UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

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` : $vida(f) == 100\text{div}(|habilidades(f)|+1) \wedge cuantoPega(f) ==$ if $\text{en}(Atacar, habilidades(f))$ then $12\textbf{div}\,|habilidades(f)|$ else 0 ;
}

`problema nuevaF` (v : $\mathbb{Z}$, cP : $\mathbb{Z}$, hs : [Habilidad]) = `res` : Flor {
    `requiere` $sinRepetidos(hs)$;
    `requiere` $v == 100\text{div}(|hs| + 1) \wedge cP ==$ if $\text{en}(Atacar, hs)$ then $12\textbf{div}\,|hs|$ 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 : ℤ  {
        asegura res == vida(v) ;
}

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

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 : ... ;
        invariante spawningOrdenado : ... ;
        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)]) = res : Nivel  {
}

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

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

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

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

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

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

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

problema comprarSoles (n: Nivel, s : 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` : $(\forall nivel \leftarrow niveles(j))$
        $((\forall tuplaFlor \leftarrow flores(nivel))\mathsf{en}(prm(tuplaFlor), flores(j)))$;
    `invariante nivelesConVampirosValidos` : $(\forall nivel \leftarrow niveles(j))$
        $((\forall tuplaVampiro \leftarrow flores(nivel))\mathsf{en}(prm(tuplaVampiro), flores(j)))$;
}

`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: $\mathbb{Z}$) {
}

`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 = $mismos(habilidades(x), habilidades(y))$;