Universidad de Huelva

Universidad de Huelva

Grado en Ingeniería Informática

Programación de Juegos 3D en Unity

Memoria

Autor: Alberto Fernández Merchán
Asignatura: Programación de Juegos
Profesor: Javier Martín Moreno

Índice

Índice de figuras

1. Introducción

Unity es una de las herramientas más utilizadas por los desarrolladores de videojuegos debido a su facilidad de uso y la capacidad de crear juegos para múltiples plataformas. En esta segunda práctica utilizaremos este motor para implementar un videojuego en tres dimensiones llamado *Tower Defense 3D*.

Tower Defense 3D es un videojuego del estilo *Tower Defense* que consiste en pasar de rondas sin que la barra de salud de la torre llegue a 0. En cada ronda irán saliendo hordas de enemigos que seguirán un camino generado mediante un algoritmo genético hasta llegar a la torre. La misión del jugador será colocar cañones al lado del camino para que ataquen a los enemigos antes de que lleguen a la torre y puedan bajarle la vida.

Finalmente, una vez se le haya acabado la barra de salud a la torre, se abrirá una escena con el menú de puntuación para guardar los puntos que el jugador haya obtenido en la partida.

En esta memoria, se describirán las implementaciones más interesantes a la hora de programar la lógica del juego. Teniendo en cuenta que los menús y el sistema de puntuación ya se explicaron en la práctica anterior, en esta nos centraremos en la utilización del algoritmo genético para generar el mapa, la lógica de los enemigos para seguir el camino y otros aspectos nuevos que he implementado en esta práctica.

2. Generación del Mapa

En esta sección hablaremos sobre cómo se genera el mapa donde se juega la partida. Para generarlo he implementado un algoritmo genético generacional en el que se reemplaza la población completa después de cada generación. A continuación se explica cada uno de los componentes de dicho algoritmo y, finalmente, se realiza un estudio de como modifica el mapa cada uno de los parámetros que se utilizan.

2.1. Algoritmo Genético

2.1.1. Cromosoma

El cromosoma que se ha implementado para utilizar este algoritmo genético es un vector *booleano* de tantas posiciones como casillas tenga el mapa. En caso de ser *True*, la casilla tendrá un obstáculo y, en caso contrario, la casilla estará libre.

Figura 1: Mapa generado mediante un algoritmo genético con un cromosoma de dimensiones 15x15

2.1.2. Selección

Para la selección, se ha utilizado el método de selección de ruleta proporcional que consiste en generar un número aleatorio entre 0 y la suma del fitness total de toda la población. De esta forma, sustraemos del número aleatorio el fitness de cada uno de los individuos y seleccionamos para cruzar el individuo que haga

dicha resta menor o igual a 0.

2.1.3. Cruce

En cuanto a la función de cruce se ha escogido un cruce en dos puntos. Este cruce consiste en generar un número aleatorio entre 0 y la longitud del cromosoma. Se dividen los cromosomas de los padres por dicho punto y se intercambia cada parte de los padres para generar los dos nuevos hijos que formarán parte de la nueva población.

2.1.4. Mutación

Para la función de mutación se accede a un elemento aleatorio del cromosoma y se niega dicho valor. De esta forma podemos cambiar levemente el valor de fitness del individuo que ha mutado e introducir más diversidad en la población.
Esto puede provocar que el camino quede obstruido o que el punto de inicio y fin se queden libres. Para solucionar esto, se ha implementado una función de reparación del camino que reconstruye el camino final una vez se haya generado el mapa definitivo.

2.1.5. Parámetros del Algoritmo

A continuación se muestran todos los parámetros que pueden ajustarse para utilizar el algoritmo genético:

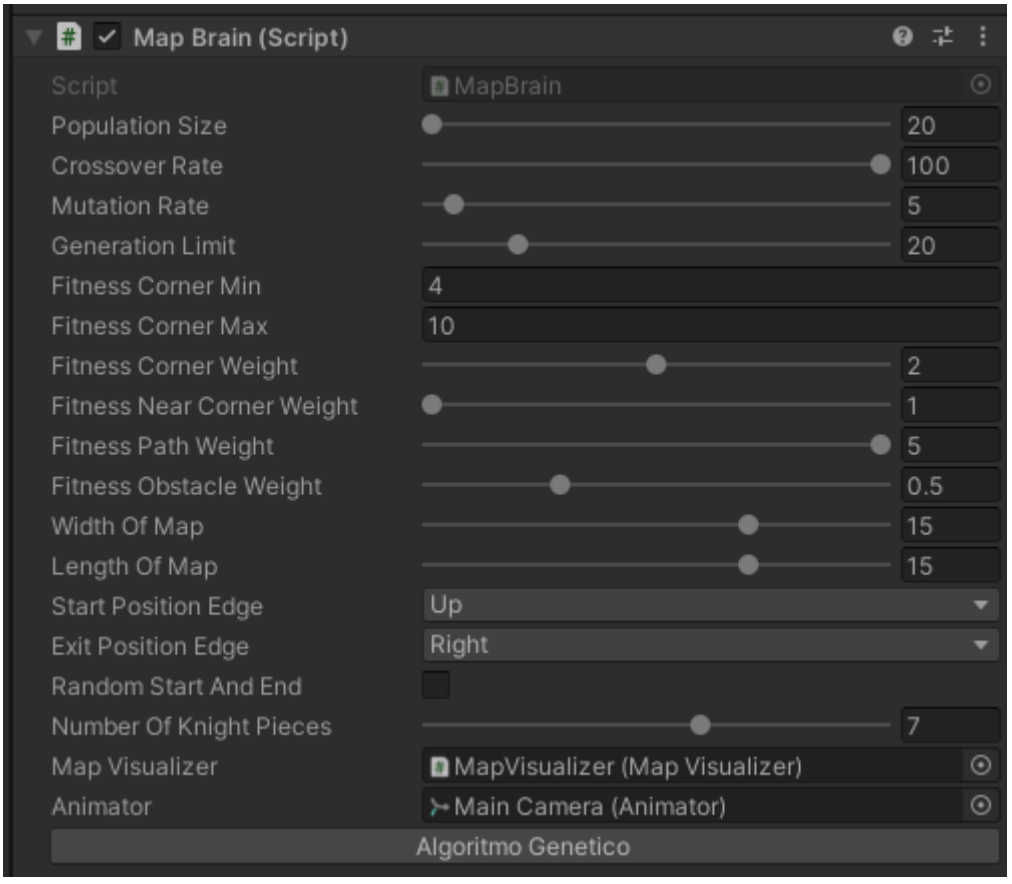


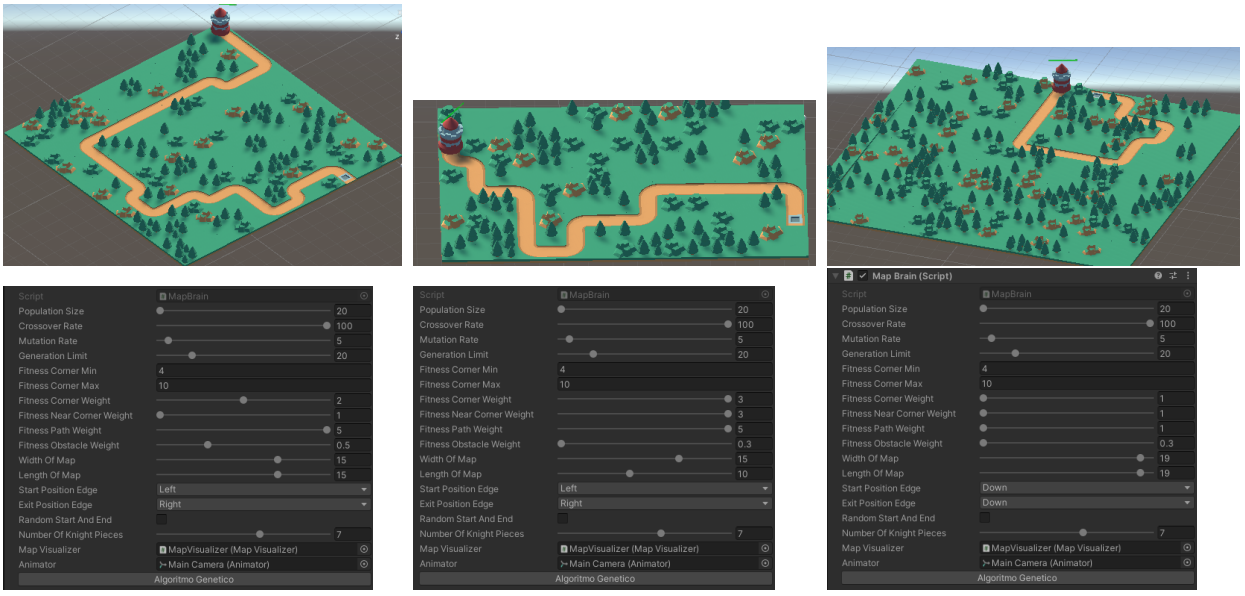
Figura 2: Parámetros que se pueden modificar en el algoritmo genético

Tamaño de la Población. En este caso se ha decidido poner un tamaño de 20 individuos debido a que si se aumentase este número, el tiempo de espera en la carga del mapa sería excesivamente grande. Con 20 individuos genera un mapa coherente y suficiente para comenzar el juego sin tener que esperar una gran cantidad de tiempo.

Longitud del Camino. En cuanto al parámetro de longitud del camino, se le ha dado un peso de 5 ya que nos interesa que el algoritmo priorice los caminos más largos para que los enemigos tarden un tiempo en avanzar hasta el final del juego.

Número de Esquinas. Para que el camino no sea recto y sea más natural es necesario añadir un número de esquinas determinado. También hay que tener en cuenta que si hay muchas esquinas cerca el camino perdería realismo y, por tanto, se penalizará si sucede.

Posición de comienzo y fin Por último, también se pueden modificar en qué lateral del mapa se encuentra la posición de comienzo y fin del camino. En las imágenes anteriores podemos observar diferentes



Cuadro 1: Diferentes Mapas utilizando el algoritmo genético

configuraciones para el algoritmo genético que genera el mapa. Cabe destacar la tercera imagen en la que tanto el punto de inicio como el punto final están en el mismo lado del mapa y, además, el peso que tiene la longitud del camino es el mínimo. De esta forma se genera un camino demasiado corto que no nos convendría.

Es por esto que hay que experimentar con los diferentes parámetros de este algoritmo y generar un mapa acorde con las necesidades del jugador.

3. Inteligencia de los enemigos

Los enemigos del juego son unos *slimes* rojos que se mueven dando saltos a través del camino generado en el mapa. Su único objetivo es atacar a la torre del final del camino. Para alcanzarla se mueven a través de un array de puntos en el mapa que representa el centro de cada casilla del camino.

Es una inteligencia básica que consiste en seguir unos puntos preestablecidos, sin embargo, en esta clase de juegos, los enemigos se rigen por esta forma de *pathfinding*.

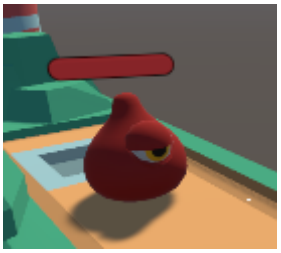
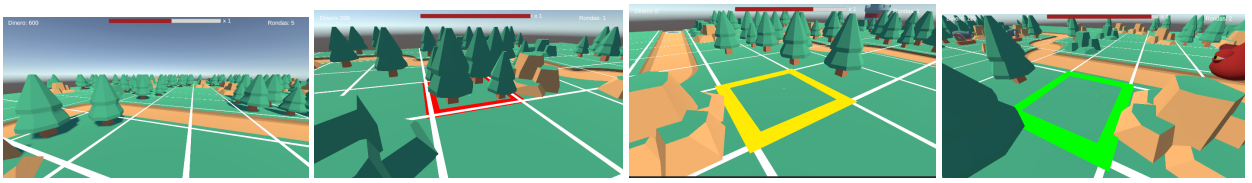


Figura 3: Enemigo del juego

En cada ronda, el número de enemigos que aparecen se incrementa. De esta forma se espera que, después de superar cada ronda, la dificultad se incremente gradualmente.

4. Mecánica de Construcción

Para colocar las torretas se utiliza la tecla **B** y se nos generará una rejilla en el mapa que nos indicará dónde podemos colocar los cañones que dispararán a los enemigos.



Cuadro 2: Rejilla de Construcción

En las imágenes anteriores podemos observar los casos que podemos encontrar a la hora de colocar un cañón. En la primera imagen podemos ver la rejilla completamente sin seleccionar ninguna casilla.

En la segunda imagen vemos que el color de selección se convierte en rojo debido a que estamos seleccionando una casilla que ya está ocupada por un obstáculo. Esto se extiende a elementos decorativos (árboles, montículos, piedras...), casillas del camino y a los propios cañones.

En la tercera imagen podemos ver que el color de selección se convierte en amarillo indicando que podemos construir allí, pero no nos dejará porque no tenemos el dinero suficiente como para colocar una torreta.

Por último, una vez obtengamos el dinero suficiente, la selección será de color verde indicando que se nos permite colocar un cañón en esa casilla.

Para salir del modo de construcción debemos pulsar *X* o *F1* para cambiar la vista a la vista aérea.

5. Futuras Mejoras

Esta práctica ha implementado una gran cantidad de elementos nuevos respecto a la práctica anterior. Sin embargo, aún podría mejorarse para aumentar el entretenimiento y la experiencia de juego. Algunos de estos elementos que podrían implementarse en un futuro para mejorar el videojuego pueden ser los siguientes:

- Añadir un modelo 3D para el jugador y eliminar la cápsula que se ve desde la vista aérea.
- Añadir un *SkyBox* para cambiar el que viene por defecto en Unity.
- Añadir más torretas con diferente rango, daño y coste.
- Añadir nuevos enemigos con diferente velocidad, daño y salud.

- Añadir en las opciones los diferentes parámetros para modificar el algoritmo genético y permitir al jugador experimentar con dichos parámetros.
- Implementar un *joystick* para jugarlo en dispositivos móviles.

6. Conclusiones

En esta práctica he implementado un juego del estilo *Tower Defense* con una novedad que es la jugabilidad en primera persona. Este videojuego hace uso de un algoritmo genético para generar un mapa de juego que se adapte a las necesidades del nivel.

Hemos visto como es posible adaptar los algoritmos de optimización que hemos visto en otras asignaturas en problemas enfocados en videojuegos como generar mapas de forma procedural. Además, también he podido experimentar con los diferentes parámetros del algoritmo genético para entender mejor como funciona y adaptarlo al videojuego.

Finalmente, creo que ha quedado un videojuego original que se puede jugar de forma casual ya que sus partidas son rápidas. Esta característica sería ideal para implementarlo en dispositivos móviles añadiendo más funcionalidades como nuevos mapas, nuevos enemigos y otras características.

Como conclusión personal, me ha parecido más fácil que la práctica anterior ya que una vez hecho un juego en dos dimensiones, ha sido más fácil de implementar uno en tres dimensiones debido a que los problemas que tuve en la práctica anterior ya he sabido resolverlos rápidamente. Además, me ha gustado más el resultado obtenido en este juego ya que es más original que el anterior.