



**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 Autónoma 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 :  $vida(f) == \frac{100}{|habilidades(f)|+1} \wedge$ 
     $if\ Atacar \in habilidades(f)\ Then\ cuantoPega(f) == \frac{12}{|habilidades(f)|}\ Else\ cuantoPega(f) ==$ 
    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 :  $vida(v) \geq 0 \wedge vida(v) \leq 100$ ;
  invariante pegaEnSerio :  $cuanPega(v) > 0$ ;
}

```

```

problema nuevoV (cv : ClaseVampiro, v :  $\mathbb{Z}$ , cP :  $\mathbb{Z}$ ) = res : Vampiro {
  requiere  $v \geq 0 \wedge v \leq 100$ ;
  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 :  $\mathbb{Z}$  {
  asegura  $res == vida(v)$ ;
}

problema cuantoPegaV (v : Vampiro) = res :  $\mathbb{Z}$  {
  asegura  $res == cuantoPega(v)$ ;
}

```

## 4. Nivel

```

tipo Nivel {
  observador ancho (n : Nivel) :  $\mathbb{Z}$ ;
  observador alto (n : Nivel) :  $\mathbb{Z}$ ;
  observador turno (n : Nivel) :  $\mathbb{Z}$ ;
  observador soles (n : Nivel) :  $\mathbb{Z}$ ;
  observador flores (n : Nivel) : [(Flor, Posicion, Vida)];
  observador vampiros (n : Nivel) : [(Vampiro, Posicion, Vida)];
  observador spawning (n : Nivel) : [(Vampiro,  $\mathbb{Z}$ ,  $\mathbb{Z}$ )];
  invariante valoresRazonables :  $ancho(n) > 0 \wedge alto(n) > 0 \wedge soles(n) \geq 0 \wedge turno(n) \geq 0$ ;
  invariante posicionesValidas :  $(\forall f \in flores(n)) posicionFlorValida(f, n) \wedge (\forall v \in$ 
     $vampiros(n)) posicionVampiroValida(v, n)$ ;
  invariante spawningOrdenado :  $((\forall i \in [0..|spawning(n)|-1]) trd(spawning(n)_i) \leq trd(spawning(n)_{i+1})) \wedge$ 
     $((\forall j \in [0..|spawning(n)|-1], trd(spawning(n)_j) == trd(spawning(n)_{j+1}) sgd(spawning(n)_j) \leq$ 
     $sgd(spawning(n)_{j+1})))$ ;
  invariante necesitoMiEspacio :  $(\forall i, j \leftarrow [0..|flores(n)|], i \neq j) sgd(flores(n)_i) \neq sgd(flores(n)_j)$ ;
  invariante vivosPeroNoTanto :  $vidaFloresOk(flores(n)) \wedge vidaVampirosOk(vampiros(n))$ ;
  invariante spawnenBien :  $(\forall t \leftarrow spawning(n)) sgd(t) \geq 1 \wedge sgd(t) \leq alto(n) \wedge trd(t) \geq 0$ ;
}

problema nuevoN (an :  $\mathbb{Z}$ , al :  $\mathbb{Z}$ , s :  $\mathbb{Z}$ , spaw : [(Vampiro,  $\mathbb{Z}$ ,  $\mathbb{Z}$ )]) = res : Nivel {
  requiere esBienRazonable :  $an > 0 \wedge al > 0 \wedge (\forall t \leftarrow spaw) sgd(t) \geq 1 \wedge sgd(t) \leq al \wedge trd(t) \geq$ 
     $0$ ;
  asegura empezasEnTurnoCero :  $turno(res) == 0$ ;
  asegura tieneTablero :  $ancho(res) == an \wedge alto(res) == al$ ;
  asegura esDiaSoleado :  $soles(res) == s$ ;
  asegura elTableroEstaVacio :  $vampiros(res) == [] \wedge flores(res) == []$ ;
  asegura habraEnemigosOrdenados :  $mismos(spawning(res), spaw)$ ;
}

```

```

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

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

problema turnoN (n : Nivel) = res :  $\mathbb{Z}$  {
    asegura  $res == turno(n)$ ;
}

problema solesN (n : Nivel) = res :  $\mathbb{Z}$  {
    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,  $\mathbb{Z}$ ,  $\mathbb{Z}$ )] {
    asegura  $mismos(res, spawning(n))$ ;
}

problema comprarSoles (n: Nivel, s :  $\mathbb{Z}$ ) {
    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 : (( $\exists i \in [0..|floresOrdenadas(flores(n))| - 1]$ )
        Ataca  $\in habilidades(prm(floresOrdenadas(flores(n)))_i)$ )  $\wedge$ 
        Ataca  $\in habilidades(prm(floresOrdenadas(flores(n)))_{i+1})$ )  $\wedge$ 
        (( $\exists j \in [0..|floresOrdenadas(flores(n))| - 1]$ ) Ataca  $\notin habilidades(prm(floresOrdenadas(flores(n)))_j)$ )  $\wedge$ 
        Ataca  $\notin habilidades(prm(floresOrdenadas(flores(n)))_{j+1})$ );
}

problema agregarFlor (n: Nivel, f : Flor, p : Posicion) {
    requiere ( $\forall f \in flores(n)$ )  $sgd(f) \neq p$ ;
    requiere  $prm(p) \leq ancho(n) \wedge prm(p) > 0$ ;
    requiere  $sgd(p) \leq alto(n) \wedge sgd(p) > 0$ ;
    requiere  $soles(n) \geq 2^{|habilidades(f)|}$ ;
    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)|])prn(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))
        + +spawnan(n, pre(n)));
    asegura noMeCambiesElNivel : ancho(n) == ancho(pre(n)) ∧ alto(n) == alto(pre(n));
    asegura graciasPachamama : soles(n) == soles(pre(n)) + solesGenerados(pre(n));
    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 ∈ [prn(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(prn(f), flores(j));
    invariante nivelesConVampirosValidos : (∀n ← niveles(j)) ((∀v ← vampiros(n)) prn(v) ∈
        vampiros(j) ∧ (∀v1 ← spawning(n)) prn(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)) prn(v1) ∈ vampiros(j);
    requiere i ≥ 0 ∧ i ≤ |niveles(j)|;
    modifica j;
    asegura losAnterioresSonIguales : (∀k ← (0..i)) nivelesIguales(niveles(j)k, niveles(pre(j))k);
    asegura esElNivelQueQuiero : nivelesIguales(niveles(j)i, n);
}

```

```

    asegura losSiguietesSonIguales : ( $\forall k \leftarrow (i..|niveles(j)|)$ )
      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 losSiguietesSonIguales2 : ( $\forall k \leftarrow (i..|niveles(j)|)$ )
    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 losSiguietesSonIguales : ( $\forall k \leftarrow (i..|niveles(j)|)$ )
    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);
}

problema nivelesSoleados (j: Juego) = res : [Nivel] {
  asegura noTerminaron : mismosNiveles(result, [n | n  $\in$  niveles(j), terminado(n) == false]);
  asegura estanOrdenados : ( $\forall i \in [0..|result| - 1]$ ) solesGenerados(resulti)  $\leq$  solesGenerados(resulti+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) :  $\mathbb{Z}$  =  $[i | i \in [1..niveles(j)], nivelGanado(niveles(j)_i)]$ ;
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) :  $\mathbb{Z}$  =  $\sum [cuantoPega(prm(f1)) | f1 \in flores(n), sgd(sgd(f1)) ==$ 
 $sgd(p) \wedge prm(sgd(f1)) \leq prm(p) \wedge noHayVampirosEnElMedio(p, f1, n)]$ ;
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 ∈ fl, Explotar ∉ Habilidades(prm(f)) ∧
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 ∈ fl, Explotar ∈ Habilidades(prm(f)) ∧ ¬hayVampiros(n,sgd(f))];
    aux vampirosSobrevivientes (vampiros:[(Vampiro,Posicion,Vida)],nuevoN,viejoN: Nivel) : [(Vam-
piro,Posicion,Vida)] = [(prm(v),seMueve(v,nuevoN),trd(v)-dañoFlores(viejoN,sgd(v)))|v ∈ vampiros,trd(v)-
dañoFlores(viejoN,sgd(v)) > 0];
    aux vaAPerderMasVida (v: (Vampiro,Posicion,Vida), n: Nivel) : Bool = (∀v1 ∈ vampiros(n))
cuantoDañoRecibe(v,n) ≥ cuantoDañoRecibe(v1,n);
    aux cuentaConDobleDeVida (v : (Vampiro,Posicion,Vida), lV : [(Vampiro,Posicion,Vida)]) : ℤ =
|[1|x ∈ lV,prm(v) == prm(x) ∧ sgd(v) == sgd(x) ∧ 2 * trd(v) == trd(x)]|;
    aux spawnear (nuevoN, viejoN: Nivel) : [(Vampiro,Posicion,Vida)] = [(prm(s), (ancho(n),sgd(s)), vida(prm(s)))|
s ∈ spawning(viejoN), s ∉ spawning(nuevoN)];
    aux solesGenerados (n: Nivel) : ℤ = 1 + |[1|j ∈ flores(n), Generar ∈ habilidades(prm(j))]|;
    aux cuantoDañoRecibe (v: (Vampiro,Posicion,Vida), n: Nivel) : ℤ = if (dañoFlores(n,sgd(v)) >
trd(v)) then trd(v) else dañoFlores(n,sgd(v));

```