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	edjikstra@ejemplo.com
Frachtenberg Goldsmit, Kevin	247/14	kevinfra94@gmail.com
Garrett, Philip	318/14	pgarret@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autńoma 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` : $cuantoPega(res) == cP$;
}

`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` $res == habilidades(f)$;
}

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 : ℤ, cP : ℤ) = res : Vampiro {
        requiere v ≥ 0 ∧ v ≤ 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 : 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 {
        requiere esBienRazonable : an > 0, al > 0, (∀t ← spaw)sgd(t) ≥ 1 ∧ sgd(t) ≤ al ∧ trd(t) ≥
            0;
        asegura empezasEnTurnoCero : turno(res) == 0;
        asegura tieneTablero : ancho(res) == an ∧ alto(res) == al;
        asegura esDiaSoleado : soles(res) == s;
        asegura elTableroEstaVacio : vampiros(res) == [] ∧ flores(res) == [];
        asegura habraEnemigosOrdenados : mismos(spawning(res), spaw);
}

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

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

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

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

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

aux terminado (n: Nivel) : Bool = ;

problema pasarTurno (n: Nivel) {
    requiere noEsElFin : $terminado(n) == False$;
    modifica n;
    asegura graciasPachamama : $soles(n) == soles(pre(n)) + 1 +$
    $|[1|j \in flores(n), Generar \in habilidadesF(j_1)]| * (turno(pre(n)) + 1)$;
    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 : $(\forall i, k \leftarrow [0..|flores(j)|), i \neq k) \neg floresIguales(flores(j)_i, flores(j)_k)$;
    invariante vampirosDistintos : $sinRepetidos(vampiros(j))$;
    invariante nivelesConFloresValidas : $(\forall n \leftarrow niveles(j))(\forall f \leftarrow flores(n))prm(f) \in flores(j)$;
    invariante nivelesConVampirosValidos : $(\forall n \leftarrow niveles(j))(\forall v \leftarrow vampiros(n))prm(v) \in vampiros(j)$;
}

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: $\mathbb{Z}$) {
requiere $turno(n) == 0$;
requiere $|flores(n)| == 0$;
requiere $|vampiros(n)| == 0$;
requiere $i \geq 0$ $landi \leq |niveles(j)|$;
modifica j;
asegura losAnterioresSonIguales : $(\forall k \leftarrow (0..i))nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
asegura esElNivelQueQuiero : $nivelesIguales(niveles(j)_i, n)$;
asegura losSiguientesSonIguales : $(\forall k \leftarrow (i..|niveles(j)|))nivelesIguales(niveles(j)_k, niveles(pre(j))_{k-1})$
asegura $mismos(flores(j), flores(pre(j)))$;
asegura $mismos(vampiros(j), vampiros(pre(j)))$;
}

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

problema jugarNivel (j: Juego, n: Nivel, i: $\mathbb{Z}$) {
requiere rangoValido : $alto(n) == alto(niveles(j)_i) \land ancho(n) == ancho(niveles(j)_i)$;
requiere noTeQuedesSinTurno : $turno(niveles(j)_i) \leq turno(n)$;
requiere noGanasteNiPerdisteAun : $|spawning(n)| \geq |spawning(niveles(j)_i)| \land (\forall k \in spawning(niveles(j)_i)$
$spawning(n)$;
modifica j;
asegura floresYVampirosNoCambian : $vampiros(j) == vampiros(pre(j)) \land flores(j) ==$
$flores(pre(j))$;
asegura losAnterioresSonIguales2 : $(\forall k \leftarrow (0..i))nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$;
asegura esElNivelQueQuiero2 : $soles(n) == soles(niveles(j)_i)$
$\land mismos(flores(n), flores(niveles(j)_i)) \land mismos(vampiros(n), vampiros(niveles(j)_i))$;
asegura losSiguientesSonIguales2 : $(\forall k \leftarrow (i..|niveles(j)|))nivelesIguales(niveles(j)_k, niveles(pre(j))_{k-1}$
}

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

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

6. Auxiliares

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

`aux nivelesIguales` (n1,n2: Nivel) : Bool $= ancho(n1) == ancho(n2) \wedge alto(n1) == alto(n2) \wedge turno(n1) == turno(n2) \wedge soles(n1) == soles(n2) \wedge mismos(flores(n1), flores(n2)) \wedge mismos(vampiros(n1), vampiros(n2)) \wedge spawning(n1) == spawning(n2)$;