SNAKE

por Ferran Illa y Marc Ruiz
Grupo 14 Clase A

TABLA DE CONTENIDO

1.JUEGO

1.1. Selección de juego

El juego seleccionado para desarrollar la práctica ha sido el Snake. Después de realizar una valoración superficial de los 3 juegos a escoger sobre la cantidad de clases que podría contener cada uno, cómo se interrelacionarían entre sí, cantidad de código requerido para crear la lógica funcional del juego...se concluyó que el Snake nos parecía el reto más adecuado a nuestro nivel de programación para el tiempo que se nos dió.
El segundo motivo fue que los dos miembros congeniamos en que nos parecía de todos, el juego màs simple (visto desde de la producción) y divertido, tanto a nivel de realización como de jugabilidad, ademàs de ser el juego que conocemos en más profundidad

1.2. Adaptaciones/cambios/mejoras respecto el juego original

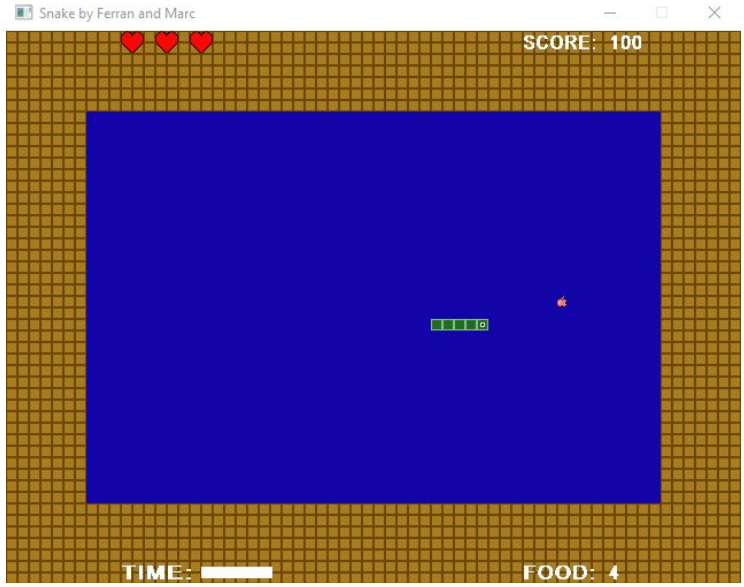
Nuestro Snake contiene tres dificultades, en cada una de ellas la velocidad y las dimensiones del plano delimitado varían de forma que el jugador experimente una dificultad más acorde a su habilidad. Cada dificultad consta de un único nivel infinito, que se termina cuando el contador de vidas del jugador llegue a 0. el jugador, al igual que en el juego original. Cada vez que choque con una parte de su propio cuerpo o con las paredes que delimitan el espacio se pierde una vida. A diferencia del original, este no consta de paredes en forma de obstàculos en el interior del espacio delimitado, al igual que tampoco contiene una cuenta atrás que limite el tiempo para consumir alimentos. Por último en cuanto a jugabilidad se refiere, el jugador tendrá una cantidad de vidas que disminuirán cada vez que colisione, .
Respecto el control, idéntico al original, al jugador se le da la mecánica de moverse en una de las 4 direcciones (arriba, abajo, derecha e izquierda).
Podemos afirmar por lo tanto que nuestra adaptación contiene como único cambio el poder seleccionar una dificultad antes de entrar en la escena de juego.

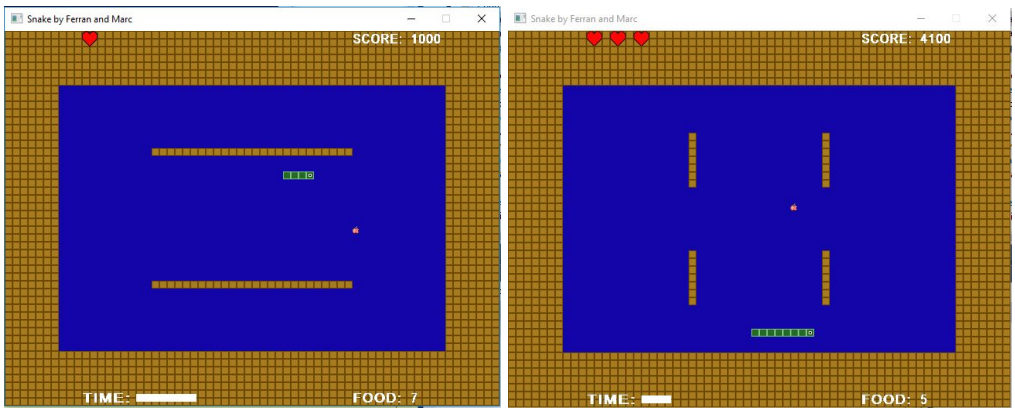
1.3. Pantallas

Solamente distinguimos 2 pantallas en el juego:

- La del menú, aparece en una nueva pantalla automáticamente después de ejecutar el código. En esta pantalla el jugador ve y elige la dificultad con la que quiere jugar. También hay una opción que te ofrece salir de la ventana (EXIT) i cerrar el juego. No hay ningún GameObject y aparte de las etiquetas de las opciones dichas solo hay un cursor para indicar al jugador la opción que quiere escoger.
- La de juego: esta se carga una vez se selecciona cualquiera de las dificultades, en la misma ventana aparecen todos los GameObjects y el jugador ya puede empezara jugar. En esta pantalla está todo lo que conforma parte del juego: el snake, la delimitación del mapa y el alimento a consumir al igual que la cantidad de vidas, la

puntuación y la cuenta atràs. Dentro hay 3 pantallas de juego distintas que van rotando cuando se consiguen los alimentos requeridos para cada uno.

- 



2.MODELO: Game Objects & Scenes

2.1 Game objects & Scenes

Diagrama de clases

clase Snake, clase Wall, clase Food, clase Level cada una por separado. Son clases hermanas. Hay un gráfico del diagrama de clases más adelante en éste documento.

Estructuras de datos usadas:

Clases: para representar los distintos Game Objects y definir su comportamiento de manera ordenada. Todas se utilizan como singleton.

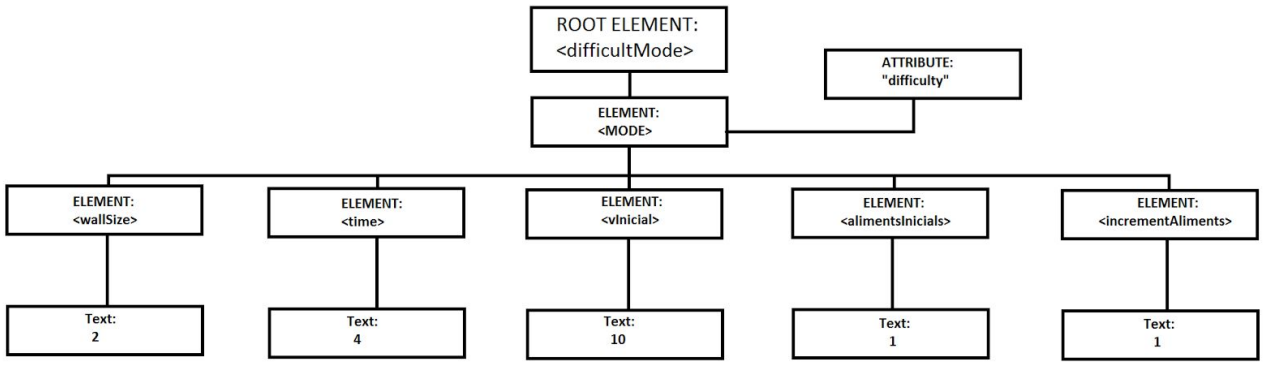
Struct: para contener grupos de datos de tamaño menor como las coordenadas de los elementos en la tabla de juego.

vector: para almacenar cadenas de elementos y tener un control más directo sobre ellos que con un array normal. El grupo de coordenadas que ocupa un mismo Game Object lo representamos con vector (como en el caso de la pared o la propia serpiente).

Map: para almacenar el conjunto de “game objects” junto con sus respectivos sprites.

2.2 XML

El fichero XML de nuestro juego es de la siguiente forma:



El doc.xml contiene un “root element” como nodo principal del documento definido como <difficultMode>, del que se ramifican 3 “childs nodes” (en el esquema solo se representa uno porque són de la misma estructura). Cada “child node” definido como <MODE> representa una dificultad distinta que contiene todos los parámetros específicos de dificultad citados en el documento del GameProject. Para saber cual de ellos necesitamos en la “GameScene” del juego le asignamos un atributo a cada uno con el nombre “diffyculty” que tendrá como valor “easy”/”medium”/”hard” respectivamente, así al recorrer el documento XML podremos identificar el “child node” en cuestión según la dificultad deseada. Aún en el nodo <MODE>, cada uno contendrá los mismos elementos pero con diferente texto. De esta forma desde Visual Studio podremos a acceder al nodo <MODE> con el valor del atributo que se desee y seguidamente cogeremos cada nodo hijo de <MODE>, esto es: <wallSize>, <time> y sus respectivos nodos hermanos. De esta forma parseamos todo el nodo y leemos el texto de cada tipo de variable que serán convertidas a Integers desde nuevas variables creadas en la función que leerá el XML mediante lenguaje DOM (lenguaje que permite lectura del XML en forma de àrbol) previamente de realizar la “GameScene”.
En resumen, simplemente hacemos una lectura del archivo sin necesidad de sobreescribir en él desde Visual Studio.

2.3 Uso y/o modificación de la clase SceneManager

No hemos usado una clase SceneManager. Se accede a la escena menú y la escena de juego de forma lineal.

3.CONTROLADOR: Game

3.1 Game. Atributos y métodos

La lógica del juego está implementada de forma lineal en la cabecera GameEngine. Desde ahí se accede a los métodos propios de cada Game Object.

3.2 Eventos

snakeCollides: a cada frame comprueba el primer elemento del snake, si las coordenades son las mismas, detecta colisión.
snakeEats: lo mismo que snake collides, en caso snakeEats TRUE desaparece el Game Object del alimento respawnea en otra posición del plano ,además la Snake aumenta de tamaño.
keyboardInput: recibe un Input a través de teclado, dependiendo la tecla habrà una acción o otra. Se utiliza la librería SDL.
readDifficultyXml:función que realiza una lectura del doc. XML y actualiza variables con las estadísticas requeridas.

3.3 Diagrama de clases

No usamos clases para el apartado Controlador.

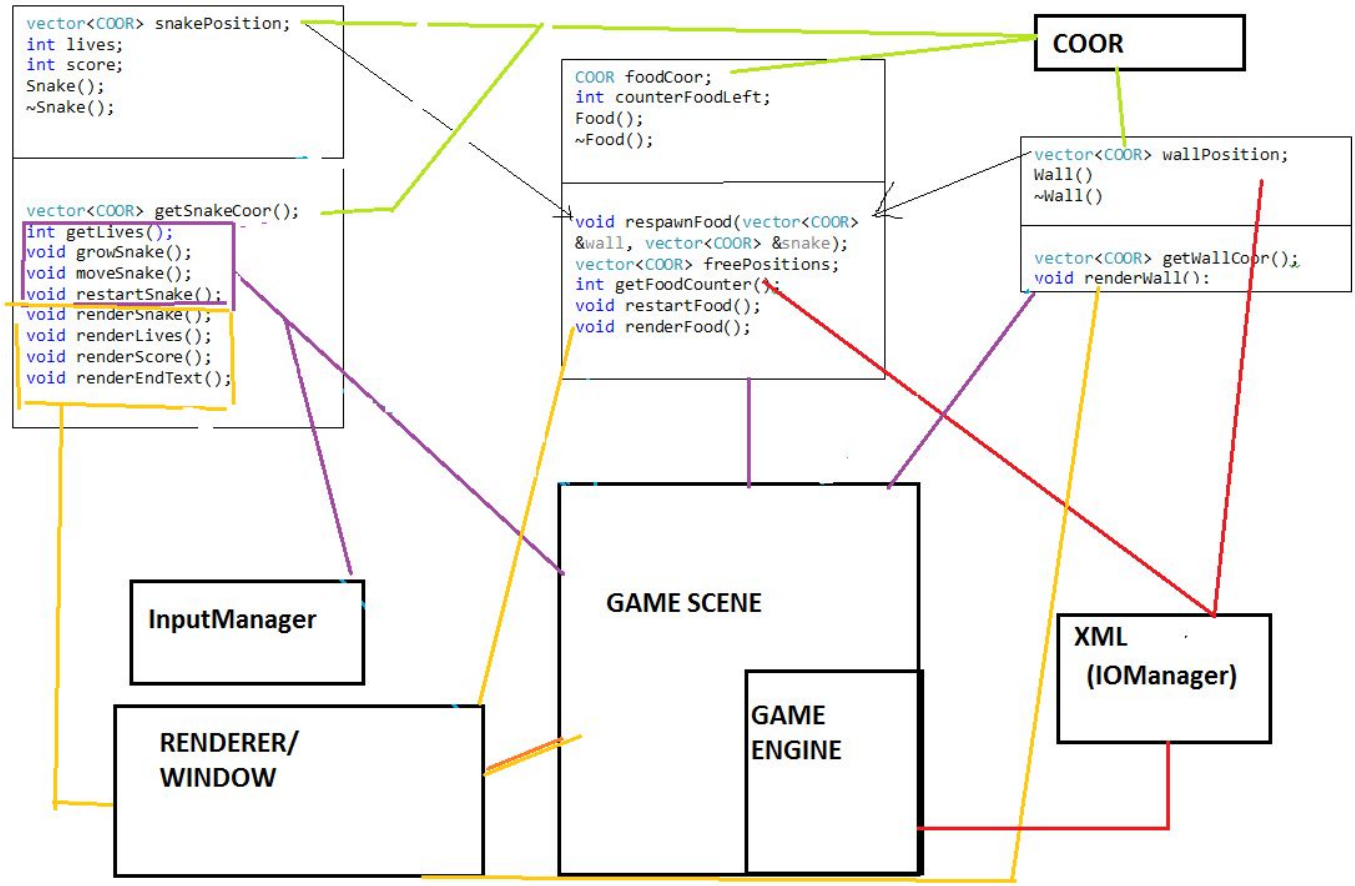
3.4 Uso y/o modificación de la clase InputManager

No hemos usado una clase InputManager. En vez de eso, usamos un evento que gestiona los inputs del teclado para controlar el juego.

4.VISTA:Renderer

Hemos usado la clase Renderer del proyecto de Enti Crush como guia. Tiene los mismos atributos y métodos. Simplemente hemos hecho cambios en algunas declaraciones con fines de clarificación. También hemos cambiado el color de fondo por defecto del renderer.

5. Diagrama de clases global



6. Deployment

(Especificar qué se necesita tener instalado en el PC para jugar fuera de Visual Studio 2015)
No lo sabemos

7. Estimación de tiempo

Los dos miembros hemos aportado en el proyecto como hemos podido. Nos nos hemos dividido el trabajo de forma equitativa ya que cuando acabáramos un apartado empezábamos otro, da igual quien fuera. Sí que Ferran se ha dedicado más a SDL y a la lógica del juego, mientras que Marc se encargaba de XML y el ranking de puntuación con archivos binarios.

Hemos perdido la cuenta de las horas dedicadas a cada fase, pero está claro que hemos dedicado mucho más tiempo a la fase 1. La fase 3 es la fase en la que menos tiempo

hemos dedicado, pero por límite de tiempo en la entrega. En general, hemos invertido unas 6 horas todos los días durante todas las navidades (3 semanas), de media entre los dos. En total cerca de 126 horas cada uno.

8. Conclusiones

Lecciones aprendidas: input/output de archivos, librería SDL. Sobretodo C++ en general.
Satisfacción: el proyecto en sí ha sido un logro para nosotros después de tantas horas, fuerzas y dedicación.
Qué fue lo más difícil de entender y/o de implementar: como organizar el proyector en lo que se refiere a estructura interna.
Fortalezas/debilidades de su software: Es compacto y completo. Es bastante claro y fácil de modificar aunque no se corresponde completamente a una estructura de Model-Vista-Controlador y eso puede hacer que cueste más de leer para las personas que no han desarrollado el software. Cumple con la guia de estilo comentada en clase.

9 Referencias

Requisitos básicos
La serpiente se mueve mediante teclado correctamente.
El jugador tiene "k" vidas y al perderlas termina la partida.
Aparece 1 alimento aleatorio en escena cada vez que la serpiente se come uno.
La serpiente incrementa su tamaño al comer alimentos y se forma una cadena contínua.
La serpiente colisiona con los límites de la pantalla y se pierde 1 vida.
La serpiente colisiona con su propio cuerpo y se pierde 1 vida.
Existe una barra de tiempo que disminuye progresivamente y al llegar a 0 el jugador pierde 1 vida.
Se puede pasar de nivel al consumir "x" alimentos.
Cada nivel sigue la fórmula de “x+y*n” alimentos, donde "x" es el número de alimentos, "n" es el número del nivel actual (suponiendo que nivel 1 es n = 0) e “y” es la cantidad de alimentos a añadir según la dificultad del juego.
La puntuación se suma correctamente según la fórmula “q*100” (siendo “q” el número del alimento en el nivel).
Aumenta la velocidad de la serpiente según la puntuación.
La matriz de casillas varía según la dificultad.
El tiempo inicial de juego varía según la dificultad.
La velocidad inicial de la serpiente varía según la dificultad.
El número inicial de alimentos varía según la dificultad.
El número incremental de alimentos varía según la dificultad.
Al perder una vida, se reinicia el nivel con el mismo tamaño que tenía la serpiente al pasar de nivel.
Requisitos adicionales
Cada nivel consta de obstáculos diferentes distribuidos por el mundo de juego.
La serpiente puede colisionar con los obstáculos y se pierde 1 vida.
Cada patrón de obstáculos está creado previamente en código o se carga desde archivo de texto plano.

Color verde: requisitos hechos.
Color rojo: requisitos no alcanzados