Editores y herramientas
Práctica 11 - Carga de mapas

Esta práctica está dividida en dos partes:

Primera parte (0,8 puntos)

Vamos a utilizar la herramienta **Tiled Map Editor** para generar los escenarios de nuestro juego. Para ello, necesitamos incluir soporte para cargar estos mapas y crear una nueva subclase de `Scene` que los maneje.

Esta subclase será `MapScene`, y descenderá de `ParallaxScene`, o de `Scene` directamente si no implementamos soporte para scroll parallax en la práctica de Control de Usuario. Su interfaz es la siguiente:

```
class MapScene : public ParallaxScene {
public:
    MapScene(Map* map, Image* imageBack = 0, Image* imageFront = 0);
    virtual const Map* GetMap() const;

    virtual void Update(double elapsed);
protected:
    virtual void RenderAfterBackground() const;
private:
    Map* map;
};
```

Sus métodos funcionan de la siguiente forma:

- El constructor debe pasar los parámetros apropiados al constructor base y establecer el valor de la variable miembro `map`.
- El método `GetMap` debe devolver el valor de la variable miembro `map`.

- El método `Update` debe llamar al método de la clase padre. En la clase padre, este método tiene dos parámetros, `elapsed` y `map`. Como segundo parámetro le enviaremos el valor de la variable miembro.
- El método `RenderAfterBackground` debe estar redefinido para dibujar el mapa sobre el scroll de fondo. El origen de coordenadas ya ha sido correctamente establecido por la clase `Scene` cuando se llama a este método. Lo que debe hacer simplemente es llamar al método `Render` de la variable miembro `map`.

Los mapas se cargarán y manejarán por medio de la clase `Map`, que tiene la siguiente interfaz:

```
class Map {
public:
    Map(const String& filename, uint16 firstColId = 0);

    virtual const String& GetFilename() const;
    virtual bool IsValid() const;
    virtual void Render() const;
    virtual bool CheckCollision(const Collision* collision) const;
    virtual const Image* GetImage() const;
    virtual Image* GetImage();
    virtual uint32 GetWidth();
    virtual uint32 GetHeight() const;
    virtual uint16 GetTileWidth() const;
    virtual uint16 GetTileHeight() const;
    virtual uint16 GetColumns() const;
    virtual uint16 GetRows() const;
    virtual int32 GetTileId(uint16 column, uint16 row) const;
    virtual double GetGroundY(double x, double y) const;
    virtual uint16 GetFirstColId() const;
private:
    bool valid;
    String filename;
    uint16 width, height;
    uint16 tileWidth, tileHeight;
    String imageFile;
    Image* image;
    Array<int32> tileIds;
    uint16 firstColId;
};
```

Se proporcionan todos los métodos implementados salvo el constructor. En él, debemos generar los datos del mapa cargando el XML con la librería **RapidXML** proporcionada con el motor. Los pasos a seguir en el constructor son los siguientes:

- Guardamos en las variables miembro los valores de `filename` y `firstColId`, y fijamos el valor de `valid` a `false`.

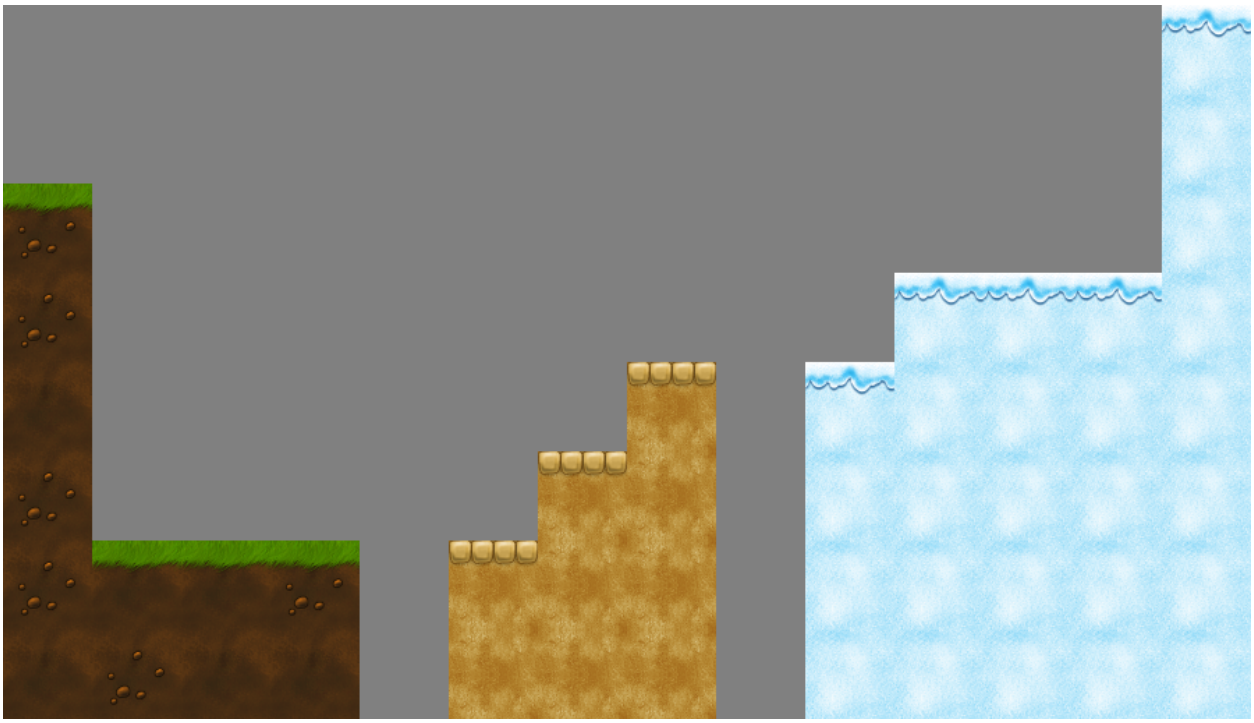
- Cargamos en un objeto `String` el contenido del fichero. Esto lo podemos hacer mediante el método `Read` de dicha clase.
- Generamos un objeto del tipo `xml_document<>`, y llamamos a su método `parse<0>` pasándole como parámetro el array de chars contenido en el string cargado anteriormente (esto lo hacemos mediante su método `ToCString`).
- Obtenemos el primer (y único) elemento (en RapidXML se llaman "nodos") del documento de nombre **"map"**, mediante el método `first_node("map")`, y lo guardamos en una variable de tipo `xml_node<>*`.
- Guardamos los atributos **"width"**, **"height"**, **"tilewidth"** y **"tileheight"** del elemento en las variables miembro correspondientes. Podemos obtener el valor de un atributo con el método `first_attribute(nombre)->value()`. Como se devuelve como un array de caracteres, podemos convertirlo en un entero con la función `atoi`.
- Obtenemos el primer elemento del nodo **"map"** llamado **"tileset"** (si hay más de un tileset, ignoraremos el resto) y lo guardamos en una variable de tipo `xml_node<>*`.
- Obtenemos el valor de los atributos **"firstgid"**, **"tilewidth"**, **"tileheight"** del elemento, y los guardamos en variables de tipo entero.
- Si el elemento "tileset" contiene un elemento **"tileoffset"**, guardamos los atributos **"x"** e **"y"** de éste en variables locales enteras.
- Obtenemos el primer (y único) elemento llamado **"image"** del tileset, y guardamos sus atributos **"source"**, **"width"** y **"height"** en las variables miembro correspondientes. Hay que tener en cuenta que el atributo **"source"** se guarda en la variable miembro `imageFile` sin ruta, cosa que se puede conseguir utilizando el método `StripDir` de la clase `String`.
- Obtenemos el primer elemento **"layer"** del mapa (el resto son ignorados), y de éste el primer (y único) elemento **"data"**, y lo guardamos en una variable de tipo `xml_node<>*`.
- Si el elemento "data" tiene atributos **"encoding"** o **"compression"**, salimos del constructor, ya que la carga de mapas codificados o comprimidos no está soportada.
- Obtenemos el primer elemento **"tile"** del elemento "data", y lo guardamos en una variable de tipo `xml_node<>*`.
- Mientras la variable anterior tenga un valor distinto de `NULL`:
    - Añadimos al vector `tileIds` el valor del atributo **"gid"** del tile. Hay que restarle el valor del atributo **"firstgid"** que habíamos leído anteriormente en el tileset.
    - Obtenemos el siguiente tile mediante el método `next_sibling("tile")` y lo guardamos en la misma variable.
- Por último, vamos a cargar la imagen del tileset. Lo haremos de la siguiente forma:
    - Su nombre de fichero debe ser la ruta al fichero .tmx (que podemos obtener con la función `ExtractDir` de la clase `String`) concatenada con el nombre de la imagen obtenido en el tileset.
    - El número de frames lo obtendremos dividiendo el ancho y alto de la imagen leídos del XML entre el ancho y el alto de tile leídos **en el tileset** respectivamente.
    - El handle de la imagen se debe establecer con los valores de los atributos **"x"** e **"y"** leídos en el elemento **"tileoffset"**.

- Antes de terminar, establecemos el valor de la variable miembro `valid` a `true`, para indicar que hemos cargado correctamente el mapa.

También debemos activar el soporte para mapas en la clase `ResourceManager`.

Con todo esto implementado, realizaremos un ejercicio para probar que los mapas se cargan de manera correcta. Crearemos un `MapScene` con un mapa cargado del fichero **"data/map.tmx"** proporcionado, y añadiremos un sprite con la imagen **"data/alien.png"** que pueda sobrevolar el mapa utilizando los cursores. La cámara debe usar el tamaño del mapa como límites y seguir al sprite.

El mapa debe verse como en la siguiente imagen:



Segunda parte (0,2 puntos)

Si tenemos implementados el método `MoveTo` de la clase `Sprite`, debemos hacer que cuando un sprite está efectuando un movimiento automático detecte colisiones con el mapa.

Con las modificaciones anteriores, el sprite ya recibe automáticamente en el método `update` un puntero al mapa si la escena en la que está incluido es de tipo `MapScene`. En el método `Update`, si está realizando un movimiento automático, debemos guardar la posición del sprite antes de moverlo en las variables miembro `prevX` y `prevY`. Movemos primero en el eje x, actualizamos la caja de colisión (método `UpdateCollisionBox` del sprite), y comprobamos con el método `CheckCollision` del sprite si colisiona con el mapa. Si es así, debemos volver a la

posición `prevX`. Repetimos el proceso con el eje Y. Si no ha podido moverse ni en x ni en y, debemos abortar el movimiento automático.

No hay que modificar el ejercicio de pruebas, simplemente asegurarse de que ahora el alien colisiona con el escenario (estableciendo su modo de colisión correctamente).