



**DEPARTAMENTO  
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

# TPE - Flores vs Vampiros

Algoritmos y Estructuras de Datos I

**Grupo: 07**

Integrante	LU	Correo electrónico
Bukovits, Nicolás Axel	546/14	aturing@ejemplo.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 :  $vida(f) == \frac{100}{|habilidades(f)|+1} \wedge$ 
     $if\ Atacar \in habilidades(f)\ Then\ cP == \frac{12}{|habilidades(f)|}\ Else\ cP == 0;$ 
}

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 :  $cuanPega(res) == cP$ ;
}

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

problema cuantoPegaF (f: Flor) = res :  $\mathbb{Z}$  {
  asegura  $res == cuanPega(f)$ ;
}

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

```

## 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 :  $(\forall f \in flores(n))(prm(sgd(f)) \geq 0) \wedge (prm(sgd(f)) \leq ancho(n) \wedge (sgd(sgd(f)) \geq 0) \wedge (sgd(sgd(f)) \leq alto(n)) \wedge (\forall v \in vampiros(n))(prm(sgd(v)) \geq 0) \wedge (prm(sgd(v)) \leq ancho(n) \wedge (sgd(sgd(v)) \geq 0) \wedge (sgd(sgd(v)) \leq alto(n)))$ ;
  invariante spawningOrdenado :  $(\forall s \in spawning(n))((\forall i \in [0..|spawning(n)|-1])sgd(spawning(n)_i) \leq sgd(spawning(n)_{i+1}) \wedge (\forall j \in [0..|spawning(n)|-1], sgd(spawning(n)_j) == sgd(spawning(n)_{j+1})trd(spawning(n)_j) \wedge trd(spawning(n)_{j+1}))$ ;
  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 :  $an > 0, al > 0, (\forall t \leftarrow spaw)sgd(t) \geq 1 \wedge sgd(t) \leq al \wedge trd(t) \geq 0$ ;
  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  $res == flores(n)$ ;
}

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

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

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  $flores(n) == flores(pre(n))$ ;
  asegura  $vampiros(n) == vampiros(pre(n))$ ;
  asegura  $spawning(n) == spawning(pre(n))$ ;
}

problema obsesivoCompusilvo (n: Nivel) = res : Bool {
  asegura estaOrdenada : ( $\exists i \in [0..|estaOrdenada|-1]) Ataca \in habilidades(prm(ordendada_i)) \wedge$ 
     $Ataca \in habilidades(prm(ordendada_{i+1}))$ ;
}

problema agregarFlor (n: Nivel, f : Flor, p : Posicion) {
  requiere ( $\forall f \in flores(n) sgd(f) \neq p$ );
  requiere ( $\forall v \in vampiros(n) sgd(v) \neq p$ );
  requiere  $prim(p) \leq ancho(n) \wedge prm(p) \geq 0$ ;
  requiere  $sgd(p) \leq alto(n) \wedge sgd(p) \geq 0$ ;
  requiere  $soles(n) \geq 2^{|habilidades(f)|}$ ;
  modifica n;
  asegura  $flores(n) == mismos(flores(n), (cons(f, p, vida(f))))$ ;
  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  $vampiros(n) == vampiros(pre(n))$ ;
  asegura  $spawning(n) == spawning(pre(n))$ ;
}

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

problema pasarTurno (n: Nivel) {
  requiere noEsElFin :  $terminado(n) == False$ ;
  modifica n;

```

```

asegura graciasPachamama : soles(n) == soles(pre(n)) + 1 +
  |[1|j ∈ flores(n), Generar ∈ habilidadesF(j1)|] * (turno(pre(n)) + 1);
asegura soloPasaUnTurno : turno(n) == turno(pre(n)) + 1;
}

```

## 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)) prm(f) ∈ flores(j);
  invariante nivelesConVampirosValidos : (∀n ← niveles(j))(∀v ← vampiros(n)) prm(v) ∈ vampiros(j);
}

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

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

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

problema agregarNivelJ (j: Juego, n: Nivel, i: ℤ) {
  requiere turno(n) == 0;
  requiere |flores(n)| == 0;
  requiere |vampiros(n)| == 0;
  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 losSiguientesSonIguales : (∀k ← (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(result, nivelesConSoles(niveles(j)));
  asegura (∀f ∈ [0..|result| - 1]) |flores(resultf)| ≤ |flores(resultf+1)|;
}

problema jugarNivel (j: Juego, n: Nivel, i: ℤ) {
  requiere rangoValido : alto(n) == alto(niveles(j)i) ∧ ancho(n) == ancho(niveles(j)i);
  requiere noTeQuedesSinTurno : turno(niveles(j)i) ≤ turno(n);
  requiere noGanasteNiPerdisteAun : |spawning(n)| ≥ |spawning(niveles(j)i)|
    ∧ (∀k ∈ spawning(niveles(j)i)) k ∈ spawning(n);
  modifica j;
  asegura floresYVampirosNoCambian : vampiros(j) == vampiros(pre(j)) ∧ flores(j) == flores(pre(j));
}

```

```

    asegura losAnterioresSonIguales2 :  $(\forall k \leftarrow (0..i)) \text{nivelesIguales}(\text{niveles}(j)_k, \text{niveles}(\text{pre}(j))_k)$  ;
    asegura esElNivelQueQuiero2 :  $\text{soles}(n) == \text{soles}(\text{niveles}(j)_i)$ 
       $\wedge \text{mismos}(\text{flores}(n), \text{flores}(\text{niveles}(j)_i)) \wedge \text{mismos}(\text{vampiros}(n), \text{vampiros}(\text{niveles}(j)_i))$  ;
    asegura losSiguietesSonIguales2 :  $(\forall k \leftarrow (i..|\text{niveles}(j)|))$ 
       $\text{nivelesIguales}(\text{niveles}(j)_k, \text{niveles}(\text{pre}(j))_{k-1})$  ;
  }

problema altoCheat (j: Juego, i:  $\mathbb{Z}$ ) {
  requiere  $i \geq 0$  ;
  requiere  $i < |\text{niveles}(\text{pre}(j))|$  ;
  modifica j ;
  asegura  $(\forall v \in \text{vampiros}(\text{niveles}(\text{pre}(j))_i)) (\exists x \in \text{vampiros}(\text{niveles}(j)_i)) \text{prm}(x) == \text{prm}(v) \wedge$ 
     $\text{sgd}(x) == \text{sgd}(v) \wedge \text{trd}(x) == \frac{\text{trd}(v)}{2}$  ;
  asegura  $|\text{vampiros}(\text{niveles}(\text{pre}(j))_i)| == |\text{vampiros}(\text{niveles}(j)_i)|$  ;
  asegura  $(\forall v \in \text{vampiros}(\text{niveles}(j)_i)) (\exists x \in \text{vampiros}(\text{niveles}(\text{pre}(j))_i)) \text{prm}(v) == \text{prm}(x) \wedge$ 
     $\text{sgd}(v) == \text{sgd}(x) \wedge \text{trd}(v) == \frac{\text{trd}(x)}{2}$  ;
}

problema muyDeExactas (j: Juego) = res : Bool {
  asegura base1 : ifThenElse $(|\text{nivGsL}(j)| \geq 1 \wedge \text{nivGsL}(j)_0 == 1, \text{True}, \text{False})$  ;
  asegura base2 : ifThenElse $(|\text{nivGsL}(j)| \geq 2 \wedge \text{nivGsL}(j)_0 == 1 \wedge \text{nivGsL}(j)_1 == 2, \text{True}, \text{False})$  ;
  asegura sumaDeDos : ifThenElse $(|\text{nivGsL}(j)| \geq 3 \wedge \text{nivGsL}(j)_0 == 1 \wedge \text{nivGsL}(j)_1 == 2$ 
     $\wedge (\forall k \in [2..|\text{nivGsL}(j)|-1]) \text{nivGsL}(j)_k == \text{nivGsL}(j)_{k-1} + \text{nivGsL}(j)_{k-2}, \text{True}, \text{False})$  ;
}

```

## 6. Auxiliares

```

  aux vidaFloresOk (fs: [(Flor, Posicion, Vida)]) : Bool =  $(\forall f \leftarrow fs) \text{trd}(f) > 0 \wedge \text{trd}(f) \leq$ 
     $\text{vida}(\text{prm}(f))$  ;
  aux vidaVampirosOk (fs: [(Vampiro, Posicion, Vida)]) : Bool =  $(\forall f \leftarrow fs) \text{trd}(f) > 0 \wedge \text{trd}(f) \leq$ 
     $\text{vida}(\text{prm}(f))$  ;
  aux floresIguales (x, y : Flor) : Bool = mismos $(\text{habilidades}(x), \text{habilidades}(y))$  ;
  aux cuenta (x: T, a: [T]) : Int =  $|\{y \mid y \in a, y == x\}|$  ;
  aux mismos (a, b : [T]) : Bool =  $(|a| == |b| \wedge (\forall c \in a) \text{cuenta}(c, a) == \text{cuenta}(c, b))$  ;
  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 =
   $(\nexists k \in \text{vampiros}(n)) k_{2,2} \geq f_{2,2} \wedge k_{2,2} < v_{2,2}$  ;
  aux nivelGanado (n:Nivel) : Bool =  $\text{vampiros}(n) == []$  ;
  aux nivGsL (j: Juego) :  $\mathbb{Z}$  =  $[i \mid i \in [1..|\text{niveles}(j)|], \text{nivelGanado}(\text{niveles}(j)_i)]$  ;
  aux nivelesIguales (n1, n2: Nivel) : Bool =  $\text{ancho}(n1) == \text{ancho}(n2) \wedge \text{alto}(n1) == \text{alto}(n2) \wedge$ 
     $\text{turno}(n1) == \text{turno}(n2) \wedge \text{soles}(n1) == \text{soles}(n2) \wedge \text{mismos}(\text{flores}(n1), \text{flores}(n2)) \wedge$ 
     $\text{mismos}(\text{vampiros}(n1), \text{vampiros}(n2)) \wedge \text{spawning}(n1) == \text{spawning}(n2)$  ;
  aux nivelesConSoles (m : [Nivel]) : [Nivel] =  $[\text{niveles}(j)_i \mid i \in [0..|\text{niveles}(j)|], (\forall k \in [0..|\text{niveles}(j)|]) \text{soles}(\text{niveles}$ 
     $\text{soles}(\text{niveles}(j)_k))]$  ;

```