DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TPE - Flores vs Vampiros

Algoritmos y Estructuras de Datos I

Grupo: 07 - MOLOTOV

Integrante	LU	Correo electrónico
Bukovits, Nicolás Axel	546/14	nicobuk@gmail.com
Chizzoli, Lucas	782/14	chizzoli.lucas13@gmail.com
Frachtenberg Goldsmit, Kevin	247/14	kevinfra94@gmail.com
Garrett, Philip	318/14	garrett.phg@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autńoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Tipos

`tipo` Habilidad = Generar, Atacar, Explotar ;
`tipo` ClaseVampiro = Caminante, Desviado ;
`tipo` Posicion = ($\mathbb{Z}$, $\mathbb{Z}$) ;
`tipo` Vida = $\mathbb{Z}$;

2. Flor

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

`invariante` $sinRepetidos(habilidades(f))$;
`invariante lasHabilidadesDeterminanLaVidayElGolpe` : $vida(f) == \frac{100}{|habilidades(f)|+1} \ \wedge$
$if\ Atacar \in habilidades(f)\ Then\ cP == \frac{12}{|habilidades(f)|}\ Else\ cP == 0$;
}

`problema nuevaF` (v : $\mathbb{Z}$, cP : $\mathbb{Z}$, hs : [Habilidad]) = `res` : Flor {
`requiere habilidadesCoherentes` : $sinRepetidos(hs)$;
`requiere viveDelInvariante` : $v == \frac{100}{|hs|+1}$;
`requiere pegameSinVariar` : $if\ Atacar \in habilidades(f)\ Then\ cP == \frac{12}{|hs|}\ Else\ cP == 0$;
`asegura tendriaHabilidades` : $mismos(habilidades(res), hs)$;
`asegura tieneVida` : $vida(res) == v$;
`asegura siAtacaTePega` : $cuantoPega(res) == cP$;
}

`problema vidaF` (f: Flor) = `res` : $\mathbb{Z}$ {
`asegura` $res == vida(f)$;
}

`problema cuantoPegaF` (f: Flor) = `res` : $\mathbb{Z}$ {
`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) : $\mathbb{Z}$;
`observador cuantoPega` (v: Vampiro) : $\mathbb{Z}$;

`invariante vidaEnRango` : $vida(v) \geq 0 \wedge vida(v) \leq 100$;
`invariante pegaEnSerio` : $cuantoPega(v) > 0$;
}

```
problema nuevoV (cv : ClaseVampiro, v : ℤ, cP : ℤ) = res : Vampiro {
        requiere v ≥ 0 ∧ v ≤ 100;
        requiere cP > 0;
        asegura tieneVida : vida(res) == v;
        asegura estePega : cuantoPega(res) == cP;
        asegura esVampiro : clase(res) == cv;
}

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))(prm(sgd(f)) > 0) ∧ (prm(sgd(f)) ≤
            ancho(n))∧(sgd(sgd(f)) > 0)∧(sgd(sgd(f)) ≤ alto(n))∧(∀v ∈ vampiros(n))(prm(sgd(v)) ≥
            0) ∧ (prm(sgd(v)) ≤ ancho(n)) ∧ (sgd(sgd(v)) > 0) ∧ (sgd(sgd(v)) ≤ alto(n));
        invariante spawningOrdenado : (∀s ∈ spawning(n))(((∀i ∈ [0..|spawning(n)|−1))trd(spawning(n)_i) ≤
            trd(spawning(n)_{i+1})) ∧
            ((∀j ∈ [0..|spawning(n)|−1), trd(spawning(n)_j) == trd(spawning(n)_{j+1})sgd(spawning(n)_j) ≤
            sgd(spawning(n)_{j+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 esBienRazonable : an > 0∧al > 0∧(∀t ← spaw)sgd(t) ≥ 1 ∧ sgd(t) ≤ al ∧ trd(t) ≥
            0;
        asegura empezasEnTurnoCero : turno(res) == 0;
        asegura tieneTablero : ancho(res) == an ∧ alto(res) == al;
        asegura esDiaSoleado : soles(res) == s;
        asegura elTableroEstaVacio : vampiros(res) == [] ∧ flores(res) == [];
        asegura habraEnemigosOrdenados : mismos(spawning(res), spaw);
        asegura elGamerNoMuereSpawnea : (∀s ∈ spawning(res))(((∀i ∈ [0..|spawning(res)|−1))trd(spawning(res)_i)
            trd(spawning(res)_{i+1})) ∧
            ((∀j ∈ [0..|spawning(n)|−1), trd(spawning(res)_j) == trd(spawning(res)_{j+1})sgd(spawning(res)_j) ≤
            sgd(spawning(res)_{j+1}));
}

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 mismos(res, flores(n));
}

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));
        asegura alto(n) == alto(pre(n));
        asegura turno(n) == turno(pre(n));
        asegura mismos(flores(n), flores(pre(n)));
        asegura mismos(vampiros(n), vampiros(pre(n)));
        asegura spawning(n) == spawning(pre(n));
}

problema obsesivoCompusilvo (n: Nivel) = res : Bool {
        asegura estaOrdenada : ((∄i ∈ [0..|floresOrdenadas(flores(n))| − 1))
            Ataca ∈ habilidades(prm(floresOrdenadas(flores(n))_i)) ∧
            Ataca ∈ habilidades(prm(floresOrdenadas(flores(n))_{i+1}))) ∧
            ((∄j ∈ [0..|floresOrdenadas(flores(n))|−1)Ataca ∉ habilidades(prm(floresOrdenadas(flores(n))_i))∧
            Ataca ∉ habilidades(prm(floresOrdenadas(flores(n))_{i+1})));
}

problema agregarFlor (n: Nivel, f : Flor, p : Posicion) {
        requiere (∀f ∈ flores(n))sgd(f) ≠ p;
        requiere prm(p) ≤ ancho(n) ∧ prm(p) > 0;
        requiere sgd(p) ≤ alto(n) ∧ sgd(p) > 0;
        requiere soles(n) ≥ 2^{|habilidades(f)|};
        modifica n;
        asegura mismos(flores(n), cons((f, p, vida(f)), (flores(pre(n)))));
        asegura soles(n) == soles(pre(n)) − 2^{|habilidades(f)|};
        asegura ancho(n) == ancho(pre(n));
        asegura alto(n) == alto(pre(n));
        asegura turno(n) == turno(pre(n));
        asegura mismos(vampiros(n), vampiros(pre(n)));
        asegura spawning(n) == spawning(pre(n));
```

}

aux terminado (n: Nivel) : Bool $= (|vampiros(n)| == 0 \land |spawning(n)| == 0)$
$\lor ((\exists i[0..|vampiros(n)|]))prm(sgd(vampiros(n)_i)) == 0$;

problema pasarTurno (n: Nivel) {
requiere noEsElFin : $terminado(n) == False$;
modifica n ;
asegura $(\forall f \in flores(pre(n)), Explotar \in Habilidades(prm(f)) \land hayVampiros(pre(n), sgd(f)))$
$f \notin flores(n)$;
asegura $(\forall f \in flores(pre(n)), Explotar \notin Habilidades(prm(f)) \land trd(f) - danoVampiros(pre(n), sgd(f)) \leq 0)$
$f \notin flores(n)$;
asegura $(\forall f \in flores(pre(n)), Explotar \notin Habilidades(prm(f)) \land trd(f)) - danoVampiros(pre(n), sgd(f)) \geq 0)$
$(\exists f1 \in flores(n))\ prm(f1) == prm(f) \land sgd(f1) == sgd(f) \land trd(f1) == trd(f) - danoVampiros(pre(n), sgd(f))$;
asegura $(\forall f \in flores(pre(n)), Explotar \in Habilidades(prm(f)) \land \neg hayVampiros(pre(n), sgd(f)))$
$f \in flores(n)$;
asegura $sinRepetidos(flores(n))$;
asegura $(\forall v \in vampiros(pre(n)), trd(v) - danoFlores(pre(n), sgd(v)) \leq 0)$
$v \notin vampiros(n)$;
asegura $(\forall v \in vampiros(pre(n)), trd(f)) - danoFlores(pre(n), sgd(v)) > 0)$
$(\exists v1 \in vampiros(n))\ prm(v1) == prm(v) \land seMueve(v1, v, n, pre(n)) \land trd(v1) == trd(v) - danoVampiros(pre(n), sgd(f))$;
asegura $sinRepetidos(vampiros(n))$;
asegura noMeCambiesElNivel : $ancho(n) == ancho(pre(n)) \land alto(n) == alto(pre(n))$;
asegura graciasPachamama : $soles(n) == soles(pre(n)) + 1 + |[1|j \in flores(n), Generar \in habilidadesF(j_1)]| * (turno(pre(n)) + 1)$;
asegura soloPasaUnTurno : $turno(n) == turno(pre(n)) + 1$;
asegura $spawning(n) == [b|b \in spawning(pre(n)), trd(b) > turno(n)]$;
asegura $(\forall v \in spawning(pre(n)), v \notin spawning(n))(prm(v), (ancho(n), sgd(v)), vida(prm(v))) \in vampiros(n)$;
}

5. Juego

tipo Juego {
observador flores (j: Juego) : [Flor] ;
observador vampiros (j: Juego) : [Vampiro] ;
observador niveles (j: Juego) : [Nivel] ;
invariante floresDistintas : $(\forall i, k \leftarrow [0..|flores(j)|), i \neq k) \neg floresIguales(flores(j)_i, flores(j)_k)$;
invariante vampirosDistintos : $sinRepetidos(vampiros(j))$;
invariante nivelesConFloresValidas : $(\forall n \leftarrow niveles(j))(\forall f \leftarrow flores(n))prm(f) \in flores(j)$;
invariante nivelesConVampirosValidos : $(\forall n \leftarrow niveles(j))(\forall v \leftarrow vampiros(n))prm(v) \in vampiros(j)$;
}

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

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

problema nivelesJ (j: Juego) = res : [Nivel] {
asegura $res == niveles(j)$;

}

problema agregarNivelJ (j: Juego, n: Nivel, i: $\mathbb{Z}$) {
requiere $turno(n) == 0$;
requiere $|flores(n)| == 0$;
requiere $|vampiros(n)| == 0$;
requiere $i \geq 0 \land i \leq |niveles(j)|$;
modifica j;
asegura losAnterioresSonIguales : $(\forall k \leftarrow (0..i))nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
asegura esElNivelQueQuiero : $nivelesIguales(niveles(j)_i, n)$;
asegura losSiguientesSonIguales : $(\forall k \leftarrow (i..|niveles(j)|))$
$nivelesIguales(niveles(j)_k, niveles(pre(j))_{k-1})$;
asegura $mismos(flores(j), flores(pre(j)))$;
asegura $mismos(vampiros(j), vampiros(pre(j)))$;
}

problema estosSalenFacil (j: Juego) = res : [Nivel] {
asegura $mismos(res, maxFlor(nivelesConSoles(niveles(j))))$;
}

problema jugarNivel (j: Juego, n: Nivel, i: $\mathbb{Z}$) {
requiere rangoValido : $alto(n) == alto(niveles(j)_i) \land ancho(n) == ancho(niveles(j)_i)$;
requiere noTeQuedesSinTurno : $turno(niveles(j)_i) \leq turno(n)$;
requiere noGanasteNiPerdisteAun : $|spawning(n)| == |spawning(niveles(j)_i)|$;
modifica j;
asegura losAnterioresSonIguales2 : $(\forall k \leftarrow (0..i))nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
asegura esElNivelQueQuiero2 : $nivelesIguales(n, niveles(j)_i)$;
asegura losSiguientesSonIguales2 : $(\forall k \leftarrow (i..|niveles(j)|))$
$nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
asegura $mismos(flores(j), sinRepetidos(flores(pre(j)) + +florcitas(n)))$;
asegura $mismos(vampiros(j), sinRepetidos(vampiros(pre(j)) + +vampiritos(n)))$;
}

problema altoCheat (j: Juego, i: $\mathbb{Z}$) {
requiere $i \geq 0$;
requiere $i < |niveles(pre(j))|$;
modifica j;
asegura $(\forall v \in vampiros(niveles(pre(j)_i)))(\exists x \in vampiros(niveles(j)_i))prm(x) == prm(v) \land$
$sgd(x) == sgd(v) \land trd(x) == \frac{trd(v)}{2}$;
asegura $|vampiros(niveles(pre(j)))| == |vampiros(niveles(j))|$;
asegura $(\forall v \in vampiros(niveles(j)_i))(\exists x \in vampiros(niveles(pre(j))_i))prm(v) == prm(x) \land$
$sgd(v) == sgd(x) \land trd(v) == \frac{trd(x)}{2}$;
asegura $mismos(flores(j), flores(pre(j)))$;
asegura $mismos(vampiros(j), vampiros(pre(j)))$;
asegura losAnterioresSonIguales3 : $(\forall k \leftarrow (0..i))nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
asegura esElNivelQueQuiero3 : $ancho(niveles(pre(j))_i) == ancho(niveles(j)_i) \land alto(niveles(pre(j))_i) ==$
$alto(niveles(j)_i) \land$
$turno(niveles(pre(j))_i) == turno(niveles(j)_i) \land soles(niveles(pre(j))_i) == soles(niveles(j)_i) \land$
$mismos(flores(niveles(pre(j))_i), flores(niveles(j)_i)) \land$
$spawning(niveles(pre(j))_i) == spawning(niveles(j)_i)$;
asegura losSiguientesSonIguales3 : $(\forall k \leftarrow (i..|niveles(j)|))$
$nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
}

problema muyDeExactas (j: Juego) = res : Bool {
asegura sumaDeDos : $ifThenElse((|nivGsL(j)| \geq 3 \land nivGsL(j)_0 == 1 \land nivGsL(j)_1 == 2$
$\land (\forall k \in [2..|nivGsL(j)| - 1])\ nivGsL(j)_k == nivGsL(j)_{k-1} + nivGsL(j)_{k-2})$
$\lor (|nivGsL(j)| \geq 2 \land nivGsL(j)_0 == 1 \land nivGsL(j)_1 == 2) \lor$
$(|nivGsL(j)| \geq 1 \land nivGsL(j)_0 == 1), True, False)$;

}

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 `floresIguales` (x, y : Flor) : Bool $= mismos(habilidades(x), habilidades(y))$;

aux `cuenta` (x: T, a: [T]) : Int $= |[y | y \in a, y == x]|$;

aux `mismos` (a, b : [T]) : Bool $= (|a| == |b| \wedge (\forall c \in a) cuenta(c, a) == cuenta(c, b))$;

aux `sinRepetidos` (xs : [T]) : Bool $= (\forall i, j \leftarrow [0..|xs|),\ i \neq j) xs_i \neq xs_j$;

aux `noHayVampirosEnElMedio` (v : (Vampiro, Posicion, Vida), f : (Flor, Posicion, Vida), n : Nivel) : Bool $=$
$(\nexists k \in vampiros(n)) k_{2,2} \geq f_{2,2} \wedge k_{2,2} < v_{2,2}$;

aux `nivelGanado` (n:Nivel) : Bool $= vampiros(n) == [\,]$;

aux `nivGsL` (j: Juego) : $[\mathbb{Z}] = [i \mid i \in [1..|niveles(j)|), nivelGanado(niveles(j)_i)]$;

aux `nivelesIguales` (n1,n2: Nivel) : Bool $= ancho(n1) == ancho(n2) \wedge alto(n1) == alto(n2) \wedge turno(n1) == turno(n2) \wedge soles(n1) == soles(n2) \wedge mismos(flores(n1), flores(n2)) \wedge mismos(vampiros(n1), vampiros(n2)) \wedge spawning(n1) == spawning(n2)$;

aux `nivelesConSoles` (j : Juego) : [Nivel] $= [niveles(j)_i | i \in [0..|niveles(j)|), (\forall k \in [0..|niveles(j)|)) soles(niveles(j$
$soles(niveles(j)_k)]$;

aux `maxFlor` (n : [Nivel]) : [Nivel] $= [n_i | i \in [0..|n|), (\forall k \in [0..|n|)$
$|flores(n_i)| \geq |flores(n_k)|]$;

aux `floresOrdenadas` (lF : [Flor]) : [Flor] $= ((\forall t \in [0..|lF|-1))\ prm(lF_t) \leq prm(lF_{t+1})) \wedge ((\forall j \in [0..|lF|-1), prm(lF_j) == prm(lF_{j+1})) sgd(lF_j) \leq sgd(lF_{j+1}))$;

aux `florcitas` (n : Nivel) : [Flor] $= [prm(flores(n)_k) | k \in [0..|flores(n)|)]$;

aux `vampiritos` (n : Nivel) : [Vampiro] $= [prm(vampiros(n)_k) | k \in [0..|vampiros(n)|)]$;

aux `hayVampiros` (n : Nivel, p : Posicion) : Bool $= (\exists v \in vampiros(n)) sgd(v) == p$;

aux `danoVampiros` (n : Nivel, p : Posicion) : $\mathbb{Z} = \sum [cuantoPega(prm(v1)) | v1 \in vampiros(n), sgd(v1) == p]$;

aux `danoFlores` (n : Nivel, p : Posicion) : $\mathbb{Z} = \sum [cuantoPega(prm(f1) | f1 \in flores(n), sgd(sgd(f1)) == sgd(p) \wedge (\nexists v1 \in vampiros(n))(sgd(p) == sgd(sgd(v1)) \wedge prm(p) > prm(sgd(v1)))$;

aux `hayFlorDelante` (n : Nivel, p : Posicion) : Bool $= (\exists f \in flores(n)) prm(sgd(f)) - 1 == prm(p) \wedge sgd(sgd(f)) == sgd(p)$;

aux `delanteExplota` (n : Nivel, p : Posicion) : Bool $= (\exists f \in flores(n), Explotar \in habilidades(f)) prm(sgd(f)) - 1 == prm(p) \wedge sgd(sgd(f)) == sgd(p)$;

aux `seMueve` (v1, v : Vampiro, nuevoN, viejoN : Nivel) : Bool $= ifThenElse(hayFlorDelante(nuevoN, sgd(v)) \wedge delanteExplota(nuevoN, sgd(v)),\ prm(sgd(v)) + 1 == prm(sgd(v1)) \wedge sgd(sgd(v1)) == sgd(sgd(v)),$
$ifThenElse(hayFlorDelante(nuevoN, sgd(v)) \wedge \neg delanteExplota(nuevoN, sgd(v)),\ sgd(v1) == sgd(v),$
$ifThenElse(clase(prm(v1)) == Caminante,\ prm(sgd(v1)) == prm(sgd(v1)) - 1 \wedge sgd(sgd(v1)) == sgd(sgd(v)),$
$ifThenElse(sgd(sgd(v)) > 1 \wedge sgd(sgd(v)) \leq alto(nuevoN),\ prm(sgd(v1)) == prm(sgd(v)) - 1 \wedge sgd(sgd(v1)) == sgd(sgd(v)) + 1,$
$prm(sgd(v1)) == prm(sgd(v)) - 1 \wedge sgd(sgd(v1)) == sgd(sgd(v))))))$;