UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2015

1 de Abril de 2015

TPE - Flores vs Vampiros

1 Tipos

```
tipo Habilidad = Generar, Atacar, Explotar;
tipo ClaseVampiro = Caminante, Desviado;
tipo Posicion = (ℤ, ℤ);
tipo Vida = ℤ;
```

2 Flor

```
tipo Flor {
        observador vida (f: Flor) : ℤ;
        observador cuantoPega (f: Flor) : ℤ;
        observador habilidades (f: Flor) : [Habilidad];

        invariante sinRepetidos(habilidades(f));
        invariante lasHabilidadesDeterminanLaVidayElGolpe :
            vida(f) == 100 div (|habilidades(f)| + 1) ∧
            cuantoPega(f) == if Atacar ∈ habilidades(f) then 12 div |habilidades(f)| else 0;
}

problema nuevaF (v : ℤ, cP : ℤ, hs : [Habilidad]) = res : Flor {
        requiere v == 100 div (|hs| + 1);
        requiere cP == if Atacar ∈ hs then 12 div |hs| else 0;
        requiere |hs| ≥ 1;
        requiere sinRepetidos(hs);
        asegura vida(res) == v;
        asegura cuantoPega(res) == cP;
        asegura mismos(habilidades(res), hs);
}

problema vidaF (f: Flor) = res : ℤ {
        asegura res == vida(f);
}

problema cuantoPegaF (f: Flor) = res : ℤ {
        asegura res == cuantoPega(f);
}

problema habilidadesF (f: Flor) = res : [Habilidad] {
        asegura mismos(res, habilidades(f));
}
```

3 Vampiro

```
tipo Vampiro {
        observador clase (v: Vampiro) : ClaseVampiro;
        observador vida (v: Vampiro) : ℤ;
        observador cuantoPega (v: Vampiro) : ℤ;

        invariante vidaEnRango : vida(v) ≥ 0 ∧ vida(v) ≤ 100;
        invariante pegaEnSerio : cuantoPega(v) > 0;
}

problema nuevoV (cV : ClaseVampiro, v : ℤ, cP : ℤ) = res : Vampiro {
```

```
        requiere v ≥ 0 ∧ v ≤ 100;
        requiere cP > 0;
        asegura clase(res) == cV;
        asegura vida(res) == v;
        asegura cuantoPega(res) == cP;
}

problema claseVampiroV (v : Vampiro) = res : ClaseVampiro {
        asegura res == clase(v);
}

problema vidaV (v : Vampiro) = res : ℤ {
        asegura res == vida(v);
}

problema cuantoPegaV (v : Vampiro) = res : ℤ {
        asegura res == cuantoPega(v);
}
```

4 Nivel

```
tipo Nivel {
        observador ancho (n: Nivel) : ℤ;
        observador alto (n: Nivel) : ℤ;
        observador turno (n: Nivel) : ℤ;
        observador soles (n: Nivel) : ℤ;
        observador flores (n: Nivel) : [(Flor, Posicion, Vida)];
        observador vampiros (n: Nivel) : [(Vampiro, Posicion, Vida)];
        observador spawning (n: Nivel) : [(Vampiro, ℤ, ℤ)];
        invariante valoresRazonables : ancho(n) > 0 ∧ alto(n) > 0 ∧ soles(n) ≥ 0 ∧ turno(n) ≥ 0;
        invariante posicionesValidas : ((∀f ← flores(n)) 0 < fila(f) ≤ alto(n) ∧ 0 < columna(f) ≤ ancho(n)) ∧
            ((∀v ← vampiros(n)) 0 < fila(v) ≤ alto(n) ∧ 0 ≤ columna(v) ≤ ancho(n));
        invariante spawningOrdenado : (∀i ← [0..|spawning(n)| − 1)) (trd(spawning(n)_i) < trd(spawning(n)_{i+1})) ∨
            (trd(spawning(n)_i) == trd(spawning(n)_{i+1}) ∧ sgd(spawning(n)_i) ≤ sgd(spawning(n)_{i+1}));
        invariante necesitoMiEspacio : (∀i, j ← [0..|flores(n)|), i ≠ j)sgd(flores(n)_i) ≠ sgd(flores(n)_j);
        invariante vivosPeroNoTanto : vidaFloresOk(flores(n)) ∧ vidaVampirosOk(vampiros(n));
        invariante spawneanBien : (∀t ← spawning(n))sgd(t) ≥ 1 ∧ sgd(t) ≤ alto(n) ∧ trd(t) ≥ 0;
}

problema nuevoN (an : ℤ, al : ℤ, s : ℤ, spaw : [(Vampiro, ℤ, ℤ)]) = res : Nivel {
        requiere |spaw| > 0;
        requiere an > 0;
        requiere al > 0;
        requiere s ≥ 0;
        requiere (∀t ← spaw) sgd(t) ≥ 1 ∧ sgd(t) ≤ al ∧ trd(t) ≥ 0;
        requiere (∀i ← [0..|spaw| − 1))(trd(spaw_i) < trd(spaw_{i+1})) ∨ (trd(spaw_i) == trd(spaw_{i+1}) ∧
            sgd(spaw_i) ≤ sgd(spaw_{i+1}));
        asegura ancho(res) == an;
        asegura alto(res) == al;
        asegura turno(res) == 0;
        asegura spawning(res) == spaw;
        asegura flores(res) == [];
        asegura vampiros(res) == [];
}

problema anchoN (n : Nivel) = res : ℤ {
        asegura res == ancho(n);
}

problema altoN (n : Nivel) = res : ℤ {
        asegura res == alto(n);
}

problema turnoN (n : Nivel) = res : ℤ {
```

```
        asegura res == turno(n);
}

problema solesN (n : Nivel) = res : ℤ {
        asegura res == soles(n);
}

problema floresN (n : Nivel) = res : [(Flor, Posicion, Vida)] {
        asegura mismasFloresDeNivel(n, res);
}

problema vampirosN (n : Nivel) = res : [(Vampiro, Posicion, Vida)] {
        asegura mismos(res, vampiros(n));
}

problema spawningN (n : Nivel) = res : [(Vampiro, ℤ, ℤ)] {
        asegura res == spawning(n);
}

problema comprarSoles (n: Nivel, s : ℤ) {
        requiere s > 0;
        modifica n;
        asegura soles(n) == soles(pre(n)) + s;
        asegura ancho(n) == ancho(pre(n)) ∧ alto(n) == alto(pre(n)) ∧ turno(n) == turno(pre(n));
        asegura mismasFloresDeNivel(n, flores(pre(n)));
        asegura mismos(vampiros(pre(n)), vampiros(n));
        asegura spawning(pre(n)) == spawning(n);
}

problema obsesivoCompusilvo (n: Nivel) = res : Bool {
        asegura (∀x, y ← flores(n), x ≠ y)sucesor(x, y, n) ⟶ atacante(x) ≠ atacante(y);
        aux atacante (f: (Flor, Posicion, Vida)) : Bool = Atacar ∈ habilidades(prm(f));
        aux sucColumna (x, s: (Flor, Posicion, Vida), niv: Nivel) : Bool =
            fila(x) == fila(s) ∧ columna(x) < columna(s) ∧
            ¬(∃i ← flores(niv), i ≠ x, i ≠ s)(fila(x) == fila(i) ∧ columna(i) ∈ (columna(x)..columna(s)));
        aux sucesor (x, s: (Flor, Posicion, Vida), niv: Nivel) : Bool =
            sucColumna(x, s, niv) ∨ (fila(x) < fila(s) ∧ ¬hayIntermedio(x, s, niv));
        aux hayIntermedio (x, s: (Flor, Posicion, Vida), niv: Nivel) : Bool =
            (∃i ← flores(niv), i ≠ x, i ≠ s)sucColumna(x, i) ∨ sucColumna(i, s) ∨ fila(i) ∈ (fila(x)..fila(s));
}

problema agregarFlor (n: Nivel, f : Flor, p : Posicion) {
        requiere posicionOk : 0 < prm(p) ≤ alto(n) ∧ 0 < sgd(p) ≤ ancho(n) ∧ ¬(∃x ← flores(n))sgd(x) == sgd(f);
        requiere solesOk : soles(n) ≥ solesReq(f);
        modifica n;
        asegura ancho(n) == ancho(pre(n)) ∧ alto(n) == alto(pre(n)) ∧ turno(n) == turno(pre(n));
        asegura mismos(vampiros(pre(n)), vampiros(n));
        asegura spawning(n) == spawning(pre(n));
        asegura mismasFloresDeNivel(n, (f, p, vida(f)) : flores(pre(n)));
        asegura soles(n) == soles(pre(n)) − solesReq(f);
        aux solesReq (f: Flor) : ℤ = 2^|habilidades(f)|;
}
aux terminado (n: Nivel) : Bool = |vampiros(n)| == 0 ∨ (∃v ← vampiros(n))columna(v) == 0;

problema pasarTurno (n: Nivel) {
        requiere ¬terminado(n);
        modifica n;
        asegura ancho(n) == ancho(pre(n)) ∧ alto(n) == alto(pre(n));
        asegura turno(n) == turno(pre(n)) + 1;
        asegura soles(n) == (soles(pre(n)) + solesRecaudados(pre(n)) + 1);
        asegura mismasFloresDeNivel(n, [ (prm(f), sgd(f), trd(f) − ataqueFlor(f, pre(n))) | f ← flores(pre(n)),
            sobrevivienteFlor(f, pre(n)) ]);
        asegura mismos(vampiros(n), avanzan(pre(n)) + +quedan(pre(n)) + +retroceden(pre(n)) + +spawnean(pre(n)));
        aux sobrevivienteFlor (f: (Flor, Posicion, Vida), niv: Nivel) : Bool = trd(f) − ataqueFlor(f, niv) > 0 ∧
            ¬(∃v ← vampiros(niv))explosion(f, v);
```

```
        aux explosion (f: (Flor, Posicion, Vida), v: (Vampiro, Posicion. Vida)) : Bool =
            Explotar ∈ habilidades(prm(f)) ∧ sgd(f) == sgd(v) ;
        aux ataqueFlor (f:(Flor,Posicion,Vida), niv:Nivel) : ℤ =
            suma([cuantoPega(prm(v))|v ← vampiros(niv), sgd(f) == sgd(v)]) ;
        aux sobrevivienteVamp (v: (Vampiro, Posicion, Vida), niv: Nivel) : Bool = trd(v) − ataqueVamp(v, pre(n)) > 0 ;
        aux avanzan (niv: Nivel) : [(Vampiro, Posicion, Vida)] =
            [ (prm(v), (nuevaFila(v), columna(v) − 1), trd(v) − ataqueVamp(v, niv) | v ← vampiros(niv),
            sobrevivienteVamp(v, niv), ¬(∃f ← flores(niv))fila(f) == fila(v) ∧ columna(f) == columna(v) ] ;
        aux quedan (niv: Nivel) : [(Vampiro, Posicion, Vida)] =
            [ (prm(v), (nuevaFila(v), columna(v)), trd(v) − ataqueVamp(v, niv) | v ← vampiros(niv),
            sobrevivienteVamp(v, niv), (∃f ← flores(niv))¬explosion(f, v) ∧ fila(f) == fila(v) ∧
            columna(f) == columna(v) ] ;
        aux retroceden (niv: Nivel) : [(Vampiro, Posicion, Vida)] =
            [ (prm(v), (nuevaFila(v), columna(v) + 1), trd(v) − ataqueVamp(v, niv) | v ← vampiros(niv),
            sobrevivienteVamp(v, niv), (∃f ← flores(niv))explosion(f, v) ∧ fila(f) == fila(v) ∧
            columna(f) == columna(v) ] ;
        aux spawnean (niv: Nivel) : [(Vampiro, Posicion, Vida)] =
            [ (prm(v), (sgd(v), ancho(niv)), vida(prm(v))) | v ← spawning(niv), trd(v) == turno(niv) + 1 ] ;
        aux nuevaFila (v: (Vampiro, Posicion, Vida)) : ℤ =
            if clase(prm(v)) == Desviado ∧ fila(v) ≠ 1 then sgd(v) − 1 else sgd(v) ;
}

problema estaEnJaque (n: Nivel) = res : Vampiro {
        asegura res ∈ vampirosEnJaque(n) ;
        aux vampirosEnJaque (niv: Nivel) : [Vampiro] = [ prm(v) | v ← vampiros(niv),
            (∀w ← vampiros(niv))ataqueVamp(v, niv) ≥ ataqueVamp(w, niv) ] ;
}
```

5 Juego

```
tipo Juego {
        observador flores (j: Juego) : [Flor] ;
        observador vampiros (j: Juego) : [Vampiro] ;
        observador niveles (j: Juego) : [Nivel] ;
        invariante floresDistintas : (∀i, k ← [0.. |flores(j)|), i ≠ k)¬floresIguales(flores(j)_i, flores(j)_k) ;
        invariante vampirosDistintos : sinRepetidos(vampiros(j)) ;
        invariante nivelesConFloresValidas :
            (∀n ← niveles(j))(∀f ← flores(n))(∃x ← flores(j))floresIguales(prm(f), x) ;
        invariante nivelesConVampirosValidos : (∀n ← niveles(j))(∀v ← vampiros(n))prm(v) ∈ vampiros(j) ;
}

problema floresJ (j: Juego) = res : [Flor] {
        asegura mismasListasDeFlores(res, flores(j)) ;
}

problema vampirosJ (j: Juego) = res : [Vampiro] {
        asegura mismos(res, vampiros(j)) ;
}

problema nivelesJ (j: Juego) = res : [Nivel] {
        asegura mismosNivelesDeJuego(j, res) ;
        aux mismosNivelesDeJuego (jg: Juego, ns: [Nivel]) : Bool =
            |niveles(j)| == |ns| ∧ (∀k ← [0.. |niveles(jg)|))nivelesIguales(niveles(jg)_k, ns_k) ;
}

problema agregarNivelJ (j: Juego, n: Nivel, i: ℤ) {
        requiere 0 ≤ i ≤ |niveles(j)| ;
        requiere turno(n) == 0 ;
        requiere |flores(n)| == 0 ;
        requiere |vampiros(n)| == 0 ;
        modifica j ;
        asegura nivelesIguales(niveles(j)_i, n) ;
        asegura mismasListasDeFlores(flores(j), flores(pre(j))) ;
```

```
        asegura mismos(vampiros(j), vampiros(pre(j)));
        asegura ((∀x ← [0..i))nivelesIguales(niveles(j)_x, niveles(pre(j))_x) ∧
            ((∀y ← (i..|niveles(j)|))nivelesIguales(niveles(j)_y, niveles(pre(j))_{y−1});
}

problema estosSalenFacil (j: Juego) = res : [Nivel] {
        asegura mismasListasDeNiveles(masPlantas(masSoles(niveles(j))), res);
        aux masSoles (ns: [Nivel]) : [Nivel] = [ n | n ← ns, tieneMaxSoles(n, ns) ];
        aux masPlantas (ns: [Nivel]) : [Nivel] = [ n | n ← ns, tieneMaxPlantas(n, ns) ];
        aux tieneMaxSoles (n: Nivel, ns: [Nivel]) : Bool = (∀niv ← ns)soles(niv) ≤ soles(n);
        aux tieneMaxPlantas (n: Nivel, ns: [Nivel]) : Bool = (∀niv ← ns) |flores(niv)| ≤ |flores(n)|;
}

problema jugarNivel (j: Juego, n: Nivel, i: ℤ) {
        requiere 0 ≤ i < long(niveles(j));
        requiere esMismoAltoYAncho(niveles(j)_i, n);
        requiere spawningFuturo(niveles(j)_i, n);
        requiere turnoFuturo : turno(niveles(j)_i) ≤ turno(n));
        requiere nivelConFloresValidas : (∀f ← flores(n))(∃x ← flores(j))floresIguales(prm(f), x);
        requiere nivelConVampirosValidos : (∀v ← vampiros(n))prm(v) ∈ vampiros(j);
        modifica j;
        asegura esMismoAltoYAncho(niveles(j)_i, n);
        asegura mismos(vampiros(niveles(j)_i), vampiros(n));
        asegura mismasFloresDeNivel(niveles(j)_i, flores(n));
        asegura spawning(niveles(j)_i) == spawning(n);
        asegura soles(niveles(j)_i) == soles(n);
        asegura turno(niveles(j)_i) == turno(n);
        asegura mismasListasDeFlores(flores(j), flores(pre(j)));
        asegura mismos(vampiros(j), vampiros(pre(j));
        asegura |niveles(j)| == |niveles(pre(j))| ∧
            (∀k ← [0..|niveles(j)|), k ≠ i)nivelesIguales(niveles(j)_k, niveles(pre(j)_k));
        aux spawningFuturo (ni, nf : Nivel) : Bool =
            (∃k ← [0..|spawning(ni)|))(∀m ← [k..|spawning(ni)|))ni_m == nf_{m−k};
        aux esMismoAltoYAncho (ni, nf : Nivel) : Bool = (alto(ni) == alto(nf)) ∧ (ancho(ni) == ancho(nf));
}

problema altoCheat (j: Juego, i: ℤ) {
        requiere 0 ≤ i < |niveles(j)|;
        modifica j;
        asegura mismasListasDeFlores(flores(j), flores(pre(j)));
        asegura mismos(vampiros(j), vampiros(pre(j)));
        asegura mismos(listaVampirosNivel(j, i), reducirVidas(listaVampirosNivel(pre(j), i)));
        asegura ancho(niveles(j)_i) == ancho(niveles(pre(j))_i)) ∧ alto(niveles(j)_i) == alto(niveles(pre(j))_i)) ∧
            soles(niveles(j)_i) == soles(niveles(pre(j))_i)) ∧ spawning(niveles(j)_i) == spawning(niveles(pre(j))_i)) ∧
            mismasFloresDeNivel(niveles(j)_i, flores(niveles(pre(j))_i));
        asegura (∀k ← [0..|niveles(j)|), k ≠ i])nivelesIguales(niveles(j)_k, niveles(pre(j))_k);
        aux listaVampirosNivel (jg: Juego, i: ℤ) : [(Vampiro, Posicion, Vida)] = vampiros(niveles(jg)_i);
        aux reducirVidas (vs: [(Vampiro, Posicion, Vida)]) : [(Vampiro, Posicion, Vida)] =
            [ (prm(v), sgd(v), trd(v) div 2) | v ← vs ];
}

problema muyDeExactas (j: Juego) = res : Bool {
        asegura res == esFibo(soloNivelesGanados(j));
        aux soloNivelesGanados (j : Juego) : [ℤ] = [ i + 1 | i ← [0..|niveles(j)|), esGanado(niveles(j)_i) ];
        aux esGanado (n: Nivel) : Bool = terminado(n) ∧ long(vampiros(n)) == 0;
        aux esFibo (s : [ℤ]) : Bool = s_0 == 1 ∧ s_1 == 2 ∧ (∀i ← [1..|s| − 2])(s_{|s|−i} == s_{|s|−i−1} + s_{|s|−i−2});
}

problema nivelesSoleados (j: Juego) = res : [Nivel] {
        asegura mismasListasDeNiveles(res, listaNivelesNoTerminados(j));
        asegura ordenadoPorSoles(res);
        aux listaNivelesNoTerminados (j: Juego) : [Nivel] = [ n | n ← niveles(j), ¬terminado(n) ];
```

aux ordenadoPorSoles (xs: [Nivel]) : Bool = $((\forall i \leftarrow [0..|xs|-1))\ (soles(xs_i) + solesRecaudados(xs_i) + 1) \geq (soles(xs_{i+1}) + solesRecaudados(xs_{i+1}) + 1)) \vee ((\forall j \leftarrow [0..|xs|-1))\ (soles(xs_j) + solesRecaudados(xs_j) + 1) \leq (soles(xs_{j+1}) + solesRecaudados(xs_{j+1}) + 1))$;

}

6 Auxiliares

aux vidaFloresOk (fs: [(Flor, Posicion, Vida)]) : Bool = $(\forall f \leftarrow fs) trd(f) > 0 \wedge trd(f) \leq vida(prm(f))$;
aux vidaVampirosOk (fs: [(Vampiro, Posicion, Vida)]) : Bool = $(\forall f \leftarrow fs) trd(f) > 0 \wedge trd(f) \leq vida(prm(f))$;
aux fila (x: (T, Posicion, Vida)) : $\mathbb{Z}$ = $\mathsf{prm}(\mathsf{sgd}(x))$;
aux columna (x: (T, Posicion, Vida)) : $\mathbb{Z}$ = $\mathsf{sgd}(\mathsf{sgd}(x))$;
aux floresIguales (x, y: Flor) : Bool = $mismos(habilidades(x), habilidades(y))$;
aux mismos (a, b: [T]) : Bool = $|a| == |b| \wedge (\forall x \in a)\mathsf{cuenta}(x, a) == \mathsf{cuenta}(x, b)$;
aux mismasFloresDeNivel (n: Nivel, fs: [(Flor, Posicion, Vida)]) : Bool =
$|flores(n)| == |fs| \wedge ((\forall f \leftarrow flores(n)) cuentaFloresDeNivel(f, flores(n)) == cuentaFloresDeNivel(x, fs))$;
aux mismasListasDeFlores (xs, ys:[Flor]) : Bool =
$|xs| == |ys| \wedge (\forall x \leftarrow xs) cuentaFlores(x, xs) == cuentaFlores(y, ys)$;
aux mismasListasDeNiveles (xs,ys : [Nivel]) : Bool =
$|xs| == |ys| \wedge (\forall x \leftarrow xs) cuentaNiveles(x, xs) == cuentaNiveles(y, ys)$;
aux nivelesIguales (n, k: Nivel) : Bool = $ancho(n) == ancho(k) \wedge alto(n) == alto(k) \wedge soles(n) == soles(k) \wedge mismos(vampiros(n), vampiros(k)) \wedge spawning(n) == spawning(k) \wedge mismasFloresDeNivel(n, flores(k)) \wedge turno(n) == turno(k)$;
aux cuentaNiveles (n: Nivel, ns: [Nivel]) : $\mathbb{Z}$ = $|[m|m \leftarrow ns, nivelesIguales(n, m)]|$;
aux cuentaFlores (f: Flor, fs: [Flor]) : $\mathbb{Z}$ = $|[g|g \leftarrow fs, floresIguales(f, g)]|$;
aux cuentaFloresDeNivel (f: (Flor, Posicion, Vida), fs: [(Flor, Posicion, Vida)]) : $\mathbb{Z}$ =
$|[g|g \leftarrow fs, floresIguales(\mathsf{prm}(g), \mathsf{prm}(f)) \wedge \mathsf{sgd}(g) == \mathsf{sgd}(f) \wedge \mathsf{trd}(g) == \mathsf{trd}(f)]|$;
aux ataqueVamp (v: (Vampiro, Posicion, Vida), niv: Nivel) : $\mathbb{Z}$ =
$\mathsf{suma}([cuantoPega(\mathsf{prm}(f))|f \leftarrow flores(niv), fila(f) == fila(v), columna(f) \leq columna(v), \neg(\exists w \leftarrow vampiros(niv)) fila(v) == fila(w) \wedge columna(w) < columna(v)])$;
aux solesRecaudados (niv:Nivel) : $\mathbb{Z}$ = $\mathsf{suma}([1|f \leftarrow flores(niv), Generar \in habilidades(\mathsf{prm}(f))])$;