Daniel Bruna
David Recuenco

Número de prácticas 29
Grupo B

Juego **2**
1.1 Selección de juego. 2
1.2 Adaptaciones, cambios o mejoras con respecto al juego original. 2
1.3 Pantallas 2

Modelo: Game Objects & Scenes **2**
2.1 Game Objects & Scenes. 2
2.2 Diseño del fichero XML de configuración del juego. 3
2.3 Uso y/o modificación de la clase SceneManager. 3

Controlador: Game **4**
3.1 Game. Atributos y métodos. 4
3.2 Eventos. 4
3.3 Diagrama de clases. 4
3.4 Uso y/o modificación de la clase InputManager 4

Vista: Renderer **5**
Uso y/o modificación de la clase Renderer. 5

Diagrama de clases de todo el proyecto **5**

Deployment **5**

Estimación de tiempo **5**

Conclusiones **6**

1. Juego

1.1 Selección de juego.

Para la selección del juego teníamos dos opciones. La primera el Frogger y la segunda el Snake. Al final nos decidimos por el Snake porque para nuestro proyecto necesitamos hacer un boss basado en el Snake. Por lo tanto nos era mucho más útil hacer esta práctica con el juego del Snake para poder reutilizar el comportamiento de esta snake para nuestro proyecto.

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

Hemos copiado el comportamiento en el funcionamiento de la serpiente con respecto al juego original. A excepción de cuando come una fruta que la serpiente se hace más grande y sigue avanzando sin esperar como en el juego original. Aparte de esto hemos copiado el resto de cosas para que se parezcan al juego original.

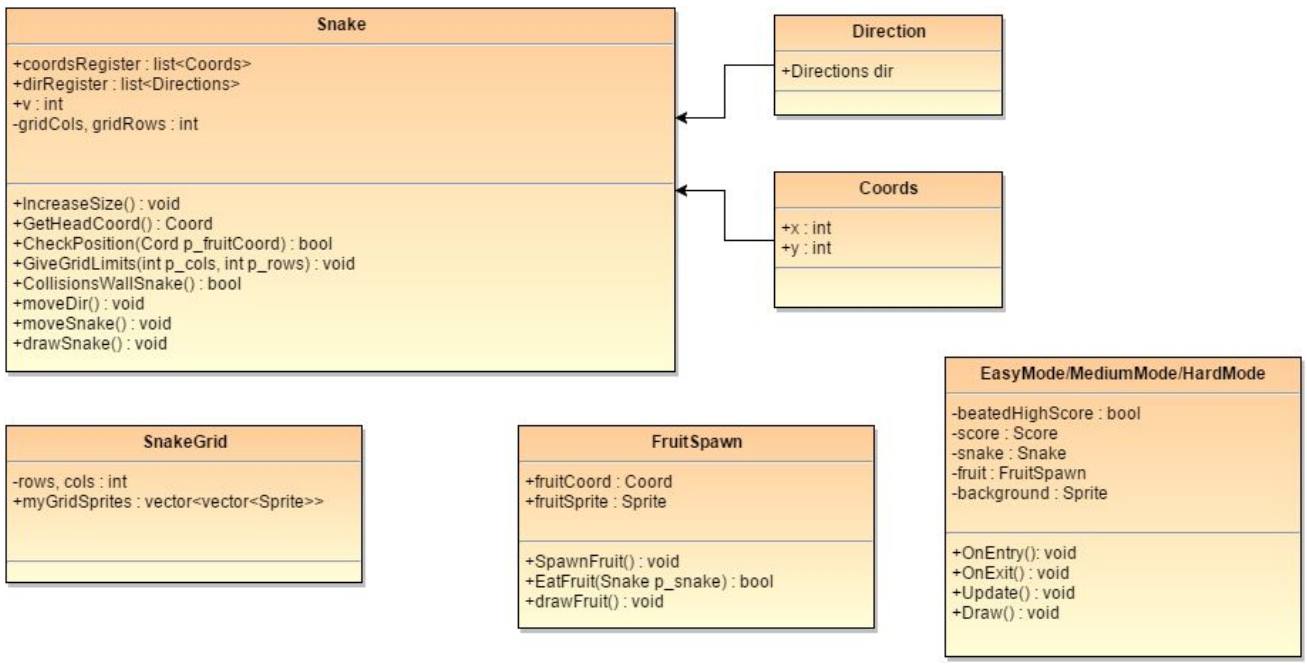
1.3 Pantallas

Una vez se ejecuta el programa, el jugador se encontrará en el Main Menu, donde el cual el jugador puede elegir entre cuales modos jugará: las pantallas de juego.
Las 3 pantallas del juego se dividen en el modo Easy, Medium y Hard.
Cada pantalla en su parte superior tiene un HUD el cual muestra el modo en el que el jugador se encuentra, juntamente con su puntuación, vidas y su highscore. Debajo de este HUD está el área por donde la snake se moverá, junto a los límites del área.

2. Modelo: Game Objects & Scenes

2.1 Game Objects & Scenes.

Diagrama de clases:



Las estructuras de datos:
Para la snake he usado dos lista, una para las coordenadas y otra para las direcciones. La lista de direcciones hace un efecto cascada para pasar las

direcciones desde la cabeza hasta la cola de una en una. De esta forma simulamos el movimiento de una serpiente. La lista de coordenadas se actualiza para cada parte en función de su dirección. Originalmente empezamos usando solo una de coordenadas y luego le sumamos una de direcciones. Pusimos dos listas para poder acceder de forma independiente a una y a otra. Además de que es más sencillo usar dos listas que una lista de pair. Aparte que el código ya estaba hecho para el uso de una lista y así no teníamos que cambiar una gran parte de este.

Para la grid del mapa hemos usado un vector de vector de sprites. Ya que lo que queríamos era una matriz de sprites, que corresponda con el tamaño de la grid directamente accesible con las coordenadas, nos decidimos por esta estructura de datos porque era lo que mejor iría. El acceso por coordenadas se hace directamente con esos valores sin necesidad de iterar por toda la estructura. Esta estructura se llena de sprites para cada hueco partiendo de la base que los mapas tienen una dimensión COLxROW. Si más tarde se quiere incluir patrones de obstáculos se puede hacer también. Solo haría falta cambiar los sprites dentro de esta estructura y añadir una nueva lista con las coordenadas de los obstáculos para poder compararlas y actualizar las colisiones para incluir estos nuevos obstáculos.

La fruta tiene una simple coordenada, y un sprite que no cambia, que va cambiando cada vez que es comida por la snake. Estas coordenadas una vez cambiadas se verifica que no haya spawneado dentro de un muro o dentro de la snake. SI es así se vuelven a cambiar hasta que encuentre un lugar vacío.

Para las escenas hemos utilizado algunas cosas que ya había hecho Jordi. El OnEntry, Update y Draw por ejemplo. Le hemos sumado nuestros objetos y sus funciones para ejecutar el juego. Hemos creado funciones dentro de los objetos para inicializar valores en función del modo/nivel. Cada escena tiene sus propios parámetros que se establecen en la función OnEntry.
Para la ejecución del juego usamos la función Update para actualizar tanto la posición como la dirección de la Snake, además verificamos las colisiones para sumar puntuación en el caso de la fruta y restar vidas en el caso de las paredes y la Snake sobre sí misma.

2.2 Diseño del fichero XML de configuración del juego.

Para cada nivel lo único que cambia son los multiplicadores para las frutas, la velocidad de la snake, las dimensiones de la grid, etc. Hemos diseñado el archivo XML para que contenga un multiplicador en cada subcategoría para poder saber a cuál corresponde cada uno. Cada nivel es un nodo con sus respectivos sub nodos con estos multiplicadores.

2.3 Uso y/o modificación de la clase SceneManager.

El SceneManager no ha sido modificado pero ha sido utilizado para cambiar entre el menú principal y las 3 diferentes escenas de juego, Easy/Medium/Hard.

3. Controlador: Game

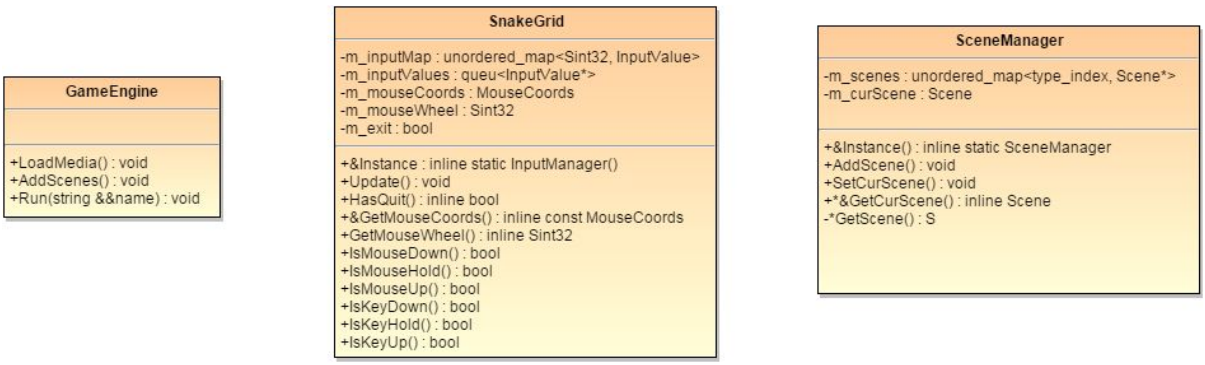
3.1 Game. Atributos y métodos.

El SceneManager no ha sido modificado, pero su propia instancia (SM) nos ha servido para resetear una escena en caso de que el jugador haya perdido la partida, en caso de avanzar de nivel una vez la snake se ha comido el número de frutas preestablecido y para seleccionar el modo de juego (escena) a iniciar en el Menú Principal del juego.
El InputManager no ha sido modificado ya que solamente lo hemos utilizado para detectar el momento en el que el usuario clickaba con el ratón. Desde un principio tuvimos la intención de modificarlo con el fin de detectar también en qué dirección quería ir el jugador, pero como que para la detección de teclas el código era bastante distinto al del InputManager preferimos hacer estas detecciones en otra clase (Snake.hh).
El GameEngine no ha sido modificado en sí, sino que ha sido completado con las imágenes, fuentes (una en este caso) y escenas en las funciones LoadMedia y AddScenes.

3.2 Eventos.

El juego detecta los clicks del ratón los cuales sólo se utilizan en el menú principal para elegir qué nivel jugar.
El juego también detecta la dirección en la que el jugador quiere que la snake vaya, estas direcciones se pueden introducir mediante las flechas del teclado o mediante las teclas WASD.
La snake detecta las colisiones con el muro, consigo misma y con la fruta.

3.3 Diagrama de clases.



3.4 Uso y/o modificación de la clase InputManager

La clase de InputManager no ha sido modificada ya que el código que detecta cuales teclas están siendo pulsadas se encuentra dentro del Snake.hh, que serán las encargadas de dar dirección a la snake. A parte de utilizar las flechas de dirección del teclado también hemos querido añadir las teclas WASD como método alternativo a las flechas, ya que en nuestro caso, nos parece más cómodo dirigir la snake con estas teclas. Aunque el InputManager no haya sido modificado, lo utilizamos igual, aprovechando las funciones que éste tiene para detectar los clicks del ratón (manera

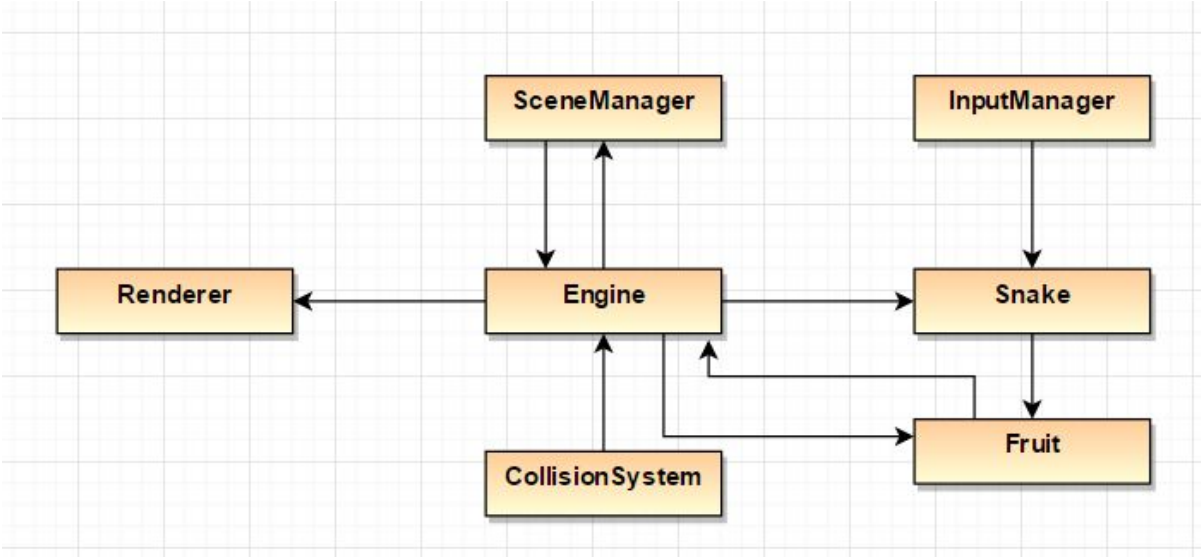
la cual el jugador selecciona el modo de juego en el Menú Principal) y para los inputs para mover la snake.

4. Vista: Renderer

Uso y/o modificación de la clase Renderer.

Tanto la clase Sprite, como la clase Transform y la clase Render han sido modificadas con el fin de cambiar el punto de anclaje de cada imágen. Para esto hemos utilizado la función SDL_Point, la cual fija el punto de la imágen en cuestión en las coordenadas deseadas. Es por esto que hemos modificado estas clases, añadiendo una segunda versión del Transform (llamada Transform2) la cual el punto de anclaje de la imágen se encuentra en el extremo izquierdo de ésta. Para utilizar este nuevo transform no nos bastó con añadirlo, sinó que también tuvimos que añadir su correspondiente Sprite (ya que el sprite se basa en un transform, el cual se incluye en el Sprite.hh). Además de esto, tuvimos que hacer un Push específico para el nuevo Sprite, aplicado a éste (esto en el Renderer.hh).

5. Diagrama de clases de todo el proyecto



6. Deployment

(Especificar qué se necesita tener instalado en el PC para jugar fuera de Visual Studio 2015)

7. Estimación de tiempo

Este trabajo lo hemos dividido en dos partes, una para cada miembro. Por un lado el comportamiento de los objetos y sus colisiones. Por otro lado el entendimiento de cómo funciona el código de Jordi como el renderer, el engine y el time management. Daniel se ha encargado de la parte de los objetos y sus colisiones. Además de diseñar el archivo XML. David se ha encargado de la parte del entendimiento del código además y de implementarlo en nuestro juego con todos los objetos. Además de diseñar las clases de los niveles y la fruta.

De la parte de trabajo más larga ha sido el pensar todas las clases y sus relaciones con detenimiento. Sobretodo la clase de la snake ya que su movimiento muy particular era complicado de diseñar. Al principio empezamos usando el método cascada que se nos ocurrió con las coordenadas pero luego nos dimos cuenta de que no funcionaba bien porque no tenía en cuenta los sprites y sus tamaños.
Después se nos ocurrió usar una cuadrícula, y finalmente juntamos todas las ideas usando dos listas con las coordenadas y las direcciones además de usar una cuadrícula del tamaño de los sprites. En cuestión de tiempo fácilmente hemos pasado unos cuantos días para diseñar todos los objetos.
Nos habrá llevado más o menos el mismo tiempo para comprender el funcionamiento de la base del engine, el renderer, etc.

8. Conclusiones

Lecciones:
Para este trabajo he tenido que usar las estructuras de datos lo que me ha permitido aprender más en cómo usarlas. Además de obtener más facilidad al decidirme por unas u otras. Por otro lado también he aprendido algunas cosas de herencias a pesar de no haberlas usado al final.
Otra parte del trabajo que nos hubiese gustado haber aprendido más es sobre la librería de SDL. No la hemos usado mucho en clase y no hemos aprendido mucho sobre. Creemos que hubiese sido mucho mejor, y más motivante, haber hecho el trabajo desde zero con más temario enseñado en clase. Además hubiese sido mejor haber hecho más clases sobre archivos. Algo que a nuestro parecer es muy importante en estos trabajos.
Sin embargo me ha parecido acertado habernos hecho investigar para hacer el trabajo, pero a nuestro parecer ha sido demasiado sobre lo que investigar.
A pesar de todo es algo que vamos a investigar por nuestra parte a partir de ahora, especialmente C++ y la manipulación de archivos.

Satisfacción:
Daniel: Personalmente me encanta programar cosas muy variadas por el reto que supone sacar el comportamiento de los objetos como el movimiento de la snake. Ha sido muy satisfactorio encontrar una forma de hacerla. Me ha parecido también muy útil saber varias estructuras de datos ya que me da más posibilidades para resolver estos puzzles.
David: Al igual que Daniel, me encanta programar, pero si algo me ha gustado mucho y me ha llenado al acabar el proyecto es el haberlo hecho en SDL, ya que a diferencia de Unity (lo que más he programado y suelo programar) SDL es como un Unity "invisible", lo cual también me ha ayudado a comprender cómo es el funcionamiento de Unity por dentro.

Dificultades:
Daniel: Lo más complicado de entender e implementar ha sido el movimiento de la snake. Ha sido para lo que he usado más tiempo en esta práctica. Sobretodo teniendo en cuenta que no podíamos compilar el trabajo, por lo que no podías testear los programas. Eso ha sido un reto al diseñarlo todo, pensando en cada detalle. Una vez

hemos podido testear todo ha ido más deprisa, y a nuestra sorpresa no habían muchos errores en nuestros códigos.
Aparte también está el uso de un archivo XML, el cual nos ha dado muchos problemas y no nos ha dado tiempo a arreglarlos a tiempo. Por lo cual hemos tenido que hacer que el juego funcione independientemente del archivo XML. Aunque el archivo está diseñado y creado.
David: Uno de los problemas que más nos ha dificultado el hacer este proyecto fue que al intentar ejecutar el proyecto, Visual Studio daba un error el cual no dejaba que el programa se ejecutara, por lo que no podíamos testear lo que íbamos haciendo para el proyecto. Finalmente pudimos resolver el problema, pero nos comió mucho tiempo ya que lo solventamos muy tarde y fuimos bastante justos para resolver bugs y testear que todo el juego funcionase correctamente.

Fortalezas y debilidades del software:
Este software tiene una buena relación entre objetos además de un buen diseño de estos. Por otro lado la debilidad del software yo creo que es, sobretodo durante su desarrollo, el uso de un código hecho por otra persona. Un código complicado y avanzado como es el caso de el código que hizo Jordi es difícil de basar un trabajo en el. Esto ha complicado mucho el desarrollo del trabajo y su fortaleza en cuanto a ejecución.

Referencias

Para este trabajo hemos usado cplusplus y stackoverflow para resolver algunas pequeñas dudas sobre las estructuras de datos. Pero por lo general lo enseñado en clase ha sido suficiente para que podamos hacer el diseño de los objetos por nuestra cuenta sin ningún problema.