



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	8
2.1.3. Implementación	9
2.2. Módulo SimCity	10
2.2.1. Interfaz	10
2.2.2. Representación	12
2.2.3. Implementación	16
2.3. Módulo Servidor	19
2.3.1. Interfaz	19
2.3.2. Representación	21
2.3.3. Implementación	22

1. Especificación

1.1. Aliases

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

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) \times conj(Nat) \longrightarrow Mapa

otras operaciones

$\bullet + \bullet$: Mapa \times Mapa \longrightarrow Mapa

esRio : Pos \times Mapa \longrightarrow Bool

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

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

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

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

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

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

Fin TAD


```

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

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

construcciones( $s$ )            $\equiv$  casas( $s$ )  $\cup_{dicc}$  comercios( $s$ )
agCasas( $cn$ ,  $cs$ )            $\equiv$  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  $\equiv$  dameUno(claves( $cs$ ))
agComercios( $s$ ,  $cn$ ,  $cs$ )  $\equiv$  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  $\equiv$  dameUno(claves( $cs$ ))
nivelComercio( $p$ ,  $cn$ )        $\equiv$  if vacio?(claves( $cn$ )) then
                               1
                               else if distManhattan( $p$ , proximo)  $\leq$  3 then
                                   max(obtener(proximo,  $cn$ ), nivelComercio( $p$ , borrar(proximo,  $cn$ )))
                               else
                                   nivelComercio( $p$ , borrar(proximo,  $cn$ ))
                               fi
                               donde proximo  $\equiv$  dameUno(claves( $cn$ ))
distManhattan( $p$ ,  $q$ )          $\equiv$  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
 $d \cup_{dicc} d'$               $\equiv$  if vacio?(claves( $d'$ )) then
                                $d$ 
                               else
                                   definir(proximo, obtener(proximo,  $d'$ ),  $d \cup_{dicc}$  borrar(proximo,  $d'$ ))
                               fi
                               donde proximo  $\equiv$  dameUno(claves( $d'$ ))
sacarRepes( $cs$ ,  $cs'$ )         $\equiv$  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  $\equiv$  dameUno(claves( $cs$ ))

```

Fin TAD

Definiciones Auxiliares de SimCity

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

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

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

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

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( $s, p, m$ ),  $p'$ )       $\equiv$  if  $p = p'$  then vacio else pendientes( $s, p'$ ) fi
pendientes(unirPartidas( $s, p, p'$ ),  $p''$ )       $\equiv$  pendientes( $s, p''$ )
pendientes(agregarCasa( $s, p, pos$ ),  $p'$ )       $\equiv$  if  $p = p'$  then
                                                definir( $pos, "casa"$ , pendientes( $s, p$ ))
                                                else
                                                    pendientes( $s, p'$ )
                                                fi
pendientes(agregarComercio( $s, p, pos$ ),  $p'$ )   $\equiv$  if  $p = p'$  then
                                                definir( $pos, "comercio"$ , pendientes( $s, p$ ))
                                                else
                                                    pendientes( $s, p'$ )
                                                fi
pendientes(avanzarTurnoPartida( $s, p$ ),  $p'$ )   $\equiv$  if  $p = p'$  then vacio else pendientes( $s, p'$ ) fi

```

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

```

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( $s$ )))

```

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( $s$ )  $\wedge_L$ 
    vacio?(claves(pendientes( $s, p$ )))  $\wedge$  vacio?(claves(pendientes( $s, p'$ )))  $\wedge$ 
    unirValido(obtener( $p$ , partidas( $s$ )), obtener( $p'$ , partidas( $s$ )))1

avanzarValido : Server  $s \times$  Nombre  $p \longrightarrow$  boolean
avanzarValido( $s, p$ )  $\equiv$  def?( $p$ , partidas( $s$ ))  $\wedge p \notin$  congeladas( $s$ )  $\wedge_L$ 
     $\neg$  vacio?(claves(pendientes(obtener( $p$ , partidas( $s$ ))))))

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?( $pos$ , pendientes( $s, p$ ))  $\wedge$ 
     $\neg$  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. Módulo Mapa

2.1.1. Interfaz

Interfaz

usa: Conjunto Lineal, Nat, Bool, Pos

exporta: todo

se explica con: MAPA

géneros: Mapa

Operaciones básicas de Mapa

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

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

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

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

Descripción: Crea un mapa.

ESRIO(**in** m : Mapa, **in** p : Pos) $\rightarrow res$: 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** $m1$: Mapa, **in** $m2$: Mapa) $\rightarrow res$: Mapa

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

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

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

Descripción: Une dos Mapas.

2.1.2. Representación

Representación

Representación de mapa

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

Mapa **se representa con** estr

donde estr es tupla(horizontales : conj (Nat),
 verticales : conj (Nat))

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

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

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

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

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

2.1.3. Implementación**Algoritmos**

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

```
crear(in hs : conj(Nat), in vs : conj(Nat))  $\longrightarrow$  res : estr
1:   estr.horizontales  $\leftarrow$  hs
2:   estr.verticales  $\leftarrow$  vs
3:   return estr
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:
3:   itConj(Nat) itH  $\leftarrow$  crearIt(horizontales(m2))
4:   while(haySiguiente(itH)) :
5:     Ag(horizontales(res), siguiente(itH))
6:     avanzar(itH)
7:
8:   itConj(Nat) itV  $\leftarrow$  crearIt(vertaicales(m2))
9:   while(haySiguiente(itV)) :
10:    Ag(vertaicales(res), siguiente(itV))
11:    avanzar(itV)
12:
13:  return res
Complejidad:  $O(\text{copiar}(\text{m1}) + \text{copiar}(\text{m2}))$ 
```

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

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 : SimCity, $N = \text{popularidad}(S)$, $\{u_0 \dots u_N\} = U$: el conjunto de SimCities en union con S^1 y S , $\text{casas} = \sum_{i=0}^N (\#(\text{casas}_i))$ y $\text{comercios} = \sum_{i=0}^N (\#(\text{comercios}_i))$, donde casas_i y comercios_i son respectivamente 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: $O(\text{casas}^2 + N)$

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. Resultando en un peor caso, holgado, de $O(\text{casas}^2)$. Dado que un SimCity puede no tener casas, se debe considerar que el peor caso puede estar dado por la cantidad de nodos a recorrer en el árbol.

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

Aliasing: No. Genera una copia.

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

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

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

Complejidad: $O(\text{comercios}^2 \times \text{casas} + N)$

justificación: similar a $\text{casas}(s)$. En este caso también se debe calcular el nivel de cada comercio en relación a las casas en distancia manhattan ≤ 3 .

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 $\text{casas} \geq \#(\text{claves}(\text{casas}(\hat{S})))$ y $\text{comercios} \geq \#(\text{claves}(\text{comercios}(\hat{S})))$, 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: $O(N + \#(\text{claves}(\text{casas}_S)) + \#(\text{claves}(\text{comercios}_S)) + \#(\text{claves}(cs)))$,

donde casas_S y comercios_S son, respectivamente, el conjunto de casas y el conjunto de comercios definidos en éste SimCity particular a través de *avanzarTurno*.

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.

2.2.2. Representación

Representación

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

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

SimCity se representa con *estr*

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

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

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

Rep : *estr*¹ \rightarrow boolean

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

donde

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

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

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

Definiciones Auxiliares

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

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

 $_ \text{unirPunteros} : \text{secu}(\text{hijo}) \ s \times \text{conj}(\text{puntero}(\text{estr})) \ ps \longrightarrow \text{conj}(\text{puntero}(\text{estr})) \quad \{\text{eq. unirPunteros}\}$
 $_ \text{unirPunteros}(s, ps) \equiv \text{if vacia?}(s) \text{ then}$
 $\quad \quad \quad ps$
 $\quad \text{else if prim}(s).sc \in ps \text{ then} \quad // \text{ por si hay loops}$
 $\quad \quad _ \text{unirPunteros}(\text{fin}(s), ps)$
 $\quad \text{else}$
 $\quad \quad _ \text{unirPunteros}((*(\text{prim}(s).sc)).\text{uniones}, \text{Ag}(\text{prim}(s).sc, ps)) \cup$
 $\quad \quad _ \text{unirPunteros}(\text{fin}(s), \text{Ag}(\text{prim}(s).sc, ps))$
 $\quad \quad // \text{ al unir se descarta el duplicado}$
 $\quad \text{fi}$

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

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

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

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

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

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

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

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

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

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

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

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

donde

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

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

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

donde:

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

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

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

donde:

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

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

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

donde

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

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

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

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

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

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

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

donde

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

2.2.3. Implementación**Algoritmos**

```

iniciar(in m: Mapa)  $\rightarrow$  res : estr
  estr.turno  $\leftarrow$  0
  estr.popularidad  $\leftarrow$  0
  estr.mapa  $\leftarrow$  m
  estr.casas  $\leftarrow$  vacio()
  estr.comercios  $\leftarrow$  vacio()
  estr.uniones  $\leftarrow$  vacia()
  return estr

```

Complejidad: $O(1)$

```

avanzarTurno(inout SimCity s, in dicc(Pos, Construcccion) cs)
  for(nat i  $\leftarrow$  0; i < long(s.uniones); i  $\leftarrow$  i + 1) :
    turnoDesdeUnion  $\leftarrow$  turnoDesdeUnion + 1;

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

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

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

```

Complejidad: $O(\text{long}(s.uniones) + \#claves(s.casas) + \#claves(s.comercios) + \#claves(cs))$

```

unir(inout SimCity s1, inout Simcity s2)
  s1.popularidad  $\leftarrow$  s1.popularidad + s2.popularidad + 1
  turno  $\leftarrow$  max(s1.turno, s2.turno)
  hijo nuevoHijo  $\leftarrow$  <direccion(s2), 0>
  agregarAtras(s1.uniones, nuevoHijo)

```

Complejidad: $O(1)$

```

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

```

Complejidad: $O()$

```

casas(in SimCity s)  $\rightarrow$  res : dicc(Pos, Nivel)
  dicc res  $\leftarrow$  copiar(s.casas)
  for(nat i  $\leftarrow$  0; i < long(s.uniones); i  $\leftarrow$  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) ∧ ¬def?(s.casas, p)) :
        definir(res, p, n + s.uniones[i].turnosDesdeUnion)
    avanzar(itCs)
return res

```

```

comercios(in SimCity s) → res : dicc(Pos, Nivel)
    return comerciosAux(s, casas(s))

```

```

comerciosAux(in SimCity s, in casasTotales dicc(Pos, Nivel)) → res : dicc(Pos, Nivel)
    dicc res ← copiar(s.comercios)
    itDicc(Pos, Nivel) itCs ← crearIt(comerciosAux(s.uniones[i].sc*, casastotales));
    while(haySiguiente(itCs)) :
        Pos p ← siguienteClave(itCs)
        Nivel n ← siguienteSignificado(itCs)
        if(¬def?(res, p) ∧ ¬def?(s.casas, p)) :
            Nivel m ← max(n + s.uniones[i].turnosDesdeUnion, nivelCom(p, casasTotales(s)))
            definir(res, p, m)
        avanzar(itCs)

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

```

```

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

```

```

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) \rightarrow 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(α, β) itCs \leftarrow crearIt(d2);

while(haySiguiente(itCs)) :

α a \leftarrow siguienteClave(itCs)

β b \leftarrow siguienteSignificado(itCs)

if($\neg \text{def?}(\text{res}, \text{a})$) :

 definir(res, a, b)

 avanzar(itCs)

return res

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

construcc(in SimCity s) \rightarrow res : Nat

return casas(s) \cup comercios(s)

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

2.3. Módulo Servidor

2.3.1. 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) $\rightarrow res : conjLineal(Nombre)$

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)

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

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

Complejidad: $O(|nombreMasLargo|) + O(avanzarTurnoPartida(sc, construcciones))$,

donde sc es el SimCity asociado a p y $construcciones$ es el diccionario de pendientes asociados a p .

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

Aliasing: No tiene.

AGREGARCasa(**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 al diccionario de pendientes de la partida.

Aliasing: No tiene.

AGREGARCOMERCIO(**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 al diccionario de pendientes de la partida.

Aliasing: No tiene.

VERPOPULARIDAD(**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.

VERTURNO(**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.

VERMAPA(**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 asociada a p

Descripción: Devuelve el mapa de la partida.

Aliasing: Devuelve una copia.

VERCASAS(**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 asociada a p

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

Aliasing: Devuelve una copia.

VERCOMERCIOS(**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 asociada 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.

2.3.2. 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).

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 $O(|nombreMasLargo|)$.

servidor se representa con diccTrie(Nombre, partida)

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

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

Rep : estr \longrightarrow boolean

Rep(e) \equiv true \iff
 $(\forall partida_1, partida_2 : \text{Nombre})$
 $((\text{def?}(partida_1, e) \wedge \text{def?}(partida_2, e) \wedge partida_1 \neq partida_2) \Rightarrow_L$
 $\text{obtener}(partida_1, e).sim \neq \text{obtener}(partida_2, e).sim$
 $) \wedge$
 $(\forall partida : \text{Nombre})(\text{def?}(partida, e) \Rightarrow_L$
 $p.sim \neq \text{NULL} \wedge$
 $(\neg p.modificable \Rightarrow_L \text{vacio?}(\text{claves}(p.pendientes))) \wedge$
 $(\forall pos : \text{Pos})(\text{def?}(pos, p.pendientes) \Rightarrow_L$
 $(\text{obtener}(pos, p.pendientes) \in \{"casa", "comercio"\} \wedge$
 $\neg \text{def?}(pos, \text{construcciones}(*p.sim)) \wedge \neg \text{esRio}(pos, \text{mapa}(*p.sim)))$
 $)$
 $)$

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

Abs : estr $e \longrightarrow$ servidor

{Rep(e)}

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

2.3.3. Implementación**Algoritmos**

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

```
1:  $res \leftarrow \text{vacío}()$ 
2: return  $res$ 
```

Complejidad: $O(1)$

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

```
1:  $res \leftarrow \text{vacío}()$ 
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: $O(\sum_{i=0}^N \text{copy}(nom_i) + \text{copy}(sc_i))$

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

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

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

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

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

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

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

Justificación: $\text{unir} \in O(1)$, $\text{definir} \in O(|\text{nombreMasLargo}|) + O(1) = O(|\text{nombreMasLargo}|)$

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

```
1:  $\text{definir}(e, p, \langle \text{true}, \text{avanzarTurno}(*(\text{significado}(p, e).sim), \text{significado}(p, e).pendientes), \text{vacío}() \rangle)$ 
```

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

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

1: *definirRapido*(*obtener*($partida$, s).*pendientes*, pos , "casa")

Complejidad: $O(|nombreMasLargo|)$

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

1: *definirRapido*(*obtener*($partida$, s).*pendientes*, pos , "comercio")

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: $O(|nombreMasLargo|)$
