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

Aclaraciones

- Para aprobar la totalidad del TP es necesario tener aprobado cada uno de sus módulos.
- **No** está permitido el uso de **acum** para la resolución.

Juegos CapPop es una reconocida empresa de videojuegos que quiere nuestros servicios para crear el juego que sea el *boom* del 2015. A la cátedra se le ocurrió una idea millonaria e innovadora, el juego *Flores vs Vampiros*. Este juego consiste en dos bandos, las flores y los vampiros, que juegan enfrentados. El jugador controla a las flores y la computadora controla a los vampiros.
Los niveles del juego consisten en una cuadrícula en donde se pueden posicionar las flores y los vampiros. Las columnas de la cuadrícula están numeradas de izquierda a derecha y las filas están numeradas de arriba hacia abajo. El nivel se va jugando por turnos. En ciertos turnos, nuevos vampiros aparecen en el extremo derecho de la cuadrícula y su misión es caminar por la fila en la que aparecieron, hasta salir por el extremo izquierdo. Para impedir esto, el jugador puede posicionar flores en el nivel. Las flores tienen diferentes habilidades para tratar de parar a los ~~zombies~~ vampiros. Colocar una nueva flor en un nivel no es gratuito sino que se necesitan *soles*, que se van juntando a lo largo del juego.

El objetivo del trabajo práctico es especificar varios tipos y problemas que conforman la base del juego que van a hablar todos los *casual gamers* del mundo.

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) : ℤ;
       observador cuantoPega (f: Flor) : ℤ;
       observador habilidades (f: Flor) : [Habilidad];

       invariante sinRepetidos(habilidades(f));
       invariante lasHabilidadesDeterminanLaVidayElGolpe : ...;
}
```

1. `invariante` lasHabilidadesDeterminanLaVidayElGolpe
   Completar el invariante de manera tal que la vida de la flor y lo que golpea queden determinados por las habilidades de la siguiente manera:
   - vida: 100HP dividido la cantidad de habilidades, más una.
   - cuantoPega: Si una de sus habilidades es atacar, 12HP dividido la cantidad de habilidades, sino 0HP.
2. `problema nuevaF` (v : $\mathbb{Z}$, cP : $\mathbb{Z}$, hs : [Habilidad]) = `result` : Flor
   Crea una nueva flor, con vida v, cuantoPega igual a cP y las habilidades presentes en hs.
3. `problema vidaF` (f: Flor) = `result` : $\mathbb{Z}$
   Devuelve la vida correspondiente a la flor f.
4. `problema cuantoPegaF` (f: Flor) = `result` : $\mathbb{Z}$
   Devuelve el valor de daño que infringe la flor f.
5. `problema habilidadesF` (f: Flor) = `result` : [Habilidad]
   Devuelve la lista de habilidades correspondiente a la flor f.

3. Vampiro

```
tipo Vampiro {
       observador clase (v: Vampiro) : ClaseVampiro;
       observador vida (v: Vampiro) : ℤ;
       observador cuantoPega (v: Vampiro) : ℤ;

       invariante vidaEnRango : vida(v) ≥ 0 ∧ vida(v) ≤ 100;
       invariante pegaEnSerio : cuantoPega(v) > 0;
}
```

6. problema nuevoV (cv : ClaseVampiro, v : ℤ, cP : ℤ) = result : Vampiro
   Crea un nuevo vampiro con clase igual a cv, vida igual a v y cuantoPega igual a cP.
7. problema claseVampiroV (v : Vampiro) = result : ClaseVampiro
   Devuelve la claseVampiro correspondiente al vampiro v.
8. problema vidaV (v : Vampiro) = result : ℤ
   Devuelve la vida correspondiente al vampiro v.
9. problema cuantoPegaV (v : Vampiro) = result : ℤ
   Devuelve cuanto pega el vampiro 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;
}
```

10. invariante posicionesValidas
    Completar el invariante de forma que se garantice que las flores y los vampiros estén en posiciones en el rango de la cuadrícula del nivel.
11. invariante spawningOrdenado
    Rellenar el invariante de manera que la lista de spawning de un nivel tenga a los vampiros ordenados por turno ascedente, como primer criterio, y por fila ascendente como segundo criterio.
12. problema nuevoN (an : ℤ, al : ℤ, s : ℤ, spaw : [(Vampiro, ℤ, ℤ)]) = result : Nivel
    Crea un nuevo nivel, con ancho an, alto al, el turno en 0, s soles iniciales, y una lista de spawning $spaw$. La lista de spawning está compuesta por triplas, en las cuales se pueden encontrar primero al vampiro que va a aparecer en el juego, luego a la fila en la que va a aparecer, y por últimos el turno en el que aparecerá. Dado que se está creando una nivel nuevo, el mismo no tiene flores ni vampiros en el mismo.
13. problema anchoN (n : Nivel) = result : ℤ
    Devuelve el ancho correspondiente al nivel n.
14. problema altoN (n : Nivel) = result : ℤ
    Devuelve el alto correspondiente al nivel n.
15. problema turnoN (n : Nivel) = result : ℤ
    Devuelve el turno correspondiente al nivel n.
16. problema solesN (n : Nivel) = result : ℤ
    Devuelve los soles correspondientes al nivel n.

17. `problema floresN` (n : Nivel) = `result` : [(Flor, Posicion, Vida)]
Devuelve la lista de flores correspondiente al nivel n.

18. `problema vampirosN` (n : Nivel) = `result` : [(Vampiro, Posicion, Vida)]
Devuelve la lista de vampiros correspondiente al nivel n.

19. `problema spawningN` (n : Nivel) = `result` : [(Vampiro, $\mathbb{Z}$, $\mathbb{Z}$)]
Devuelve el spawning de vampiros correspondiente al nivel n.

20. `problema comprarSoles` (n: Nivel, s : $\mathbb{Z}$)
Modifica el nivel n de manera que lo único que se modifica es que se pueda contar con s soles más.

21. `problema obsesivoCompusilvo` (n: Nivel) = `result` : Bool
Devuelve verdadero si y solo si al ordenar las flores del nivel primero por fila y luego por columna, se puede observar un patrón alternante de flores *atacantes* y flores *no atacantes*.

22. `problema agregarFlor` (n: Nivel, f : Flor, p : Posicion)
Modifica el nivel n de manera que se agrega la flor f a la posición p. Los soles necesarios para agregar dicha flor se calculan como 2 elevado a la cantidad de habilidades de la misma.

23. `aux terminado` (n: Nivel) : Bool
Este auxiliar debe indicar si el nivel está terminado. Un nivel se encuentra terminado cuando un vampiro logra llegar a la columna 0 (caso en el que ganaron los vampiros), o cuando no existen más vampiros.

24. `problema pasarTurno` (n: Nivel)
Modifica el nivel pasando el turno. Se necesita que el nivel no este terminado para poder realizar esta acción. El hecho de pasar el turno consiste en que:
    - Los soles se modifican recaudando todo lo generado por la plantas presentes hasta el turno anterior. Por cada planta con habilidad para generar, se recauda un sol por turno. Además, la madre naturaleza, nos provee un sol por turno.
    - El turno se incrementa en uno.
    - Las flores y los vampiros se modifican primero por sus ataques y luego por los movimientos.
    - Las flores reciben un ataque igual a la suma de lo que atacan todos los vampiros que comparten su posición. Las flores que resulten con vida no positiva, deben ser extraídas de la lista de flores del nivel. Las flores que tienen la habilidad de explotar y se encuentran en la misma posición que algún vampiro, mueren directamente.
    - Los vampiros reciben un ataque igual a la suma de lo que atacan todas las flores que estén en su misma fila, en una columna menor o igual a la del vampiro y que no haya un vampiro en una columna intermedia. Los vampiros que resulten con vida no positiva, deben ser extraídos de la lista de vampiros del nivel.
    - Todos los vampiros que sobreviven a los anteriores ataques tratan de avanzar en el nivel. Si se encuentran en la misma posición que una flor sobreviviente, entonces no pueden avanzar en este turno. Si se encuentran en la misma posición que una flor con la habilidad de explotar, que acaba de morir, entonces retroceden una posición, lo que significa que incrementan su valor de columna en 1. De lo contrario, avanzan una posición, lo que significa que decrementan su valor de columna en 1. Además si el vampiro es de tipo *Desviado* y no se encuentra en la fila 1 del nivel, decrementa su valor de fila en 1.
    - Por último, aparecen los vampiros de la lista de spawning que coincidan con el turno actual, en la fila pertinente. Estos vampiros deben ser trasladados de la lista de spawning a la lista de vampiros del nivel.

5. Juego

```
tipo Juego {
       observador flores (j: Juego) : [Flor] ;
       observador vampiros (j: Juego) : [Vampiro] ;
       observador niveles (j: Juego) : [Nivel] ;
       invariante floresDistintas : (∀i, k ← [0.. |flores(j)|), i ≠ k)¬floresIguales(flores(j)_i, flores(j)_k) ;
       invariante vampirosDistintos : sinRepetidos(vampiros(j)) ;
       invariante nivelesConFloresValidas : ... ;
       invariante nivelesConVampirosValidos : ... ;
}
```

25. `invariante` nivelesConFloresValidas
Este invariante indica que los niveles de un juego solo poseen flores que están en la lista de flores del juego.

26. `invariante` nivelesConVampirosValidos
Este invariante indica que los niveles de un juego solo poseen vampiros que están en la lista de vampiros del juego.

27. `problema floresJ` (j: Juego) = `result` : [Flor]
Devuelve las flores correspondientes al juego j.

28. `problema vampirosJ` (j: Juego) = `result` : [Vampiro]
Devuelve los vampiros correspondientes al juego j.

29. `problema nivelesJ` (j: Juego) = `result` : [Nivel]
Devuelve los niveles correspondientes al juego j.

30. `problema agregarNivelJ` (j: Juego, n: Nivel, i: $\mathbb{Z}$)
Modifica el juego j, agregando el nivel n como el i-ésimo nivel del juego. El nivel debe estar en su comienzo, es decir en el primer turno y sin flores o vampiros sobre el tablero.

31. `problema estosSalenFacil` (j: Juego) = `result` : [Nivel]
Dado el juego j, selecciona los niveles que más soles tiene y, dentro de este grupo, devuelve los que más plantas tienen.

32. `problema jugarNivel` (j: Juego, n: Nivel, i: $\mathbb{Z}$)
Este problema toma un juego j, una *foto* de un nivel n y un indice i. Luego, modifica el juego j de manera que el estado del nivel i-ésimo pasa a ser n. Para que esto sea posible el nivel n debe ser un posible estado futuro del i-ésimo nivel del juego j.

33. `problema altoCheat` (j: Juego, i: $\mathbb{Z}$)
Dado el juego j y el indice i. Modifica el i-ésimo nivel de j haciendo que todos los vampiros vivos del nivel (no los que están esperando para spawnear) reduzcan su vida a la mitad. Reducir la vida a la mitad significa realizar una división entera por 2 de la vida actual.

34. `problema muyDeExactas` (j: Juego) = `result` : Bool
Este problema toma un juego j e indica si los niveles ganados son exactamente los que pertenecen a la sucesión de Fibonacci.

6. Auxiliares

`aux sinRepetidos` (ls:[T]) : Bool $= (\forall i, j \leftarrow [0..|ls|), i \neq j) ls_i \neq ls_j$;
`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) : Bool $= mismos(habilidades(x), habilidades(y))$;