Algoritmos y Estructura de Datos 2

Trabajo Práctico 1 - Especificación de *Sokoban*

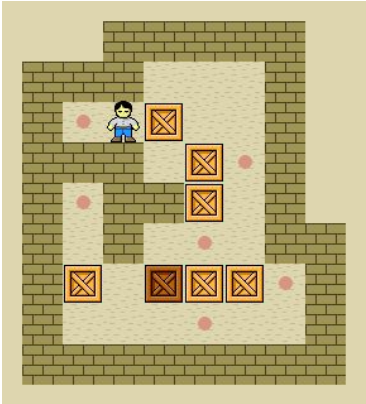
Alumno: Leandro Carreira
LU: 669/18
Grupo: 15

Este trabajo consiste en especificar el juego de ingenio Sokoban[1], diseñado en 1981 por Hiroyuki Imabayashi.

Reglas del juego.

El juego tiene lugar en una **grilla infinita** dividida en celdas de 1 × 1.

El jugador controla una persona que puede moverse de a una celda por vez en cualquiera de las **cuatro direcciones** (Norte, Este, Sur y Oeste).



Algunas celdas de la grilla tienen **paredes** por las que la persona **no** puede pasar.

Las celdas que **no** tienen **paredes** son **transitables**.

Algunas de las celdas transitables están marcadas como **depósitos**.

En las celdas transitables de la grilla puede haber **cajas**.

Si la persona se mueve en dirección hacia una celda en la que hay una caja, se mueve hacia esa dirección y además **empuja la caja hacia dicha dirección**.

Esta acción de **empujar una caja** solamente se puede realizar si la celda a la que debe ir a parar la caja **no tiene una pared ni otra caja**.

En un nivel del juego suponemos que hay una cierta **cantidad k de cajas** y el mismo **número k de depósitos**.

El **objetivo** del juego es **ubicar cada caja sobre un depósito**.

---

[1] Se puede jugar online, por ejemplo en https://sokoban.info/

Algunas aclaraciones

(1) **la grilla es infinita**, pero solo puede haber un **número finito de paredes, cajas y depósitos**

(2) **no** puede haber **dos cajas**, ni **dos paredes**, ni **dos depósitos** en una **misma celda**

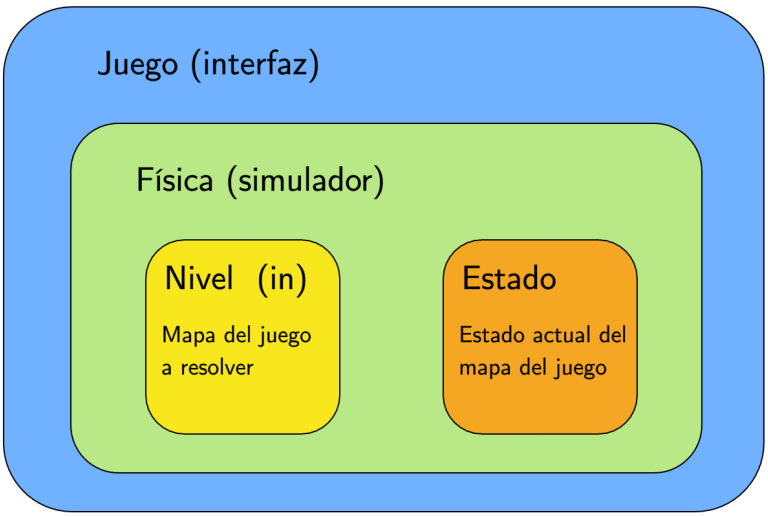
(3) una **caja** no puede estar en la **misma celda** que una **pared**

(4) un **depósito** no puede estar en la **misma celda** que una **pared**

(5) la **persona** no puede estar en la **misma celda** que una **caja** ni que una **pared**

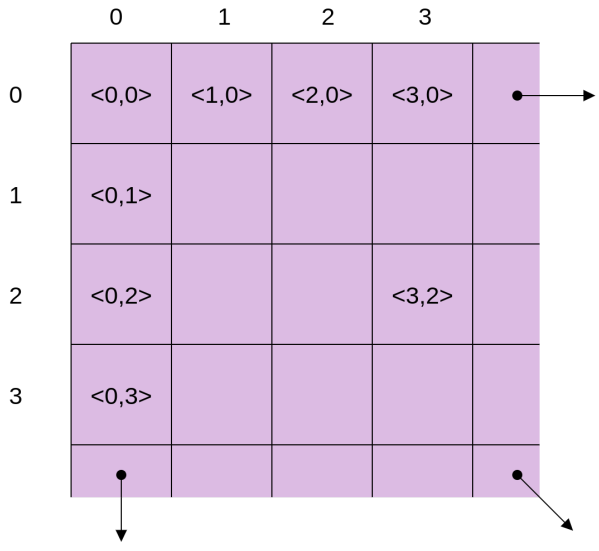
Diagramas

Jerarquía de los TADs usados



(in) : es parámetro de entrada

Disposición de grilla



TAD Posición **es** tupla(nat, nat)
TAD Personaje **es** Posición
TAD Caja **es** Posición
TAD Depósito **es** Posición
TAD Pared **es** Posición

TAD Nivel **es** Diccionario(String, conj(Posición))
TAD Estado **es** Diccionario(String, conj(Posición))
TAD Acción **es** Enum { Arriba, Abajo, Izquierda, Derecha }

TAD Física(Nivel)

géneros física

exporta observadores, generadores, nivelVálido?

usa Bool, Nat, Conjunto, Posición, Personaje, Caja, Depósito, Pared, Nivel, Estado, Acción, Física

igualdad observacional

$$(\forall f1, f2 : \textbf{física}) \; (f1 =_{obs} f2 \iff ((\forall n : \textbf{Nivel}) \; (computarEstado(f1(n)) =_{obs} computarEstado(f2(n)))))$$

observadores básicos

computarEstado : física → Estado

generadores

crearFísica : Nivel n → física { nivelVálido?(n) }
simularAcción : física × Acción → física

otras operaciones

personaje : física → personaje
cajas : física → conj(caja)
depósitos : física → conj(depósito)
paredes : física → conj(pared)

accionarPersonaje : física × tupla(nat, nat) → personaje
accionarCaja : física × tupla(nat, nat) → conj(caja)
moverCaja : Posición × conj(Caja) → tupla(nat, nat)

nivelVálido? : Nivel → bool
esPared? : conj(Pared) × Posición → bool
esCaja? : conj(Caja) × Posición → bool
estáLibre? : Estado × Posición → bool
seSolapan? : conj(Posición) × conj(Posición) → bool

```
    ningunoDeXenY     : conj(Posición)  × conj(Posición)  → bool
    sumar2Tuplas      : tupla(nat, nat) × tupla(nat, nat) → tupla(nat, nat)
    sumar3Tuplas      : tupla(nat, nat) × tupla(nat, nat) × tupla(nat, nat)
                                                          → tupla(nat, nat)

axiomas
    computarEstado(crearFísica(n)) ≡ n
    computarEstado(simularAcción(f, a)) ≡
        Dict("personaje", personaje(simularAcción(f, a)),
        Dict("cajas",         cajas(simularAcción(f, a)),
        Dict("paredes",     paredes(simularAcción(f, a)),
        Dict("depósitos", depósitos(simularAcción(f, a)),
        vacío))))

    -- Leo datos del Nivel de entrada
    personaje(crearFísica(n)) ≡ dameUno(n["personaje"]) -- es único
    cajas(crearFísica(n))     ≡ n["cajas"]
    depósitos(crearFísica(n)) ≡ n["depósitos"]
    paredes(crearFísica(n))   ≡ n["paredes"]

    -- Estos objetos nunca se mueven
    paredes(simularAcción(f, a))   ≡ paredes(f)
    depósitos(simularAcción(f, a)) ≡ depósitos(f)

    personaje(simularAcción(f, a)) ≡
                if a = Arriba then
                    accionarPersonaje(f, ⟨0,-1⟩ )
                else if a = Abajo then
                    accionarPersonaje(f, ⟨0, 1⟩ )
                else if a = Izquierda then
                    accionarPersonaje(f, ⟨-1, 0⟩ )
                else if a = Derecha then
                    accionarPersonaje(f, ⟨1, 0⟩ )
                else fi fi fi fi

    cajas(simularAcción(f, a)) ≡
                if a = Arriba then
                    accionarCaja(f,  ⟨0,-1⟩ )
                else if a = Abajo then
                    accionarCaja(f,  ⟨0, 1⟩ )
                else if a = Izquierda then
                    accionarCaja(f,  ⟨-1, 0⟩ )
                else if a = Derecha then
                    accionarCaja(f,  ⟨1, 0⟩ )
                else fi fi fi fi

    -- funciones auxiliares
```

```
accionarPersonaje : física × tupla(nat, nat) → personaje
accionarPersonaje(f, paso) ≡
    if estáLibre?( computarEstado(f), sumar2Tuplas(personaje(f), paso) ) then
        -- me muevo
        sumar2Tuplas(personaje(f), paso)
    else
        if esPared?( paredes(f), sumar2Tuplas(personaje(f), paso) ) then
            -- no se mueve, yo tampoco
            personaje(f)
        else
            -- esCaja! Veo si se puede mover
            if estáLibre?( computarEstado(f),
                           sumar3Tuplas(personaje(f), paso, paso) ) then
                sumar2Tuplas(personaje(f), paso)
            else
                -- la caja no se mueve, yo tampoco
                personaje(f)
            fi
        fi
    fi

accionarCaja : física × tupla(nat, nat) → conj(caja)
accionarCaja(f, paso) ≡
    if esCaja?( sumar2Tuplas(personaje(f), paso) ) then
        if estáLibre?( computarEstado(f),
                       sumar3Tuplas(personaje(f), paso, paso) ) then
            moverCaja( sumar2Tuplas(personaje(f), paso) )
        else
            cajas(f)
        fi
    else
        -- No me interesan las no-cajas en esta función
    fi

moverCaja : Posición × conj(Caja) → tupla(nat, nat)
moverCaja(pos, cajas, paso) ≡
    if dameUno(cajas) = pos then
        Ag(sumar2Tuplas(pos, paso), sinUno(cajas))
    else
        Ag(dameUno(cajas), moverCaja(pos, sinUno(cajas), paso))
    fi

-- Validación y Preguntas sobre elementos en posición <x, y>
esPared? : conj(Pared) × Posición → bool
esPared?(c, p) ≡ if Ø?(c) then
                    false
                else
```

```
                    if dameUno(c) = p then
                        true
                    else
                        esPared?(sinUno(c), p)
                    fi
                fi

esCaja? : conj(Caja) × Posición → bool
esCaja?(c, a) ≡ if Ø?(c) then
                    false
                else
                    if dameUno(c) = a then
                        true
                    else
                        esCaja?(sinUno(c), a)
                    fi
                fi

estáLibre? : Estado × Posición → bool
estáLibre?(e, p) ≡ ¬esPared?(e["paredes"], p) ∧ ¬esCaja?(e["cajas"], p)

nivelVálido? : Nivel → bool
nivelVálido?(n) ≡ def?("personaje", n)    ∧
                  def?("cajas", n)        ∧
                  def?("paredes", n)      ∧
                  def?("depósitos", n)    ∧
                  #(claves(n)) = 4        ∧L
                  -- requerimientos del enunciado
                  #(n["personaje"]) = 1                          ∧
                  ¬ dameUno(n["personaje"]) ∈ n["cajas"]         ∧
                  ¬ dameUno(n["personaje"]) ∈ n["paredes"]       ∧
                  n["cajas"] = n["depósitos"]                    ∧
                  ¬ seSolapan?(n["paredes"], n["cajas"])         ∧
                  ¬ seSolapan?(n["paredes"], n["depósitos"])

seSolapan? : conj(Posición) × conj(Posición) → bool
seSolapan?(a, b) ≡ ningunoDeXenY(a, b) ∧ ningunoDeXenY(b, a)

ningunoDeXenY : conj(Posición) × conj(Posición) → bool
ningunoDeXenY(X, Y) ≡ if Ø?(X) ∨ Ø?(Y) then
                        true
                    else
                        if dameUno(X) ∈ Y then
                            false
                        else
                            ningunoDeXenY(sinUno(X), Y)
                        fi
```

fi

-- **Obs:** Algo 'free' que vino con elegir conjuntos distintos para cada objeto del mapa es que **no** puede haber dos elementos repetidos, y por ende, dos elementos del mismo tipo en la misma posición.

sumar2Tuplas : tupla(nat, nat) × tupla(nat, nat) → tupla(nat, nat)
sumar2Tuplas(a, b) ≡ < π_1(a) + π_1(b) , π_2(a) + π_2(b) >

sumar3Tuplas : tupla(nat, nat) × tupla(nat, nat) × tupla(nat, nat) → tupla(nat, nat)
sumar3Tuplas(a, b, c) ≡ < π_1(a) + π_1(b) + π_1(c) , π_2(a) + π_2(b) + π_2(c) >

Fin TAD

TAD Juego(Nivel)

géneros juego

exporta observadores, generadores, completado?

usa Bool, Nat, Nivel, Estado, Acción, Física

igualdad observacional

(∀j1,j2 : **juego**)
(j1 $=_{obs}$ j2 ⇐⇒ ((∀n : **Nivel**)(estado(j1(n)) $=_{obs}$ estado(j2(n)))))

observadores básicos

estado : juego → Estado

generadores

iniciar : Nivel n → juego { nivelVálido?(n) }
step : juego × Acción → juego

otras operaciones

completado? : juego → bool
todoEnOrden : Estado → bool

axiomas

estado(iniciar(n)) ≡ computarEstado(crearFísica(n)) -- uso de Física
estado(step(j, a)) ≡ computarEstado(simularAcción(a)) -- uso de Física

completado?(iniciar(n)) ≡ todoEnOrden(n)
completado?(step(j, a)) ≡ todoEnOrden(estado(step(j, a)))

```
-- Estado ≡ Diccionario con 4 keys { String : Conjunto(Posición) }
todoEnOrden(e) ≡ e[“cajas”] = e[“depósitos”]
```

Fin TAD