Integrantes: David Pérez, David López
Núm. de grupo: 3
Grupo de clase: A

Tabla de contenido

Requisitos básicos	
La serpiente se mueve mediante teclado correctamente.	Sí
El jugador tiene "k" vidas y al perderlas termina la partida.	Sí
Aparece 1 alimento aleatorio en escena cada vez que la serpiente se come uno.	Sí
La serpiente incrementa su tamaño al comer alimentos y se forma una cadena contínua.	Sí
La serpiente colisiona con los límites de la pantalla y se pierde 1 vida.	Sí
La serpiente colisiona con su propio cuerpo y se pierde 1 vida.	Sí
Existe una barra de tiempo que disminuye progresivamente y al llegar a 0 el jugador pierde 1 vida.	No
Se puede pasar de nivel al consumir "x" alimentos.	Sí
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.	Sí
La puntuación se suma correctamente según la fórmula “q*100” (siendo “q” el número del alimento en el nivel).	Sí
Aumenta la velocidad de la serpiente según la puntuación.	No
La matriz de casillas varía según la dificultad.	Sí
El tiempo inicial de juego varía según la dificultad.	No
La velocidad inicial de la serpiente varía según la dificultad.	Sí
El número inicial de alimentos varía según la dificultad.	Sí
El número incremental de alimentos varía según la dificultad.	Sí
Al perder una vida, se reinicia el nivel con el mismo tamaño que tenía la serpiente al pasar de nivel.	No
Requisitos adicionales	
Cada nivel consta de obstáculos diferentes distribuidos por el mundo de juego.	No
La serpiente puede colisionar con los obstáculos y se pierde 1 vida.	Sí
Cada patrón de obstáculos está creado previamente en código o se carga desde archivo de texto plano.	Sí

1.Juego

1.1.Selección del juego

Escogimos el juego de snake debido a que a ambos integrantes del grupo nos gusta mucho el juego y es el juego(de los 3 a escoger) que mas hemos jugado.

1.2.Adaptaciones, cambios o mejora respecto al juego original

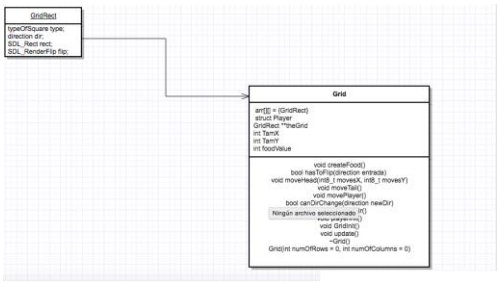
Ninguna

1.3.Pantallas

En el juego hay nueve pantallas principales que son las de los diferentes niveles, esta cambian según en el modo de dificultad en el que te encuentres, haciendo que varíe el tamaño dependiendo de el modo de juego.

2.Modelo, Game Objects & Scenes

2.1.GameObjects y escenas



A pesar que en nuestro código no hay apenas GameObjects como tal, en este diagrama la clases se ve representado la clase Grid la que esta formada por diferentes GridRect (structs), que casi se podrían catalogar como GameObject

Las estructuras de datos usadas han sido sobretodo structs, unordered maps, lists, queues, estructuras propias de SDL y enum class.
La mas utilizada de estas estructuras han sido los structs, ya que nos servian tanto para hacer las “celdas” de la grid, como de keys para los unordered maps que guardaban los datos de renderizado.

2.2.Diseño del fichero XML de configuración del juego

El fichero XML contiene las estadísticas basé del videojuego que luego como seria el número de columnas predeterminado o el número de alimentos predeterminado, y luego estos se manipularan en código para adaptarlos a cada nivel en el SceneManager.

2.3. Uso y/o modificación de la clase SceneManager

3.Controlador: Game

3.1.Game. Atributos y métodos

Los atributos y métodos usados para controlar el juego han sido en gran parte inicializados en el InputManager, que luego se han usado en el Grid i el GameEngine para determinar el movimiento del player.

Los atributos usados han sido inpValMap, inpVal y outGame, los cuales se han usado para guardar los estados de las teclas para después pasarse al jugador en el Grid i GameEngine, en el caso de los dos primeros atributos y en el caso del segundo esté se usa para determinar si la partida sigue en marcha o no.

Por lo que se refiere a los métodos usados para el control del juego estos aparecen en la clase Grid y la Input Manager, en el caso de InputManager el método utilizado es el Updating que se encarga de guardar un estado dependiendo de la tecla que se haya apretado para que luego se pase a un unordered map, en el caso del Grid este utiliza un método que servirá para la dirección del player.

3.2.Eventos

Los eventos que hemos utilizados han sido implementados en su mayoría en el InputManager, estos eventos son SDL_QUIT, SDL_KEYDOWN, SDL_KEYUP, SDLK_UP, SDLK_DOWN, SDLK_LEFT y SDLK_RIGHT, todos estos eventos en su mayoría se han usado para identificar eventos de teclado a excepción de SDL_QUIT que se ha usado para determinar si la ventana del juego esta en funcionamiento o no.

3.3.Diagrama de clases

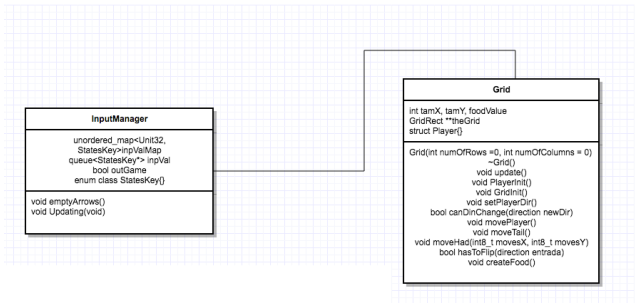


Diagrama de clases que indica la interrelación entre InputManager y Grid.

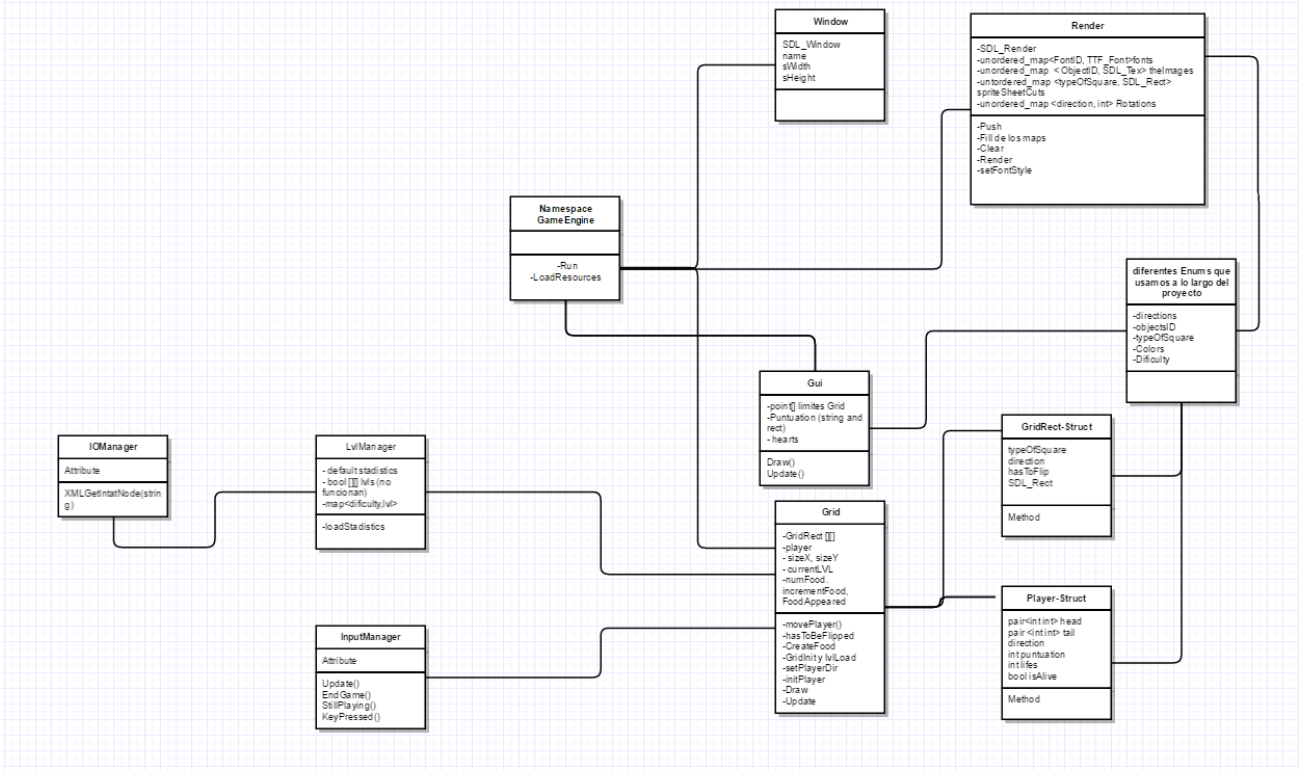
3.4.Uso y/o modificación de la clase InputManager

La clase InputManager se encarga de detectar los cambios de estado que se producen en el juego, en este se controlara cuando se salga del juego (apretando la x de la derecha de la ventana), o de que si las teclas están presionadas o no, esto lo hará ha trabes de un siwtch al que se le pasaran los eventos de SDL y este determinará el estado de las teclas.

4.Vista: Renderer

La clase Renderer se encarga de cargar los sprites i las texturas del juego, también guarda las rotaciones de los sprites. En las modificaciones de esta clase cabe destacar la creación de mapas para guardar los colores, imágenes, donde se debían cortar los spritesheets y les rotaciones; también el uso de push que sirve para coger un objeto de la clase GridRect que se encargará de traer la información del tipo de cuadrado, la rotación, etc.

5.Diagrama de clases de todo el proyecto



6.Deployment

Windows.

7.Estimación de tiempo

Investigación 30-40 horas e implementación ~25 horas(10-15 sin los debugs).

8.Conclusiones

Como usar la librería SDL y sus funciones y características y como usar XML, también a sido útil para repasar estructuras de datos como la queue, los maps o la list.

Nos hubiera gustado poder dar un mejor acabado al juego (audio, menús, etc), pero debido a la falta de conocimientos sobre la librería usada hemos tardado más de lo planeado.
También nos gustaría comentar un aspecto que nos ha parecido curioso. Durante las primeras semanas, como era la primera vez que tratábamos con C++ a este nivel de complejidad y, además, también era la primera vez que usábamos la librería SDL, nos sentiamos bastante “perdidos”, y nos centramos mucho en seguir el código de ejemplo del EntiCrush. Sin embargo, este código era muy complejo en ciertos aspectos y provocó que nuestra sensación de no saber qué hacer aumentase aún más.
Por esta misma razón, al volver de vacaciones de navidad decidimos abandonar ese código y empezar “de cero” a hacer el nuestro. Tras tomar esta decisión fue cuando el proyecto empezó a tomar forma de verdad y pudimos empezar a avanzar

Posiblemente, lo más difícil de implementar ha sido el polimorfismo, hasta el punto que nos ha impedido acabar los menús y el cambio de escenas.
Aparte de esto, también sufrimos un error de compilación que nos costó un día entero de resolver (ya que el error saltaba en un header interno de std).

Fortalezas: si hay algo en lo que nuestro proyecto pueda destacar, creemos que es el movimiento del player, ya que hemos ideado un sistema que nos permite alargarlo “infinitamente” simplemente con guardar 2 coordenadas.

Referencias

http://www.cplusplus.com/reference/

https://www.libsdl.org/

http://stackoverflow.com/