Práctica 3: Arrows 3.0

Curso 2019-2020. Tecnología de la Programación de Videojuegos. UCM

Fecha de entrega: 17 de diciembre de 2019

El objetivo fundamental de esta práctica es introducir una arquitectura escalable para el manejo de los estados de un juego. Para ello, partiremos del Arrows 2.0 de la práctica anterior y desarrollaremos la siguiente extensión: Al arrancar el programa aparecerá el *menú principal*, el cual permite al menos: (1) empezar una partida nueva, (2) cargar una partida guardada a partir del código numérico usado al guardarla (como en la práctica 2), (3) salir del programa. Por otro lado, mientras se está jugando, si se pulsa la tecla *esc*, el juego se detiene y se visualiza el *menú pausa*, un menú que permite al menos: reanudar la partida, guardar la partida (para lo cual se solicitará el código numérico), y volver al menú principal (acabando la partida en curso). Finalmente, cuando se acaba la partida deberá visualizarse el *menú fin*, una pantalla en la que además de informar al usuario si ha ganado o perdido, aparecerá un menú con opciones para volver al menú principal y para salir de la aplicación. En todos los menús, se usará el ratón para seleccionar la opción elegida.

Detalles de implementación

El juego usará una máquina de estados para manejar las transiciones entre estados del juego. Implementa por tanto la clase `GameStateMachine`, que incluye como atributo una pila de estados (tipo `stack<GameState*>`), y los métodos `currentState`, `pushState`, `changeState` y `popState`. Los métodos `update` y `render` de la clase `Game` delegarán respectivamente en los métodos `update` y `render` del estado actual (el obtenido llamando a `currentState` que se encuentra en la cima de la pila). Debes implementar al menos las siguientes clases para manejar estados del juego:

GameState: Es la clase raíz de la jerarquía de estados del juego y tiene al menos tres atributos: el escenario del estado del juego (`list<GameObject*>`), los manejadores de eventos (`list<EventHandler*>`) y el puntero al juego. Implementa además los métodos `update`, `render` y `handleEvent`.

PlayState: Implementa el estado del juego del Arrows propiamente dicho. Incluye por tanto gran parte de los atributos y funcionalidad que antes teníamos en la clase `Game`. La antigua lista de objetos del juego ahora sería el escenario (`list<GameObject*>`) heredado de `GameState`. El cambio de nivel lo debe desencadenar el método `update`.

MainMenuState, PauseState y EndState: Implementan respectivamente los estados del juego correspondientes a los menús *principal*, *pausa* y *fin*. El escenario de los menús estará compuesto por objetos de tipo `MenuButton`, que se manejan como objetos del juego, es decir, almacenan su textura, saben dibujarse, actualizarse (si es necesario) y reaccionan a eventos. Implementa por tanto la clase `MenuButton` como subclase de `SDLGameObject` (ver abajo) con atributos para su(s) textura(s), y para la función a ejecutar en caso de ser pulsado (de tipo `CallBackOnClick*`), que se invocará desde el método `handleEvent`. El tipo de la función será: `using CallBackOnClick = void(Game* game)`. Se darán más detalles en clase (ver también el capítulo 5 del libro de SDL).

SDLGameObject: Debes definir una nueva clase abstracta en la jerarquía de objetos del juego entre `GameObject` y `ArrowsGameObject`, que contendrá los atributos para la posición, altura y anchura, textura y puntero al estado dueño del objeto, e implementará las funcionalidades que correspondan. De esta clase también heredará la nueva clase `MenuButton`.

Observa que ahora la clase `Game` quedaría solo con los siguientes atributos básicos: los punteros a `SDL_Window` y `SDL_Renderer`, el booleano de final de la aplicación, el array de texturas y la máquina de estados. De hecho, esta clase podría pasar a llamarse `SDLApplication` pues ya no tiene nada referente al juego propiamente dicho.

Pautas generales obligatorias

A continuación se indican algunas pautas generales que vuestro código debe seguir:

- Se debe hacer un buen uso de la herencia y del polimorfismo de manera que el código sea claro, se eviten las repeticiones de código, distinciones de casos innecesarias, y no se abuse de castings ni de consultas de tipos en ejecución (mediante `typeid`).
- Asegúrate de que el programa no deje basura. Para que Visual te informe debes escribir al principio de la función `main` esta instrucción

  ```
  _CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
  ```

  Para obtener información más detallada y legible debes incluir en todos los módulos el fichero `checkML.h` (disponible en la plantilla en el proyecto `HolaSDL`).
- Todos los atributos deben ser privados excepto quizás algunas constantes del juego en caso de que se definan como atributos estáticos.
- Define las constantes que sean necesarias. En general, no deben aparecer literales que pudiesen corresponder con configuraciones del programa en el código.
- No debe haber métodos que superen las 30-40 líneas de código.
- Escribe comentarios en el código, al menos uno por cada método que explique de forma clara qué hace el método. Sé cuidadoso también con los nombres que eliges para variables, parámetros, atributos y métodos. Es importante que denoten realmente lo que son o hacen.

Funcionalidades opcionales (2 puntos adicionales máximo)

- Implementa el soporte para que la carga de texturas se realice a partir de la información contenida en un fichero (en lugar de a partir del array de atributos de texturas). Utiliza además una estructura de tipo tabla (`map` o `unordered_map`) para almacenarlas y obtenerlas a partir de un identificador unívoco (tipo `string`) para cada una (que estaría también en el fichero).
- Implementa el soporte para que se registren de manera ordenada las puntuaciones (o tiempos) de las partidas acabadas junto con sus fechas y nicks asociados. Debes utilizar para ello una tabla (tipo `map`) indexada por puntuación y fecha.
- Utiliza el paquete `TTF` de SDL para manejar los distintos textos (contador, botones, etc.).

Entrega

En la tarea del campus virtual *Entrega de la práctica 3* y dentro de la fecha límite (17 de diciembre), cada uno de los miembros del grupo, debe subir un fichero comprimido (.zip) que contenga la carpeta de la solución y el proyecto limpio de archivos temporales (asegúrate de borrar la carpeta oculta .vs y ejecuta en Visual Studio la opción "limpiar solución" antes de generar el .zip). La carpeta debe incluir un archivo `info.txt` con los nombres de los componentes del grupo y unas líneas explicando las funcionalidades opcionales incluidas y/o las cosas que no estén funcionando correctamente. Ese mismo texto debes subirlo también en el cuadro de texto (sección "texto en línea") asociado a la entrega.

Además, para que la práctica se considere entregada, deberá pasarse una *entrevista* en la que el profesor comprobará, con los dos autores de la práctica, su funcionamiento en ejecución, y si es correcto realizará preguntas (posiblemente individuales) sobre la implementación. Las entrevistas se realizarán durante las sesiones de laboratorio de los días 17 y 18 de diciembre, o si fuese necesario en otro horario acordado con el profesor.