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` : $vida(f) == \frac{100}{|habilidades(f)|+1} \land$ $if\ Atacar \in habilidades(f)\ Then\ cuantoPega(f) == \frac{12}{|habilidades(f)|}\ Else\ cuantoPega(f) == 0$;
}

`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` : $mismos(habilidades(res), hs)$;
}

`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` : $vida(v) \geq 0 \land vida(v) \leq 100$;
    `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))posicionFlorValida(f, n) ) ∧ ( (∀v ∈
            vampiros(n))posicionVampiroValida(v, n));
        invariante spawningOrdenado : (((∀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);
}
```

```
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 mismosFloresNivel(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 mismos(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 mismosFloresNivel(flores(n), flores(pre(n)));
        asegura mismos(vampiros(n), vampiros(pre(n)));
        asegura mismos(spawning(n), spawning(pre(n)));
}

problema obsesivoCompusilvo (n: Nivel) = res : Bool {
        asegura hayPatron : ((∄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))_j))∧
            Ataca ∉ habilidades(prm(floresOrdenadas(flores(n))_{j+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 mismosFloresNivel(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 mismos(spawning(n), spawning(pre(n)));
}
aux terminado (n: Nivel) : Bool = (|vampiros(n)| == 0 ∧ |spawning(n)| == 0)
∨ ((∃i[0..|vampiros(n)|]))prm(sgd(vampiros(n)_i)) == 0;
problema pasarTurno (n: Nivel) {
        requiere noEsElFin : terminado(n) == False;
        modifica n;
        asegura mismosFloresNivel(flores(n), floresSobrevivientesNoExplotan(flores(pre(n)), pre(n))
            ++floresSobrevivientesExplotan(flores(pre(n)), pre(n)));
        asegura mismos(vampiros(n), vampirosSobrevivientes(vampiros(pre(n)), n, pre(n))++spawnean(n, pre(n)
        asegura noMeCambiesElNivel : ancho(n) == ancho(pre(n)) ∧ alto(n) == alto(pre(n));
        asegura graciasPachamama : soles(n) == soles(pre(n)) + 1 +
            |[1|j ∈ flores(pre(n)), Generar ∈ habilidades(prm(j))]|;
        asegura soloPasaUnTurno : turno(n) == turno(pre(n)) + 1;
        asegura mismos(spawning(n), [b|b ∈ spawning(pre(n)), trd(b) > turno(n)]);
}

problema estaEnJaque (n:Nivel) = res : Vampiro {
        requiere |vampiros(n)| > 0;
        asegura result ∈ [prm(v)|v ∈ vampiros(n), vaAPerderMasVida(v, n)];
}
```

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))estaIncluido(prm(f), flores(j)
        invariante nivelesConVampirosValidos : (∀n ← niveles(j))((∀v ← vampiros(n))prm(v) ∈
            vampiros(j) ∧ (∀v1 ← spawning(n))prm(v1) ∈ vampiros(j));
}

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

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

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

problema agregarNivelJ (j: Juego, n: Nivel, i: ℤ) {
        requiere turno(n) == 0;
        requiere |flores(n)| == 0;
        requiere |vampiros(n)| == 0;
        requiere (∀v1 ← spawning(n))prm(v1) ∈ vampiros(j);
        requiere i ≥ 0 ∧ i ≤ |niveles(j)|;
        modifica j;
        asegura losAnteriores SonIguales : (∀k ← (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 $mismosFlores(flores(j), flores(pre(j)))$;
asegura $mismos(vampiros(j), vampiros(pre(j)))$;
}

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

problema jugarNivel (j: Juego, n: Nivel, i: $\mathbb{Z}$) {
requiere rangoValido : $alto(n) == alto(niveles(j)_i) \wedge ancho(n) == ancho(niveles(j)_i)$;
requiere noTeQuedesSinTurno : $turno(niveles(j)_i) \leq turno(n)$;
requiere spawningCorrido : $mismos(spawning(n), [s|s \in spawning(niveles(j)_i), turno(n) < trd(s)]$;
requiere VampirosFloresValidas : $(\forall f \leftarrow flores(n))estaIncluido(prm(f), flores(j)) \wedge ((\forall v \leftarrow vampiros(n))prm(v) \in vampiros(j)$;
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 $mismosFlores(flores(j), flores(pre(j)))$;
asegura $mismos(vampiros(j), vampiros(pre(j)))$;
}

problema altoCheat (j: Juego, i: $\mathbb{Z}$) {
requiere $i \geq 0$;
requiere $i < |niveles(pre(j))|$;
modifica j ;
asegura $(\forall v \in vampiros(niveles(pre(j)_i)))(\exists x \in vampiros(niveles(j)_i))prm(x) == prm(v) \wedge sgd(x) == sgd(v) \wedge trd(x) == \frac{trd(v)}{2}$;
asegura $|vampiros(niveles(pre(j)))| == |vampiros(niveles(j))|$;
asegura $(\forall v \in vampiros(niveles(j)_i))(\exists x \in vampiros(niveles(pre(j))_i))prm(v) == prm(x) \wedge sgd(v) == sgd(x) \wedge trd(v) == \frac{trd(x)}{2}$;
asegura $(\forall v \in vampiros(n))cuenta(v, vampiros(n)) == cuentaConDobleDeVida(v, vampiros(pre(n))$;
asegura $mismosFlores(flores(j), flores(pre(j)))$;
asegura $mismos(vampiros(j), vampiros(pre(j)))$;
asegura losAnterioresSonIguales : $(\forall k \leftarrow (0..i))nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
asegura esElNivelQueQuiero : $ancho(niveles(pre(j))_i) == ancho(niveles(j)_i) \wedge alto(niveles(pre(j))_i) == alto(niveles(j)_i) \wedge turno(niveles(pre(j))_i) == turno(niveles(j)_i) \wedge soles(niveles(pre(j))_i) == soles(niveles(j)_i) \wedge mismosFloresNivel(flores(niveles(pre(j))_i), flores(niveles(j)_i)) \wedge mismos(spawning(niveles(pre(j))_i), spawning(niveles(j)_i))$;
asegura losSiguientesSonIguales : $(\forall k \leftarrow (i..|niveles(j)|))$
$nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
}

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

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 `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` (p : Posicion, f : (Flor, Posicion, Vida), n : Nivel) : Bool $= (\nexists k \in vampiros(n)) prm(sgd(k)) \geq prm(sgd(f)) \wedge prm(sgd(k)) < prm(p)$;

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

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

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

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

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

aux `estaEnLaPosicion` (i,j: $\mathbb{Z}$, p: Posicion) : Bool $= (i == sgd(p) \wedge j == prm(p))$;

aux `floresOrdenadas` (lF : [(Flor,Posicion,Vida)], n: Nivel) : [(Flor,Posicion,Vida)] $= [x | i \in [0..alto(n)], j \in [0..ancho(n)], x \in lF, estaEnLaPosicion(i, j, sgd(x))]$;

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

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

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` (v : (Vampiro,Posicion,Vida), nuevoN : Nivel) : ($\mathbb{Z}$,$\mathbb{Z}$) $= ifThenElse(hayFlorDelante(nuevoN, sgd(v)) \wedge delanteExplota(nuevoN, sgd(v)),\ (prm(sgd(v)) + 1, sgd(sgd(v))),$
$ifThenElse(hayFlorDelante(nuevoN, sgd(v)) \wedge \neg delanteExplota(nuevoN, sgd(v)),\ sgd(v),$
$ifThenElse(clase(prm(v1)) == Caminante,\ (prm(sgd(v)) - 1, sgd(sgd(v))),$
$ifThenElse(sgd(sgd(v)) > 1 \wedge sgd(sgd(v)) \leq alto(nuevoN),\ (prm(sgd(v)) - 1, sgd(sgd(v)) + 1),$
$(prm(sgd(v)) - 1, sgd(sgd(v)))))))$;

aux `posicionFlorValida` (f: Flor, n: Nivel) : Bool $= (prm(sgd(f)) > 0) \wedge (prm(sgd(f)) \leq ancho(n)) \wedge (sgd(sgd(f)) > 0) \wedge (sgd(sgd(f)) \leq alto(n))$;

aux `posicionVampiroValida` (v: Vampiro, n: Nivel) : Bool $= (prm(sgd(v)) \geq 0) \wedge (prm(sgd(v)) \leq ancho(n)) \wedge (sgd(sgd(v)) > 0) \wedge (sgd(sgd(v)) \leq alto(n))$;

aux `cuentaFlorN` (f: (Flor,Posicion,Vida), lista: [(Flor,Posicion,Vida)]) : $\mathbb{Z} = |[x | x \in lista, floresIguales(x, f) \wedge sgd(f) == sgd(x) \wedge trd(x) == trd(f)]|$;

aux `mismosFloresNivel` (f1,f2: [(Flor,Posicion,Vida)]) : Bool $= |f1| == |f2| \wedge (\forall f \in f1) cuentaFlorN(f, f1) == cuentaFlorN(f, f2)$;

aux `cuentaFlor` (f: Flor, lista: [Flor]) : $\mathbb{Z} = |[x | x \in lista, floresIguales(x, f)]|$;

aux `mismosFlores` (f1,f2: [Flor]) : Bool $= |f1| == |f2| \wedge (\forall f \in f1) cuentaFlor(f, f1) == cuentaFlor(f, f2)$;

aux `cuentaNivel` (n: Nivel, lista: [Nivel]) : $\mathbb{Z} = |[x | x \in lista, nivelesIguales(n, x)]|$;

aux `mismosNiveles` (n1,n2: [Nivel]) : Bool $= |n1| == |n2| \wedge (\forall n \in n1) cuentaNivel(n, n1) == cuentaNivel(n, n2)$;

aux `estaIncluido` (f: Flor, lista: [Flor]) : Bool $= (\exists f1 \in lista) floresIguales(f1, f)$;

`aux floresSobrevivientesNoExplotan` (fl: [(Flor,Posicion,Vida)] , n: Nivel) : [(Flor,Posicion,Vida)] $= [(prm(f), sgd(f), trd(f) - dañoVampiros(n, sgd(f)))|f \in fl, Explotar \notin Habilidades(prm(f)) \wedge trd(f)) - dañoVampiros(n, sgd(f)) > 0]$;

`aux floresSobrevivientesExplotan` (fl: [(Flor,Posicion,Vida)] , n: Nivel) : [(Flor,Posicion,Vida)] $= [(prm(f), sgd(f), trd(f))|f \in fl, Explotar \in Habilidades(prm(f)) \wedge \neg hayVampiros(n, sgd(f))]$;

`aux vampirosSobrevivientes` (vampiros:[(Vampiro,Posicion,Vida)],nuevoN,viejoN: Nivel) : [(Vampiro,Posicion,Vida)] $= [(prm(v), seMueve(v, nuevoN), trd(v) - dañoFlores(viejoN, sgd(v)))|v \in vampiros, trd(v) - dañoFlores(viejoN, sgd(v)) > 0]$;

`aux vaAPerderMasVida` (v: (Vampiro,Posicion,Vida), n: Nivel) : Bool $= (\forall v1 \in vampiros(n))$ $dañoFlores(n, sgd(v)) \geq dañoFlores(n, sgd(v1))$;

`aux cuentaConDobleDeVida` (v : (Vampiro,Posicion,Vida), lV : [(Vampiro,Posicion,Vida)]) : $\mathbb{Z} = |[1|x \in lV, prm(v) == prm(x) \wedge sgd(v) == sgd(x) \wedge 2 * trd(v) == trd(x)]|$;

`aux spawnean` (nuevoN, viejoN: Nivel) : [(Vampiro,Posicion,Vida)] $= [(prm(s), (ancho(n), sgd(s)), vida(prm(s)))|s$ $spawning(viejoN), s \notin spawning(nuevoN)]$;