DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

TP3

29 de mayo de 2021 — Algoritmos y Estructuras de Datos II

Grupo: 6

Integrante	LU	Correo electrónico
Collasius, Federico	164/20	fede.collasius@gmail.com
Fernández Olivares Esnaola, Joaquín	11/20	joaquinfernandezolivares@gmail.com
Totaro, Facundo Ariel	43/20	facutotaro@gmail.com
Venturini, Julia	159/20	juliaventurini00@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

```
// Una coordenada se representa con tupla (x:nat, y:nat)

Representación:
  Mapa se representa con estr donde
    estr es tupla ( largo:      nat,
                    alto:       nat,
                    inicio:     coordenada,
                    llegada:    coordenada,
                    paredes:    conjlineal(coordenada),
                    fantasmas:  conjlineal(coordenada),
                    chocolates: conjlineal(coordenada),
                    tablero:    arreglo[e.largo] de arreglo[e.alto] de char)

Solución Informal:
  - El e.inicio es distinto a e.llegada.
  - e.chocolates, e.paredes y e.fantasmas son disjuntos.
  - e.inicio. e.llegada y todas las coordenadas en e.chocolates, e.paredes
    y e.fantasmas son menores o iguales que <e.largo, e.alto>.
  - Las coordenadas de los 3 conjuntos, e.inicio y e.llegada son mayores a 0.
  - e.inicio y e.llegada no pertencen a e.paredes o e.fantasmas.
  - Los valores de e.tablero pueden ser P, C, F, I, L o son V. X,Y denotan el caso en el que un chocolate se encuentra en la misma
    coordenada que inicio y llegada respectivamente.
  - Para toda coordenada coor en los conjuntos, e.paredes, e.fantasmas,
    arreglo[π1(coor)][π2(coor)] = inicial correspondiente.
    En el caso de una posicion vacia será un V. En el caso de e.inicio y e.llegada la inicial
    puede ser X, I para el caso de e.inicio y Y, L para e.llegada.
  - Para toda coordenada coor perteneciente a e.chocolates, arreglo[π1(coor)][π2(coor)]
    pueden ser igual a C, V, X, Y según lo que corresponda.

---
Función de abstracción:
  Abs:  estr  ->  tipoDato  {Rep(e)}
  (∀e: estr) Abs(e) =obs m: Mapa | (1) ∧ (2) ∧ (3) ∧ (4) ∧ (5) ∧ (6) ∧ (7)
    donde:
    (1) ≡ largo(m) =obs e.largo
    (2) ≡ alto(m) =obs e.alto
    (3) ≡ inicio(m) =obs e.inicio
    (4) ≡ llegada(m) =obs e.llegada
    (5) ≡ paredes(m) =obs e.paredes
    (6) ≡ fantasmas(m) =obs e.fantasmas
    (7) ≡ chocolates(m) =obs e.chocolates
---
Interfaz:
  Parámetros formales:
    Géneros:  mapa
    Función:
      Copiar(in a: mapa) -> res: mapa
      Pre ≡ {true}
      Post ≡ {res = a}
      Complejidad: Θ(copy(a))
      Descripción: Función copia de mapa.
      ---
  Se explica con: Mapa
  Géneros:    mapa

  Operaciones básicas:
    NuevoMapa(in largo, alto: nat, in inicio, llegada: coordenada, in paredes, fantasmas, chocolates: conj(coordenada)) -> res: mapa
    Pre ≡ {largo = largo0 ∧ alto = alto0 ∧ inicio = inicio0 ∧ llegada = llegada0 ∧ paredes = paredes0 ∧ fantasmas = fantasmas0 ∧ chocolates = chocolates0}
    Post ≡ {res.largo = largo0 ∧ res.alto = alto0 ∧ res.inicio = inicio0
            ∧ res.llegada = llegada0 ∧ res.paredes = paredes0 ∧
            ∧ res.fantasmas = fantasmas0 ∧ res.chocolates = chocolates0 ∧L
          (∀c:coordenada)( (( c = res.inicio ∧ c ∈ res.chocolate) =>L res.tablero[ c.x ][ c.y ] = 'X' )                                    ∧
                          (( c = res.llegada ∧ c ∈ res.chocolate) =>L res.tablero[ c.x ][ c.y ] = 'Y' )                                    ∧
                           ( c ∈ res.paredes =>L res.tablero[ c.x ][ c.y ] = 'P' )                                                     ∧
                           ( c ∈ res.fantasmas =>L res.tablero[ c.x ][ c.y ] = 'F' )                                                   ∧
                          (( c ∈ res.chocolate ∧ c ≠ res.inicio ∧ c ≠ res.llegada) =>L res.tablero[ c.x ][ c.y ] = 'C')                    ∧
                           ( c ∉ (res.paredes ∪ res.fantasma ∪ res.chocolate ∪ {res.inicio, res.llegada}) =>L res.tablero[ c.x ][ c.y ] = 'V' )
                          )
                        }
    Complejidad: Θ(alto * largo)
    Descripción: Genera un nuevo mapa.
    Aliasing: No presenta aspectos de aliasing.
    ---
    Largo(in mapa: mapa) -> res: nat
    Pre  ≡ {true}
    Post ≡ {res =obs largo(mapa)}
    Complejidad: Θ(1)
    Descripción: Devuelve el largo del mapa.
    Aliasing: No presenta aspectos de aliasing.
    ---
    Alto(in mapa: mapa) -> res: nat
    Pre  ≡ {true}
    Post ≡ {res =obs alto(mapa)}
    Complejidad: Θ(1)
    Descripción: Devuelve el alto del mapa.
    Aliasing: No presenta aspectos de aliasing.
    ---
    Inicio(in mapa: mapa) -> res: coordenada
    Pre  ≡ {true}
    Post ≡ {res =obs inicio(mapa)}
    Complejidad: Θ(1)
    Descripción: Devuelve la coordenada de inicio.
    Aliasing: No presenta aspectos de aliasing.
    ---
    Llegada(in mapa: mapa) -> res: coordenada
    Pre  ≡ {true}
    Post ≡ {res =obs llegada(mapa)}
    Complejidad: Θ(1)
    Descripción: Devuelve la coordenada de llegada.
    Aliasing: No presenta aspectos de aliasing.
    ---
```

```
Interfaz:
  Paredes(in mapa: mapa) -> res: conj(coordenada)
  Pre  ≡ {true}
  Post ≡ {res =obs paredes(mapa)}
  Complejidad: 0(1)
  Descripción: Devuelve el conjunto de coordenadas donde se encuentran paredes.
  Aliasing: No presenta aspectos de aliasing.
  ---
  Fantasmas(in mapa: mapa) -> res: conj(coordenada)
  Pre  ≡ {true}
  Post ≡ {res =obs fantasmas(mapa)}
  Complejidad: 0(1)
  Descripción: Devuelve el conjunto de coordenadas donde se encuentran fantasmas.
  Aliasing: No presenta aspectos de aliasing.
  ---
  Chocolates(in mapa: mapa) -> res: conj(coordenada)
  Pre  ≡ {true}
  Post ≡ {res =obs chocolates(mapa)}
  Complejidad: 0(1)
  Descripción: Devuelve el conjunto de coordenadas donde se encuentran chocolates.
  Aliasing: No presenta aspectos de aliasing.
  ---
  Distancia(in pos1, pos2: coordenada) -> res: nat
  Pre  ≡ {true}
  Post ≡ {res =obs distancia(pos1, pos2)}
  Complejidad: 0(1)
  Descripción: Calcula la distancia Manhattan entre dos coordenadas.
  Aliasing: No presenta aspectos de aliasing.
  ---
  DistConFantasmaMasCercano(in fantasmas: conj(coordenada), in pos: coordenada) -> res: nat
  Pre  ≡ {¬vacío?(fantasmas)}
  Post ≡ {(∀f:coordenada)( f ∈ fantasmas₀ =>L distancia(pos,f) ≥ res  ) ∧
          (∃f:coordenada)( f ∈ fantasmas₀ ∧L  distancia(pos,f) = res  )}
  Complejidad: 0(#(fantasmas₀))
  Descripción: Calcula la distancia con el fantasma mas cercano.
  Aliasing: No presenta aspectos de aliasing.
  ---
  EnRango(in pos: coordenada, in largo, alto: nat) -> res: bool
  Pre  ≡ {true}
  Post ≡ {res =obs enRango(pos, largo, alto)}
  Complejidad: 0(1)
  Descripción: Devuelve si una coordenada está dentro de los limites del mapa.
  Aliasing: No presenta aspectos de aliasing.
  ---
  TodasEnRango(in posiciones: conj(coordenada), in largo, alto: nat) -> res: bool
  Pre  ≡ {true}
  Post ≡ {res =obs todasEnRango(posiciones, largo, alto)}
  Complejidad: 0(#(posiciones₀))
  Descripción: Devuelve si todas las posiciones en un conjunto de cordenadas se
               encuenteran en los limites del mapa.
  Aliasing: No presenta aspectos de aliasing.
  ---
Algoritmos del módulo:
  iNuevoMapa(in largo, alto: nat, in inicio, llegada: coordenada,
  in paredes, fantasmas, chocolates: conj(coordenada)) -> res: mapa
    res <- (largo, ancho, inicio, llegada, paredes, fantasmas, chocolates, tablero: arreglo[e.largo] de arreglo[e.alto] de char)
    for i ← 1, i ≤ largo, i++
      for j ← 1, j ≤ alto, j++
        res.tablero[i][j] <- V
      end for
    end for

    for e in res.paredes do
      res.tablero[e.x][e.y] <- P
    end for

    for e in res.fantasmas do
      res.tablero[e.x][e.y] <- F
    end for

    for e in res.chocolates do
      res.tablero[e.x][e.y] <- C
    end for

    if res.tablero[res.inicio.x][res.inicio.y] = C then
      res.tablero[res.inicio.x][res.inicio.y] <- X
    else
      res.tablero[res.inicio.x][res.inicio.y] <- I
    end if

    if res.tablero[res.llegada.x][res.llegada.y] = C then
      res.tablero[res.llegada.x][res.llegada.y] <- Y
    else
      res.tablero[res.llegada.x][res.llegada.y] <- L
    end if
  ---
  iLargo(in mapa: estr) -> res: nat
  res <- mapa.largo
  ---
  iAlto(in mapa: estr) -> res: nat
  res <- mapa.alto
  ---
  iInicio(in mapa: estr) -> res: coordenada
  res <- mapa.inicio
  ---
  iLlegada(in mapa: estr) -> res: coordenada
  res <- mapa.llegada
```

```
Interfaz:
    iParedes(in mapa: estr) -> res: conj(coordenada)
    res <- mapa.paredes
    ---
    iFantasmas(in mapa: estr) -> res: conj(coordenada)
    res <- mapa.fantasmas
    ---
    iChocolates(in mapa: estr) -> res: conj(coordenada)
    res <- mapa.chocolates
    ---
    iDistancia(in pos1, pos2: coordenada) -> res: nat
    res <- |pos1.x - pos2.x| + |pos1.y - pos2.y|
    ---
    iDistConFantasmaMasCercano(in fantasmas: conj(coordenada), in pos: coordenada) -> res: nat
    j <- CrearIt(fantasmas)
    min <- Distancia(pos, *j)
    for e in fantasmas do
     if min > Distancia(e, pos) then
       min <- e
     endif
    end for
    ---
    iEnRango(in pos: coordenada, in largo, alto: nat) -> res: bool
    res <- pos.x > 0 && pos.x ≤ largo && pos.y > 0 && pos.y ≤ alto
    ---
    iTodasEnRango(in posiciones: conj(coordenada), in largo, alto: nat) -> res: bool
    res <- true
    for e in posiciones do
     res <- res && EnRango(e, largo, alto)
    end for
     ---

```

Representación:

Partida se representa con estr donde

estr es tupla (mapa: puntero(Mapa),
jugador: coordenada,
chocolates: puntero(*mapa).chocolates, // el puntero hace referencia al mapa del Modulo Mapa.
cantMov: nat,
inmunidad: nat)

Solución Informal:

- e.jugador tiene que ser una coordenada en el rango del mapa.
- las coordendas dentro de e.(*mapa).chocolates están en el rango del mapa.
- e.inmunidad es menor o igual a 10.

Función de abstracción:

Abs: estr -> tipoDato {Rep(e)}
(∀e: estr) Abs(e) =obs p: Partida | (1)∧(2)
donde:
(1) ≡ mapa(p) =obs *e.mapa
(2) ≡ jugador(p) =obs e.jugador
(3) ≡ chocolates(p) =obs *e.chocolates
(4) ≡ cantMov(p) =obs e.cantMov
(5) ≡ inmunidad(p) =obs e.inmunidad

Interfaz:

Parámetros formales:

Géneros: partida

Función:

Copiar(in p: partida) -> res: partida
Pre ≡ {true}
Post ≡ {res = p}
Complejidad: Θ(copy(p))
Descripción: Función copia de partida's.

Se explica con: Partida

Géneros: partida

Operaciones básicas: // corregir post con funciones del tad

Mapa(in p: partida) -> res: mapa
Pre ≡ {true}
Post ≡ {res =obs mapa(p)}
Complejidad: Θ(1)
Descripción: Devuelve una copia del Mapa
Aliasing: no genera aliasing

Jugador(in p: partida) -> res: coordenada
Pre ≡ {true}
Post ≡ {res =obs jugador(p)}
Complejidad: Θ(1)
Descripción: Devuelve una copia de la coordenada actual del jugador.
Aliasing: No genera aliasing.

Chocolates(in p: partida) -> res: conj(coordenada)
Pre ≡ {true}
Post ≡ {res =obs chocolates(p)}
Complejidad: Θ(1)
Descripción: Devuelve por copia el conjunto de chocolates en la partida.
Aliasing: No genera aliasing.

CantMov(in p: partida) -> res: nat
Pre ≡ {true}
Post ≡ {res =obs cantMov(p)}
Complejidad: Θ(1)
Descripción: Devuelve por copia la cantidad de movimientos que realizo el jugador hasta ahora.
Aliasing: No genera aliasing.

Inmunidad(in p: partida) -> res: nat
Pre ≡ {true}
Post ≡ {res =obs inmunidad(p)}
Complejidad: Θ(1)
Descripción: Devuelve por copia la cantidad de turnos de inmunidad.
Aliasing: No genera aliasing.

NuevaPartida(in m: mapa*) -> res: partida
Pre ≡ {true}
Post ≡ {res =obs nuevaPartida(*m) ∧
(∀c:coordenada)((c ∈ res.chocolates) =>L *(res.mapa).tablero[c.x][c.y] = 'C'}
Complejidad: Θ(#(*(res.chocolates)))
Descripción: Genera una partida dado un mapa pasado por referencia
Aliasing: Genera aliasing porque crea un puntero a un mapa al que ya apuntaba otro puntero que fue creado en el Módulo Mapa.

Mover(in p: partida, in d: dirección) -> res: partida
Pre ≡ {¬ganó?(p) ∧ ¬perdió?(p) ∧ p = p_0}
Post ≡ {if esValida(posMovimiento(p.jugador, d)) then
if posMovimiento(p.jugador, d) ∈ (*p.mapa).paredes then
res = p_0
else
res.mapa = p_0.mapa ∧ res.jugador = posMovimiento(p.jugador, d) ∧ res.chocolates* = ((*p_0.mapa).chocolates) ∧
res.cantMov = p_0.cantMov + 1 ∧ res.Inmunidad = max(0,p_0.Inmunidad - 1)
if *(p_0.mapa).tablero[π_0(posMovimiento(p.jugador, d))][π_1(posMovimiento(p.jugador, d))] = C then
res.mapa = p_0.mapa ∧ res.jugador = posMovimiento(p.jugador, d) ∧ res.chocolates* = ((*p_0.mapa).chocolates) ∧
res.cantMov = p_0.cantMov + 1 ∧ res.Inmunidad = 10 ∧ *(p_0.mapa).tablero[π_0(posMovimiento(p.jugador, d))][π_1(posMovimiento(p.jugador, d))] =
fi
else
res = p_0
fi
fi
}
Complejidad: Θ(1)
Descripción: Mueve al jugador según corresponda.
Aliasing: Genera aliasing porque modifica al mapa.

```
Interfaz:
  Ganó(in p: partida) -> res: bool
  Pre ≡ {true}
  Post ≡ {res =obs ((*p.mapa).llegada = p.jugador) }
  Complejidad: 0(1)
  Descripción: Devuelve si el jugador está en la posición de llegada del tablero.
  Aliasing: No genera aliasing.
  ---
  Perdió(in p: partida) -> res: bool
  Pre ≡ {true}
  Post ≡ {res =obs perdió?(p) }
  Complejidad: 0(1)
  Descripción: Devuelve si el jugador está en la posición de llegada del tablero.
  Aliasing: No genera aliasing.
  ---
  SiguienteMovimiento(in p: partida, in d: dirección) -> res: coordenada
  Pre ≡ {true}
  Post ≡ {res =obs siguienteMovimiento(p,d) }
  Complejidad: 0(1)
  Descripción: Devuelve la coordenada de la proxima posicion del jugador dada una dirección
               a menos que haya una pared, en cuyo caso se queda en el mismo lugar.
  Aliasing: No genera aliasing.
  ---
  PosMovimiento(in c: coordenada, in d: dirección) -> res: coordenada
  Pre ≡ {true}
  Post ≡ {res =obs posMovimiento(c, d)}
  Complejidad: 0(1)
  Descripción: Dada una coordenada y una dirección devuelve la coordenada inmediatamente
               en la dirección pasada.
  Aliasing: No genera aliasing.
  ---
  RestringirMovimiento(in p: partida, in c: coordenada) -> res: coordenada
  Pre  ≡ {true}
  Post ≡ {res =obs restringirMovimiento(p, c)}
  Complejidad: 0(1)
  Descripción: Devuelve una coordenada tal que esté en los limites del mapa.
  Aliasing: No genera aliasing.
  ---
  Algoritmos del módulo:
  iMapa(in p: partida) -> res: mapa
    res <- *(p.mapa)
  ---
  iJugador(in p: partida) -> res: coordenada
    res <- p.jugador
  ---
  iChocolates(in p: partida) -> res: conj(coordenada)
    res <- p.chocolates
  ---
  iCantMov(in p: partida) -> res: nat
    res <- p.CantMov
  ---
  iInmunidad(in p: partida) -> res: nat
    res <- p.inmunidad
  ---
  iNuevaPartida(in m: mapa) -> res: partida
    res.mapa* = m
    res.jugador = *m.inicio
    res.chocolates* = m.chocolates
    res.inmunidad = 0
    for e in *res.chocolates do
      if e = (*m).inicio then
        m[ e.x ][ e.y ] = 'X'
        res.inmunidad = 10;
        m[ e.x ][ e.y ] = 'S'
      else
        if e = m.llegada then
          m[ e.x ][ e.y ] = 'Y'
        else
          m[ e.x ][ e.y ] = 'C'
        end if
      end if
    endfor
    res.cantMov = 0;
  ---
  iMover(in p: partida, in d: dirección) -> res: partida
    res <- p
    if !(siguienteMovimiento(p, d) = p.jugador) then
      res.jugador = siguienteMovimiento(p,d)
      p <- *(p.mapa).tablero[siguienteMovimiento(p,d).x][siguienteMovimiento(p,d).y]
      res.inmunidad = max(0, p.inmunidad-1)
      if p = 'C' then
        *(p.mapa).tablero[siguienteMovimiento(p,d).x][siguienteMovimiento(p,d).y] <- 'V'
        res.inmunidad = 10
      else
        if p = 'Y' then
        *(p.mapa).tablero[siguienteMovimiento(p,d).x][siguienteMovimiento(p,d).y] <- 'L'
        res.inmunidad = 10
        end if
      res.cantMov = p.cantMov + 1
    end if
    ---
  iGanó(in p: partida) -> res: bool
    res <- *(p.mapa).llegada = p.jugador
    ---
  iPerdió(in p: partida) -> res: bool
    res <-  !Ganó(p) && (DistConFantasmaMasCercano(*(p.mapa).fantasmas, p.jugador) ≤ 3) && p.Inmunidad = 0
    ---
```

```
Interfaz:
  iSiguienteMovimiento(in p: partida, in d: dirección) -> res: coordenada
    if (RestringirMovimineto(p,posMovimiento(p.jugador,d)) != p.jugador && *(p.mapa).tablero[posMovimiento(p.jugador,d).x][posMovimiento(p.jugador,d).y] != 'P
      res <- posMovimiento(p.jugador, d)
    else
      res <- p.jugador
    end if
    coor <- RestringirMovimiento(p, PosMovimiento(p.jugador, d))
    if !(*(p.mapa).tablero[ coor.x ][ coor.y ] = 'P') then
       res <- coor
    else
       res <- p.jugador
    end if
    ---
  iPosMovimiento(in c: coordenada, in d: dirección) -> res: coordenada
    res <- ( c.x + (if d = DERECHA then 1 else 0),
             c.y + (if d = Arriba then 1 else 0) )
    ---
  iRestringirMovimiento(in p: partida, in c: coordenada) -> res: coordenada
    res <- ( max(0, min(*(p.mapa).largo - 1, c.x),
             max(0, min(*(p.mapa).alto - 1, c.y) )
  ---
  Auxiliares:
  Extendemos Modulo Int:

  max(in a: int, in b: int) -> res: int
  Pre  ≡ {true}
  Post ≡ {res =obs max(a, b)}
  Compejidad: Θ(1)
  Descripción: Devuelve el máximo entre dos enteros.
  Aliasing: No genera aliasing.

  iMax(in a: int, in b: int) -> res: int
  if b > a then
  res <- b
  else
  res <- a
  fi

  ---
  min(in a: int, in b: int) -> res: int
  Pre  ≡ {true}
  Post ≡ {res =obs min(a, b)}
  Compejidad: Θ(1)
  Descripción: Devuelve el mínimo entre dos enteros.
  Aliasing: No genera aliasing.

  iMin(in a: int, in b: int) -> res: int
  if b < a then
  res <- b
  else
  res <- a
  fi
```

Representación:

Fichin se representa con estr donde

estr es tupla (mapa: puntero(mapa),
alguienJugando?: bool,
jugadorActual: jugador,
partidaActual: puntero(partida),
ranking: dicTrie(jugador, nat))

Solución Informal:

- las coordendas que no sean chocolates en *e.mapa van a ser iguales a esas mismas coordenadas en *(*(e.partidaActual).mapa).
- e.alguienJugando? = true ⟺ se inició la partida y el jugador no ganó.

Función de abstracción:

Abs: estr -> Fichin {Rep(e)}
(∀e: estr) Abs(e) =obs f: Fichin | (1) ∧ (5) ∧ (2) ∧L (3) ∧ (4)
donde:
(1) ≡ mapa(f) =obs *e.mapa
(2) ≡ alguienJugando(f) =obs e.alguienJugando
(3) ≡ (alguienJugando(f) =>L (jugadorActual(f) =obs e.jugadorActual))
(4) ≡ (alguienJugando(f) =>L (partidaActual(f) =obs *e.partidaActual))
(5) ≡ ranking(f) =obs e.ranking

Interfaz:

Parámetros formales:

Géneros: fichin

Función:

Copiar(in f:fichin) -> res: fichin
Pre ≡ {true}
Post ≡ {res = f}
Complejidad: Θ(copy(f))
Descripción: Función copia de fichin's.

Se explica con: Fichin

Géneros: fichin

Operaciones básicas:

Mapa(in f: fichin) -> res: mapa
Pre ≡ {true}
Post ≡ {res =obs mapa(f)}
Complejidad: Θ(1)
Descripción: Devuelve el mapa de la partida.
Aliasing: No presenta aspectos de aliasing.

AlguienJugando?(in f: fichin) -> res: bool
Pre ≡ {true}
Post ≡ {res =obs alguienJugando?(f)}
Complejidad: Θ(1)
Descripción: Devuelve si hay un jugador actualmente jugando.
Aliasing: No presenta aspectos de aliasing.

JugadorActual(in f: fichin) -> res: jugador
Pre ≡ {alguienJugando(f)}
Post ≡ {res =obs jugadorActual(f)}
Complejidad: Θ(1)
Descripción: Devuelve al jugador actual.
Aliasing: No presenta aspectos de aliasing.

PartidaActual(in f: fichin) -> res: partida
Pre ≡ {alguienJugando(f)}
Post ≡ {res =obs partidaActual(f)}
Complejidad: Θ(1)
Descripción: Devuelve la partida actual.
Aliasing: No presenta aspectos de aliasing.

Ranking(in f: fichin) -> res: ranking
Pre ≡ {true}
Post ≡ {res =obs ranking(f)}
Complejidad: Θ(1)
Descripción: Devuelve el ranking del fichin.
Aliasing: No presenta aspectos de aliasing.

NuevoFichin(in m: mapa) -> res: fichin
Pre ≡ {true}
Post ≡ {res =obs nuevoFichin(m)}
Complejidad: Θ(1)
Descripción: Dado un mapa devuelve un fichin.
Aliasing: No presenta aspectos de aliasing.

NuevaPartida(in f: fichin, in j: jugador) -> res: fichin
Pre ≡ {¬AlguienJugando(f)}
Post ≡ {res =obs nuevaPartida(f, j))}
Complejidad: Θ(NuevaPartida(*f.mapa))
Descripción: Dado un fichin y un jugador arma una nueva partida.
Aliasing: No presenta aspectos de aliasing.

Mover(in f: fichin, in d: dirección) -> res: fichin
Pre ≡ {alguienJugando?(f)}
Post ≡ {res =obs mover(f, d)}
Complejidad: Θ(1)
Descripción: Mueve de posición al jugador de un fichin dada una dirección.
Aliasing: Presenta aspectos de aliasing porque modificamos la partida.

Objetivo(in f: fichin) -> res: tupla (jugador, nat)
Pre ≡ {alguienJugando?(f) ∧ def?(jugadorActual(f), ranking(f))}
Post ≡ {res =obs objetivo(f) }
Complejidad: O(J.|J|)
Descripción: Devuelve una tupla con el nombre del jugador al que tiene que superar el jugador actual con su puntaje.
Aliasing: No presenta aspectos de aliasing.

```
Interfaz:
  Oponente(in f: fichin) -> res: jugador
  Pre  ≡ {alguienJugando?(f) ∧ def?(jugadorActual(f), ranking(f))}
  Post ≡ {res =obs oponente(f)}
  Complejidad: O(1)
  Descripción: Devuelve uno de los jugadores que tiene mejor puntaje que el actual.
  Aliasing: No presenta aspectos de aliasing.
  ---
  Oponentes(in f: fichin) -> res: conj(jugador)
  Pre  ≡ {alguienJugando?(f) ∧ def?(jugadorActual(f), ranking(f))}
  Post ≡ {res =obs oponentes(f)}
  Complejidad: O(J.|J|)
  Descripción: Devuelve el conjunto de jugadores con puntaje inmediatamente superior al jugador actual.
  Aliasing: No presenta aspectos de aliasing.
  ---
  MejoresQue(in r: ranking, in cj: conj(jugador), in p: nat) -> res: conj(jugador)
  Pre  ≡ {cj ⊆ claves(r)}
  Post ≡ {res =obs mejoresQue(r, cj, p)}
  Complejidad: O(n. |J|) donde n es la cantidad de elementos de cj.
  Descripción: Devuelve el conjunto con todos los jugadores que tienen mejor puntaje que el natural pasado por parametro.
  Aliasing: No presenta aspectos de aliasing.
  ---
  PeoresJugadores(in r: ranking, in cj: conj(jugador)) -> res: conj(jugador)
  Pre  ≡ {cj ⊆ claves(r)∧ ¬∅(cj)}
  Post ≡ {res =obs peoresJugadores(r, cj)}
  Complejidad: Θ(n.|J|) donde n es la cantidad de elementos de cj.
  Descripción: Devuelve el conjunto de jugadores con el peor puntaje del conjunto ingresado.
  Aliasing: No presenta aspectos de aliasing.
  ---
  JugadoresConPuntaje(in r: ranking, in cj: conj(jugador), in p: nat) -> res: conj(jugador)
  Pre  ≡ {cj ⊆ claves(r)∧ ¬∅?(cj)}
  Post ≡ {res =obs jugadoresConPuntaje(r, cj, p)}
  Complejidad: Θ(n.|J|) donde n es la cantidad de elementos de cj.
  Descripción: Devuelve el conjunto de jugadores con un puntaje igual al pasado por parámetro.
  Aliasing: No presenta aspectos de aliasing.
  ---
  PeorPuntaje(in r: ranking, in cj: conj(jugador)) -> res: nat
  Pre  ≡ {cj ⊆ claves(r) ∧ ¬∅?(cj)}
  Post ≡ {res =obs peorPuntaje(r, cj)}
  Complejidad: O(n.|J|) donde n es la cantidad de elementos de cj.
  Descripción: Devuelve el puntaje más bajo del conjunto de jugadores del ranking.
  Aliasing: No presenta aspectos de aliasing.
  ---
  Algoritmos del módulo:
  iMapa(in f: fichin) -> res: mapa
    res <- *(f.mapa)
  ---
  iAlguienJugando?(in f: fichin) -> res: bool
    res <- f.alguienJugando?
  ---
  iJugadorActual(in f: fichin) -> res: jugador
    res <- f.jugadorActual
  ---
  iPartidaActual(in f: fichin) -> res: partida
    res <- *(f.partidaActual)
  ---
  iRanking(in f: fichin) -> res: ranking
    res <- f.ranking
  ---
  iNuevoFichin(in m: mapa) -> res: fichin
    res.mapa* = m
    res.alguienJugando = false
    res.jugadorActual = " "
    res.partidaActual = NULL
    res.ranking = Vacío()
  ---
  iNuevaPartida(in f: fichin, in j: jugador) -> res: fichin
    res.alguienJugando = true
    res.jugadorActual = j
    res.partidaActual* = NuevaPartida(*(f.mapa))
  ---
  iMover(in f: fichin, in d: dirección) -> res: fichin
    res <- fichin
    if Ganó(Mover(*(f.partidaActual), d)) then
        if (( def?(f.jugadorActual, f.ranking) && obtener(f.jugadorActual, f.ranking) > *(f.partidaActual).CantMov) || !def?(f.jugadorActual, f.ranking)) then
            definir(f.jugadorActual,*(f.partidaActual).CantMov), f.ranking)
        endif
        res.alguienJugando = false
    else
      if Perdió(Mover(*(f.partidaActual), d)) then
          res.alguienJugando = false
      else
          Mover(*(f.partidaActual), d)
      end if
    endif
  ---
  iObjetivo(in f: fichin) -> res: tupla (jugador, nat)
    res <- (Oponente(f), obtener(Ranking(f), Oponente(f)))
  ---
  iOponente(in f: fichin) -> res: jugador
      if #Oponentes(f) = 0 then
        res <- JugadorActual(f)
      else
        it <- crearIt(Oponentes(f))
        res <- Siguiente(it)
      end if
```

```
Interfaz:
  iOponentes(in f: fichin) -> res: conj(jugador)
    if vacio?(MejoresQue(Ranking(f), claves(Ranking(f)), obtener(Ranking(f), jugadorActual(f)))) then
      res <- vacio()
    else
      res <- PeoresJugadores(Ranking(f), MejoresQue(Ranking(f), claves(Ranking(f)), obtener(Ranking(f), JugadorActual(f))))
    end if
  ---
  iMejoresQue(in r: ranking, in cj: conj(jugador), in p: nat) -> res: conj(jugador)
    res <- vacio()
      for j in cj do
        if obtener(j, r) > p then
          Agregar(res, j)
        end if
      end for
  ---
  iPeoresJugadores(in r: ranking, in cj: conj(jugador)) -> res: conj(jugador)
    res <- JugadoresConPuntaje(r, cj, PeorPuntaje(r, cj))
  ---
  iJugadoresConPuntaje(in r: ranking, in cj: conj(jugador), in p: nat) -> res: conj(jugador)
    res <- vacio()
    for j in cj do
      if obtener(j, r) = p then
        Agregar(res, j)
      end if
    end for
  ---
  iPeorPuntaje(in r: ranking, in cj: conj(jugador)) -> res: nat
    it <- crearIt(cj)
    peor <- Siguiente(it)
    for j in cj do
      if obtener(j, r) > peor then
        peor <- obtener(j, r)
      end if
    end for
    res <- peor
```