Universidad de Murcia

Facultad de Informática

IA para el Desarrollo de Videojuegos

Real Time Wargame

Autores

Antonio López Martínez-Carrasco

antonio.lopez31@um.es

José María Sánchez Salas

josemaria.sanchez12@um.es

Profesores

Francisco Javier Martín-Blázquez Gómez

jgmarin@um.es

Luis Daniel Hernández Molinero

ldaniel@um.es

Índice

I Introducción y estructura de la aplicación 3

1 Introducción 3

2 Estructura de la aplicación 3

II Manual de uso 4

3 Manual de uso 4

III Clase principal y mapa 5

4 Clase principal 5

5 Mapa 5

IV IA reactiva 7

6 Steerings 7

7 Comportamientos 7

8 Árbitros 7

9 Modelo 7

V PathFinding 8

10 PathFinding 8

VI IA táctica 9

VII Conclusiones 10

11 Conclusiones 10

VIII Bibliografía 11

Parte I
Introducción y estructura de la aplicación

1 Introducción

2 Estructura de la aplicación

Parte II
Manual de uso

3 Manual de uso

Parte III
Clase principal y mapa

4 Clase principal

Todo lo referente a esta sección, se encuentra dentro del paquete `com.mygdx.iadevproject` del proyecto. La clase principal de nuestro proyecto es la clase `IADeVProject` y es la que contiene los siguientes elementos:

- **Constantes**. Estas son: el tamaño del mapa (alto y ancho); el tamaño de las celdas de los grids, así como el tamaño de estos (alto y ancho); valores de infinito, coste y terreno por defecto; y el tamaño de los objetos del mundo (alto y ancho, obtenidos a partir de la información del mapa).
- **Mapas**. Estos son: el mapa real que se dibuja; el mapa de costes para el PathFinding; el mapa de terrenos para el PathFinding táctico (exceptuando el mapa real, todos los demás son los que se denominan grids y todos tienen el mismo tamaño).
- **Variables globales**. Estas son: los objetos y obstáculos del mundo; el conjunto de objetos seleccionados por el usuario (utilizando el ratón); la cámara; el flag que indica si se dibujan, o no, las líneas de depuración de los distintos comportamientos; así como las variables necesarias para que desde los comportamientos se puedan dibujar las líneas de depuración.

Esta clase se encarga de crear e inicializar todos los elementos anteriores, de renderizar el seguimiento del juego y, una vez terminado, eliminar todos los elementos creados para el renderizado.

El control de la interacción del usuario con el juego se ha implementado en la clase `InputProcessorIADeVProject`, que contiene todos los manejadores para las posibles acciones que puede realizar el usuario con el teclado y el ratón. Debido a que el usuario puede seleccionar objetos del mundo, cuando se realiza la selección para que añadan los objetos al conjunto de objetos seleccionados, se hace uso del método `addToSelectedObjectsList()` de la clase `IADeVProject`. Como también se proporciona la funcionalidad de eliminar la selección de objetos, dicha clase también proporciona el método `clearSelectedObjectsList()`.

Debido a que el tamaño de los distintos grids y el tamaño del mapa pueden no ser iguales (pues depende del tamaño de las celdas), es necesario hacer una conversión entre las posiciones reales del mapa y las posiciones correspondientes del grid. Por eso mismo, la clase `IADeVProject` proporciona los dos siguientes métodos:

- `mapPositionTOgridPosition() y gridPositionTOmapPosition()`. Ambos reciben como parámetros el tamaño de la celda del grid y un vector con la posición que se quiere convertir. El primero transforma una posición del mapa a una posición del grid, y el segundo hace la inversa: transforma una posición del grid a una posición del mapa.

5 Mapa

Todo lo referente a esta sección, se encuentra dentro del paquete `com.mygdx.iadevproject.map` del proyecto. El mapa que se ha diseñado para este proyecto se muestra en la Figura 1.

Como se puede observar, existen seis tipos de terrenos:

- **Agua y montañas**. Representan los terrenos infranqueables. Como indica el enunciado del proyecto, los dos países se encuentran separados por un río con tres puentes para cruzarlo.
- **Desierto y bosque**. Se identifican de manera directa en el mapa. También se tiene como bosque los árboles que se encuentran dentro la pradera situada en la parte inferior izquierda del mapa.
- **Pradera**. Se corresponde con los tramos de color verde más claro.
- **Sendero**. Se corresponde con el fondo del mapa.

Figura 1: Mapa diseñado para el proyecto.

- **Camino**. Se corresponde con los tramos de color gris. Las bases de los países se encuentran encima de este tipo de terreno.

Para representar los terrenos en el proyecto, se ha implementado el enumerado `Ground` que proporciona los siguientes métodos:

- `getCost()`. Debido a que cada terreno tiene un coste asociado, este método devuelve dicho coste.
- `getGround(int cost)`. Se trata de un método estático que dado un coste, devuelve el terreno al que pertenece. Si el coste que se pasa como argumento no se corresponde con ningún terreno, devuelve `null`.

Para diseñar el mapa, se ha hecho uso de la herramienta `Tiled Map Editor` [1] que proporciona una sencilla forma de diseñar mapas por capas, permitiendo que puedas crear capas de distintos tipos (de tiles, de objetos, etc), lo que nos facilita la creación e inicialización de los distintos grids que utilizamos. Lógicamente, las capas que se encuentran en el mapa, se corresponden con los distintos tipos de terrenos mencionados anteriormente (excluyendo la capa de objetos).

Para introducir el mapa diseñado en el proyecto, `LibGDX` nos proporciona la clase `TiledMap`. Sin embargo, esta clase, a la hora de renderizar, si existe alguna capa de objetos por defecto, no la dibuja. Por este motivo, implementamos la clase `TiledMapIADeVProject`, que sobreescribe los métodos de renderizado para que se dibujen las capas de objetos.

Una vez que tenemos el mapa diseñado e introducido en nuestro proyecto, hay que inicializar los distintos grids en correspondencia con dicho mapa. Para ello, está la clase `MapsCreatorIADeVProject`, que proporciona un único método estático `createMaps()` y que se encarga de recorrer, para cada terreno, su capa correspondiente e inicializar los grids a los valores correspondientes.

Parte IV
IA reactiva

6 Steerings

7 Comportamientos

8 Árbitros

Todo lo referente a esta sección, se encuentra dentro del paquete `com.mygdx.iadevproject.aiReactive.arbitrator` del proyecto. Para generalizar el uso de los árbitros, se ha decidido crear la interfaz `Arbitrator` que tienen que implementar todos los distintos tipos de árbitros. Esta interfaz proporciona un solo método:

- `getSteering()`, que recibe como parámetro un objeto del tipo `Map<Float, Behaviour>` que contiene el conjunto de comportamientos (con sus valores de importancia correspondientes) del que se quiere obtener un steering final.

Dependiendo del tipo de árbitro que utilicemos, este método devolverá un steering u otro. Los distintos tipos de árbitros que se han implementado son los siguientes:

- **Árbitro de mezcla ponderada**. Este tipo de árbitro, lo que hace es obtener un steering final como resultado de la mezcla de todos los steerings obtenidos por el conjunto de comportamientos, de manera ponderada. Es decir, para cada comportamiento, se obtiene su steering y el resultado del mismo se añade al steering final multiplicado por el valor de importancia asociado al comportamiento (de ahí que se reciba un objeto de tipo `Map<Float, Behaviour>` que para cada comportamiento se tiene su valor de importancia asociado). Debido a que hay dos tipos de steerings, como se ha comentado en la sección 6, se han implementado dos tipos de árbitros de mezcla ponderada: uno para los comportamientos que devuelve steerings acelerados (clase `WeightedBlendArbitrator_Accelerated`) y otro para los comportamientos que devuelven steerings no acelerados (clase `WeightedBlendArbitrator_NoAccelerated`). El funcionamiento es el mismo en ambos casos, solamente que si un comportamiento devuelve un steering del otro tipo, no se tiene en cuenta.

  Ambos árbitros, en su constructor, reciben como parámetros la máxima aceleración y la máxima rotación (en el caso del acelerado), y la máxima velocidad y máxima orientación (en el caso del no acelerado) que dicho árbitro tiene permitido devolver. Por lo que, una vez que se ha calculado el steering final, se comprueba que el valor del steering no supere ninguno de los valores anteriores.

- **Árbitro de prioridad**. Este tipo de árbitro considera que el conjunto de comportamientos recibido como parámetro se encuentra ordenado por prioridad, es decir: el primer objeto del conjunto es el que tiene mayor prioridad. Así pues, recorre dicho conjunto, obteniendo para el comportamiento actual su steering y si este steering es válido (es decir, es distinto de `null` y supera un determinado valor `epsilon`), el árbitro directamente devuelve este steering y termina. Si resulta que de todo el conjunto de comportamientos, ninguno es válido porque no ha superado el valor `epsilon`, el árbitro devuelve, en este caso, el steering obtenido por el último comportamiento del conjunto, independientemente del valor que tenga.

  A diferencia del árbitro anterior, con este árbitro se puede usar comportamientos que devuelvan tanto steerings acelerados como no acelerados; lo que implica que el árbitro puede devolver cualquier tipo de steering. El determinado valor `epsilon` es un valor que se le pasa al árbitro en su constructor, y que indica el valor mínimo que un steering debe de tener para que se considere válido.

9 Modelo

Parte V
PathFinding

10 PathFinding

Parte VI
IA táctica

Parte VII
Conclusiones

11 Conclusiones

Parte VIII
Bibliografía

Bibliografía

[1] Thorbørn Lindeijer et al., *Tiled Map Editor.* Enlace (última visita 06/05/2017).