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

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2015

6 de Mayo de 2015

TPI - Flores vs Vampiros

1. Introducción

En el TPE, hemos especificado el comportamiento de una serie de problemas relacionados con el juego Flores vs Vampiros. Nuestro próximo paso será obtener una implementación en lenguaje imperativo, para esto se brinda una adaptación de la especificación original.

El objetivo de este trabajo es programar toda la especificación, usando los comandos de C++ vistos en clase. Pueden utilizar las clase `vector` provista en la stl de C++, pero sólo se permitirá el uso de los métodos especificados en este enunciado.

En la página de la materia estarán disponibles los *headers* de las clases que deberán implementar y la especificación de cada uno de sus métodos.

Pueden agregar los métodos auxiliares que necesiten. Estos deben estar *en todos los casos* en la parte privada de la clase. No se permite modificar la parte pública de las clases ni agregar atributos adicionales, ya sean públicos o privados.

2. Clase `std::vector`

Usaremos el tipo [T] (lista) para especificar la clase vector<T>.

`problema nuevoV` () = `this` : [T] {
    `asegura` $|this| == 0$;
}

`problema size` (this: [T]) = `res` : $\mathbb{Z}$ {
    `asegura` $res == |this|$;
}

`problema empty` (this: [T]) = `res` : Bool {
    `asegura` $res == (|this| == 0)$;
}

`problema at` (this: [T], i: $\mathbb{Z}$) = `res` : T {
    `requiere` $0 \leq i < |this|$;
    `asegura` $res == this_i$;
} Nota: el operador [] cumple la misma especificación, también pueden usarlo.

`problema push_back` (this: [T], e: T) {
    `modifica` $this$;
    `asegura` $this == pre(this) + +[e]$;
}

`problema pop_back` (this: [T]) {
    `requiere` $|this| > 0$;
    `modifica` $this$;
    `asegura` $pre(this) == this + +[e]$;
}

3. Implementación de tipos

Los tipos Flor, Vampiro, Nivel y Juego mantienen, dentro de la clase, atributos que son similares a sus observadores y por lo tanto es trivial comprender su equivalencia. Dentro de la clase Nivel pueden encontrarse los structs `FlorEnJuego`, `VampiroEnJuego` y `VampiroEnEspera`, los cuales representan las triplas que contienen las flores, los vampiros y el spawning de cada nivel.

4. Entrada/Salida

Todas las clases del proyecto tienen tres métodos relacionados con entrada salida:

mostrar : que se encarga de mostrar todo el contenido de la instancia de la clase en el flujo de salida indicado. El formato es a gusto del consumidor, pero esperamos que sea algo más o menos informativo.

guardar : que se encarga de escribir la información de cada instancia en un formato predeterminado que debe ser decodificable por el método que se detalla a continuación.

cargar : que se encarga de leer e interpretar información generada por el método anterior, modificando el valor del parámetro implícito para que coincida con aquel que generó la información.

En definitiva, `guardar` se usará para "grabar" en un archivo de texto el contenido de una instancia mientras que "cargar" se usará para "recuperar" dicha información. En todos los casos, sus interfaces serán:

- `void mostrar(std::ostream& ) const;`
- `void guardar(std::ostream& ) const;`
- `void cargar(std::istream& );`

El detalle del formato que se debe usar para "guardar" y "leer" en cada clase se indica a continuación. Tener en cuenta que los números se deben guardar como texto. Es decir, si escriben a un archivo y después lo miran con un editor de texto, donde escribieron un número 65 deben ver escrito 65 y no una letra A.

Flor: Se debe guardar `"{ F "`, su vida, `" "`, su cuantoPega, `" "`, su lista de habilidades separadas por espacio y `" }"`. Por ejemplo, una flor que pueda Atacar y Explotar se guarda:

```
{ F 33 6 [ Atacar Explotar ] }
```

Vampiro: Se debe guardar `"{ V "`, su clase de vampiro, `" "`, su vida, `" "`, su cuanto pega y `" }"`. Por ejemplo, un vampiro de clase Desviado con 50 de vida y 8 de cuantoPega se guarda:

```
{ V Desviado 50 8 }
```

Nivel: Se debe guardar `"{ N "`, su ancho, `" "`, su alto, `" "`, su turno, `" "`, sus soles, `" "`, su lista de flores en el juego, `" "`, su lista de vampiros en el juego, `" "`, su lista de spawning y `" }"`. A continuación se muestra el ejemplo de un nivel al quedar guardado:

```
{ N 5 5 3 56
  [ ( { F 33 6 [ Atacar Explotar ] } ( 2 3 ) 10 ) ( { F 33 0 [ Generar Explotar ] } ( 2 5 ) 33 ) ]
  [ ( { V Desviado 50 8 } ( 4 5 ) 10 ) ]
  [ ( { V Caminante 30 7 } 2 5 ) ] }
```

Juego: Se debe guardar `"{ J "`, su lista de flores, `" "`, su lista de vampiros, , `" "`, su lista de niveles y `" }"`. A continuación se muestra el ejemplo de un juego al quedar guardado:

```
{ J [ { F 33 6 [ Atacar Explotar ] } { F 33 0 [ Generar Explotar ] } ]
    [ { V Desviado 50 8 } { V Caminante 30 7 } ]
    { N 5 5 3 56
      [ ( { F 33 6 [ Atacar Explotar ] } ( 2 3 ) 10 ) ( { F 33 0 [ Generar Explotar ] } ( 2 5 ) 33 ) ]
      [ ( { V Desviado 50 8 } ( 4 5 ) 10 ) ]
      [ ( { V Caminante 30 7 } 2 5 ) ] } }
```

Aclaración: Tanto las listas, como las tuplas, tienen todas sus componentes separadas por espacios. Además los saltos de linea en los ejemplos de nivel y juego son simplemente para la presentación en este documento.