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
construcc	: 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(construcc(s), construcc(s')))
agCasas(cn, cs)	$\equiv$	**if** *vacio*?(claves(cs)) **then**
		cn
		else
		if obtener(dameUno(claves(cs)), cs) $=_{\text{obs}}$ 1 **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(construcc(s), construcc(s')))
agComercios(s, cn, cs) $\equiv$ **if** *vacio*?(claves(cs)) **then**
cn
else
if obtener(dameUno(claves(cs)), cs) $=_{\text{obs}}$ 2 **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
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**
construcc(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:

$$\begin{aligned}
&\text{avanzarTurnoValido} : \text{SimCity } s \times \text{dicc(Pos} \times \text{Construccion) } cs \longrightarrow \text{boolean}\\
&\text{avanzarTurnoValido}(s, cs) \equiv \neg vacio?(claves(cs)) \wedge\\
&\quad (\forall p : Pos)(def?(p, cs) \Rightarrow_{\text{L}}\\
&\quad\quad (\neg p \in claves(construcc(s)) \wedge\\
&\quad\quad \neg\pi_0(p) \in horizontales(mapa(\text{s})) \wedge \neg\pi_1(p) \in verticales(mapa(\text{s})) \wedge\\
&\quad\quad (obtener(p, cs) =_{\text{obs}} 1 \vee obtener(p, cs) =_{\text{obs}} 2))\\
&\quad)\\
&\text{unirValido} : \text{Simcity } a \times \text{SimCity } b \longrightarrow \text{boolean}\\
&\text{unirValido}(a, b) \equiv (\forall p : Pos)(def?(p, construcc(a)) \Rightarrow_{\text{L}}\\
&\quad\quad (\neg\pi_0(p) \in horizontales(mapa(b)) \wedge \neg\pi_1(p) \in verticales(mapa(b)) \wedge\\
&\quad\quad (\neg\ (\exists\ otra : Pos)(def?(otra, construcc(a)) \wedge_{\text{L}}\\
&\quad\quad\quad obtener(otra, construcc(a)) > obtener(p, construcc(a)) \Rightarrow_{\text{L}}\\
&\quad\quad\quad\quad \neg\ def?(p, construcc(b)))))\\
&\quad\quad) \wedge\\
&\quad\quad (\forall p : Pos)(def?(p, construcc(b)) \Rightarrow_{\text{L}}\\
&\quad\quad\quad (\neg\pi_0(p) \in horizontales(mapa(a)) \wedge \neg\pi_1(p) \in verticales(mapa(a)) \wedge\\
&\quad\quad\quad (\neg\ (\exists\ otra : Pos)(def?(otra, construcc(b)) \wedge_{\text{L}}\\
&\quad\quad\quad\quad obtener(otra, construcc(b)) > obtener(p, construcc(b)) \Rightarrow_{\text{L}}\\
&\quad\quad\quad\quad\quad \neg\ def?(p, construcc(a))))))
\end{aligned}$$

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

$$(\forall s, s' : \text{server}) \left(s =_{\text{obs}} s' \iff \begin{pmatrix} \text{partidas}(s) =_{\text{obs}} \text{partidas}(s') \wedge \\ \text{congeladas}(s) =_{\text{obs}} \text{congeladas}(s') \end{pmatrix} \right)$$

observadores básicos

partidas : server $\longrightarrow$ dicc(Nombre, SimCity)
congeladas : server $\longrightarrow$ conj(Nombre)

generadores

nuevoServer : $\longrightarrow$ server
nuevaPartida : server s × Nombre p × Mapa $\longrightarrow$ server $\{\neg def?(p, partidas(s))\}$
unirPartidas : server s × Nombre p1 × Nombre p2 $\longrightarrow$ server $\{*unionValida(s, p1, p2, cs)\}$
avanzarTurnoPartida : server s × Nombre p × dicc(Pos × Construccion) cs $\longrightarrow$ server $\{*avanzarTurnoValido(s, p, cs)\}$

otras operaciones

verMapa	: server s × Nombre p	$\longrightarrow$ Mapa	$\{def?(p, partidas(s))\}$
verCasas	: server s × Nombre p	$\longrightarrow$ dicc(Pos, Nivel)	$\{def?(p, partidas(s))\}$
verComercios	: server s × Nombre p	$\longrightarrow$ dicc(Pos, Nivel)	$\{def?(p, partidas(s))\}$
verPopularidad	: server s × Nombre p	$\longrightarrow$ Nat	$\{def?(p, partidas(s))\}$
verTurno	: server s × 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, cs))	$\equiv definir(p, avanzarTurno(obtener(p, partidas(s)), cs), partidas(s))$
congeladas(nuevaPartida)	$\equiv \emptyset$
congeladas(nuevaPartida(s, p, m))	$\equiv congeladas(s)$
congeladas(avanzarTurnoPartida(s, p, cs))	$\equiv congeladas(s)$
congeladas(unirPartidas(s, p1, p2))	$\equiv Ag(p2, congeladas(s))$
// *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) $\equiv$ $def?(p1,\ partidas(s)) \land def?(p2,\ partidas(s)) \land$
$p1 \notin congeladas(s) \land_{\text{L}} \%$
$(\forall\ pos :\ Pos)(pos \in claves(constr1) \Rightarrow_{\text{L}}$
$\neg sobreRio(pos,\ sim2) \land$
$((\nexists\ otra :\ Pos)(otra \in constr1 \land_{\text{L}}$
$obtener(pos,\ constr1).nivel < obtener(otra,\ constr1).nivel$
$) \Rightarrow_{\text{L}} \neg def?(pos,\ constr2))$
$) \land$
$(\forall\ pos :\ Pos)(pos \in claves(constr2) \Rightarrow_{\text{L}}$
$\neg sobreRio(pos,\ sim1) \land$
$((\nexists otra :\ Pos)(otra \in constr2 \land_{\text{L}}$
$obtener(pos,\ constr2).nivel < obtener(otra,\ constr2).nivel$
$) \Rightarrow_{\text{L}} \neg def?(pos,\ constr1))$
$)$
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) $\equiv$ $def?(p,\ partidas(s)) \land$
$p \notin congeladas(s) \land$
$\neg vacia?(claves(cs)) \land_{\text{L}}$
$(\forall pos :\ Pos)(pos \in claves(cs) \Rightarrow_{\text{L}}$
$obtener(pos,\ cs) \in \{"casa",\ "comercio"\} \land$
$\neg sobreRio(pos,\ mapa(sim)) \land$
$\neg def?(pos,\ casas(sim)) \land$
$\neg def?(pos,\ comercios(sim))$
$)$
donde $sim \equiv obtener(p,\ partidas(s))$

$\bullet \cup \bullet$: 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`

TP de Especificación y DiseñoOperaciones 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

TP de Especificación y DiseñoRepresentación 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) $\equiv$ 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 $\leftarrow$ copiar(m1)

itConj(Nat) itH $\leftarrow$ 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) ⟶ res : Bool
    return p.x ∈ verticales(m1) ∨ p.y ∈ horizontales(m1)
```
Complejidad: O(#horizontales(m1) + #verticales(m1))

2.2. Módulo SimCity

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`

TP de Especificación y DiseñoOperaciones básicas de server

NUEVOSERVER() $\rightarrow$ *res* : `server`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} nuevoServer\}$
Complejidad: $O()$
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: $O()$
Descripción: Devuelve un diccionario con todas las partidas del servidor
Aliasing: Devuelve una referencia no modificable

CONGELADAS(**in** *s*: `server`) $\rightarrow$ *res* : `conj(string)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} congeladas(s)\}$
Complejidad: $O()$
Descripción: Devuelve el conjunto con los nombres de las partidas no modificables
Aliasing: Devuelve una referencia no modificable

NUEVAPARTIDA(**in/out** *s*: `server`, **in** *p*: `string`, **in** *m*: `mapa`)
Pre $\equiv$ $\{s =_{\text{obs}} s0 \land \neg def?(p, partidas(s))\}$
Post $\equiv$ $\{s =_{\text{obs}} nuevaPartida(s0, p, m)\}$
Complejidad: $O()$
Descripción: agrega una partida nueva al servidor
Aliasing: No tiene

UNIRPARTIDAS(**in/out** *s*: `server`, **in** *p*1: `string`, **in** *p*2: `string`)
Pre $\equiv$ $\{*unionValida(s, p1, p2)\}$
Post $\equiv$ $\{s =_{\text{obs}} nuevaPartida(s0, p, m)\}$
Complejidad: $O()$
Descripción: une dos partidas de simcity en una, y p2 pasa a ser no modificable
Aliasing: No tiene

AVANZARTURNOPARTIDA(**in/out** *s*: `server`, **in** *p*: `string`, **in** *cs*: `dicc(Pos, Nat)`)
Pre $\equiv$ $\{*avanzarTurnoValido(s, p, cs)\}$
Post $\equiv$ $\{s =_{\text{obs}} avanzarTurnoPartida(s0, p, cs)\}$
Complejidad: $O()$
Descripción: avanza el turno de una partida y agrega las construcciones definidas en el diccionario de entrada
Aliasing: No tiene

AGREGARCASA(**in/out** *s*: `server`, **in** *p*: `string`, **in** *pos*: `Pos`)
Pre $\equiv$ $\{s =_{\text{obs}} s0 \land def?(p, partidas(s)) \land_{\text{L}} \neg p \in congeladas(s) \land \neg def?(pos, verCasas(s, p)) \land \neg esRio(pos, verMapa(s, p))\}$
Post $\equiv$ $\{construccionesSeMantienen(s, s0, p, pos) \land casaAgregada(s, p, pos)\}$
Complejidad: $O()$
Descripción: agrega una nueva casa a la partida
Aliasing: No tiene

AGREGARCOMERCIO(**in/out** s : `server`, **in** $p1$: `string`, **in** $p2$: `string`)
Pre $\equiv \{s =_{\text{obs}} s0 \land def?(p, partidas(s)) \land_{\text{L}} \neg p \in congeladas(s) \land \neg def?(pos, verComercios(s,p)) \land \neg esRio(pos, verMapa(s,p))\}$
Post $\equiv \{s =_{\text{obs}} nuevaPartida(s0, p, m)\}$
Complejidad: $O()$
Descripción: agrega un nuevo comercio a 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 referencia no modificable

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 referencia no modificable

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 referencia no modificable

*donde:

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

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

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 × Nombre p × dicc(Pos × Construccion) cs $\longrightarrow$ boolean

$$\begin{aligned}\text{avanzarTurnoValido}(s, p, cs) \equiv\ & def?(p,\ partidas(s)) \wedge \\ & p \notin congeladas(s) \wedge \\ & \neg vacia?(claves(cs)) \wedge_{\text{L}} \\ & (\forall pos :\ Pos)(pos \in claves(cs) \Rightarrow_{\text{L}} \\ & \quad obtener(pos,\ cs) \in \{\text{"}casa\text{"},\ \text{"}comercio\text{"}\} \wedge \\ & \quad \neg sobreRio(pos,\ mapa(sim)) \wedge \\ & \quad \neg def?(pos,\ casas(sim)) \wedge \\ & \quad \neg def?(pos,\ comercios(sim)) \\ &)\end{aligned}$$

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

• $\cup_{dicc}$ • : dicc($\alpha \times \beta$) × 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

casaAgregada : servidor × string × pos $\longrightarrow$ boolean

$$\text{casaAgregada}(s, p, pos) \equiv def?(pos, verCasas(s, p)) \wedge_{\text{L}} obtener(pos, verCasas(s, p)) =_{\text{obs}} 1$$

construccionesSeMantienen : servidor × servidor × string × Pos $\longrightarrow$ boolean

$$\begin{aligned}
\text{construccionesSeMantienen}(s, s0, p, pos) \equiv\ & partidas(s)(\forall\ partida : string)(def?(partida, s) \Leftrightarrow \\
& def?(partida, s0)) \\
& \wedge_{\text{L}} \\
& (\forall\ partida : string)(def?(partida, s0)\ \wedge\ partida \neq p \Rightarrow_{\text{L}} \\
& \quad (\forall\ pos2 : Pos)(def?(pos2, verCasas(partida, s0)) \Leftrightarrow \\
& def?(pos2, verCasas(partida, s))) \\
& \quad \wedge \\
& \quad (\forall\ pos2 : Pos)(def?(pos2, verCasas(partida, s0)) \Leftrightarrow \\
& def?(pos2, verCasas(partida, s))))
\end{aligned}$$

Representación

TP de Especificación y DiseñoRepresentación 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 y su estado (si es modificable o no).

`servidor` **se representa con** `estr`

donde estr es diccTrie(nombre, tupla<modificable: bool, sim: puntero(SimCity)>)
donde nombre es string

Rep : estr $\longrightarrow$ bool

$$
\begin{aligned}
\text{Rep}(e) \equiv\ & \text{true} \iff \\
& (\forall\ partida_1, partida_2\text{: string}) \\
& (\text{def?}(partida_1, e) \land \text{def?}(partida_2, e) \land partida_1 \neq partida_2 \Rightarrow_{\text{L}} \\
& \quad \text{obtener}(partida_1, e).\text{sim} \neq \text{obtener}(partida_2, e).\text{sim} \\
&) \land \\
& (\forall\ partida\text{: string})(\text{def?}(partida, e) \Rightarrow_{\text{L}} \text{obtener}(partida, e) \neq \text{NULL})
\end{aligned}
$$

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

$$
\begin{aligned}
\text{Abs}(e) \equiv\ & \text{s: servidor} \mid \\
& (\forall\ nombre\text{: Nombre}) \\
& \quad (nombre \in \text{congelados}(s) \Leftrightarrow \\
& \quad (\text{def?}(nombre, \text{partidas}(e)) \land_{\text{L}} \text{obtener}(nombre, e).\text{modificable} =_{\text{obs}} false)) \\
& \land \\
& (\forall\ nombre\text{: Nombre}) \\
& \quad (\text{def?}(nombre, \text{partidas}(s)) \Leftrightarrow \text{def?}(nombre, e)) \\
& \land_{\text{L}} \\
& (\forall\ nombre\text{: Nombre}) \\
& \quad (\text{def?}(nombre, \text{partidas}(s)) \Rightarrow_{\text{L}} \\
& \quad \text{obtener}(nombre, \text{partidas}(s)) =_{\text{obs}} \text{obtener}(nombre, e).\text{sim})
\end{aligned}
$$