DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP de Especificación y Diseño

Modelado de 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

1. Especificación

1.1. Mapa

TAD MAPA

igualdad observacional

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

géneros Mapa

exporta Mapa, observadores, generadores, $\bullet + \bullet$, esRio

usa Nat, conj(a), Pos, Bool

observadores básicos

horizontales : Mapa $\longrightarrow$ conj(Nat)
verticales : Mapa $\longrightarrow$ conj(Nat)

Mapa

generadores

crear : conj(Nat) $\times$ conj(Nat) $\longrightarrow$ Mapa

otras operaciones

Mapa

$\bullet + \bullet$: Mapa $\times$ Mapa $\longrightarrow$ Mapa
esRio : Mapa $\times$ Pos $\longrightarrow$ Bool

axiomas $\forall hs, vs$: conj(Nat), $\forall m1, m2$: Mapa, $\forall p$: Pos

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

1.2. SimCity

TAD SimCity

igualdad observacional

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

géneros SimCity

exporta SimCity, observadores, generadores, turnos

usa Mapa, Nat, Pos, Construccion, dicc(α, β), Nivel

observadores básicos

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

generadores

iniciar	: Mapa	$\longrightarrow$	SimCity	
avanzarTurno	: SimCity s $\times$ dicc(Pos $\times$ Construccion) cs	$\longrightarrow$	SimCity	{**avanzarTurnoValido*(s, cs)}
unir	: SimCity a $\times$ SimCity b	$\longrightarrow$	SimCity	{**unirValido*(a, b)}

otras operaciones

turnos	: SimCity	$\longrightarrow$	Nat
construcciones	: SimCity	$\longrightarrow$	dicc(Pos, Nivel)
$\bullet \cup_{dicc} \bullet$	: dicc($\alpha \times \beta$) $\times$ dicc($\alpha \times \beta$)	$\longrightarrow$	dicc(α, β)
agCasas	: dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Construccion)	$\longrightarrow$	dicc(Pos,Nivel)
agComercios	: SimCity $\times$ dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Construccion)	$\longrightarrow$	dicc(Pos,Nivel)
nivelCom	: Pos $\times$ dicc(Pos $\times$ Nivel)	$\longrightarrow$	Nat
distManhatt	: Pos $\times$ Pos	$\longrightarrow$	Nat
sacarRepes	: dicc(Pos $\times$ Construccion) $\times$ dicc(Pos $\times$ Construccion)	$\longrightarrow$	dicc(Pos, Construccion)

axiomas $\forall s, s'$: simcity, $\forall cs, cs'$: dicc(Pos, Construccion), $\forall cn, cn'$: dicc(Pos, Nivel), $\forall d, d'$: dicc(α, β)

mapa(iniciar(m))	$\equiv$	m
mapa(avanzarTurno(s, cs))	$\equiv$	mapa(s)
mapa(unir(s, s'))	$\equiv$	crear(horizontales(s) $\cup$ horizontales(s'), verticales(s) $\cup$ verticales(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')))
agCasas(cn, cs)	$\equiv$	**if** *vacio*?(claves(cs)) **then**
		cn
		else
		if obtener(dameUno(claves(cs)), cs) $=_{\text{obs}}$ "*casa*" **then**
		agCasas(definir(dameUno(claves(cs)), 1, cn),
		borrar(dameUno(claves(cs)), cs))
		else
		agCasas(cn, borrar(dameUno(claves(cs)), cs))
		fi
		fi

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')))
agComercios(s, cn, cs) $\equiv$ **if** *vacio*?(claves(cs)) **then**
cn
else
if obtener(dameUno(claves(cs)), cs) $=_{\text{obs}}$ "*comercio*" **then**
agComercios(definir(dameUno(claves(cs)),
nivelCom(dameUno(claves(cs)), casas(s)), cn),
borrar(dameUno(claves(cs)), cs))
else
agComercios(cn, borrar(dameUno(claves(cs)), cs))
fi
fi
nivelCom(p, cn) $\equiv$ **if** *vacio*?(claves(cn)) **then**
1
else
if distManhatt(p, dameUno(claves(cn))) ≤ 3 **then**
max(obtener(dameUno(claves(cn)), cn),
nivelCom(p, borrar(dameUno(claves(cn)), cn)))
else
nivelCom(p, borrar(dameUno(claves(cn)), cn))
fi
fi
distManhatt(p, q) $\equiv$ **if** $\pi_0(p) < \pi_0(q)$ **then** $q - p$ **else** $p - q$ **fi**
$+$
if $\pi_1(p) < \pi_1(q)$ **then** $q - p$ **else** $p - q$ **fi**
popularidad(iniciar(m)) $\equiv$ 0
popularidad(avanzarTurno(s, cs)) $\equiv$ popularidad(s)
popularidad(unir(s, s')) $\equiv$ popularidad(s) + 1 + popularidad s'
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**
construcciones(s) $\equiv$ casas(s) $\cup_{dicc}$ comercios(s)
$d \cup_{dicc} d'$ $\equiv$ **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
sacarRepes(cs, cs') $\equiv$ **if** *vacio*?(claves(cs)) **then**
cs'
else
if def?(dameUno(claves(cs)), cs') **then**
sacarRepes(borrar(dameUno(claves(cs)), cs),
borrar(dameUno(claves(cs)), cs'))
else
sacarRepes(borrar(dameUno(claves(cs)), cs), cs')
fi
fi

Fin TAD

*donde:

avanzarTurnoValido : SimCity s × dicc(Pos × Construccion) cs $\longrightarrow$ boolean

$$\begin{aligned}\text{avanzarTurnoValido}(s, cs) \equiv\ & \neg vacio?(claves(cs)) \wedge \\ & (\forall p : Pos)(def?(p, cs) \Rightarrow_{\text{L}} \\ & \quad (\neg p \in claves(construcciones(s)) \wedge \\ & \quad \neg\pi_0(p) \in horizontales(mapa(\text{s})) \wedge \neg\pi_1(p) \in verticales(mapa(\text{s})) \wedge \\ & \quad (obtener(p, cs) =_{\text{obs}} \text{"}casa\text{"} \vee obtener(p, cs) =_{\text{obs}} \text{"}comercio\text{"})) \\ &)\end{aligned}$$

unirValido : Simcity a × SimCity b $\longrightarrow$ boolean

$$\begin{aligned}\text{unirValido}(a, b) \equiv\ & (\forall p : Pos)(def?(p, construcciones(a)) \Rightarrow_{\text{L}} \\ & \quad (\neg\pi_0(p) \in horizontales(mapa(b)) \wedge \neg\pi_1(p) \in verticales(mapa(b)) \wedge \\ & \quad (\neg\ (\exists\ otra : Pos)(def?(otra, construcciones(a)) \wedge_{\text{L}} \\ & \quad\quad obtener(otra, construcciones(a)) > obtener(p, construcciones(a)) \Rightarrow_{\text{L}} \\ & \quad\quad\quad \neg\ def?(p, construcciones(b))))) \\ &) \wedge \\ & (\forall p : Pos)(def?(p, construcciones(b)) \Rightarrow_{\text{L}} \\ & \quad (\neg\pi_0(p) \in horizontales(mapa(a)) \wedge \neg\pi_1(p) \in verticales(mapa(a)) \wedge \\ & \quad (\neg\ (\exists\ otra : Pos)(def?(otra, construcciones(b)) \wedge_{\text{L}} \\ & \quad\quad obtener(otra, construcciones(b)) > obtener(p, construcciones(b)) \Rightarrow_{\text{L}} \\ & \quad\quad\quad \neg\ def?(p, construcciones(a))))))\end{aligned}$$

1.3. Servidor

TAD SERVIDOR

géneros server

exporta observadores, generadores, verMapa, verCasas, verComercios, verPopularidad y verTurno

usa SimCity, Mapa, Nombre, Pos, Construccion, Nivel, Nat, bool, dicc(α, β), conj(α)

igualdad observacional

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

observadores básicos

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

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	$\{*unionValida(s,\ p1,\ p2,\ cs)\}$
avanzarTurnoPartida	:	server s $\times$ Nombre p	$\longrightarrow$	server	$\{def?(p,\ s) \wedge_{\text{L}} *avanzarTurnoValido(s,\ p,\ pendientes(s,\ p))\}$
agregarCasa	:	server s $\times$ Nombre p $\times$ Pos pos	$\longrightarrow$	server	$\{*agregarValido(s,\ p,\ pos)\}$
agregarComercio	:	server s $\times$ Nombre p $\times$ Pos pos	$\longrightarrow$	server	$\{*agregarValido(s,\ p,\ pos)\}$

otras operaciones

verMapa	:	server s $\times$ Nombre p	$\longrightarrow$	Mapa	$\{def?(p,\ partidas(s))\}$
verCasas	:	server s $\times$ Nombre p	$\longrightarrow$	dicc(Pos, Nivel)	$\{def?(p,\ partidas(s))\}$
verComercios	:	server s $\times$ Nombre p	$\longrightarrow$	dicc(Pos, 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, $\forall p,\ p1,\ p2$: Nombre, $\forall m$: Mapa, $\forall cs$: conj(Pos)

partidas(nuevoServer)	$\equiv$	$vacio$
partidas(nuevaPartida(s, p, m))	$\equiv$	$definir(p,\ iniciar(m),\ partidas(s))$
partidas(unirPartidas(s, p1, p2))	$\equiv$	$definir(p1,\ unir(obtener(p1,\ partidas(s)),\ obtener(p2,\ partidas(s))),\ partidas(s))$
partidas(avanzarTurnoPartida(s, p))	$\equiv$	$definir(p,\ avanzarTurno(obtener(p,\ partidas(s)),\ pendientes(s,\ p)),\ partidas(s))$
partidas(agregarCasa(s, p, pos))	$\equiv$	$partidas(s)$
partidas(agregarComercio(s, p, pos))	$\equiv$	$partidas(s)$
congeladas(nuevoServer)	$\equiv$	$\emptyset$
congeladas(nuevaPartida(s, p, m))	$\equiv$	$congeladas(s)$
congeladas(avanzarTurnoPartida(s, p))	$\equiv$	$congeladas(s)$
congeladas(unirPartidas(s, p1, p2))	$\equiv$	$Ag(p2,\ congeladas(s))$
congeladas(agregarCasa(s, p, pos))	$\equiv$	$congeladas(s)$
congeladas(agregarComercio(s, p, pos))	$\equiv$	$congeladas(s)$
pendientes(nuevaPartida(s, p1, m), p)	$\equiv$	**if** $p =_{\text{obs}} p1$ **then** $vacio$ **else** $pendientes(s,\ p)$ **fi**
pendientes(unirPartidas(s, p1, p2), p)	$\equiv$	$pendientes(s,\ p)$

pendientes(agregarCasa(s, p1, pos), p) $\equiv$ **if** $p =_{\text{obs}} p1$ **then**
    $definir(pos,\ "casa",\ pendientes(s,\ p))$
else
    $pendientes(s,\ p)$
fi

pendientes(agregarComercio(s, p1, pos), p) $\equiv$ **if** $p =_{\text{obs}} p1$ **then**
    $definir(pos,\ "comercio",\ pendientes(s,\ p))$
else
    $pendientes(s,\ p)$
fi

pendientes(avanzarTurnoPartida(s, p1), p) $\equiv$ **if** $p =_{\text{obs}} p1$ **then** *vacio* **else** $pendientes(s,\ p)$ **fi**

// *oo*

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

*donde:

unionValida : server s $\times$ Nombre p1 $\times$ Nombre p2 $\longrightarrow$ boolean
unionValida(s, p1, p2) $\equiv$ $def?(p1,\ partidas(s)) \land def?(p2,\ partidas(s)) \land$
$p1 \notin congeladas(s) \land_{\text{L}}$
$vacio?(claves(pendientes(s,\ p1))) \land vacio?(claves(pendientes(s,\ p2))) \land$
$(\forall\ pos :\ Pos)(pos \in claves(constr1) \Rightarrow_{\text{L}}$
$\neg sobreRio(pos,\ sim2) \land$
$((\nexists\ otra :\ Pos)(otra \in constr1 \land_{\text{L}}$
$obtener(pos,\ constr1).nivel < obtener(otra,\ constr1).nivel$
$) \Rightarrow_{\text{L}} \neg def?(pos,\ constr2))$
$) \land$
$(\forall\ pos :\ Pos)(pos \in claves(constr2) \Rightarrow_{\text{L}}$
$\neg sobreRio(pos,\ sim1) \land$
$((\nexists otra :\ Pos)(otra \in constr2 \land_{\text{L}}$
$obtener(pos,\ constr2).nivel < obtener(otra,\ constr2).nivel$
$) \Rightarrow_{\text{L}} \neg def?(pos,\ constr1))$
$)$
donde $sim1 \equiv obtener(p1,\ partidas(s))$,
$sim2 \equiv obtener(p2,\ partidas(s))$,
$constr1 \equiv casas(sim1) \cup comercios(sim1)$,
$constr2 \equiv casas(sim2) \cup comercios(sim2)$

avanzarTurnoValido : server s $\times$ Nombre p $\times$ dicc(Pos $\times$ Construccion) cs $\longrightarrow$ boolean
avanzarTurnoValido(s, p, cs) $\equiv$ $def?(p,\ partidas(s)) \land$
$p \notin congeladas(s) \land$
$\neg vacia?(claves(cs)) \land_{\text{L}}$

donde $sim \equiv obtener(p,\ partidas(s))$

agregarValido : server s $\times$ Nombre p $\times$ Pos pos $\longrightarrow$ boolean
agregarValido(s, p, pos) $\equiv$ $def?(p, partidas(s)) \land_{\text{L}} \neg p \in congeladas(s) \land$
$\neg def?(pos, verCasas(s, p)) \land \neg def?(pos, verComercios(s, p)) \land$
$\neg def?(pos,\ pendientes(s,\ p)) \land \neg esRio(pos,\ verMapa(s,\ p))$

$\bullet \cup \bullet$: dicc$(\alpha \times \beta) \times$ dicc$(\alpha \times \beta) \longrightarrow$ dicc(α, β)
$a \cup b \equiv \_definir(a,\ b,\ claves(b))$

_union : dicc$(\alpha \times \beta) \times$ dicc$(\alpha \times \beta)$ b $\times$ conj(α) cs $\longrightarrow$ dicc(α, β) $\{cs \subseteq claves(b)\}$
_union(a, b, cs) $\equiv$ **if** $vacio?(cs)$ **then**
a
else
$\_union(definir(dameUno(cs),\ obtener(dameUno(cs),\ b)),\ b,\ sinUno(cs))$
fi

2. Módulos de referencia

2.1. Módulo Mapa

Interfaz

se explica con: MAPA

géneros: `mapa`

Operaciones básicas de mapa

CREAR(**in** hs: `conj(Nat)`,**in** vs: `conj(Nat)`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} mapa(hs, vs)\}$
Complejidad: $O(copy(hs), copy(vs))$
Descripción: crea un mapa

ESRIO(**in** $m1$: `Mapa`,**in** p: `Pos`) $\rightarrow$ res : `Bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} esRio(m1, p)\}$
Complejidad: $O(1)$
Descripción: verifica si en determinada pos hay rio

SUMA(**in** $m1$: `Mapa`,**in** $m2$: `Mapa`) $\rightarrow$ res : `Mapa`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} m1 + m2\}$
Complejidad: $O(crear(m1) + crear(m2)$
Descripción: une 2 mapas

Representación

Representación de mapa

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

`mapa` **se representa con** `estr`

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

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

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

Algoritmos

---

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
```
mapa res ← copiar(m1)

itConj(Nat) itH ← crearIt(horizontales(m2))
while(haySiguiente(itH)) :
    Ag(horizontales(res), siguiente(itH))
    avanzar(itH)

itConj(Nat) itV ← crearIt(verticales(m2))
while(haySiguiente(itV)) :
    Ag(verticales(res), siguiente(itV))
    avanzar(itV)

return res
```
Complejidad: O(copiar(m1) + copiar(m2))

---

esRio(**in** m1: mapa, **in** p: Pos) $\longrightarrow$ res : Bool
```
return p.x ∈ verticales(m1) ∨ p.y ∈ horizontales(m1)
```
Complejidad: O(#horizontales(m1) + #verticales(m1))

2.2. Módulo SimCity

Interfaz

usa: Mapa, Diccionario Lineal, Pos ≡ tupla<Nat, Nat>, Nivel ≡ Nat, 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: $\Theta(\sum_{i=0}^{N} copy(mapa_i))$, 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: $\mathbf{O}(casas^2) \supset^3 \Theta(\sum_{i=0}^{N} (hasta_i \times casas_i + camino_i))$,
donde $hasta_i$ y $casas_i$ son respectivamente la cantidad de casas definidas[4] en $\{u_0 \ ... \ u_{i-1}\}$ y u_i[5], y $camino_i$ representa la cantidad de uniones para llegar de S a u_i[6].
Descripción: Retorna las posiciones y respectivos niveles de todas las casas en el juego actual.
Aliasing: No. Genera una copia.

COMERCIOS(**in** S: `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. se proveen dos complejidades, una más abstracta y una evaluada en consideración de las representaciones posibles dadas las restricciones impuestas.
4. Donde se entiende por 'definida' como aquellas casas que provienen del propio simCity y no de alguna de sus uniones.
5. Dado que consideramos la resolución de colisiones durante una unión válida como 'queda el primero', y se requiere una complejidad de $\Theta(1)$ para la unión, es esperable que crear una copia del conjunto total de casas en U requiera chequear para cada casa definida en u_i su pertenencia al resultado parcial de la copia. Donde, en el peor caso, equivale al total de casas definidas hasta entonces.
6. Entendiendo las relaciones en U como un rosetree con $raiz = S$ y la necesidad de inmutabilidad de cada $u_i \neq S$, es razonable considerar que el nivel de cada casa o comercio en u_i va a tener que calcularse en relación con su posición en el árbol.
7. Similar a $casas(S)$. En este caso se agrega la posibilidad de tener que evaluar por pertenencia en el total de las casas al conjunto de posiciones a distancia manhattan ≤ 3 del comercio actualmente siendo copiado, para conocer su nivel.

TURNOS(**in** S: 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: dicc(Pos, Construccion))
Pre $\equiv$ {$avanzarTurnoValido(\hat{s},\ \hat{cs})\ \wedge\ \hat{S}\ =\ S_0$}
Post $\equiv$ {$\hat{S}$ $=_{\text{obs}}$ $avanzarTurno(S_0,\ \hat{cs})$}
Complejidad: $\mathbf{O}(casas \times \#(claves(\hat{cs})) + N) \supset \Theta(casas_S \times \#(claves(\hat{cs})) + U_S)$,
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 $\equiv$ {$unionValida(\hat{S1},\ \hat{S2})\ \wedge\ \hat{S1}\ =\ S_0$}
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.

Representación

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

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

`SimCity` **se representa con** `estr`

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

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

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

Rep : estr[1] $\longrightarrow$ boolean

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

donde

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

Abs : estr e $\longrightarrow$ SimCity {Rep(e)}

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

donde

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

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

auxiliares para la representación

unirPunteros : secu(hijo) $\longrightarrow$ conj(puntero(estr))
unirPunteros(s) $\equiv$ $\_unirPunteros(s,\ \emptyset)$

$_$unirPunteros : secu(hijo) $\times$ conj(puntero(estr)) $\longrightarrow$ conj(puntero(estr))
$_$unirPunteros(s, p) $\equiv$ **if** $vacia?(s)$ **then**
p
else if $prim(s).sc \in p$ **then** // por si hay loops
$\_unirPunteros(fin(s),\ p)$
else
$\_unirPunteros((*(prim(s).sc)).uniones,\ Ag(prim(s).sc,\ p)) \cup$
$\_unirPunteros(fin(s),\ Ag(prim(s).sc,\ p))$
fi

unirMapas : conj(puntero(estr)) $\longrightarrow$ Mapa
unirMapas(ps) $\equiv$ **if** $vacio?(ps)$ **then**
$crear(\emptyset,\ \emptyset)$
else
$(*(dameUno(ps))).mapa\ +\ UnirMapas(sinUno(ps))$
fi

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

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

remover : secu(α) $\times$ α $\longrightarrow$ secu(α) // remueve la primer aparición
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 $\times$ secu(hijo) $\longrightarrow$ bool
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	$\equiv$	$claves(casasult) \cup claves(comerult)$

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.

```
filtrar : dicc(Pos × Nat) × Nat ⟶ dicc(Pos, Nat)
filtrar(d, n) ≡ if vacio?(d) then
                    vacio
                else if sig ≤ n then
                    filtrar(borrar(clave, d), n)
                else
                    definir(clave, sig, filtrar(borrar(clave, d), n))
                fi
donde
    clave ≡ dameUno(claves(d))
    sig   ≡ obtener(clave, d)

maxcons : estr × conj(Pos) ⟶ conj(Pos)
maxcons(e, c) ≡ _maxcons(e, c, ∅, 0)

_maxcons : estr × conj(Pos) × conj(Pos) × Nat ⟶ conj(Pos)
_maxcons(e, c, max, n) ≡ if vacio?(c) then
                             max
                         else if nivel_i > n then
                             _maxcons(e, sinUno(c), Ag(pos_i, ∅), 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
donde
    pos_i   ≡ dameUno(c)
    nivel_i ≡ nivel(e, pos_i)

nivel : estr × pos ⟶ Nat
nivel(e, pos) ≡ if def?(pos, Casas) then
                    obtener(pos, Casas) + nivelesPorUnion(e, pos)
                else
                    maximo(obtener(pos, Comercios) • nManhattan) + nivelesPorUnion(e, pos)
                fi
donde:
    Unidos     ≡ unirPunteros(e.uniones)
    Casas      ≡ unirCasas(Ag(&e, Unidos))
    Comercios  ≡ unirComercios(Ag(&e, Unidos))
    nManhattan ≡ significados(manhattan(pos, 3), Casas)

nivelesPorUnion : estr × pos ⟶ Nat
nivelesPorUnion(e, pos) ≡ if def?(pos, e.casas) ∨ def?(pos, e.comercios) then
                              0
                          else
                              hijoCorrecto.turnosDesdeUnion +
                                  nivelesPorUnion(hijoCorrecto.sc, pos)
                          fi
donde:
    hijoCorrecto ≡ llegar(e.uniones, pos)

llegar : secu(hijo) × pos ⟶ hijo
llegar(s, p) ≡ if def?(pos, unirCasas(hijo) ∨ def?(pos, unirComercios(hijo)) then
                   prim(s)
               else
                   llegar(fin(s))
               fi
donde
```

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

manhattan : Pos $\times$ Nat $\longrightarrow$ Conj(Pos)
manhattan(p, dist) $\equiv$ **if** $dist = 0$ **then**
$Ag(p,\ \emptyset)$
else
$diagonal(\{p.x,\ p.y + dist\},\ \{p.x + dist,\ p.y\}) \cup$
if $p.x - dist \geq 0$ **then**
$diagonal(\{p.x,\ p.y + dist\},\ \{p.xdist,\ p.y\})$
else
$\emptyset$
fi $\cup$
if $p.y - dist \geq 0$ **then**
$diagonal(\{p.x,\ p.ydist\},\ \{p.x + dist,\ p.y\})$
else
$\emptyset$
fi $\cup$
if $p.x - dist \geq 0 \wedge p.y - dist \geq 0$ **then**
$diagonal(\{p.x,\ p.ydist\},\ \{p.xdist,\ p.y\})$
else
$\emptyset$
fi $\cup$
$manhattan(p,\ dist - 1)$
fi

diagonal : pos $\times$ pos $\times$ Nat n $\longrightarrow$ Conj(pos)
diagonal(d, h, y) $\equiv$ **if** $y = |h.y - d.y|$ **then**
$Ag(h,\ \emptyset)$
else
$Ag(caminar(d,\ h,\ y),\ diagonal(d,\ h,\ y + 1))$
fi

caminar : pos $\times$ pos $\times$ Nat $\longrightarrow$ pos
caminar(d, h, y) $\equiv$ **if** $d.y \leq h.y$ **then**
if $d.x \leq h.x$ **then**
$\{d.x + y,\ d.y + y\}$
else
$\{d.x - y,\ d.y + y\}$
fi
else
if $d.x \leq h.x$ **then**
$\{d.x + y,\ d.y - y\}$
else
$\{d.x - y,\ d.y - y\}$
fi
fi

significados : conj(α) $\times$ dicc($\alpha \times \beta$) $\longrightarrow$ secu(α)
significados(c, d) $\equiv$ **if** $vacio?(c)$ **then**
<>
else if $def?(dameUno(c),\ d)$ **then**
$obtener(c,\ d) \bullet significados(sinUno(c),\ d)$
else
$significados(sinUno(c),\ d)$
fi

maximo : secu(Nat) a $\longrightarrow$ Nat $\{long(a) > 0\}$
maximo(s) $\equiv$ **if** $long(s) = 1$ **then** $prim(s)$ **else** $max(prim(s),\ maximo(fin(s)))$ **fi**

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

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

Algoritmos

========== **Generadores** ==========

---

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

---

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

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

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

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

    estr.turno ⟵ estr.turno + 1

---

unir(**inout** SimCity s1, **inout** Simcity s2)
    s1.popularidad ← s1.popularidad + s2.popularidad
    turno ← max(s1.turno, s2.turno)
    hijo nuevoHijo ← <direccion(s2), 0>
    agregarAtras(s1.uniones, nuevoHijo)

==========

========== **Observadores** ==========

---

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

---

casas(**in** SimCity s) → res : dicc(Pos, Nivel)
    dicc res ← copiar(s.casas)

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

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

```
popularidad(in SimCity s) → res : Nat
    return s.popularidad
```

==========

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

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

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

```
agComercio(inout dicc(Pos, Nivel) comercios, in Pos p, in Nivel n) :
    definirRapido(comercio, p, n)
```

```
turnos(in SimCity s) → res : Nat
    return s.turno
```

```
• ∪ •(in dicc(α, β) d1, in dicc(α, β) d2) → res : dicc(α, β)
    dicc(α, β) res = copiar(d1)
    itDicc(α, β) itCs ← crearIt(d2);
    while(haySiguiente(itCs)) :
        α a ← siguienteClave(itCs)
        β b ← siguienteSignificado(itCs)
        if(¬def?(res, a)) :
```

```
            definir(res, a, b)
        avanzar(itCs)
    return res
```

```
construcc(in SimCity s) → res : Nat
    return casas(s) ∪ comercios(s)
```

==========

2.3. Módulo Servidor

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

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))

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

---

nuevoServer() → *res* : estr

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

Complejidad: $O(1)$

---

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

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

Complejidad: $O(copy(estr))$

---

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

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

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

---

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

```
1: definirRapido(e, p, ⟨true, &(iniciar(m)), vacio()⟩)    ▷ Reservamos memoria para el nuevo SimCity
```

Complejidad: $O(|p|)$

---

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

```
1: definir(e, p1, ⟨true, &(unir(*significado(p1, e).sim, *significado(p2, e).sim)), vacio()⟩)
2: definir(e, p2, ⟨false, significado(p2, e).sim, vacio()⟩)
```

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

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

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

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

Complejidad: $O(|partida|)$

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

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

Complejidad: $O(|partida|)$