

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 lasHabilidadesDeterminanLaVidaYElGolpe : vida(f) == 100div(|habilidades(f)|+1) $\wedge$ 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)  $\wedge$  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 spawnanBien : ( $\forall t \leftarrow spawning(n)$ ) 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 mismos(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 :  $\mathbb{Z}$ ) {
  modifica n;
  asegura soles(n) == soles(pre(n)) + s;
}

problema obsesivoCompusilvo (n: Nivel) = res : Bool {
}

problema agregarFlor (n: Nivel, f : Flor, p : Posicion) {
  requiere soles(pre(n)) > pot(2, |habilidades(f)|);
  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 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 : ( $\forall i, k \leftarrow [0..|flores(j)|], i \neq k$ )  $\neg$  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 :  $ancho(indice(i, niveles(pre(j)))) == ancho(n)$ ;
  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:  $\mathbb{Z}$ ) {
  modifica j;
  ; requiere rangoOk :  $0 \leq i \leq |niveles(pre(j))|$ ;
  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  $sum(i, i \leftarrow ganados(niveles(j))) == indice(|ganados(niveles(j))| - 1, ganados(niveles(j)))$ ;
}

```

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] =  $[nivel, nivel \leftarrow ns, maxS(ns, soles(nivel))]$ ;
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] =  $[nivel, nivel \leftarrow ns, |spawning(nivel)| == 0 \wedge]$ ;

```