



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

1. Especificación	2
1.1. Aliases	2
1.2. Mapa	2
1.3. SimCity	3
1.4. Servidor	6
2. Diseño	8
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(α), 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

$\text{popularidad}(\text{unir}(s, s')) \equiv \text{popularidad}(s) + 1 + \text{popularidad}(s')$
 $\text{turnos}(\text{iniciar}(m)) \equiv 0$
 $\text{turnos}(\text{avanzarTurno}(s, cs)) \equiv \text{turnos}(s) + 1$
 $\text{turnos}(\text{unir}(s, s')) \equiv \text{if } \text{turnos}(s) < \text{turnos}(s') \text{ then } \text{turnos}(s') \text{ else } \text{turnos}(s) \text{ fi}$

otros ax. $\forall s: \text{simcity}, \forall p, q: \text{Pos}, \forall cs, cs': \text{dicc}(\text{Pos} \times \text{Construcción}), \forall cn, cn': \text{dicc}(\text{Pos} \times \text{Nivel}), \forall d, d': \text{dicc}(\alpha \times \beta)$

$\text{construcciones}(s) \equiv \text{casas}(s) \cup_{\text{dicc}} \text{comercios}(s)$
 $\text{agCasas}(cn, cs) \equiv \text{if } \text{vacio?}(\text{claves}(cs)) \text{ then}$
 $\quad \text{cn}$
 $\quad \text{else if } \text{obtener}(\text{proximo}, cs) = \text{"casa"} \text{ then}$
 $\quad \quad \text{agCasas}(\text{definir}(\text{proximo}, 1, cn), \text{borrar}(\text{proximo}, cs))$
 $\quad \text{else}$
 $\quad \quad \text{agCasas}(cn, \text{borrar}(\text{proximo}, cs))$
 $\quad \text{fi}$
 $\quad \text{donde } \text{proximo} \equiv \text{dameUno}(\text{claves}(cs))$
 $\text{agComercios}(cn, cs) \equiv \text{if } \text{vacio?}(\text{claves}(cs)) \text{ then}$
 $\quad \text{cn}$
 $\quad \text{else if } \text{obtener}(\text{proximo}, cs) = \text{"comercio"} \text{ then}$
 $\quad \quad \text{agComercios}(\text{definir}(\text{proximo}, 1, cn), \text{borrar}(\text{proximo}, cs))$
 $\quad \text{else}$
 $\quad \quad \text{agComercios}(cn, \text{borrar}(\text{proximo}, cs))$
 $\quad \text{fi}$
 $\quad \text{donde } \text{proximo} \equiv \text{dameUno}(\text{claves}(cs))$
 $\text{nivelComercio}(p, n, cn) \equiv \text{if } \text{vacio?}(\text{claves}(cn)) \text{ then}$
 $\quad \text{n}$
 $\quad \text{else if } \text{distManhattan}(p, \text{proximo}) \leq 3 \wedge \text{obtener}(\text{proximo}, cn) > n \text{ then}$
 $\quad \quad \text{nivelComercio}(p, \text{obtener}(\text{proximo}, cn), \text{borrar}(\text{proximo}, cn))$
 $\quad \text{else}$
 $\quad \quad \text{nivelComercio}(p, n, \text{borrar}(\text{proximo}, cn))$
 $\quad \text{fi}$
 $\quad \text{donde } \text{proximo} \equiv \text{dameUno}(\text{claves}(cn))$
 $\text{distManhattan}(p, q) \equiv \text{if } p.x < q.x \text{ then } q.x - p.x \text{ else } p.x - q.x \text{ fi}$
 $\quad +$
 $\quad \text{if } p.y < q.y \text{ then } q.y - p.y \text{ else } p.y - q.y \text{ fi}$
 $d \cup_{\text{dicc}} d' \equiv \text{if } \text{vacio?}(\text{claves}(d')) \text{ then}$
 $\quad d$
 $\quad \text{else if } \neg \text{def?}(\text{proximo}, d) \text{ then}$
 $\quad \quad \text{definir}(\text{proximo}, \text{obtener}(\text{proximo}, d'), d) \cup_{\text{dicc}} \text{borrar}(\text{proximo}, d')$
 $\quad \text{else}$
 $\quad \quad d \cup_{\text{dicc}} \text{borrar}(\text{proximo}, d')$
 $\quad \text{fi}$
 $\quad \text{donde } \text{proximo} \equiv \text{dameUno}(\text{claves}(d'))$
 $\text{sacarRepes}(cs, cs') \equiv \text{if } \text{vacio?}(\text{claves}(cs)) \text{ then}$
 $\quad cs'$
 $\quad \text{else if } \text{def?}(\text{proximo}, cs') \text{ then}$
 $\quad \quad \text{sacarRepes}(\text{borrar}(\text{proximo}, cs), \text{borrar}(\text{proximo}, cs'))$
 $\quad \text{else}$
 $\quad \quad \text{sacarRepes}(\text{borrar}(\text{proximo}, cs), cs')$
 $\quad \text{fi}$
 $\quad \text{donde } \text{proximo} \equiv \text{dameUno}(\text{claves}(cs))$

1. definido en el apartado Definiciones Auxiliares de SimCity.
2. 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 podrían 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
unirConst(cs, cn, cn') ≡ if vacio?(cn') then
                        cn
                        else if ¬ def?(proximo, cs) then
                        unirConst(cs,
                                definir(proximo, obtener(proximo, cn'), cn),
                                borrar(proximo, cn'))
                        else
                        unirConst(cs, cn, borrar(proximo, cn'))
                        fi
                        donde proximo ≡ dameUno(claves(cn'))
manhatizar(cn, cn')  ≡ if vacio?(cn') then
                        vacio
                        else
                        definir(proximo,
                                nivelComercio(proximo, obtener(proximo, cn'), cn),
                                manhatizar(cn, borrar(proximo, cn')))
                        fi
                        donde proximo ≡ dameUno(claves(cn'))

```

Fin TAD

Definiciones Auxiliares de SimCity

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

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

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

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

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

géneros	Server
exporta	Server, observadores, generadores, verMapa, verCasas, verComercios, verPopularidad, agregarCasa, agregarComercio y verTurno
usa	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. $\forall s$: Server, $\forall p$: Nombre, $\forall pos$: Pos

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

Fin TAD

Definiciones Auxiliares de Servidor

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

$unionValida(s, p, p') \equiv def?(p, partidas(s)) \wedge def?(p', partidas(s)) \wedge p \notin congeladas(s) \wedge_L$
 $unirValido(obtener(p, partidas(s)), obtener(p', partidas(s)))^1$

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

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

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

$agregarValido(s, p, pos) \equiv def?(p, partidas(s)) \wedge p \notin congeladas(s) \wedge_L$
 $\neg def?(pos, verCasas(s, p)) \wedge \neg def?(pos, verComercios(s, p)) \wedge$
 $\neg esRio(pos, verMapa(s, p))$

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 `STRING`

NOMBRE es `STRING`

NIVEL es `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** estr

donde 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 : conj(Nat), in vs : conj(Nat)) \longrightarrow res : estr

1: res.horizontales \leftarrow hs

2: res.verticales \leftarrow vs

3: **return** res

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

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

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

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

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

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

1: **return** m.verticales

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

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

1: **return** m.horizontales

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

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 construcciones_i es el conjunto de casas y comercios definidos en u_i por medio de *AvanzarTurno*.²

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 mapa_i es el Mapa original³ 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 : dicccLineal(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 ≤ 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.

Cabe recordar que para la resolución de colisiones en las uniones, tiene prioridad el nodo izquierdo.

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)

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

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

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

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(punero(estr)) $\{(\forall h : \text{hijo})(h \in s \Rightarrow_L h.sc \neq \text{NULL})\}$

// crea un conjunto de punteros a todos los simcities en la unión.

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

_unirPunteros : secu(hijo) $s \times \text{conj}(\text{punero}(\text{estr})) \text{ } ps \longrightarrow \text{conj}(\text{punero}(\text{estr}))$ {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 sombrero).
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  → Mapa
                                                    { (∀p : puntero(estr)) (p ∈ ps ⇒L (p ≠ NULL ∧L rep(*p))) }

unirMapas(ps) ≡ if vacio?(ps) then
                  crear(∅, ∅)
                else
                  (*(dameUno(ps))).mapa + UnirMapas(sinUno(ps))
                fi

unirCasas : conj(puntero(estr)) ps  → dicc(Pos × Nat)
                                                    {eq. unirMapas}
// crea el diccionario de casas en la unión cuyo significado es igual al turno individual en
// que se agregó la casa a uno de los simCities en particular.
unirCasas(ps) ≡ if vacio?(ps) then
                  vacio
                else
                  it(*(dameUno(ps))).construcciones, "casa") ∪dicc unirCasas(sinUno(ps))
                  // notar que ∪dicc prioriza el argumento izquierdo.
                fi

unirComercios : conj(puntero(estr)) ps  → dicc(Pos × Nat)
                                                    {eq. unirMapas}
// crea el diccionario de comercio en la unión cuyo significado es igual al turno individual
// en que se agregó la casa a uno de los simCities en particular.
unirComercios(ps) ≡ if vacio?(ps) then
                     vacio
                   else
                     it(*(dameUno(ps))).construcciones, "comercio") ∪dicc unirComercios(sinUno(ps))
                   fi

it : secu(puntero(dicc(Pos × Construcccion))) s × Construcccion c  → dicc(Pos × Nat)
// el significado representa el turno en que se agregó la construcción.
it(s, c) ≡ _it(s, c, 0)

_it : secu(puntero(dicc(Pos × Construcccion))) s × Construcccion c × Nat n  → dicc(Pos × Nat)
_it(s, c, n) ≡ if vacia?(s) then vacio else filtrar(*(prim(s)), c, n) ∪dicc it(fin(s), c, n + 1) fi

filtrar : dicc(Pos × Construcccion) × Construcccion × Nat  → dicc(Pos × Nat)
filtrar(d, c, n) ≡ 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 ≡ dameUno(claves(d))

pos : secu(α) s × α a  → Nat
pos(s, a) ≡ if prim(s) = a then 0 else 1 + pos(fin(s), a) fi
                                                    {esta?(a, s)}

unionesValidas : estr e × secu(hijo) s  → bool
                                                    {s ⊆ e.uniones ∧L (∀h : hijo)(h ∈ s ⇒L h.sc ≠ NULL ∧L rep(*h.sc))}
unionesValidas(e, s) ≡ vacio?(s) ∨L (maxcons(e, izq) ∩ maxcons(e, der) = ∅ ∧ unionesValidas(e, com(s)))
donde
com      ≡ unirPunteros(com(s))
ult      ≡ ult(s) • <>
casascom ≡ unirCasas(com)
comercom ≡ unirComercios(com)
casasult ≡ unirCasas(ult)

```



```

comerult    ≡ unirComercios(ult)
izq         ≡ claves(casascom) ∪ claves(comercom)
der         ≡ claves(casasult) ∪ claves(comerult)

```

```

maxcons : estr e × conj(Pos) c → conj(Pos)      {(∀p : Pos)(p ∈ c ⇒L p ∈ posiciones(e))}
maxcons(e, c) ≡ _maxcons(e, c, ∅, 0)

```

```

_maxcons : estr e × conj(Pos) c × conj(Pos) max × Nat n → conj(Pos)      {eq. maxcons}
_maxcons(e, c, max, n) ≡ if vacio?(c) then
    max
  else if niveli > n then
    _maxcons(e, sinUno(c), Ag(posi, ∅), niveli)
  else if niveli = n then
    _maxcons(e, sinUno(c), Ag(posi, max), n)
  else
    _maxcons(e, sinUno(c), max, n)
fi

```

donde

```

posi    ≡ dameUno(c)
niveli  ≡ nivel(e, posi)

```

```

nivel : estr e × Pos pos → Nat      {p ∈ posiciones(e)}

```

// calcula el nivel propiamente de una construcción.

```

nivel(e, pos) ≡ if def?(pos, Casas) then
    nivelesPorUnion(e, pos)
  else
    max(nivelesPorUnion(e, pos), nManhattan)
fi

```

donde:

```

Unidos      ≡ unirPunteros(e.uniones)
Casas       ≡ unirCasas(Ag(&e, Unidos))
Comercios   ≡ unirComercios(Ag(&e, Unidos))
nManhattan  ≡ manhattan(e, pos, Casas)

```

```

nivelesPorUnion : estr e × Pos pos → Nat      {eq. nivel}

```

```

nivelesPorUnion(e, pos) ≡ if def?(pos, Construcciones) then
    e.turno - obtener(pos, Construcciones)
  else
    e.turno - hijoCorrecto.turnoUnion + nivelesPorUnion(hijoCorrecto.sc, pos)
fi

```

donde:

```

Construcciones ≡ it(e.construcciones, "casa") ∪dicc it(e.construcciones, "comercio")
hijoCorrecto  ≡ llegar(e.uniones, pos)

```

```

llegar : secu(hijo) s × Pos p → hijo  {(∀ h : hijo)(esta?(h, s) ⇒L h.sc ≠ NULL ∧L rep(*(h.sc))) ∧L
                                          (∃ h : hijo)(esta?(h, s) ∧L p ∈ posiciones(h.sc))}

```

// busca el SimCity inmediato en el cual está definida una posición.

```

llegar(s, p) ≡ if def?(pos, unirCasas(hijoi)) ∨ def?(pos, unirComercios(hijoi)) then
    prim(s)
  else
    llegar(fin(s))
fi

```

donde

```

hijoi ≡ Ag(prim(s).sc, ∅)

```

$\text{manhattan} : \text{estr } e \times \text{Pos } p \times \text{Dicc}(\text{Pos} \times \text{Nat}) \ d \longrightarrow \text{Conj}(\text{Pos})$
 $\{(\forall p' : \text{Pos})(p' \in \text{claves}(d) \Rightarrow_{\text{L}} p' \in \text{posiciones}(e))\}$
 $\text{manhattan}(e, p, d) \equiv \text{if } \text{vacio?}(\text{claves}(d)) \text{ then}$
 $\quad 0$
 $\quad \text{else if } \text{distManhattan}(p, \text{proximo}) \leq 3 \wedge p \neq \text{proximo} \text{ then}$
 $\quad \quad \text{max}(\text{nivel}(e, \text{proximo}), \text{manhattan}(p, \text{borrar}(\text{proximo}, d))$
 $\quad \text{else}$
 $\quad \quad \text{manhattan}(e, p, \text{borrar}(\text{proximo}, d))$
 $\quad \text{fi}$
 donde
 $\text{proximo} \equiv \text{dameUno}(\text{claves}(d))$

 $\text{posiciones} : \text{estr } e \longrightarrow \text{dicc}(\text{Pos} \times \text{Nat})$
 $\{(\forall h : \text{hijo})(h \in e.\text{uniones} \Rightarrow_{\text{L}} h.\text{sc} \neq \text{NULL} \wedge_{\text{L}} \text{rep}(*(\text{h.sc})))\}$
 $\text{posiciones}(e) \equiv \text{claves}(\text{unirCasas}(\text{Ag}(\&e, \text{unirPunteros}(e.\text{uniones})))) \cup$
 $\text{claves}(\text{unirComercios}(\text{Ag}(\&e, \text{unirPunteros}(e.\text{uniones}))))$

 $\text{nivelar} : \text{estr } e \times \text{dicc}(\text{Pos} \times \text{Nat}) \ d \longrightarrow \text{dicc}(\text{Pos} \times \text{Nat})$
 $\{(\forall p : \text{Pos})(p \in \text{claves}(d) \Rightarrow_{\text{L}} p \in \text{posiciones}(e))\}$
 $\text{nivelar}(e, d) \equiv \text{if } \text{vacio?}(d) \text{ then } \text{vacio} \text{ else } \text{definir}(\text{clave}, \text{nivel}(e, \text{clave}), \text{nivelar}(e, \text{borrar}(\text{clave}, d))) \text{ fi}$
 donde
 $\text{clave} \equiv \text{dameUno}(\text{claves}(d))$

2.3.3. 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(α , β) d1, in dicc(α , β) d2) \rightarrow res : dicc(α , β)

1: dicc(α , β) res = copiar(d1)

2: itDicc(α , β) itCs \leftarrow crearIt(d2);

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

4: α a \leftarrow siguienteClave(itCs)

5: β 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))`

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

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

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

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

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

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**($\text{haySiguiete}(it)$) :
 4: $nom \leftarrow \text{siguieteClave}(it)$
 5: $sc \leftarrow *(\text{siguieteSignificado}(it).sim)$
 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**($\text{haySiguiete}(it)$) :
 4: **if** ($\neg \text{siguieteSignificado}(it).modificable$) :
 5: $nom \leftarrow \text{siguieteClave}(it)$
 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: $O(|nombreMasLargo|)$
