

Trabajo práctico 2: Diseño

v1.4

ExorcismoExtremo

Normativa

Límite de entrega: Miércoles 5 de junio *hasta las 23:59 hs.* Enviar el zip al mail `algo2.dc+tp2@gmail.com`. Ver detalles de entrega abajo.

Normas de entrega: Ver “Información sobre la cursada” en el sitio Web de la materia.
(<http://campus.exactas.uba.ar>)

1. Enunciado

El segundo trabajo práctico consiste en el diseño de los módulos necesarios para implementar una versión del juego Exorcismo Extremo, respetando la especificación de referencia que les presentamos desde la cátedra. La descripción del juego es la misma que ya introdujimos en el primer trabajo práctico. El **Juego** se va a iniciar pasando como parámetros un conjunto de **Jugadores** y un **Mapa** que contenga la información de tamaño y casillas ocupadas.

Las siguientes funciones son las que van a modificar el estado del juego realizando los cambios relacionados al paso de un turno:

- `ejecutarAccion`, va a tomar un **Jugador** y una **Acción** y va a resolver el estado de los elementos involucrados de acuerdo a las reglas definidas anteriormente
- `pasarTiempo`, ejecuta un paso de tiempo cuando ningún jugador realiza una acción.

Para las definiciones de las cotas de complejidad vamos a emplear los siguientes términos:

- $|j|$ para el largo máximo del nombre de un **Jugador**
- $\#j$ para la cantidad de **Jugadores**
- $\#jv$ para la cantidad de **Jugadores** vivos
- r para el índice entero que representa la cantidad de **Rondas** ocurridas hasta e incluyendo a la actual
- $\#fv$ para la cantidad de **Fantasmas** vivos
- m para el ancho o alto del **Mapa**

Se deben proveer las siguientes operaciones, con las complejidades temporales en **peor caso** indicadas:

1. Conocer identidad, posición y dirección actual de los **Jugadores** vivos en $O(1)$.
2. Conocer posición y dirección actual de los **Fantasmas** vivos en $O(1)$.
3. Conocer posición y dirección actual del **FantasmaEspecial** en $O(1)$.
4. Conocer posición y dirección de **Fantasmas** vivos que disparan en el último **Paso** ejecutado en el **Juego** $O(\#fv)$
5. Conocer si un jugador está vivo en $O(|j|)$.
6. Actualizar con **Acción** de **Jugador** en $O(|j| + \#fv * m + \#jv)$ si no cambia la ronda.
7. Actualizar sin **Acción** de **Jugador** en $O(\#fv * m + \#jv)$ si no cambia la ronda.
8. Conocer las posiciones ocupadas por disparos de fantasmas de la última ronda en $O((\#fv * m)^2)$. El conjunto de casilleros resultados no puede tener repetidos (bonus: se puede resolver en $O(\#fv * m)$).

Al igual que en Trabajo Práctico 1, se puede asumir la existencia de una función global `dict<Jugador, pair<Pos, Dir>> localizar_jugadores(const Juego& j)` que se encargará de definir la posición de los jugadores en la ronda actual del juego pasado por parámetro.

2. Documentación a entregar

Todos los módulos diseñados deben contar con las siguientes partes. Se debe diseñar el módulo principal (*ExorcismoExtremo*) y todos los módulos auxiliares. La única excepción son los módulos disponibles en el Apunte de Módulos Básicos, que se pueden utilizar sin diseñarlos: lista enlazada, pila, cola, vector, diccionario lineal, conjunto lineal y conjunto acotado de naturales.

1. Interfaz.

- 1.1. *Tipo abstracto* (“se explica con ...”). Género (TAD) que sirve para explicar las instancias del módulo, escrito en el lenguaje de especificación **formal** de la materia. Pueden utilizar la especificación que se incluye en el apéndice.
- 1.2. *Signatura*. Listado de todas las funciones públicas que provee el módulo. La signatura se puede escribir, dependiendo de sus preferencias:
 - Con la notación de módulos de la materia: `apilar(in/out pila : PILA, in x : ELEMENTO)`
 - Con notación de C++: `void Pila::apilar(const Elemento& x)`
- 1.3. *Contrato*. Precondición y postcondición de todas las funciones públicas. Las precondiciones de las funciones de la interfaz deben estar expresadas **formalmente** en lógica de primer orden.¹
- 1.4. *Complejidades*. Complejidades de todas las funciones públicas, cuando corresponda.
- 1.5. *Aspectos de aliasing*. De ser necesario, aclarar cuáles son los parámetros y resultados de los métodos que se pasan por copia y cuáles por referencia, y si hay *aliasing* entre algunas de las estructuras.

2. Implementación.

- 2.1. *Representación* (“se representa con ...”). Módulo con el que se representan las instancias del módulo actual.
 - 2.2. *Invariante de representación*. Puede estar expresado en lenguaje natural o formal.
 - 2.3. *Función de abstracción*. Puede estar expresada en lenguaje natural o formal.
 - 2.4. *Algoritmos*. Pueden estar expresados en pseudocódigo, usando si es necesario la notación del lenguaje de módulos de la materia o notación tipo C++. Las pre y postcondiciones de las funciones auxiliares pueden estar expresadas en lenguaje natural (no es necesario que sean formales). Indicar de qué manera los algoritmos cumplen con el contrato declarado en la interfaz y con las complejidades pedidas. No se espera una demostración formal, pero sí una justificación adecuada.
3. **Servicios usados**. Módulos que se utilizan, detallando las complejidades, *aliasing* y otros aspectos que dichos módulos deben proveer para que el módulo actual pueda cumplir con su interfaz.

Sobre el uso de lenguaje natural y formal.

Las precondiciones y poscondiciones de las funciones auxiliares, el invariante y la función de abstracción pueden estar expresados en lenguaje natural. No es necesario que sean formales. Asimismo, los algoritmos pueden estar expresados en pseudocódigo. Por otro lado, está permitido que utilicen fórmulas en lógica de primer orden en algunos lugares puntuales, si consideran que mejora la presentación o subsana alguna ambigüedad. El objetivo del diseño es convencer al lector, y a ustedes mismos, de que la interfaz pública se puede implementar usando la representación propuesta y respetando las complejidades pedidas. Se recomienda aplicar el sentido común para priorizar la **claridad** y **legibilidad** antes que el rigor lógico por sí mismo. Por ejemplo:

Más claro

“Cada clave del diccionario *D* debe ser una lista sin elementos repetidos.” ✓
 “sinRepetidos?(claves(*D*))” ✓

“Ordenar la lista *A* usando mergesort.” ✓
 “*A*.mergesort()” ✓

“Para cada tupla (*x*, *y*) en el conjunto *C* {
 x.apilar(*y*)
 n++
 }” ✓

Menos claro

“No puede haber repetidos.” (¿En qué estructura?).

“Ordenar los elementos.” (¿Qué elementos? ¿Cómo se ordenan?).

“Miro las tuplas del conjunto, apilo la segunda componente en la primera y voy incrementando un contador.” (Ambiguo y difícil de entender).

¹ Si la implementación requiere usar funciones auxiliares, sus pre y postcondiciones pueden estar escritas en lenguaje natural, pero esto no forma parte de la interfaz.

Entrega

La entrega consistirá de un único documento digital con la documentación de los módulos diseñados. Se recomienda el uso de los paquetes de L^AT_EX de la cátedra para lograr una mejor visualización del informe.

La entrega se realizará por mail a la dirección `algo2.dc+tp2@gmail.com`. El documento desarrollado se entregará como un archivo en formato **pdf** hasta el día 2 de Junio a las 23:59hs. El mail deberá tener como **Asunto** los números de libreta separados por `;`. Por ejemplo:

To: algo2.dc+tp2@gmail.com
From: alumno-algo2@dc.uba.ar
Subject: 102/09; 98/10
Adjunto: tp2.zip

A. Especificación formal

Usaremos los siguientes tipos auxiliares. El tipo **STRING** es un arreglo dimensionable de caracteres, extendido con una operación $\bullet = \bullet : \text{string} \times \text{string} \rightarrow \text{bool}$ para compararlas por igualdad. Cada caracter es un número entre 0 y 255. $\bullet = \bullet : \text{operación} \times \text{operación} \rightarrow \text{bool}$ para compararlas por igualdad.

- **TAD CHAR** es **ENUM**(0, 1, ..., 255)
- **TAD STRING** es **ARREGLO DIMENSIONABLE**(CHAR)
- **TAD POSICIÓN** es **TUPLA**(NAT, NAT) (**genero:** pos)
- **TAD JUGADOR** es **STRING** (**genero:** jug)
- **TAD FANTASMA** es **SECU**(EVENTO) (**genero:** fantasma)
- **TAD EVENTO** es **TUPLA**(POSICIÓN, DIRECCIÓN, BOOL) (**genero:** evt)

A.1. HABITACIÓN

TAD HABITACIÓN

géneros hab

exporta hab, generadores, observadores, ...

usa BOOL, NAT, POSICIÓN, ACCIÓN, DIRECCIÓN, ...

igualdad observacional

$$(\forall h, h' : \text{hab})) \left(h =_{\text{obs}} h' \iff \left(\begin{array}{l} \text{tam}(h) =_{\text{obs}} \text{tam}(h') \wedge_L \\ \forall c: \text{pos} (c \in \text{casilleros}(h) \\ \Rightarrow \text{libre}(c, h) =_{\text{obs}} \text{libre}(c, h')) \end{array} \right) \right)$$

observadores básicos

tam	: hab	→ Nat	
libre	: pos c × hab h	→ Bool	{c ∈ casilleros(h)}

generadores

nuevaHab	: Nat n	→ hab	
ocupar	: pos c × hab h	→ hab	
			{c ∈ casilleros(h) ∧ _L libre(c,h) ∧ alcanzan(libres(h)-c, libres(h)-c)}

otras operaciones

casilleros	: hab	→ conj(pos)	
armarPares	: Nat i ₁ × Nat f ₁ × Nat i ₂ × Nat f ₂	→ conj(pos)	{f ₁ ≥ i ₁ ∧ f ₂ ≥ i ₂ }
libres	: hab	→ conj(pos)	
libresEn	: conj(pos) cs × hab h	→ conj(pos)	
alcanzan	: conj(pos) cs ₁ × conj(pos) cs ₂ × hab h	→ Bool	{cs ₁ ⊆ casilleros(h) ∧ cs ₂ ⊆ casilleros(h)}
explorar	: conj(pos) cs ₁ × pos cs ₂ × hab h	→ conj(pos)	{cs ₁ ⊆ casilleros(h) ∧ cs ₂ ⊆ casilleros(h)}
hayAdy↑	: pos c × hab h	→ Bool	{c ∈ casilleros(h)}
hayAdy↓	: pos c × hab h	→ Bool	{c ∈ casilleros(h)}
hayAdy←	: pos c × hab h	→ Bool	{c ∈ casilleros(h)}
hayAdy→	: pos c × hab h	→ Bool	{c ∈ casilleros(h)}
ady↑	: pos c × hab h	→ pos	{c ∈ casilleros(h) ∧ hayAdy↑(c,h)}
ady↓	: pos c × hab h	→ pos	{c ∈ casilleros(h) ∧ hayAdy↓(c,h)}
ady←	: pos c × hab h	→ pos	{c ∈ casilleros(h) ∧ hayAdy←(c,h)}
ady→	: pos c × hab h	→ pos	{c ∈ casilleros(h) ∧ hayAdy→(c,h)}
vecinos	: pos c × hab h	→ conj(pos)	{c ∈ casilleros(h)}

vecinosLibres	: pos $c \times$ hab h	\longrightarrow conj(pos) $\{c \in \text{casilleros}(h)\}$
mover	: pos $p \times$ dir \times hab h	\longrightarrow pos $\{p \in \text{posiciones}(h)\}$
alcanceDisparo	: pos $p \times$ dir $d \times$ hab h	\longrightarrow conj(pos) $\{p \in \text{posiciones}(h)\}$
alcanceDisparo \uparrow	: pos $p \times$ hab h	\longrightarrow conj(pos) $\{p \in \text{posiciones}(h)\}$
alcanceDisparo \downarrow	: pos $p \times$ hab h	\longrightarrow conj(pos) $\{p \in \text{posiciones}(h)\}$
alcanceDisparo \leftarrow	: pos $p \times$ hab h	\longrightarrow conj(pos) $\{p \in \text{posiciones}(h)\}$
alcanceDisparo \rightarrow	: pos $p \times$ hab h	\longrightarrow conj(pos) $\{p \in \text{posiciones}(h)\}$

axiomas

tam(nuevaHab(n))	\equiv n
tam(ocupar(c, h))	\equiv tam(h)
libre(c, nuevaHab(n))	\equiv True
libre(c, ocupar(c', h))	\equiv if $c = c'$ then False else libre(c, h) fi
casilleros(h)	\equiv armarPares(0, tam(h)-1, 0, tam(h)-1)
armarPares(i_1, f_1, i_2, f_2)	\equiv Ag($\langle i_1, i_2 \rangle, \emptyset$) \cup if $i_2 = f_2$ then if $i_1 = f_1$ then \emptyset else armarPares($i_1 + 1, f_1, i_2, f_2$) fi else if $i_1 = f_1$ then armarPares(0, $f_1, i_2 + 1, f_2$) else armarPares($i_1 + 1, f_1, i_2, f_2$) fi fi
libres(h)	\equiv libresEn(casilleros(h), h)
libresEn(cs, h)	\equiv if vacio(cs) then \emptyset else if libre(dameUno(cs), h) then Ag(dameUno(cs), libresEn(sinUno(cs), h)) else libresEn(sinUno(cs), h) fi fi
alcanzan(cs_1, cs_2, h)	\equiv if vacio(cs_1) then True else if explorar($\{dameUno(cs_1)\}, dameUno(cs_1), h$) = cs_2 then alcanzan(sinUno(cs_1), cs_2, h) else False fi fi

```

explorar(exp,c,h)      ≡ if vacio(vecinosLibres(c,h)-exp) then
    exp
  else
    exp ∪
    if hayAdy↑(c,h) ∧L ady↑(c,h) ∈ (vecinosLibres(c,h)-exp) then
      explorar(exp ∪ vecinosLibres(c,h), ady↑(c,h),h)
    else
      ∅
    fi ∪
    if hayAdy↑(c,h) ∧L ady↑(c,h) ∈ (vecinosLibres(c,h)-exp) then
      explorar(exp ∪ vecinosLibres(c,h), ady↓(c,h),h)
    else
      ∅
    fi ∪
    if hayAdy←(c,h) ∧L ady←(c,h) ∈ (vecinosLibres(c,h)-exp) then
      explorar(exp ∪ vecinosLibres(c,h), ady←(c,h),h)
    else
      ∅
    fi ∪
    if hayAdy→(c,h) ∧L ady→(c,h) ∈ (vecinosLibres(c,h)-exp) then
      explorar(exp ∪ vecinosLibres(c,h), ady→(c,h),h)
    else
      ∅
    fi
  fi
hayAdy↑(c,h)          ≡  $\Pi_2(c) < \text{tam}(h) - 1$ 
hayAdy↓(c,h)          ≡  $\Pi_2(c) > 0$ 
hayAdy←(c,h)          ≡  $\Pi_1(c) > 0$ 
hayAdy→(c,h)          ≡  $\Pi_1(c) < \text{tam}(h) - 1$ 
ady↑(c,h)              ≡  $\langle \Pi_1(c), \Pi_2(c) + 1 \rangle$ 
ady↓(c,h)              ≡  $\langle \Pi_1(c), \Pi_2(c) - 1 \rangle$ 
ady←(c,h)              ≡  $\langle \Pi_1(c) - 1, \Pi_2(c) \rangle$ 
ady→(c,h)              ≡  $\langle \Pi_1(c) + 1, \Pi_2(c) \rangle$ 
vecinos(c,h)           ≡ if hayAdy↑(c,h) ∧ then { ady↑(c,h) } else ∅ fi ∪
    if hayAdy↓(c,h) ∧ then { ady↓(c,h) } else ∅ fi ∪
    if hayAdy←(c,h) ∧ then { ady←(c,h) } else ∅ fi ∪
    if hayAdy→(c,h) ∧ then { ady→(c,h) } else ∅ fi
vecinosLibres(c,h)     ≡ libresEn(vecinos(c,h),h)
mover(p,↑,h)           ≡ if hayAdy↑(p,h) ∧L libre(ady↑(p,h),h) then ady↑(p,h) else p fi
mover(p,↓,h)           ≡ if hayAdy↓(p,h) ∧L libre(ady↓(p,h),h) then ady↓(p,h) else p fi
mover(p,←,h)           ≡ if hayAdy←(p,h) ∧L libre(ady←(p,h),h) then ady←(p,h) else p fi
mover(p,→,h)           ≡ if hayAdy→(p,h) ∧L libre(ady→(p,h),h) then ady→(p,h) else p fi

```

```

alcanceDisparo(p,d,h)  ≡  if mover(p, d, h) = p then
    ∅
  else
    if d = ↑ then
      alcanceDisparo↑(mover(p, d, h), h)
    else
      if d = ↓ then
        alcanceDisparo↓(mover(p, d, h), h)
      else
        if d = ← then
          alcanceDisparo←(mover(p, d, h), h)
        else
          if d = → then
            alcanceDisparo→(mover(p, d, h), h)
          else
            fi
          fi
        fi
      fi
    fi
  fi

alcanceDisparo↑(p,h)  ≡  if p ∈ libres(h) then
    if hayAdy↑(p,h) then
      Ag(p, alcanceDisparo(ady↑(p,h)))
    else
      Ag(p, ∅)
    fi
  else
    ∅
  fi

alcanceDisparo↓(p,h)  ≡  if p ∈ libres(h) then
    if hayAdy↓(p,h) then
      Ag(p, alcanceDisparo(ady↓(p,h)))
    else
      Ag(p, ∅)
    fi
  else
    ∅
  fi

alcanceDisparo←(p,h)  ≡  if p ∈ libres(h) then
    if hayAdy←(p,h) then
      Ag(p, alcanceDisparo(ady←(p,h)))
    else
      Ag(p, ∅)
    fi
  else
    ∅
  fi

```

```

alcanceDisparo → (p, h)  ≡  if p ∈ libres(h) then
                                if hayAdy → (p, h) then
                                    Ag(p, alcanceDisparo(ady → (p, h)))
                                else
                                    Ag(p, ∅)
                                fi
                                else
                                    ∅
                                fi

```

Fin TAD**TAD ACCIÓN**

géneros acc

exporta acc, generadores, observadores, ...

usa BOOL, NAT, ACCIÓN, SECUENCIA(α)

igualdad observacional

$$(\forall a, a' : \text{acc}) \left(a =_{\text{obs}} a' \iff \left(\begin{array}{l} \text{esPasar}(a) =_{\text{obs}} \text{esPasar}(a') \wedge_L \\ \text{esDisparar}(a) =_{\text{obs}} \text{esDisparar}(a') \wedge_L \\ \text{esDireccion}(a) =_{\text{obs}} \text{esDireccion}(a') \wedge_L \\ \text{esDireccion}(a) \Rightarrow_L (\text{direccion}(a) =_{\text{obs}} \text{direccion}(a')) \end{array} \right) \right)$$

observadores básicos

esDisparar	: acc	→ Bool	
esPasar	: acc	→ Bool	
esMover	: acc	→ Bool	
direccion	: acc a	→ dir	{esMover(a)}

generadores

mover	: dir	→ acc
pasar	:	→ acc
disparar	:	→ acc

otras operaciones

aplicar	: acc × juego × evt	→ evt
pasar	: evt	→ evt
invertir	: evt	→ evt
inversa	: secu(evt)	→ secu(evt)

axiomas

esDisparar(disparar))	≡ True
esDisparar(pasar))	≡ False
esDisparar(mover(d)))	≡ False
esPasar(disparar))	≡ False
esPasar(pasar))	≡ True
esPasar(mover(d)))	≡ False
esMover(disparar))	≡ False
esMover(pasar))	≡ False
esMover(mover(d)))	≡ True
pasar(e)	≡ $\langle \Pi_1(e), \Pi_2(d), \text{False} \rangle$
aplicar(disparar, g, e)	≡ $\langle \Pi_1(e), \Pi_2(e), \text{True} \rangle$
aplicar(pasar, g, e)	≡ pasar(e)
aplicar(mover(d), g, e)	≡ $\langle \text{mover}(\Pi_1(e), d, \text{habitacion}(g)), \Pi_2(e), \text{False} \rangle$
invertir(e)	≡ $\langle \Pi_1(e), \text{invertir}(\Pi_2(e)), \Pi_3(e) \rangle$

$\text{inversa}(s) \equiv \text{if vacia}(s) \text{ then } \langle \rangle \text{ else } \text{inversa}(\text{fin}(s)) \bullet \text{invertir}(\text{prim}(s)) \text{ fi}$

Fin TAD

TAD DIRECCIÓN

géneros dir

exporta dir, generadores, observadores, ...

igualdad observacional

$$\left((\uparrow =_{\text{obs}} \uparrow) \wedge (\downarrow =_{\text{obs}} \downarrow) \wedge (\leftarrow =_{\text{obs}} \leftarrow) \wedge (\rightarrow =_{\text{obs}} \rightarrow) \wedge \neg(\uparrow =_{\text{obs}} \downarrow) \wedge \neg(\uparrow =_{\text{obs}} \rightarrow) \wedge \neg(\uparrow =_{\text{obs}} \leftarrow) \wedge \neg(\downarrow =_{\text{obs}} \rightarrow) \wedge \neg(\downarrow =_{\text{obs}} \leftarrow) \wedge \neg(\leftarrow =_{\text{obs}} \rightarrow) \right)$$

generadores

\uparrow	:	\rightarrow dir
\downarrow	:	\rightarrow dir
\leftarrow	:	\rightarrow dir
\rightarrow	:	\rightarrow dir

otras operaciones

invertir	:	dir	\rightarrow dir
girar	:	dir \times acc	\rightarrow dir

axiomas

invertir(\uparrow)	\equiv	\downarrow
invertir(\downarrow)	\equiv	\uparrow
invertir(\leftarrow)	\equiv	\rightarrow
invertir(\rightarrow)	\equiv	\leftarrow
girar(d, pasar)	\equiv	d
girar(d, mover(d'))	\equiv	d'
girar(d, disparar)	\equiv	d

Fin TAD

TAD JUEGO

géneros juego

exporta juego, generadores, observadores, ...

usa HABITACIÓN, NAT, POSICIÓN, ACCIÓN, CASILLERO, DIRECCIÓN, ...

igualdad observacional

$$(\forall j, j' : \text{juego}) \left(j =_{\text{obs}} j' \iff \left(\begin{array}{l} \text{habitacion}(j) =_{\text{obs}} \text{habitacion}(j') \wedge_L \\ \text{jugadores}(j) =_{\text{obs}} \text{jugadores}(j') \wedge_L \\ \text{fantasmas}(j) =_{\text{obs}} \text{fantasmas}(j') \wedge_L \\ \text{acciones}(j) =_{\text{obs}} \text{acciones}(j') \end{array} \right) \right)$$

generadores

nuevoJuego	:	hab $h \times \text{conj}(\text{jug}) \text{ } j s \times \text{fantasma } f$	\rightarrow juego
			$\{\neg \text{vacio}(js) \wedge \Pi_1(f) \in \text{posiciones}(h)\}$
step	:	jug $j \times \text{acc } a \times \text{juego } g$	\rightarrow juego
			$\{j \in \text{jugadores}(g) \wedge_L \text{jugadorVivo}(j, g) \wedge \neg \text{esPasar}(a)\}$
pasar	:	juego g	\rightarrow juego

observadores básicos

habitacion	:	juego	\rightarrow hab
fantasmas	:	juego	$\rightarrow \text{conj}(\text{fantasma})$
fantasmaEspecial	:	juego	$\rightarrow \text{fantasma}$
jugadores	:	juego	$\rightarrow \text{conj}(\text{jug})$
habitacion	:	juego	$\rightarrow \text{hab}$
acciones	:	jug $j \times \text{juego } g$	$\rightarrow \text{secu}(\text{evt}) \quad \{j \in \text{jugadores}(g)\}$

otras operaciones

ronda	: juego	\rightarrow Nat	
step	: juego	\rightarrow Nat	
jugadorVivo	: jug $j \times$ juego g	\rightarrow Bool	$\{j \in \text{jugadores}(g)\}$
fantasmaVivo	: fantasma $f \times$ juego g	\rightarrow Bool	$\{f \in \text{fantasmas}(g)\}$
posJugador	: jug $j \times$ juego g	\rightarrow pos	$\{j \in \text{jugadores}(g) \wedge_L \text{jugadorVivo}(j,g)\}$
posFantasma	: fantasma $f \times$ juego g	\rightarrow pos	$\{(f \in \text{fantasmas}(g)) \wedge_L \text{fantasmaVivo}(f,g)\}$
dirJugador	: jug $j \times$ juego g	\rightarrow dir	$\{j \in \text{jugadores}(g) \wedge_L \text{jugadorVivo}(j,g)\}$
dirFantasma	: fantasma $f \times$ juego g	\rightarrow dir	$\{f \in \text{fantasmas}(g) \wedge_L \text{fantasmaVivo}(f,g)\}$
termino	: juego	\rightarrow Bool	
maxCantDeAcciones	: conj(jug) $js \times$ juego g	\rightarrow Nat	$\{\neg \text{vacio}(js) \wedge js \subseteq \text{jugadores}(g)\}$
todosMuertos	: conj(jug) $js \times$ juego g	\rightarrow Bool	$\{js \subseteq \text{jugadores}(g)\}$
alcanceDisparosFantasmas	: conj(fantasma) $fs \times$ juego g	\rightarrow conj(pos)	$\{fs \subseteq \text{fantasmas}(g)\}$
disparando	: secu(evt) \times Nat	\rightarrow Bool	
recorrer	: secu(evt) \times Nat	\rightarrow evt	
posicionInicial	: jug $j \times$ juego g	\rightarrow evt	$\{j \in \text{jugadores}(g)\}$
localizarJugadores	: juego	\rightarrow dicc(jug, tupla(pos, dir))	

axiomas

habitacion(nuevoJuego(h,js,f))	\equiv h
habitacion(step(j,a,g))	\equiv habitacion(g)
habitacion(pasar(g))	\equiv habitacion(g)
jugadores(nuevoJuego(h,js,f))	\equiv js
jugadores(step(j,a,g))	\equiv jugadores(g)
jugadores(pasar(g))	\equiv jugadores(g)
fantasmas(nuevoJuego(h,js,f))	\equiv Ag(f, \emptyset)
fantasmas(step(j,a,g))	\equiv if a = disparar \wedge posFantasmas(fantasmaEspecial(g),g) \in alcanceDisparo(posJugador(j,g),dirJugador(j,g),habitacion(g)) then Ag(acciones(j,g) \circ aplicar(disparar, ult(acciones(j, g))), fantasmas(g)) else fantasmas(g) fi
fantasmas(pasar(g))	\equiv fantasmas(g)
fantasmaEspecial(nuevoJuego(h,js,f))	\equiv f
fantasmaEspecial(step(j,a,g))	\equiv if a = disparar \wedge posFantasmas(fantasmaEspecial(g),g) \in alcanceDisparo(posJugador(j,g),dirJugador(j,g),habitacion(g)) then acciones(j,g) \circ aplicar(disparar, ult(acciones(j, g))) else fantasmaEspecial(g) fi
fantasmaEspecial(pasar(g))	\equiv fantasmaEspecial(g)
acciones(j,nuevoJuego(h,js,f))	\equiv \langle posicionInicial(j, nuevoJuego(h, js, f)) \rangle

$\text{acciones}(j, \text{step}(j', a, g))$	\equiv if $a = \text{disparar} \wedge \text{posFantasma}(\text{fantasmaEspecial}(g), g) \in \text{alcanceDisparo}(\text{posJugador}(j, g), \text{dirJugador}(j, g), \text{habitacion}(g))$ then $\langle \text{posicionInicial}(j, \text{step}(j', a, g)) \rangle$ else $\text{acciones}(j, g) \circ$ if $j = j'$ then $\text{aplicar}(a, \text{ult}(\text{acciones}(j, g)), g)$ else $\text{aplicar}(\text{pasar}, \text{ult}(\text{acciones}(j, g), g))$ fi fi
$\text{acciones}(j, \text{pasar}(g))$	$\equiv \text{acciones}(j, g) \circ \text{aplicar}(\text{pasar}, \text{ult}(\text{acciones}(j, g)), g)$
$\text{ronda}(g)$	$\equiv \text{long}(\text{fantasmas}(g))$
$\text{step}(g)$	$\equiv \text{maxCantDeAcciones}(\text{jugadores}(g), g)$
$\text{maxCantDeAcciones}(c, g)$	\equiv if $\text{vacio}(\text{sinUno}(c))$ then $\text{long}(\text{acciones}(\text{dameUno}(g), g))$ else if $\text{long}(\text{acciones}(\text{dameUno}(g), g)) > \text{long}(\text{acciones}(\text{dameUno}(\text{sinUno}(c)), g))$ then $\text{maxCantDeAcciones}(\text{Ag}(\text{dameUno}(c), \text{sinUno}(\text{sinUno}(c))), g)$ else $\text{maxCantDeAcciones}(\text{sinUno}(c), g)$ fi fi
$\text{termino}(g)$	$\equiv \text{todosMuertos}((\text{jugadores}(g), g))$
$\text{todosMuertos}(js, g)$	\equiv if $\text{vacio}(js)$ then True else if $\text{jugadorVivo}(\text{dameUno}(js), g)$ then False else $\text{todosMuertos}(\text{sinUno}(js), g)$ fi fi
$\text{fantasmaVivo}(f, \text{nuevoJuego}(h, js, f))$	\equiv True
$\text{fantasmaVivo}(f, \text{step}(j, a, g))$	$\equiv \text{fantasmaVivo}(f, g) \wedge_L$ if $a = \text{disparar}$ then if $\text{posFantasma}(f, g) \notin \text{alcanceDisparo}(\text{posJugador}(j, g), \text{dirJugador}(j, g), \text{habitacion}(g))$ then False else True fi else True fi
$\text{fantasmaVivo}(f, \text{pasar}(g))$	$\equiv \text{fantasmaVivo}(f, g)$
$\text{dirFantasma}(f, g)$	$\equiv \Pi_2(\text{recorrer}(f, \text{step}(g)))$
$\text{posFantasma}(f, g)$	$\equiv \Pi_1(\text{recorrer}(f, \text{step}(g)))$
$\text{dirJugador}(j, g)$	$\equiv \Pi_2(\text{recorrer}(\text{acciones}(j, g), \text{step}(g)))$
$\text{posJugador}(j, g)$	$\equiv \Pi_1(\text{recorrer}(\text{acciones}(j, g), \text{step}(g)))$
$\text{jugadorVivo}(j, \text{nuevoJuego}(h, js, f))$	\equiv True

```

jugadorVivo(j,step(j',a,g))      ≡ if step(step(j',a,g)) = 0 then
                                   True
                                   else
                                     jugadorVivo(j,g)  $\wedge_L$ 
                                     if j=j'  $\wedge$  esMover(a) then
                                       mover(posJugador(j,g), direccion(a), habitacion(g))
                                        $\notin$  alcanceDisparosFantasmas(fantasmas(g),
                                       step(j',a,g))
                                     else
                                       posJugador(j,g)
                                        $\notin$  alcanceDisparosFantasmas(fantasmas(g),
                                       step(j',a,g))
                                     fi
                                   fi

jugadorVivo(j,pasar(g))          ≡ jugadorVivo(j,g)  $\wedge_L$ 
                                   posJugador(j,g)
                                    $\notin$  alcanceDisparosFantasmas(fantasmas(g), step(j',a,g))

alcanceDisparosFantasmas(fs, g)  ≡ if vacio(fs) then
                                    $\emptyset$ 
                                   else
                                     if fantasmaVivo(prim(fs),g)  $\wedge$ 
                                     disparando(prim(fs),step(g)) then
                                       Ag(alcanceDisparo(posFantasma(prim(fs),g),
                                       dirFantasma(prim(fs), g), habitacion(g)),
                                       alcanceDisparosFantasmas(fin(fs),g))
                                     else
                                       alcanceDisparosFantasmas(fin(fs), g)
                                     fi
                                   fi

disparando(es, n)                ≡  $\Pi_3$ (recorrer(es, n))
recorrer(es,r)                   ≡ if r < long(s) then
                                   es[r]
                                   else
                                     if r - long(s)  $\leq$  5 then
                                       aplicar(pasar, ult(r))
                                     else
                                       recorrer(inversa(es), r - long(s) - 5)
                                   fi
                                   fi

posicionInicial(j, g)            ≡  $\langle \Pi_1$ (obtener(j, localizarJugadores(g))),
                                    $\Pi_2$ (obtener(j, localizarJugadores(g))),
                                   False  $\rangle$ 

posicionIncial(j)                ≡ A definirse en diseño como requisito externo

```

Fin TAD