Fundamentos de la programación

Práctica 4. WALL·E[1]

Indicaciones generales:

- La entrega se realizará a través del campus virtual, subiendo únicamente el archivo fuente `Program.cs`, con el programa completo.
- Las líneas 1 y 2 del programa deben ser comentarios de la forma `// Nombre Apellido1 Apellido2` con los nombres de los alumnos autores de la práctica.
- **Lee atentamente** el enunciado e implementa el programa según el esquema propuesto, con los métodos que se especifican, respetando los parámetros y tipos de los mismos. Pueden implementarse los métodos auxiliares que se consideren oportunos comentando su cometido, parámetros, etc.
- El programa, además de ser correcto, debe estar bien estructurado y comentado. Se valorarán la claridad, la concisión y la eficiencia.
- El **plazo de entrega** finaliza el 10 de mayo de 2018.

Las aventuras conversacionales son un género de juegos de ordenador (también conocido como *Interactive Fiction*[2]) que se caracteriza por establecer un diálogo con el jugador en el que la máquina presenta descripciones (principalmente textuales) de lo que ocurre en el juego y el jugador responde escribiendo órdenes para indicar las acciones que desea realizar. Surgido en la década de los años 70, y abandonado comercialmente a primeros de los tras ser sucedido por las populares aventuras gráficas, este género aún sigue vigente gracias a una fuerte comunidad de aficionados que continúan creando nuevos títulos y mejorando las herramientas para su desarrollo.

El motor de una aventura conversacional es habitualmente un intérprete que recibe como entrada la especificación de una aventura concreta, que incluye datos y descripciones narrativas (a menudo impregnadas de cierto valor literario) sobre las localizaciones del mundo del juego, los objetos y personajes se encuentran allí, etc. El motor procesa dicha especificación creando una experiencia interactiva para que los jugadores se sitúen en ese entorno virtual, lo exploren y traten de superar con éxito los diversos retos que presenta el juego. Para ello el jugador tiene a su disposición un repertorio de acciones posibles a realizar, como moverse de una localización a otra, examinar y usar objetos, hablar con otros personajes, etc.

En esta práctica vamos a implementar la aventura conversacional WALL·E, basada una práctica propuesta en la asignatura de Tecnología de la Programación[3]. El juego consiste controlar el comportamiento de un robot utilizando un pequeño repertorio de instrucciones para moverlo por un mapa recogiendo basura. El mapa consta de un conjunto de lugares conectados por calles que pueden ir en las 4 direcciones básicas (norte, sur, este y oeste). Cada lugar tiene un *nombre* y una *descripción* con información sobre lo que hay en el lugar. Cuando WALL·E llega a un lugar se muestra en pantalla su descripción. Algunos lugares representan la nave del robot, que es su lugar de descanso, donde termina la aventura. Algunos lugares contienen objetos basura que WALL·E debe recoger y almacenar en su inventario. El objetivo del juego es recoger la máxima cantidad de basura antes de alcanzar la nave-salida en el mapa.

Los comandos que entiende WALL·E son los siguientes:

- `go` seguido de una dirección (`north`, `south`, etc). Mueva a WALL·E en la dirección indicada

[1] Esta práctica ha sido diseñada en colaboración con Guillermo Jiménez Díaz. Gracias Guille!

[2] Para saber algo más: `http://en.wikipedia.org/wiki/Interactive_fiction`

[3] Por los profesores P. Arenas, M.A. Gómez, J. Gómez y G. Jiménez.

- `pick`: seguido de un número. WALL·E recoge el ítem indicado.
- `drop`: seguido de un número. WALL·E suelta el ítem indicado.
- `items`: muestra los ítems disponibles en la posición actual.
- `info`: muestra la información del lugar actual.
- `bag`: muestra los ítems del inventario de WALL·E.
- `quit`: termina el juego a petición del jugador.

Los mapas se leen de un archivo con el siguiente formato (ver mapa de ejemplo "madrid.map"):

```
...
place 3 PlazaDeCallao noSpaceShip
"In this small square you can find some some fuel.
Go to fnac and take the fuel for the heating"
...

place 8 PuertaDeAlcala spaceShip
"Ok, finally you have found your spaceship...."
...

street 0 place 0 north place 3
street 1 place 0 south place 9
...

garbage 0 Newspapers1 2 place 0 "News"
garbage 1 MoreNewspapers 6 place 0 "More News"
...
```

Los mapas contienen información de 3 tipos, que se identifica por la primera palabra:

- `place 3 PlazaDeCallao noSpaceShip`: define el lugar 3, que se llama `PlazaDeCallao` y no es salida (`noSpaceShip`). Todo el texto que aparece en las siguientes líneas entrecomillado corresponde a la descripción del lugar. En este mapa el lugar 8 correspondería a una salida.
- `street 0 place 0 north place 3`: define el camino 0, que conecta el lugar 0 al norte, con el lugar 3. Las calles pueden conectar en las 4 direcciones habituales: `north`, `south`, `east`, `west`, y son bidireccionales, es decir, la calle del ejemplo también establece una conexión entre el lugar 3 y el lugar 0 en dirección `south`.
- `garbage 0 Newspapers1 place 0 "News"`: define el ítem basura número 0 con nombre `Newspapers`, en el lugar 0 y con una descripción entrecomillada `"News"` (en este caso el texto de la descripción aparece en la misma línea, a diferencia de lo que ocurre con los lugares).

Los lugares los lugares aparecerán numerados en orden creciente, desde el 0 en adelante, así como las calles y los ítems de basura. Asumimos que el mapa es correcto, es decir, que las calles conectan lugares existentes, los items aparecen lugares también existentes, etc. Además los ítems ubicados en los lugares son únicos, es decir, dos lugares no pueden contener el mismo ítem.

Para implementar el juego vamos a utilizar la clase `Lista` vista en clase, **extendiéndola con los métodos que sean necesarios**. Además definiremos otras clases más, todas ellas en el espacio de nombres WALL·E. La primera es la clase `Map`, para la carga y manejo de los mapas:

```
namespace WallE{
 // posibles direcciones
 public enum Direction {North,South,East,West};

 class Map{
  // items basura
  struct Item{
    public string name, description;
  }

  // lugares del mapa
  struct Place {
    public string name, description;
    public bool spaceShip;
    public int [] connections; // vector de 4 componentes
                               // con el lugar al norte, sur, este y oeste
                               // -1 si no hay conexion
    public Lista itemsInPlace; // liste de enteros, indices al vector de items
  }

  Place [] places;       // vector de lugares del mapa
  Item [] items;         // vector de items del juego
  int nPlaces, nItems;   // numero de lugares y numero de items del mapa
```

Los items basura y los lugares del mapa vienen definidos por el nombre y la descripción. Todos los items del mapa se guardan en el vector `items` (el ítem 0 en la componente 0, el 1 en la componente 1, etc) y los lugares se guardan en el vector `places`. Los lugares contienen además otra información:

- Si el lugar es o no salida (campo `spaceShip`).
- Las calles que salen de ese lugar se guardan mediante un vector `connections`. La componente 0 contiene **el numero del lugar** (referente al vector `places`) con el que está conectada al norte; la componente 1, el lugar al sur al sur, etc. En caso de no haber conexión en una dirección dada, esa componente valdrá -1.
- Los objetos basura que hay en el lugar, que se guardan en una lista `itemsInPlace` de índices que se refieren al vector `items`.

Antes de continuar conviene hacer un **dibujo de esta estructura** y comprender bien cómo se almacena la información de un mapa. Los métodos a implementar para esta clase son:

- `public Map(int numPlaces, int numItems)`: constructora de la clase. Genera un mapa vacío con el número de lugares y de ítems indicados.
- `public void ReadMap(string file)`: lee el mapa del archivo `file` y lo almacena en la estructura descrita anteriormente. Para facilitar la lectura implementaremos tres métodos privados auxiliares: `CreatePlace`, `CreateStreet`, `CreateItem`. Una vez leída la primera línea del ítem y determinado el tipo de ítem, se llamará al método correspondiente que creará el ítem en cuestión (utilizando los parámetros adecuados).

  Para facilitar la lectura, también será útil un método para leer la descripción de los lugares (`private string ReadDescription(StreamReader f)`), que tendrá en cuenta que dicha descripción puede aparecer en varias líneas del archivo de entrada (la descripción va delimitada por comillas, y puede contener saltos de línea).
- `public string GetPlaceInfo(int pl)`: devuelve toda la información del lugar indicado `pl`. Por ejemplo, para el lugar 3, devolverá la cadena (incluidos saltos de línea):

```
In this small square you can find some fuel.
Go to fnac and take the fuel for the heating
```

- `public string GetMoves(int pl)`: devuelve los movimientos posibles desde el lugar `pl`. Por ejemplo, para el lugar 1 (véase el mapa "madrid.map"), devolverá:

  ```
  north: PlazaEspaña
  east: Sol
  ```

- `public int GetNumItems(int pl)`: devuelve el número de ítems que hay en el lugar `pl`.
- `public string GetItemInfo(int it)`: devuelve la información sobre el ítem `it` de la lista de ítems. Para el ítem 0 devuelve:

  ```
  0: Newspapers1 News
  ```

- `public string GetItemsPlace(int pl)`: devuelve la información sobre los ítems que hay en el lugar `pl`. Para el lugar 0 devuelve:

  ```
  0: Newspapers1 News
  1: Newspapers2 More News
  ```

- `public void PickItemPlace(int pl, int it)`: elimina el ítem `it` del lugar `pl`.
- `public void DropItemPlace(int pl, int it)`: deja el ítem `it` en el lugar `pl`.
- `public int Move(int pl, Direction dir)`: devuelve el lugar al que se llega desde el lugar `pl` avanzando en la dirección `dir` (-1 en caso de error).
- `public bool isSpaceShip(int pl)`: comprueba si el lugar `pl` es la nave de WALL·E.

A continuación implementaremos la clase `wallE` para representar el estado de WALL·E. Esta clase tendrá únicamente dos atributos para determinar su posición y los ítems que lleva recogidos:

```
namespace WallE {
    class wallE {
        int pos;   // posicion de Wall-e en el mapa
        Lista bag; // lista de items recogidos por wall-e
                   // (son indices a la lista de items del mapa)
```

Implementará los siguientes métodos:

- `public wallE()`: constructora de la clase, que sitúa a WALL·E en la posición 0 y con una lista vacía de ítems.
- `public int GetPosition()`: devuelve la posición actual de WALL·E.
- `public void Move(Map m, Direction dir)`: mueve a WALL·E en la dirección `dir` a partir de la posición actual de acuerdo con el `Mapa m` (utiliza el método `Move` de la clase `Map`).
- `public void PickItem(Map m, int it)`: recoge el ítem `it` del lugar actual y lo inserta en la lista de ítems de WALL·E.
- `public void DropItem(Map m, int it)`: deja el ítem `it` en el lugar actual de acuerdo con el mapa `m`.

- `public string Bag(Map m)`: devuelve una cadena con la lista de ítems que lleva recogidos WALL·E.
- `public bool atSpaceShip()`: indica si WALL·E está en la nave.

Por último, la clase `Main`, también dentro del *namespace* `WallE`, hace uso de las clases anteriores e implementará los métodos:

- `static void ProcesaInput(string com, wallE w, Map m)`: procesa el comando representado en la cadena `com`.
- `Main`: lee el mapa e implementa el bucle principal del juego. En cada iteración se escribe en pantalla el prompt ">", se lee un comando para WALL·E y se procesa con el método anterior hasta alcanzar un punto de salida o hasta que el jugador aborte el juego (comando `quit`). Al final del juego se indicará la lista de ítems recogidos por WALL·E.

Una vez implementadas las clases descritas, se pide implementar las funcionalidades siguientes:

- Utilizar excepciones para hacer un tratamiento adecuado de los casos inesperados en los métodos definidos.
- Implementar un método que permita leer de archivo una lista de comandos para WALL·E (un comando por línea) y haga que WALL·E ejecute las instrucciones.
- Extender la implementación para poder interrumpir una partida y recuperarla posteriormente. Debe guardarse el estado actual del juego en un formato adecuado, para poder restaurar el estado del juego posteriormente.