UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación
Algoritmos y Estructura de Datos I
Primer cuatrimestre de 2015
6 de Mayo de 2015

# TPI - Flores vs Vampiros

## 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` :
        $vidaDeFlorValida(vida(f), habilidades(f)) \land cuantoPegaFlorValido(cuantoPega(f), habilidades(f))$ ;
}

`problema nuevaF` (v : $\mathbb{Z}$, cP : $\mathbb{Z}$, hs : [Habilidad]) = `this` : Flor {
    `requiere` $sinRepetidos(hs)$ ;
    `requiere` $vidaDeFlorValida(v, hs)$ ;
    `requiere` $cuantoPegaFlorValido(cP, hs)$ ;
    `asegura` $mismos(hs, habilidades(this))$ ;
}

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

`problema cuantoPegaF` (this : Flor) = `res` : $\mathbb{Z}$ {
    `asegura` $res == cuantoPega(this)$ ;
}

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

## 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 : $\mathbb{Z}$, cP: $\mathbb{Z}$) = `this` : Vampiro {
    `requiere` $v \geq 0 \land v \leq 100$ ;
    `requiere` $cP > 0$ ;
    `asegura` $clase(this) == cv$ ;

```
        asegura vida(this) == v ;
        asegura cuantoPega(this) == cP ;
}

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

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

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

# 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 : vampirosEnCuadricula(n) ∧ floresEnCuadricula(n) ;
        invariante spawningOrdenado :
            duplasOrdenadas([(turnoSpawn(spawn), filaSpawn(spawn))|spawn ← spawning(n)]) ;
        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)]) = this : Nivel  {
        requiere an > 0 ;
        requiere al > 0 ;
        requiere s ≥ 0 ;
        requiere filasSpawnValidas : (∀ s ← spaw) filaSpawn(s) ≥ 1 ∧ filaSpawn ≤ al ;
        requiere turnosSpawnValidos : (∀ s ← spaw) turnoSpawn(s) ≥ 0 ;
        requiere spawningOrdenado : duplasOrdenadas([(turnoSpawn(spawn), filaSpawn(spawn))|spawn ← spaw]) ;
        asegura ancho(this) == an ;
        asegura alto(this) == al ;
        asegura soles(this) == s ;
        asegura mismos(spawning(this), spaw) ;
        asegura |vampiros(this)| == 0 ;
        asegura |flores(this)| == 0 ;
        asegura turno(n) == 0 ;
}

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

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

problema turnoN (this: Nivel) = res : ℤ  {
        asegura res == turno(this) ;
}

problema solesN (this: Nivel) = res : ℤ  {
```

```
        asegura res == soles(this);
}

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

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

problema spawningN (this : Nivel) = res : [(Vampiro, ℤ, ℤ)] {
        asegura mismos(res, spawning(this));
        asegura duplaOrdenada([(turnoSpawn(spawn), filaSpawn(spawn))|spawn ← res]);
}

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

problema obsesivoCompulsivo (this : Nivel) = res : Bool {
        asegura res ↔ ((∀ i ← [0..|flores(this)|), j ← [0..|flores(this)|), i ≠ j∧posicionMayorInmediata(i, j, flores(this)))
            Atacar ∈ habilidades(prm(flores(this)_i)) == ¬(Atacar ∈ habilidades(prm(flores(this)_j))));
}

problema agregarFlor (this : Nivel, f : Flor, p : Posicion) {
        requiere sinColisiones : (∀ flor ← flores(this)) sgd(flor) ≠ p;
        requiere solesSuficientes : soles(this) ≥ 2^|habilidades(f)|;
        modifica this;
        asegura ancho(this) == ancho(pre(this));
        asegura alto(this) == alto(pre(this));
        asegura turno(this) == turno(pre(this));
        asegura soles(this) == soles(pre(this)) − 2^|habilidades(f)|;
        asegura mismasFloresDeNivel(flores(this), (f, p, vida(f)) : flores(pre(this)));
        asegura mismos(vampiros(this), vampiros(pre(this)));
        asegura mismos(spawning(this), spawning(pre(this)));
}

problema pasarTurno (this : Nivel) {
        requiere ¬terminado(this);
        modifica this;
        asegura ancho(this) == ancho(pre(this));
        asegura alto(this) == alto(pre(this));
        asegura turno(this) == turno(pre(this)) + 1;
        asegura soles(this) == soles(pre(this)) +
            |[flor|flor ← flores(pre(this)), Generar ∈ habilidades(prm(flor))]| + 1;
        asegura mismasFloresDeNivel(flores(this), floresDaniadas(pre(this)));
        asegura mismos(vampiros(this), vampirosMovidos(vampirosDaniados(pre(this)), pre(this))++nuevosVampiros(pre(th
        asegura mismos(spawning(this), proximosVampiros(pre(this)));
}
```

# 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 :
            (∀ nivel ← niveles(j)) ((∀ flor ← flores(nivel)) florEnLista(flor, flores(j))) ;
        invariante nivelesConVampirosValidos : (∀ nivel ← niveles(j)) ((∀ vampiro ← vampiros(nivel)) vampiro ∈
            vampiros(j)) ;
}

problema nuevoJ (fs:[Flor], vs:[Vampiro]) = this : Juego  {
        asegura mismasFlores(flores(this), fs) ;
        asegura mismos(vampiros(this), vs) ;
        asegura |niveles(this)| == 0 ;
}

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

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

problema nivelesJ (this: Juego) = res : [Nivel]  {
        asegura |res| == |niveles(this)| ;
        asegura (∀ i ← [0..|niveles(this)|)) nivelesIguales(res_i, niveles(this)_i) ;
}

problema agregarNivelJ (this: Juego , n : Nivel , i : ℤ)  {
        requiere 0 ≤ i ≤ |niveles(this)| ;
        requiere |flores(n)| == 0 ;
        requiere |vampiros(n)| == 0 ;
        requiere turno(n) == 0 ;
        modifica this ;
        asegura mismasFlores(flores(this), flores(pre(this))) ;
        asegura mismos(vampiros(this), vampiros(pre(this))) ;
        asegura |niveles(this)| == |niveles(pre(this))| + 1 ;
        asegura (∀ k ← [0..i)) nivelesIguales(niveles(this)_k, niveles(pre(this))_k) ;
        asegura nivelesIguales(n, niveles(this)_i) ;
        asegura (∀ k ← [i + 1..|niveles(pre(this)) + 1|)) nivelesIguales(niveles(this)_k, niveles(pre(this))_{k−1}) ;
}

problema estosSalenFacil (this: Juego) = res : [Nivel]  {
        asegura mismosNiveles(res, nivelesFaciles(this)) ;
}

problema jugarNivel (this : Juego, n : Nivel, i : ℤ)  {
        requiere 0 ≤ i < |niveles(this)| ;
        requiere ancho(n) == ancho(niveles(this)_i) ;
        requiere alto(n) == alto(niveles(this)_i) ;
        requiere turno(n) > turno(niveles(this)_i) ;
        requiere mismos(spawning(n), [spawn|spawn ← spawning(niveles(this)_i), turnoSpawn(spawn) > turno(n)]) ;
        modifica this ;
        asegura mismasFlores(flores(this), flores(pre(this))) ;
        asegura mismos(vampiros(this), vampiros(pre(this))) ;
        asegura |niveles(this)| == |niveles(pre(this))| ;
        asegura (∀ k ← [0..|niveles(pre(this))|), k ≠ i) nivelesIguales(niveles(this)_k, niveles(pre(this))_k) ;
        asegura ancho(niveles(this)_i) == ancho(n) ;
        asegura alto(niveles(this)_i) == alto(n) ;
        asegura turno(niveles(this)_i) == turno(n) ;
        asegura soles(niveles(this)_i) == soles(n) ;
        asegura mismasFloresDeNivel(flores(niveles(this)_i), flores(n)) ;
        asegura mismos(vampiros(niveles(this)_i), vampiros(n)) ;
        asegura mismos(spawning(niveles(this)_i), spawning(n)) ;
}
```

problema altoCheat (this : Juego, i : $\mathbb{Z}$) {
requiere $0 \leq i < |niveles(this)|$ ;
modifica $this$ ;
asegura $mismasFlores(flores(this), flores(\text{pre}(this)))$ ;
asegura $mismos(vampiros(this), vampiros(\text{pre}(this)))$ ;
asegura $|niveles(this)| == |niveles(\text{pre}(this))|$ ;
asegura $(\forall\ k\ \leftarrow\ [0..|niveles(\text{pre}(this))|), k \neq i)\ nivelesIguales(niveles(this)_k, niveles(\text{pre}(this))_k)$ ;
asegura $ancho(niveles(this)_i) == ancho(niveles(\text{pre}(this))_i)$ ;
asegura $alto(niveles(this)_i) == alto(niveles(\text{pre}(this))_i)$ ;
asegura $turno(niveles(this)_i) == turno(niveles(\text{pre}(this))_i)$ ;
asegura $soles(niveles(this)_i) == soles(niveles(\text{pre}(this))_i)$ ;
asegura $mismasFloresDeNivel(flores(niveles(this)_i), flores(niveles(\text{pre}(this))_i))$ ;
asegura $mismos(vampiros(niveles(this)_i), vampirosMitadVida(vampiros(niveles(\text{pre}(this))_i))$ ;
asegura $mismos(spawning(niveles(this)_i), spawning(niveles(\text{pre}(this))_i))$ ;
}

problema muyDeExactas (this : Juego) = res : Bool {
requiere $|nivelesGanados(this)| > 0$ ;
asegura $res \leftrightarrow ($
$(|nivelesGanados(this)| \geq 1 \longrightarrow nivelesGanados(this)_0 == 1) \wedge$
$(|nivelesGanados(this)| \geq 2 \longrightarrow nivelesGanados(this)_1 == 2) \wedge$
$(|nivelesGanados(this)| > 2 \longrightarrow (\forall\ i\ \leftarrow\ [2..|nivelesGanados(this)|))\ nivelesGanados(this)_i == nivelesGanados(this$
$nivelesGanados(this)_{i-2}))$ ;
}

# 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) : Bool $= mismos(habilidades(x), habilidades(y))$ ;
aux vidaDeFlorValida (vida : $\mathbb{Z}$, habilidades : [Habilidad]) : Bool $= (vida == (100\ div\ |habilidades| + 1))$ ;
aux cuantoPegaFlorValido (cuantoPega : $\mathbb{Z}$, habilidades : [Habilidad]) : Bool $= (Atacar \in habilidades \wedge cuantoPega ==$
$(12\ div\ |habilidades|)) \vee (Atacar \notin habilidades \wedge cuantoPega == 0)$ ;
aux fila (pos: Posicion) : $\mathbb{Z} = sgd(pos)$ ;
aux columna (pos: Posicion) : $\mathbb{Z} = prm(pos)$ ;
aux posicionEnNivel (pos : Posicion, n : Nivel) : Bool $= columna(pos) \geq 1 \wedge columna(pos) \leq ancho(n)$
$\wedge\ fila(pos) \geq 1 \wedge fila(pos) \leq alto(n)$ ;
aux posicionVampiroEnNivel (posVampiro : Posicion, n : Nivel) : Bool $= columna(posVampiro) \geq 0$
$\wedge\ columna(posVampiro) \leq ancho(n) \wedge fila(posVampiro) \geq 1 \wedge fila(posVampiro) \leq alto(n)$ ;
aux vampirosEnCuadricula (n : Nivel) : Bool $= (\forall\ v\ \leftarrow\ vampiros(n))\ posicionVampiroEnNivel(sgd(v), n)$ ;
aux floresEnCuadricula (n : Nivel) : Bool $= (\forall\ f\ \leftarrow\ flores(n))\ posicionEnNivel(sgd(f), n)$ ;
aux turnoSpawn (spawn : (Vampiro, $\mathbb{Z}$, $\mathbb{Z}$)) : $\mathbb{Z} = trc(spawn)$ ;
aux filaSpawn (spawn : (Vampiro, $\mathbb{Z}$, $\mathbb{Z}$)) : $\mathbb{Z} = sgd(spawn)$ ;
aux ordenada (lista : [$\mathbb{Z}$]) : Bool $= |lista| \leq 1 \vee (\forall\ i\ \leftarrow\ [1..|lista|))\ lista_{i-1} \leq lista_i$ ;
aux duplasOrdenadas (listDuplas : [($\mathbb{Z}$, $\mathbb{Z}$)]) : Bool $= |listaDuplas| \leq 1 \vee (ordenada([prm(dupla)|dupla\ \leftarrow\ listaDuplas]) \wedge$
$(\forall\ i\ \leftarrow\ [1..|listaDuplas|), prm(listaDupla_{i-1}) == prm(listaDupla_i))\ sgd(listaDupla_{i-1}) \leq sgd(listaDupla_i))$ ;
aux cuentaFloresDeNivel (flor : (Flor, Posicion, Vida), flores: [(Flor, Posicion, Vida)]) : $\mathbb{Z} = |[f|f\ \leftarrow\ flores,$
$floresIguales(prm(flor), prm(f)) \wedge sgd(flor) == sgd(f) \wedge trc(flor) == trc(f)]|$ ;
aux mismasFloresDeNivel (floresA : [(Flor, Posicion, Vida)], floresB : [(Flor, Posicion, Vida)]) : Bool $= |floresA| ==$
$|floresB| \wedge (\forall\ flor\ \leftarrow\ floresA)\ cuentaFloresDeNivel(flor, floresA) == cuentaFloresDeNivel(flor, floresB)$ ;
aux posicionMayor (a : Posicion, b : Posicion) : Bool $= fila(a) > fila(b) \vee (fila(a) == fila(b) \wedge columna(a) >$
$columna(b))$ ;
aux posicionMenor (a : Posicion, b : Posicion) : Bool $= a \neq b \wedge \neg posicionMayor(a, b)$ ;
aux posicionMayorInmediata (a : $\mathbb{Z}$, b : $\mathbb{Z}$, lista : [Posicion]) : Bool $= posicionMayor(lista_a, lista_b) \wedge$
$(\forall\ i\ \leftarrow\ [0..|lista|), a \neq i \wedge b \neq i)\ \neg(posicionMayor(lista_i, lista_b) \wedge posicionMenor(lista_i, lista_a))$ ;
aux vampirosEnCasa (vampiros : [(Vampiro, Posicion, Vida)]) : $\mathbb{Z} =$
$|[vampiro|vampiro\ \leftarrow\ vampiros, columna(sgd(vampiro)) == 0]|$ ;
aux florExploto (flor : (Flor, Posicion, Vida), vampiros : [(Vampiro, Posicion, Vida)]) : Bool $=$
$Explotar \in habilidades(prm(flor)) \wedge (\exists\ v\ \leftarrow\ vampiros)\ sgc(v) == sgd(flor)$ ;
aux daniarFlor (flor : (Flor, Posicion, Vida), vampiros : [(Vampiro, Posicion, Vida)]) : (Flor, Posicion, Vida) $=$
$(prm(flor), sgd(flor), trc(flor) - \sum[cuantoPega(prm(v))|v\ \leftarrow\ vampiros, sgd(v) == sgd(flor)])$ ;

aux florMuerta (flor : (Flor, Posicion, Vida), vampiros : [(Vampiro, Posicion, Vida)]) : Bool =
$florExploto(flor, vampiros) \vee trc(daniarFlor(flor, vampiros)) \leq 0$ ;
aux floresDaniadas (n : Nivel) : [(Flor, Posicion, Vida)] = $[daniarFlor(flor, vampiros(n))|$
$flor \leftarrow flores(n), \neg florMuerta(flor, vampiros(n))]$ ;
aux enMira (flor : (Flor, Posicion, Vida), vampiro : (Vampiro, Posicion, Vida)) : Bool =
$fila(sgd(flor)) == fila(sgd(vampiro)) \wedge columna(sgd(flor)) \leq columna(sgd(vampiro))$ ;
aux intercepta (flor : (Flor, Posicion, Vida), vampiro : (Vampiro, Posicion, Vida), vampiros : [(Vampiro, Posicion, Vida)]) : Bool = $(\exists\ v \leftarrow vampiros)\ fila(sgd(v)) == fila(sgd(flor)) \wedge columna(sgd(flor)) \leq columna(sgd(v)) < columna(sgd(vampiro))$ ;
aux daniarVampiro (vampiro : (Vampiro, Posicion, Vida), flores : [(Flor, Posicion, Vida)], vampiros : [(Vampiro, Posicion, Vida)]) : (Vampiro, Posicion, Vida) = $(prm(vampiro), sgd(vampiro), trc(vampiro) -$
$\sum[cuantoPega(prm(f))|f \leftarrow flores(n), enMira(f, vampiro) \wedge \neg intercepta(f, vampiro, vampiros)])$ ;
aux vampiroMuerto (vampiro : (Vampiro, Posicion, Vida), flores : [(Flor, Posicion, Vida)], vampiros : [(Vampiro, Posicion, Vida)]) : Bool = $trc(daniarVampiro(vampiro, flores, vampiros)) \leq 0$ ;
aux vampirosDaniados (n : Nivel) : [(Vampiro, Posicion, Vida)] = $[daniarVampiro(vampiro, flores(n), vampiros(n))|$
$vampiro \leftarrow vampiros(n), \neg vampiroMuerto(vampiro, flores(n), vampiros(n))]$ ;
aux florSobreviviente (pos : Posicion, flores: [(Flor, Posicion, Vida)], vampiros : [(Vampiro, Posicion, Vida)]) : Bool
$= (\exists\ flor \leftarrow flores)\ sgd(flor) == pos \wedge \neg florMuerta(flor, vampiros)$ ;
aux florExplotada (pos : Posicion, flores : [(Flor, Posicion, Vida)], vampiros : [(Vampiro, Posicion, Vida)]) : Bool =
$(\exists\ flor \leftarrow flores)\ sgd(flor) == pos \wedge florExploto(flor, vampiros)$ ;
aux intentarRetroceder (pos : Posicion, anchoNivel : $\mathbb{Z}$) : Posicion =
if $columna(pos) < anchoNivel$
then $(columna(pos) + 1, fila(pos))$
else $pos$ ;
aux intentarDesvio (vampiro : (Vampiro, Posicion, Vida)) : Posicion =
if $clase(prm(vampiro)) == Desviado \wedge fila(sgd(vampiro)) > 1$
then $(columna(sgd(vampiro)) - 1, fila(sgd(vampiro)) - 1)$
else $(columna(sgd(vampiro)) - 1, fila(sgd(vampiro)))$ ;
aux intentarAvanzar (vampiro : (Vampiro, Posicion, Vida), n : Nivel) : Posicion =
if $florExplotada(sgd(vampiro), flores(n), vampiros(n))$
then $intentarRetroceder(sgd(vampiro), ancho(n))$
else $intentarDesvio(vampiro)$ ;
aux mover (vampiro : (Vampiro, Posicion, Vida), n : Nivel) : (Vampiro, Posicion, Vida) =
$(prm(vampiro)$,
if $florSobreviviente(sgd(vampiro), flores(n), vampiros(n))$
then $sgd(vampiro)$
else $intentarAvanzar(vampiro, n)$,
$trc(vampiro))$ ;
aux vampirosMovidos (vampiros : [(Vampiro, Posicion, Vida)], n : Nivel) : [(Vampiro, Posicion, Vida)] = $[mover(vampiro, n)|$
$vampiro \leftarrow vampiros]$ ;
aux nuevosVampiros (n : Nivel) : [(Vampiro, Posicion, Vida)] = $[(prm(spawn), (ancho(n), filaSpawn(spawn)),$
$vida(prm(spawn)))|spawn \leftarrow spawning(n), turnoSpawn(spawn) == turno(n) + 1]$ ;
aux proximosVampiros (n : Nivel) : [(Vampiro, $\mathbb{Z}$, $\mathbb{Z}$)] = $[spawn|spawn \leftarrow spawning(n), turnoSpawn(spawn) > turno(n) + 1]$ ;
aux florEnLista (flor : Flor, listaFlores : [Flor]) : Bool = $(\exists\ f \leftarrow listaFlores)\ floresIguales(flor, f)$ ;
aux cuentaFlores (flor : Flor, flores : [Flor]) : $\mathbb{Z}$ = $|[f|f \leftarrow flores, floresIguales(f, flor)]|$ ;
aux mismasFlores (floresA : [Flor], floresB : [Flor]) : Bool = $|floresA| == |floresB| \wedge$
$(\forall\ flor \leftarrow floresA)\ cuentaFlores(flor, floresA) == cuentaFlores(flor, floresB)$ ;
aux nivelesIguales (nA : Nivel, nB : Nivel) : Bool =
$ancho(nA) == ancho(nB) \wedge$
$alto(nA) == alto(nB) \wedge$
$turno(nA) == turno(nB) \wedge$
$soles(nA) == soles(nB) \wedge$
$mismasFloresDeNivel(flores(nA), flores(nB)) \wedge$
$mismos(vampiros(nA), vampiros(nB)) \wedge$
$mismos(spawning(nA), spawning(nB))$ ;
aux masFacil (nA : Nivel, nB : Nivel) : Bool = $soles(nA) > soles(nB) \vee (soles(nA) == soles(nB) \wedge |flores(nA)| > |flores(nb)|)$ ;
aux nivelesFaciles (j: Juego) : [Nivel] =
$[\, n \,|\, n \leftarrow niveles(j), \neg((\exists\ candidato \leftarrow niveles(j))\ (masFacil(candidato, nivel)))\,]$ ;
aux mismosNiveles (a,b: [Nivel]) : Bool = $|a| == |b| \wedge$
$(\forall x \leftarrow a) cantidadNivelesIguales(x, a) == cantidadNivelesIguales(x, b)$ ;
aux cantidadNivelesIguales (niv: Nivel, ns: [Nivel]) : $\mathbb{Z}$ = $|[\, n \,|\, n \leftarrow ns, nivelesIguales(n, niv)\,]|$ ;

aux vampirosMitadVida (vampiros : [(Vampiro, Posicion, Vida)]) : [(Vampiro, Posicion, Vida)] = [ $(prm(vampiro), sgd(vampiro), trc(vampiro)\ div\ 2) | vampiro \leftarrow vampiros]$ ;

aux terminado (n : Nivel) : Bool = $(|vampiros(n)| == 0 \wedge |spawning(n)| == 0) \vee vampirosEnCasa(vampiros(n)) > 0$ ;

aux nivelesGanados (j : Juego) : $[\mathbb{Z}] = [i | i \leftarrow [0..|niveles(j)|), |spawning(niveles(j)_i)| == 0 \wedge |vampiros(niveles(j)_i)| == 0]$ ;