

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 : ...;
    invariante spawningOrdenado : ...;
    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 {
}

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

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

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

problema solesN (n : Nivel) = res :  $\mathbb{Z}$  {
}

problema floresN (n : Nivel) = res : [(Flor, Posicion, Vida)] {
}

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

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

problema comprarSoles (n: Nivel, s :  $\mathbb{Z}$ ) {
}

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

problema agregarFlor (n: Nivel, f : Flor, p : Posicion) {
}

```

```

}
aux terminado (n: Nivel) : Bool = ;
problema pasarTurno (n: Nivel) {
}

```

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  $0 \leq i \wedge i \leq |niveles(j)|$ ;
  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] {
}

problema jugarNivel (j: Juego, n: Nivel, i:  $\mathbb{Z}$ ) {
}

problema altoCheat (j: Juego, i:  $\mathbb{Z}$ ) {
}

problema muyDeExactas (j: Juego) = res : Bool {
}

```

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));

```