] \neq \text{NULL} \wedge_L \text{estáDefinido}(r.\text{siguientes}[\text{int}(\text{prim}(s))], \text{fin}(s))$  **fi**

$\text{significado}(r, s) \equiv \text{if } \text{vacía?}(s)$

**then**  $r \rightarrow \text{significado}$

**else**  $\text{significado}(r.\text{siguientes}[\text{int}(\text{prim}(s))], \text{fin}(s))$  **fi**

## Algoritmos

---

**iVacía()**  $\rightarrow res : \&\text{dic}$

1: // Le asigna un nuevo nodo a la raiz

2:  $res \leftarrow \langle \text{raiz} : \text{nuevoNodo}() \rangle$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

Justificación: La complejidad de crear un nuevo nodo es  $\Theta(1)$

---



---

**iSignificado(in/out  $d : \text{dic}$ , in  $k : \text{string}$ )**  $\rightarrow res : \&\alpha$

1:  $\text{Nodo actual} \leftarrow d.\text{raiz}$

$\triangleright \Theta(1)$

2: **for** ( $\text{char } c : k$ ) **do**

$\triangleright \mathcal{O}(|k|)$

3:  $\text{actual} \leftarrow (\text{actual} \rightarrow \text{siguientes}[\text{toInt}(c)])$

$\triangleright \Theta(1)$

4: **end for**

5:  $res \leftarrow *(\text{actual} \rightarrow \text{significado})$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(|k|)$

Justificación: Los accesos y las asignaciones de punteros son  $\Theta(1)$ . Como el ciclo se ejecuta  $|k|$  veces, se ejecutarán dichas asignaciones  $|k|$  veces. Luego la complejidad será  $\Theta(|k|)$ .

---



---

**iClaves(in  $d : \text{dic}$ )**  $\rightarrow res : \text{conj}(\text{string})$

1:  $res \leftarrow e.\text{claves}$

$\triangleright \Theta(1)$

Complejidad:  $\Theta(1)$

Justificación: Devolver por referencia un conjunto es  $\Theta(1)$ .

---

**iDefinido?**(in/out  $d$ : dic, in  $k$ : string)  $\rightarrow res$ : bool

```

1: Nodo  $actual \leftarrow d.raiz$   $\triangleright \Theta(1)$ 
2: for ( $char\ c : k$ ) do  $\triangleright \mathcal{O}(|k|)$ 
3:   if ( $actual \rightarrow siguientes[toInt(c)] \neq NULL$ )  $\triangleright \Theta(1)$ 
4:     then  $actual \leftarrow (actual \rightarrow siguientes[toInt(c)])$   $\triangleright \Theta(1)$ 
5:     else  $res \leftarrow false$   $\triangleright \Theta(1)$ 
6:   end if
7: end for
8:  $res \leftarrow ((actual \rightarrow significado) \neq NULL)$   $\triangleright \Theta(1)$ 

```

Complejidad:  $\mathcal{O}(|k|)$

Justificación: Los accesos y las asignaciones de punteros son  $\Theta(1)$ . Como el ciclo se ejecuta a lo sumo  $|k|$  veces, se ejecutarán dichas asignaciones  $|k|$  veces como máximo. Luego la complejidad será  $\mathcal{O}(|k|)$ .

**iDefinir**(in/out  $d$ : dic, in  $k$ : string, in  $s$ :  $\alpha$ )  $\rightarrow res$ :  $\&\alpha$

```

1: Nodo  $actual \leftarrow d.raiz$ 
2: for ( $char\ c : k$ ) do  $\triangleright \Theta(|k|)$ 
3:   // Si no tengo siguiente, lo creo
4:   if ( $actual \rightarrow siguientes[toInt(c)] == NULL$ ) then  $\triangleright \Theta(1)$ 
5:      $actual \rightarrow siguientes[toInt(c)] = nuevoNodo()$   $\triangleright \Theta(1)$ 
6:   end if
7:    $actual \leftarrow (actual \rightarrow siguientes[toInt(c)])$   $\triangleright \Theta(1)$ 
8: end for
9:
10: // Estoy parado en el nodo que va a tener el puntero al significado.
11: // Reservo un lugar en memoria y hago una copia del provisto en dicho lugar.
12:  $sig \leftarrow s$   $\triangleright \Theta(copy(s))$ 
13:
14: // Si el significado no está definido, agrego la nueva clave al conjunto de claves
15: if ( $actual \rightarrow significado == NULL$ ) then  $\triangleright \Theta(1)$ 
16:   // Como precondition se que no existe así que la agrego rápido
17:   AgregarRapido(e.claves, k)  $\triangleright \Theta(copy(k))$ 
18: end if
19:
20: // Asigno al significado del nodo el puntero creado con s y libero la memoria que contenía al valor anterior.
21:  $(actual \rightarrow significado) \leftarrow \&sig$   $\triangleright \Theta(1)$ 
22:
23: // Devuelvo por referencia el significado.
24:  $res \leftarrow sig$ 

```

Complejidad:  $\Theta(|k| + copy(s))$

Justificación: Siempre se recorre toda la palabra para definirla, entonces el *for* siempre tiene  $|k|$  ciclos. La dereferenciación y comparación de punteros, e indexación en arreglos estáticos son  $\Theta(1)$ . Además  $\Theta(|k| + copy(s) + copy(k)) \in \Theta(|k| + copy(s))$ .

**inuevoNodo**()  $\rightarrow res$ : puntero(nodo)

$\triangleright$  Función privada que crea un nuevo nodo

```

1: // Reserva la memoria para un nuevo nodo con significado null y siguientes vacios
2:  $res \leftarrow \&\langle significado : NULL, siguientes : arreglo\_estatico[256] \text{ de puntero}(Nodo) \rangle$   $\triangleright \Theta(1)$ 

```

Complejidad:  $\Theta(1)$

Justificación: El tiempo de creación de un array de 255 posiciones es  $\mathcal{O}(255) \in \mathcal{O}(1)$