



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. Módulo Mapa	8
2.1.1. Interfaz	8
2.1.2. Representación	9
2.1.3. Implementación	10
2.2. Módulo SimCity	11
2.2.1. Interfaz	11
2.2.2. Representación	13
2.2.3. Implementación	19
2.3. Módulo Servidor	22
2.3.1. Interfaz	22
2.3.2. Representación	25
2.3.3. Implementación	26

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 : Mapa × Pos \longrightarrow Bool

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

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

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

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

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

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

Fin TAD

1.3. SimCity

TAD SIMCITY

géneros SimCity

exporta SimCity, observadores, generadores, turnos, nivelComercio, distManhattan, • \cup_{dicc} •

usa Mapa, Nat, Pos, Construcción, Nivel, $dicc(\alpha \times \beta)$

igualdad observacional

$$(\forall s, s' : \text{SimCity}) \left(s =_{\text{obs}} s' \iff \left(\begin{array}{l} \text{mapa}(s) =_{\text{obs}} \text{mapa}(s') \wedge_{\text{L}} \\ \text{casas}(s) =_{\text{obs}} \text{casas}(s') \wedge \\ \text{comercios}(s) =_{\text{obs}} \text{comercios}(s') \wedge \\ \text{popularidad}(s) =_{\text{obs}} \text{popularidad}(s') \end{array} \right) \right)$$

observadores básicos

mapa : SimCity \longrightarrow Mapa
casas : SimCity \longrightarrow $dicc(\text{Pos} \times \text{Nivel})$
comercios : SimCity \longrightarrow $dicc(\text{Pos} \times \text{Nivel})$
popularidad : SimCity \longrightarrow Nat

generadores

iniciar : Mapa \longrightarrow SimCity
avanzarTurno : SimCity $s \times$ $dicc(\text{Pos} \times \text{Construcción})$ $cs \longrightarrow$ SimCity $\{\text{avanzarValido}(s, cs)^1\}$
unir : SimCity $a \times$ SimCity $b \longrightarrow$ SimCity $\{\text{unirValido}(a, b)^1\}$

otras operaciones

turnos : SimCity \longrightarrow Nat
nivelComercio : $\text{Pos} \times$ $dicc(\text{Pos} \times \text{Nivel}) \longrightarrow$ Nat
distManhattan : $\text{Pos} \times \text{Pos} \longrightarrow$ Nat
• \cup_{dicc} • : $dicc(\alpha \times \beta) \times$ $dicc(\alpha \times \beta) \longrightarrow$ $dicc(\alpha \times \beta)$
construcciones : SimCity \longrightarrow $dicc(\text{Pos} \times \text{Nivel})$
agCasas : $dicc(\text{Pos} \times \text{Nivel}) \times$ $dicc(\text{Pos} \times \text{Construcción}) \longrightarrow$ $dicc(\text{Pos} \times \text{Nivel})$
agComercios : SimCity \times $dicc(\text{Pos} \times \text{Nivel}) \times$ $dicc(\text{Pos} \times \text{Construcción}) \longrightarrow$ $dicc(\text{Pos} \times \text{Nivel})$
sacarRepes : $dicc(\text{Pos} \times \text{Construcción}) \times$ $dicc(\text{Pos} \times \text{Construcción}) \longrightarrow$ $dicc(\text{Pos} \times \text{Construcción})$

axiomas $\forall s, s' : \text{simcity}, \forall m : \text{Mapa}, \forall cs : dicc(\text{Pos} \times \text{Construcción})$

mapa(iniciar(m)) \equiv m
mapa(avanzarTurno(s, cs)) \equiv mapa(s)
mapa(unir(s, s')) \equiv mapa(s) + mapa(s')
casas(iniciar(m)) \equiv vacio
casas(avanzarTurno(s, cs)) \equiv agCasas(casas(s), cs)
casas(unir(s, s')) \equiv agCasas(casas(s), sacarRepes(construcciones(s), construcciones(s')))
comercios(iniciar(m)) \equiv vacio
comercios(avanzarTurno(s, cs)) \equiv agComercios(s , comercios(s), cs)
comercios(unir(s, s')) \equiv agComercios(s , comercios(s),
 sacarRepes(construcciones(s), construcciones(s')))
popularidad(iniciar(m)) \equiv 0
popularidad(avanzarTurno(s, cs)) \equiv popularidad(s)
popularidad(unir(s, s')) \equiv popularidad(s) + 1 + popularidad(s')

1. definido en el apartado Definiciones Auxiliares de SimCity.

```

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

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

nivelComercio(p, cn)      ≡ if vacio?(claves(cn)) then
                              1
                              else if distManhattan(p, proximo) ≤ 3 then
                                  max(obtener(proximo, cn), nivelComercio(p, borrar(proximo, cn)))
                              else
                                  nivelComercio(p, borrar(proximo, cn))
                              fi
                              donde proximo ≡ dameUno(claves(cn))
distManhattan(p, q)      ≡ if p.x < q.x then q.x - p.x else p.x - q.x fi
                              +
                              if p.y < q.y then q.y - p.y else p.y - q.y fi
d ∪dicc d'              ≡ if vacio?(claves(d')) then
                              d
                              else
                                  definir(proximo, obtener(proximo, d'), d ∪dicc borrar(proximo, d'))
                              fi
                              donde proximo ≡ dameUno(claves(d'))
construcciones(s)          ≡ casas(s) ∪dicc comercios(s)
agCasas(cn, cs)          ≡ if vacio?(claves(cs)) then
                              cn
                              else if obtener(proximo, cs) = "casa" then
                                  agCasas(definir(proximo, 1, cn), borrar(proximo, cs))
                              else
                                  agCasas(cn, borrar(proximo, cs))
                              fi
                              donde proximo ≡ dameUno(claves(cs))
agComercios(s, cn, cs)  ≡ if vacio?(claves(cs)) then
                              cn
                              else if obtener(proximo, cs) = "comercio" then
                                  agComercios(s, definir(proximo, nivelComercio(proximo, casas(s)), cn),
                                              borrar(proximo, cs))
                              else
                                  agComercios(s, cn, borrar(proximo, cs))
                              fi
                              donde proximo ≡ dameUno(claves(cs))
sacarRepes(cs, cs')      ≡ if vacio?(claves(cs)) then
                              cs'
                              else if def?(proximo, cs') then
                                  sacarRepes(borrar(proximo, cs), borrar(proximo, cs'))
                              else
                                  sacarRepes(borrar(proximo, cs), cs')
                              fi
                              donde proximo ≡ dameUno(claves(cs))

```

Fin TAD

Definiciones Auxiliares de SimCity

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

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

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

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

1.4. Servidor

TAD SERVIDOR

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

igualdad observacional

$$(\forall s, s' : \text{Server}) \left(s =_{\text{obs}} s' \iff \left(\begin{array}{l} \text{partidas}(s) =_{\text{obs}} \text{partidas}(s') \wedge_L \\ \text{congeladas}(s) =_{\text{obs}} \text{congeladas}(s') \wedge \\ (\forall p : \text{Nombre})(\text{def?}(p, \text{partidas}(s)) \Rightarrow_L \\ \text{pendientes}(s, p) =_{\text{obs}} \text{pendientes}(s', p)) \end{array} \right) \right)$$

observadores básicos

partidas	: Server	→	$\text{dicc}(\text{Nombre} \times \text{SimCity})$	
congeladas	: Server	→	$\text{conj}(\text{Nombre})$	
pendientes	: Server $s \times \text{Nombre } p$	→	$\text{dicc}(\text{Pos} \times \text{Construcción})$	$\{\text{def?}(p, \text{partidas}(s))\}$

generadores

nuevoServer	:	→	Server	
nuevaPartida	: Server $s \times \text{Nombre } p \times \text{Mapa}$	→	Server	$\{\neg \text{def?}(p, \text{partidas}(s))\}$
unirPartidas	: Server $s \times \text{Nombre } p1 \times \text{Nombre } p2$	→	Server	$\{\text{unionValida}(s, p1, p2)^1\}$
agregarCasa	: Server $s \times \text{Nombre } p \times \text{Pos } pos$	→	Server	$\{\text{agregarValido}(s, p, pos)^1\}$
agregarComercio	: Server $s \times \text{Nombre } p \times \text{Pos } pos$	→	Server	$\{\text{agregarValido}(s, p, pos)^1\}$
avanzarTurnoPartida	: Server $s \times \text{Nombre } p$	→	Server	$\{\text{avanzarValido}(s, p)^1\}$

otras operaciones

verMapa	: Server $s \times \text{Nombre } p$	→	Mapa	$\{\text{def?}(p, \text{partidas}(s))\}$
verCasas	: Server $s \times \text{Nombre } p$	→	$\text{dicc}(\text{Pos} \times \text{Nivel})$	$\{\text{def?}(p, \text{partidas}(s))\}$
verComercios	: Server $s \times \text{Nombre } p$	→	$\text{dicc}(\text{Pos} \times \text{Nivel})$	$\{\text{def?}(p, \text{partidas}(s))\}$
verPopularidad	: Server $s \times \text{Nombre } p$	→	Nat	$\{\text{def?}(p, \text{partidas}(s))\}$
verTurno	: Server $s \times \text{Nombre } p$	→	Nat	$\{\text{def?}(p, \text{partidas}(s))\}$

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

partidas(nuevoServer)	\equiv	vacio
partidas(nuevaPartida(s, p, m))	\equiv	definir(p , iniciar(m), partidas(s))
partidas(unirPartidas(s, p, p'))	\equiv	definir($p1$, unir(obtener($p1$, partidas(s)), obtener($p2$, partidas(s))), partidas(s))
partidas(agregarCasa(s, p, pos))	\equiv	partidas(s)
partidas(agregarComercio(s, p, pos))	\equiv	partidas(s)
partidas(avanzarTurnoPartida(s, p))	\equiv	definir(p , avanzarTurno(obtener(p , partidas(s)), pendientes(s, p)), partidas(s))
congeladas(nuevoServer)	\equiv	\emptyset
congeladas(nuevaPartida(s, p, m))	\equiv	congeladas(s)
congeladas(unirPartidas(s, p, p'))	\equiv	Ag(p' , congeladas(s))
congeladas(agregarCasa(s, p, pos))	\equiv	congeladas(s)
congeladas(agregarComercio(s, p, pos))	\equiv	congeladas(s)
congeladas(avanzarTurnoPartida(s, p))	\equiv	congeladas(s)

1. definido en el apartado Definiciones Auxiliares de Servidor.

```

pendientes(nuevaPartida(s, p, m), p')  ≡ if p = p' then vacio else pendientes(s, p') fi
pendientes(unirPartidas(s, p, p'), p'') ≡ pendientes(s, p'') acá no hace falta dif casos no?
pendientes(agregarCasa(s, p, pos), p') ≡ if p = p' then
                                         definir(pos, "casa", pendientes(s, p))
                                         else
                                         pendientes(s, p')
                                         fi
pendientes(agregarComercio(s, p, pos), p') ≡ if p = p' then
                                         definir(pos, "comercio", pendientes(s, p))
                                         else
                                         pendientes(s, p')
                                         fi
pendientes(avanzarTurnoPartida(s, p), p') ≡ if p = p' then vacio else pendientes(s, p') fi

```

otros ax. $\forall s$: Server, $\forall p$: Nombre

```

verMapa(s, p)      ≡ mapa(obtener(p, partidas(s)))
verCasas(s, p)     ≡ casas(obtener(p, partidas(s)))
verComercios(s, p) ≡ comercios(obtener(p, partidas(s)))
verPopularidad(s, p) ≡ popularidad(obtener(p, partidas(s)))
verTurno(s, p)     ≡ turnos(obtener(p, partidas(s)))

```

Fin TAD

Definiciones Auxiliares de Servidor

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

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

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

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

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

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

1. definido en el apartado Definiciones Auxiliares de SimCity.

2. Diseño

2.1. Módulo Mapa

2.1.1. Interfaz

Interfaz

se explica con: MAPA

géneros: mapa

Operaciones básicas de mapa

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

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

Post $\equiv \{res =_{\text{obs}} \text{mapa}(hs, vs)\}$

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

Descripción: crea un mapa

ESRIO(**in** $m1 : \text{Mapa}$, **in** $p : \text{Pos}$) $\rightarrow res : \text{Bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{esRio}(m1, p)\}$

Complejidad: $O(1)$

Descripción: verifica si en determinada pos hay rio

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

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

Post $\equiv \{res =_{\text{obs}} m1 + m2\}$

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

Descripción: une 2 mapas

2.1.2. Representación

Representación

Representación de mapa

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

mapa **se representa con** *estr*

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

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

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

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

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

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

2.1.3. Implementación

Algoritmos

crear(in $hs : \text{conj}(\text{Nat})$, in $vs : \text{conj}(\text{Nat})$) \longrightarrow $res : \text{estr}$
1: $\text{estr.horizontales} \leftarrow hs$
2: $\text{estr.verticales} \leftarrow vs$
3: **return** $estr$
Complejidad: $O(\text{copy}(hs) + \text{copy}(vs))$

Suma(in $m1: \text{mapa}$, in $m2: \text{mapa}$) \longrightarrow $res : \text{mapa}$
 $\text{mapa } res \leftarrow \text{copiar}(m1)$

 $\text{itConj}(\text{Nat}) \text{ itH} \leftarrow \text{crearIt}(\text{horizontales}(m2))$
 while($\text{haySiguiente}(\text{itH})$) :
 $\text{Ag}(\text{horizontales}(res), \text{siguiente}(\text{itH}))$
 $\text{avanzar}(\text{itH})$

 $\text{itConj}(\text{Nat}) \text{ itV} \leftarrow \text{crearIt}(\text{verticales}(m2))$
 while($\text{haySiguiente}(\text{itV})$) :
 $\text{Ag}(\text{verticales}(res), \text{siguiente}(\text{itV}))$
 $\text{avanzar}(\text{itV})$

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

esRio(in $m1: \text{mapa}$, in $p: \text{Pos}$) \longrightarrow $res : \text{Bool}$
 return $p.x \in \text{verticales}(m1) \vee p.y \in \text{horizontales}(m1)$
Complejidad: $O(\#\text{horizontales}(m1) + \#\text{verticales}(m1))$

2.2. Módulo SimCity

2.2.1. Interfaz

Interfaz

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

exporta: todo

se explica con: SIMCITY

géneros: SimCity

Operaciones básicas de SimCity

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

$\text{casas} = \#(\text{claves}(\text{casas}(\hat{S})))$ y $\text{comercios} = \#(\text{claves}(\text{comercios}(\hat{S})))$.

MAPA(in $S : \text{SimCity}$) $\rightarrow res : \text{Mapa}$

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

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

Complejidad: $\Theta(\sum_{i=0}^N \text{copy}(\text{mapa}_i))$, donde mapa_i es el Mapa original² de u_i .

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

Aliasing: No. Genera una copia.

CASAS(in $S : \text{SimCity}$) $\rightarrow res : \text{DiccLineal}(\text{Pos}, \text{Nivel})$

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

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

Complejidad: $\mathcal{O}(\text{casas}^2) \supset^3 \Theta(\sum_{i=0}^N (\text{hasta}_i \times \text{casas}_i + \text{camino}_i))$,

donde hasta_i y casas_i son respectivamente la cantidad de casas definidas⁴ en $\{u_0 \dots u_{i-1}\}$ y u_i ⁵,
y camino_i representa la cantidad de uniones para llegar de S a u_i ⁶.

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

Aliasing: No. Genera una copia.

COMERCIOS(in $S : \text{SimCity}$) $\rightarrow res : \text{DiccLineal}(\text{Pos}, \text{Nivel})$

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

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

Complejidad: $\mathcal{O}(\text{comercios}^2 \times \text{casas}) \supset \Theta(\sum_{i=0}^N (\text{hasta}_i \times \text{comercios}_i \times \text{casas} + \text{camino}_i))$,

donde hasta_i y comercios_i son respectivamente la cantidad de comercios definidos en $\{u_0 \dots u_{i-1}\}$ y u_i ,
y camino_i representa la cantidad de uniones para llegar de S a u_i ⁷.

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

Aliasing: No. Genera una copia.

POPULARIDAD(in $S : \text{SimCity}$) $\rightarrow res : \text{Nat}$

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

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

Complejidad: $\Theta(1)$

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

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

7. Similar a *casas(S)*. En este caso se agrega la posibilidad de tener que evaluar por pertenencia en el total de las casas al conjunto de posiciones a distancia manhattan ≤ 3 del comercio actualmente siendo copiado, para conocer su nivel.

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

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

Post $\equiv \{res =_{\text{obs}} \text{turnos}(\hat{S})\}$

Complejidad: $\Theta(1)$

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

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

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

Post $\equiv \{res =_{\text{obs}} \text{iniciar}(\hat{m})\}$

Complejidad: $\Theta(1)$

Descripción: crea un nuevo SimCity.

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

AVANZARTURNO(**in/out** $S: \text{SimCity}$, **in** $cs: \text{dicc}(\text{Pos}, \text{Construccion})$)

Pre $\equiv \{\text{avanzarTurnoValido}(\hat{s}, \hat{cs}) \wedge \hat{S} = S_0\}$

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

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

Descripción: avanza el turno de un SimCity.

Aliasing: genera una copia de las posiciones en el diccionario.

UNIR(**in/out** $S1: \text{SimCity}$, **in** $S2: \text{SimCity}$)

Pre $\equiv \{\text{unionValida}(\hat{S1}, \hat{S2}) \wedge \hat{S1} = S_0\}$

Post $\equiv \{\hat{S1} =_{\text{obs}} \text{unir}(S_0, \hat{S2})\}$

Complejidad: $\Theta(1)$

Descripción: Une dos SimCities.

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

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

2.2.2. Representación

Representación

SimCity se compone por la *ubicacion* y *nivel* de una serie de *construcciones*, de tipo *casa* o *comercio*, sobre un *Mapa*, y de una *popularidad* respecto a la cantidad de uniones que lo modificaron.

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

SimCity se representa con *estr*

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

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

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

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

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

donde

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

Abs : $\text{estr } e \longrightarrow \text{SimCity}$ {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

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

1. Se asume el traspaso de toda estructura de representación a su equivalente abstracto (se aplica el sombrerito).
2. la estructura no loopea consigo misma.
3. el turno actual es mayor o igual al turno de cualquier simCity hijo.
4. la popularidad es igual a la cantidad de uniones.
5. ninguna casa en la unión está en la posición de uno de los comercios de este simCity particular.
6. ninguna casa en la unión está sobre un río perteneciente a cualquier mapa en la unión.
7. el turno es mas grande que el nivel de cualquier casa en la unión.
8. ningún comercio en la unión está en la posición de una de las casas de este simCity particular.
9. ningún comercio en la unión está sobre un río perteneciente a cualquier mapa en la unión.
10. el turno es mas grande que el nivel base de cualquier comercio en la unión.
11. Cada hijo apunta a un SimCity y su puntero no aparece en ningún otro SimCity de la unión.
12. Cada hijo apunta a un Simcity válido.
13. El turno es mayor o igual a la cantidad de turnos que pasaron desde la unión.
14. Las uniones están ordenadas de más antiguas a más recientes.
15. No se solapan posiciones máximas entre esta estructura hasta el hijo 'x', descontando construcciones agregadas después de la unión, y ese hijo, para todo hijo.

auxiliares para la representación

$\text{unirPunteros} : \text{secu}(\text{hijo}) \rightarrow \text{conj}(\text{puntero}(\text{estr}))$

$\text{unirPunteros}(s) \equiv _ \text{unirPunteros}(s, \emptyset)$

$_ \text{unirPunteros} : \text{secu}(\text{hijo}) \times \text{conj}(\text{puntero}(\text{estr})) \rightarrow \text{conj}(\text{puntero}(\text{estr}))$

$_ \text{unirPunteros}(s, p) \equiv \text{if } \text{vacía?}(s) \text{ then}$
 $\quad p$
 else if $\text{prim}(s).sc \in p$ **then** // por si hay loops
 $_ \text{unirPunteros}(\text{fin}(s), p)$
 else
 $_ \text{unirPunteros}((*(\text{prim}(s).sc)).\text{uniones}, \text{Ag}(\text{prim}(s).sc, p)) \cup$
 $_ \text{unirPunteros}(\text{fin}(s), \text{Ag}(\text{prim}(s).sc, p))$
 fi

$\text{unirMapas} : \text{conj}(\text{puntero}(\text{estr})) \rightarrow \text{Mapa}$

$\text{unirMapas}(ps) \equiv \text{if } \text{vacío?}(ps) \text{ then}$
 $\text{crear}(\emptyset, \emptyset)$
 else
 $(*(\text{dameUno}(ps))).\text{mapa} + \text{UnirMapas}(\text{sinUno}(ps))$
 fi

$\text{unirCasas} : \text{conj}(\text{puntero}(\text{estr})) \rightarrow \text{dicc}(\text{Pos}, \text{Nivel})$

$\text{unirCasas}(ps) \equiv \text{if } \text{vacío?}(ps) \text{ then}$
 vacío
 else
 $(*p).\text{casas} \cup_{\text{dicc}} \text{unirCasas}(\text{sinUno}(ps))$
 fi

$\text{unirComercios} : \text{conj}(\text{puntero}(\text{estr})) \rightarrow \text{dicc}(\text{Pos}, \text{Nat})$

$\text{unirComercios}(ps) \equiv \text{if } \text{vacío?}(ps) \text{ then}$
 vacío
 else
 $(*p).\text{comercios} \cup_{\text{dicc}} \text{unirComercios}(\text{sinUno}(ps))$
 fi

$\text{remove} : \text{secu}(\alpha) \times \alpha \rightarrow \text{secu}(\alpha)$ // remueve la primer aparición

$\text{remove}(s, a) \equiv \text{if } \text{vacía?}(s) \text{ then}$
 $\langle \rangle$
 else if $a = \text{prim}(s)$ **then**
 $\text{fin}(s)$
 else
 $\text{prim}(s) \bullet \text{remove}(\text{fin}(s), a)$
 fi

$\text{unionesValidas} : \text{estr} \times \text{secu}(\text{hijo}) \rightarrow \text{bool}$

$\text{unionesValidas}(e, s) \equiv \text{vacío?}(s) \vee_{\text{L}} (\text{maxcons}(e, \text{izq}) \cap \text{maxcons}(e, \text{der}) = \emptyset \wedge \text{unionesValidas}(e, \text{com}(s)))$

donde

$\text{com} \equiv \text{unirPunteros}(\text{com}(s))$

$\text{ult} \equiv \text{ult}(s) \bullet \langle \rangle$

$\text{casascom} \equiv \text{unirCasas}(\text{com}) \cup_{\text{dicc}} \text{filtrar}(e.\text{casas}, \text{ult}(s).\text{turnosDesdeUnion})^1$

$\text{comercom} \equiv \text{unirComercios}(\text{com}) \cup_{\text{dicc}} \text{filtrar}(e.\text{comercios}, \text{ult}(s).\text{turnosDesdeUnion})^1$

$\text{casasult} \equiv \text{unirCasas}(\text{ult})$

$\text{comerult} \equiv \text{unirComercios}(\text{ult})$

$\text{izq} \equiv \text{claves}(\text{casascom}) \cup \text{claves}(\text{comercom})$

$\text{der} \equiv \text{claves}(\text{casasult}) \cup \text{claves}(\text{comerult})$

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

$\text{filtrar} : \text{dicc}(\text{Pos} \times \text{Nat}) \times \text{Nat} \longrightarrow \text{dicc}(\text{Pos}, \text{Nat})$

$\text{filtrar}(d, n) \equiv \text{if } \text{vacio?}(d) \text{ then}$
 vacio
 else if $\text{sig} \leq n$ **then**
 $\text{filtrar}(\text{borrar}(\text{clave}, d), n)$
 else
 $\text{definir}(\text{clave}, \text{sig}, \text{filtrar}(\text{borrar}(\text{clave}, d), n))$
 fi

donde

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

$\text{maxcons} : \text{estr} \times \text{conj}(\text{Pos}) \longrightarrow \text{conj}(\text{Pos})$

$\text{maxcons}(e, c) \equiv _ \text{maxcons}(e, c, \emptyset, 0)$

$_ \text{maxcons} : \text{estr} \times \text{conj}(\text{Pos}) \times \text{conj}(\text{Pos}) \times \text{Nat} \longrightarrow \text{conj}(\text{Pos})$

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

donde

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

$\text{nivel} : \text{estr} \times \text{pos} \longrightarrow \text{Nat}$

$\text{nivel}(e, \text{pos}) \equiv \text{if } \text{def?}(\text{pos}, \text{Casas}) \text{ then}$
 $\text{obtener}(\text{pos}, \text{Casas}) + \text{nivelesPorUnion}(e, \text{pos})$
 else
 $\text{maximo}(\text{obtener}(\text{pos}, \text{Comercios}) \bullet n\text{Manhattan}) + \text{nivelesPorUnion}(e, \text{pos})$
 fi

donde:

$\text{Unidos} \equiv \text{unirPunteros}(e.\text{uniones})$
 $\text{Casas} \equiv \text{unirCasas}(\text{Ag}(\&e, \text{Unidos}))$
 $\text{Comercios} \equiv \text{unirComercios}(\text{Ag}(\&e, \text{Unidos}))$
 $n\text{Manhattan} \equiv \text{significados}(\text{manhattan}(\text{pos}, 3), \text{Casas})$

$\text{nivelesPorUnion} : \text{estr} \times \text{pos} \longrightarrow \text{Nat}$

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

donde:

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

$\text{llegar} : \text{secu}(\text{hijo}) \times \text{pos} \longrightarrow \text{hijo}$

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

donde

hijo $\equiv Ag(prim(s).sc, \emptyset)$

manhattan : Pos \times Nat \longrightarrow Conj(Pos)

```
manhattan(p, dist)  $\equiv$  if  $dist = 0$  then
    Ag(p,  $\emptyset$ )
else
    diagonal( $\{p.x, p.y + dist\}, \{p.x + dist, p.y\}) \cup$ 
    if  $p.x - dist \geq 0$  then
        diagonal( $\{p.x, p.y + dist\}, \{p.xdist, p.y\})$ 
    else
         $\emptyset$ 
    fi  $\cup$ 
    if  $p.y - dist \geq 0$  then
        diagonal( $\{p.x, p.ydist\}, \{p.x + dist, p.y\})$ 
    else
         $\emptyset$ 
    fi  $\cup$ 
    if  $p.x - dist \geq 0 \wedge p.y - dist \geq 0$  then
        diagonal( $\{p.x, p.ydist\}, \{p.xdist, p.y\})$ 
    else
         $\emptyset$ 
    fi  $\cup$ 
    manhattan(p, dist - 1)
fi
```

diagonal : pos \times pos \times Nat n \longrightarrow Conj(pos)

```
diagonal(d, h, y)  $\equiv$  if  $y = |h.y - d.y|$  then
    Ag(h,  $\emptyset$ )
else
    Ag(caminar(d, h, y), diagonal(d, h, y + 1))
fi
```

caminar : pos \times pos \times Nat \longrightarrow pos

```
caminar(d, h, y)  $\equiv$  if  $d.y \leq h.y$  then
    if  $d.x \leq h.x$  then
         $\{d.x + y, d.y + y\}$ 
    else
         $\{d.x - y, d.y + y\}$ 
    fi
else
    if  $d.x \leq h.x$  then
         $\{d.x + y, d.y - y\}$ 
    else
         $\{d.x - y, d.y - y\}$ 
    fi
fi
```

significados : conj(α) \times dicc($\alpha \times \beta$) \longrightarrow secu(α)

```
significados(c, d)  $\equiv$  if vacio?(c) then
     $\langle \rangle$ 
else if def?(dameUno(c), d) then
    obtener(c, d)  $\bullet$  significados(sinUno(c), d)
else
    significados(sinUno(c), d)
fi
```

maximo : secu(Nat) a \longrightarrow Nat

$\{long(a) > 0\}$

```
maximo(s)  $\equiv$  if long(s) = 1 then prim(s) else max(prim(s), maximo(fin(s))) fi
```

nivelar : estr \times dicc(Pos \times Nat) \longrightarrow dicc(Pos, Nat)

```
nivelar(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))
```

2.2.3. Implementación

Algoritmos

===== **Generadores** =====**iniciar**(in m: Mapa) → res : estr

```

    estr.turno ← 0
    estr.popularidad ← 0
    estr.mapa ← m
    estr.casas ← vacio()
    estr.comercios ← vacio()
    estr.uniones ← vacia()
    return estr

```

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

```

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

        itDicc(Pos, Nivel) itCasas ← crearIt(s.casas);
        while(haySiguiente(itCasas)) :
            siguienteSignificado(itCasas) ← siguienteSignificado(itCasas) + 1
            avanzar(itCasas)

```

```

        itDicc(Pos, Nivel) itComercios ← crearIt(s.comercios);
        while(haySiguiente(itComercios)) :
            siguienteSignificado(itComercios) ← siguienteSignificado(itComercios) + 1
            avanzar(itComercios)

```

```

        itDicc(Pos, Nivel) itCs ← crearIt(cs);
        while(haySiguiente(itCs)) :
            if(siguienteSignificado(itCs) =obs "casa") :
                agCasa(s.casas, siguienteClave(itCs), 1)
            else if(siguienteSignificado(itCs) =obs "comercio") :
                agComercio(s.comercio, siguienteClave(itCs), 1)
            avanzar(itCs)

```

```

    estr.turno ← estr.turno + 1
Complejidad: O(long(s.uniones) + #claves(s.casas) +
#claves(s.comercios) + #claves(cs))

```

unir(inout SimCity s1, inout Simcity s2)

```

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

```

Complejidad: O(1)===== **Observadores** =====**mapa**(in SimCity s) → res : Mapa

```

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

```

Complejidad: $O()$

```

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

```

```

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

```

```

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

```

=====

===== **Otras Operaciones** =====

```

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

```

```

agCasa(inout dicc(Pos, Nivel) casas, in Pos p, in Nivel n) :
  definirRapido(casas, p, n)
Complejidad:  $O(1)$ 

```

```

agComercio(inout dicc(Pos, Nivel) comercios, in Pos p, in Nivel n) :
  definirRapido(comercio, p, n)
Complejidad:  $O(1)$ 

```

```

turnos(in SimCity s) → res : Nat

```

```

return s.turno
Complejidad:  $O(1)$ 

```

```

•  $\cup$  •(in  $\text{dicc}(\alpha, \beta)$  d1, in  $\text{dicc}(\alpha, \beta)$  d2)  $\rightarrow$  res :  $\text{dicc}(\alpha, \beta)$ 
   $\text{dicc}(\alpha, \beta)$  res = copiar(d1)
  itDicc( $\alpha, \beta$ ) itCs  $\leftarrow$  crearIt(d2);
  while(haySiguiente(itCs)) :
     $\alpha$  a  $\leftarrow$  siguienteClave(itCs)
     $\beta$  b  $\leftarrow$  siguienteSignificado(itCs)
    if( $\neg \text{def?}(\text{res}, a)$ ) :
      definir(res, a, b)
    avanzar(itCs)
  return res

```

```

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

```

```

=====

```

2.3. Módulo Servidor

2.3.1. Interfaz

Interfaz

se explica con: SERVIDOR

géneros: server

Operaciones básicas de server

NUEVOSEVER() $\rightarrow res : server$

Pre $\equiv \{true\}$

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

Complejidad: $O(1)$

Descripción: Crea un servidor

Aliasing: No tiene

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

Pre $\equiv \{true\}$

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

Complejidad: $O(copy(server))$

Descripción: Devuelve un diccionario con todas las partidas del servidor

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

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

Pre $\equiv \{true\}$

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

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

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

Aliasing: Devuelve una copia

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

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

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

Complejidad: $O(|p|)$

Descripción: Agrega una partida nueva al servidor

Aliasing: No tiene

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

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

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

Complejidad: $O()$

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

Aliasing: No tiene

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

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

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

Complejidad: $O()$

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

Aliasing: No tiene

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

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

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

Complejidad: $O()$

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

Aliasing: No tiene

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

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

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

Complejidad: $O()$

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

Aliasing: No tiene

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

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

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

Complejidad: $O()$

Descripción: Devuelve la popularidad de la partida

Aliasing: Devuelve una referencia no modificable

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

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

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

Complejidad: $O()$

Descripción: Devuelve la antigüedad de la partida

Aliasing: Devuelve una referencia no modificable

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

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

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

Complejidad: $O()$

Descripción: Devuelve el mapa de la partida

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

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

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

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

Complejidad: $O()$

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

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

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

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

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

Complejidad: $O()$

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

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

*donde:

$\text{unionValida} : \text{server } s \times \text{Nombre } p1 \times \text{Nombre } p2 \longrightarrow \text{boolean}$

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

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

$\text{avanzarTurnoValido} : \text{server } s \times \text{Nombre } p \times \text{dicc}(\text{Pos} \times \text{Construccion}) \text{ cs} \longrightarrow \text{boolean}$

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

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

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

$\bullet \cup_{\text{dicc}} \bullet : \text{dicc}(\alpha \times \beta) \times \text{dicc}(\alpha \times \beta) \longrightarrow \text{dicc}(\alpha, \beta)$

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

2.3.2. Representación

Representación

Representación de servidor

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

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

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

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

donde `Nombre` es string

donde `Construcccion` es string

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

$\text{Rep}(e) \equiv \text{true} \iff$
 $(\forall \text{partida}_1, \text{partida}_2 : \text{Nombre})$
 $((\text{def?}(\text{partida}_1, e) \wedge \text{def?}(\text{partida}_2, e) \wedge \text{partida}_1 \neq \text{partida}_2) \Rightarrow_L$
 $\quad \text{obtener}(\text{partida}_1, e).\text{sim} \neq \text{obtener}(\text{partida}_2, e).\text{sim}) \wedge$
 $(\forall \text{partida} : \text{Nombre})(\text{def?}(\text{partida}, e) \Rightarrow_L$
 $\quad p.\text{sim} \neq \text{NULL} \wedge$
 $\quad (p.\text{modificable} =_{\text{obs}} \text{false} \Rightarrow_L \text{vacio?}(\text{claves}(p.\text{pendientes}))) \wedge$
 $\quad (\forall \text{pos} : \text{Pos})(\text{def?}(\text{pos}, p.\text{pendientes}) \Rightarrow_L$
 $\quad \quad (\text{obtener}(\text{pos}, p.\text{pendientes}) \in \{\text{"casa"}, \text{"comercio"}\} \wedge$
 $\quad \quad \neg \text{def?}(\text{pos}, \text{construcciones}(*p.\text{sim})))$
 $\quad)$
 $)$

donde p es $\text{obtener}(\text{partida}, e)$

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

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

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

2.3.3. Implementación**Algoritmos**

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

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

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

 1: $\text{dicc}(\text{Nombre}, \text{SimCity}) \text{ } res \leftarrow \text{vacío}()$

 2: $\text{itDicc}(\text{Nombre}, \text{Partida}) \text{ } it \leftarrow \text{crearIt}(e)$

 3: **while**($\text{haySiguiente}(it)$)

 4: $\text{definirRapido}(res, \text{siguienteClave}(it), \text{siguienteSignificado}(it).sim)$

 5: $\text{avanzar}(it)$ **return** res
Complejidad: $O(\text{copy}(estr))$

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

 1: $\text{conj}(\text{Nombre}) \text{ } res \leftarrow \text{vacío}()$

 2: $\text{itDicc}(\text{Nombre}, \text{Partida}) \text{ } it \leftarrow \text{crearIt}(e)$

 3: **while**($\text{haySiguiente}(it)$)

 4: **if**($\text{siguienteSignificado}(it).modificable == \text{false}$)

 5: $\text{agregarRapido}(res, \text{siguienteClave}(it))$

 6: $\text{avanzar}(it)$ **return** res
Complejidad: $O(\#Partidas + |\text{NombreMasLargo}|)$

nuevaPartida(in/out e: estr, in p: Nombre, in m: Mapa)

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

Complejidad: $O(|p|)$

unirPartidas(in/out e: estr, in p1: Nombre, in p2: Nombre)

 1: $\text{definir}(e, p1, \langle \text{true}, \&(\text{unir}(*\text{significado}(p1, e).sim, *\text{significado}(p2, e).sim)), \text{vacío}() \rangle)$

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

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

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

partidas(in e : estr) $\rightarrow res$: diccTrie(Nombre, SimCity)

```

1: dicc(Nombre, SimCity)  $res \leftarrow vacio()$ 
2: itDicc(Nombre, Partida)  $it \leftarrow crearIt(e)$ 
3: while(haySiguiente(it))
4:   definirRapido( $res$ , siguienteClave(it), siguienteSignificado(it).sim)
5:   avanzar(it) return  $res$ 
```

Complejidad: $O(copy(estr))$

congeladas(in e : estr) $\rightarrow res$: conj(Nombre)

```

1: conj(Nombre)  $res \leftarrow vacio()$ 
2: itDicc(Nombre, Partida)  $it \leftarrow crearIt(e)$ 
3: while(haySiguiente(it))
4:   if(siguienteSignificado(it).modificable == false)
5:     agregarRapido( $res$ , siguienteClave(it))
6:   avanzar(it) return  $res$ 
```

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

nuevaPartida(in/out e : estr, in p : Nombre, in m : Mapa)

```

1: definirRapido( $e$ ,  $p$ ,  $\langle true, \&(iniciar(m)), vacio() \rangle$ )  $\triangleright$  Reservamos memoria para el nuevo SimCity
```

Complejidad: $O(|p|)$

unirPartidas(in/out e : estr, in $p1$: Nombre, in $p2$: Nombre)

```

1: definir( $e$ ,  $p1$ ,  $\langle true, \&(unir(*significado(p1, e).sim, *significado(p2, e).sim)), vacio() \rangle$ )
2: definir( $e$ ,  $p2$ ,  $\langle false, significado(p2, e).sim, vacio() \rangle$ )
```

Complejidad: $O(|Nombre|)$

Justificacion: $unir \in O(1)$, $definir \in O(|Nombre|) + O(1) = O(|Nombre|)$

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

```

1: definirRapido( $Pos$ , "casa", obtener(partida,  $s$ ))
```

Complejidad: $O(|partida|)$

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

```

1: definirRapido( $Pos$ , "comercio", obtener(partida,  $s$ ))
```

Complejidad: $O(|partida|)$

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

```

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

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

verCasas(*in s*: *estr*, *in partida*: *String*) → *res*: *DiccLineal*(*Pos*, *Nivel*)

1: *sc* ← *obtener*(*partida*, *s*).*sim*
2: *res* ← *casas*(**sc*)
3: **return** *res*

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

verComercios(*in s*: *estr*, *in partida*: *String*) → *res*: *DiccLineal*(*Pos*, *Nivel*)

1: *sc* ← *obtener*(*partida*, *s*).*sim*
2: *res* ← *comercios*(**sc*)
3: **return** *res*

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

verPopularidad(*in s*: *estr*, *in partida*: *String*) → *res*: *Nat*

1: *sc* ← *obtener*(*partida*, *s*).*sim*
2: *res* ← *popularidad*(**sc*)
3: **return** *res*

Complejidad: $O(|partida|)$

verTurno(*in s*: *estr*, *in partida*: *String*) → *res*: *Nat*

1: *sc* ← *obtener*(*partida*, *s*).*sim*
2: *res* ← *turnos*(**sc*)
3: **return** *res*

Complejidad: $O(|partida|)$
