

# Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2015

1 de Abril de 2015

## TPE - 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 lasHabilidadesDeterminanLaVidaElGolpe : vida(f) == 100div(|habilidades(f)|+1) ^ cuantoPega(f) ==
    if en(Atacar, habilidades(f)) then 12div |habilidades(f)| else 0 ;
}

problema nuevaF (v :  $\mathbb{Z}$ , cP :  $\mathbb{Z}$ , hs : [Habilidad]) = res : Flor {
  requiere sinRepetidos(hs) ;
  requiere v == 100div(|hs| + 1) ^ cP == if en(Atacar, hs) then 12div |hs| else 0 ;
  asegura v == vida(res) ;
  asegura cP == cuantoPega(res) ;
  asegura mismos(hs, habilidades(res)) ;
}

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 :  $\mathbb{Z}$ , cP :  $\mathbb{Z}$ ) = res : Vampiro {
  requiere vidaEnRango : v  $\geq$  0  $\wedge$  v  $\leq$  100 ;
  requiere pegaEnSerio : cP > 0 ;
}
```

```

    asegura clase(res) == cV;
    asegura vida(res) == v;
    asegura cuantoPega(res) == cP;
}

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 : ( $\forall f \leftarrow flores(n)$ )( $1 \leq prm(sgd(f)) \leq ancho(n) \wedge 1 \leq sgd(sgd(f)) \leq alto(n)$ )  $\wedge$ 
        ( $\forall v \leftarrow vampiros(n)$ )( $1 \leq prm(sgd(v)) \leq ancho(n) \wedge 1 \leq sgd(sgd(v)) \leq alto(n)$ );
    invariante spawningOrdenado : ( $\forall i, j \leftarrow [0..|spawning(n)|], i < j$ ) peso(spawning(n)[i], n) < peso(spawning(n)[j], n);
    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 spawneanBien : ( $\forall t \leftarrow spawning(n)$ ) 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 unBuenSpawn : ( $\forall t \leftarrow spaw$ )  $1 \leq sgd(t) \leq al \wedge trd(t) \geq 0$ ;
    requiere entradasValidas : an > 0  $\wedge$  al > 0  $\wedge$  s  $\geq$  0;
    asegura vacio : flores(res) == []  $\wedge$  vampiros(res) == [];
    asegura dimensiones : ancho(res) == an  $\wedge$  alto(res) == al;
    asegura turno(res) == 0;
    asegura soles(res) == s;
    asegura 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  $|res| == |flores(n)| \wedge \forall(fr \leftarrow res)(\exists(fn \leftarrow flores(n))(floresIguales(prm(fr), prm(fn)) \wedge sgd(fr) ==$ 
       $sgd(fn) \wedge trd(fr) == trd(fn))$ );
  }

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  $s \geq 0$ ;
  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 res == ( $\forall i \leftarrow enOrden(flores(n), n)$ ) Obscom(i);
}

problema agregarFlor (n: Nivel, f : Flor, p : Posicion) {
  requiere  $1 \leq prm(p) \leq ancho(n) \wedge 1 \leq sgd(p) \leq alto(n)$ ;
  requiere  $soles(n) > pot(2, |habilidades(f)|)$ ;
  requiere ( $\forall i \leftarrow flores(n)$ )  $sgd(flores(n)_i) \neq p$ ;
  modifica n;
  asegura flores(n) == flores(pre(n)) ++ (f, p, vida(f));
}

aux terminado (n: Nivel) : Bool = if (!vNoCero(vampiros(n))  $\vee$  vampiros(n) == 0) then true else false;

problema pasarTurno (n: Nivel) {
  requiere terminado(pre(n)) == false;
  modifica n;
  asegura ancho(n) == ancho(pre(n));
  asegura alto(n) == alto(pre(n));
  asegura turno(n) == turno(pre(n)) + 1;
  asegura soles(n) == soles(pre(n)) + |floresGeneradoras(flores(pre(n)))| solesGenerados(n) + 1;
  asegura cambioFlores : flores(n) == floreSobrevivientes(pre(n));
  asegura cambioVampiros : vampiros(n) == vampiroSobrevivientes(pre(n)) ++ spawnedos(pre(n));
  asegura spawning(n) == nuevoSpawning(pre(n));
}

problema estaEnJaque (n: Nivel) = res : Vampiro {
}

```

## 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 : ( $\forall nivel \leftarrow niveles(j)$ )
    ( $(\forall tuplaFlor \leftarrow flores(nivel)) en(prm(tuplaFlor), flores(j))$ );
  invariante nivelesConVampirosValidos : ( $\forall nivel \leftarrow niveles(j)$ )
    ( $(\forall tuplaVampiro \leftarrow flores(nivel)) en(prm(tuplaVampiro), flores(j))$ );
}

```

```

}

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

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

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

problema agregarNivelJ (j: Juego, n: Nivel, i:  $\mathbb{Z}$ ) {
  modifica j;
  requiere  $0 \leq i \wedge i \leq |niveles(pre(j))|$ ;
  requiere nivelValido : turno(n) == 0  $\wedge$  |flores(n)| == 0  $\wedge$  |vampiros(n)| == 0;
  asegura indice(niveles(j), i) == n;
  asegura niveles(j)[0..i] ++ niveles(j)(i..|niveles(j)|) == niveles(pre(j));
}

problema estosSalenFacil (j: Juego) = res : [Nivel] {
  requiere |niveles(j)| > 0;
  asegura mismos(res, maxPlantas(maxSoles(niveles(res))));
}

problema jugarNivel (j: Juego, n: Nivel, i:  $\mathbb{Z}$ ) {
  requiere altoOK : alto(indice(i, niveles(pre(j)))) == alto(n);
  requiere anchoOK : alto(indice(i, niveles(pre(j)))) == ancho(n);
  requiere turnoOK : turno(indice(i, niveles(pre(j)))) < turno(n);
  requiere spawnOk : todos(spawnI, spawnI  $\leftarrow$  spawning(indice(i, niveles(pre(j))))) , todos(spawnN, spawnN  $\leftarrow$  spawning(N), prm(spawnI) == prm(spawnN)  $\wedge$  sgd(spawnI) == sgd(spawnN)  $\wedge$  tcr(spawnI) == tcr(spawnN));
  modifica j;
  asegura indice(i, niveles(j)) == n;
  asegura |niveles(j)| == |niveles(pre(j))|;
  asegura losDemasNoCambian : [indice(k, niveles(j)), k  $\leftarrow$  [0..i]] ++ [indice(k, niveles(j)), k  $\leftarrow$  (i..|niveles(j)|)] == niveles(pre(j));
}

problema altoCheat (j: Juego, i:  $\mathbb{Z}$ ) {
  requiere rangoOk :  $0 \leq i \leq |niveles(pre(j))|$ ;
  modifica j;
  ; asegura mitadVida : todos(truplaPreV, truplaPreV < -vampiros(indice(i, niveles(pre(j))))) , todos(truplaV, truplaV < -vampiros(indice(i, niveles((j))))) , tcr(truplaV) == tcr(truplaPreV)/2;
  asegura nadaMasCambio : todos(truplaPreV, truplaPreV  $\leftarrow$  vampiros(indice(i, niveles(pre(j))))) , todos(truplaV, truplaV  $\leftarrow$  vampiros(indice(i, niveles((j))))) , prm(truplaV) == prm(truplaPreV)  $\wedge$  sgd(truplaV) == sgd(truplaPreV);
}

problema muyDeExactas (j: Juego) = res : Bool {
  asegura sum(i, i  $\leftarrow$  ganados(niveles(j))) == indice(|ganados(niveles(j))| - 1, ganados(niveles(j)));
}

problema nivelesSoleados (j: Juego) = res : [Nivel] {
}

```

## 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 peso (sp: (Vampiro,  $\mathbb{Z}$ ,  $\mathbb{Z}$ ), n: Nivel) :  $\mathbb{Z}$  = trd(sp) * alto(n) + sgd(sp);
aux vNoCero (vmprs: [(Vampiro, Posicion, Vida)]) : Bool = ( $\forall v \leftarrow vmprs$ ) sgd(sgd(v))  $\neq$  0;
aux maxSoles (ns: [Nivel]) : [Nivel] = [nivel | nivel  $\leftarrow ns$ , maxS(nivel)];

```

```

aux maxS (ns: [Nivel], maximo) : Bool = ( $\forall nivel \leftarrow ns, soles(nivel) \leq maximo$ ) nivel ;
aux maxFlores (ns: [Nivel]) : [Nivel] = [nivel | nivel  $\leftarrow ns, maxP(ns, |flores(nivel)|)$ ] ;
aux maxP (ns: [Nivel], maximo) : Bool = ( $\forall nivel \leftarrow ns, |flores(nivel)| \leq maximo$ ) ;
aux ganados (ns : [Nivel]) : [Nivel] = [nivel, nivel  $\leftarrow ns, |spawning(nivel)| == 0$ ] ;
aux obsCom (x: [(Flores, Posicion, Vida)]) : Bool = if ( $\exists j \leftarrow x, j \neq 0 \vee j \neq |x| - 1$ ) habilidadesF(prm(j)) ==
Atacar then habilidadesF(prm(j - 1))  $\neq$  Atacar  $\wedge$  habilidadesF(prm(j + 1))  $\neq$  Atacar else habilidadesF(prm(j - 1)) ==
Atacar  $\wedge$  habilidadesF(prm(j + 1)) == Atacar ;
aux enOrden ((flor: [(Flor, Posicion, Vida)], niv: Nivel)) : [(Flor, Posicion, Vida)] = [(prm(f), (posx, posy), trd(f)) | f  $\leftarrow$ 
flor, posx  $\leftarrow$  [0..ancho(niv)], posy  $\leftarrow$  [0..alto(niv)]] ;
aux mismos (a, b: [T]) : Bool = (|a| == |b|)  $\wedge$  ( $\forall c \leftarrow a, cuenta(c, a) == cuenta(c, b)$ ) ;
aux cuenta (x: T, a: [T]) :  $\mathbb{Z}$  = [|y |  $\forall y \leftarrow a, y == x$ ] ;
aux floresGeneradoras (flores : [Flor]) : [Flor] = [flor | flor  $\leftarrow$  flores, esGeneradora(flor)] ;
aux esGeneradora (flor : Flor) : Bool = alguno(h, h  $\leftarrow$  [1..|habilidades(flor)|], indice(habilidades(flor), h) == Generar) ;
aux floreSobrevivientes (n: Nivel) : [(Flor, Posicion, Vida)] = [(prm(sbrvF), sgd(sbrvF), nuevaVidaFlor(sbrvF, n)) |
sbrvF  $\leftarrow$  flores(n), !florMuerta(sbrvF, n)  $\wedge$  !florExplotaConVampiro(sbrvF, n)] ;
aux vampiroSobrevivientes (n: Nivel) : [(Vampiro, Posicion, Vida)] = [(prm(sbrV),
nuevaPosicionVampiro(sbrV, n), nuevaVidaVampiro(sbrV, n)) | sbrV  $\leftarrow$  vampiros(n), !vampiroMuerto(sbrV, n)] ;
aux florExplotaConVampiro (i: (Flor, Posicion, Vida), n: Nivel) : Bool = esExplotadora(prm(i))  $\wedge$ 
compartenPosicionF(sgd(i), vampiros(n)) ;
aux esExplotadora (flor : Flor) : Bool = ( $\exists h \leftarrow$  [0..|habilidades(flor)|]) habilidades(flor)h == Explotar ;
aux compartenPosicionF (posF: Posicion, v: [(Vampiro, Posicion, Vida)]) : Bool = ( $\exists h \leftarrow v$ ) sgd(h) == sgd(posF) ;
aux vampiroMuerto (i: (Vampiro, Posicion, Vida), n: Nivel) : Bool = fueAtacado(sgd(i), n)  $\geq$  trd(i) ;
aux florMuerta (i: (Flor, Posicion, Vida), n: Nivel) : Bool = fueAtacada(sgd(i), n)  $\geq$  trd(i) ;
aux fueAtacada (posF: Posicion, n: Nivel) :  $\mathbb{Z}$  = sum([cuantoPega(prm(v)) | v  $\leftarrow$  vampiros(n), sgd(v) == posF]) ;
aux fueAtacado (posV: Posicion, n: Nivel) :  $\mathbb{Z}$  = sum([cuantoPega(prm(f)) | f  $\leftarrow$  flores(n), prm(sgd(f)) == prm(posV)
 $\wedge$  sgd(sgd(f)) < sgd(posV)]) ;
aux nuevaVidaFlor (flor: (Flor, Posicion, Vida), n: Nivel) :  $\mathbb{Z}$  = trd(flor) - fueAtacada(sgd(flor), n) ;
aux nuevaVidaVampiro (vamp: (Vampiro, Posicion, Vida), n: Nivel) :  $\mathbb{Z}$  = trd(vamp) - fueAtacado(sgd(vamp), n) ;
aux nuevaPosicionVampiro (vam: (Vampiro, Posicion, Vida), n: Nivel) : Posicion = if
compartenPosicionV(sgd(vam), flores(n)) then if compartePosExp(sgd(vam), flores(n)) then
posMas1(vam) else sgd(vam) else if vampiroDesviado(prm(vam)) then if prm(sgd(prm))  $\leq$  0 then
posMenos2(vam) else posMenos1(vam) else posMenos1(vam) ;
aux posMas1 (vam: (Vampiro, Posicion, Vida)) : Posicion = (prm(sgd(vam)), sgd(sgd(vam)) + 1) ;
aux posMenos1 (vam: (Vampiro, Posicion, Vida)) : Posicion = (prm(sgd(vam)), sgd(sgd(vam)) - 1) ;
aux posMenos2 (vam: (Vampiro, Posicion, Vida)) : Posicion = (prm(sgd(vam)) - 1, sgd(sgd(vam)) - 1) ;
aux compartenPosicionV (posV: Posicion, f: [(Flor, Posicion, Vida)]) : Bool = ( $\exists h \leftarrow f$ ) sgd(h) == sgd(posV) ;
aux compartenPosExp (posV: Posicion, f: [(Flor, Posicion, Vida)]) : Bool =  $\exists h \leftarrow f$  sgd(h) == sgd(posV)  $\wedge$ 
esExplotadora(prm(h)) ;
aux vampiroDesviado (borracho: Vampiro) : Bool = clase(borracho) == Desviado ;
aux spawnados (n: Nivel) : [(Vampiro, Posicion, Vida)] = [(prm(i), (sgd(i), ancho(n)), vida(sgd(i))) | i  $\leftarrow$  spawning(n),
trd(i) == turno(n)] ;
aux nuevoSpawning (n: Nivel) : [(Vampiro,  $\mathbb{Z}$ ,  $\mathbb{Z}$ )] = [i | i  $\leftarrow$  spawning(n), trd(i) > turno(n)] ;

```