



DEPARTAMENTO  
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# TP 1: Diseño

SimCity

1 de Junio de 2022

Algoritmos y Estructuras de Datos II

**Grupo 01 - hasTADlaVista, turno mañana**

Integrante	LU	Correo electrónico
Lakowsky, Manuel	511/21	mlakowsky@gmail.com
Vekselman, Natán	338/21	natanvek11@gmail.com
Arienti, Federico	316/21	fa.arianti@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>

## Resumen

El siguiente trabajo busca desarrollar el diseño de algunas de las estructuras de datos centrales al juego SimCity de 1989. Se buscará modelar e implementar el TAD *SimCity*, y los TADs satélite *Mapa* y *Servidor*, en concordancia con las interpretaciones y restricciones consignadas. En cada caso, se detallarán las motivaciones detrás de las soluciones propuestas.

Un SimCity consistirá, en términos generales, de un *Mapa* y un conjunto de *Construcciones* de tipo *casa* o *comercio*. El mismo permitirá la *union* con otras instancias del tipo, y deberá permitir conocer el *turno* y la *popularidad* de la partida, entendido éste último atributo como la cantidad de uniones que componen a la instancia.

## Índice

<b>1. Especificación</b>	<b>2</b>
1.1. Aliases . . . . .	2
1.2. Mapa . . . . .	2
1.3. SimCity . . . . .	3
1.4. Servidor . . . . .	6
<b>2. Diseño</b>	<b>8</b>
2.1. Módulo Mapa . . . . .	8
2.1.1. Interfaz . . . . .	8
2.1.2. Representación . . . . .	8
2.1.3. Implementación . . . . .	9
2.2. Módulo SimCity . . . . .	10
2.2.1. Interfaz . . . . .	10
2.2.2. Representación . . . . .	12
2.2.3. Implementación . . . . .	16
2.3. Módulo Servidor . . . . .	19
2.3.1. Interfaz . . . . .	19
2.3.2. Representación . . . . .	22
2.3.3. Implementación . . . . .	23

# 1. Especificación

## 1.1. Aliases

**TAD** POS es TUPLA<X: NAT × Y: NAT>

**TAD** CONSTRUCCIÓN es STRING

**TAD** NOMBRE es STRING

**TAD** NIVEL es NAT

## 1.2. Mapa

**TAD** MAPA

**géneros** Mapa

**exporta** Mapa, observadores, generadores,  $\bullet + \bullet$ , esRio

**usa** Nat, conj( $\alpha$ ), Pos, Bool

**igualdad observacional**

$$(\forall m, m' : \text{Mapa}) \left( m =_{\text{obs}} m' \iff \left( \begin{array}{l} \text{horizontales}(m) =_{\text{obs}} \text{horizontales}(m') \wedge_L \\ \text{verticales}(m) =_{\text{obs}} \text{verticales}(m') \end{array} \right) \right)$$

**observadores básicos**

horizontales : Mapa  $\longrightarrow$  conj(Nat)

verticales : Mapa  $\longrightarrow$  conj(Nat)

**generadores**

crear : conj(Nat) × conj(Nat)  $\longrightarrow$  Mapa

**otras operaciones**

$\bullet + \bullet$  : Mapa × Mapa  $\longrightarrow$  Mapa

esRio : Mapa × Pos  $\longrightarrow$  Bool

**axiomas**  $\forall hs, vs: \text{conj}(\text{Nat})$

horizontales(crear( $hs, vs$ ))  $\equiv hs$

verticales(crear( $hs, vs$ ))  $\equiv vs$

**otros ax.**  $\forall m1, m2: \text{Mapa}, \forall p: \text{Pos}$

$m1 + m2 \equiv \text{crear}(\text{horizontales}(m1) \cup \text{horizontales}(m2), \text{verticales}(m1) \cup \text{verticales}(m2))$

$\text{esRio}(m1, p) \equiv p.x \in \text{verticales}(m1) \vee p.y \in \text{horizontales}(m1)$

**Fin TAD**



```

turnos(iniciar(m))           ≡ 0
turnos(avanzarTurno(s, cs)) ≡ turnos(s) + 1
turnos(unir(s, s'))        ≡ if turnos(s) < turnos(s') then turnos(s') else turnos(s) fi

otros ax.    ∀s: simcity, ∀p, q: Pos, ∀cs, cs': dicc(Pos × Construcción), ∀cn: dicc(Pos × Nivel),
              ∀d, d': dicc(α × β)

nivelComercio(p, cn)      ≡ if vacio?(claves(cn)) then
                             1
                             else if distManhattan(p, proximo) ≤ 3 then
                                 max(obtener(proximo, cn), nivelComercio(p, borrar(proximo, cn)))
                             else
                                 nivelComercio(p, borrar(proximo, cn))
                             fi
                             donde proximo ≡ dameUno(claves(cn))

distManhattan(p, q)      ≡ if p.x < q.x then q.x - p.x else p.x - q.x fi
                             +
                             if p.y < q.y then q.y - p.y else p.y - q.y fi

d ∪dicc d'              ≡ if vacio?(claves(d')) then
                             d
                             else
                                 definir(proximo, obtener(proximo, d'), d ∪dicc borrar(proximo, d'))
                             fi
                             donde proximo ≡ dameUno(claves(d'))

construcciones(s)          ≡ casas(s) ∪dicc comercios(s)
agCasas(cn, cs)          ≡ if vacio?(claves(cs)) then
                             cn
                             else if obtener(proximo, cs) = "casa" then
                                 agCasas(definir(proximo, 1, cn), borrar(proximo, cs))
                             else
                                 agCasas(cn, borrar(proximo, cs))
                             fi
                             donde proximo ≡ dameUno(claves(cs))

agComercios(s, cn, cs)   ≡ if vacio?(claves(cs)) then
                             cn
                             else if obtener(proximo, cs) = "comercio" then
                                 agComercios(s, definir(proximo, nivelComercio(proximo, casas(s)), cn),
                                             borrar(proximo, cs))
                             else
                                 agComercios(s, cn, borrar(proximo, cs))
                             fi
                             donde proximo ≡ dameUno(claves(cs))

sacarRepes(cs, cs')      ≡ if vacio?(claves(cs)) then
                             cs'
                             else if def?(proximo, cs') then
                                 sacarRepes(borrar(proximo, cs), borrar(proximo, cs'))
                             else
                                 sacarRepes(borrar(proximo, cs), cs')
                             fi
                             donde proximo ≡ dameUno(claves(cs))

```

**Fin TAD**

## Definiciones Auxiliares de SimCity

avanzarValido : SimCity  $s \times \text{dicc}(\text{Pos} \times \text{Construcción}) \text{ cs} \longrightarrow \text{boolean}$

$$\begin{aligned} \text{avanzarValido}(s, cs) \equiv & \neg \text{vacío?}(\text{claves}(cs)) \wedge \\ & (\forall p : \text{Pos}) (\text{def?}(p, cs) \Rightarrow_{\text{L}} \\ & \quad (\neg p \in \text{claves}(\text{construcciones}(s)) \wedge \\ & \quad \neg \text{esRio}(p, \text{mapa}(s)) \wedge \\ & \quad (\text{obtener}(p, cs) = \text{"casa"} \vee \text{obtener}(p, cs) = \text{"comercio"})) \\ & ) \end{aligned}$$

unirValido : Simcity  $a \times \text{SimCity } b \longrightarrow \text{boolean}$

$$\begin{aligned} \text{unirValido}(a, b) \equiv & (\forall p : \text{Pos})(\text{def?}(p, \text{construcciones}(a)) \Rightarrow_{\text{L}} \\ & \quad \neg \text{esRio}(p, \text{mapa}(b)) \wedge \\ & \quad (\nexists otra : \text{Pos})(\text{def?}(otra, \text{construcciones}(a)) \wedge_{\text{L}} \\ & \quad \quad \text{obtener}(otra, \text{construcciones}(a)) > \text{obtener}(p, \text{construcciones}(a)) \\ & ) \Rightarrow_{\text{L}} \neg \text{def?}(p, \text{construcciones}(b)) \\ & ) \wedge \\ & (\forall p : \text{Pos})(\text{def?}(p, \text{construcciones}(b)) \Rightarrow_{\text{L}} \\ & \quad \neg \text{esRio}(p, \text{mapa}(a)) \wedge \\ & \quad (\nexists otra : \text{Pos})(\text{def?}(otra, \text{construcciones}(b)) \wedge_{\text{L}} \\ & \quad \quad \text{obtener}(otra, \text{construcciones}(b)) > \text{obtener}(p, \text{construcciones}(b)) \\ & ) \Rightarrow_{\text{L}} \neg \text{def?}(p, \text{construcciones}(a)) \\ & ) \end{aligned}$$



$\text{pendientes}(\text{nuevaPartida}(s, p, m), p') \equiv \text{if } p = p' \text{ then vacio else pendientes}(s, p') \text{ fi}$   
 $\text{pendientes}(\text{unirPartidas}(s, p, p'), p'') \equiv \text{pendientes}(s, p'')$  *acá no hace falta dif casos no?*  
 $\text{pendientes}(\text{agregarCasa}(s, p, pos), p') \equiv \text{if } p = p' \text{ then}$   
 $\quad \text{definir}(pos, "casa", \text{pendientes}(s, p))$   
 $\quad \text{else}$   
 $\quad \text{pendientes}(s, p')$   
 $\quad \text{fi}$   
 $\text{pendientes}(\text{agregarComercio}(s, p, pos), p') \equiv \text{if } p = p' \text{ then}$   
 $\quad \text{definir}(pos, "comercio", \text{pendientes}(s, p))$   
 $\quad \text{else}$   
 $\quad \text{pendientes}(s, p')$   
 $\quad \text{fi}$   
 $\text{pendientes}(\text{avanzarTurnoPartida}(s, p), p') \equiv \text{if } p = p' \text{ then vacio else pendientes}(s, p') \text{ fi}$

**otros ax.**  $\forall s: \text{Server}, \forall p: \text{Nombre}$

$\text{verMapa}(s, p) \equiv \text{mapa}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{verCasas}(s, p) \equiv \text{casas}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{verComercios}(s, p) \equiv \text{comercios}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{verPopularidad}(s, p) \equiv \text{popularidad}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{verTurno}(s, p) \equiv \text{turnos}(\text{obtener}(p, \text{partidas}(s)))$

**Fin TAD**

## Definiciones Auxiliares de Servidor

$\text{unionValida} : \text{Server } s \times \text{Nombre } p \times \text{Nombre } p' \longrightarrow \text{boolean}$

$\text{unionValida}(s, p, p') \equiv \text{def?}(p, \text{partidas}(s)) \wedge \text{def?}(p', \text{partidas}(s)) \wedge p \notin \text{congeladas}(s) \wedge_{\text{L}}$   
 $\quad \text{vacío?}(\text{claves}(\text{pendientes}(s, p))) \wedge \text{vacío?}(\text{claves}(\text{pendientes}(s, p'))) \wedge$   
 $\quad \text{unirValido}(\text{obtener}(p, \text{partidas}(s)), \text{obtener}(p', \text{partidas}(s)))^1$

$\text{avanzarValido} : \text{Server } s \times \text{Nombre } p \longrightarrow \text{boolean}$

$\text{avanzarValido}(s, p) \equiv \text{def?}(p, \text{partidas}(s)) \wedge p \notin \text{congeladas}(s) \wedge_{\text{L}}$   
 $\quad \neg \text{vacío?}(\text{claves}(\text{pendientes}(\text{obtener}(p, \text{partidas}(s)))))$

$\text{agregarValido} : \text{Server } s \times \text{Nombre } p \times \text{Pos } pos \longrightarrow \text{boolean}$

$\text{agregarValido}(s, p, pos) \equiv \text{def?}(p, \text{partidas}(s)) \wedge p \notin \text{congeladas}(s) \wedge \neg \text{def?}(pos, \text{pendientes}(s, p)) \wedge$   
 $\quad \neg \text{def?}(pos, \text{verCasas}(s, p)) \wedge \neg \text{def?}(pos, \text{verComercios}(s, p)) \wedge$   
 $\quad \neg \text{esRio}(pos, \text{verMapa}(s, p))$

1. definido en el apartado Definiciones Auxiliares de SimCity.



## 2. Diseño

### 2.1. Módulo Mapa

#### 2.1.1. Interfaz

#### Interfaz

**usa:** se requiere?

**exporta:** se requiere?

**se explica con:** MAPA

**géneros:** Mapa

#### Operaciones básicas de Mapa

no se si hace falta agregar para los obs?

**CREAR**(**in**  $hs: \text{conj}(\text{Nat})$ , **in**  $vs: \text{conj}(\text{Nat})$ )  $\rightarrow res: \text{Mapa}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{mapa}(\hat{h}s, \hat{v}s)\}$

**Complejidad:**  $O(\text{copy}(hs) + \text{copy}(vs))$  puede ser theta?

**Descripción:** Crea un mapa.

**ESRIO**(**in**  $m: \text{Mapa}$ , **in**  $p: \text{Pos}$ )  $\rightarrow res: \text{Bool}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{esRio}(\hat{m}, \hat{p})\}$

**Complejidad:**  $O(1)$  como rep los  $\text{conj}(\text{Nat})$ ?, si es sobre un array creo que hay q implementarlo. Tmb podria ser theta creo

**Descripción:** Verifica si en determinada pos hay un río.

**SUMA**(**in**  $m1: \text{Mapa}$ , **in**  $m2: \text{Mapa}$ )  $\rightarrow res: \text{Mapa}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \hat{m}1 + \hat{m}2\}$

**Complejidad:**  $O(\text{crear}(m1) + \text{crear}(m2))$

**Descripción:** Une dos Mapas.

#### 2.1.2. Representación

### Representación

#### Representación de mapa

Un mapa contiene ríos infinitos horizontales y verticales. Los ríos se representan como conjuntos lineales de naturales que indican la posición en los ejes cartesianos de los ríos.

Mapa **se representa con**  $\text{estr}$

donde  $\text{estr}$  es  $\text{tupla}(\text{horizontales: } \text{conj}(\text{Nat}), \text{verticales: } \text{conj}(\text{Nat}))$

$\text{Rep} : \text{estr} \rightarrow \text{boolean}$

$\text{Rep}(e) \equiv \text{true} \iff \text{true}$

$\text{Abs} : \text{estr } m \rightarrow \text{Mapa}$

$\{\text{Rep}(m)\}$

$\text{Abs}(m) \equiv \text{horizontales}(m) = \text{estr.horizontales} \wedge \text{verticales}(m) = \text{estr.verticales}$  igobs?

**2.1.3. Implementación****Algoritmos**

aca estaria bueno tenerlo mas cercano al formato de los de servidor, o pasar los de servidor a este formato.  
 No se si hace falta declarar el tipo de variables nuevas  
 habria que agregar justificaciones?

---

```
crear(in hs : conj(Nat), in vs : conj(Nat)) → res : estr
1:   estr.horizontales ← hs
2:   estr.verticales ← vs
3:   return estr
Complejidad: O(copy(hs) + copy(vs))
```

---



---

```
Suma(in m1: mapa, in m2: mapa) → res : mapa
1:   mapa res ← copiar(m1)
2:
3:   itConj(Nat) itH ← crearIt(horizontales(m2))
4:   while(haySiguiente(itH)) :
5:     Ag(horizontales(res), siguiente(itH))
6:     avanzar(itH)
7:
8:   itConj(Nat) itV ← crearIt(vertaicales(m2))
9:   while(haySiguiente(itV)) :
10:    Ag(vertaicales(res), siguiente(itV))
11:    avanzar(itV)
12:
13:   return res
Complejidad: O(copiar(m1) + copiar(m2))
```

---



---

```
esRio(in m1: mapa, in p: Pos) → res : Bool
1:   return p.x ∈ verticales(m1) ∨ p.y ∈ horizontales(m1)
Complejidad: O(#horizontales(m1) + #verticales(m1))
```

---

## 2.2. Módulo SimCity

### 2.2.1. Interfaz

#### Interfaz

hace falta restringir las complejidades en los casos que no pedían?, esta bien restringir en terminos de theta?  
Onda estaría diciendo que tampoco puede ser mejor la solución

**usa:** Mapa, Diccionario Lineal, Pos, Nivel, Nat

**exporta:** todo

**se explica con:** SIMCITY

**géneros:** SimCity

#### Operaciones básicas de SimCity

Sea  $S$ : SimCity,  $N = \text{popularidad}(S)$ ,  $\{u_0 \dots u_N\} = U$ : el conjunto de SimCities en union con  $S^1$  y  $S$ ,  $\text{casas} = \#(\text{claves}(\text{casas}(\hat{S})))$  y  $\text{comercios} = \#(\text{claves}(\text{comercios}(\hat{S})))$ .

**MAPA**(in  $S$ : SimCity)  $\rightarrow res$ : Mapa

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{mapa}(\hat{S})\}$

**Complejidad:**  $\Theta(\sum_{i=0}^N (\text{mapa}_i))$ , donde  $\text{mapa}_i$  es el Mapa original<sup>2</sup> de  $u_i$ .

**Descripción:** Retorna el mapa sobre el que se desarrolla el juego actual.

**Aliasing:** No. Genera una copia.

**CASAS**(in  $S$ : SimCity)  $\rightarrow res$ : DiccLineal(Pos, Nivel)

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{casas}(\hat{S})\}$

**Complejidad:**  $O(\text{casas}^2) \supset \Theta(\sum_{i=0}^N (\text{hasta}_i \times \text{casas}_i + \text{camino}_i))$ ,  
donde  $\text{hasta}_i$  y  $\text{casas}_i$  son respectivamente la cantidad de casas definidas<sup>3</sup> en  $\{u_0 \dots u_{i-1}\}$  y  $u_i$ <sup>4</sup>,  
y  $\text{camino}_i$  representa la cantidad de uniones para llegar de  $S$  a  $u_i$ <sup>5,6</sup>.

**Descripción:** Retorna las posiciones y respectivos niveles de todas las casas en el juego actual.

**Aliasing:** No. Genera una copia.

**COMERCIOS**(in  $S$ : SimCity)  $\rightarrow res$ : DiccLineal(Pos, Nivel)

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{comercios}(\hat{S})\}$

**Complejidad:**  $O(\text{comercios}^2 \times \text{casas}) \supset \Theta(\sum_{i=0}^N (\text{hasta}_i \times \text{comercios}_i \times \text{casas} + \text{camino}_i))$ ,  
donde  $\text{hasta}_i$  y  $\text{comercios}_i$  son respectivamente la cantidad de comercios definidos en  $\{u_0 \dots u_{i-1}\}$  y  $u_i$ ,  
y  $\text{camino}_i$  representa la cantidad de uniones para llegar de  $S$  a  $u_i$ <sup>7</sup>.

**Descripción:** Retorna las posiciones y respectivos niveles de todos los comercios en el juego actual.

**Aliasing:** No. Genera una copia.

**POPULARIDAD**(in  $S$ : SimCity)  $\rightarrow res$ : Nat

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{popularidad}(\hat{S})\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** Retorna la cantidad total de uniones que se realizaron para conformar la partida actual.

1. Este conjunto incluye también a los SimCities provenientes de las uniones propias a cada SimCity en unión directa con  $S$ .
2. Es decir, aquel con el que se inició originalmente el simCity.
3. Donde se entiende por 'definida' como aquellas casas que provienen del propio simCity y no de alguna de sus uniones.
4. Dado que consideramos la resolución de colisiones durante una unión válida como 'queda el primero', y se requiere una complejidad de  $\Theta(1)$  para la unión, es esperable que crear una copia del conjunto total de casas en  $U$  requiera chequear para cada casa definida en  $u_i$  su pertenencia al resultado parcial de la copia. Donde, en el peor caso, equivale al total de casas definidas hasta entonces.
5. Entendiendo las relaciones en  $U$  como un rosetree con  $\text{raiz} = S$  y la necesidad de inmutabilidad de cada  $u_i \neq S$ , es razonable considerar que el nivel de cada casa o comercio en  $u_i$  va a tener que calcularse en relación con su posición en el árbol.
6. Se proveen dos complejidades, una más abstracta y una evaluada en consideración de las representaciones posibles dadas las restricciones impuestas.
7. Similar a  $\text{casas}(S)$ . En este caso se agrega la posibilidad de tener que evaluar por pertenencia en el total de las casas al conjunto de posiciones a distancia manhattan  $\leq 3$  del comercio actualmente siendo copiado, para conocer su nivel.

**TURNOS**(**in**  $S$ : SimCity)  $\rightarrow res$  : Nat

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{turnos}(\hat{S})\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** Retorna la cantidad de turnos que pasaron desde que se inició el SimCity.

**INICIAR**(**in**  $m$ : Mapa)  $\rightarrow res$  : SimCity

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{iniciar}(\hat{m})\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** Crea un nuevo SimCity.

**Aliasing:** Se guarda una referencia a  $m$  en  $res$ . No se modifica.

**AVANZARTURNO**(**in/out**  $S$ : SimCity, **in**  $cs$ : diccLineal(Pos, Construcccion))

**Pre**  $\equiv \{\text{avanzarValido}^2(\hat{s}, \hat{c}s) \wedge \hat{S} = S_0\}$

**Post**  $\equiv \{\hat{S} =_{\text{obs}} \text{avanzarTurno}(S_0, \hat{c}s)\}$

**Complejidad:**  $O(\text{casas} \times \#(\text{claves}(\hat{c}s)) + N) \supset \Theta(\text{casas}_S \times \#(\text{claves}(\hat{c}s)) + U_S)$ ,  
donde  $\text{casas}_S$  es el conjunto de casas definidas en éste SimCity particular y  $U_S$  es el conjunto de uniones directas a  $S$ .<sup>1</sup>

**Descripción:** Avanza el turno de un SimCity.

**Aliasing:** Genera una copia de las posiciones en el diccionario.

**UNIR**(**in/out**  $S1$ : SimCity, **in**  $S2$ : SimCity)

**Pre**  $\equiv \{\text{unionValida}^2(\hat{S}1, \hat{S}2) \wedge \hat{S}1 = S_0\}$

**Post**  $\equiv \{\hat{S}1 =_{\text{obs}} \text{unir}(S_0, \hat{S}2)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** Une dos SimCities.

**Aliasing:** Se guarda una referencia a  $S2$  en  $S1$ . Cualquier cambio sobre  $S2$  modificará a  $S1$ .

1. Esto se debe a que avanzar turno requiere agregar  $\#(\text{claves}(\hat{c}s))$  elementos a un diccionario lineal y, nuevamente en relación al cálculo de niveles, requiere al menos avanzar los niveles propios e, indirectamente, los de todos los simCities en unión directa.  
2. definido en Definiciones Auxiliares de SimCity.

### 2.2.2. Representación

## Representación

Un *SimCity* se compone por la *ubicacion* y *nivel* de una serie de *construcciones*, de tipo *Casa* o *Comercio*, sobre un *Mapa*, y de una *Popularidad* respecto a la cantidad de uniones que lo modificaron.

La ubicación de las casas se representan sobre un diccionario lineal con clave *Pos* y significado *Nivel*. La ubicación de los comercios se representan similarmente, pero su significado responde a un *NivelBase* de tipo *Nat* a partir del cual se calcula propiamente su *Nivel*. El mapa es de tipo *Mapa* y las uniones se representan a través de una *lista* de *Hijos* que contiene punteros a los *SimCities* unidos e información relevante para calcular el nivel de sus construcciones. Ya que, una vez unido a otro, un *SimCity* debe permanecer sin modificar.

*SimCity* se representa con *estr*

donde *estr* es tupla(*turno* : Nat,  
*popularidad* : Nat,  
*mapa* : Mapa,  
*casas* : diccLineal(*pos*, *Nivel*) ,  
*comercios* : diccLineal(*pos*, *NivelBase*) ,  
*uniones* : lista(*hijo*) )

donde *hijo* es tupla(*sc* : puntero(*estr*) ,  
*turnosDesdeUnion* : Nat )

donde *pos* es tupla(*x* : Nat , *y* : Nat )

*Rep* : *estr*<sup>1</sup>  $\longrightarrow$  boolean

*Rep*(*e*)  $\equiv$  true  $\iff$  (  
 $(\forall h : \text{hijo})(\text{esta?}(h, e.\text{uniones}) \Rightarrow_L$   
 $h.sc \neq \text{NULL} \wedge_L \text{rep}(*h.sc)^2 \wedge h.sc \notin \text{unirPunteros}(\text{remove}(p, e.\text{uniones}))^3 \wedge_L$   
 $(e.\text{turno} \geq h.\text{turnosDesdeUnion})^4 \wedge$   
 $(\forall h_2 : \text{hijo})(\text{esta?}(h_2, e.\text{uniones}) \wedge_L \text{pos}(h_2, e.\text{uniones}) > \text{pos}(h, e.\text{uniones}) \Rightarrow_L$   
 $h_2.\text{turnosDesdeUnion} \leq h.\text{turnosDesdeUnion}$   
 $)^5$   
 $) \wedge_L$   
 $(\&e \notin \text{Unidos})^6 \wedge_L$   
 $(e.\text{popularidad} = \#(\text{Unidos}))^7 \wedge$   
 $(\forall p : \text{puntero}(\text{estr}))(p \in \text{Unidos} \Rightarrow_L$   
 $e.\text{turno} \geq (*p).\text{turno}$   
 $)^8 \wedge$   
 $(\forall p : \text{Pos})(p \in \text{claves}(e.\text{casas}) \Rightarrow_L$   
 $\neg \text{def}(p, e.\text{comercios})^9 \wedge \neg \text{esRio}(p, \text{Mapas})^{10} \wedge (\text{obtener}(p, e.\text{casas}) < e.\text{turno})^{11}$   
 $) \wedge$   
 $(\forall p : \text{pos})(p \in \text{claves}(e.\text{comercios}) \Rightarrow_L$   
 $\neg \text{def}(p, e.\text{casas})^{12} \wedge \neg \text{esRio}(p, \text{Mapas})^{13} \wedge (\text{obtener}(p, e.\text{comercios}) < e.\text{turno})^{14}$   
 $) \wedge_L$   
 $\text{unionesValidas}(e, e.\text{uniones})^{15}$   
 $)$

donde

*Unidos*  $\equiv$  unirPunteros(*e.uniones*)  
*Mapas*  $\equiv$  *e.mapa* + unirMapas(*Unidos*)

1. Se asume el traspaso de toda estructura de representación a su equivalente abstracto (se aplica el sombrerito).
2. Cada hijo apunta a un *Simcity* válido.
3. El puntero de cada hijo no aparece en ningún otro *SimCity* de la unión.
4. El turno es mayor o igual a la cantidad de turnos que pasaron desde la unión.
5. Las uniones están ordenadas de más antiguas a más recientes.
6. La estructura no loopea consigo misma.
7. La popularidad es igual a la cantidad de uniones.
8. El turno actual es mayor o igual al turno de cualquier *simCity* hijo.

$Abs : \text{estr } e \longrightarrow \text{SimCity} \quad \{\text{Rep}(e)\}$   
 $Abs(e) \equiv sc : \text{SimCity} \mid$   
 $\quad \text{mapa}(sc) =_{\text{obs}} \text{Mapas} \wedge$   
 $\quad \text{casas}(sc) =_{\text{obs}} \text{nivelar}(\text{Casas}) \wedge$   
 $\quad \text{comercios}(sc) =_{\text{obs}} \text{nivelar}(\text{Comercios}) \wedge$   
 $\quad \text{popularidad}(sc) =_{\text{obs}} e.\text{popularidad}$   
 donde  
 $\text{Unidos} \equiv \text{unirPunteros}(e.\text{uniones})$   
 $\text{Casas} \equiv \text{unirCasas}(\text{Ag}(\&e, \text{Unidos}))$   
 $\text{Comercios} \equiv \text{unirComercios}(\text{Ag}(\&e, \text{Unidos}))$   
 $\text{Mapas} \equiv \text{unirMapas}(\text{Ag}(\&e, \text{Unidos}))$

## Definiciones Auxiliares

Los siguientes reemplazos sintácticos están contenidos al contexto del invariante de representación y la función de abstracción del SimCity. En éste sentido, se considera restricción implícita, para cada uno, ser evaluado en un estado que satisfaga parcialmente la representación,-en términos de lógica ternaria-, al momento de 'llamada' dentro de la misma.

$\text{unirPunteros} : \text{secu}(\text{hijo}) \ s \longrightarrow \text{conj}(\text{puntero}(\text{estr})) \quad \{(\forall h : \text{hijo})(h \in s \Rightarrow_L h.sc \neq \text{NULL})\}$   
 $\text{unirPunteros}(s) \equiv \_ \text{unirPunteros}(s, \emptyset)$   
  
 $\_ \text{unirPunteros} : \text{secu}(\text{hijo}) \ s \times \text{conj}(\text{puntero}(\text{estr})) \ ps \longrightarrow \text{conj}(\text{puntero}(\text{estr})) \quad \{\text{eq. unirPunteros}\}$   
 $\_ \text{unirPunteros}(s, ps) \equiv \text{if vacia?}(s) \text{ then}$   
 $\quad \quad \quad ps$   
 $\quad \text{else if prim}(s).sc \in ps \text{ then} \quad // \text{ por si hay loops}$   
 $\quad \quad \_ \text{unirPunteros}(\text{fin}(s), ps)$   
 $\quad \text{else}$   
 $\quad \quad \_ \text{unirPunteros}((*(\text{prim}(s).sc)).\text{uniones}, \text{Ag}(\text{prim}(s).sc, ps)) \cup$   
 $\quad \quad \_ \text{unirPunteros}(\text{fin}(s), \text{Ag}(\text{prim}(s).sc, ps))$   
 $\quad \quad // \text{ al unir se descarta el duplicado}$   
 $\quad \text{fi}$   
  
 $\text{unirMapas} : \text{conj}(\text{puntero}(\text{estr})) \ ps \longrightarrow \text{Mapa}$   
 $\quad \quad \quad \{(\forall p : \text{puntero}(\text{estr}))(p \in ps \Rightarrow_L (p \neq \text{NULL} \wedge_L \text{rep}(*p)))\}$   
 $\text{unirMapas}(ps) \equiv \text{if vacio?}(ps) \text{ then}$   
 $\quad \text{crear}(\emptyset, \emptyset)$   
 $\quad \text{else}$   
 $\quad \quad (*(dameUno(ps))).\text{mapa} + \text{UnirMapas}(\text{sinUno}(ps))$   
 $\quad \text{fi}$   
  
 $\text{unirCasas} : \text{conj}(\text{puntero}(\text{estr})) \ ps \longrightarrow \text{dicc}(\text{Pos} \times \text{Nivel}) \quad \{\text{eq. unirMapas}\}$   
 $\text{unirCasas}(ps) \equiv \text{if vacio?}(ps) \text{ then}$   
 $\quad \text{vacio}$   
 $\quad \text{else}$   
 $\quad \quad (*(p).\text{casas} \cup_{\text{dicc}} \text{unirCasas}(\text{sinUno}(ps)))$   
 $\quad \text{fi}$

9. Ninguna casa está en la posición de uno de los comercios de este simCity particular.
10. Ninguna casa está sobre un río perteneciente a cualquier mapa en la unión.
11. El turno es mas grande que el nivel de cualquier casa.
12. Ningún comercio está en la posición de una de las casas de este simCity particular.
13. Ningún comercio en la unión está sobre un río perteneciente a cualquier mapa en la unión.
14. El turno es mas grande que el nivel base de cualquier comercio.
15. No se solapan posiciones máximas entre esta estructura hasta el hijo 'x', descontando construcciones agregadas después de la unión, y ese hijo, para todo hijo.

$\text{unirComercios} : \text{conj}(\text{puntero}(\text{estr})) \text{ } ps \longrightarrow \text{dicc}(\text{Pos} \times \text{Nat})$  {eq. unirMapas}  
 $\text{unirComercios}(ps) \equiv \text{if } \text{vacio?}(ps) \text{ then}$   
      $\text{vacio}$   
   **else**  
      $(*p).comercios \cup_{\text{dicc}} \text{unirComercios}(\text{sinUno}(ps))$   
   **fi**

$\text{remove} : \text{secu}(\alpha) \text{ } s \times \alpha \text{ } a \longrightarrow \text{secu}(\alpha)$     // remueve la primer aparición, si hay  
 $\text{remove}(s, a) \equiv \text{if } \text{vacio?}(s) \text{ then}$   
      $\langle \rangle$   
   **else if**  $a = \text{prim}(s)$  **then**  
      $\text{fin}(s)$   
   **else**  
      $\text{prim}(s) \bullet \text{remove}(\text{fin}(s), a)$   
   **fi**

$\text{unionesValidas} : \text{estr } e \times \text{secu}(\text{hijo}) \text{ } s \longrightarrow \text{bool}$   
 $\{s \subseteq e.uniones \wedge_L (\forall h : \text{hijo})(h \in s \Rightarrow_L h.sc \neq \text{NULL} \wedge_L \text{rep}(*h.sc))\}$   
 $\text{unionesValidas}(e, s) \equiv \text{vacio?}(s) \vee_L (\text{maxcons}(e, \text{izq}) \cap \text{maxcons}(e, \text{der}) = \emptyset \wedge \text{unionesValidas}(e, \text{com}(s)))$   
 donde

$\text{com} \equiv \text{unirPunteros}(\text{com}(s))$   
 $\text{ult} \equiv \text{ult}(s) \bullet \langle \rangle$   
 $\text{casascom} \equiv \text{unirCasas}(\text{com}) \cup_{\text{dicc}} \text{filtrar}(e.casas, \text{ult}(s).turnosDesdeUnion)^1$   
 $\text{comercom} \equiv \text{unirComercios}(\text{com}) \cup_{\text{dicc}} \text{filtrar}(e.comercios, \text{ult}(s).turnosDesdeUnion)^1$   
 $\text{casasult} \equiv \text{unirCasas}(\text{ult})$   
 $\text{comerult} \equiv \text{unirComercios}(\text{ult})$   
 $\text{izq} \equiv \text{claves}(\text{casascom}) \cup \text{claves}(\text{comercom})$   
 $\text{der} \equiv \text{claves}(\text{casasult}) \cup \text{claves}(\text{comerult})$

$\text{filtrar} : \text{dicc}(\text{Pos} \times \text{Nat}) \text{ } d \times \text{Nat } n \longrightarrow \text{dicc}(\text{Pos} \times \text{Nat})$   
 $\text{filtrar}(d, n) \equiv \text{if } \text{vacio?}(d) \text{ then}$   
      $\text{vacio}$   
   **else if**  $\text{sig} \leq n$  **then**  
      $\text{filtrar}(\text{borrar}(\text{clave}, d), n)$   
   **else**  
      $\text{definir}(\text{clave}, \text{obtener}(\text{clave}, d), \text{filtrar}(\text{borrar}(\text{clave}, d), n))$   
   **fi**

donde  
 $\text{clave} \equiv \text{dameUno}(\text{claves}(d))$

$\text{maxcons} : \text{estr } e \times \text{conj}(\text{Pos}) \text{ } c \longrightarrow \text{conj}(\text{Pos})$   $\{(\forall p : \text{Pos})(p \in c \Rightarrow_L p \in \text{posiciones}(e))\}$   
 $\text{maxcons}(e, c) \equiv \_ \text{maxcons}(e, c, \emptyset, 0)$

$\_ \text{maxcons} : \text{estr } e \times \text{conj}(\text{Pos}) \text{ } c \times \text{conj}(\text{Pos}) \text{ } max \times \text{Nat } n \longrightarrow \text{conj}(\text{Pos})$  {eq. maxcons}  
 $\_ \text{maxcons}(e, c, max, n) \equiv \text{if } \text{vacio?}(c) \text{ then}$   
      $max$   
   **else if**  $\text{nivel}_i > n$  **then**  
      $\_ \text{maxcons}(e, \text{sinUno}(c), \text{Ag}(\text{pos}_i, \emptyset), \text{nivel}_i)$   
   **else if**  $\text{nivel}_i = n$  **then**  
      $\_ \text{maxcons}(e, \text{sinUno}(c), \text{Ag}(\text{pos}_i, max), n)$   
   **else**  
      $\_ \text{maxcons}(e, \text{sinUno}(c), max, n)$   
   **fi**

1. las casas o comercios de éste simCity particular con nivel o nivel base  $\leq$  turnosDesdeUnion son aquellas que se agregaron después de la unión.

donde

$pos_i \equiv dameUno(c)$   
 $nivel_i \equiv nivel(e, pos_i)$

$nivel : \text{estr } e \times \text{Pos } pos \longrightarrow \text{Nat} \quad \{p \in \text{posiciones}(e)\}$

$nivel(e, pos) \equiv \text{if } def?(pos, Casas) \text{ then}$   
      $\text{obtener}(pos, Casas) + \text{nivelesPorUnion}(e, pos)$   
   **else**  
      $\text{max}(\text{obtener}(pos, Comercios) + \text{nivelesPorUnion}(e, pos), nManhattan)$   
   **fi**

donde:

$\text{Unidos} \equiv \text{unirPunteros}(e.uniones)$   
 $\text{Casas} \equiv \text{unirCasas}(\text{Ag}(\&e, \text{Unidos}))$   
 $\text{Comercios} \equiv \text{unirComercios}(\text{Ag}(\&e, \text{Unidos}))$   
 $nManhattan \equiv \text{manhattan}(e, pos, Casas)$

$\text{nivelesPorUnion} : \text{estr } e \times \text{Pos } pos \longrightarrow \text{Nat} \quad \{\text{eq. nivel}\}$

$\text{nivelesPorUnion}(e, pos) \equiv \text{if } def?(pos, e.casas) \vee def?(pos, e.comercios) \text{ then}$   
     0  
   **else**  
      $\text{hijoCorrecto.turnosDesdeUnion} + \text{nivelesPorUnion}(\text{hijoCorrecto.sc}, pos)$   
   **fi**

donde:

$\text{hijoCorrecto} \equiv \text{llegar}(e.uniones, pos)$

$\text{llegar} : \text{secu}(\text{hijo}) s \times \text{Pos } p \longrightarrow \text{hijo} \quad \{(\forall h : \text{hijo})(\text{esta?}(h, s) \Rightarrow_L h.sc \neq \text{NULL} \wedge_L \text{rep}(*h.sc)) \wedge_L$   
      $(\exists h : \text{hijo})(\text{esta?}(h, s) \wedge_L p \in \text{posiciones}(h.sc))\}$

$\text{llegar}(s, p) \equiv \text{if } def?(pos, \text{unirCasas}(\text{hijo}_i)) \vee def?(pos, \text{unirComercios}(\text{hijo}_i)) \text{ then}$   
      $\text{prim}(s)$   
   **else**  
      $\text{llegar}(\text{fin}(s))$   
   **fi**

donde

$\text{hijo}_i \equiv \text{Ag}(\text{prim}(s).sc, \emptyset)$

$\text{manhattan} : \text{estr } e \times \text{Pos } p \times \text{Dicc}(\text{Pos} \times \text{Nat}) d \longrightarrow \text{Conj}(\text{Pos})$   
      $\{(\forall p' : \text{Pos})(p' \in \text{claves}(d) \Rightarrow_L p' \in \text{posiciones}(e))\}$

$\text{manhattan}(e, p, d) \equiv \text{if } \text{vacio?}(\text{claves}(d)) \text{ then}$   
     1  
   **else if**  $\text{distManhattan}(p, \text{proximo}) \leq 3 \wedge p \neq \text{proximo} \text{ then}$   
      $\text{max}(\text{obtener}(\text{proximo}, d) + \text{nivelesPorUnion}(e, \text{proximo}),$   
        $\text{manhattan}(p, \text{borrar}(\text{proximo}, d)))$   
   **else**  
      $\text{manhattan}(e, p, \text{borrar}(\text{proximo}, d))$   
   **fi**  
   donde  $\text{proximo} \equiv \text{dameUno}(\text{claves}(d))$

$\text{posiciones} : \text{estr } e \longrightarrow \text{dicc}(\text{Pos} \times \text{Nat})$   
      $\{(\forall h : \text{hijo})(h \in e.uniones \Rightarrow_L h.sc \neq \text{NULL} \wedge_L \text{rep}(*h.sc))\}$

$\text{posiciones}(e) \equiv \text{claves}(e.casas \cup_{\text{dicc}} \text{unirCasas}(\text{unirPunteros}(e.uniones))) \cup$   
      $\text{claves}(e.comercios \cup_{\text{dicc}} \text{unirComercios}(\text{unirPunteros}(e.uniones)))$

$\text{nivelar} : \text{estr } e \times \text{dicc}(\text{Pos} \times \text{Nat}) d \longrightarrow \text{dicc}(\text{Pos} \times \text{Nat})$   
      $\{(\forall p : \text{Pos})(p \in \text{claves}(d) \Rightarrow_L p \in \text{posiciones}(e))\}$

$\text{nivelar}(e, d) \equiv \text{if } \text{vacio?}(d) \text{ then } \text{vacio} \text{ else } \text{definir}(\text{clave}, \text{nivel}(e, \text{clave}), \text{nivelar}(e, \text{borrar}(\text{clave}, d))) \text{ fi}$

donde

$\text{clave} \equiv \text{dameUno}(\text{claves}(d))$



**2.2.3. Implementación**

Algoritmos

===== **Generadores** =====**iniciar**(in m: Mapa)  $\rightarrow$  res : estr

```

    estr.turno  $\leftarrow$  0
    estr.popularidad  $\leftarrow$  0
    estr.mapa  $\leftarrow$  m
    estr.casas  $\leftarrow$  vacio()
    estr.comercios  $\leftarrow$  vacio()
    estr.uniones  $\leftarrow$  vacia()
    return estr

```

**Complejidad:** O(1)**avanzarTurno**(inout SimCity s, in dicc(Pos, Construccion) cs)

```

    for(nat i  $\leftarrow$  0; i < long(s.uniones); i  $\leftarrow$  i + 1) :
        turnoDesdeUnion  $\leftarrow$  turnoDesdeUnion + 1;

```

```

    itDicc(Pos, Nivel) itCasas  $\leftarrow$  crearIt(s.casas);

```

```

while(haySiguiente(itCasas)) :
    siguienteSignificado(itCasas)  $\leftarrow$  siguienteSignificado(itCasas) + 1
    avanzar(itCasas)

```

```

    itDicc(Pos, Nivel) itComercios  $\leftarrow$  crearIt(s.comercios);

```

```

while(haySiguiente(itComercios)) :
    siguienteSignificado(itComercios)  $\leftarrow$  siguienteSignificado(itComercios) + 1
    avanzar(itComercios)

```

```

    itDicc(Pos, Nivel) itCs  $\leftarrow$  crearIt(cs);

```

```

while(haySiguiente(itCs)) :
    if(siguienteSignificado(itCs) =obs "casa") :
        agCasa(s.casas, siguienteClave(itCs), 1)
    else if(siguienteSignificado(itCs) =obs "comercio") :
        agComercio(s.comercio, siguienteClave(itCs), 1)
    avanzar(itCs)

```

```

    estr.turno  $\leftarrow$  estr.turno + 1

```

**unir**(inout SimCity s1, inout Simcity s2)

```

    s1.popularidad  $\leftarrow$  s1.popularidad + s2.popularidad
    turno  $\leftarrow$  max(s1.turno, s2.turno)
    hijo nuevoHijo  $\leftarrow$  <direccion(s2), 0>
    agregarAtras(s1.uniones, nuevoHijo)

```

=====

===== **Observadores** =====**mapa**(in SimCity s)  $\rightarrow$  res : Mapa

```

    Mapa res  $\leftarrow$  s.mapa
    for(nat i  $\leftarrow$  0; i < long(s.uniones); i  $\leftarrow$  i + 1) :
        res  $\leftarrow$  res + mapa(s.uniones[i].sc)
    return res

```

---

```

casas(in SimCity s) → res : dicc(Pos, Nivel)
  dicc res ← copiar(s.casas)
  for(nat i ← 0; i < long(s.uniones); i ← i + 1) :
    itDicc(Pos, Nivel) itCs ← crearIt(casas(s.uniones[i].sc*));
    while(haySiguiente(itCs)) :
      Pos p ← siguienteClave(itCs)
      Nivel n ← siguienteSignificado(itCs)
      if(¬def?(res, p) ∧ ¬esRio(mapa(s))) :
        definir(res, p, n + s.uniones[i].turnosDesdeUnion)
      avanzar(itCs)
  return res

```

---

```

comercios(in SimCity s) → res : dicc(Pos, Nivel)
  dicc res ← copiar(s.comercios)
  for(nat i ← 0; i < long(s.uniones); i ← i + 1) :
    itDicc(Pos, Nivel) itCs ← crearIt(comercios(s.uniones[i].sc*));
    while(haySiguiente(itCs)) :
      Pos p ← siguienteClave(itCs)
      Nivel n ← siguienteSignificado(itCs)
      if(¬esRio(Mapa(s)) ∧ ¬def?(res, p) ∧ ¬def?(casas(s), p)) :
        Nivel m ← max(n + s.uniones[i].turnosDesdeUnion, nivelCom(p, casas(s)))
        definir(res, p, m)
      avanzar(itCs)
  return res

```

---

```

popularidad(in SimCity s) → res : Nat
  return s.popularidad

```

---

===== Otras Operaciones =====

---

```

nivelCom(in Pos p, in dicc(pos, Nivel) casas) → Nat
  nat maxLvl ← 1
  for(int i = -3; i ≤ 3; ++i) :
    for(int j = |i|-3; j ≤ 3-|i|; ++j) :
      if(p.x + i ≥ 0 ∧ p.y + j ≥ 0) :
        Pos p2 ← <p.x+i, p.y+j>
        if(def?(casas, p2)) :
          maxLvl = max(maxLvl, obtener(casas, p2))
  return maxLvl

```

---

```

agCasa(inout dicc(Pos, Nivel) casas, in Pos p, in Nivel n) :
  definirRapido(casas, p, n)

```

---

```

agComercio(inout dicc(Pos, Nivel) comercios, in Pos p, in Nivel n) :
  definirRapido(comercio, p, n)

```

---

```

turnos(in SimCity s) → res : Nat
  return s.turno

```

---

```

• ∪ •(in dicc(α, β) d1, in dicc(α, β) d2) → res : dicc(α, β)
  dicc(α, β) res = copiar(d1)
  itDicc(α, β) itCs ← crearIt(d2);
  while(haySiguiente(itCs)) :
    α a ← siguienteClave(itCs)

```

---

```
     $\beta$  b  $\leftarrow$  siguienteSignificado(itCs)
    if ( $\neg$ def?(res, a)) :
        definir(res, a, b)
    avanzar(itCs)
return res
```

---

```
construcc(in SimCity s)  $\rightarrow$  res : Nat
    return casas(s)  $\cup$  comercios(s)
```

---

=====

## 2.3. Módulo Servidor

### 2.3.1. Interfaz

#### Interfaz

se explica con: SERVIDOR

géneros: server

#### Operaciones básicas de server

NUEVOSEVER()  $\rightarrow res : server$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} nuevoServer\}$

**Complejidad:**  $O(1)$

**Descripción:** Crea un servidor

**Aliasing:** No tiene

PARTIDAS(**in**  $s : server$ )  $\rightarrow res : dicc(string, SimCity)$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} partidas(s)\}$

**Complejidad:**  $O(copy(server))$

**Descripción:** Devuelve un diccionario con todas las partidas del servidor

**Aliasing:** Devuelve una copia (Esto habria que verlo, ya que no tenemos este dicc(nombre, simcity) por asi decirlo. Maybe hacemos uno de cero? Y tambien habria que ver si lo devolvemos con los SimCity en las hojas o son punteros?)

CONGELADAS(**in**  $s : server$ )  $\rightarrow res : conj(string)$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} congeladas(s)\}$

**Complejidad:**  $O(\#Partidas + |NombreMasLargo|)$

**Descripción:** Devuelve el conjunto con los nombres de las partidas no modificables

**Aliasing:** Devuelve una copia

NUEVAPARTIDA(**in/out**  $s : server$ , **in**  $p : string$ , **in**  $m : mapa$ )

**Pre**  $\equiv \{s =_{obs} s0 \wedge \neg def?(p, partidas(s))\}$

**Post**  $\equiv \{s =_{obs} nuevaPartida(s0, p, m)\}$

**Complejidad:**  $O(|p|)$

**Descripción:** Agrega una partida nueva al servidor

**Aliasing:** No tiene

UNIRPARTIDAS(**in/out**  $s : server$ , **in**  $p1 : string$ , **in**  $p2 : string$ )

**Pre**  $\equiv \{*unionValida(s, p1, p2)\}$

**Post**  $\equiv \{s =_{obs} unirPartidas(s0, p1, p2)\}$

**Complejidad:**  $O()$

**Descripción:** Une dos partidas de simcity en una, p2 pasa a ser no modificable

**Aliasing:** No tiene

AVANZARTURNOPARTIDA(**in/out**  $s : server$ , **in**  $p : string$ )

**Pre**  $\equiv \{def?(p, s) \wedge_L *avanzarTurnoValido(s, p, pendientes(p, s))\}$

**Post**  $\equiv \{s =_{obs} avanzarTurnoPartida(s0, p)\}$

**Complejidad:**  $O()$

**Descripción:** Avanza el turno de una partida y agrega las construcciones definidas en el diccionario de pendientes

**Aliasing:** No tiene

AGREGARCASA(**in/out**  $s : server$ , **in**  $p : string$ , **in**  $pos : Pos$ )

**Pre**  $\equiv \{s =_{obs} s0 \wedge *agregarValido(s, p, pos)\}$

**Post**  $\equiv \{s =_{obs} agregarCasa(s0, p, pos)\}$

**Complejidad:**  $O()$

**Descripción:** Agrega una nueva casa al diccionario de pendientes de la partida

**Aliasing:** No tiene

AGREGARCOMERCIO(**in/out**  $s$ : server, **in**  $p1$ : string, **in**  $p2$ : string)

**Pre**  $\equiv \{s =_{\text{obs}} s0 \wedge *agregarValido(s, p, pos)\}$

**Post**  $\equiv \{s =_{\text{obs}} agregarComercio(s0, p, pos)\}$

**Complejidad:**  $O()$

**Descripción:** Agrega un nuevo comercio al diccionario de pendientes de la partida

**Aliasing:** No tiene

POPULARIDAD(**in**  $s$ : server, **in**  $p$ : string)  $\rightarrow res$ : Nat

**Pre**  $\equiv \{def?(p, partidas(s))\}$

**Post**  $\equiv \{res =_{\text{obs}} verPopularidad(s, p)\}$

**Complejidad:**  $O()$

**Descripción:** Devuelve la popularidad de la partida

**Aliasing:** Devuelve una referencia no modificable

ANTIGUEDAD(**in**  $s$ : server, **in**  $p$ : string)  $\rightarrow res$ : Nat

**Pre**  $\equiv \{def?(p, partidas(s))\}$

**Post**  $\equiv \{res =_{\text{obs}} verTurno(s, p)\}$

**Complejidad:**  $O()$

**Descripción:** Devuelve la antigüedad de la partida

**Aliasing:** Devuelve una referencia no modificable

MAPA(**in**  $s$ : server, **in**  $p$ : string)  $\rightarrow res$ : mapa

**Pre**  $\equiv \{def?(p, partidas(s))\}$

**Post**  $\equiv \{res =_{\text{obs}} verMapa(s, p)\}$

**Complejidad:**  $O()$

**Descripción:** Devuelve el mapa de la partida

**Aliasing:** Devuelve una copia? (habria que ver como funciona mapa en la implementacion del simcity)

VERCASAS(**in**  $s$ : server, **in**  $p$ : string)  $\rightarrow res$ : dicc(Pos, Nat)

**Pre**  $\equiv \{def?(p, partidas(s))\}$

**Post**  $\equiv \{res =_{\text{obs}} verCasas(s, p)\}$

**Complejidad:**  $O()$

**Descripción:** Devuelve un diccionario con las posiciones y niveles de las casas de la partida

**Aliasing:** Devuelve una copia? (habria que ver como funciona casas en la implementacion del simcity)

VERCOMERCIOS(**in**  $s$ : server, **in**  $p$ : string)  $\rightarrow res$ : dicc(Pos, Nat)

**Pre**  $\equiv \{def?(p, partidas(s))\}$

**Post**  $\equiv \{res =_{\text{obs}} verComercios(s, p)\}$

**Complejidad:**  $O()$

**Descripción:** Devuelve un diccionario con las posiciones y niveles de los comercios de la partida

**Aliasing:** Devuelve una copia? (habria que ver como funciona comercios en la implementacion del simcity)

\*donde:

unionValida : server s × Nombre p1 × Nombre p2 → boolean

$$\begin{aligned} \text{unionValida}(s, p1, p2) \equiv & \text{def?}(p1, \text{partidas}(s)) \wedge \text{def?}(p2, \text{partidas}(s)) \wedge \\ & p1 \notin \text{congeladas}(s) \wedge_L \\ & \text{vacio?}(\text{claves}(\text{pendientes}(s, p1))) \wedge \text{vacio?}(\text{claves}(\text{pendientes}(s, p2))) \wedge \\ & (\forall \text{pos} : \text{Pos})(\text{pos} \in \text{claves}(\text{constr1}) \Rightarrow_L \\ & \quad \neg \text{sobreRio}(\text{pos}, \text{sim2}) \wedge \\ & \quad ((\nexists \text{otra} : \text{Pos})(\text{otra} \in \text{constr1} \wedge_L \\ & \quad \quad \text{obtener}(\text{pos}, \text{constr1}).\text{nivel} < \text{obtener}(\text{otra}, \text{constr1}).\text{nivel} \\ & \quad ) \Rightarrow_L \neg \text{def?}(\text{pos}, \text{constr2})) \\ & ) \wedge \\ & (\forall \text{pos} : \text{Pos})(\text{pos} \in \text{claves}(\text{constr2}) \Rightarrow_L \\ & \quad \neg \text{sobreRio}(\text{pos}, \text{sim1}) \wedge \\ & \quad ((\nexists \text{otra} : \text{Pos})(\text{otra} \in \text{constr2} \wedge_L \\ & \quad \quad \text{obtener}(\text{pos}, \text{constr2}).\text{nivel} < \text{obtener}(\text{otra}, \text{constr2}).\text{nivel} \\ & \quad ) \Rightarrow_L \neg \text{def?}(\text{pos}, \text{constr1})) \\ & ) \end{aligned}$$

donde  $\text{sim1} \equiv \text{obtener}(p1, \text{partidas}(s))$ ,  
 $\text{sim2} \equiv \text{obtener}(p2, \text{partidas}(s))$ ,  
 $\text{constr1} \equiv \text{casas}(\text{sim1}) \cup_{\text{dicc}} \text{comercios}(\text{sim1})$ ,  
 $\text{constr2} \equiv \text{casas}(\text{sim2}) \cup_{\text{dicc}} \text{comercios}(\text{sim2})$

avanzarTurnoValido : server s × Nombre p × dicc(Pos × Construccion) cs → boolean

$$\begin{aligned} \text{avanzarTurnoValido}(s, p, \text{cs}) \equiv & \text{def?}(p, \text{partidas}(s)) \wedge \\ & p \notin \text{congeladas}(s) \wedge \\ & \neg \text{vacía?}(\text{claves}(\text{cs})) \end{aligned}$$

agregarValido : server s × Nombre p × Pos pos → boolean

$$\begin{aligned} \text{agregarValido}(s, p, \text{pos}) \equiv & \text{def?}(p, \text{partidas}(s)) \wedge_L \neg p \in \text{congeladas}(s) \wedge \\ & \neg \text{def?}(\text{pos}, \text{verCasas}(s, p)) \wedge \neg \text{def?}(\text{pos}, \text{verComercios}(s, p)) \wedge \\ & \neg \text{def?}(\text{pos}, \text{pendientes}(s, p)) \wedge \neg \text{esRio}(\text{pos}, \text{verMapa}(s, p)) \end{aligned}$$

•  $\cup_{\text{dicc}}$  • :  $\text{dicc}(\alpha \times \beta) \times \text{dicc}(\alpha \times \beta) \rightarrow \text{dicc}(\alpha, \beta)$

$d \cup_{\text{dicc}} d' \equiv$  **if**  $\text{vacío?}(\text{claves}(d'))$  **then**  
 $d$   
**else**  
 $\text{definir}(\text{dameUno}(\text{claves}(d')),$   
 $\text{obtener}(\text{dameUno}(\text{claves}(d')), d'),$   
 $d \cup_{\text{dicc}} \text{borrar}(\text{dameUno}(\text{claves}(d')), d'))$   
**fi**

### 2.3.2. Representación

## Representación

### Representación de servidor

Un servidor almacena y actualiza los diferentes SimCity. Se representa como un diccionario implementado en un trie, donde las claves son los nombres de las partidas y los significados un puntero al SimCity, un diccionario de construcciones pendientes a agregar, y su estado (si es modificable o no).

servidor se representa con `diccTrie(Nombre, partida)`

donde `partida` es tupla(`modificable : bool` ,  
`sim : puntero(SimCity)` ,  
`pendientes : dicc(Pos, Construcion)` )

donde `pos` es tupla(`x : Nat` , `y : Nat` )

donde `Nombre` es string

donde `Construcion` es string

`Rep : estr  $\rightarrow$  boolean`

`Rep(e)  $\equiv$  true  $\iff$`   
`( $\forall$  partida1, partida2 : Nombre)`  
`((def?(partida1, e)  $\wedge$  def?(partida2, e)  $\wedge$  partida1  $\neq$  partida2)  $\Rightarrow_L$`   
`obtener(partida1, e).sim  $\neq$  obtener(partida2, e).sim`  
`)  $\wedge$`   
`( $\forall$  partida : Nombre)(def?(partida, e)  $\Rightarrow_L$`   
`p.sim  $\neq$  NULL  $\wedge$`   
`(p.modificable =obs false  $\Rightarrow_L$  vacio?(claves(p.pendientes)))  $\wedge$`   
`( $\forall$  pos : Pos)(def?(pos, p.pendientes)  $\Rightarrow_L$`   
`(obtener(pos, p.pendientes)  $\in$  {"casa", "comercio"}  $\wedge$`   
`$\neg$  def?(pos, construcciones(*p.sim)))`  
`)`  
`)`

donde `p` es obtener(`partida`, `e`)

`Abs : estr e  $\rightarrow$  servidor`

`{Rep(e)}`

`Abs(e)  $\equiv$  s: servidor |`  
`( $\forall$  nombre : Nombre)`  
`(nombre  $\in$  congelados(s)  $\Leftrightarrow$`   
`(def?(nombre, e)  $\wedge_L$  obtener(nombre, e).modificable =obs false))`  
 `$\wedge$`   
`( $\forall$  nombre : Nombre)`  
`(def?(nombre, partidas(s))  $\Leftrightarrow$  def?(nombre, e))`  
 `$\wedge_L$`   
`( $\forall$  nombre : Nombre)`  
`(def?(nombre, partidas(s))  $\Rightarrow_L$`   
`(obtener(nombre, partidas(s)) =obs *(obtener(nombre, e).sim)  $\wedge$   
  pendientes(s, nombre) =obs obtener(nombre, e).pendientes))`

**2.3.3. Implementación****Algoritmos**


---

**nuevoServer()**  $\rightarrow res : \text{estr}$ 

 1:  $res \leftarrow \text{vacío}()$   
 2: **return**  $res$ 
Complejidad:  $O(1)$ 


---



---

**partidas(in**  $e : \text{estr}$ **)**  $\rightarrow res : \text{diccTrie}(\text{Nombre}, \text{SimCity})$ 

 1:  $\text{dicc}(\text{Nombre}, \text{SimCity}) \text{ } res \leftarrow \text{vacío}()$   
 2:  $\text{itDicc}(\text{Nombre}, \text{Partida}) \text{ } it \leftarrow \text{crearIt}(e)$   
 3: **while**( $\text{haySiguiete}(it)$ )  
 4:      $\text{definirRapido}(res, \text{siguienteClave}(it), \text{siguienteSignificado}(it).sim)$   
 5:      $\text{avanzar}(it)$   
 6: **return**  $res$ 
Complejidad:  $O(\text{copy}(estr))$ 


---



---

**congeladas(in**  $e : \text{estr}$ **)**  $\rightarrow res : \text{conj}(\text{Nombre})$ 

 1:  $\text{conj}(\text{Nombre}) \text{ } res \leftarrow \text{vacío}()$   
 2:  $\text{itDicc}(\text{Nombre}, \text{Partida}) \text{ } it \leftarrow \text{crearIt}(e)$   
 3: **while**( $\text{haySiguiete}(it)$ )  
 4:     **if**( $\text{siguienteSignificado}(it).modificable == \text{false}$ )  
 5:          $\text{agregarRapido}(res, \text{siguienteClave}(it))$   
 6:      $\text{avanzar}(it)$   
 7: **return**  $res$ 
Complejidad:  $O(\#Partidas + |\text{NombreMasLargo}|)$ 


---



---

**nuevaPartida(in/out**  $e : \text{estr}$ **, in**  $p : \text{Nombre}$ **, in**  $m : \text{Mapa}$ **)**

 1:  $\text{definirRapido}(e, p, \langle \text{true}, \&(\text{iniciar}(m)), \text{vacío}() \rangle)$       $\triangleright$  Reservamos memoria para el nuevo SimCity

Complejidad:  $O(|p|)$ 


---



---

**unirPartidas(in/out**  $e : \text{estr}$ **, in**  $p1 : \text{Nombre}$ **, in**  $p2 : \text{Nombre}$ **)**

 1:  $\text{definir}(e, p1, \langle \text{true}, \&(\text{unir}(*\text{significado}(p1, e).sim, *\text{significado}(p2, e).sim)), \text{vacío}() \rangle)$   
 2:  $\text{definir}(e, p2, \langle \text{false}, \text{significado}(p2, e).sim, \text{vacío}() \rangle)$ 
Complejidad:  $O(|\text{Nombre}|)$ 
Justificación:  $\text{unir} \in O(1), \text{definir} \in O(|\text{Nombre}|) + O(1) = O(|\text{Nombre}|)$ 


---



---

**agregarCasa(in/out**  $s : \text{estr}$ **, in**  $partida : \text{String}$ **, in**  $pos : \text{Pos}$ **)**

 1:  $\text{definirRapido}(\text{Pos}, \text{"casa"}, \text{obtener}(partida, s))$ 
Complejidad:  $O(|partida|)$ 


---



---

---

**agregarComercio**(in/out  $s$ : estr, in  $partida$ : String, in  $pos$ : Pos)1:  $definirRapido(Pos, "comercio", obtener(partida, s))$ Complejidad:  $O(|partida|)$ 

---

---

---

**verMapa**(in  $s$ : estr, in  $partida$ : String)  $\rightarrow res$ : Mapa1:  $sc \leftarrow obtener(partida, s).sim$ 2:  $res \leftarrow mapa(*sc)$ 3: **return**  $res$ Complejidad:  $O(|partida|) + O(mapa(*sc))$ 

---

---

---

**verCasas**(in  $s$ : estr, in  $partida$ : String)  $\rightarrow res$ : DiccLineal(Pos, Nivel)1:  $sc \leftarrow obtener(partida, s).sim$ 2:  $res \leftarrow casas(*sc)$ 3: **return**  $res$ Complejidad:  $O(|partida|) + O(casas(*sc))$ 

---

---

---

**verComercios**(in  $s$ : estr, in  $partida$ : String)  $\rightarrow res$ : DiccLineal(Pos, Nivel)1:  $sc \leftarrow obtener(partida, s).sim$ 2:  $res \leftarrow comercios(*sc)$ 3: **return**  $res$ Complejidad:  $O(|partida|) + O(comercios(*sc))$ 

---

---

---

**verPopularidad**(in  $s$ : estr, in  $partida$ : String)  $\rightarrow res$ : Nat1:  $sc \leftarrow obtener(partida, s).sim$ 2:  $res \leftarrow popularidad(*sc)$ 3: **return**  $res$ Complejidad:  $O(|partida|)$ 

---

---

---

**verTurno**(in  $s$ : estr, in  $partida$ : String)  $\rightarrow res$ : Nat1:  $sc \leftarrow obtener(partida, s).sim$ 2:  $res \leftarrow turnos(*sc)$ 3: **return**  $res$ Complejidad:  $O(|partida|)$ 

---