Trabajo práctico 2: Diseño

v1.4.1

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 párametros 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 ningun 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 del último `Paso` en $O((\#fv*m)^2)$. El conjunto de casilleros resultados no puede tener repetidos (bonus: se puede resolver en $O(\#fv * m)$).[1]

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.

[1]En versiones anteriores del enunciado se pedían las posiciones ocupadas por disparos de la última ronda. Debido a la confusión producida, ambas soluciones se consideraran correctas para el problema.

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.[2]
   - 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	**Menos claro**
"Cada clave del diccionario D debe ser una lista sin elementos repetidos." ✓ "sinRepetidos?(claves(D))" ✓	"No puede haber repetidos." (¿En qué estructura?).
"Ordenar la lista A usando mergesort." ✓ "A.mergesort()" ✓	"Ordenar los elementos." (¿Qué elementos? ¿Cómo se ordenan?).
"Para cada tupla (x, y) en el conjunto C { x.apilar(y) n++ }" ✓	"Miro las tuplas del conjunto, apilo la segunda componente en la primera y voy incrementando un contador." (Ambiguo y difícil de entender).

[2] 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 LaTeX 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 $\bullet = \bullet$: string $\times$ string $\rightarrow$ bool para compararlas por igualdad. Cada caracter es un número entre 0 y 255. $\bullet = \bullet$: operación $\times$ operación $\rightarrow$ 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

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

observadores básicos

tam : hab $\longrightarrow$ Nat

libre : pos c $\times$ hab h $\longrightarrow$ Bool {c $\in$ casilleros(h)}

generadores

nuevaHab : Nat n $\longrightarrow$ hab

ocupar : pos c $\times$ hab h $\longrightarrow$ hab
{c $\in$ casilleros(h) $\wedge_L$ libre(c,h) $\wedge$ alcanzan(libres(h)-c, libres(h)-c)}

otras operaciones

casilleros : hab $\longrightarrow$ conj(pos)

armarPares : Nat i_1 $\times$ Nat f_1 $\times$ Nat i_2 $\times$ Nat f_2 $\longrightarrow$ conj(pos)
$\{f_1 \geq i_1 \wedge f_2 \geq i_2\}$

libres : hab $\longrightarrow$ conj(pos)

libresEn : conj(pos) cs $\times$ hab h $\longrightarrow$ conj(pos)

alcanzan : conj(pos) cs_1 $\times$ conj(pos) cs_2 $\times$ hab h $\longrightarrow$ Bool
$\{cs_1 \subseteq casilleros(h) \wedge cs_2 \subseteq casilleros(h)\}$

explorar : conj(pos) cs_1 $\times$ pos cs_2 $\times$ hab h $\longrightarrow$ conj(pos)
$\{cs_1 \subseteq casilleros(h) \wedge cs_2 \subseteq casilleros(h)\}$

hayAdy↑ : pos c $\times$ hab h $\longrightarrow$ Bool $\{c \in casilleros(h)\}$

hayAdy↓ : pos c $\times$ hab h $\longrightarrow$ Bool $\{c \in casilleros(h)\}$

hayAdy← : pos c $\times$ hab h $\longrightarrow$ Bool $\{c \in casilleros(h)\}$

hayAdy→ : pos c $\times$ hab h $\longrightarrow$ Bool $\{c \in casilleros(h)\}$

ady↑ : pos c $\times$ hab h $\longrightarrow$ pos
$\{c \in casilleros(h) \wedge$ hayAdy↑(c,h)$\}$

ady↓ : pos c $\times$ hab h $\longrightarrow$ pos
$\{c \in casilleros(h) \wedge$ hayAdy↓(c,h)$\}$

ady← : pos c $\times$ hab h $\longrightarrow$ pos
$\{c \in casilleros(h) \wedge$ hayAdy←(c,h)$\}$

ady→ : pos c $\times$ hab h $\longrightarrow$ pos
$\{c \in casilleros(h) \wedge$ hayAdy→(c,h)$\}$

vecinos : pos c $\times$ hab h $\longrightarrow$ conj(pos)
{c $\in$ casilleros(h)}

vecinosLibres : pos c × hab h ⟶ conj(pos) {c ∈ casilleros(h)}
mover : pos p × dir × hab h ⟶ pos {p ∈ posiciones(h)}
alcanceDisparo : pos p × dir d × hab h ⟶ conj(pos) {p ∈ posiciones(h)}
alcanceDisparo↑ : pos p × hab h ⟶ conj(pos) {p ∈ posiciones(h)}
alcanceDisparo↓ : pos p × hab h ⟶ conj(pos) {p ∈ posiciones(h)}
alcanceDisparo← : pos p × hab h ⟶ conj(pos) {p ∈ posiciones(h)}
alcanceDisparo→ : pos p × hab h ⟶ conj(pos) {p ∈ posiciones(h)}

axiomas

tam(nuevaHab(n)) ≡ n
tam(ocupar(c, h)) ≡ tam(h)
libre(c, nuevaHab(n)) ≡ True
libre(c, ocupar(c',h)) ≡ **if** c = c' **then** False **else** libre(c,h) **fi**
casilleros(h) ≡ armarPares(0, tam(h)-1,0, tam(h)-1)
armarPares(i_1,f_1,i_2,f_2) ≡ Ag(⟨ i_1, i_2 ⟩, ∅) ∪
  if $i_2 = f_2$ **then**
    if $i_1 = f_1$ **then** ∅ **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) ≡ libresEn(casilleros(h),h)
libresEn(cs,h) ≡ **if** vacio(cs) **then**
    ∅
  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) ≡ **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) $\wedge_L$ ady↑(c,h) ∈ (vecinosLibres(c,h)-exp) **then**
        explorar(exp ∪ vecinosLibres(c,h), ady↑(c,h),h)
    else
        ∅
    fi ∪
    if hayAdy↑(c,h) $\wedge_L$ ady↑(c,h) ∈ (vecinosLibres(c,h)-exp) **then**
        explorar(exp ∪ vecinosLibres(c,h), ady↓(c,h),h)
    else
        ∅
    fi ∪
    if hayAdy←(c,h) $\wedge_L$ ady←(c,h) ∈ (vecinosLibres(c,h)-exp) **then**
        explorar(exp ∪ vecinosLibres(c,h), ady←(c,h),h)
    else
        ∅
    fi ∪
    if hayAdy→(c,h) $\wedge_L$ ady→(c,h) ∈ (vecinosLibres(c,h)-exp) **then**
        explorar(exp ∪ vecinosLibres(c,h), ady→(c,h),h)
    else
        ∅
    fi
fi

hayAdy↑(c,h) ≡ $\Pi_2(c) <$ tam(h) - 1
hayAdy↓(c,h) ≡ $\Pi_2(c)$ >0
hayAdy←(c,h) ≡ $\Pi_1(c)$ >0
hayAdy→(c,h) ≡ $\Pi_1(c)$ <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) $\wedge_L$ libre(ady↑(p,h),h) **then** ady↑(p,h) **else** p **fi**
mover(p,↓,h) ≡ **if** hayAdy↓(p,h) $\wedge_L$ libre(ady↓(p,h),h) **then** ady↓(p,h) **else** p **fi**
mover(p,←,h) ≡ **if** hayAdy←(p,h) $\wedge_L$ libre(ady←(p,h),h) **then** ady←(p,h) **else** p **fi**
mover(p,→,h) ≡ **if** hayAdy→(p,h) $\wedge_L$ 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
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

$$(\forall a, a' : \text{acc})) \left(a =_{\text{obs}} a' \iff \begin{pmatrix} \text{esPasar(a)} =_{\text{obs}} \text{esPasar(a')} \wedge_{\text{L}} \\ \text{esDisparar(a)} =_{\text{obs}} \text{esDisarar(a')} \wedge_{\text{L}} \\ \text{esDireccion(a)} =_{\text{obs}} \text{esDireccion(a')} \wedge_{\text{L}} \\ \text{esDireccion(a)} \Rightarrow_L (\text{direccion(a)} =_{\text{obs}} \text{direccion(a')}) \end{pmatrix} \right)$$

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
direccion(mover(d))	≡	d
pasar(e)	≡	⟨Π_1(e), Π_2(e), False ⟩
aplicar(disparar, g, e)	≡	⟨Π_1(e), Π_2(e), True ⟩
aplicar(pasar, g, e)	≡	pasar(e)
aplicar(mover(d), g, e)	≡	⟨mover(Π_1(e), d, habitacion(g)), d, False ⟩
invertir(e)	≡	⟨Π_1(e), invertir(Π_2(e)), Π_3(e) ⟩

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

Fin TAD

TAD Dirección

géneros dir

exporta dir, generadores, observadores, ...

igualdad observacional

$$\begin{pmatrix}(\uparrow =_{obs} \uparrow) \land (\downarrow =_{obs} \downarrow) \land (\leftarrow =_{obs} \leftarrow) \land (\rightarrow =_{obs} \rightarrow) \land \neg(\uparrow =_{obs} \downarrow) \land \neg(\uparrow =_{obs} \rightarrow) \land \neg(\uparrow =_{obs} \leftarrow) \land \\ \neg(\downarrow =_{obs} \rightarrow) \land \neg(\downarrow =_{obs} \leftarrow) \land \neg(\leftarrow =_{obs} \rightarrow)\end{pmatrix}$$

generadores

$\uparrow$	:		$\longrightarrow$ dir
$\downarrow$	:		$\longrightarrow$ dir
$\leftarrow$	:		$\longrightarrow$ dir
$\rightarrow$	:		$\longrightarrow$ dir

otras operaciones

invertir	:	dir	$\longrightarrow$ dir
girar	:	dir $\times$ acc	$\longrightarrow$ 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

$$(\forall j, j' : \text{juego}) \left(j =_{obs} j' \iff \begin{pmatrix}\text{habitacion(j)} =_{obs} \text{habitacion(j')} \land_L \\ \text{jugadores(j)} =_{obs} \text{jugadores(j')} \land_L \\ \text{fantasmas(j)} =_{obs} \text{fantasmas(j')} \land_L \\ \text{fantasmaEspecial(j)} =_{obs} \text{fantasmaEspecial(j')} \land_L \\ (\forall c\text{: jug})\ (c \in \text{jugadores(j)} \Rightarrow \text{acciones(c, j)} =_{obs} \text{acciones(c, j'))}\end{pmatrix}\right)$$

generadores

nuevoJuego	:	hab h $\times$ conj(jug) js $\times$ fantasma f	$\longrightarrow$ juego
			$\{\neg$vacio(js) $\land \Pi_1$(f) $\in$ posiciones(h)$\}$
step	:	jug j $\times$ acc a $\times$ juego g	$\longrightarrow$ juego
			$\{$j $\in$ jugadores(g) $\land_L$ jugadorVivo(j,g)i $\land$ $\neg$esPasar(a)$\}$
pasar	:	juego g	$\longrightarrow$ juego

observadores básicos

habitacion	:	juego	$\longrightarrow$ hab
fantasmas	:	juego	$\longrightarrow$ conj(fantasma)
fantasmaEspecial	:	juego	$\longrightarrow$ fantasma
jugadores	:	juego	$\longrightarrow$ conj(jug)

acciones : jug j × juego g ⟶ secu(evt) {j ∈ jugadores(g)}

otras operaciones

ronda : juego ⟶ Nat
step : juego ⟶ Nat
jugadorVivo : jug j × juego g ⟶ Bool {j ∈ jugadores(g)}
fantasmaVivo : fantasma f × juego g ⟶ Bool {f ∈ fantasmas(g)}
posJugador : jug j × juego g ⟶ pos {j ∈ jugadores(g) $\wedge_L$ jugadorVivo(j,g)}
posFantasma : fantasma f × juego g ⟶ pos {(f ∈ fantasmas(g)) $\wedge_L$ fantasmaVivo(f,g)}
dirJugador : jug j × juego g ⟶ dir {j ∈ jugadores(g) $\wedge_L$ jugadorVivo(j,g)}
dirFantasma : fantasma f × juego g ⟶ dir {f ∈ fantasmas(g) $\wedge_L$ fantasmaVivo(f,g)}
termino : juego ⟶ Bool
maxCantDeAcciones : conj(jug) js × juego g ⟶ Nat {¬ vacio(js) ∧ js ⊆ jugadores(g)}
todosMuertos : conj(jug) js × juego g ⟶ Bool {js ⊆ jugadores(g)}
alcanceDisparosFantasmas : conj(fantasma) fs × juego g ⟶ conj(pos) {fs ⊆ fantasmas(g)}
disparando : secu(evt) × Nat ⟶ Bool
recorrer : secu(evt) × Nat ⟶ evt
posicionInicial : jug j × juego g ⟶ evt {j ∈ jugadores(g)}
localizarJugadores : juego ⟶ dicc(jug, tupla(pos, dir))

axiomas

habitacion(nuevoJuego(h,js,f)) ≡ h
habitacion(step(j,a,g)) ≡ habitacion(g)
habitacion(pasar(g)) ≡ habitacion(g)
jugadores(nuevoJuego(h,js,f)) ≡ js
jugadores(step(j,a,g)) ≡ jugadores(g)
jugadores(pasar(g)) ≡ jugadores(g)
fantasmas(nuevoJuego(h,js,f)) ≡ Ag(f, ∅)
fantasmas(step(j,a,g)) ≡ **if** a = disparar ∧ posFantasmas(fantasmaEspecial(g),g) ∈ alcanceDisparo(posJugador(j,g),dirJugador(j,g),habitacion(g))
then
Ag(acciones(j,g) ∘ aplicar(disparar, ult(acciones(j, g))), fantasmas(g))
else
fantasmas(g)
fi
fantasmas(pasar(g)) ≡ fantasmas(g)
fantasmaEspecial(nuevoJuego(h,js,f)) ≡ f
fantasmaEspecial(step(j,a,g)) ≡ **if** a = disparar ∧ posFantasmas(fantasmaEspecial(g),g) ∈ alcanceDisparo(posJugador(j,g),dirJugador(j,g),habitacion(g))
then
acciones(j,g) ∘ aplicar(disparar, ult(acciones(j, g)))
else
fantasmaEspecial(g)
fi
fantasmaEspecial(pasar(g)) ≡ fantasmaEspecial(g)
acciones(j,nuevoJuego(h,js,f)) ≡ ⟨ posicionInicial(j, nuevoJuego(h, js, f)) ⟩

acciones(j,step(j',a,g)) $\equiv$ **if** a = disparar $\land$ posFantasma(fantasmaEspecial(g), g) $\in$ alcanceDisparo(posJugador(j,g),dirJugador(j,g),habitacion(g))
then
$\langle$ posicionInicial(j, step(j', a, g)) $\rangle$
else
acciones(j,g) $\circ$ **if** j = j' **then**
aplicar(a, ult(acciones(j, g)), g)
else
aplicar(pasar, ult(acciones(j, g), g))
fi
fi

acciones(j,pasar(g)) $\equiv$ acciones(j, g) $\circ$ aplicar(pasar,ult(acciones(j, g)), g)

ronda(g) $\equiv$ long(fantasmas(g))

step(g) $\equiv$ maxCantDeAcciones(jugadores(g),g)

maxCantDeAcciones(c,g) $\equiv$ **if** vacio(sinUno(c)) **then**
long(acciones(dameUno(g),g))
else
if long(acciones(dameUno(g),g)) $>$ long(acciones(dameUno(sinUno(c))),g) **then**
maxCantDeAcciones(Ag(dameUno(c),sinUno(sinUno(c)),g)
else
maxCantDeAcciones(sinUno(c),g)
fi
fi

termino(g) $\equiv$ todosMuertos((jugadores(g),g))

todosMuertos(js,g) $\equiv$ **if** vacio(js) **then**
True
else
if jugadorVivo(dameUno(js),g) **then**
False
else
todosMuertos(sinUno(js),g)
fi
fi

fantasmaVivo(f,nuevoJuego(h,js,f)) $\equiv$ True

fantasmaVivo(f,step(j,a,g)) $\equiv$ fantasmaVivo(f,g) $\land_L$
if a = disparar **then**
if posFantasma(f,g) $\notin$ alcanceDisparo(posJugador(j,g),dirJugador(j,g),habitacion(g))
then
False
else
True
fi
else
True
fi

fantasmaVivo(f,pasar(g)) $\equiv$ fantasmaVivo(f, g)

dirFantasma(f,g) $\equiv$ Π_2(recorrer(f, step(g)))

posFantasma(f,g) $\equiv$ Π_1(recorrer(f, step(g)))

dirJugador(j,g) $\equiv$ Π_2(recorrer(acciones(j,g), step(g)))

posJugador(j,g) $\equiv$ Π_1(recorrer(acciones(j,g), step(g)))

jugadorVivo(j,nuevoJuego(h,js,f)) $\equiv$ True

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

alcanceDisparosFantasmas(fs, g) $\equiv$ **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) $\equiv$ Π_3(recorrer(es, n))

recorrer(es,r) $\equiv$ **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) $\equiv$ $\langle$ Π_1(obtener(j, localizarJugadores(g))), Π_2(obtener(j, localizarJugadores(g))), False $\rangle$

posicionIncial(j) $\equiv$ A definirse en diseño como requisito externo

Fin TAD