



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. Aliases . . . . .	8
2.2. Módulo Mapa . . . . .	8
2.2.1. Interfaz . . . . .	8
2.2.2. Representación . . . . .	9
2.2.3. Implementación . . . . .	10
2.3. Módulo SimCity . . . . .	11
2.3.1. Interfaz . . . . .	11
2.3.2. Representación . . . . .	13
2.3.3. Implementación . . . . .	18
2.4. Módulo Servidor . . . . .	21
2.4.1. Interfaz . . . . .	21
2.4.2. Representación . . . . .	23
2.4.3. Implementación . . . . .	24

# 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 : Pos × Mapa  $\longrightarrow$  Bool

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

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

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

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

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

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

**Fin TAD**



```

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)

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

```

- definido en el apartado Definiciones Auxiliares de SimCity.
- las funciones **agCasas** y **agComercios** agregan respectivamente al diccionario de casas/comercios las construcciones de entrada sin importar si esas posiciones ya se encontraban ocupadas o no. Esto no genera un problema en avanzar turno, por sus restricciones, pero si al unir SimCitys (ya que podrian haber colisiones). Para solucionar esto, sacarRepes quita del diccionario de entrada las posiciones ocupadas por construcciones ya establecidas. Es decir, como resolución al conflicto de colisiones, las construcciones que permanecen son las del SimCity original.

```

avanzarNivel(cs)      ≡ if vacio?(claves(cs)) then
                        cs
                        else
                            definir(dameUno(claves(cs)),
                                obtener(dameUno(claves(cs)), cs) + 1,
                                avanzarNivel(borrar(dameUno(claves(cs))), cs))
                        fi

```

**Fin TAD**

## Definiciones Auxiliares de SimCity

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

avanzarValido( $s, cs$ )  $\equiv \neg \text{vacio?}(\text{claves}(cs)) \wedge$   
 $(\forall p : \text{Pos}) (\text{def?}(p, cs) \Rightarrow_L$   
 $(\neg p \in \text{claves}(\text{construcciones}(s)) \wedge$   
 $\neg \text{esRio}(p, \text{mapa}(s)) \wedge$   
 $(\text{obtener}(p, cs) = \text{"casa"} \vee \text{obtener}(p, cs) = \text{"comercio"}))$   
 $)$

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

unirValido( $a, b$ )  $\equiv (\forall p : \text{Pos})(\text{def?}(p, \text{construcciones}(a)) \Rightarrow_L$   
 $\neg \text{esRio}(p, \text{mapa}(b)) \wedge$   
 $(\nexists otra : \text{Pos})^1(\text{def?}(otra, \text{construcciones}(a)) \wedge_L$   
 $\text{obtener}(otra, \text{construcciones}(a)) > \text{obtener}(p, \text{construcciones}(a))$   
 $) \Rightarrow_L \neg \text{def?}(p, \text{construcciones}(b))$   
 $) \wedge$   
 $(\forall p : \text{Pos})(\text{def?}(p, \text{construcciones}(b)) \Rightarrow_L$   
 $\neg \text{esRio}(p, \text{mapa}(a)) \wedge$   
 $(\nexists otra : \text{Pos})^1(\text{def?}(otra, \text{construcciones}(b)) \wedge_L$   
 $\text{obtener}(otra, \text{construcciones}(b)) > \text{obtener}(p, \text{construcciones}(b))$   
 $) \Rightarrow_L \neg \text{def?}(p, \text{construcciones}(a))$   
 $)$

1. Si en la posición hay una construcción de nivel máximo, no puede colisionar con una construcción del otro SimCity.

## 1.4. Servidor

### TAD SERVIDOR

<b>géneros</b>	Server
<b>exporta</b>	Server, observadores, generadores, verMapa, verCasas, verComercios, verPopularidad, agregarCasa, agregarComercio y verTurno
<b>usa</b>	SimCity, Mapa, Nombre, Pos, Construcción, Nivel, Nat, Bool, $\text{dicc}(\alpha \times \beta)$ , $\text{conj}(\alpha)$

### igualdad observacional

$$(\forall s, s' : \text{Server}) \left( s =_{\text{obs}} s' \iff \left( \text{partidas}(s) =_{\text{obs}} \text{partidas}(s') \wedge_{\text{L}} \text{congeladas}(s) =_{\text{obs}} \text{congeladas}(s') \right) \right)$$

### observadores básicos

partidas	: Server	$\longrightarrow \text{dicc}(\text{Nombre} \times \text{SimCity})$
congeladas	: Server	$\longrightarrow \text{conj}(\text{Nombre})$

### generadores

nuevoServer	:	$\longrightarrow \text{Server}$	
nuevaPartida	: Server $s \times \text{Nombre } p \times \text{Mapa}$	$\longrightarrow \text{Server}$	$\{\neg \text{def?}(p, \text{partidas}(s))\}$
unirPartidas	: Server $s \times \text{Nombre } p1 \times \text{Nombre } p2$	$\longrightarrow \text{Server}$	$\{\text{unionValida}(s, p1, p2)^1\}$
avanzarTurnoPartida	: Server $s \times \text{Nombre } p \times \text{dicc}(\text{Pos} \times \text{Construcción}) \text{ } cs$	$\longrightarrow \text{Server}$	$\{\text{avanzarValido}(s, p)^1\}$

### otras operaciones

verMapa	: Server $s \times \text{Nombre } p$	$\longrightarrow \text{Mapa}$	$\{\text{def?}(p, \text{partidas}(s))\}$
verCasas	: Server $s \times \text{Nombre } p$	$\longrightarrow \text{dicc}(\text{Pos} \times \text{Nivel})$	$\{\text{def?}(p, \text{partidas}(s))\}$
verComercios	: Server $s \times \text{Nombre } p$	$\longrightarrow \text{dicc}(\text{Pos} \times \text{Nivel})$	$\{\text{def?}(p, \text{partidas}(s))\}$
verPopularidad	: Server $s \times \text{Nombre } p$	$\longrightarrow \text{Nat}$	$\{\text{def?}(p, \text{partidas}(s))\}$
verTurno	: Server $s \times \text{Nombre } p$	$\longrightarrow \text{Nat}$	$\{\text{def?}(p, \text{partidas}(s))\}$
agregarCasa	: Server $s \times \text{Nombre } p \times \text{Pos } pos$	$\longrightarrow \text{Server}$	$\{\text{agregarValido}(s, p, pos)^1\}$
agregarComercio	: Server $s \times \text{Nombre } p \times \text{Pos } pos$	$\longrightarrow \text{Server}$	$\{\text{agregarValido}(s, p, pos)^1\}$

### axiomas $\forall s: \text{Server}, \forall p, p', p'': \text{Nombre}, \forall m: \text{Mapa}, \forall pos: \text{Pos}$

partidas(nuevoServer)	$\equiv \text{vacío}$
partidas(nuevaPartida( $s, p, m$ ))	$\equiv \text{definir}(p, \text{iniciar}(m), \text{partidas}(s))$
partidas(unirPartidas( $s, p, p'$ ))	$\equiv \text{definir}(p,$ $\text{unir}(\text{obtener}(p, \text{partidas}(s)),$ $\text{obtener}(p', \text{partidas}(s))),$ $\text{partidas}(s))$
partidas(avanzarTurnoPartida( $s, p, cs$ ))	$\equiv \text{definir}(p,$ $\text{avanzarTurno}(\text{obtener}(p, \text{partidas}(s)), cs),$ $\text{partidas}(s))$
congeladas(nuevoServer)	$\equiv \emptyset$
congeladas(nuevaPartida( $s, p, m$ ))	$\equiv \text{congeladas}(s)$
congeladas(unirPartidas( $s, p, p'$ ))	$\equiv \text{Ag}(p', \text{congeladas}(s))$
congeladas(avanzarTurnoPartida( $s, p$ ))	$\equiv \text{congeladas}(s)$

1. definido en el apartado Definiciones Auxiliares de Servidor.

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

$\text{verMapa}(s, p) \equiv \text{mapa}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{verCasas}(s, p) \equiv \text{casas}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{verComercios}(s, p) \equiv \text{comercios}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{verPopularidad}(s, p) \equiv \text{popularidad}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{verTurno}(s, p) \equiv \text{turnos}(\text{obtener}(p, \text{partidas}(s)))$   
 $\text{agregarCasa}(s, p, pos) \equiv \text{avanzarTurnoPartida}(s, p, \text{definir}(pos, \text{"casa"}, \text{vacio}))$   
 $\text{agregarComercio}(s, p, pos) \equiv \text{avanzarTurnoPartida}(s, p, \text{definir}(pos, \text{"comercio"}, \text{vacio}))$

**Fin TAD**

## Definiciones Auxiliares de Servidor

$\text{unionValida} : \text{Server } s \times \text{Nombre } p \times \text{Nombre } p' \longrightarrow \text{boolean}$   
 $\text{unionValida}(s, p, p') \equiv \text{def?}(p, \text{partidas}(s)) \wedge \text{def?}(p', \text{partidas}(s)) \wedge p \notin \text{congeladas}(s) \wedge_{\text{L}}$   
 $\quad \text{unirValido}(\text{obtener}(p, \text{partidas}(s)), \text{obtener}(p', \text{partidas}(s)))^1$

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

$\text{avanzarValido}(s, p, cs) \equiv \text{def?}(p, \text{partidas}(s)) \wedge$   
 $\quad p \notin \text{congeladas}(s) \wedge$   
 $\quad \neg \text{vacía?}(\text{claves}(cs)) \wedge_{\text{L}}$   
 $\quad (\forall pos: \text{Pos})(pos \in \text{claves}(cs) \Rightarrow_{\text{L}}$   
 $\quad \quad \text{obtener}(pos, cs) \in \{\text{"casa"}, \text{"comercio"}\} \wedge$   
 $\quad \quad \neg \text{esRio}(pos, \text{verMapa}(s, p)) \wedge$   
 $\quad \quad \neg \text{def?}(pos, \text{verCasas}(s, p)) \wedge$   
 $\quad \quad \neg \text{def?}(pos, \text{verComercios}(s, p))$   
 $\quad )$

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

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

1. definido en el apartado Definiciones Auxiliares de SimCity.



## 2. Diseño

### 2.1. Aliases

Pos es  $\text{TUPLA} \langle X: \text{NAT} \times Y: \text{NAT} \rangle$

CONSTRUCCIÓN es  $\text{STRING}$

NOMBRE es  $\text{STRING}$

NIVEL es  $\text{NAT}$

### 2.2. Módulo Mapa

#### 2.2.1. Interfaz

#### Interfaz

**usa:** Conjunto Lineal, Nat, Bool, Pos

**exporta:** todo

**se explica con:** MAPA

**géneros:** Mapa

#### Operaciones básicas de Mapa

**HORIZONTALES**(**in**  $m: \text{Mapa}$ )  $\rightarrow res: \text{Conj}(\text{Nat})$

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

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

**Complejidad:**  $O(\#\text{horizontales}(\hat{m}))$

**Descripción:** Devuelve el conjunto de ríos horizontales.

**Aliasing:** Por copia.

**VERTICALES**(**in**  $m: \text{Mapa}$ )  $\rightarrow res: \text{Conj}(\text{Nat})$

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

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

**Complejidad:**  $O(\#\text{verticales}(\hat{m}))$

**Descripción:** Devuelve el conjunto de ríos verticales.

**Aliasing:** Por copia.

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

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

**Post**  $\equiv \{r\hat{e}s =_{\text{obs}} \text{crear}(\hat{hs}, \hat{vs})\}$

**Complejidad:**  $O(\text{copy}(hs) + \text{copy}(vs))$

**Descripción:** Crea un mapa.

**Aliasing:** Por copia.

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

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

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

**Complejidad:**  $O(\#\text{horizontales}(m) + \#\text{verticales}(m))$

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

**Descripción:** Une dos Mapas.

**Aliasing:** Por copia.

**2.2.2. 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  $\text{tupla}(\text{horizontales: conj}(\text{Nat}),$   
 $\text{verticales: conj}(\text{Nat}))$

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

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

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

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

$\text{Abs}(e) \equiv m: \text{Mapa} \mid \text{horizontales}(m) =_{\text{obs}} e.\text{horizontales} \wedge \text{verticales}(m) =_{\text{obs}} e.\text{verticales}$

### 2.2.3. Implementación

## Algoritmos

---

**crear**(in  $hs : \text{conj}(\text{Nat})$ , in  $vs : \text{conj}(\text{Nat})$ )  $\longrightarrow$  res : estr

1: res.horizontales  $\leftarrow$  hs

2: res.verticales  $\leftarrow$  vs

3: **return** res

**Complejidad:**  $O(\text{copy}(hs) + \text{copy}(vs))$

---

**Suma**(in  $m1$ : mapa, in  $m2$ : mapa)  $\longrightarrow$  res : mapa

1: mapa res  $\leftarrow$  copiar( $m1$ )

2: itConj(Nat) itH  $\leftarrow$  crearIt(horizontales( $m2$ ))

3: **while**(haySiguiete(itH)) :

4:     Ag(horizontales(res), siguiente(itH))

5:     avanzar(itH)

6: itConj(Nat) itV  $\leftarrow$  crearIt(vertales( $m2$ ))

7: **while**(haySiguiete(itV)) :

8:     Ag(vertales(res), siguiente(itV))

9:     avanzar(itV)

10: **return** res

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

---

**esRio**(in  $p$ : Pos, in  $m$ : Mapa)  $\longrightarrow$  res : Bool

1: **return**  $p.x \in \text{verticales}(m) \vee p.y \in \text{horizontales}(m)$

**Complejidad:**  $O(\#\text{horizontales}(m) + \#\text{verticales}(m))$

---

**verticales**(in  $m$ : mapa)  $\longrightarrow$  res : Conj(Nat)

1: **return** m.verticales

**Complejidad:**  $O(\text{copy}(m.\text{verticales}))$

---

**horizontales**(in  $m$ : mapa)  $\longrightarrow$  res : Conj(Nat)

1: **return** m.horizontales

**Complejidad:**  $O(\text{copy}(m.\text{horizontales}))$

---

## 2.3. Módulo SimCity

### 2.3.1. Interfaz

#### Interfaz

**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  $S$ , y  $\text{construcciones} = \sum_{i=0}^N (\#(\text{construcciones}_i))$ , donde  $\text{construcciones}_i$  es el conjunto de casas y comercios definidos en  $u_i$  por medio de *AvanzarTurno*.<sup>2</sup>

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

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

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

**Complejidad:**  $O(\sum_{i=0}^N (\text{mapa}_i))$ , donde  $\text{mapa}_i$  es el Mapa original<sup>3</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:**  $O(\text{construcciones}^2) + O(N)$

**justificación:** Una implementación de SimCity debe poder, en el peor caso, retornar una copia de todas sus casas en tiempo lineal sobre el conjunto que las representa. Pero, dado que es esperable que un SimCity, compuesto por un conjunto de uniones, sea representado como un árbol, podemos suponer que para resolver colisiones del tipo 'se queda el primero', cada nueva casa a copiar debe primero evaluar si no pertenece ya a los niveles superiores, incluyendo también a los comercios. Resultando en un peor caso, holgado, de  $O(\text{construcciones}^2)$ , donde no hay colisiones. Dado que un SimCity puede no tener construcciones si está recién iniciado, se debe considerar que el peor caso también puede estar dado por la cantidad de nodos a recorrer en el árbol. Es decir  $O(N)$ . Por ejemplo, si se anidan 100 SimCities recién iniciados.

**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:**  $O(\text{construcciones}^2) + O(N)$

**justificación:** similar a *casas(s)*.

**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:**  $O(1)$

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

**Aliasing:** No. Por copia.

1. Este conjunto incluye también a los SimCities provenientes de las uniones propias a cada SimCity en unión directa con  $S$ .
2. En particular, notar que  $\text{construcciones}_i \geq \#(\text{claves}(\text{construcciones}(\hat{S})))$ , por posibles colisiones permitidas entre los  $u_i$ . Dado la necesidad de resolver la union en  $O(1)$ , no se puede mantener un registro de construcciones sin repetidos. De éste modo, se contempla el total real de construcciones definidas al momento de calcular la complejidad que tendrán las operaciones.
3. Es decir, aquel con el que se inició originalmente el simCity.

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

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

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

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

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

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

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

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

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

**Complejidad:**  $O(\text{copy}(m))$

**Descripción:** Crea un nuevo SimCity.

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

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

**Pre**  $\equiv \{\text{avanzarValido}^1(\hat{s}, \hat{c}s) \wedge \hat{S} = S_0\}$

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

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

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

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

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

**Descripción:** Une dos SimCities.

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

1. definido en Definiciones Auxiliares de SimCity.

### 2.3.2. 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 *antigüedad* o *turno* de la partida se representará a través de un Natural. A su vez, se guardará un *turno* interno, representativo de la antigüedad 'original' del *simCity*, si este no perteneciera a ninguna unión.

La ubicación de las casas y comercios se representará sobre una secuencia de punteros a diccionarios lineales con clave *Pos* y significado *Construccion*. Bajo esta representación, el *nivel* de cada casa es el *turno* interno actual del *SimCity* menos la posición del diccionario donde se encuentra definida en la secuencia. Esto se debe a que en cada turno se agrega un diccionario nuevo. Similarmente, el *nivel* de los comercios se representará igual, pero en este caso también se deberá calcular a partir del *nivel* de las casas a distancia manhattan  $\leq 3$ .

La *popularidad* se representará a través de un Natural. El mapa será de tipo *Mapa* y las uniones se representarán 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(*antigüedad* : Nat , *turno* : Nat, *popularidad* : Nat, *mapa* : Mapa,   
 *construcciones* : lista(puntero(diccLineal(Pos, Construccion))) ,   
 *uniones* : lista(hijo) )

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

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

$\text{Rep} : \text{estr}^1 \rightarrow \text{boolean}$

$\text{Rep}(e) \equiv \text{true} \iff ($   
 $(\forall h : \text{hijo})(\text{esta?}(h, e.\text{uniones}) \Rightarrow_L$   
 $(h.\text{sc} \neq \text{NULL} \wedge_L \text{rep}(*h.\text{sc}))^2 \wedge_L$   
 $(e.\text{turno} \geq h.\text{turnoUnido})^3 \wedge$   
 $(\forall h' : \text{hijo})(\text{esta?}(h', e.\text{uniones}) \wedge_L \text{pos}(h', e.\text{uniones}) > \text{pos}(h, e.\text{uniones}) \Rightarrow_L$   
 $h'.\text{turnoUnido} \leq h.\text{turnoUnido}$   
 $)^4$   
 $) \wedge_L$   
 $(\forall p : \text{puntero}(\text{dicc}(\text{Pos}, \text{Construccion})))(\text{esta?}(p, e.\text{construcciones}) \Rightarrow_L$   
 $\neg \text{vacio?}(*p) \wedge$   
 $(\nexists p' : \text{puntero}(\text{dicc}(\text{Pos}, \text{Construccion})))(\text{esta?}(p', e.\text{construcciones}) \wedge_L$   
 $\text{pos}(p, e.\text{construcciones}) \neq \text{pos}(p', e.\text{construcciones}) \wedge_L$   
 $\text{claves}(*p) \cap \text{claves}(*p') \neq \emptyset$   
 $)$   
 $)^5 \wedge_L$   
 $(\text{long}(e.\text{construcciones}) = e.\text{turno})^6 \wedge_L$   
 $(\&e \notin \text{Unidos})^7 \wedge_L$   
 $(\forall p : \text{puntero}(\text{estr}))(p \in \text{Unidos} \Rightarrow_L$   
 $e.\text{antigüedad} \geq (*p).\text{antigüedad}$   
 $)^8 \wedge$   
 $(e.\text{popularidad} = \#(\text{Unidos}))^9 \wedge$   
 $(\forall p : \text{Pos})(p \in \text{Casas} \Rightarrow_L$   
 $\neg \text{esRio}(p, \text{Mapas})$   
 $)^{10} \wedge$   
 $(\forall p : \text{Pos})(p \in \text{Comercios} \Rightarrow_L$   
 $\neg \text{esRio}(p, \text{Mapas})$   
 $)^{11} \wedge$   
 $\text{unionesValidas}(e, e.\text{uniones})^{13}$   
 $)$

donde

Unidos  $\equiv$  unirPunteros( $e.uniones$ )  
 Casas  $\equiv$  unirCasas(Ag(& $e$ , Unidos))  
 Comercios  $\equiv$  unirComercios(Ag(& $e$ , Unidos))  
 Mapas  $\equiv$  unirMapas(Ag(& $e$ , Unidos))

Abs : estr  $e \longrightarrow$  SimCity

{Rep( $e$ )}

Abs( $e$ )  $\equiv$   $sc : \text{SimCity} \mid$   
     mapa( $sc$ )  $=_{\text{obs}}$  Mapas  $\wedge$   
     casas( $sc$ )  $=_{\text{obs}}$  nivelar(Casas)  $\wedge$   
     comercios( $sc$ )  $=_{\text{obs}}$  nivelar(Comercios)  $\wedge$   
     popularidad( $sc$ )  $=_{\text{obs}}$   $e.popularidad$

donde

Unidos  $\equiv$  unirPunteros( $e.uniones$ )  
 Casas  $\equiv$  unirCasas(Ag(& $e$ , Unidos))  
 Comercios  $\equiv$  unirComercios(Ag(& $e$ , Unidos))  
 Mapas  $\equiv$  unirMapas(Ag(& $e$ , Unidos))

## Definiciones Auxiliares

Los siguientes reemplazos sintácticos están confinados 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.

unirPunteros : secu(hijo)  $s \longrightarrow$  conj(puntero(estr))  $\{(\forall h : \text{hijo})(h \in s \Rightarrow_L h.sc \neq \text{NULL})\}$

unirPunteros( $s$ )  $\equiv$  \_unirPunteros( $s$ ,  $\emptyset$ )

\_unirPunteros : secu(hijo)  $s \times$  conj(puntero(estr))  $ps \longrightarrow$  conj(puntero(estr)) {eq. unirPunteros}

\_unirPunteros( $s$ ,  $ps$ )  $\equiv$  **if** vacia?( $s$ ) **then**  
      $ps$   
     **else if** prim( $s$ ). $sc \in ps$  **then** // por si hay loops  
         \_unirPunteros(fin( $s$ ),  $ps$ )  
     **else**  
         \_unirPunteros((\*(prim( $s$ ). $sc$ )). $uniones$ , Ag(prim( $s$ ). $sc$ ,  $ps$ ))  $\cup$   
         \_unirPunteros(fin( $s$ ), Ag(prim( $s$ ). $sc$ ,  $ps$ ))  
     // al unir se descarta el duplicado  
**fi**

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 turno es mayor o igual al turno de cualquier unión inmediata.
4. Las uniones están ordenadas de más antiguas a más recientes.
5. Las construcciones en éste nivel no se solapan.
6. Se agregó al menos un conjunto de construcciones por cada turno interno.
7. La estructura no loopea consigo misma.
8. La antigüedad es mayor o igual a la antigüedad de cualquier simCity hijo.
9. La popularidad es igual a la cantidad de uniones total.
10. Ninguna casa está sobre un río perteneciente a cualquier mapa en la unión.
11. Ninguna construcción está sobre un río perteneciente a cualquier mapa en la unión .
12. No se solapan posiciones máximas entre esta estructura hasta el hijo 'x', y ese hijo, para todo hijo. Notar que no importan los turnos posteriores al momento de union, porque cualquier construcción agregada posteriormente no podrá ser máxima o modificar los máximos al momento de la unión. Dadas las condiciones ya establecidas hasta este momento.

```

unirMapas : conj(puntero(estr))  $ps \rightarrow$  Mapa
                                                     $\{(\forall p : \text{puntero(estr)})(p \in ps \Rightarrow_L (p \neq \text{NULL} \wedge_L \text{rep}(*p)))\}$ 
unirMapas( $ps$ )  $\equiv$  if vacio?( $ps$ ) then
                  crear( $\emptyset$ ,  $\emptyset$ )
                  else
                     $(*(\text{dameUno}(ps))).\text{mapa} + \text{UnirMapas}(\text{sinUno}(ps))$ 
                  fi

unirCasas : conj(puntero(estr))  $ps \rightarrow$  dicc(Pos  $\times$  Nat)
                                                    {eq. unirMapas}
unirCasas( $ps$ )  $\equiv$  if vacio?( $ps$ ) then
                  vacio
                  else
                     $it((*(\text{dameUno}(ps))).\text{construcciones}, "casa") \cup_{\text{dicc}} \text{unirCasas}(\text{sinUno}(ps))$ 
                  fi

unirComercios : conj(puntero(estr))  $ps \rightarrow$  dicc(Pos  $\times$  Nat)
                                                    {eq. unirMapas}
unirComercios( $ps$ )  $\equiv$  if vacio?( $ps$ ) then
                  vacio
                  else
                     $it((*(\text{dameUno}(ps))).\text{construcciones}, "comercio") \cup_{\text{dicc}} \text{unirComercios}(\text{sinUno}(ps))$ 
                  fi

it : secu(puntero(dicc(Pos  $\times$  Construcccion)))  $s \times$  Construcccion  $c \rightarrow$  dicc(Pos  $\times$  Nat)
// el significado representa el turno en que se agregó la construcción.
it( $s, c$ )  $\equiv$  _it( $s, c, 0$ )

_it : secu(puntero(dicc(Pos  $\times$  Construcccion)))  $s \times$  Construcccion  $c \times$  Nat  $n \rightarrow$  dicc(Pos  $\times$  Nat)
_it( $s, c, n$ )  $\equiv$  if vacia?( $s$ ) then vacio else filtrar( $*(\text{prim}(s)), c, n$ )  $\cup_{\text{dicc}}$  it( $\text{fin}(s), c, n + 1$ ) fi

filtrar : dicc(Pos  $\times$  Construcccion)  $\times$  Construcccion  $\times$  Nat  $\rightarrow$  dicc(Pos  $\times$  Nat)
filtrar( $d, c, n$ )  $\equiv$  if vacio?( $d$ ) then
                  vacio
                  else if obtener(clave,  $d$ ) =  $c$  then
                    definir(clave,  $n$ , filtrar(borrar(clave,  $d$ ),  $c, n$ ))
                  else
                    filtrar(borrar(clave,  $d$ ),  $c, n$ )
                  fi

donde
clave  $\equiv$  dameUno(claves( $d$ ))

pos : secu( $\alpha$ )  $s \times \alpha a \rightarrow$  Nat
                                                    {esta?( $a, s$ )}
pos( $s, a$ )  $\equiv$  if prim( $s$ ) =  $a$  then 0 else 1 + pos(fin( $s$ ),  $a$ ) fi

unionesValidas : estr  $e \times$  secu(hijo)  $s \rightarrow$  bool
                                                     $\{s \subseteq e.\text{uniones} \wedge_L (\forall h : \text{hijo})(h \in s \Rightarrow_L h.\text{sc} \neq \text{NULL} \wedge_L \text{rep}(*h.\text{sc}))\}$ 
unionesValidas( $e, s$ )  $\equiv$  vacio?( $s$ )  $\vee_L$  (maxcons( $e$ , izq)  $\cap$  maxcons( $e$ , der) =  $\emptyset \wedge$  unionesValidas( $e$ , com( $s$ )))
donde
com  $\equiv$  unirPunteros(com( $s$ ))
ult  $\equiv$  ult( $s$ )  $\bullet$  <>
casascom  $\equiv$  unirCasas(com)
comercom  $\equiv$  unirComercios(com)
casasult  $\equiv$  unirCasas(ult)
comerult  $\equiv$  unirComercios(ult)
izq  $\equiv$  claves(casascom)  $\cup$  claves(comercom)
der  $\equiv$  claves(casasult)  $\cup$  claves(comerult)
```



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

$\_ \text{maxcons} : \text{estr } e \times \text{conj}(\text{Pos}) \ c \times \text{conj}(\text{Pos}) \ \text{max} \times \text{Nat } n \longrightarrow \text{conj}(\text{Pos}) \quad \{\text{eq. maxcons}\}$

$\_ \text{maxcons}(e, c, \text{max}, n) \equiv \text{if vacio?}(c) \text{ then}$   
 $\quad \text{max}$   
 $\quad \text{else if nivel}_i > n \text{ then}$   
 $\quad \quad \_ \text{maxcons}(e, \text{sinUno}(c), \text{Ag}(\text{pos}_i, \emptyset), \text{nivel}_i)$   
 $\quad \text{else if nivel}_i = n \text{ then}$   
 $\quad \quad \_ \text{maxcons}(e, \text{sinUno}(c), \text{Ag}(\text{pos}_i, \text{max}), n)$   
 $\quad \text{else}$   
 $\quad \quad \_ \text{maxcons}(e, \text{sinUno}(c), \text{max}, n)$   
 $\quad \text{fi}$

donde

$\text{pos}_i \equiv \text{dameUno}(c)$   
 $\text{nivel}_i \equiv \text{nivel}(e, \text{pos}_i)$

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

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

donde:

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

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

$\text{nivelesPorUnion}(e, pos) \equiv \text{if def?}(pos, \text{Construcciones}) \text{ then}$   
 $\quad e.\text{turno} - \text{obtener}(pos, \text{Construcciones})$   
 $\quad \text{else}$   
 $\quad e.\text{turno} - \text{hijoCorrecto}.\text{turnoUnion} + \text{nivelesPorUnion}(\text{hijoCorrecto}.\text{sc}, pos)$   
 $\quad \text{fi}$

donde:

$\text{Construcciones} \equiv \text{it}(e.\text{construcciones}, \text{"casa"}) \cup_{\text{dicc}} \text{it}(e.\text{construcciones}, \text{"comercio"})$   
 $\text{hijoCorrecto} \equiv \text{llegar}(e.\text{uniones}, pos)$

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

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

donde

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

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

$\text{manhattan}(e, p, d) \equiv \text{if vacio?}(\text{claves}(d)) \text{ then}$   
 $\quad 0$   
 $\quad \text{else if distManhattan}(p, \text{proximo}) \leq 3 \wedge p \neq \text{proximo} \text{ then}$   
 $\quad \quad \text{max}(\text{nivel}(e, \text{proximo}), \text{manhattan}(p, \text{borrar}(\text{proximo}, d))$   
 $\quad \text{else}$   
 $\quad \quad \text{manhattan}(e, p, \text{borrar}(\text{proximo}, d))$   
 $\quad \text{fi}$

donde

proximo  $\equiv$  dameUno(claves( $d$ ))

posiciones : estr  $e \longrightarrow \text{dicc}(\text{Pos} \times \text{Nat})$

$\{(\forall h : \text{hijo})(h \in e.\text{uniones} \Rightarrow_{\text{L}} h.\text{sc} \neq \text{NULL} \wedge_{\text{L}} \text{rep}(*h.\text{sc}))\}$

posiciones( $e$ )  $\equiv$  claves(unirCasas(Ag(& $e$ , unirPunteros( $e.\text{uniones}$ ))))  $\cup$   
 claves(unirComercios(Ag(& $e$ , unirPunteros( $e.\text{uniones}$ ))))

nivelar : estr  $e \times \text{dicc}(\text{Pos} \times \text{Nat}) \ d \longrightarrow \text{dicc}(\text{Pos} \times \text{Nat})$

$\{(\forall p : \text{Pos})(p \in \text{claves}(d) \Rightarrow_{\text{L}} p \in \text{posiciones}(e))\}$

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

donde

clave  $\equiv$  dameUno(claves( $d$ ))

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


---

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

```

1: estr.turno  $\leftarrow$  0
2: estr.antigüedad  $\leftarrow$  0
3: estr.popularidad  $\leftarrow$  0
4: estr.mapa  $\leftarrow$  m
5: estr.construcciones  $\leftarrow$  vacio()
6: estr.uniones  $\leftarrow$  vacia()
7: return estr

```

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


---

**avanzarTurno**(inout SimCity s, in dicc(Pos, Construcccion) cs)

```

1: s.turno  $\leftarrow$  s.turno + 1
2: s.construcciones.agregarAtras(direccion(cs))

```

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


---

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

```

1: s1.popularidad  $\leftarrow$  s1.popularidad + s2.popularidad + 1
2: s1.antigüedad  $\leftarrow$  max(s1.antigüedad, s2.antigüedad)
3: hijo nuevoHijo  $\leftarrow$  <direccion(s2), s1.turno>
4: agregarAtras(s1.uniones, nuevoHijo)

```

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


---

Asumo que existe un conjunto  $U \equiv \{u_1, u_2, \dots, u_n\}$ 
tal que cada uno de esos  $u_i$  son los simcities que componen al simcity originalllamamos nodos :  $\#U$ llamamos sumConstrucciones :  $\sum_{i=1}^{nodos} (\text{copiar}(u_i.construcciones))$ llamamos sumMapas :  $\sum_{i=1}^{nodos} (u_i.mapa)$ 


---

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

```

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

```

**Complejidad:**  $O(\text{sumMapas})$ 

cabe aclarar que la suma de los mapas esta definida

---

**listDeTipo**(in SimCity s, in Construcccion tipo)  $\rightarrow$  res : dicc(Pos, Nivel)

```

1: dicc(pos, nivel) res  $\leftarrow$  vacio()
2: for(nat i  $\leftarrow$  0; i < long(s.construcciones); i  $\leftarrow$  i + 1) :
3:   itDicc(Pos, Construcccion) itCs  $\leftarrow$  crearIt(*s.construcciones[i]);
4:   while(haySiguiente(itCs)) :
5:     Pos p  $\leftarrow$  siguienteClave(itCs)
6:     construccion c  $\leftarrow$  siguienteSignificado(itCs)
7:     if(c == tipo) :
8:       definir(res, p, s.turno - (i+1))
9:     avanzar(itCs)

```

**Complejidad:**  $O(\sum_{i=0}^{long(s.construcciones)} (\# \text{ claves}(*s.construcciones[i])))$ 


---

**casas**(in SimCity s)  $\rightarrow$  res : dicc(Pos, Nivel)

```

1: dicc(pos, nivel) res  $\leftarrow$  copiar(listDeTipo(s, "casa"))

```

---

```

2: dicc(pos, nivel) comerciosTotales ← copiar(listDeTipo(s, "comercio"))
3: for(nat i ← 0; i < long(s.uniones); i ← i + 1) :
4:   itDicc(Pos, Nivel) itCs ← crearIt(casas(*s.uniones[i].sc))
5:   while(haySiguiente(itCs)) :
6:     Pos p ← siguienteClave(itCs)
7:     Nivel n ← siguienteSignificado(itCs)
8:     if(¬def?(res, p) ∧ ¬def?(comerciosTotales, p)) :
9:       definir(res, p, s.turno - s.uniones[i].turnoUnido + n)
10:    avanzar(itCs)
11: comerciosTotales ← comerciosTotales ∪ listDeTipo(*s.uniones[i].sc, "comercio")
12: return res
Complejidad: O([sumConstrucciones]2 + nodos)

```

---

```

comercios(in SimCity s) → res : dicc(Pos, Nivel)
1: dicc(Pos, Nivel) casasTotales ← casas(s)
2: return manhatizar(comerciosAux(s, casasTotales), casasTotales)
Complejidad: O(manhatizar(comerciosAux(s, casasTotales), casasTotales) + casas(s))

```

```

comerciosAux(in SimCity s, in casasTotales) → res : dicc(Pos, Nivel)
1: dicc(pos, nivel) res ← copiar(listDeTipo(s, "casa"))
3: for(nat i ← 0; i < long(s.uniones); i ← i + 1) :
4:   itDicc(Pos, Nivel) itCs ← crearIt(comerciosAux(*s.uniones[i].sc), casasTotales)
5:   while(haySiguiente(itCs)) :
6:     Pos p ← siguienteClave(itCs)
7:     Nivel n ← siguienteSignificado(itCs)
8:     if(¬def?(res, p) ∧ ¬def?(casasTotales, p)) :
9:       definir(res, p, s.turno - s.uniones[i].turnoUnido + n)
10:    avanzar(itCs)
11: return res
Complejidad: O([sumConstrucciones]2 + nodos)

```

```

manhatizar(inout dicc(Pos, Nivel) comercios, in dicc(Pos, Nivel) casasTotales) {
1: itDicc(Pos, Nivel) itCs ← crearIt(comercios)
2: while(haySiguiente(itCs)) :
3:   Pos p ← siguienteClave(itCs)
4:   Nivel n ← siguienteSignificado(itCs)
5:   eliminarSiguiente(itCs)
6:   definirRapido(comercios, p, max(n, nivelCom(p, casasTotales)))
7:   avanzar(itCs)
}
Complejidad: O(#claves(comercios) * #claves(casasTotales))

```

---

```

popularidad(in SimCity s) → res : Nat
1: return s.popularidad
Complejidad: O(1)

```

---

```

nivelCom(in Pos p, in dicc(pos, Nivel) cs) → Nat
1: nat maxLvl ← 1
2: for(int i = -3; i ≤ 3; ++i) :
3:   for(int j = |i|-3; j ≤ 3-|i|; ++j) :
4:     if(p.x + i ≥ 0 ∧ p.y + j ≥ 0) :
5:       Pos p2 ← <p.x+i, p.y+j>
6:       if(def?(cs, p2)) :
7:         maxLvl = max(maxLvl, obtener(cs, p2))
8: return maxLvl
Complejidad: O(#claves(cs))

```

---

---

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

1: definirRapido(casas, p, n)

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

---

---

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

1: definirRapido(comercio, p, n)

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

---

---

**turnos**(in SimCity s)  $\rightarrow$  res : Nat

1: **return** s.turno

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

---

---

**$\bullet \cup \bullet$** (in dicc( $\alpha$ ,  $\beta$ ) d1, in dicc( $\alpha$ ,  $\beta$ ) d2)  $\rightarrow$  res : dicc( $\alpha$ ,  $\beta$ )

1: dicc( $\alpha$ ,  $\beta$ ) res = copiar(d1)

2: itDicc( $\alpha$ ,  $\beta$ ) itCs  $\leftarrow$  crearIt(d2);

3: **while**(haySiguiente(itCs)) :

4:    $\alpha$  a  $\leftarrow$  siguienteClave(itCs)

5:    $\beta$  b  $\leftarrow$  siguienteSignificado(itCs)

6:   **if**( $\neg$ def?(res, a)) :

7:     definir(res, a, b)

8:   avanzar(itCs)

9: **return** res

**Complejidad:**  $O(\text{copy}(d1) + \#claves(d2))$

---

---

**construcc**(in SimCity s)  $\rightarrow$  res : Nat

1: **return** casas(s)  $\cup$  comercios(s)

**Complejidad:**  $O(\text{casas}(d1) + \text{comercios}(d2))$

---

## 2.4. Módulo Servidor

### 2.4.1. Interfaz

#### Interfaz

**usa:** SimCity, Mapa, Pos, Nombre, Construcción, Diccionario Trie, Diccionario Lineal

**exporta:** todo

**se explica con:** SERVIDOR

**géneros:** server

#### Operaciones básicas de server

Sea  $S$ : *servidor*, donde  $N$  es la cantidad de partidas definidas en  $S$ ,  $nom_i$  es el nombre de la partida  $i$  y  $sc_i$  es el SimCity asociado a  $nom_i$ .

**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)**  $\rightarrow res : diccTrie(Nombre, SimCity)$

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

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

**Complejidad:**  $O(\sum_{i=0}^N copy(nom_i) + copy(sc_i))$

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

**Aliasing:** Por copia.

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

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

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

**Complejidad:**  $O(\sum_{i=0}^N copy(nom_i))$

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

**Aliasing:** Por copia.

**NUEVAPARTIDA(in/out s: server, in p: Nombre, in m: Mapa)**

**Pre**  $\equiv \{\hat{s} =_{obs} s_0 \wedge \neg def?(\hat{p}, partidas(\hat{s}))\}$

**Post**  $\equiv \{\hat{s} =_{obs} nuevaPartida(s_0, \hat{p}, \hat{m})\}$

**Complejidad:**  $O(copy(p) + copy(m))$

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

**Aliasing:** No tiene.

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

**Pre**  $\equiv \{unionValida^1(\hat{s}, \hat{p1}, \hat{p2}) \wedge \hat{s} = s_0\}$

**Post**  $\equiv \{\hat{s} =_{obs} unirPartidas(s_0, \hat{p1}, \hat{p2})\}$

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

**Descripción:** Agrega el SimCity  $s2$  asociado a  $p2$ , al SimCity  $s1$ , asociado a  $p1$ .  $s2$  pasa a ser no modificable.

**Aliasing:** Se guarda una referencia a  $s2$  en  $s1$ . Cualquier cambio sobre  $s2$  modificará la representación de  $s1$ . Se garantiza que  $s1$  no modificará a  $s2$ .

**AVANZARTURNOPARTIDA(in/out s: server, in p: Nombre, in cs: dicc(Pos  $\times$  Construcción))**

**Pre**  $\equiv \{avanzarValido^1(\hat{s}, \hat{p}, \hat{cs}) \wedge \hat{s} = s_0\}$

**Post**  $\equiv \{\hat{s} =_{obs} avanzarTurnoPartida(s_0, \hat{p}, \hat{cs})\}$

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

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

**Aliasing:** Se guarda un puntero al diccionario en el SimCity. Cualquier cambio en este modificará a la partida.

AGREGAR\_CASA(**in/out**  $s$ : server, **in**  $p$ : Nombre, **in**  $pos$ : Pos)

**Pre**  $\equiv \{\hat{s} = s_0 \wedge \text{agregarValido}^1(\hat{s}, \hat{p}, \hat{pos})\}$

**Post**  $\equiv \{\hat{s} =_{\text{obs}} \text{agregarCasa}(s_0, \hat{p}, \hat{pos})\}$

**Complejidad:**  $O(|\text{nombreMasLargo}|)$

**Descripción:** Agrega una nueva casa.

**Aliasing:** No tiene.

AGREGAR\_COMERCIO(**in/out**  $s$ : server, **in**  $p1$ : Nombre, **in**  $p2$ : Nombre)

**Pre**  $\equiv \{\hat{s} = s_0 \wedge \text{agregarValido}^1(\hat{s}, \hat{p}, \hat{pos})\}$

**Post**  $\equiv \{\hat{s} =_{\text{obs}} \text{agregarComercio}(s_0, \hat{p}, \hat{pos})\}$

**Complejidad:**  $O(|\text{nombreMasLargo}|)$

**Descripción:** Agrega un nuevo comercio.

**Aliasing:** No tiene.

VER\_POPULARIDAD(**in**  $s$ : server, **in**  $p$ : Nombre)  $\rightarrow res$ : Nat

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

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

**Complejidad:**  $O(|\text{nombreMasLargo}|)$

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

**Aliasing:** Devuelve una referencia no modificable.

VER\_TURNO(**in**  $s$ : server, **in**  $p$ : Nombre)  $\rightarrow res$ : Nat

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

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

**Complejidad:**  $O(|\text{nombreMasLargo}|)$

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

**Aliasing:** Devuelve una referencia no modificable.

VER\_MAPA(**in**  $s$ : server, **in**  $p$ : Nombre)  $\rightarrow res$ : Mapa

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

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

**Complejidad:**  $O(|\text{nombreMasLargo}|) + O(\text{mapa}(sc))$ , donde  $sc$  es el SimCity asociado a  $p$

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

**Aliasing:** Devuelve una copia.

VER\_CASAS(**in**  $s$ : server, **in**  $p$ : Nombre)  $\rightarrow res$ : dicc(Pos, Nat)

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

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

**Complejidad:**  $O(|\text{nombreMasLargo}|) + O(\text{casas}(sc))$ , donde  $sc$  es el SimCity asociado a  $p$

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

**Aliasing:** Devuelve una copia.

VER\_COMERCIOS(**in**  $s$ : server, **in**  $p$ : Nombre)  $\rightarrow res$ : dicc(Pos, Nat)

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

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

**Complejidad:**  $O(|\text{nombreMasLargo}|) + O(\text{comercios}(sc))$ , donde  $sc$  es el SimCity asociado a  $p$

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

**Aliasing:** Devuelve una copia.

1. definido en Definiciones Auxiliares de Servidor.

### 2.4.2. 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 y su estado (si es modificable o no).

Elegimos esta estructura para cumplir con las restricciones dadas de complejidad. Buscar una clave en el diccTrie está acotado por la clave más larga definida en el mismo. Con lo cual, todas las operaciones del servidor en relación a una partida específica serán por lo menos  $\mathbf{O}(|\text{nombreMasLargo}|)$ .

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

donde partida es tupla(*modificable* : bool ,  
                                  *sim* : puntero(SimCity) )

Rep : estr  $\longrightarrow$  boolean

Rep(*e*)  $\equiv$  true  $\iff$   
 $(\forall \text{partida}_1, \text{partida}_2 : \text{Nombre})$   
 $((\text{def?}(\text{partida}_1, e) \wedge \text{def?}(\text{partida}_2, e) \wedge \text{partida}_1 \neq \text{partida}_2) \Rightarrow_L$   
 $\quad \text{obtener}(\text{partida}_1, e).\text{sim} \neq \text{obtener}(\text{partida}_2, e).\text{sim})$   
 $) \wedge$   
 $(\forall \text{partida} : \text{Nombre})(\text{def?}(\text{partida}, e) \Rightarrow_L \text{obtener}(\text{partida}, e).\text{sim} \neq \text{NULL})$

Abs : estr *e*  $\longrightarrow$  servidor

{Rep(*e*)}

Abs(*e*)  $\equiv$  s: servidor |  
 $(\forall \text{nombre} : \text{Nombre})$   
 $(\text{nombre} \in \text{congelados}(s) \iff$   
 $\quad (\text{def?}(\text{nombre}, e) \wedge_L \neg \text{obtener}(\text{nombre}, e).\text{modificable}))$   
 $\wedge$   
 $(\forall \text{nombre} : \text{Nombre})$   
 $(\text{def?}(\text{nombre}, \text{partidas}(s)) \iff \text{def?}(\text{nombre}, e))$   
 $\wedge_L$   
 $(\forall \text{nombre} : \text{Nombre})$   
 $(\text{def?}(\text{nombre}, \text{partidas}(s)) \Rightarrow_L$   
 $\quad \text{obtener}(\text{nombre}, \text{partidas}(s)) =_{\text{obs}} *(\text{obtener}(\text{nombre}, e).\text{sim}))$



## 2.4.3. Implementación

## Algoritmos

---



---

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

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


---



---



---

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

 1:  $res \leftarrow \text{vacío}()$   
 2:  $it \leftarrow \text{crearIt}(e)$   
 3: **while**( $\text{haySiguiente}(it)$ ) :  
 4:      $nom \leftarrow \text{siguienteClave}(it)$   
 5:      $sc \leftarrow *(\text{siguienteSignificado}(it).sim)$   
 6:      $\text{definirRapido}(res, nom, sc)$   
 7:      $\text{avanzar}(it)$   
 8: **return**  $res$ 
Complejidad:  $\mathbf{O}(\sum_{i=0}^N (\text{copy}(nom_i) + \text{copy}(sc_i)))$ 


---



---



---

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

 1:  $res \leftarrow \text{vacío}()$   
 2:  $it \leftarrow \text{crearIt}(e)$   
 3: **while**( $\text{haySiguiente}(it)$ ) :  
 4:     **if** ( $\neg \text{siguienteSignificado}(it).modificable$ ) :  
 5:          $nom \leftarrow \text{siguienteClave}(it)$   
 6:          $\text{agregarRapido}(res, nom)$   
 7:      $\text{avanzar}(it)$   
 8: **return**  $res$ 
Complejidad:  $\mathbf{O}(\sum_{i=0}^N \text{copy}(nom_i))$ 


---



---



---

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

 1:  $\text{definirRapido}(e, p, \langle \text{true}, \&(\text{iniciar}(m)) \rangle)$  ▷ Reservamos memoria para el nuevo SimCity
Complejidad:  $\mathbf{O}(\text{copy}(p) + \text{copy}(m))$ 


---



---



---

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

 1:  $\text{definir}(e, p1, \langle \text{true}, \&(\text{unir}(*(\text{significado}(p1, e).sim), *(\text{significado}(p2, e).sim))) \rangle)$   
 2:  $\text{definir}(e, p2, \langle \text{false}, \text{significado}(p2, e).sim \rangle)$ 
Complejidad:  $\mathbf{O}(|\text{nombreMasLargo}|)$ 
Justificacion:  $\text{unir} \in \mathbf{O}(1) + \text{buscar en diccTrie} \in \mathbf{O}(|\text{nombreMasLargo}|) \rightarrow \mathbf{O}(|\text{nombreMasLargo}|)$ 


---



---



---

**avanzarTurnoPartida(in/out**  $e : \text{estr}$ , **in**  $p : \text{Nombre}$ , **in**  $cs : \text{diccLineal}(\text{Pos} \times \text{Construccion})$  **)**

 1:  $\text{definir}(e, p, \langle \text{true}, \&(\text{avanzarTurno}(*(\text{significado}(p, e).sim), cs)) \rangle)$  ▷  $cs$  por referencia
Complejidad:  $\mathbf{O}(|\text{nombreMasLargo}|)$ 
Justificacion:  $\text{avanzarTurno} \in \mathbf{O}(1) + \text{buscar en diccTrie} \in \mathbf{O}(|\text{nombreMasLargo}|)$   
 $\rightarrow \mathbf{O}(|\text{nombreMasLargo}|)$ 


---

---



---

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

- 1:  $cs \leftarrow definirRapido(vacio(), pos, "casa")$
- 2:  $definir(e, p, (true, \&(avanzarTurno(*significado(p, e).sim), cs)))$

Complejidad:  $O(|nombreMasLargo|)$

---



---



---

**agregarComercio**(in/out  $s$ : estr, in  $partida$ : String, in  $pos$ : Pos)

- 1:  $cs \leftarrow definirRapido(vacio(), pos, "comercio")$
- 2:  $definir(e, p, (true, \&(avanzarTurno(*significado(p, e).sim), cs)))$

Complejidad:  $O(|nombreMasLargo|)$

---



---



---

**verMapa**(in  $s$ : estr, in  $partida$ : Nombre)  $\rightarrow res$ : Mapa

- 1:  $sc \leftarrow obtener(partida, s).sim$
- 2:  $res \leftarrow mapa(*sc)$
- 3: **return**  $res$

Complejidad:  $O(|nombreMasLargo|) + O(mapa(*sc))$

---



---



---

**verCasas**(in  $s$ : estr, in  $partida$ : Nombre)  $\rightarrow res$ : DiccLineal(Pos, Nivel)

- 1:  $sc \leftarrow obtener(partida, s).sim$
- 2:  $res \leftarrow casas(*sc)$
- 3: **return**  $res$

Complejidad:  $O(|nombreMasLargo|) + O(casas(*sc))$

---



---



---

**verComercios**(in  $s$ : estr, in  $partida$ : Nombre)  $\rightarrow res$ : DiccLineal(Pos, Nivel)

- 1:  $sc \leftarrow obtener(partida, s).sim$
- 2:  $res \leftarrow comercios(*sc)$
- 3: **return**  $res$

Complejidad:  $O(|nombreMasLargo|) + O(comercios(*sc))$

---



---



---

**verPopularidad**(in  $s$ : estr, in  $partida$ : Nombre)  $\rightarrow res$ : Nat

- 1:  $sc \leftarrow obtener(partida, s).sim$
- 2:  $res \leftarrow popularidad(*sc)$
- 3: **return**  $res$

Complejidad:  $O(|nombreMasLargo|)$

---



---



---

**verTurno**(in  $s$ : estr, in  $partida$ : Nombre)  $\rightarrow res$ : Nat

- 1:  $sc \leftarrow obtener(partida, s).sim$
- 2:  $res \leftarrow turnos(*sc)$
- 3: **return**  $res$

Complejidad:  $O(|nombreMasLargo|)$

---