DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II: Diseño

Exorcismo Extremo

Algoritmos y Estructuras de Datos II

Primer Cuatrimestre de 2019

Integrante	LU	Correo electrónico
Liza, Franco	258/16	franco.s.liza@gmail.com
Sosa, Patricio	218/16	patriciososa91@gmail.com
Martin, Moran	650/17	martinmoran1994@gmail.com
Crego, Franco	345/16	francocrego@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

1. Modulo Habitacion

Interfaz

parámetros formales
géneros *habitacion*

se explica con: TAD HABITACION

géneros: `habitacion`.

Operaciones básicas:

NUEVAHABITACION(**in** n : `nat`) $\rightarrow$ res : `habitacion`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ nuevaHabitacion(c)}
Complejidad: O(n^2)
Descripción: Genera una habitacion vacia

OCUPAR(**in** c : `posicion`, **in/out** h : `habitacion`)
Pre $\equiv$ {c $\in$ casilleros(h) $\wedge$ alcanzan(libres(h)-c, libres(h)-c) $\wedge$ $h =_{\text{obs}} h_0$}
Post $\equiv$ {$\neg$ libre(c,h)}
Complejidad: O(1)

LIBRE?(**in** c : `posicion`, **in** h : `habitacion`) $\rightarrow$ res : `bool`
Pre $\equiv$ {c $\in$ casilleros(h)}
Post $\equiv$ {res $=_{\text{obs}}$ libre?(c,h)}
Complejidad: O(1)

HAYADY$\uparrow$(**in** p : `posicion`, **in** h : `habitacion`) $\rightarrow$ res : `bool`
Pre $\equiv$ {c $\in$ casilleros(h)}
Post $\equiv$ {res $=_{\text{obs}}$ hayAdy$\uparrow$(p,h)}
Complejidad: O(1)

HAYADY$\downarrow$(**in** p : `posicion`, **in** h : `habitacion`) $\rightarrow$ res : `bool`
Pre $\equiv$ {c $\in$ casilleros(h)}
Post $\equiv$ {res $=_{\text{obs}}$ hayAdy$\downarrow$(p,h)}
Complejidad: O(1)

HAYADY$\leftarrow$(**in** p : `posicion`, **in** h : `habitacion`) $\rightarrow$ res : `bool`
Pre $\equiv$ {c $\in$ casilleros(h)}
Post $\equiv$ {res $=_{\text{obs}}$ hayAdy$\leftarrow$(p,h)}
Complejidad: O(1)

HAYADY$\rightarrow$(**in** p : `posicion`, **in** h : `habitacion`) $\rightarrow$ res : `bool`
Pre $\equiv$ {c $\in$ casilleros(h)}
Post $\equiv$ {res $=_{\text{obs}}$ hayAdy$\rightarrow$(p,h)}
Complejidad: O(1)

ALCANCEDISPARO(**in** p : `posicion`,**in** d : `direccion`,**in** h : `habitacion`) $\rightarrow$ res : `conj(posicion)`
Pre $\equiv$ {c $\in$ casilleros(h)}
Post $\equiv$ {res $=_{\text{obs}}$ AlcanceDisparo(c,d,h)}
Complejidad: $O(m)$
Descripción: Devuelve el conjunto con todas las posiciones válidas a las que afecta el disparo.

Representación

`Habitacion` se representa con `estr`

donde es `tupla`(*casilleros*: `arreglo_dimensionable[tamaño] de arreglo_dimensionable[tamaño] de bool`
, *tamaño*: `nat`
)

Rep : Habitacion $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$

1. Casileros libres no puede ser mayor que la cantidad de celdas en e.hab
2. El tamaño del arreglo es de tamaño n y cada arreglo dentro del arreglo también es de tamaño n, es decir el amcho y el alto de la matriz es de n x n

Función de abstracción:
Abs : estr e $\longrightarrow$ Habitacion {Rep(e)}
Abs(e) $\equiv$ c / tam(c) = e.tamaño $\wedge$ e.casilleros = casilleros(c)

Algoritmos

---

iNuevaHabitacion(**in** n : `nat`) $\rightarrow$ res : estr

```
1: res.tamaño ← n
2: res.hab ← vacio(n,n)                    ▷ Crea una matriz de booleanos. lo hacemos asi por facilidad
3: i ← 0
4: j ← 0
5: while i < n do                                                          ▷ Inicializo la matriz
6:     while j < n do
7:         res.hab[i][j] ← true            ▷ La cargo de True para decir que estan "libres.ª cada una de las celdas
8:
9:         j++
10:    end while
11:    i++
12: end while
```

Complejidad: O(n^2)
Justificación: Crea una matriz de n x n (de booleanos), la cual es la habitacion e inicializa toda la estructura de este módulo.

---

iOcupar(**in** c : `posicion` **in/out** e : `estr`)

```
1: i ← π1(c)
2: j ← π2(c)
3: e.hab[i][j] ← false
```

Complejidad: $O(1)$

---

iLibre?(**in** c : `posicion`, **in** e : `estr`) $\rightarrow$ res : bool

```
1: i ← π1(c)
2: j ← π2(c)
3: res ← e.hab[i][j]                                          ▷ Indexo en la matriz habitacion
4: return res;                                  ▷ Si devuelve True es porque esta libre, caso contrario no
```

Complejidad: $O(1)$

---

iHayAdy↑(in c: `posicion`, **in** e: `estr`) → res : bool

```
1: i ← π1(c)
2: j ← π2(c)
3: if j == 0 then
4:     false
5: else
6:     e.casilleros[i][j-1]
7: end if
```

Complejidad: $O(1)$

iHayAdy↓(in c: `posicion`, **in** e: `estr` → res : bool

```
1: i ← π1(c)
2: j ← π2(c)
3: if j == e.tamaño then
4:     false
5: else
6:     e.casilleros[i][j+1]
7: end if
```

Complejidad: $O(1)$

iHayAdy←(in c: `posicion`, **in** e: `estr` → res : bool

```
1: i ← π1(c)
2: j ← π2(c)
3: if i == 0 then
4:     false
5: else
6:     e.casilleros[i-1][j]
7: end if
```

Complejidad: $O(1)$

iHayAdy→(in c: `posicion`, **in** e: `estr` → res : bool

```
1: i ← π1(c)
2: j ← π2(c)
3: if i == e.tamaño then
4:     false
5: else
6:     e.casilleros[i+1][j-1]
7: end if
```

Complejidad: $O(1)$

iAlcanceDisparo(**in** c: `posicion`,**in** d: `direccion`, **in** e: `estr` $\rightarrow$ res : conj(posicion)

```
1: i ← π1(c)
2: j ← π2(c)
3: if d == ↑ then
4:     if ¬ iHayAdy↑(c,e) then
5:         ∅
6:     else
7:         AgregarRapido(iAlcanceDisparo(< i − 1, j >, d, e), c)
8:     end if
9: end if
10: if d == ↓ then
11:     if ¬ iHayAdy↓(c,e) then
12:         ∅
13:     else
14:         AgregarRapido(iAlcanceDisparo(< i + 1, j >, d, e), c)
15:     end if
16: end if
17: if d == ← then
18:     if ¬ iHayAdy←(c,e) then
19:         ∅
20:     else
21:         AgregarRapido(iAlcanceDisparo(< i, j − 1 >, d, e), c)
22:     end if
23: end if
24: if d == → then
25:     if ¬ iHayAdy→(c,e) then
26:         ∅
27:     else
28:         AgregarRapido(iAlcanceDisparo(< i, j + 1 >, d, e), c)
29:     end if
30: end if
```

Complejidad: $O(m)$

Justificación: Recorre la línea de disparo hasta llegar al borde de la habitacion, o hasta encontrar una casilla ocupada.

2. Módulo JUEGO

Interfaz

parámetros formales
géneros

se explica con: TAD JUEGO

géneros: `juego`.

Operaciones básicas:

NUEVOJUEGO(**in** h : `habitacion`, **in** c : `conj(jugador)`, **in** f : `fantasma`) $\rightarrow$ res : `juego`
Pre $\equiv$ {$\neg$ vacio(c) $\wedge$ posicionesDelFantasma(f) $\in$ $posiciones(h)$}
Post$\equiv$ {$res =_{\text{obs}}$ nuevoJuego(h,c,f)}
Complejidad: $O(\#(c) * (\#(c) + |j|) + m^2)$
Descripción: Crea un nuevo Juego

EJECUTARACCION(**in/out** jue : `juego`, **in** j : `jugador`,**in** a : `accion`)
Pre $\equiv$ {$jue =_{\text{obs}} jue_0 \wedge j \in$ Jugadores(jue) $\wedge_{\text{L}}$jugadorVivo(j,jue) $\wedge \neg$ esPasar(a)}
Post $\equiv$ {jue $=_{\text{obs}}$ Step(jue,j,a)}
Complejidad: Si no cambia ronda: $O(|j| + fv * m + jv)$. Si cambia: $O(m + \#accionesDeJ + f + j^2)$
Descripción: Ejecuta la accion del jugador.

PASARSINACCION(**in/out** jue : `juego`)
Pre $\equiv$ {$jue =_{\text{obs}} jue_0$}
Post $\equiv$ {$jue =_{\text{obs}}$ Pasar(jue_0)}
Complejidad: $O()$
Descripción: Pasa el juego sin realizar accion alguna de los jugadores.

JUGADORES(**in** jue : `juego`) $\rightarrow$ res : `conj(jugador)`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ Jugadores(jue)}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de jugadores.

FANTASMAS(**in** jue : `juego`) $\rightarrow$ res : `conj(fantasma)`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ Fantasmas(jue)}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de fantasmas.

HABITACION(**in** jue : `juego`) $\rightarrow$ res : `habitacion`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ Habitacion(jue)}
Complejidad: $O(1)$
Descripción: Devuelve la habitacion asociada al juego.

ACCIONES(**in** j : `jugador`,**in** jue : `juego`) $\rightarrow$ res : `secu(evento)`
Pre $\equiv$ {j $\in$ $Jugadores(jue)$}
Post$\equiv$ {res $=_{\text{obs}}$ Acciones(j,jue)}
Complejidad: $O(|j|)$
Descripción: Devuelve la secuencia de acciones de un jugador.

FANTASMASVIVOS(**in** jue : `juego`) $\rightarrow$ res : `conj(fantasma)`
Pre $\equiv$ {true}
Post $\equiv$ {($\forall f$: fantasma) f $\in$ res $\Leftrightarrow$ ($f \in Fantasmas(jue) \wedge_{\text{L}} FantasmaVivo(j, jue)$)}
Complejidad:O(1)
Descripción: Devuelve el conjunto de fantasmas que estan vivos en la ronda actual.

JUGADORESVIVOS(**in** jue : `juego`) $\rightarrow$ res : `conj(jugador)`
Pre $\equiv$ {true}
Post $\equiv$ {$\forall j$: jugador j $\in$ res $\Leftrightarrow$ ($j \in Jugadores(jue) \wedge_{\text{L}} jugadorVivo(j, jue)$)}

Complejidad:O(1)
Descripción: Devuelve el conjunto de jugadores que estan vivos en la ronda actual.

FantasmaEspecial(**in** jue: `juego`) $\rightarrow$ res : `fantasma`
Pre $\equiv$ {true}
Post $\equiv$ {(res $=_{\text{obs}}$ FantasmaEspecial(jue)}
Complejidad: $O(1)$
Descripción: Devuelve el fantasma especial.

FantasmasDisparando(**in** jue: `juego`) $\rightarrow$ res : `conj(fantasma)`
Pre $\equiv$ {true}
Post $\equiv$ {$\forall f$: fantasma(f $\in$ res) $\Leftrightarrow$ ($f \in Fantasmas(jue) \wedge_{\text{L}} FantasmaVivo(f, jue) \wedge$ $disparando(f, maxCantAcciones(Jugadores(jue), jue))$)}
Complejidad:O(#fv)
Descripción: Devuelve el conjunto de los fantasmas vivos que disparan.

AlcanceDisparosFantasmas(**in** fc: `conj(fantasma)`,**in** jue: `juego`) $\rightarrow$ res : `conj(posicion)`
Pre $\equiv$ {fc $\subseteq$ Fantasmas(jue)}
Post $\equiv$ {res $=_{\text{obs}}$ AlcanceDisparosFantasmas(fc,jue)}
Complejidad: $O(fv * m + m^2)$
Descripción: Devuelve las posiciones disparadas por fantasmas.

Representación

El juego

`TAD JUEGO` **se representa con** `Struct`

donde `Struct` es `tupla`(*Jugadores*: `DiccString(jugador:String,Tupla(Lista(evento),`
`vivo?:bool, id:nat))`
, *Fantasmas*: `Lista(fantasma:Lista(evento))`
, *Habitacion*: `habitacion`
, *FantasmaEspecial*: `fantasma`
, *JugadoresVivos*: `DiccLineal(jugador,tupla(Lista(evento), id:nat)`
, *FantasmasVivos*: `Lista(fantasma:Lista(evento))`
, *DatosJugadoresVivos*: `DiccLineal(jugador,`
`tupla(itDiccLineal(jugador,tupla(Lista(evento), nat),`
`itDiccString(jugador,tupla(lista(evento),bool, nat))))`
, *DatosFantasmasVivos*: `Lista(tupla(itLista(fantasma),itLista(evento))`
, *casillerosDisparadosPorFan*: `arreglo_dimensionable[tamaño] de`
`arreglo_dimensionable[tamaño] de bool`
, *Turnos*: `nat`
)

donde `evento` es `tupla`(*posicion*: `tupla(nat,nat)`, *direccion*: `nat`, *Disparo*: `bool`)

Rep : Juego $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$

1. e.FantasmaEspecial siempre esta contenido en e.Fantasmas
2. Todas las claves de e.JugadoresVivos estan incluidas en claves de e.Jugadores
3. Para todo jugador que pernezca a $claves(e.JugadoresVivos)$ vale que $\pi_1(obtener(j, e.Jugadores)) = obtener(j, e.JugadoresVivos)$
4. Todos los fantasmas de e.Fantasmas vivos estan incluidas en e.Fantasmas
5. e.FantasmaEspecial siempre esta contenido en e.fantasmasVivos
6. las claves de e.DatosJugadoresVivos estan incluidas en e.Jugadores
7. claves(e.JugadoresVivos)=claves(e.DatosJugadoresVivos)
8. Para cada jugador j que es clave de e.DatosJugadoresVivos se tiene que $siguienteClave(\pi_1(obtener(j, e, DatosJugadoresVivos))) \in claves(e.JugadoresVivos)$ y $siguienteSignificado(\pi_1(obtener(j, e, DatosJugadoresVivos))) = obtener(j, e.JugadoresVivos)$ analogamente para los it al DiccTrie $siguienteClave(\pi_2(obtener(j, e, DatosJugadoresVivos))) \in claves(e.Jugadores)$ y $siguienteSignificado(\pi_2(obtener(j, e, DatosJugadoresVivos))) = obtener(j, e.Jugadores)$
9. Para cada elemento i de e.DatosFantasmasVivos $\pi_1(i) \in$ A e.FantasmasVivos y a e.Fantanmas y $\pi_2(i)$ esta incluido en en sus respectivos eventos en e.FantasmasVivos y e.Fantasmas
10. En e.habitacion y e.casillerosDisparadosPorFan el tamaño del arreglo es de tamaño n y cada arreglo dentro del arreglo también es de tamaño n, es decir el amcho y el alto de la matriz es de n x n

Función de abstracción:
Abs : estr e $\longrightarrow$ Juego {Rep(e)}
Abs(e) $\equiv$ c / e.jugadores = jugadores(c) $\wedge$
e.fantasmas = fantasmas(c) $\wedge$
e.fantasmaEspecial = fantasmaEspecial(c) $\wedge$
($\forall$ j $\in$ claves(e.jugadores))(obtener(e.jugadores) = accion(j,c))

Algoritmos

inuevoJuego(**in** h : `habitacion`,**in** c : `con(jugador)`,**in** f : `fantasma`) $\rightarrow$ res : estr

1: $itJugs \leftarrow CrearIt(c)$ ▷ $O(1)$
2: $DiccJugadores \leftarrow localizarjugadores(res)$ ▷ $O(1)$
3: id $\leftarrow Cardinal(c)$
4: **while** haySiguiente(itJugs) **do** ▷ $O(\#(c))$
5: $jugadorActual \leftarrow siguiente(itJugs)$ ▷ $O(1)$
6: $posIni \leftarrow \pi_1$(obtener(jugadorActual,DiccJugadores)) ▷ $O(\#(c))$
7: $dirIni \leftarrow \pi_2$(obtener(jugadorActual,DiccJugadores)) ▷ $O(\#(c))$
8: $evenIni \leftarrow$ Lista Vacia de evento ▷ $O(1)$
9: agregarAtras($< posIni, dirIni, False >$,evenIni) ▷ $O(1)$
10: $itTrie \leftarrow$ definir(jugadorActual,$< evenIni, True, id >$,res.jugadores) ▷ $O(|j|)$
11: $itJugVivo \leftarrow$ definir(jugadorActual, $< evenIni, id >$, res.JugadoresVivos)
12: definir(jugadorActual,$< itJugVivo, itTrie >$,res.DatosJugadoresVivos) ▷ $O(\#(c))$
13: $id--$
14: $avanzar(itJugs)$ ▷ $O(1)$
15: **end while**
16: $res.Habitacion \leftarrow h$ ▷ $O(1)$
17: $f \leftarrow CrearFantasma(f)$ ▷ longitud(f)
18: $res.FantasmaEspecial \leftarrow f$ ▷ $O(1)$
19: itFan $\leftarrow agregarAtras(res.FantasmasVivos, f)$ ▷ $O(1)$
20: itEveFan $\leftarrow$ CrearIt(f) ▷ $O(1)$
21: agregarAtras($< itFan, itEveFan >$,res.DatosFantasmasVivo) ▷ $O(1)$
22: $res.casillerosDisparadosPorFans \leftarrow arreglo\_dimensionable[Tamaño(h)]$ *de*
23: $arreglo\_dimensionable[Tamaño(h)]$ *de bool en* $false$ ▷ $O(m^2)$
24: $e.Turnos \leftarrow 0$ ▷ $O(1)$

Complejidad: $O(\#(c) * (\#(c) + |j|) + m^2 + longitud(f))$

Justificacion: Definimos a cada uno de los jugadores del conjunto en e.jugadores y guardamos en sus significados sus posiciones iniciales, obtenidas del diccionario de la función localizarJugadores. Ademas guardamos sus iteradores en las partes pertinentes de la estructura. Además, construimos un arreglo de arreglos de tamaño igual a la habitación lo cuál cuesta m^2.

iEjecutarAccion(**in/out** e : `estr`,**in** a : `accion`,**in** j : `jugador`)

1: $listaDeEventosDelJugadorActual \leftarrow \pi_1(obtener(j, e.Jugadores))$ ▷ $O(|j|)$
2: $posActualDelJugador \leftarrow \pi_1(ult(listaDeEventosDelJugadorActual))$ ▷ $O(1)$
3: $dirActualDelJugador \leftarrow \pi_2(ult(listaDeEventosDelJugadorActual))$ ▷ $O(1)$
4: $posicionDelFanEsp \leftarrow siguiente(\pi_2(ult(e.DatosFantasmaVivos)))$ ▷ $O(1)$
5: **if** $a == disparar \wedge posicionDelFanEsp \in$
6: $AlcanceDisparo(posActualDelJugador, dirActualDelJugador, e)$ **then** ▷ $O(m)$
7: $ultimoEventoDelJugador \leftarrow < posActualDelJugador, dirActualDelJugador, True >$ ▷ $O(1)$
8: $AgregarAtras(ultimoEventoDelJugador, listaDeEventosDelJugadorActual)$ ▷ $O(1)$
9: $nuevoFantasma \leftarrow CrearFantasma(listaDeEventosDelJugadorActual)$ ▷ $O(\#(accionesDeJ))$
10: $e.FantasmaEspecial \leftarrow nuevoFantasma$ ▷ $O(1)$
11: $AgregarAtras(nuevoFantasma, e.Fantasmas)$ ▷ $O(1)$
12: $e.FantasmasVivos \leftarrow e.Fantasmas$ ▷ $O(1)$
13: $ReiniciarDatosDeFantasmasVivos(e)$ ▷ $O(\#f)$
14: $ReiniciarJugadores(e)$ ▷ $O(\#j^2)$
15: $e.Turnos \leftarrow 0$ ▷ $O(1)$
16: **else**
17: **if** $a == disparar$ **then**
18: $ActualizarFantasmasVivos(e)$ ▷ $O(|j| + \#fv * m)$
19: **end if**
20: $AplicarAccionAlJugador(e, a, j)$ ▷ $O(\#jv + |j|)$
21: $ActualizarCasillasDisparadasPorFantasmas(e)$ ▷ $O(\#fv * m)$
22: $ActualizarDatosJugadoresVivosYJugadoresVivos(e)$ ▷ $O(\#jv)$
23: $ReiniciarCasillasDisparadasPorFantasmas(e)$ ▷ $O(\#fv * m)$
24: $ActualizarDatosDeFantasmasVivos(e)$ ▷ $O(\#fv)$
25: $AplicarPasarAJugadores(e)$ ▷ $O(\#jv)$
26: $e.Rondas++$ ▷ $O(1)$
27: **end if**

Complejidad:
Si no cambia ronda: $O(|j| + fv * m + jv)$
Si cambia: $O(m + \#accionesDeJ + \#f + \#j^2)$
Justificación:

28: Si no cambia ronda: $O(|j|) + 3 * O(1) + O(m) + O(|j| + \#fv * m) + O(\#jv + |j|) + O(\#fv * m) + O(\#jv) + O(\#fv * m) + O(\#fv) + O(\#jv) + O(1) =$
29: $O(|j|) + O(m) + O(|j| + \#fv * m) + O(\#jv + |j|) + O(\#fv * m) + O(\#jv) + O(\#fv * m) + O(\#fv) + O(\#jv) =$
30: $O(|j| + \#fv * m) + O(\#jv + |j|) + O(\#fv * m) =$
31: $O(|j| + \#fv * m) + O(\#jv + |j|) =$
32: $O(|j| + \#fv * m + \#jv)$
33: Si cambia ronda: $O(|j|) + 3 * O(1) + O(m) + 2 * O(1) + O(\#(accionesDeJ)) + 3 * O(1) + O(\#f) + O(j^2) + O(1) =$
34: $O(|j|) + O(m) + O(\#(accionesDeJ)) + O(\#f) + O(j^2) =$
35: $O(m + \#accionesDeJ + f + j^2)$

---

iJugadores(**in** e : `estr`) → res : conj(jugador)

1: res ← e.Jugadores
Complejidad: $O(1)$

---

iFantasmas(**in** e : `estr`) → res : conj(fantasma)

1: res ← e.Fantasmas
Complejidad: $O(1)$

---

iHabitacion(**in** e : `estr`)) → res :habitacion

1: res ← e.habitacion
Complejidad: $O(1)$

---

iAcciones(**in** j : `jugador`,**in** e : `estr`)) $\rightarrow$ res : secu(evento)

1: res $\leftarrow$ π_1(obtener(j,e.Jugadores))
Complejidad: $O(|j|)$

iFantasmasVivos(**in** e : `estr`)) $\rightarrow$ res : conj(fantasma)

1: res $\leftarrow$ e.FantasmasVivos
Complejidad: $O(1)$

iJugadoresVivos(**in** e : `estr`)) $\rightarrow$ res : conj(jugador)

1: res $\leftarrow$ e.jugadoresVivos
Complejidad: $O(1)$

iFantasmaEspecial(**in** e : `estr`)) $\rightarrow$ res : fantasma

1: res $\leftarrow$ e.FantasmaEspecial
Complejidad: $O(1)$

iFantasmasDisparando(**in** e : `estr`)) $\rightarrow$ res : conj(fantasma)

1: $itDFV \leftarrow crearIt(e.DatosFantasmasVivos)$ ▷ O(1)
2: **while** $haySiguiente(itDFV)$ **do** ▷ O(#fv)
3: $disparo? \leftarrow \pi_3(siguiente(\pi_2(siguiente(itFDV))))$ ▷ O(1)
4: $fantasma \leftarrow siguiente(\pi_1(siguiente(itFDV)))$ ▷ O(1)
5: **if** $disparo?$ **then** ▷ O(1)
6: $AgregarRapido(fantasma, res)$ ▷ O(1)
7: **end if**
8: $avanzar(itDFV)$ ▷ O(1)
9: **end while**
Complejidad: $O(\#fv)$
Justficacion: Se crea un iterador sobre todos los fantasmas vivos y se recorre la lista entera (la cantidad total de fantasmas vivos) por cada ciclo se hacen 5 operaciones con costo O(1), y esto se repite #fv veces

iPasarSinAccion(**in/out** e : `estr`))

1: $ActualizarDatosFantasmasVivos(e)$ ▷ $O(|j| + \#fv * m)$
2: $ActualizarDatosJugadoresVivosYJugadoresVivos(e)$ ▷ $O(\#jv)$
3: $itDatosJugadoresVivos \leftarrow CrearIt(e.DatosJugadoresVivos)$ ▷ $O(1)$
4: **while** $haySiguiente(itDatosJugadoresVivos)$ **do** ▷ $O(\#jv)$
5: $ListEvenJugVivos \leftarrow siguienteSignificado(\pi_1(siguienteSignificado(itDatosJugadoresVivos)))$ ▷ O(1)
6: $ultEven \leftarrow < \pi_1(ult(ListEvenJugVivos)), \pi_2(ult(ListEvenJugVivos)), false >$ ▷ $O(1)$
7: $ListEvenJug \leftarrow \pi_1(siguienteSignificado(\pi_2(siguienteSignificado(itDatosJugadoresVivos))))$ ▷ $O(1)$
8: $AgregarAtras(ListEvenJugVivos, ultimoEvento)$ ▷ $O(1)$
9: $AgregarAtras(ListEvenJug, ultEven)$ ▷ $O(1)$
10: $avanzar(itDatosJugadoresVivos)$ ▷ $O(1)$
11: **end while**
Complejidad: $O()$
Justificacion:

Algoritmos (Funciones no exportables)

---

iCrearFantasma(**in** f : `fantasma`)→ res :fantasma

1: $res \leftarrow extenderAccionesFantasma(f)$ ▷ O(1)
2: $reverso \leftarrow invertirAcciones(res)$ ▷ O(longitud(f))
3: $res \leftarrow res + +reverso$ ▷ O(1)
4: $res \leftarrow extenderAccionesFantasma(res)$ ▷ O(1)
5: $res \leftarrow res + +f$ ▷ O(1)

Complejidad: $O(longitud(f))$
Justificacion: Extendemos la secuencia con 5 "pasar", la recorremos para obtener su inverso y agregárselo y agregamos 5 "pasar"más.
Pre: f no es vacía.
Post: res es la secuencia original concatenada con aplicar "pasar"5 veces, su reverso y aplicar 5 pasar mas.

---

iExtenderAccionesFantasma(**in/out** f : `fantasma`)

1: $ultimaPosicion \leftarrow \pi_1(ult(f))$ ▷ O(1)
2: $ultimaDireccion \leftarrow \pi_2(ult(f))$ ▷ O(1)
3: i ← 0 ▷ O(1)
4: **while** $i < 5$ **do** ▷ O(1)
5: $AgregarAtras(< ultimaPosicion, ultimaDireccion, false >, f)$ ▷ O(1)
6: i++ ▷ O(1)
7: **end while**

Complejidad: $O(1)$
Justificacion: Por cada ciclo se hacen operaciones con costo O(1), y el while se ejecuta solo 5 veces.
Pre: f no es vacía y metavariable
Post: res es la secuencia original concatenada con aplicar "pasar"5 veces.

---

iInvertirAcciones(**in** f : `fantasma`) → res : lista(evento)

1: res ← lista() ▷ O(1)
2: itEventos ← CrearIt(f) ▷ O(1)
3: **while** haySiguiente(itEventos) **do** ▷ O(longitud(f))
4: $AgregarAdelante(< \pi_1(siguiente(itEventos))$,
5: $invertir(\pi_2(siguiente(itEventos)))$,
6: $\pi_3(siguiente(itEventos)) >, res)$ ▷ O(1)
7: avanzar(itEventos) ▷ O(1)
8: **end while**

Complejidad: $O(longitud(f))$,
Justificacion: Se crea un iterador para recorrer la lista de eventos. Por cada ciclo se realizan operaciones con costo O(1) y en total se hacen longitud(f) veces.
Pre: f no es vacía.
Post: res es la secuencia original con sus direcciones invertidas.

---

invertir(**in** d: `direccion`) $\rightarrow$ res : direccion

```
1: if d == arriba then                                                    ▷ O(1)
2:     res ← abajo                                                        ▷ O(1)
3: end if
4: if d == abajo then                                                     ▷ O(1)
5:     res ← arriba                                                       ▷ O(1)
6: end if
7: if d == derecha then                                                   ▷ O(1)
8:     res ← izquierda                                                    ▷ O(1)
9: end if
10: if d == izquierda then                                                ▷ O(1)
11:     res ← derecha                                                     ▷ O(1)
12: end if
```

Complejidad: $O(1)$

Justificacion: Todas las operaciones son comparaciones entre nats o asignacion de una nat.

Pre: true

Post: res es el inverso de d.

---

iReiniciarDatosDeFantasmasVivos(**in/out** e: `estr`)

```
1: itFantasmas ← CrearIt(e.Fantasmas)                                     ▷ O(1)
2: e.DatosFantasmasVivos ← Lista()                                        ▷ O(1)
3: while haysiguiente(itFantasmas) do                                     ▷ O(#f)
4:     itAlFantasma ← itFantasmas                                         ▷ O(1)
5:     iteradorAlEventoDelFantasma ← CrearIt(siguiente(itFantasmas))      ▷ O(1)
6:     AgregarAtras(< itAlFantasma, iteradorAlEventoDelFantasma >, e.DatosFantasmasVivos)   ▷ O(1)
7:     avanzar(itFantasmas)                                               ▷ O(1)
8: end while
```

Complejidad: $O(\#f)$

Justificacion: Se crea un iterador para recorrer toda la lista de Fantasmas, por cada ciclo se realizan operaciones con costo O(1), por algreba de ordenes se tiene que cada ciclo cuesta O(1) y en total se hacen #f ciclos.

Pre: metavariable

Post: Los iteradores de e.DatosFantasmasVivos apunta al principio de sus secuencias de eventos.

---

iReiniciarJugadores(**in/out** e: `estr`)

```
1: DiccJugadores ← localizarJugadores(e)                                  ▷ O(1)
2: itAJugadores ← CrearIt(e.Jugadores)                                    ▷ O(1)
3: while haySiguiente(itAJugadores) do                                    ▷ O(#j)
4:     id ← π3(SiguienteSignificado(itAJugadores))
5:     sigJugador ← siguienteClave(itAJugadores)                          ▷ O(1)
6:     listaDeEvento ← ListaVacia()                                       ▷ O(1)
7:     posYdirInicialDelJugador ← obtener(sigJugador, DiccJugadores)      ▷ O(#j)
8:     1erEventoDelJugador ← < posYdirInicialDelJugador, False >          ▷ O(1)
9:     nuevaListaDeEvento ← agregarAtras(1erEventoDelJugador, listaDeEvento)   ▷ O(1)
10:    siguienteSignificado(itAJugadores) ← < nuevaListaDeEvento, True, id >   ▷ O(1)
11:    itJugadoresVivos ← definir(sigJugador,
12:    < π1(siguienteSignificado(itAJugadores)), id >, e.JugadoresVivos)   ▷ O(#j)
13:    definir(sigJugador, < itJugadoresVivos, itAJugadores >, e.DatosJugadoresVivos)   ▷ O(#j)
14:    avanzar(itAJugadores)                                              ▷ O(1)
15: end while
```

Complejidad: $O(\#j^2)$

Justificacion: Por cada ciclo se tiene operaciones con costo O(1) y una operacion O(#j), y en total se hacen #j veces.

Pre: metavariable

Post: Las listas de eventos de los jugadores son de tamaño 1 y sus elementos coinciden con el resultado de localizarJugadores. Los iteradores de e.DatosJugadoresVivos apuntas a sus respectivos jugadores de e.Jugadores y e.JugadoresVivos.

---

iActualizarFantasmasVivos(**in/out** e : `estr`)

1: $listaDeEventosDelJugadorActual \leftarrow \pi_1(obtener(j, e.Jugadores))$ ▷ O($|j|$)
2: $posActualDelJugador \leftarrow \pi_1(ult(listaDeEventosDelJugadorActual))$ ▷ O(1)
3: $dirActualDelJugador \leftarrow \pi_2(ult(listaDeEventosDelJugadorActual))$ ▷ O(1)
4: $alcanceDelDisparoDelJugador \leftarrow AlcanceDisparo(posActualDelJugador, dirActualDelJugador, e)$ ▷ O(m)
5: $itDatFanVi \leftarrow CrearIt(e.DatosFantasmasVivos)$ ▷ O(1)
6: **while** $haySiguiente(itDatFanVi)$ **do** ▷ O(#fv)
7: posDelFantasma $\leftarrow \pi_1(siguiente(\pi_2(siguiente(itDatFanVi))))$ ▷ O(1)
8: **if** $posdelFantasma \in alcanceDelDisparoDelJugador$ **then** ▷ O(m)
9: $eliminarSiguiente(\pi_1(siguienteSignificado(itDatFanVi)))$ ▷ O(1)
10: $eliminarSiguiente(itDatFanVi)$ ▷ O(1)
11: **end if**
12: avanzar(itDatFanVi) ▷ O(1)
13: **end while**

Complejidad: $O(|j| + \#fv * m)$

Justificacion: Por cada ciclo se tienen opreaciones con costo O(1) y una con costo O(m) donde m representa el ancho de la habitacion,y en total se hace #fv veces, ademas afuera del ciclo se tiene una operacion con costo $O(|j|)$ donde j es largo maximo de un jugador , luego por algebra de ordenes se tiene $O(|j|)+O(\#fv)*O(m) = O(|j|+\#fv*m)$

Pre: metavariable

Post: Si la posición de un fantasma vivo estaba en el rango del disparo, entonces sus datos ya no van estar en e.DatosFantasmasVivos y en e.FantasmasVivos. Si no, los datos son iguales a como estaban antes.

---

iActualizarDatosJugadoresVivosYJugadoresVivos(**in/out** e : `estr`)

1: $itDatJv \leftarrow CrearIt(e.DatosJugadoresVivos)$ ▷ O(1)
2: **while** $haySiguiente(itDatJv)$ **do** ▷ O(#jv)
3: $posJ \leftarrow \pi_1(ult(siguienteSignificado(\pi_1(siguienteSignificado(itDatJv)))))$ ▷ O(1)
4: $i \leftarrow \pi_1(posJ)$ ▷ O(1)
5: $j \leftarrow \pi_2(posJ)$ ▷ O(1)
6: **if** $e.casillerosDisparadosPorFan[i][j]$ **then** ▷ O(1)
7: $eliminarSiguiente(\pi_1(siguienteSignificado(itDatJv)))$ ▷ O(1)
8: $\pi_2(siguienteSignificado(\pi_2(siguienteSignificado(itDatJv)))) \leftarrow False$ ▷ O(1)
9: $eliminarSiguiente(itDatJv)$ ▷ O(1)
10: **end if**
11: $avanzar(itDatJv)$ ▷ O(1)
12: **end while**

Complejidad: $O(\#jv)$

Justificacion: Se tiene un iterador que recorre la lista DatosFantasmasVivos, por cada ciclo se ejecutan en el peor caso 8 operaciones con costo O(1) y en total se hace #jv veces .

Pre: metavariable

Post: Si un jugador estaba vivo y su posición fue afectada por el disparo de un fantasma, entonces tiene un false en su significado en e.Jugadores y no aparece en e.JugadoresVivos ni en e.DatosJugadoresVivos.

iAplicarPasarAJugadores(**in/out** e : `estr`, **in** j : `jugador`)

1: itDatosJugadoresVivos ← CrearIt(e.DatosJugadoresVivos) ▷ O(1)
2: idJugador ← $\pi_3(obtener(j, e.Jugadores))$ ▷ $O(|j|)$
3: **while** haySiguiente(itDatosJugadoresVivos) **do** ▷ O(#jv)
4: **if** $\pi_3(siguiente(\pi_2(siguiente(itDatosJugadoresVivos))))$ ≠ idJugador **then** ▷ O(1)
5: ListEvenJugadoresVivos ← π_1(siguienteSignificado(
6: π_1(siguienteSignificado(itDatosJugadoresVivos)))) ▷ O(1)
7: ultEvento ← $< \pi_1(ult(ListEvenJugadoresVivos)), \pi_2(ult(ListEvenJugadoresVivos)), false >$ ▷ O(1)
8: ListaEvenJugadores ← π_1(siguienteSignificado(π_2(siguienteSignificado(itDatosJugadoresVivos)))) ▷ O(1)
9: AgregarAtras(ListaEvenJugadoresVivos, ultEvento) ▷ O(1)
10: AgregarAtras(ListaEvenJugadores, ulEvento) ▷ O(1)
11: **end if**
12: avanzar(itDatosJugadoresVivos) ▷ O(1)
13: **end while**
14:

Complejidad: $O(|j| + \#jv)$
Justificación: $O(1) + O(|j|) + O(\#jv) * O(1) + O(1) = O(|j| + \#jv)$
Pre: metavariable y j es un jugador del juego
Post: El último evento de las lista de eventos de los jugadores es igual a sus anteúltimos eventos con un false en el tercer componente, sin contar al jugador pasado por parámetro.

---

iActualizarDatosFantasmasVivos(**in** e : `estr`))

1: $itDFV \leftarrow CrearIt(e.DatosFantasmasVivos)$ ▷ $O(1)$
2: **while** $haySiguiente(itDFV)$ **do** ▷ $O(\#fv)$
3: $itListaEvento \leftarrow \pi_2(siguiente(itDFV))$ ▷ $O(1)$
4: **if** $haySiguiente(itListaEvento)$ **then** ▷ $O(1)$
5: $avanzar(itListaEvento)$ ▷ $O(1)$
6: **else**
7: $fantasmaActual \leftarrow siguiente(\pi_1(siguiente(itDFV)))$ ▷ $O(1)$
8: $\pi_2(siguiente(itDFV)) \leftarrow CrearIt(siguiente(fantasmaActual))$ ▷ $O(1)$
9: **end if**
10: $avanzar(itDVF)$ ▷ $O(1)$

Complejidad: $O(\#\text{fv})$
Justificación: Recorro la lista de fantasmas vivos. En cada iteracion se crean, piden siguiente o avanzan iteradores lo cual lleva tiempo constante.
Pre: metavariable
Post: Los iteradores en e.DatosFantasmasVivos son el resultado de aplicarles Siguiente a los iteradores del juego anterior. Si no tenían siguiente entonces apuntan al principio.

---

iActualizarCasillasDisparadasPorFantasmas(in/out e : estr))

```
1:  PyDFD ← ObtenerPosicionesYDirecciomesFantasmasDisparando(e)        ▷ O(#fv)
2:  itPyDFD ← CrearIt(PyDFD)                                           ▷ O(1)
3:  while haySiguiente(itPyDFD) do                                     ▷ O(#fv)
4:      i ← π1(π1(siguiente(itPyDFD)))                                 ▷ O(1)
5:      j ← π2(π1(siguiente(itPyDFD)))                                 ▷ O(1)
6:      if π2(siguiente(itPyDFD)) == arriba then                       ▷ O(1)
7:          while 0 ≤ i ∧ libre?(< i, j >, e.habitacion) do             ▷ O(m)
8:              e.casillerosDisparadosPorFans[i][j] ← True             ▷ O(1)
9:              i−−                                                    ▷ O(1)
10:         end while
11:     else
12:         if π2(siguiente(itPyDFD)) == abajo then                    ▷ O(1)
13:             while i < Tamaño(e.habitacion) ∧ libre?(< i, j >, e.habitacion) do   ▷ O(m)
14:                 e.casillerosDisparadosPorFans[i][j] ← True         ▷ O(1)
15:                 i++                                                ▷ O(1)
16:             end while
17:         else
18:             if π2(siguiente(itPyDFD)) == derecha then              ▷ O(1)
19:                 while j < Tamaño(e.habitacion) ∧ libre?(< i, j >, e.habitacion) do   ▷ O(1)
20:                     e.casillerosDisparadosPorFans[i][j] ← True     ▷ O(1)
21:                     j++                                            ▷ O(1)
22:                 end while
23:             else
24:                 if π2(siguiente(itPyDFD)) == abajo then            ▷ O(1)
25:                     while 0 ≤ j ∧ libre?(< i, j >, e.habitacion) do   ▷ O(m)
26:                         e.casillerosDisparadosPorFans[i][j] ← True ▷ O(1)
27:                         j−−                                        ▷ O(1)
28:                     end while
29:                 end if
30:             end if
31:         end if
32:     end if
33:     Avanzar(itPyDFD)                                               ▷ O(1)
34: end while
```

Complejidad: $O(\#\text{fv}^*m)$

Justificación: Recorro la lista de fantasmas vivos que disparan. Por cada uno verifico el alcance de sus disparos.

Pre: metavariable

Post: Las posiciones en true de e.casillasDisparadasPorFantasmas coinciden con las posiciones disparadas por fantasmas. Las que no fueron disparadas quedan en false.

=0

iReiniciarCasillasDisparadasPorFantasmas(**in** e : `estr`))

```
1:  PyDFD ← ObtenerPosicionesYDirecciomesFantasmasDisparando(e)                ▷ O(#fv)
2:  itPyDFD ← CrearIt(PyDFD)                                                   ▷ O(1)
3:  while haySiguiente(itPyDFD) do                                             ▷ O(#fv)
4:      i ← π1(π1(siguiente(itPyDFD)))                                         ▷ O(1)
5:      j ← π2(π1(siguiente(itPyDFD)))                                         ▷ O(1)
6:      if π2(siguiente(itPyDFD)) == arriba then                               ▷ O(1)
7:          while 0 ≤ i ∧ libre?(< i, j >, e.habitacion) do                    ▷ O(m)
8:              e.casillerosDisparadosPorFans[i][j] ← False                    ▷ O(1)
9:              i−−                                                            ▷ O(1)
10:         end while
11:     else
12:         if π2(siguiente(itPyDFD)) == abajo then                            ▷ O(1)
13:             while i < Tamaño(e.habitacion) ∧ libre?(< i, j >, e.habitacion) do   ▷ O(m)
14:                 e.casillerosDisparadosPorFans[i][j] ← False                ▷ O(1)
15:                 i++                                                        ▷ O(1)
16:             end while
17:         else
18:             if π2(siguiente(itPyDFD)) == derecha then                      ▷ O(1)
19:                 while j < Tamaño(e.habitacion) ∧ libre?(< i, j >, e.habitacion) do   ▷ O(m)
20:                     e.casillerosDisparadosPorFans[i][j] ← False            ▷ O(1)
21:                     j++                                                    ▷ O(1)
22:                 end while
23:             else
24:                 if π2(siguiente(itPyDFD)) == abajo then                    ▷ O(1)
25:                     while 0 ≤ j ∧ libre?(< i, j >, e.habitacion) do        ▷ O(m)
26:                         e.casillerosDisparadosPorFans[i][j] ← False        ▷ O(1)
27:                         j−−                                                ▷ O(1)
28:                     end while
29:                 end if
30:             end if
31:         end if
32:     end if
33:     Avanzar(itPyDFD)                                                       ▷ O(1)
34: end while
```

Complejidad: $O(\#\text{fv}^*m)$

Justificación: Recorro la lista de fantasmas vivos que disparan. Por cada uno verifico el alcance de sus disparos.

Pre: metavariable

Post: Todas las casillas de e.casillasDisparadasPorFantasmas están en false.

---

obtenerPosYdirFantasmasDisparando(**in** e : `estr`)) → res :$conj(tupla < posicion, direccion >)$

```
1:  res ← vacio()                                                              ▷ O(1)
2:  itDatosFantasmasVivos ← CrearIt(e.DatosFantasmasVivos)                     ▷ O(1)
3:  while haySiguiente(itDatosFantasmasVivos) do                               ▷ O(#fv)
4:      dispara? ← π3(siguiente(π2(siguiente(itDatosFantasmasVivos))))         ▷ O(1)
5:      if dispara? then                                                       ▷ O(1)
6:          posActualFan ← π1(siguiente(π2(siguiente(itDatosFantasmasVivos))))   ▷ O(1)
7:          disActualFan ← π2(siguiente(π2(siguiente(itDatosFantasmasVivos))))   ▷ O(1)
8:          agregarRapido(< posActualFan, disActualFan >, res)                 ▷ O(1)
9:      end if
10:     avanzar(itDatosFantasmasVivos)                                         ▷ O(1)
11: end while
```

Complejidad: $O(\#fv)$

Justificación: Por cada ciclo del while se realizan operaciones con costo O(1),y se realizan en total #fv veces.

Pre: true

Post: Las posiciones en res coinciden con las casillas disparadas por fantasmas.

iAplicarAccionAlJugador(**in/out** e : `estr`,**in** a : `accion`,**in** j : `jugador`)

```
1: datosJugador ← obtener(j, e.Jugadores)                                          ▷ O(|j|)
2: listaDeEventosDeJugadores ← π1(datosJugador)                                    ▷ O(1)
3: idJugador ← π3(datosJugador)                                                    ▷ O(1)
4: itJugadoresVivos ← CrearIt(e.JugadoresVivos)
5: encontrado ← false
6: while ¬ encontrado do                                                           ▷ O(#jv)
7:     if π2(siguiente(itJugadoresVivos)) == idJugador then                        ▷ O(1)
8:         encontrado ← true                                                       ▷ O(1)
9:     end if
10:    Avanzar(itJugadoresVivos)                                                   ▷ O(1)
11: end while
12: listaDeEventosDeJugadoresVivos ← π1(siguiente(itJugadoresVivos))               ▷ O(1)
13: ultPosDelJugador ← π1(ult(listaDeEventosDeJugadores))                          ▷ O(1)
14: ultDirDelJugador ← π2(ult(listaDeEventosDeJugadores))                          ▷ O(1)
15: if a == disparar  then                                                         ▷ O(1)
16:    nuevoEvento ← < ultPosDelJugador, ultDirDelJugador, True >                  ▷ O(1)
17:    agregarAtras(nuevoEvento, listaDeEventosDeJugadores)                        ▷ O(1)
18:    agregarAtras(nuevoEvento, listaDeEventosDeJugadoresVivos)                   ▷ O(1)
19: else
20:    if a == arriba then                                                         ▷ O(1)
21:        nuevaPosicion ← < π1(ultPosDelJugador), π2(ultPosDelJugador) + 1 >      ▷ O(1)
22:        nuevoEvento ← < nuevaPosicion, a, False >                               ▷ O(1)
23:        if libre?(nuevaPosicion,e.habitacion) then                              ▷ O(1)
24:            agregarAtras(nuevoEvento, listaDeEventosDeJugadores)                ▷ O(1)
25:            agregarAtras(nuevoEvento, listaDeEventosDeJugadoresVivos)           ▷ O(1)
26:        end if
27:    else
28:        if a == abajo then                                                      ▷ O(1)
29:            nuevaPosicion ← < π1(ultPosDelJugador) + 1, π2(ultPosDelJugador) >  ▷ O(1)
30:            nuevoEvento ← < nuevaPosicion, a, False >                           ▷ O(1)
31:            if libre?(nuevaPosicion,e.habitacion) then                          ▷ O(1)
32:                agregarAtras(nuevoEvento, listaDeEventosDeJugadores)            ▷ O(1)
33:                agregarAtras(nuevoEvento, listaDeEventosDeJugadoresVivos)       ▷ O(1)
34:            end if
35:        else
36:            if a == izquierda then                                              ▷ O(1)
37:                nuevaPosicion ← < π1(ultPosDelJugador) + 1, π2(ultPosDelJugador) − 1 >  ▷ O(1)
38:                nuevoEvento ← < nuevaPosicion, a, False >                       ▷ O(1)
39:                if libre?(nuevaPosicion,e.habitacion) then                      ▷ O(1)
40:                    agregarAtras(nuevoEvento, listaDeEventosDeJugadores)        ▷ O(1)
41:                    agregarAtras(nuevoEvento, listaDeEventosDeJugadoresVivos)   ▷ O(1)
42:                end if
43:            else
44:                if a == derecha then                                            ▷ O(1)
45:                    nuevaPosicion ← < π1(ultPosDelJugador) + 1, π2(ultPosDelJugador) − 1 >  ▷ O(1)
46:                    nuevoEvento ← < nuevaPosicion, a, False >                   ▷ O(1)
47:                    if libre?(nuevaPosicion,e.habitacion) then                  ▷ O(1)
48:                        agregarAtras(nuevoEvento, listaDeEventosDeJugadores)    ▷ O(1)
49:                        agregarAtras(nuevoEvento, listaDeEventosDeJugadoresVivos)  ▷ O(1)
50:                    end if
51:                else
52:                    nuevoEvento ← < ultPosDelJugador, ultDirDelJugador, False >  ▷ O(1)
53:                    agregarAtras(nuevoEvento, listaDeEventosDeJugadores)        ▷ O(1)
54:                    agregarAtras(nuevoEvento, listaDeEventosDeJugadoresVivos)   ▷ O(1)
55:                end if
56:            end if
57:        end if
58:    end if
59: end if
```

Complejidad: $O(\#jv + |j|)$

Justificación: $O(|j|) + O(1) + O(\#jv) * O(1) + O(1) = O(\#jv + |j|)$

Pre: metavariable y a no es "pasarz j está en el juego18

Pots: Las listas de eventos del jugador en la estructura, coinciden con aplicarle la acción a las listas de eventos del jugador en la estructura anterior.

3. Módulo DiccString($string$, σ)

Interfaz

parámetros formales
géneros $string$, σ

se explica con: DICCIONARIO(κ, σ)

géneros: `diccString`($string, \sigma$).

Operaciones básicas:

VACÍO() $\rightarrow$ res : `diccString`($string, \sigma$)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ vacio()}
Complejidad: $O(1)$
Descripción: genera un diccionario vacío.

BUSCAR(**in/out** d: `diccString`($string, \sigma$), **in** k: $string$) $\rightarrow$ res : σ
Pre $\equiv$ {$d =_{\text{obs}} d_0$}
Post $\equiv$ {($\neg$definido?(d_0, k) $\Rightarrow$ $d =_{\text{obs}}$ definir(d_0, k, s)) $\wedge$ ($res =_{\text{obs}}$ Obtener(d, k))}
Complejidad: $O(|k|)$. Si el significado no existe: $O(|k| + CrearSignificado)$
Descripción: Busca la clave k en el diccionario, la define si no existe y, en ese caso, crea un significado por defecto. Luego lo devuelve.
Aliasing: Devuelve una referencia mutable al significado.

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

OBTENER(**in** d: `diccString`($string, \sigma$), **in** k: $string$) $\rightarrow$ res : σ
Pre $\equiv$ {def?(d, k)}
Post $\equiv$ {alias($res =_{\text{obs}}$ obtener(d, k))}
Complejidad: $O(|k|)$
Descripción: devuelve el significado de la clave k en d.
Aliasing: Devuelve una referencia mutable al significado.

Representación

El diccionario se representa con un árbol Trie, representado con vectores de punteros a nodos, que contienen el significado y los hijos.

`diccString`($string, \sigma$) **se representa con** `dic`

donde `dic` es `tupla`(*raiz*: `puntero(Nodo)`)

donde `Nodo` es `tupla`(*hijos*: `arreglo_estatico[256](puntero(Nodo))`, *significado*: `puntero(`σ`)`)

Invariante de Representación
Rep : DiccString(string $\times$ σ) $\longrightarrow$ bool
Rep($dicc$) $\equiv$ true $\Longleftrightarrow$

1. El Trie no debe tener ciclos, ni nodos con dos padres.
2. Una clave definida en el trie lo és si y solo si el final de esa clave tiene un puntero a un significado.
3. Todas las hojas del trie tienen un significado

Función de abstracción:
Abs : dicc dic $\longrightarrow$ Diccionario($string$, σ) {Rep(dic)}

Abs(dic) $\equiv$ dt : Diccionario($string$, σ) |
((∀$clave$:$string$)(Definido?($dic, clave$) $=_{\text{obs}}$ def?($clave$, dt)))∧
Definido?($dic, clave$) ⇒ obtener($clave, dt$) $=_{\text{obs}}$ Obtener($dic, clave$))

Representación del iterador

El iterador se representará como un diccionario lineal con clave string y significado de tipo puntero a Nodo. Cada vez que se defina una nueva clave en el trie, se agregará en el diccionario lineal dicha clave y su correspondiente Nodo como significado por lo tanto bastará utilizar los iteradores de un diccionario lineal para recorrerlo. Luego se tendrán las típicas funciones de iteradores de un diccionario como Siguiente, HaySiguiente, etc.. SiguienteClave devolverá el string al cuál pertenece el Nodo al que apunta el iterador, mientras que SiguienteSignificado devolvera una referencia al significado de dicho Nodo. Por cuestiones de implementación, no se encontrarán disponibles las funciones relacionadas con eliminar elementos del trie por medio de iteradores.

Algoritmos

---

iVacio() → res : $DiccTrie(string, \sigma)$

1: $res \leftarrow \langle$ $raiz$: NULL $\rangle$
Complejidad: $O(1)$

NuevoNodo() → res : Puntero(Nodo)

1: $res \leftarrow$ NuevoPuntero(⟨$hijos$: NuevoArreglo_estatico[256](), $significado$: NULL ⟩)
2: **for** i desde 0 a 255 **do**
3: $res.hijos[i] \leftarrow$ NULL
4: **end for**
Complejidad: $O(1)$
Justificación El arreglo de hijos se tiene que inicializar en NULL y recorrerlo toma siempre 256 operaciones que son independientes del tamaño de la entrada por lo tanto se puede acotar por una constante.
Pre: True
Post: Devuelve una estructura del tipo Nodo

---

posicion($char$) → res : nat

1: Devuelve un valor entre 0 y 255 que representa el código ascii del $char$
Complejidad: $O(1)$
Pre: True
Post: res está entre 0 y 255

---

iBuscar(in/out d: `diccTrie(`$string, \sigma$`)`, **in** k: $string$) $\rightarrow res : \sigma$

1: **if** $d.raiz ==$ NULL **then**
2: $d.raiz \leftarrow$ NuevoNodo()
3: **end if**
4: $nodo \leftarrow d.raiz$
5: $posicion \leftarrow 0$
6: $def \leftarrow true$
7: **for** $letra$ en k **do**
8: **if** no está definida la posicion($letra$) en $nodo \rightarrow hijos$ **then**
9: $nodo \rightarrow hijos$[posicion($letra$)] $\leftarrow$ NuevoNodo()
10: $def \leftarrow false$
11: **end if**
12: $posicion \leftarrow$ posicion($letra$)
13: $nodo \leftarrow nodo \rightarrow hijos[posicion]$
14: **end for**
15: $significado \leftarrow nodo \rightarrow hijos[posicion] \rightarrow significado$
16: **if** $!def$ || $significado == NULL$ **then**
17: $significado \leftarrow$ NuevoSignificado()
18: **end if**
19: $res \leftarrow *significado$

Complejidad: $O(|k|)$. En el caso de agregar un significado nuevo: $O(|k| + NuevoSignificado())$
Justificación: Toma el largo del *string* k ya que debe recorrerlo para agregarlo al diccTrie y devuelve una referencia mutable al significado. Si este no existe, crea un significado por defecto.

---

iDefinido?(in d: `diccTrie(`$string, \sigma$`)`, **in** k: $string$) $\rightarrow res$: Bool

1: **if** $d.raiz ==$ NULL **then**
2: $res \leftarrow false$
3: return
4: **end if**
5: $arreglo \leftarrow *(d.raiz)$
6: $posicion \leftarrow 0$
7: **for** $letra$ en k **do**
8: **if** no está definida la posicion($letra$) en $arreglo$ **then**
9: $res \leftarrow false$
10: return
11: **end if**
12: $posicion \leftarrow$ posicion($letra$)
13: $arreglo \leftarrow arreglo[posicion] \rightarrow hijos$
14: **end for**
15: **if** $arreglo[posicion] \rightarrow significado$ es NULL **then**
16: $res \leftarrow false$
17: return
18: **end if**
19: $res \leftarrow true$

Complejidad: $O(|k|)$
Justificación: Toma solo el largo del *string* k ya que debe recorrerlo para saber si está en el diccTrie.

---

iObtener(in/out d: `diccTrie(`$string, \sigma$`)`, **in** k: $string$) $\rightarrow res : \sigma$

1: $res \leftarrow$ *Buscar(d, k)

Complejidad: $O(|k|)$
Justificación: Toma el largo del *string* k ya que debe recorrerlo para buscarlo en el diccTrie y devuelve el significado por referencia.