



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 1

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 Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax : (5411)4576 - 3359

[http : //www.fcen.uba.ar](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 :  $cuanPega(res) == cP$ ;
}

problema vidaF (f: Flor) = res :  $\mathbb{Z}$  {
  asegura  $res == vida(f)$ ;
}

problema cuantoPegaF (f: Flor) = res :  $\mathbb{Z}$  {
  asegura  $res == cuanPega(f)$ ;
}

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 :  $cuanPega(v) > 0$ ;
}

```

```

problema nuevoV (cv : ClaseVampiro, v :  $\mathbb{Z}$ , cP :  $\mathbb{Z}$ ) = res : Vampiro {
  requiere  $v \geq 0 \wedge v \leq 100$ ;
  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 :  $\mathbb{Z}$  {
  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 spawnenBien :  $(\forall t \leftarrow spawning(n))sgd(t) \geq 1 \wedge sgd(t) \leq alto(n) \wedge trd(t) \geq 0$ ;
}

problema nuevoN (an :  $\mathbb{Z}$ , al :  $\mathbb{Z}$ , s :  $\mathbb{Z}$ , spaw : [(Vampiro,  $\mathbb{Z}$ ,  $\mathbb{Z}$ )]) = res : Nivel {
  requiere esBienRazonable :  $an > 0, al > 0, (\forall t \leftarrow spaw)sgd(t) \geq 1 \wedge sgd(t) \leq al \wedge trd(t) \geq 0$ ;
  asegura empezasEnTurnoCero :  $turno(res) == 0$ ;
  asegura tieneTablero :  $ancho(res) == an \wedge alto(res) == al$ ;
  asegura esDiaSoleado :  $soles(res) == s$ ;
  asegura elTableroEstaVacio :  $vampiros(res) == [] \wedge flores(res) == []$ ;
  asegura habraEnemigosOrdenados :  $mismos(spawning(res), spaw)$ ;
}

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);
}

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(j1)|] * (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  $i \geq 0$  land  $i \leq |niveles(j)|$ ;
  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:  $alto(n) == alto(niveles(j)_i) \wedge ancho(n) == ancho(niveles(j)_i)$ ;
  requiere noTeQuedesSinTurno:  $turno(niveles(j)_i) \leq turno(n)$ ;
  requiere noGanasteNiPerdisteAun:  $|spawning(n)| \geq |spawning(niveles(j)_i)| \wedge (\forall k \in spawning(niveles(j)_i))$ 
     $spawning(n)$ ;
  modifica  $j$ ;
  asegura floresYVampirosNoCambian:  $vampiros(j) == vampiros(pre(j)) \wedge flores(j) == flores(pre(j))$ ;
  asegura losAnterioresSonIguales2:  $(\forall k \leftarrow (0..i))nivelesIguales(niveles(j)_k, niveles(pre(j))_k)$ ;
  asegura esElNivelQueQuiero2:  $soles(n) == soles(niveles(j)_i)$ 
     $\wedge mismos(flores(n), flores(niveles(j)_i)) \wedge mismos(vampiros(n), vampiros(niveles(j)_i))$ ;
  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  $listaNivelesGanados(j) == fibonacci(|listaNivelesGanados(j)|)$ ;
}

```

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: 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 =  $(|a| == |b| \wedge (\forall c \in a) cuenta(c, a) == cuenta(c, b))$ ;
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 =
   $(\nexists k \in vampiros(n)) k_{2,2} \geq f_{2,2} \wedge k_{2,2} < v_{2,2}$ ;
aux nivelGanado (n: Nivel) : Bool =  $vampiros(n) == []$ ;
aux listaNivelesGanados (j: Juego) : [ $\mathbb{Z}$ ] =  $[i | i \in [1..|niveles(j)|], nivelGanado(niveles(j)_i)]$ ;
aux fibonacci (t:  $\mathbb{Z}$ ) : [ $\mathbb{Z}$ ] =  $[\frac{\phi^n - (1-\phi)^n}{\sqrt{5}} \mid n \in [1..t]]$ ;
aux nivelesIguales (n1, n2: Nivel) : Bool =  $ancho(n1) == ancho(n2) \wedge alto(n1) == alto(n2) \wedge$ 
   $turno(n1) == turno(n2) \wedge soles(n1) == soles(n2) \wedge mismos(flores(n1), flores(n2)) \wedge$ 
   $mismos(vampiros(n1), vampiros(n2)) \wedge spawning(n1) == spawning(n2)$ ;

```