Trabajo práctico 3: **Pacalgo2**

Normativa

Límite de entrega: Domingo 30 de Mayo, 23:59hs.

Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia. (`http://campus.exactas.uba.ar`)

Versión: 1.0 del 24 de mayo de 2021

El objetivo de este TP es diseñar módulos destinados a implementar el juego Pacalgo2 que se describió en los TPs anteriores. En la siguiente sección pueden encontrar la **especificación formal** del juego provista por la cátedra. El diseño propuesto debe cumplir con las siguientes **complejidades temporales en peor caso**, donde: J representa la cantidad de jugadores en el ranking, c representa la cantidad de chocolates en el mapa, $|J|$ representa el largo del nombre de jugador más extenso en el ranking, F representa la cantidad de fantasmas y A y L es el alto y largo del mapa.

1. Iniciar una partida debe ser $O(c)$.
2. Realizar un movimiento que termina la partida (ya sea ganada o perdida), debe ser $O(|J|)$.
3. El resto de las acciones dentro de una partida (moverse, comer chocolates) deben ser $O(1)$.

La entrega consistirá de un único documento digital con el diseño completo de todos los módulos. Se debe diseñar el módulo principal (`Pacalgo2`) 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. Además, en el caso de usar implementaciones de abstracciones vistos en la materia (e.j.: Trie para Diccionario, AVL para Conjunto, Heap para cola de prioridad), es suficiente con definir un módulo con una interfaz plausible para la abstracción con las complejidades vistas en la teórica. Es decir, no es necesario aclarar estructura de representación, invariante de representación, función de abstracción o algoritmos.

Todos los módulos diseñados deben contar con las siguientes partes.

1. **Interfaz.**
   a) *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.
   b) *Signatura.* Listado de todas las funciones públicas que provee el módulo. La signatura se debe escribir con la notación de módulos de la materia, por ejemplo, apilar(**in**/**out** pila : Pila, **in** x : Elemento).
   c) *Contrato.* Precondición y postcondición de todas las funciones públicas. Las pre y postcondiciones de las funciones de la interfaz deben estar expresadas **formalmente** en lógica de primer orden.[1] Las pre y postcondiciones deben usar los TADs provistos en el apartado **Especificación de la cátedra** más abajo en este documento, y **no** a la especificación escrita por ustedes en el TP2.
   d) *Complejidades.* Complejidades de todas las funciones públicas, cuando corresponda.
   e) *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.**
   a) *Representación ("se representa con ...").* Módulo con el que se representan las instancias del módulo actual.
   b) *Invariante de representación.* Puede estar expresado en lenguaje natural o formal.
   c) *Función de abstracción.* Puede estar expresada en lenguaje natural o formal. La función de abstracción debe referirse a los TADs provistos en el apartado **Especificación de la cátedra** más abajo en este documento, y **no** a la especificación escrita por ustedes en el TP2.
   d) *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.

[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.

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 uso de tablas de hash.

Recomendamos **no** usar tablas de *hash* como parte de la solución a este TP. El motivo es que, si bien las tablas de *hash* proveen buenas garantías de complejidad *en caso promedio*—asumiendo ciertas propiedades sobre la función de *hash* y condiciones de buena distribución de la entrada—, no proveen en cambio buenas garantías de complejidad *en peor caso*. (En términos asintóticos, una tabla de *hash* se comporta en peor caso tan mal como una lista enlazada).

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).

Especificación de la cátedra

TAD Coordenada **es** Tupla⟨nat, nat⟩
TAD Direccion **es** ENUM{ARRIBA, ABAJO, IZQUIERDA, DERECHA}
TAD Jugador **es** String
TAD Ranking **es** dicc(jugador, nat)

TAD Mapa

géneros mapa

observadores básicos

largo : mapa ⟶ nat
alto : mapa ⟶ nat
inicio : mapa ⟶ coordenada
llegada : mapa ⟶ coordenada
paredes : mapa ⟶ conj(coordenada)
fantasmas : mapa ⟶ conj(coordenada)
chocolates : mapa ⟶ conj(coordenada)

generadores

nuevoMapa : nat *largo* × nat *alto* × coordenada *inicio* × coordenada *llegada* × conj(coordenada) *paredes* × conj(coordenada) *fantasmas* × conj(coordenada) *chocolates* ⟶ mapa

$$\left\{\begin{array}{l}\text{inicio} \neq \text{llegada} \land \text{todosEnRango(paredes} \cup \text{fantasmas} \cup \text{chocolates} \cup \{\text{inicio, llegada}\},\\ \text{largo, alto)} \land \{\text{inicio, llegada}\} \cap (\text{fantasmas} \cup \text{paredes}) = \emptyset \land \text{disjuntosDeAPares(paredes,}\\ \text{fantasmas, chocolates)}\end{array}\right\}$$

otras operaciones

distancia	: coordenada $pos1$ × coordenada $pos2$	$\longrightarrow$	nat
distConFantasmaMasCercano	: conj(coordenada) $fantasmas$ × coordenada pos	$\longrightarrow$	nat {¬ vacío?(fantasmas)}
enRango	: coordenada pos × nat $largo$ × nat $alto$	$\longrightarrow$	bool
todasEnRango	: conj(coordenada) $posiciones$ × nat $largo$ × nat $alto$	$\longrightarrow$	bool
disjuntosDeAPares	: conj(α) × conj(α) × conj(α)	$\longrightarrow$	bool

axiomas

largo(nuevoMapa(l,a,ini,fin,p,f,c)) $\equiv$ l
alto(nuevoMapa(l,a,ini,fin,p,f,c)) $\equiv$ a
inicio(nuevoMapa(l,a,ini,fin,p,f,c)) $\equiv$ ini
llegada(nuevoMapa(l,a,ini,fin,p,f,c)) $\equiv$ fin
paredes(nuevoMapa(l,a,ini,fin,p,f,c)) $\equiv$ p
fantasmas(nuevoMapa(l,a,ini,fin,p,f,c)) $\equiv$ f
chocolates(nuevoMapa(l,a,ini,fin,p,f,c)) $\equiv$ c
distancia(pos1, pos2) $\equiv$ $|+pos1_1 - +pos2_1| + |+pos1_2 - +pos2_2|$
distConFantasmaMasCercano(fantasmas, pos) $\equiv$ **if** $\#fantasmas = 1$ **then**
distancia(pos, dameUno(fantasmas))
else
mín(distancia(pos, dameUno(fantasmas)), distConFantasmaMasCercano(sinUno(fantasmas), pos))
fi
enRango(pos,largo,alto) $\equiv$ $pos_1 > 0 \land pos_1 \leq largo \land pos_2 > 0 \land pos_2 \leq alto$
todosEnRango(posiciones,largo,alto) $\equiv$ vacío?(posiciones) $\lor$ (enRango(dameUno(posiciones), largo, alto) $\land$ todosEnRango(sinUno(posiciones), largo, alto))
disjuntosDeAPares(a, b, c) $\equiv$ #(a) + #(b) + #(c) = #(a $\cup$ b $\cup$ c)

Fin TAD

TAD Partida

géneros partida

observadores básicos

mapa : partida $\longrightarrow$ mapa
jugador : partida $\longrightarrow$ coordenada
chocolates : partida $\longrightarrow$ conj(coordenada)
cantMov : partida $\longrightarrow$ nat
inmunidad : partida $\longrightarrow$ nat

generadores

nuevaPartida : mapa m	$\longrightarrow$	partida
mover : partida p × direccion d	$\longrightarrow$	partida {¬ganó?(p) ∧ ¬perdió?(p)}

otras operaciones

ganó? : partida $\longrightarrow$ bool
perdió? : partida $\longrightarrow$ bool
siguienteMovimiento : partida × direccion $\longrightarrow$ coordenada
posMovimiento : coordenada × direccion $\longrightarrow$ coordenada
restringirMovimiento : partida × coordenada $\longrightarrow$ coordenada

axiomas

mapa(nuevaPartida(m)) $\equiv$ m

mapa(mover(p, pos)) ≡ mapa(p)
jugador(nuevaPartida(m)) ≡ inicio(m)
jugador(mover(p, d)) ≡ siguienteMovimiento(p, d)
cantMov(nuevaPartida(m)) ≡ 0
cantMov(mover(p,d)) ≡ cantMov(p) + β(siguienteMovimiento(p, d) ≠ jugador(p))
chocolates(nuevaPartida(m)) ≡ chocolates(m) - inicio(m)
chocolates(mover(p,d)) ≡ chocolates(p) - siguienteMovimiento(p, d)
inmunidad(nuevaPartida(m)) ≡ **if** inicio(m) ∈ chocolates(m) **then** 10 **else** 0 **fi**
inmunidad(mover(p,d)) ≡ **if** siguienteMovimiento(p, d) ∈ chocolates(p) **then**
10
else
max(0, inmunidad(p) - β(siguienteMovimiento(p, d) ≠ jugador(p)))
fi
ganó?(p) ≡ jugador(p) = llegada(mapa(p))
perdió?(p) ≡ ¬ganó?(p) ∧ distConFantasmaMasCercano(fantasmas(mapa(p)), jugador(p))≤ 3 ∧ inmunidad(p) = 0
siguienteMovimiento(p, d) ≡ **if** posMovimiento(jugador(p), d) ∈ paredes(mapa(p)) **then**
jugador(p)
else
restringirMovimiento(p, posMovimiento(jugador(p), d))
fi
posMovimiento(c, d) ≡ ⟨ c_1 + β(d = DERECHA) - β(d = IZQUIERDA),
c_2 + β(d = ARRIBA) - β(d = ABAJO) ⟩
restringirMovimiento(p, c) ≡ ⟨ max(0, min(largo(mapa(p)) - 1, c_1)),
max(0, min(alto(mapa(p)) - 1, c_2)) ⟩

Fin TAD

TAD Fichin

géneros fichin

observadores básicos

mapa : fichin ⟶ mapa
alguienJugando? : fichin ⟶ bool
jugadorActual : fichin f ⟶ jugador {alguienJugando?(f)}
partidaActual : fichin f ⟶ partida {alguienJugando?(f)}
ranking : fichin ⟶ ranking

generadores

nuevoFichin : mapa ⟶ fichin
nuevaPartida : fichin f × jugador ⟶ fichin {¬alguienJugando?(f)}
mover : fichin f × direccion ⟶ fichin {alguienJugando?(f)}

otras operaciones

objetivo : fichin f ⟶ tupla⟨jugador, nat⟩ {alguienJugando?(f) ∧ definido?(jugadorActual(f), ranking(f))}
oponente : fichin f ⟶ jugador {alguienJugando?(f) ∧ definido?(jugadorActual(f), ranking(f))}
oponentes : fichin f ⟶ conj(jugador) {alguienJugando?(f) ∧ definido?(jugadorActual(f), ranking(f))}
mejoresQue : ranking r × conj(jugador) ch × nat ⟶ conj(jugador) {cj ⊆ claves(r)}
peoresJugadores : ranking r × conj(jugador) cj ⟶ conj(jugador) {cj ⊆ claves(r)∧ ¬∅?(cj)}
jugadoresConPuntaje : ranking r × conj(jugador) cj × nat ⟶ conj(jugador) {cj ⊆ claves(r)∧ ¬∅?(cj)}
peorPuntaje : ranking r × conj(jugador) cj ⟶ nat {cj ⊆ claves(r) ∧ ¬∅?(cj)}

axiomas

mapa(nuevoFichin(m)) ≡ m

mapa(nuevaPartida(f, j)) ≡ mapa(f)
mapa(mover(f, d)) ≡ mapa(f)
alguienJugando?(nuevoFichin(m)) ≡ false
alguienJugando?(nuevaPartida(f, j)) ≡ true
alguienJugando?(mover(f, d)) ≡ ¬ (ganó?(mover(partidaActual(f), d)) ∨ perdió?(mover(partidaActual(f), d))
jugadorActual(nuevaPartida(f, j)) ≡ j
jugadorActual(mover(f, d)) ≡ jugadorActual(f)
partidaActual(nuevaPartida(f, j)) ≡ nuevaPartida(mapa(f))
partidaActual(mover(f, d)) ≡ mover(partidaActual(f), m)
ranking(nuevoFichin(m)) ≡ vacío
ranking(nuevaPartida(f, j)) ≡ ranking(f)
ranking(mover(f, d)) ≡ **if** ganó?(mover(partidaActual(f), d)) **then**
    if def?(jugadorActual(f), ranking(f)) **then**
        definir(jugadorActual(f),
        min(obtener(jugadorActual(f), ranking(f)),
        cantMov(mover(partidaActual(f), d)))
)
    else
        definir(jugadorActual(f),
        cantMov(mover(partidaActual(f), d)))
    fi
else
    ranking(f)
fi
objetivo(f) ≡ ⟨oponente(f), obtener(ranking(f), oponente(f))⟩
oponente(f) ≡ **if** #oponentes(f) = 0 **then**
    jugadorActual(f)
else
    dameUno(oponentes(f))
fi
oponentes(f) ≡ **if** ∅?(mejoresQue(ranking(f), claves(ranking(f)), obtener(ranking(f), jugadorActual(f))) **then**
    ∅
else
    peoresJugadores(ranking(f), mejoresQue(ranking(f), claves(ranking(f)), obtener(ranking(f), jugadorActual(f))))
fi
mejoresQue(r, cj, n) ≡ **if** vacio?(cj) **then**
    ∅
else
    mejoresQue(r, sinUno(cj), n) ∪
    if obtener(dameUno(cj), r) > n **then**
        {dameUno(cj)}
    else
        ∅
    fi
fi
peoresJugadores(r, cj) ≡ jugadoresConPuntaje(r, cj, peorPuntaje(r, cj))
jugadoresConPuntaje(r, cj, n) ≡ **if** vacio?(cj) **then**
    ∅
else
    jugadoresConPuntaje(sinUno(cj), n) ∪
    if obtener(dameUno(cj), r) = n **then**
        {dameUno(cj)}
    else
        ∅
    fi
fi

```
peorPuntaje(r, cj)  ≡ if #cj = 1 then
                        obtener(dameUno(cj), r)
                      else
                        if obtener(dameUno(cj), r) < peorPuntaje(sinUno(cj), r)
                        then
                          obtener(dameUno(cj), r)
                        else
                          peorPuntaje(sinUno(cj), r)
                        fi
                      fi
```

Fin TAD

Entrega

Para la entrega deben hacer `commit` y `push` de un único documento digital en formato `pdf` en el repositorio **grupal** en el directorio `tpg3/`. El documento debe incluir el diseño completo del enunciado incluyendo todos los módulos, cada uno con su interfaz, estructuras de representación, invariante de representación, función de abstracción, implementación de los algoritmos y descripción de los servicios usados. Se recomienda el uso de los paquetes de LaTeX de la cátedra para lograr una mejor visualización del informe.

Rubricas

Agregamos a continuación rúbricas que exponen qué se espera de la entrega. Las mismas presentan una serie criterios con los que se evaluarán las producciones entregadas. En términos generales, se considera que entregas con soluciones que solo logren los criterios parcialmente deberán ser reentregados con correcciones en estos aspectos en particular.

Por ser criterios generales, pueden no cubrir todos los detalles relacionados con este enunciado. No obstante buscamos que sean lo más completas posibles. Esperamos que las mismas les sirvan de orientación para la resolución del TP.

	Logra Totalmente	Logra	Logra Parcialmente	No Logra
Abstraccion funcional (o modularización)	Los Módulos tienen una responsabilidad adecuada (ni mucha ni poca). Cada módulo tiene una semántica clara. Las partes del problema se separan en módulos de forma de hacer cada módulo comprensible por si mismo. Siempre que es posible, se delega responsabilidad a módulos básicos.	Algún módulo asume demasiadas responsabilidades y es difícil entenderlo como una unidad. Dividirlo ayudaría a una mejor comprensión de la solución. En algunos casos, no se usan módulos basicos que ayudan a resolver el problema.	Varios módulos asume demasiadas responsabilidades y es difícil entenderlos como una unidad. Los módulos básicos se usan poco o incorrectamente. E.j.: secu de tuplas de diccionario lineal, secu en lugar de conjunto lineal.	Se utiliza un único módulo que resuelve todo el problema, por lo que su comportamiento y representación se hace muy difícil de entender. Prácticamente no se usan módulos básicos (todos arreglos/vectores).
Funciones de la interfaz pública	La inferfaz de cada módulo cuenta con suficientes funciones para hacerlo útil para el problema que resuelve. Los nombres de las funciones ayudan a entender la funcionalidad del módulo.	En algún módulo, falta alguna función para poder operar todas funcionalidades del TAD.	En algún módulo, falta varias funciónes para poder operar todas funcionalidades del TAD. Los nombres de las funciones son muy poco claros.	En algún módulo, faltan muchas funciones para poder hacer uso de la funcionalidad del TAD. El módulo se utiliza directamente desde su representación.
Contrato de las funciones de la interfaz pública	Todas las funciones de la interfaz pública cuentan con pre y post-condiciones correctas. Las descripciones son entendibles y representativas. La complejidad está correctamente reportada. Se reporta aliasing donde corresponde.	Algunos pre y post estan incorrectos/incompletos. Algunas descripciones están no se entienden. Algunas complejidades faltan.	Algunos pre y post están expresados sobre la representación. Faltan descripciones. Las complejidades faltan, no se entienden o no dependen de la entrada.	Todas las pre y post condiciones están sobre la estructura de representación o faltan. No hay descripciones ni complejidades.
Funciones privadas	Todas las funciones auxiliares utlizadas estan en la interfaz, cuentan con aridad y descripción. Las funciones cuya complejidad es relevante para la interfaz pública tienen complejidad declarada.	Faltan algunas funciones.	Faltan algunas funciones y carecen de descripción.	Ninguna función auxilar se declara en la intefaz.
Invariante de representación	Todas las restricciones se encuentran expresadas y se pueden entender (ya sea en lenguaje natural y/o formal).	Faltan algunas restriccones pero las definidas se pueden entender (ya sea en lenguaje natural y/o formal).	Faltan algunas restriccones y son difíciles de entender.	Faltan muchas restricciones, se lo asume incorrectamente trivial (True), se expresan sobre el TAD en lugar de la estructura de representación o es el Abs.
Función de abstracción	Explica (en lenguaje natural y/o formal) cómo se definen todos los observadores a partir de la estructura de representación.	Explica cómo se definen los observadores pero tiene casos donde se inhabilita que no estan limitados en el Rep y no se resuelven en el Abs.	No explica cómo se definen todos los observadores o es entendible.	Falta o es totalmente inentendible.
Implementación	Las implementación de las funciones públicas mantienen el rep al finalizar, cumplen la postcondición y puede entenderse su funcionamiento. La complejidad está bien justificada.	La implementación no siempre es fácil de entender o la complejidad no está correctamente explicada.	Hay implementaciones que no cumplen la postcondición, no mantienen el rep o no cumplen la complejidad establecida.	Faltan implementaciones o son muy difíciles de leer.
Correctitud y Complejidad	Los módulos resuelven completamente la especificación cumpliendo todas las restricciones de complejidad.	Los módulos tienen errores que no simplifican el enunciado. Se cumplen todas las restricciones de complejidad.	Los módulos tienen errores que simplifican levemente el enunciado. Se cumplen todas las restricciones de complejidad.	Faltan resolver restricciones de complejidad del enunciado. Numerosas secciones de la especificación sin resolver en el diseño.

Algoritmos

Algoritmos II

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

TPIII

Iron Managers

Integrante	LU	Correo electrónico
Mariano Oca	[redacted]	[redacted]
Ilicic Tobías	[redacted]	[redacted]
Teo Kohan	[redacted]	[redacted]
Marcos Filipich	[redacted]	[redacted]

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1 Coordenada

1.1 Interfaz

1.1.1 Tipos

se explica con: Nat, Int, Tupla.
géneros: coord.

1.1.2 Operaciones básicas de coordenada

$\langle \bullet, \bullet \rangle$ (**in** a: Nat, **in** b: Nat) $\rightarrow res$: Coord
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \langle a,\ b \rangle\}$
Complejidad: $\Theta(1)$
Descripción: Crea una coordenada.
Aliasing: No presenta aspectos de aliasing.

Copy (**in** v: Coord) $\rightarrow res$: Coord
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} v\}$
Complejidad: $\Theta(1)$
Descripción: Copia una coordenada.
Aliasing: No presenta aspectos de aliasing.

$\bullet + \bullet$ (**in** v: Coord, **in** w: Coord) $\rightarrow res$: Coord
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \langle \pi_1(v) + \pi_1(w),\ \pi_2(v) + \pi_2(w) \rangle\}$
Complejidad: $\Theta(1)$
Descripción: Devuelve la suma de dos coordenadas.
Aliasing: No presenta aspectos de aliasing.

$\bullet - \bullet$ (**in** v: Coord, **in** w: Coord) $\rightarrow res$: Coord
Pre $\equiv$ $\{\pi_1(v) \geq \pi_1(w) \land \pi_2(v) \geq \pi_2(w)\}$
Post $\equiv$ $\{res =_{\text{obs}} \langle \pi_1(v) - \pi_1(w),\ \pi_2(v) - \pi_2(w) \rangle\}$
Complejidad: $\Theta(1)$
Descripción: Devuelve la resta de dos coordenadas.
Aliasing: No presenta aspectos de aliasing.

$\bullet < \bullet$ (**in** v: Coord, **in** w: Coord) $\rightarrow res$: Bool
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \pi_1(v) < \pi_1(w) \land \pi_2(v) < \pi_2(w)\}$
Complejidad: $\Theta(1)$
Descripción: Devuelve verdadero si solo si ambas coordenadas de v son menores a las de w o lo que es equivalente, si v está 'dentro' de w.
Aliasing: No presenta aspectos de aliasing.

1.2 Implementación

1.2.1 Representación

COORDENADA **se representa con** estr **donde** estr **es** tupla$\langle x : \text{nat},\ y : \text{nat}\rangle$

1.2.2 Invariante

Rep : estr $\to$ boolean

Rep(e) $\equiv$ true

1.2.3 Abstracción

Abs(e): estr $\to$ coord ———— {Rep(e)}
($\forall e$: estr) Abs(e) $=_{\text{obs}}$ c : coord | c $=_{\text{obs}}$ $\langle e.x,\ e.y\rangle$

1.2.4 Algoritmos

---

i$\langle\bullet,\ \bullet\rangle$(**in** a : `nat`, **in** b : `nat`) $\to$ res : coord

1: $res.x \leftarrow a$ ▷ $\Theta(1)$
2: $res.y \leftarrow b$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$

---

iCopy(**in** v : `coord`) $\to$ res : coord

1: $res \leftarrow v$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$

---

i$\bullet + \bullet$(**in** v : `coord`, **in** w : `coord`) $\to$ res : coord

1: $res.x \leftarrow v.x + w.x$ ▷ $\Theta(1)$
2: $res.y \leftarrow v.y + w.y$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$

---

i$\bullet - \bullet$(**in** v : `coord`, **in** w : `coord`) $\to$ res : coord

1: $res.x \leftarrow v.x - w.x$ ▷ $\Theta(1)$
2: $res.y \leftarrow v.y - w.y$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$

---

i$\bullet < \bullet$(**in** v : `coord`, **in** w : `coord`) $\to$ res : coord

1: $res \leftarrow v.x < w.x$ **and** $v.y < w.y$ ▷ $\Theta(1)$

Complejidad: $\Theta(1)$

2 Mapa

2.1 Renombres

TAD Coordenada **es** ⟨Nat, Nat⟩
TAD Elemento **es** ENUM{PARED, CHOCOLATE, VACIO, FANTASMA}

2.2 Interfaz

2.2.1 Tipos

se explica con: Bool, Nat, Coordenada, Elemento, Conjunto(α), Mapa.
géneros: mapa.

2.2.2 Operaciones básicas de mapa

NuevoMapa $\begin{pmatrix}\textbf{in}\text{ largo, alto: Nat, }\textbf{in}\text{ inicio, llegada: Coord,}\\ \textbf{in}\text{ paredes, fantasmas, chocolates: Conj(Coord)}\end{pmatrix}$ $\rightarrow res$: Mapa

Pre $\equiv$ $\left\{\begin{array}{l}\text{inicio} \neq \text{llegada} \wedge \text{disjuntosDeAPares(paredes,fantasmas,chocolates)}\\ \text{todosEnRango}(\text{paredes} \cup \text{fantasmas} \cup \text{chocolates} \cup \{\text{inicio, llegada}\}, \text{largo}, \text{alto}) \wedge\\ \{\text{inicio, llegada}\} \cap (\text{fantasmas} \cup \text{paredes}) = \varnothing\end{array}\right\}$

Post $\equiv$ $\{res =_{\text{obs}} \text{nuevoMapa(largo, alto, inicio, llegada, paredes, fantasmas, chocolates)}\}$
Complejidad: $\Theta(|\text{paredes}| + F + c)$, $O(L \cdot A)$
Descripción: Crea un mapa de dimensión alto por largo, poblándolo con los datos pasados.
Aliasing: No presenta aspectos de aliasing.

Largo (**in** mapa: Mapa) $\rightarrow res$: Nat
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{largo(mapa)}\}$
Complejidad: $\Theta(1)$
Descripción: Devuelve el largo del mapa
Aliasing: No presenta aspectos de aliasing.

Alto (**in** mapa: Mapa) $\rightarrow res$: Nat
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{alto(m)}\}$
Complejidad: $\Theta(1)$
Descripción: Devuelve el alto del mapa
Aliasing: No presenta aspectos de aliasing.

•[•] (**in** mapa: Mapa, **in** v: Coord) $\rightarrow res$: Elemento
Pre $\equiv$ $\{\pi_2(v) < \text{largo(mapa)} \wedge \pi_1(v) < \text{alto(mapa)}\}$

Post $\equiv$ $\left\{\begin{array}{r}res = \text{PARED} \iff v \in \text{paredes(mapa)} \wedge\\ res = \text{FANTASMA} \iff v \in \text{fantasmas(mapa)} \wedge\\ res = \text{CHOCOLATE} \iff v \in \text{chocolates(mapa)} \wedge\\ res = \text{VACIO} \iff v \notin (\text{paredes(mapa)} \cup \text{fantasmas(mapa)} \cup \text{chocolates(mapa)})\end{array}\right\}$

Complejidad: $\Theta(1)$
Descripción: Devuelve el elemento en la coordenada v del mapa.
Aliasing: res es modificable y presenta aliasing.

2.3 Implementación

2.3.1 Representación

MAPA **se representa con** estr
donde estr **es** tupla$\langle largo, alto$: nat, $inicio, llegada$: coord, $elementos$: arreglo(elemento)$\rangle$

2.3.2 Invariante

① — *inicio* y *llegada* están dentro del mapa

② — *inicio* y *llegada* son distintos

③ — *elementos* en las coordenas de *inicio* y *llegada* no contiene PARED ni FANTASMA

④ — el tamaño de *elementos* es igual al producto de *largo* y *alto*

Rep : estr → boolean
Rep(e) ≡ true $\iff$ ① $\wedge_{\text{L}}$ ② $\wedge$ ③ $\wedge$ ④
donde:

① ≡ dentro($\{e.inicio,\ e.llegada\}, e.largo, e.alto$)

② ≡ $e.inicio \neq e.llegada$

③ ≡ $\{e.elementos[e.inicio.x,\ e.inicio.y],\ e.elementos[e.llegada.x,\ e.llegada.y]\} \cap \{\text{PARED},\ \text{FANTASMA}\} = \varnothing$

④ ≡ $e.elementos.longitud = e.largo \cdot e.alto$

dentro(cs, l, a) ≡ $(\forall c \in cs)(\pi_1(\ c\ ) < a \wedge \pi_2(\ c\ ) < l)$

2.3.3 Abstracción

Abs(e): estr → mapa ———— {Rep(e)}
$(\forall e : \text{estr}\ )$ Abs(e) $=_{\text{obs}}$ m : mapa | ① $\wedge$ ② $\wedge$ ③ $\wedge$ ④ $\wedge$ ⑤ $\wedge$ ⑥ $\wedge$ ⑦

① ≡ largo(m) $=_{\text{obs}}$ $e.largo$

② ≡ alto(m) $=_{\text{obs}}$ $e.alto$

③ ≡ inicio(m) $=_{\text{obs}}$ $e.inicio$

④ ≡ llegada(m) $=_{\text{obs}}$ $e.llegada$

⑤ ≡ paredes(m) $=_{\text{obs}}$ $\bigcup_{i=0}^{e.largo \cdot e.alto-1}$ **if** $e.elementos[i] = \text{PARED}$ **then** posACoord(i, $e.largo$) **else** $\varnothing$ **fi**

⑥ ≡ fantasmas(m) $=_{\text{obs}}$ $\bigcup_{i=0}^{e.largo \cdot e.alto-1}$ **if** $e.elementos[i] = \text{FANTASMA}$ **then** posACoord(i, e.largo) **else** $\varnothing$ **fi**

⑦ ≡ chocolates(m) $=_{\text{obs}}$ $\bigcup_{i=0}^{e.largo \cdot e.alto-1}$ **if** e.elementos[i] = CHOCOLATE **then** posACoord(i, e.largo) **else** $\varnothing$ **fi**

Axiomas: $\forall i, l$: nat
posACoord(i, l) ≡ $\langle i$ **mod** $l,\ i : l\rangle$
• **mod** •(i, l) ≡ **if** $i \geq l$ **then** $(i - l)$ **mod** l **else** i **fi**
• : •(i, l) ≡ **if** $i \geq l$ **then** $1 + (i - l) : l$ **else** 0 **fi**

2.3.4 Algoritmos

iNuevoMapa(**in** $largo, alto$: `nat`, **in** $inicio, llegada$: `coord`,
in $paredes, fantasmas, chocolates$: `conj(coord)`) → res : mapa

```
1:  res.largo ← largo                                                    ▷ Θ(1)
2:  res.alto ← alto                                                      ▷ Θ(1)
3:  res.inicio ← inicio                                                  ▷ Θ(1)
4:  res.llegada ← llegada                                                ▷ Θ(1)
5:  res.elementos ← CrearArreglo(largo · alto)                           ▷ Θ(L · A)
6:  for i : 0 ≤ i < largo · alto do                                      ▷ Θ(L · A)
7:      res.elementos[i] ← VACIO                                         ▷ Θ(1)
8:  end for
9:  for i : 0 ≤ i < #paredes do                                          ▷ Θ(|paredes|)
10:     res[paredes[i]] ← PARED                                          ▷ Θ(1)
11: end for
12: for i : 0 ≤ i < #fantasmas do                                        ▷ Θ(F)
13:     res[fantasmas[i]] ← FANTASMA                                     ▷ Θ(1)
14: end for
15: for i : 0 ≤ i < #chocolates do                                       ▷ Θ(c)
16:     res[chocolates[i]] ← CHOCOLATE                                   ▷ Θ(1)
17: end for
```

Complejidad: $\Theta(L \cdot A)$
Justificación: $|e.paredes| < L \cdot A$ y $F < L \cdot A$ y $c < L \cdot A$
$|e.paredes| + F + c < 3 \cdot L \cdot A$
$O(\text{máx}\,(|e.paredes|,\ F,\ c|)) \subseteq O(L \cdot A)$

---

iLargo(**in** $mapa$: `mapa`) → res : nat

```
1:  res ← mapa.largo                                                     ▷ Θ(1)
```

Complejidad: $\Theta(1)$

---

iAlto(**in** $mapa$: `mapa`) → res : nat

```
1:  res ← mapa.alto                                                      ▷ Θ(1)
```

Complejidad: $\Theta(1)$

•[•](**in** $mapa$: `mapa`, **in** v : `coord`) → res : elemento

```
1:  res ← mapa.elementos[v.x · mapa.largo + v.y]                         ▷ Θ(1)
```

Complejidad: $\Theta(1)$

3 Partida

3.1 Renombres

TAD Dirección **es** ENUM{ARRIBA, ABAJO, IZQUIERDA, DERECHA}

3.2 Interfaz

3.2.1 Tipos

se explica con: Bool, Dirección, Partida.
géneros: partida.

3.2.2 Operaciones básicas de partida

NuevaPartida (**in** mapa: Mapa) $\rightarrow$ *res* : Partida
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{nuevaPartida(mapa)}\}$
Complejidad: $\Theta(A \cdot L)$
Descripción: Crea una partida nueva con el mapa que ingresa.
Aliasing: No presenta aspectos de aliasing.

IniciarPartida (**inout** partida: Partida)
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{\text{partida} =_{\text{obs}} \text{nuevaPartida(mapa(partida))}\}$
Complejidad: $\Theta(c)$
Descripción: Reinicia el estado del mapa e inicia una nueva partida.
Aliasing: Modifica a partida.

Ganó? (**in** partida: Partida) $\rightarrow$ *res* : Bool
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{ganó?(partida)}\}$
Complejidad: $\Theta(1)$
Descripción: Informa si el jugador ganó la partida.
Aliasing: No presenta aspectos de aliasing.

Perdió? (**in** partida: Partida) $\rightarrow$ *res* : Bool
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{perdió?(partida)}\}$
Complejidad: $\Theta(1)$
Descripción: Informa si el jugador perdió la partida.
Aliasing: No presenta aspectos de aliasing.

MoverJugador (**inout** partida: Partida, **in** dir: Dirección) $\rightarrow$ *res* : Bool
Pre $\equiv$ $\{\text{partida}_0 = \text{partida}\}$
Post $\equiv$ $\{\text{partida} =_{\text{obs}} \text{mover}(\text{partida}_0, \text{dir}) \wedge res =_{\text{obs}} \text{ganó?(partida)} \vee \text{perdió?(partida)}\}$
Complejidad: $\Theta(1)$
Descripción: Mueve al jugador en la dirección especificada y devuelve si la partida terminó.
Aliasing: Modifica a partida.

3.3 Implementación

3.3.1 Representación

PARTIDA **se representa con** estr
donde estr **es** tupla⟨*puntaje* : nat, *jugador* : jugador, *mapa* : mapa, *chocolates* : vector(coord)⟩
donde jugador **es** tupla⟨*posicion* : coord, *inmunidad* : nat⟩

3.3.2 Invariante

① — *posicion* en *jugador* y *chocolates* están dentro del mapa.

② — No hay posiciones repetidas en *chocolates*.

③ — Todo CHOCOLATE en *mapa* está en *chocolates*.

④ — Toda coordenada de *chocolates* debe ser CHOCOLATE o VACIO en mapa.

⑤ — *posicion* en *jugador* no es una PARED.

⑥ — si *posicion* en *jugador* no es *inicio* en *mapa* entonces *posicion* en *mapa* no es un CHOCOLATE.

⑦ — *inmunidad* de *jugador* es menor o igual a 10.

Rep : estr → boolean
Rep(e) ≡ true ⟺ ① $\wedge_{\text{L}}$ ② ∧ ③ ∧ ④ ∧ ⑤ ∧ ⑥ ∧ ⑦
donde:

① ≡ $(\forall c : coord)(c \in \{e.jugador.posicion\} \cup e.chocolates \Rightarrow_{\text{L}} c < \langle e.mapa.largo,\ e.mapa.alto\rangle)$

② ≡ $(\forall i, j : nat)(0 \leq i < j < \text{long}(chocolates) \Rightarrow_{\text{L}} chocolates[i] \neq chocolates[j])$

③ ≡ $(\forall c : coord)(c < \langle e.mapa.largo,\ e.mapa.alto\rangle \wedge_{\text{L}} e.mapa[c] = \text{CHOCOLATE} \Rightarrow_{\text{L}} c \in e.chocolates)$

④ ≡ $(\forall c \in e.chocolates)(e.mapa[c] \in \{\text{VACIO},\ \text{CHOCOLATE}\})$

⑤ ≡ $e.mapa[e.jugador.posicion] \neq \text{PARED}$

⑥ ≡ $e.jugador.posicion \neq e.mapa.inicio \Rightarrow e.mapa[e.jugador.posicion] \neq \text{CHOCOLATE}$

⑦ ≡ $e.jugador.inmunidad \leq 10$

3.3.3 Abstracción

Abs(e): estr → partida ———— {Rep(e)}
$(\forall e : \text{estr})$ Abs(e) $=_{\text{obs}}$ p : partida | ① ∧ ② ∧ ③ ∧ ④ ∧ ⑤

① ≡ mapa(p) $=_{\text{obs}}$ colocarChocolates($e.chocolates$, $e.mapa$)

② ≡ jugador(p) $=_{\text{obs}}$ $e.jugador.posicion$

③ ≡ chocolates(p) $=_{\text{obs}}$ $\bigcup_{c<\langle e.mapa.largo,\ e.mapa.alto\rangle}$ **if** $e.mapa[c] = \text{CHOCOLATE}$ **then** c **else** $\varnothing$ **fi**

④ ≡ cantMov(p) $=_{\text{obs}}$ $e.puntaje$

⑤ ≡ inmunidad(p) $=_{\text{obs}}$ $e.jugador.inmunidad$

Axiomas: $\forall cs$: conj(coord), $\forall m$: mapa
colocarChocolates(cs, m) ≡ nuevoMapa($m.largo$, $m.alto$, $m.inicio$, $m.llegada$, paredes($\hat{m}$), fantasmas($\hat{m}$), cs)

3.3.4 Algoritmos

iNuevaPartida(**in** *mapa* : `mapa`) → *res* : partida

```
1: res.mapa ← mapa                                                    ▷ Θ(L · A)
2: res.chocolates ← VACÍO()                                           ▷ Θ(1)
3: for i : 0 ≤ i < mapa.alto do                                       ▷ Θ(A)
4:     for j : 0 ≤ i < mapa.largo do                                  ▷ Θ(L)
5:         if mapa[⟨i, j⟩] = CHOCOLATE then                           ▷ Θ(1)
6:             AGREGARATRAS(chocolates, ⟨i, j⟩)                       ▷ Θ(1)
7:         end if
8:     end for
9: end for
```

Complejidad: $\Theta(L \cdot A)$

Justificación:

$f(|chocolates|) \leq f(L \cdot A) \leq L \cdot A$ ▷ (f es la función de costo de inserción en vector)

$$\sum_{i=0}^{\lfloor \log_2 L \cdot A \rfloor} 2^i \leq 2 \cdot 2^{\lfloor \log_2 L \cdot A \rfloor} \leq 2 \cdot L \cdot A$$

$\Theta(f(|chocolates|) + L \cdot A + A \cdot L \cdot k)$ ▷ ($k \geq 1$)

$\Theta(f(|chocolates|) + (1 + k) \cdot L \cdot A)$

$\Theta(\text{máx}\,(f(|chocolates|),\ (1 + k) \cdot L \cdot A))$

$\Theta((1 + k) \cdot L \cdot A)$

$\Theta(L \cdot A)$

iIniciarPartida(**in/out** *partida* : `partida`)

```
1: for i : 0 ≤ i < LONGITUD(partida.chocolates) do                    ▷ Θ(c)
2:     partida.mapa[partida.chocolates[i]] ← CHOCOLATE                ▷ Θ(1)
3: end for
4: res.jugador.posicion ← mapa.inicio                                 ▷ Θ(1)
5: res.jugador.inmunidad ← β(mapa[mapa.inicio] = CHOCOLATE) · 10      ▷ Θ(1)
6: res.mapa[mapa.inicio] ← VACIO                                      ▷ Θ(1)
7: res.puntaje ← 0                                                    ▷ Θ(1)
```

Complejidad: $\Theta(c)$

iGanó?(**in** *partida* : `partida`) → *res* : bool

```
1: res ← partida.jugador.posicion = partida.mapa.llegada              ▷ Θ(1)
```

Complejidad: $\Theta(1)$

iPerdió?(**in** *partida*: `partida`) → *res* : bool

```
1:  res ← false                                                                          ▷ Θ(1)
2:  d ← 3                                                                                ▷ Θ(1)
3:  v ← partida.jugador.posicion                                                         ▷ Θ(1)
4:  for i : v.x − mín (d, v.x) ≤ i ≤ máx (v.x + d, partida.mapa.alto − 1) do              ▷ Θ(1)
5:      y ← d − |v.x − i|                                                                ▷ Θ(1)
6:      for j : v.y − mín (y, v.y) ≤ j ≤ máx (v.y + y, partida.mapa.largo − 1) do        ▷ Θ(1)
7:          if partida.mapa[⟨i, j⟩] = FANTASMA ∧ partida.jugador.inmunidad = 0 then      ▷ Θ(1)
8:              res ← true                                                               ▷ Θ(1)
9:          end if
10:     end for
11: end for
12: res ← res ∧ ¬Ganó?(partida)                                                          ▷ Θ(1)
```

Complejidad: $\Theta(1)$

iMoverJugador(**in/out** *partida*: `partida`, **in** *dir*: `direccion`)

```
1:  jugador ← partida.jugador                                                            ▷ Θ(1)
2:  nuevaPosicion ← partida.jugador.posicion                                             ▷ Θ(1)
3:  switch dir do
4:      case ARRIBA:
5:          if jugador.posicion.x > 0 then                                               ▷ Θ(1)
6:              nuevaPosicion ← nuevaPosicion − ⟨1, 0⟩                                   ▷ Θ(1)
7:          end if
8:      case ABAJO:
9:          if jugador.posicion.x < partida.mapa.alto − 1 then                           ▷ Θ(1)
10:             nuevaPosicion ← nuevaPosicion + ⟨1, 0⟩                                   ▷ Θ(1)
11:         end if
12:     case IZQUIERDA:
13:         if jugador.posicion.y > 0 then                                               ▷ Θ(1)
14:             nuevaPosicion ← nuevaPosicion − ⟨0, 1⟩                                   ▷ Θ(1)
15:         end if
16:     case DERECHA:
17:         if jugador.posicion.y < partida.mapa.largo − 1 then                          ▷ Θ(1)
18:             nuevaPosicion ← nuevaPosicion + ⟨0, 1⟩                                   ▷ Θ(1)
19:         end if
20: end switch
21: if jugador.posicion ≠ nuevaPosicion ∧ partida.mapa[nuevaPosicion] ≠ PARED then       ▷ Θ(1)
22:     jugador.posicion ← nuevaPosicion                                                 ▷ Θ(1)
23:     jugador.inmunidad ← jugador.inmunidad − β(jugador.inmunidad ≠ 0)                 ▷ Θ(1)
24:     partida.puntaje ← partida.puntaje + 1                                            ▷ Θ(1)
25: end if
26: if partida.mapa[jugador.posicion] = CHOCOLATE then                                   ▷ Θ(1)
27:     jugador.inmunidad ← 10                                                           ▷ Θ(1)
28:     partida.mapa[jugador.posicion] = VACIO                                           ▷ Θ(1)
29: end if
30: partida.jugador ← jugador                                                            ▷ Θ(1)
```

Complejidad: $\Theta(1)$

4 DiccTrie

4.1 Interfaz

4.1.1 Tipos

parámetros formales
géneros clave, significado.
función $\bullet = \bullet$ (**in** clave$_0$: Clave, **in** clave$_1$: Clave) $\to$ *res* : Bool
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} (\text{clave}_0 = \text{clave}_1)\}$
Complejidad: $\Theta(\text{equal}(\text{clave}_0, \text{clave}_1))$
Descripción: Función de igualdad entre claves.

función Copiar (**in** clave : Clave) $\to$ *res* : Clave
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{clave}\}$
Complejidad: $\Theta(\text{copy}(\text{clave}))$
Descripción: Función de copia de claves.

función Copiar (**in** significado : Significado) $\to$ *res* : Significado
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{significado}\}$
Complejidad: $\Theta(\text{copy}(\text{significado}))$
Descripción: Función de copia de significados.

se explica con: Bool, Diccionario(Clave, Significado).
géneros: dicctrie.

4.1.2 Operaciones básicas de dicctrie

Vacío $\to$ *res* : DiccTrie
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{vacío}\}$
Complejidad: $\Theta(1)$
Descripción: Crea un diccionario vacío.
Aliasing: No presenta aspectos de aliasing.

Definir (**in** clave: Clave, **in** significado: Significado, **inout** dicctrie: Dicctrie)
Pre $\equiv$ $\{\text{dicctrie}_0 = \text{dicctrie}\}$
Post $\equiv$ $\{\text{dicctrie} =_{\text{obs}} \text{definir}(\text{clave}, \text{significado}, \text{dicctrie}_0)\}$
Complejidad: $\Theta(|\text{clave}|)$
Descripción: Define o redefine la clave en el diccionario con el significado provisto.
Aliasing: Modifica a dicctrie.

Definido? (**in** clave: Clave, **in** dicctrie: Dicctrie) $\to$ *res* : Bool
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{def?}(\text{clave}, \text{dicctrie})\}$
Complejidad: $\Theta(|\text{clave}|)$
Descripción: Devuelve true si solo si la clave está definida en el diccionario.
Aliasing: No presenta aspectos de aliasing.

Obtener (**in** clave: Clave, **in** dicctrie: Dicctrie) $\to$ *res* : Significado
Pre $\equiv$ $\{\text{def?}(\text{clave}, \text{dicctrie})\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{obtener}(\text{clave}, \text{dicctrie})\}$
Complejidad: $\Theta(|\text{clave}|)$
Descripción: Dada una clave en el diccionario devuelve su valor.
Aliasing: No presenta aspectos de aliasing.

5 Fichín

5.1 Renombres

TAD JUGADOR **es** STRING
TAD RANKING **es** DICC(JUGADOR, NAT)

5.2 Interfaz

5.2.1 Tipos

se explica con: BOOL, NAT, CONJUNTO(α), MAPA, PARTIDA, FICHÍN.
géneros: fichin.

5.2.2 Operaciones básicas de fichin

NUEVOFICHÍN (**in** mapa: MAPA) $\rightarrow$ *res* : FICHÍN
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{nuevoFichin(mapa)}\}$
Complejidad: $\Theta(|paredes| + F + c)$, $O(L \cdot A)$
Descripción: Crea un fichín con el mapa especificado.
Aliasing: No presenta aspectos de aliasing.

EMPEZARPARTIDA (**inout** fichin: FICHIN **in** jugador: JUGADOR)
Pre $\equiv$ $\{\neg\text{alguienJugando?(fichin)} \land \text{fichin}_0 = \text{fichin}\}$
Post $\equiv$ $\{\text{fichin} =_{\text{obs}} \text{nuevaPartida}(\text{fichin}_0, \text{jugador})\}$
Complejidad: $\Theta(c + J)$
Descripción: Comienza una partida.
Aliasing: Modifica a fichín.

MOVERFICHÍN (**inout** fichin: FICHÍN, **in** dirección: DIR)
Pre $\equiv$ $\{\text{alguienJugando?(fichin)} \land \text{fichin}_0 = \text{fichin}\}$
Post $\equiv$ $\{\text{fichin} =_{\text{obs}} \text{mover(direccion)}\}$
Complejidad: $\Theta(J)$ si el movimiento gana la partida, $\Theta(1)$ en otro caso
Descripción: Mueve al jugador en la dirección seleccionada dentro de la partida.
Aliasing: Modifica a fichín.

RANKING (**in** fichin: FICHIN) $\rightarrow$ *res* : RANKING
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{Ranking(Fichin)}\}$
Complejidad: $\Theta(1)$
Descripción: Dado un fichin devuelve su ranking.
Aliasing: no presenta aspectos de aliasing.

OBJETIVO (**in** fichin: FICHIN) $\rightarrow$ *res* : TUPLA$\langle$JUGADOR, NAT$\rangle$
Pre $\equiv$ $\{\text{definido}(\text{jugadorActual(Fichin)}, \text{ranking(Fichin)}))\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{Objetivo(Fichin)}\}$
Complejidad: $\Theta(J + |ranking|)$
Descripción: Dado un jugador devuelve el nombre y el puntaje de su oponente a superar, aquel que tiene un puntaje mejor al suyo, pero por la mínima diferencia posible.
Aliasing: no presenta aspectos de aliasing.

5.3 Implementación

5.3.1 Representación

FICHÍN **se representa con** estr
donde estr **es** tupla⟨*enpartida*? : bool, *jugador* : string, *ranking* : dicctrie, *partida* : partida⟩

5.3.2 Invariante

① — *enpartida*? es verdadero si no perdí ni gané en *partida*

② — ningún puntaje en *ranking* es 0

Rep : estr → boolean

Rep(e) ≡ true ⟺ ① ∧ ②
donde:

① ≡ $e.enpartida?$ ⇒ ¬(ganó?($e.partida$) ∨ perdió?($e.partida$))

② ≡ $(\forall j \in e.ranking)(obtener(j, e.ranking) > 0)$

5.3.3 Abstracción

Abs(e): estr → fichin ——————— {Rep(e)}
$(\forall e : \text{estr})$ Abs(e) $=_{\text{obs}}$ f : fichin | ① ∧ ② ∧ ③ ∧ ④ ∧ ⑤ ∧ ⑥ ∧ ⑦

① ≡ mapa(f) $=_{\text{obs}}$ colocarChocolates($e.partida.chocolates$, $e.partida.mapa$)

② ≡ alguienJugando?(f) $=_{\text{obs}}$ $e.enpartida?$

③ ≡ ranking(f) $=_{\text{obs}}$ $e.ranking$

④ ≡ jugadorActual(f) $=_{\text{obs}}$ $e.jugador$

⑤ ≡ partidaActual(f) $=_{\text{obs}}$ $e.partida$

Axiomas: $\forall cs$: conj(coord), $\forall m$: mapa
colocarChocolates(cs, m) ≡ nuevoMapa($m.largo$, $m.alto$, $m.inicio$, $m.llegada$, paredes($\hat{m}$), fantasmas($\hat{m}$), cs)

5.3.4 Algoritmos

---

iNuevoFichin(**in** *mapa* : `MAPA`) → *res* : fichin

1: $res.enpartida? \leftarrow false$ ▷ $\Theta(1)$
2: $res.jugador \leftarrow$ "␣" ▷ $\Theta(1)$
3: $res.ranking \leftarrow$ VACIO( ) ▷ $\Theta(1)$
4: $res.partida \leftarrow$ NUEVAPARTIDA($mapa$) ▷ $\Theta(|paredes| + F + c)$

Complejidad: $\Theta(|paredes| + F + c)$, $O(L \cdot A)$

---

iEmpezarPartida(**in/out** *fichin* : `fichin`, **in** *jugador* : `jugador`)

1: INICIARPARTIDA($fichin.partida$) ▷ $\Theta(c)$
2: $fichin.jugador \leftarrow jugador$ ▷ $\Theta(J)$
3: $fichin.enpartida? \leftarrow \neg$PERDIÓ?($fichin.partida$) ▷ $\Theta(1)$

Complejidad: $\Theta(c + J)$

---

iMoverFichin(**in/out** *fichin* : `fichin`, **in** *dir* : `dir`)

```
1: MoverJugador(fichin.partida, dir)                                          ▷ Θ(1)
2: if Gano?(fichin.partida) ∨ Perdió?(fichin.partida) then                    ▷ Θ(1)
3:     fichin.enpartida? ← false                                              ▷ Θ(1)
4:     if Gano?(fichin.partida) then                                          ▷ Θ(1)
5:         AnotarEnRanking(fichin, fichin.jugador, fichin.partida.puntaje)    ▷ Θ(J)
6:     end if
7: end if
```

Complejidad: $\Theta(J)$ si el movimiento gana la partida, $\Theta(1)$ en otro caso

AnotarEnRanking(**in/out** *fichin* : `fichin`, **in** *jugador* : `jugador`, **in** *puntaje* : `nat`)

```
1: Definir(jugador, puntaje, fichin.ranking)                                  ▷ Θ(J)
```

Complejidad: $\Theta(J)$

iRanking(**in** *fichin* : `fichin`) → *res* : ranking

```
1: res ← fichin.ranking                                                       ▷ Θ(1)
```

Complejidad: $\Theta(1)$

iObjetivo(**in** *fichin* : `fichin`) → *res* : tupla(jugador, nat)

```
1: miPuntaje ← Obtener(fichin.jugador, fichin.ranking)                        ▷ Θ(J)
2: res ← ⟨fichin.jugador, miPuntaje)⟩                                         ▷ Θ(1)
3: it ← CrearIt(fichin.ranking)                                               ▷ Θ(1)
4: while HaySiguiente(it) do                                                  ▷ Θ(|ranking|)
5:     oponente ← Siguiente(it)                                               ▷ Θ(1)
6:     if π2(oponente) < miPuntaje ∧ (π2(res) < π2(oponente) ∨ π2(res) = miPuntaje) then   ▷ Θ(1)
7:         res ← oponente                                                     ▷ Θ(1)
8:     end if
9: end while
```

Complejidad: $\Theta(J + |ranking|)$

Comentarios

- En el TPII se definió el radio en que el jugador se asusta como ≤ 2 pero acá se lo define como ≤ 3.
  Se implementó como dice la especificación de este TP.