

SNAKE.

Nombre de juego

Snake

Nombre de los integrantes del grupo

Guillem Rodríguez i Arnau Ruiz

Número de grupo de prácticas

17

Grupo de clase (A o B)

A

Tabla de contenidos

1. Juego	2
1.1. Selección del juego (juego y el porqué de la elección)	2
1.2. Adaptaciones, cambios o mejoras con respecto al juego original	2
1.3. Pantallas	2
2.Modelo: Game Objects & Scenes	3
2.1 Game Objects y escenas	3
→ Diagrama de clases	3
→ Explicación breve de las estructuras de datos usadas y el porqué	3
2.2 Diseño del fichero XML de configuración del juego	3
→ Estructura.	3
2.3 Uso y/o modificación de la clase SceneManager	3
3. Controlador: Game	3
3.1 Game. Atributos y métodos	3
3.2 Eventos	4
3.3 Diagrama de clases	4
3.4 Uso y/o modificación de la clase InputManager	4
4. Vista: Renderer	4
→ Uso y/o modificación de la clase Renderer	4
5. Diagrama de clases de todo el proyecto	4
6. Deployment	4
7. Estimación de tiempo	5
8. Conclusiones	5
→ Lecciones aprendidas	5
→ Satisfacción	5
→ Qué fue lo más difícil de entender y/o de implementar	6
→ Fortalezas/debilidades de su software Referencias (Bibliografía, páginas web, etc.)	6
9.Annexo	7

1. Juego

1.1. Selección del juego (juego y el porqué de la elección)

El juego que hemos elegido ha sido Snake. Primeramente por ser el que más conocemos entre las tres opciones de juegos facilitadas. También por hacer uso de cuatro direcciones en dos dimensiones sin necesidad de nada más que cuatro teclas para poder jugar, como si se usara una cruceta de un controlador.

1.2. Adaptaciones, cambios o mejoras con respecto al juego original

En nuestra versión de Snake penalizamos a los jugadores apresurados, aquellos que pulsen más de una tecla entre las 4 direcciones posibles podrían verse con una muerte inminente. Esto ocurre si pulsan una dirección opuesta inmediatamente después de pulsar otra dirección, antes de que la serpiente avance de casilla. De esta forma recompensamos al jugador que actúa a una velocidad acorde con la de la serpiente.

Por otro lado, hacemos que la velocidad de la serpiente incremente según el nivel en el que se encuentra y no con la puntuación, a mayor nivel mayor velocidad. No obstante, para conservar la jugabilidad, limitamos esta velocidad a un máximo, a partir del cual se puede seguir subiendo niveles pero no influyen en la velocidad.

1.3. Pantallas

Disponemos de 3 pantallas principales. Un menú con opción de jugar y salir: si se pulsa salir se acaba el juego y si se pulsa jugar aparecen tres caras que simbolizan las dificultades fácil, normal y difícil. La siguiente pantalla es el juego en sí: una vez seleccionada la dificultad, la pantalla de juego aparece. Cuando se acaba la partida, tanto si se sale pulsando "Esc" o al perder la última vida, la consola espera a que el jugador introduzca su nombre, el cual se guarda y se muestra la puntuación junto a este.

2.Modelo: Game Objects & Scenes

2.1 Game Objects y escenas

→ Diagrama de clases

→ Explicación breve de las estructuras de datos usadas y el porqué

Utilizamos arrays para guardar el ranking y también para las posiciones de la serpiente.

2.2 Diseño del fichero XML de configuración del juego

→ Estructura.

El fichero XML está estructurado en 3 nodos(easy, medium y hard), dentro están guardados en diferentes nodos: el número de casillas que tendrá el juego, el tiempo que tienes para comer el número de frutas con tal de subir de nivel, el número de frutas que tienes que comer para subir de nivel y el multiplicador de puntos por dificultad.

2.3 Uso y/o modificación de la clase SceneManager

No existe dicha classe.

3. Controlador: Game

3.1 Game. Atributos y métodos

El game loop cuenta con 3 funciones: la draw que controla la impresión en la ventana, logic que controla el avance del juego y input que controla la entrada de datos por parte del jugador.

3.2 Eventos

Si el jugador pulsa una flecha de dirección durante el juego, la serpiente siempre que pueda cambiará su dirección. Además están los eventos del menú para jugar y aumentar la dificultad, y el ranking para entrar tu nombre y que imprima el ranking.

3.3 Diagrama de clases

Nuestro juego no dispone de clases, cuenta con funciones que hacen que se pueda jugar.

3.4 Uso y/o modificación de la clase InputManager

No existe una clase InputManager en sí pero usamos una función Input, esta controla la entrada de datos mientras el juego se está ejecutando.

4. Vista: Renderer

→ Uso y/o modificación de la clase Renderer

En nuestro proyecto hemos puesto la función draw que gestiona las imágenes que se verán por pantalla.

5. Diagrama de clases de todo el proyecto

No hemos usado clases, hemos creado funciones que controlan el correcto desarrollo del juego durante su funcionamiento.

6. Deployment

La posibilidad de jugar fuera de Visual Studio 2015 no ha sido contemplada hasta realizar este report. En caso de poder se necesitaría principalmente el ejecutable y los datos en la carpeta (audio, imágenes y scripts).

7. Estimación de tiempo

Guillem: 39 horas

- Draw
- Gestión de imágenes
- Gestión de ficheros
- Ranking

Arnau: 49 horas

- Menus
- Draw
- Inputs
- Lógica del juego
- Intento de realizar el proyecto con clases 2 veces, usando ficheros .h para declarar los atributos y métodos de cada clase y .cpp para implementarlos

8. Conclusiones

→ Lecciones aprendidas

Es necesario tener un nivel de conocimiento sobre C++ y SDL más allá de nuestro alcance de programación para llevar a cabo debidamente este proyecto.

Es muy desfavorable no tener la ocasión de realizar pequeñas piezas o pequeños trabajos que nos ayuden a construir el puzle final como este proyecto.

→ Satisfacción

El nivel de satisfacción de esta práctica es realmente bajo dada la elevada dificultad que ha presentado y los pocos recursos y ayudas que nos ha ofrecido el temario dado en clase. No obstante, el resultado de nuestro esfuerzo nos satisface en su medida.

→ Qué fue lo más difícil de entender y/o de implementar

Lo que más esfuerzo requirió para que funcionara fue la parte del renderizado o función Draw e implementar el sistema de clases que finalmente no pudimos realizar.

→ Fortalezas/debilidades de su software Referencias (Bibliografía, páginas web, etc.)

El juego cuenta con música.

El juego es jugable y no tiene errores.

El ranking no está implementado al 100%.

No cuenta con classes.

<http://stackoverflow.com/>












<http://www.cplusplus.com/>










<http://en.cppreference.com/w>

9. Anexo

REQUISITOS ESPECÍFICOS DEL JUEGO: SNAKE

Marcar el fondo de la casilla de cada requisito del proyecto en verde o en rojo según si ha sido completado o no respectivamente para la entrega final. **Incluir esta tabla en el anexo del *report* del proyecto.**

	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 continua.
	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.