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.arienti@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

Resúmen

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

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

$$(\forall m, m' : \text{Mapa}) \left(m =_{\text{obs}} m' \iff \begin{pmatrix} \text{horizontales}(m) =_{\text{obs}} \text{horizontales}(m') \wedge_{\text{L}} \\ \text{verticales}(m) =_{\text{obs}} \text{verticales}(m') \end{pmatrix} \right)$$

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 $\forall hs,\ vs$: conj(Nat)

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

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

$m1 + m2$ $\equiv$ crear(horizontales($m1$) $\cup$ horizontales($m2$), verticales($m1$) $\cup$ verticales($m2$))
esRio($m1$, p) $\equiv$ $p.x \in$ verticales($m1$) $\vee$ $p.y \in$ horizontales($m1$)

Fin TAD

1.3. SimCity

TAD SimCity

géneros SimCity

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

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

igualdad observacional

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

observadores básicos

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

generadores

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

otras operaciones

turnos	: SimCity	$\longrightarrow$	Nat
nivelComercio	: Pos $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	Nat
distManhattan	: Pos $\times$ Pos	$\longrightarrow$	Nat
$\bullet \cup_{dicc} \bullet$	: dicc($\alpha \times \beta$) $\times$ dicc($\alpha \times \beta$)	$\longrightarrow$	dicc($\alpha \times \beta$)

construcciones	: SimCity	$\longrightarrow$	dicc(Pos $\times$ Nivel)
agCasas	: dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Construcción)	$\longrightarrow$	dicc(Pos $\times$ Nivel)
agComercios	: SimCity $\times$ dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Construcción)	$\longrightarrow$	dicc(Pos $\times$ Nivel)
sacarRepes	: dicc(Pos $\times$ Construcción) $\times$ dicc(Pos $\times$ Construcción)	$\longrightarrow$	dicc(Pos $\times$ Construcción)

axiomas $\forall s,\ s'$: simcity, $\forall m$: Mapa, $\forall cs$: dicc(Pos $\times$ 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)) $\equiv$ 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($\alpha \times \beta$)

nivelComercio(p, cn) $\equiv$ **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 $\equiv$ 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'))

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))

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 × dicc(Pos × Construcción) cs $\longrightarrow$ boolean
avanzarValido(s, cs) $\equiv$ $\neg$ vacio?(claves(cs)) $\wedge$
(∀ p : Pos) (def?(p, cs) $\Rightarrow_{\text{L}}$
$\neg$ $p \in$ claves(construcciones(s)) $\wedge$
$\neg$ esRio(p, mapa(s)) $\wedge$
(obtener(p, cs) = "*casa*" $\vee$ obtener(p, cs) = "*comercio*"))
)

unirValido : Simcity a × SimCity b $\longrightarrow$ boolean
unirValido(a, b) $\equiv$ (∀ p : Pos)(def?(p, construcciones(a)) $\Rightarrow_{\text{L}}$
$\neg$ esRio(p, mapa(b)) $\wedge$
($\nexists$ *otra* : Pos)(def?(*otra*, construcciones(a)) $\wedge_{\text{L}}$
obtener(*otra*, construcciones(a)) > obtener(p, construcciones(a))
) $\Rightarrow_{\text{L}}$ $\neg$ def?(p, construcciones(b))
) $\wedge$
(∀ p : Pos)(def?(p, construcciones(b)) $\Rightarrow_{\text{L}}$
$\neg$ esRio(p, mapa(a)) $\wedge$
($\nexists$ *otra* : Pos)(def?(*otra*, construcciones(b)) $\wedge_{\text{L}}$
obtener(*otra*, construcciones(b)) > obtener(p, construcciones(b))
) $\Rightarrow_{\text{L}}$ $\neg$ def?(p, construcciones(a))
)

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, dicc($\alpha \times \beta$), conj(α)

igualdad observacional

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

observadores básicos

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

generadores

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

otras operaciones

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

axiomas $\forall s$: Server, $\forall p,\ p',\ p''$: Nombre, $\forall m$: Mapa, $\forall pos$: 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

unionValida : Server s × Nombre p × Nombre p' ⟶ boolean
unionValida(s, p, p') ≡ def?(p, partidas(s)) ∧ def?(p', partidas(s)) ∧ $p \notin$ congeladas(s) $\wedge_{\text{L}}$
vacio?(claves(pendientes(s, p))) ∧ vacio?(claves(pendientes(s, p'))) ∧
unirValido(obtener(p, partidas(s)), obtener(p', partidas(s)))[1]

avanzarValido : Server s × Nombre p ⟶ boolean
avanzarValido(s, p) ≡ def?(p, partidas(s)) ∧ $p \notin$ congeladas(s) $\wedge_{\text{L}}$
¬ vacio?(claves(pendientes(obtener(p, partidas(s)))))

agregarValido : Server s × Nombre p × Pos pos ⟶ boolean
agregarValido(s, p, pos) ≡ def?(p, partidas(s)) ∧ $p \notin$ congeladas(s) ∧ ¬ def?(pos, pendientes(s, p)) ∧
¬ def?(pos, verCasas(s, p)) ∧ ¬ def?(pos, verComercios(s, p)) ∧
¬ 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: se requiere?

exporta: se requiere?

se explica con: MAPA

géneros: `Mapa`

Operaciones básicas de Mapa

no se si hace falta agregar para los obs?

CREAR(**in** hs: `conj(Nat)`, **in** vs: `conj(Nat)`) $\rightarrow$ res : `Mapa`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s =_{\text{obs}}$ mapa($\hat{hs}$, $\hat{vs}$)}
Complejidad: O(copy(hs) + copy(vs)) puede ser theta?
Descripción: Crea un mapa.

ESRIO(**in** m: `Mapa`, **in** p: `Pos`) $\rightarrow$ res : `Bool`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s =_{\text{obs}}$ esRio($\hat{m}$, $\hat{p}$)}
Complejidad: $O(1)$ como rep los conj(Nat)?, si es sobre un array creo que hay q implementarlo. Tmb podria ser theta creo
Descripción: Verifica si en determinada pos hay un río.

SUMA(**in** $m1$: `Mapa`, **in** $m2$: `Mapa`) $\rightarrow$ res : `Mapa`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s =_{\text{obs}}$ $\hat{m1}$ + $\hat{m2}$}
Complejidad: O(crear($m1$) + 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)`)

Rep : estr $\longrightarrow$ boolean
Rep(e) $\equiv$ true $\iff$ true

Abs : estr m $\longrightarrow$ Mapa {Rep(m)}
Abs(m) $\equiv$ horizontales(m) = $estr$.horizontales $\wedge$ verticales(m) = $estr$.verticales igobs?

2.1.3. Implementación

---

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(copy(hs) + copy(vs))

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

2: itConj(Nat) itH $\leftarrow$ crearIt(horizontales(m2))
3: **while**(haySiguiente(itH)) :
4: Ag(horizontales(res), siguiente(itH))
5: avanzar(itH)

6: itConj(Nat) itV $\leftarrow$ crearIt(verticales(m2))
7: **while**(haySiguiente(itV)) :
8: Ag(verticales(res), siguiente(itV))
9: avanzar(itV)
10: **return** res
Complejidad: O(copiar(m1) + copiar(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(#horizontales(m1) + #verticales(m1))

2.2. Módulo SimCity

2.2.1. Interfaz

Interfaz

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

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

exporta: todo

se explica con: SimCity

géneros: `SimCity`

Operaciones básicas de SimCity

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

MAPA(**in** S : `SimCity`) → res : `Mapa`
Pre ≡ {true}
Post ≡ {$r\hat{e}s$ =$_{\text{obs}}$ mapa($\hat{S}$)}
Complejidad: $\Theta(\sum_{i=0}^{N} \text{copy}(mapa_i))$, donde $mapa_i$ es el Mapa original[2] de u_i.
Descripción: Retorna el mapa sobre el que se desarrolla el juego actual.
Aliasing: No. Genera una copia.

CASAS(**in** S : `SimCity`) → res : `DiccLineal(Pos, Nivel)`
Pre ≡ {true}
Post ≡ {$r\hat{e}s$ =$_{\text{obs}}$ casas($\hat{S}$)}
Complejidad: $\mathbf{O}(casas^2) \supset \Theta(\sum_{i=0}^{N} (hasta_i \times casas_i + camino_i))$,
donde $hasta_i$ y $casas_i$ son respectivamente la cantidad de casas definidas[3] en $\{u_0 \ ... \ u_{i-1}\}$ y u_i[4],
y $camino_i$ representa la cantidad de uniones para llegar de S a u_i[5].[6]
Descripción: Retorna las posiciones y respectivos niveles de todas las casas en el juego actual.
Aliasing: No. Genera una copia.

COMERCIOS(**in** S : `SimCity`) → res : `DiccLineal(Pos, Nivel)`
Pre ≡ {true}
Post ≡ {$r\hat{e}s$ =$_{\text{obs}}$ comercios($\hat{S}$)}
Complejidad: $\mathbf{O}(comercios^2 \times casas) \supset \Theta(\sum_{i=0}^{N} (hasta_i \times comercios_i \times casas + camino_i))$,
donde $hasta_i$ y $comercios_i$ son respectivamente la cantidad de comercios definidos en $\{u_0 \ ... \ u_{i-1}\}$ 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 : `SimCity`) → res : `Nat`
Pre ≡ {true}
Post ≡ {$r\hat{e}s$ =$_{\text{obs}}$ popularidad($\hat{S}$)}
Complejidad: $\Theta(1)$
Descripción: Retorna la cantidad total de uniones que se realizaron para conformar la partida actual.

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

INICIAR(**in** m : `Mapa`) $\rightarrow$ res : `SimCity`
Pre $\equiv$ {true}
Post $\equiv$ {$r\hat{e}s$ $=_{\text{obs}}$ iniciar($\hat{m}$)}
Complejidad: $\Theta(1)$
Descripción: Crea un nuevo SimCity.
Aliasing: Se guarda una referencia a m en res. No se modifica.

AVANZARTURNO(**in/out** S : `SimCity`, **in** cs : `diccLineal(Pos, Construccion)`)
Pre $\equiv$ {avanzarValido[2]($\hat{s}$, $\hat{cs}$) $\wedge$ $\hat{S}$ $=$ S_0}
Post $\equiv$ {$\hat{S}$ $=_{\text{obs}}$ avanzarTurno(S_0, $\hat{cs}$)}
Complejidad: $\mathbf{O}(casas \times \#(\text{claves}(\hat{cs})) + N) \supset \Theta(casas_S \times \#(\text{claves}(\hat{cs})) + U_S)$,
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** $S1$: `SimCity`, **in** $S2$: `SimCity`)
Pre $\equiv$ {unionValida[2]($\hat{S1}$, $\hat{S2}$) $\wedge$ $\hat{S1}$ $=$ S_0}
Post $\equiv$ {$\hat{S1}$ $=_{\text{obs}}$ 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 #(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. 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`)

$$
\begin{aligned}
&\text{Rep} : \text{estr}^{1} \longrightarrow \text{boolean} \\
&\text{Rep}(e) \equiv \text{true} \iff (\\
&\quad (\forall\, h : \text{hijo})(\text{esta?}(h,\ e.uniones) \Rightarrow_{\text{L}} \\
&\qquad h.sc \neq \text{NULL} \wedge_{\text{L}} rep(*(h.sc))^{2} \wedge h.sc \notin \text{unirPunteros}(\text{remover}(p,\ e.uniones))^{3} \wedge_{\text{L}} \\
&\qquad (e.turno \geq h.turnosDesdeUnion)^{4} \wedge \\
&\qquad (\forall\, h_2 : \text{hijo})(\text{esta?}(h_2,\ e.uniones) \wedge_{\text{L}} \text{pos}(h_2,\ e.uniones) > \text{pos}(h,\ e.uniones) \Rightarrow_{\text{L}} \\
&\qquad\quad h_2.turnosDesdeUnion \leq h.turnosDesdeUnion \\
&\qquad)^{5} \\
&\quad) \wedge_{\text{L}} \\
&\quad (\&e \notin \text{Unidos})^{6} \wedge_{\text{L}} \\
&\quad (e.popularidad = \#(\text{Unidos}))^{7} \wedge \\
&\quad (\forall\, p : \text{puntero(estr)})(p \in \text{Unidos} \Rightarrow_{\text{L}} \\
&\qquad e.turno \geq (*p).turno \\
&\quad)^{8} \wedge \\
&\quad (\forall\, p : \text{Pos})(p \in \text{claves(e.casas)} \Rightarrow_{\text{L}} \\
&\qquad \neg\text{def}(p,\ e.comercios)^{9} \wedge \neg\text{esRio}(p, \text{Mapas})^{10} \wedge (\text{obtener}(p, \text{e.casas}) < e.turno)^{11} \\
&\quad) \wedge \\
&\quad (\forall\, p :\ pos)(p \in \text{claves(e.comercios)} \Rightarrow_{\text{L}} \\
&\qquad \neg\text{def}(p,\ e.casas)^{12} \wedge \neg\text{esRio}(p, \text{Mapas})^{13} \wedge (\text{obtener}(p, \text{e.comercios}) < e.turno)^{14} \\
&\quad) \wedge_{\text{L}} \\
&\quad \text{unionesValidas}(e,\ e.uniones)^{15} \\
&\quad) \\
&\text{donde} \\
&\quad \text{Unidos} \equiv \text{unirPunteros}(e.uniones) \\
&\quad \text{Mapas} \equiv e.mapa + \text{unirMapas(Unidos)}
\end{aligned}
$$

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

donde

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

Definiciones Auxiliares

Los siguientes reemplazos sintácticos están 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.

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

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

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

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

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

remover : secu(α) $s \times \alpha$ a $\longrightarrow$ secu(α) // remueve la primer aparición, si hay
remover(s, a) $\equiv$ **if** vacia?(s) **then**
<>
else if a = prim(s) **then**
fin(s)
else
prim(s) $\bullet$ remover(fin(s), a)
fi

unionesValidas : estr e $\times$ secu(hijo) s $\longrightarrow$ bool
$\{s \subseteq e.uniones \wedge_{\text{L}} (\forall h : \text{hijo})(h \in s \Rightarrow_{\text{L}} h.sc \neq \text{NULL} \wedge_{\text{L}} \text{rep}(*(h.sc)))\}$
unionesValidas(e, s) $\equiv$ vacio?(s) $\vee_{\text{L}}$ (maxcons(e, izq) $\cap$ maxcons(e, der) $=$ $\emptyset \wedge$ unionesValidas(e, com(s)))
donde
com $\equiv$ unirPunteros(com(s))
ult $\equiv$ ult(s) $\bullet$ <>
casascom $\equiv$ unirCasas(com) $\cup_{dicc}$ filtrar($e.casas$, ult(s).$turnosDesdeUnion$)[1]
comercom $\equiv$ unirComercios(com) $\cup_{dicc}$ filtrar($e.comercios$, ult(s).$turnosDesdeUnion$)[1]
casasult $\equiv$ unirCasas(ult)
comerult $\equiv$ unirComercios(ult)
izq $\equiv$ claves(casascom) $\cup$ claves(comercom)
der $\equiv$ claves(casasult) $\cup$ claves(comerult)

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

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

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

nivel : estr e × Pos pos $\longrightarrow$ Nat $\{p \in \text{posiciones}(e)\}$
nivel(e, pos) $\equiv$ **if** def?$(pos,$ Casas$)$ **then**
obtener$(pos,$ Casas$)$ + nivelesPorUnion$(e,\ pos)$
else
max(obtener$(pos,$ Comercios$)$ + nivelesPorUnion$(e,\ pos)$, nManhattan)
fi
donde:
- Unidos $\equiv$ unirPunteros$(e.uniones)$
- Casas $\equiv$ unirCasas(Ag(&e, Unidos))
- Comercios $\equiv$ unirComercios(Ag(&e, Unidos))
- nManhattan $\equiv$ manhattan$(e,\ pos,$ Casas$)$

nivelesPorUnion : estr e × Pos pos $\longrightarrow$ Nat {eq. nivel}
nivelesPorUnion(e, pos) $\equiv$ **if** def?$(pos,\ e.casas) \vee$ def?$(pos,\ e.comercios)$ **then**
0
else
hijoCorrecto.$turnosDesdeUnion$ + nivelesPorUnion(hijoCorrecto.$sc,\ pos$)
fi
donde:
- hijoCorrecto $\equiv$ llegar$(e.uniones,\ pos)$

llegar : secu(hijo) s × Pos p $\longrightarrow$ hijo $\{(\forall\ h : \text{hijo})(\text{esta?}(h,\ s) \Rightarrow_{\text{L}} h.sc \neq \text{NULL} \wedge_{\text{L}} \text{rep}(*(h.sc))) \wedge_{\text{L}}$
$(\exists h : \text{hijo})(\text{esta?}(h,\ s) \wedge_{\text{L}} p \in posiciones(h.sc))\}$
llegar(s, p) $\equiv$ **if** def?$(pos,$ unirCasas$(hijo_i))$ $\vee$ def?$(pos,$ unirComercios$(hijo_i))$ **then**
prim(s)
else
llegar(fin(s))
fi
donde
- $\text{hijo}_i \equiv$ Ag(prim$(s).sc, \emptyset$)

manhattan : estr e × Pos p × Dicc(Pos × Nat) d $\longrightarrow$ Conj(Pos)
$\{(\forall p' : \text{Pos})(p' \in \text{claves}(d) \Rightarrow_{\text{L}} p' \in posiciones(e))\}$
manhattan(e, p, d) $\equiv$ **if** vacio?(claves(d)) **then**
1
else if distManhattan$(p,$ proximo$) \leq 3 \wedge p \neq$ proximo **then**
max(obtener(proximo, d) + nivelesPorUnion$(e,$ proximo),
manhattan$(p,$ borrar(proximo, d)))
else
manhattan$(e,\ p,$ borrar(proximo, d))
fi
donde proximo $\equiv$ dameUno(claves(d))

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

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

2.2.3. Implementación

---

iniciar(**in** m: Mapa) $\longrightarrow$ res : estr
  estr.turno $\leftarrow$ 0
  estr.popularidad $\leftarrow$ 0
  estr.mapa $\leftarrow$ m
  estr.casas $\leftarrow$ vacio()
  estr.comercios $\leftarrow$ vacio()
  estr.uniones $\leftarrow$ vacia()
  return estr
Complejidad: O(1)

---

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

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

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

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

Complejidad: O(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)

Asumo que existe un conjunto U $\equiv$ {u_1, u_2, ... , u_n}
tal que cada uno de esos U_i son los simcities que componen al simcity original
llamamos nodos : #U
llamamos sumCasas : $\sum$_{i=1, nodos}^{copiar(u_i.casas)}
llamamos sumComercios : $\sum$_{i=1, nodos}^{copiar(u_i.comercios)}
llamamos sumMapas : $\sum$_{i=1, nodos}^{u_i.mapa}

---

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(sumMapas)
cabe aclarar que la suma de los mapas esta definida

---

casas(**in** SimCity s) → res : dicc(Pos, Nivel)
    dicc res ← copiar(s.casas)
    dicc comerciosTotales ← copiar(s.comercios)
    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) ∧ ¬def?(comerciosTotales, p)) :
                definir(res, p, n + s.uniones[i].turnosDesdeUnion)
            avanzar(itCs)
        comerciosTotales ← comerciosTotales ∪ s.uniones[i].sc->comercios
    return res
Complejidad: O([sumComercios + sumCasas]^2 + nodos)

---

comercios(**in** SimCity s) → res : dicc(Pos, Nivel)
1: **return** comerciosAux(s, casas(s), 0)
Complejidad: O(casas(s))

comerciosAux(**in** SimCity s, **in** casasTotales dicc(Pos, Nivel), **in** Nat tdu) → res : dicc(Pos, Nivel)
1: dicc res ← vacio()
2: itDicc(Pos, Nivel) itCs ← crearIt(s.comercios);
3: **while**(haySiguiente(itCs)) :
4: Pos p ← siguienteClave(itCs)
5: Nivel n ← siguienteSignificado(itCs)
6: Nivel m ← max(n + tdu, nivelCom(p, casasTotales))
7: definir(res, p, m)
8: avanzar(itCs)
9: **for**(nat i ← 0; i < long(s.uniones); i ← i + 1) :
10: tdu ← s.uniones[i].turnosDesdeUnion
11: itDicc(Pos, Nivel) itCs ← crearIt(**comerciosAux**(s.uniones[i].sc*, casasTotales, tdu));
12: **while**(haySiguiente(itCs)) :
13: Pos p ← siguienteClave(itCs)
14: Nivel n ← siguienteSignificado(itCs)
15: **if**(¬def?(res, p) ∧ ¬def?(casasTotales, p)) :
16: definir(res, p, n)
17: avanzar(itCs)
18: **return** res
Complejidad: O(sumComercios*sumCasas + sumComercios^2 + nodos)

---

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

---

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

---

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

1: definirRapido(casas, p, n)
Complejidad: O(1)

---

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

---

turnos(**in** SimCity s) → res : Nat
1: **return** s.turno
Complejidad: O(1)

• ∪ •(**in** dicc(α, β) d1, **in** dicc(α, β) d2) → res : dicc(α, β)
1: dicc(α, β) res = copiar(d1)
2: itDicc(α, β) itCs ← crearIt(d2);
3: **while**(haySiguiente(itCs)) :
4: α a ← siguienteClave(itCs)
5: β b ← siguienteSignificado(itCs)
6: **if**(¬def?(res, a)) :
7: definir(res, a, b)
8: avanzar(itCs)
9: **return** res
Complejidad: O(copy(d1) + #claves(d2))

---

construcc(**in** SimCity s) → res : Nat
1: **return** **casas**(s) ∪ **comercios**(s)
Complejidad: O(**casas**(d1) + **comercios**(d2))

2.3. Módulo Servidor

2.3.1. Interfaz

Interfaz

se explica con: SERVIDOR

géneros: `server`

Operaciones básicas de server

NUEVOSERVER() $\rightarrow$ *res* : `server`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} nuevoServer$}
Complejidad: $O(1)$
Descripción: Crea un servidor
Aliasing: No tiene

PARTIDAS(**in** s: `server`) $\rightarrow$ *res* : `dicc(string, SimCity)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} partidas(s)$}
Complejidad: $O(copy(server))$
Descripción: Devuelve un diccionario con todas las partidas del servidor
Aliasing: Devuelve una copia (Esto habria que verlo, ya que no tenemos este dicc(nombre, simcity) por asi decirlo. Maybe hacemos uno de cero? Y tambien habria que ver si lo devolvemos con los SimCity en las hojas o son punteros?)

CONGELADAS(**in** s: `server`) $\rightarrow$ *res* : `conj(string)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} congeladas(s)$}
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 $\equiv$ {$s =_{\text{obs}} s0 \ \wedge \ \neg def?(p, \ partidas(s))$}
Post $\equiv$ {$s =_{\text{obs}} nuevaPartida(s0, \ p, \ m)$}
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 $\equiv$ {$s =_{\text{obs}} unirPartidas(s0, \ p1, \ p2)$}
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 $\equiv$ {$def?(p, \ s) \ \wedge_{\text{L}} \ *avanzarTurnoValido(s, \ p, \ pendientes(p, \ s))$}
Post $\equiv$ {$s =_{\text{obs}} avanzarTurnoPartida(s0, \ p)$}
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 $\equiv$ {$s =_{\text{obs}} s0 \ \wedge \ *agregarValido(s, \ p, \ pos)$}
Post $\equiv$ {$s =_{\text{obs}} agregarCasa(s0, \ p, \ pos)$}
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 antiguedad de la partida
Aliasing: Devuelve una referencia no modificable

MAPA(**in** s: `server`, **in** p: `string`) $\rightarrow$ res : `mapa`
Pre $\equiv \{def?(p, partidas(s))\}$
Post $\equiv \{res =_{\text{obs}} verMapa(s,\ p)\}$
Complejidad: $O()$
Descripción: Devuelve el mapa de la partida
Aliasing: Devuelve una copia? (habria que ver como funciona mapa en la implementacion del simcity)

VERCASAS(**in** s: `server`, **in** p: `string`) $\rightarrow$ res : `dicc(Pos, Nat)`
Pre $\equiv \{def?(p,\ partidas(s))\}$
Post $\equiv \{res =_{\text{obs}} verCasas(s,\ p)\}$
Complejidad: $O()$
Descripción: Devuelve un diccionario con las posiciones y niveles de las casas de la partida
Aliasing: Devuelve una copia? (habria que ver como funciona casas en la implementacion del simcity)

VERCOMERCIOS(**in** s: `server`, **in** p: `string`) $\rightarrow$ res : `dicc(Pos, Nat)`
Pre $\equiv \{def?(p,\ partidas(s))\}$
Post $\equiv \{res =_{\text{obs}} verComercios(s,\ p)\}$
Complejidad: $O()$
Descripción: Devuelve un diccionario con las posiciones y niveles de los comercios de la partida
Aliasing: Devuelve una copia? (habria que ver como funciona comercios en la implementacion del simcity)

*donde:

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

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

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

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

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

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

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

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

$d \cup_{dicc} d' \equiv$ **if** $vacio?$(claves(d')) **then**
d
else
definir(dameUno(claves(d')),
obtener(dameUno(claves(d')), d'),
$d \cup_{dicc}$ borrar(dameUno(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, Construccion)`)

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

donde Nombre es string

donde Construccion es string

Rep : estr $\longrightarrow$ boolean
Rep(e) $\equiv$ true $\Longleftrightarrow$
(∀ $partida_1$, $partida_2$: Nombre)
((def?($partida_1$, e) ∧ def?($partida_2$, e) ∧ $partida_1 \neq partida_2$) $\Rightarrow_{\text{L}}$
obtener($partida_1$, e).sim ≠ obtener($partida_2$, e).sim
) ∧
(∀ *partida*: Nombre)(def?(*partida*, e) $\Rightarrow_{\text{L}}$
p.sim ≠ NULL ∧
(p.modificable $=_{\text{obs}}$ *false* $\Rightarrow_{\text{L}}$ *vacio*?(claves(p.pendientes))) ∧
(∀ *pos*: Pos)(def?(*pos*, p.pendientes) $\Rightarrow_{\text{L}}$
(obtener(*pos*, p.pendientes) ∈ {"*casa*", "*comercio*"} ∧
¬ def?(*pos*, construcciones(*p.sim)))
)
)

donde p es obtener(*partida*, e)

Abs : estr e $\longrightarrow$ servidor {Rep(e)}
Abs(e) $\equiv$ s: servidor |
(∀ *nombre*: Nombre)
(*nombre* ∈ congelados(s) ⇔
(def?(*nombre*, e) $\wedge_{\text{L}}$ obtener(*nombre*, e).modificable $=_{\text{obs}}$ *false*))
∧
(∀ *nombre*: Nombre)
(def?(*nombre*, partidas(s)) ⇔ def?(*nombre*, e))
$\wedge_{\text{L}}$
(∀ *nombre*: Nombre)
(def?(*nombre*, partidas(s)) $\Rightarrow_{\text{L}}$
(obtener(*nombre*, partidas(s)) $=_{\text{obs}}$ *(obtener(*nombre*, e).sim) ∧
pendientes(s, *nombre*) $=_{\text{obs}}$ obtener(*nombre*, e).pendientes))

2.3.3. Implementación

Algoritmos

---

nuevoServer() → *res* : estr

1: $res \leftarrow vacio()$ **return** res

Complejidad: $O(1)$

---

partidas(**in** e : `estr`) → *res* : diccTrie(Nombre, SimCity)

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

Complejidad: $O(copy(estr))$

---

congeladas(**in** e : `estr`) → *res* : conj(Nombre)

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

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

---

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

1: $definirRapido(e,\ p,\ \langle true,\ \&(iniciar(m)), vacio()\rangle)$ ▷ 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`) $\rightarrow$ *res* : DiccLineal(Pos, Nivel)

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

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

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

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

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

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

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

Complejidad: $O(|partida|)$

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

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

Complejidad: $O(|partida|)$