1. Especificación

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, horizontales, verticales, crear

usa NAT, CONJUNTO(Nat)

observadores básicos

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

Mapa

generadores

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

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

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

Fin TAD

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, unirDiccionarios, MayorNivel, juntarSecu

usa MAPA, NAT, POS, NIVEL, CONSTRUCCION, DICCIONARIO(Pos, Nivel), STRING

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

$$\left\{ \begin{array}{l} (\neg \text{vacío(cs)}) \wedge (\forall \text{ p : Pos) (clave} \in \text{claves(cs))} \Rightarrow_{\text{L}} (\pi_1(clave) \notin \text{horizontales(mapa(s))} \wedge \\ \pi_2(clave) \notin \text{verticales(mapa(s)))} \end{array} \right\}$$

unir : SimCity a $\times$ SimCity b $\longrightarrow$ SimCity

$$\left\{ \begin{array}{l} \text{Habilitado(sc1)} \wedge \text{Habilitado(sc2)} \wedge (\forall \text{ pa, pb : Pos) (pa} \in \text{unirDiccionarios(casas(a), co-} \\ \text{mercios(a))) } \wedge \text{ (pb} \in \text{unirDiccionarios(casas(b), comercios(b))) } \Rightarrow_{\text{L}} \text{ ((}\pi_1\text{(pa)} \notin \text{horizonta-} \\ \text{les(mapa(b))} \wedge \pi_2\text{(pa)} \notin \text{verticales(mapa(b)))} \wedge (\pi_1\text{(pb)} \notin \text{horizontales(mapa(a))} \wedge \pi_2\text{(pb)} \\ \notin \text{verticales(mapa(a))))} \wedge (\forall \text{ mna, mnb : Pos)(mna} \in \text{MayorNivel(a)) (mnb} \in \text{Mayor-} \\ \text{Nivel(b))} \Rightarrow_{\text{L}} \text{ (mna} \notin \text{unirDiccionarios(casas(b), comercios(b)))} \wedge \text{ (mnb} \notin \text{unirDiccionа-} \\ \text{rios(casas(a), comercios(a)))} \end{array} \right\}$$

otras operaciones

turnos : SimCity $\longrightarrow$ Nat
subirNivelDicc : dicc(Pos $\times$ Nivel) $\longrightarrow$ dicc(Pos, Nivel)
unirDiccionarios : dicc(Pos $\times$ Nivel) $\times$ dicc(Pos $\times$ Nivel) $\longrightarrow$ dicc(Pos, Nivel)

filtrarConstr : dicc(Pos × Construccion) × string ⟶ dicc(Pos, Nivel)
MayorNivel : SimCity × conj(Nivel) ⟶ conj(Pos)
esDistanciaManhattanTres : Pos × Pos ⟶ Bool
juntarSecu : dicc(Pos × Nivel) × dicc(Pos × Nivel) ⟶ sec(tupla(Pos, Nivel))
habilitado : SimCity ⟶ Bool
ComerciosCorrectos : dicc(Pos × Nivel) × conj(Pos) ⟶ dicc(Pos, Nivel)

axiomas $\forall s, s', a, b$: SimCity, $\forall cs$: dicc(Pos, Construccion), $\forall d, d1, d2$: dicc(Pos, Nivel, $\forall p$: Pos, $\forall m$: Mapa

mapa(iniciar(m)) ≡ m
mapa(avanzarTurno(s, cs)) ≡ mapa(s)
mapa(unir(a, b)) ≡ crear((horizontales(mapa(a)) ∪ horizontales(mapa(b)), verticales(mapa(a)) ∪ verticales(mapa(b))))
casas(iniciar(m)) ≡ vacío
casas(avanzarTurno(s, definir(c,s vacio)) ≡ subirNivelDicc(definir(c,s,casas(s)))
casas(avanzarTurno(s, cs))) ≡ unirDiccionarios(subirNivelDicc(casas(s)), subirNivelDicc(filtrarConstr(cs, "*Casa*"))
casas(unir(a, b)) ≡ unirDiccionarios(casas(b), casas(a))
comercios(iniciar(m)) ≡ vacío
comercios(avanzarTurno(s,definir(c, s, vacio))) ≡ subirNivelDicc(definir(c, s, comercios(s)))
comercios(avanzarTurno(s, cs)) ≡ unirDiccionarios(subirNivelDicc(comercios(s)), subirNivelDicc(filtrarConstr(cs, "*Comercio*"))
comercios(unir(a, b)) ≡ ComerciosCorrectos(comercios(a), comercios(b))
ComerciosCorrectos(comercios1, claves(comercios2)) ≡
if dameUno(claves(comercios1)) = dameUno(claves(comercios2)) **then**
ComerciosCorrectos(comercios1, SinUno(claves(comercios2)))
else
ComerciosCorrectos(definir(comercios1, dameUno(claves(comercios2), NuevoNivel(comercios1, comercios2, dameUno(claves(comercios2)))))
fi

popularidad(iniciar(m)) ≡ 0
popularidad(avanzarTurno(s, cs)) ≡ popularidad(s)
popularidad(unir(a, b)) ≡ popularidad(a) + popularidad(b) + 1
turnos(iniciar(m)) ≡ 0
turnos(avanzarTurno(s, cs)) ≡ turnos(s) + 1
turnos(unir(a, b)) ≡ MAX(turnos(a), turnos(b))
subirNivelDicc(d) ≡ definir(dameUno(claves(d)), obtener(dameUno(claves(d)))+1, subirNivelDicc(borrar(dameUno(claves(d)),d))
unirDiccionarios(d, vacio) ≡ d
unirDiccionarios(d1, d2) ≡ unirDiccionarios(definir(dameUno(claves(d1)), obtener(dameUno(claves(d1)), d2), borrar(dameUno(claves(d1))))
filtrarConstr(vacio, s) ≡ vacio
filtrarConstr(d, s) ≡ **if** obtener(dameUno(claves(d))) = s **then**
definir(dameUno(claves(d)), obtener(dameUno(claves(d))), filtrarConstr(sinUno(claves(d)), s)
else
filtrarConstr(sinUno(claves(d)), s)
fi
subirNivelDiccionario(vacio) ≡ vacio
subirNivelDiccionario(cs) ≡ definir(dameUno(claves(cs)), obtener(dameUno(claves(cs))+1), subirNivelDiccionario(borrar(dameUno(claves(cs)),cs))

MayorNivel(s, conj(Nivel)) ≡ **if** obtener(dameUno(claves(unirDiccionarios(casas(s), comercios(s))))) = turnos(s) **then**
Ag(obtener(dameUno(claves(juntarSecu(casas(s), comercios(s))))), MayorNivel(s, sinUno((claves(juntarSecu(casas(s), comercios(s)))))))
else
MayorNivel(s, sinUno((claves(juntarSecu(casas(s), comercios(s))))))
fi

esDistanciaManhattanTres(p1, p2) ≡ $(|\pi_1(p2) - \pi_1(p1)| + |\pi_2(p2) - \pi_2(p1|) = 3$

juntarSecu(d1, d2) ≡ (dameUno(claves(d1)), obtener(dameUno(claves(d1)))) • (dameUno(claves(d2)), obtener(dameUno(claves(d2)))) • juntarSecu(borrar(dameUno(claves(d1)), d1), borrar(dameUno(claves(d2)), d2)))

habilitado(iniciar(m)) ≡ True

habilitado(unir(a,b), s) ≡ **if** s $=_{obs}$ b **then** false **else** habilitado(s) **fi**

Fin TAD

TAD SERVIDOR

igualdad observacional

$$(\forall s, s' : \text{Servidor}) \left(s =_{obs} s' \iff \begin{pmatrix} \text{Nombre(s)} =_{obs} \text{Nombre(s')} \wedge_{\text{L}} (\forall \text{ ID : Nombre}) \\ (\text{ID} \in \text{Nombre(s)}) \Rightarrow_{\text{L}} \text{MapaServidor(s, ID)} =_{obs} \\ \text{MapaServidor(s', ID)} \wedge \text{CasasServidor(s, ID)} =_{obs} \\ \text{CasasServidor(s', ID)} \wedge \text{ComerciosServidor(s, ID)} \\ =_{obs} \text{ComerciosServidor(s', ID)} \wedge (\text{PopularidadServidor(s, ID)} =_{obs} \text{PopularidadServidor(s', ID)} \wedge \\ \text{TurnosServidor(s, ID)} =_{obs} \text{TurnosServidor(s', ID)} \end{pmatrix} \right)$$

géneros Servidor

exporta Servidor, observadores, generadores

usa SIMCITY, MAPA, NAT, NOMBRE, POS, CONSTRUCCION, DICCIONARIO(Pos, Nivel)

observadores básicos

Nombres : Servidor ⟶ dicc(Nombre, SimCity)

generadores

IniciarServidor : ⟶ Servidor

AgregarPartida : Servidor × Nombre × SimCity ⟶ Servidor

AvanzarTurnoServidor : Servidor s × dicc(Pos × Construccion) d × Nombre ID ⟶ Servidor
$\{(\neg \text{vacío(d)}) \wedge \text{ ID} \in \text{claves(Nombres(s))} \Rightarrow_{\text{L}} (\forall \text{ p : Pos}) (\text{clave} \in \text{claves(d)}) \Rightarrow_{\text{L}} (\pi_1(clave) \notin \text{horizontales(mapa(obtener(ID, Nombres(s))))} \wedge \pi_2(clave) \notin \text{verticales(mapa(obtener(ID, Nombres(s)))))}\}$

UnirSC : Servidor s × Nombres IDa × Nombre IDb ⟶ Servidor
$\{\text{IDa, IDb} \in \text{claves(Nombres(s))} \Rightarrow_{\text{L}} (\text{Habilitado(obtener(IDa, Nombres(s)))} \wedge \text{Habilitado(obtener(IDb, Nombres(s)))} \wedge (\forall \text{ pa, pb : Pos}) (\text{pa} \in \text{unirDiccionarios(casas(obtener(IDa, Nombres(s))), comercios(obtener(IDa, Nombres(s)))))} \wedge (\text{pb} \in \text{unirDiccionarios(casas(obtener(IDb, Nombres(s))), comercios(obtener(IDb, Nombres(s)))))} \Rightarrow_{\text{L}} ((\pi_1(\text{pa}) \notin \text{horizontales(mapa(obtener(IDb, Nombres(s))))} \wedge \pi_2(\text{pa}) \notin \text{verticales(mapa(obtener(IDb, Nombres(s)))))} \wedge (\pi_1(\text{pb}) \notin \text{horizontales(mapa(obtener(IDa, Nombres(s))))} \wedge \pi_2(\text{pb}) \notin \text{verticales(mapa(obtener(IDa, Nombres(s))))))} \wedge (\forall \text{ mna, mnb : Pos})(\text{mna} \in \text{MayorNivel(obtener(IDa, Nombres(s))))} (\text{mnb} \in \text{MayorNivel(obtener(IDb, Nombres(s))))} \Rightarrow_{\text{L}} (\text{mna} \notin \text{unirDiccionarios(casas(obtener(IDb, Nombres(s))), comercios(obtener(IDb, Nombres(s)))))} \wedge (\text{mnb} \notin \text{unirDiccionarios(casas(obtener(IDa, Nombres(s))), comercios(obtener(IDa, Nombres(s))))))\}$

axiomas $\forall sc$: SimCity, $\forall s$: Servidor, $\forall d$: dicc(Pos, Nivel), $\forall ID, ID1, ID2$: Nombre

Nombres(IniciarServidor()) ≡ vacío

Nombres(AgregarPartida(s, ID, sc)) ≡ definir(ID, sc, Nombres(s))

Nombres(AvanzarTurnoServidor(s, d, ID)) ≡ avanzarTurno(obtener(ID, Nombres(s)), d)

Nombres(UnirSC(s, ID1, ID2) ≡ unir(obtener(ID1, Nombres(s)), obtener(ID2, Nombres(s)))

TAD Nivel es Nat
TAD Pos es Tupla(Nat, Nat)
TAD Casa es Tupla(Pos, Nivel)
TAD Comercio es Tupla(Pos, Nivel)
TAD Mapa es Tupla(conj(nat), conj(nat))
TAD Nombre es String
TAD Construccion es string

Fin TAD

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)`) → *res* : `mapa`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} mapa(hs, vs)$}
Complejidad: $O(copy(hs), copy(vs))$
Descripción: crea un mapa

HORIZONTALES(**in** *mapa*: `mapa(hs, vs)`) → *res* : `Conj(Nat)`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} hs$}
Complejidad: $O(copy(hs))$
Descripción: Toma un mapa y devuelve las lineas horizontales del mismo.

VERTICALES(**in** *mapa*: `mapa(hs, vs)`) → *res* : `Conj(Nat)`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} vs$}
Complejidad: $O(copy(vs))$
Descripción: Toma un mapa y devuelve las lineas verticales del mismo.

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 $\longrightarrow$ bool
Rep(e) ≡ true $\Longleftrightarrow$ true

Abs : estr m $\longrightarrow$ mapa {Rep(m)}
Abs(m) ≡ horizontales(m) = *estr*.horizontales ∧ verticales(m) = *estr*.verticales

Algoritmos

---

crear(**in** *hs*: `conj(Nat)`, **in** *vs*: `conj(Nat)`) → *res* : estr

1: *estr.horizontales* ← *hs*
2: *estr.verticales* ← *vs* **return** estr

Complejidad: $O(copy(hs) + copy(vs))$

Horizontales(**in** *mapa* : `mapa (Nat)`) → *res* : Conj(Nat)
1: **return** estr.horizontales

Complejidad: $O(1)$

Verticales(**in** *mapa* : `mapa (Nat)`) → *res* : Conj(Nat)
1: **return** estr.verticales

Complejidad: $O(1)$

2.2. Módulo Simcity

Interfaz

se explica con: SIMCITY

géneros: `SimCity`

Usa: `Bool, Nat, diccLineal, Conjunto, Lista Enlazada, Tupla`

Operaciones básicas de SimCity

MAPA(**in** *sc*: `Simcity`) → *res* : `mapa`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} sc.mapa$}
Complejidad: $O(copy(sc.mapa))$
Descripción: Devuelve el mapa de un SimCity

CASAS(**in** *sc*: `Simcity`) → *res* : `dicc(Pos, Nat)`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} sc.casas$}
Complejidad: $O(|Uniones(Sm)| * max|Uniones(Hijos)|)$
Descripción: Devuelve las casas de un SimCity

COMERCIOS(**in** *sc*: `Simcity`) → *res* : `dicc(Pos, Nat)`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} sc.comercios$}
Complejidad: $O(|Uniones(Sm)| * max|Uniones(Hijos)|)$
Descripción: Devuelve los comercios de un SimCity

POPULARIDAD(**in** *sc*: `Simcity`) → *res* : `Nat`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} sc.popularidad$}
Complejidad: $O(1)$
Descripción: Devuelve la popularidad de un SimCity

INICIAR(**in** *mapa*: `mapa`) → *res* : `estr`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} tupla(mapa, vacio(), vacio(), 0, 0$}
Complejidad: $O(1)$
Descripción: Crea una instancia nueva de un Simcity

AVANZARTURNO(**in** *Sc*: `Simcity`, **in** *d*: `dicc(Pos, Construccion)`) → *res* : `simcity`
Pre ≡ {habilitado(Sm) ∧ $(\forall p : Pos)(p \in$ claves(d)) ⇒ ($p \notin$ claves(casas(Sc)) ∧$p \notin$ claves(comercios(Sc)))}
Post ≡ {$res =_{\text{obs}} AvanzarTurno(Sc, d)$}
Complejidad: $O(claves(d))$
Descripción: Si las construcciones a agregar no chocan con ninguna ya existente, entones avanza el turno, agregando las construcciones correspondientes.

UNIR(**in** *Sc*1 : `Simcity`, **in** *Sc*2 : `Simcity`) → *res* : `simcity`

Pre $\equiv$ {habilitado(Sc2)}
Post $\equiv$ $\{res =_{\text{obs}} Unir(Sc1, Sc2)\}$
Complejidad: $O(1)$
Descripción: Agrega a la lista de uniones del Sc1, las uniones del Sc2; deshabilita Sc2, y aumenta la popularidad de Sc1.

Representación

Representación de SimCity

Un SimCity contiene un mapa el cual designa los ríos y además contiene casas, comercios un nivel de popularidad y un número de turno. Las casas y comercios son representados por diccionarios de sus posiciones y sus niveles, la popularidad y el turno son números naturales.

`SimCity` **se representa con** `simcity`

donde `simcity` es `tupla`(*Mapa*: `mapa`, *Casas*: `diccLineal(Pos, Nivel)`, *Comercios*: `diccLineal(Pos, Nivel)`, *Turno*: `Nat`, *Popularidad*: `Nat`, *Uniones*: `ListaEnlazada(Simcity)` , *Habilitado*: `Bool`)

Rep : simcity $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ ($\forall$ p : Pos)(p $\in$ claves(e.casas) $\Rightarrow_{\text{L}}$ (p $\notin$ claves(e.comercios) $\wedge$ π_1(p) $\notin$ verticales(e.mapa) $\wedge$ π_2(p) $\notin$ horizontales(e.mapa)) $\wedge$ (e.turno $\geq$ obtener(p, e.casas)) $\wedge$ ($\forall$ p' : Pos) (p' $\in$ claves(e.comercios) $\Rightarrow_{\text{L}}$ (p' $\notin$ claves(e.casas) $\wedge$ π_1(p') $\notin$ verticales(e.mapa) $\wedge$ π_2(p') $\notin$ horizontales(e.mapa)) $\wedge$ (e.turno $\geq$ obtener(p', e.casas))))

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

Abs(e) $\equiv$ ($e.Mapa = tupla(Horizontales(e), Verticales(e))$
$\wedge$(e.Casas=$_{\text{obs}}$Casas(e))
$\wedge$(e.Comercios=$_{\text{obs}}$Comercios(e))
$\wedge$(e.Popularidad=$_{\text{obs}}$Popularidad(e))
$\wedge$(e.habilitado =$_{\text{obs}}$ habilitado(e))

Algoritmos

---

exists(**in** D : `diccLineal(T, signif)`, **in** *elem* : `T`) $\rightarrow$ *res* : bool

```
1:  CrearIt(L) ← It
2:
3:  while HaySiguiente?(It) do
4:      if It == elem
5:   then res ← True
8:   else Siguiente(It) fi
9:  res ← False
10:     endwhile
```

Complejidad: $O(n)$

Mapa(**in** Sm : `simcity`) $\rightarrow$ *res* : mapa

```
1:  res ← Sm.Mapa
```

Complejidad:$O(copy(Sm.mapa)$ =0

Casas(**in** Sm : `simcity`) → res : diccLineal(Pos, Nivel)

```
1: CrearIt(Uniones(Sm)) ← Hijo
2:
3: while HaySiguiente?(Hijo) do
4:     CreatIt(Casas(Hijo)) ← CasasHijo
5:     while HaySiguiente?(CasasHijo) do
6:         if !exists(Casas(Sm), CasasHijo.pos) then
7:8:   AgregarAtras(Casas(Sm), CasasHijo);
       Siguiente(CasasHijo)
10: else
11:    Siguiente(CasasHijo)
12: fi
13:        endwhile
14:        CreatIt(Comercios(Hijo)) ← ComerciosHijo
15:        while HaySiguiente?(ComerciosHijo) do
16:            if !exists(Casas(Sm), ComerciosHijo.pos) then
17:18:  AgregarAtras(Casas(Sm), ComerciosHijo);
       Siguiente(ComerciosHijo)
20: else
21:    Siguiente(ComerciosHijo)
22: fi
23:            endwhile
24:            Siguiente(Hijo)
25: endwhile
26:            return res ← Sm.Casas
```

Complejidad: $O(|Uniones(Sm)| * max|Uniones(Hijos)|)$

Comercios(**in** Sm : `simcity`) → res : diccLineal(Pos, Nivel)

```
1: CrearIt(Uniones(Sm)) ← Hijo
2:
3: while HaySiguiente?(Hijo) do
4:     CrearIt(Comercios(Hijo)) ← ComerciosHijo
5:     while HaySiguiente?(ComerciosHijo) do
6:         if !exists(Comercios(Sm), ComerciosHijo.pos) then
7:8:   AgregarAtras(Comercios(Sm), ComerciosHijo),
       updateNivel(Sm, ComercioaHijo),
       Siguiente(ComerciosHijo)
10: else
11:    Siguiente(ComerciosHijo)
12: fi
13:        endwhile
14:        CrearIt(Casas(Hijo)) ← CasasHijo
15:        while HaySiguiente?(CasasHijo) do
16:            if !exists(Comercios(Sm), CasasHijo.pos) then
17:18:  AgregarAtras(Comercios(Sm), CasasHijo),
       Siguiente(CasasHijo)
19: else
20:    Siguiente(CasasHijo)
21: fi
22:            endwhile
23:            Siguiente(Hijo)
24: endwhile
25:            return res ← Sm.Comercios
```

Complejidad: $O(|Uniones(Sm)| * max|Uniones(Hijos)|)$

uniones(**in**/**out** *Sm*: `simcity`) → *res* : ListaEnlazada(simcity))

1: **return** res ← Sm.Uniones

Complejidad: $O(1)$ =0

Popularidad(**in** *Sm*: `simcity` → *res* : Nat)

1: **return** res ← Sm.Popularidad

Complejidad: $O(1)$

Iniciar(**in** *m*: `Mapa`) → *res* : simcity

1: simcity.Mapa ← m
2: simcity.Popularidad ← 0
3: simcity.Turno ← 0
4: simcity.Casas ← vacío()
5: simcity.Comercios ← vacío()
6: simcity.Uniones ← vacía()
7: simcity.habilitado ← true
8: **return** res ← simcity

Complejidad: $O(copy(m))$

AvanzarTurno(**in**/**out** *Sm*: `simcity`, **in** *d*: `diccLineal: (Pos, Construccion)`) → *res* : simcity

```
1:  CrearIt(d) ← i
2:
3:  while (HaySiguiente?(i) == True) do
4:
5:      if (i.valor == casa) then
6:          definirRapido(casas(sm), i.clave, i.valor) ← i
7:          Avanzar(i)
8:      else
9:          definirRapido(comercios(sm), i.clave, i.valor)← i
10:         Avanzar(i)
11:     end if
12:     endwhile
13:     simcity.Turno + = 1
14:     return res
```

Complejidad: $O(\#claves(d))$

Unir(**in**/**out** *sc*1: `simcity`, **in**/**out** *sc*2: `simcity`)

1: AgregarTree(uniones(sc1), uniones(sc2))
2: sc2.habilitado = False
3: turnos(sc1) + = 1
4: popularidad(sc1) = popularidad(sc1) + popularidad(sc2)

Complejidad: $O(1)$ (Aclaración: si usted ve un =0 cerca, por alguna razón no conseguimos encontrar el por qué se encuentra ni dónde está escrito para borrarlo.) =0

AgregarTree(**in/out** $Un1$: `ListaEnlazada(simcity)`, **in/out** $Un2$: `ListaEnlazada(simcity)`)

```
1: CrearItUlt(Un1) ← it1
2: CrearIt(Un2) ← it2
3: it1.siguiente ← it2.siguiente
4: it2.anterior ← it1.anterior
```

Complejidad: $O(1)$

updateNivel(**in/out** Sm: `simcity`, **in** $comercio$: `pair<Posicion,Nivel>`)

```
1: CreatIt(Casas(Sm)) ← casa
2:
3: while HaySiguiente?(casa) do
4:     if esDistManTres(casa.pos, comercio.pos) ∧ casa.nivel >comercio.nivel then
6:     comercio.nivel ← casa.nivel
       Siguiente(casa)
7: else
8:     Siguiente(casa)
9: fi
```

Complejidad: $O(claves(Sm.casas))$

esDistManTres(**in** $p1$: `Posicion`, **in** $p2$: `Posicion`) → res : bool

```
1: if |p1.first − p2.first| + |p1.second − p2.second| = 3  then
8: res ← True
7:  else
8: res ← False
9:  fi
```

Complejidad: $O(copy(p1) + copy(p2))$ =0

2.3. Módulo Servidor

Interfaz

se explica con: SERVIDOR

géneros: `Servidor`

Usa: `SimCity, diccLineal, char, Nombre, Construccion`

Operaciones básicas de Servidor

NOMBRES(**in** s: `Servidor`) → res : `diccLineal(Nombre, Simcity)`
Pre ≡ {true}
Post ≡ {res $=_{obs}$ s.Nombres}

[$O(|Nombre|)$] [Devuelve todos los simcitys con sus IDs]

Representación

Representación de Servidor

Un Servidor contiene un diccionario de Nombres, mapeados a sus correspondientes SimCitys. Los nombres

se representan como strings (conjuntos de chars en un trie).

`Servidor` **se representa con** `servidor`

donde `servidor` es `diccTrie(char, SimCity)`

Rep : servidor $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$ true

Abs : servidor s $\longrightarrow$ Servidor {Rep(s)}
Abs(s) $\equiv$ ($servidor.Nombres =_{\text{obs}} Nombres(s)$)

Algoritmos

IniciarServidor(**out** s : `Servidor`)

1: s $\leftarrow$ vacío()

Complejidad: $O(1)$

AgregarPartida(**in/out** s : `Servidor`, **in** ID : `Nombre`, **in** sc : `SimCity`)

1: $s.AgregarPartida(s, ID, sc) \leftarrow definir(ID, sc, s)$

Complejidad: $O(|ID|)$

UnirPartidas(**in/out** s : `Servidor`, **in** $ID1$: `Nombre`, **in** $ID2$: `Nombre`)

1: $s.UnirPartidas(ID1, ID2) \leftarrow unir(s(ID1), s(ID2))$

Complejidad: $O(|ID|)$

---

avanzarTurnoServidor(**in/out** s : `Servidor`, **in** ID : `Nombre`, **in** d : `dicc(Pos, Construccion)`)

1: $s.avanzarTurnoServidor(ID, d) \leftarrow avanzarTurno(s(ID), d)$

Complejidad: $O(|ID|)$
