Práctica 3: Pacman 3.0

Curso 2017-2018. Tecnología de la Programación de Videojuegos. UCM

Fecha de entrega: 17 de enero de 2018

Los objetivos fundamentales de esta práctica son introducir el mecanismo de excepciones de la programación orientada a objetos para el manejo de errores, e introducir una arquitectura escalable para el manejo de los estados de un juego. Para ello, partiremos del Pacman 2.0 de la práctica anterior y desarrollaremos una serie de extensiones que se explican a continuación.

1. **Estados del juego:** 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.

2. **Manejo de errores:** El programa debe manejar posibles errores de ejecución como errores de carga de SDL o errores de formato o no existencia de ficheros. En caso de ser posible, el programa debe tratar de recuperarse del error. Por ejemplo, si se produce algún error al cargar una partida guardada de un fichero, se deberá informar del error, pero el juego deberá volver al menú principal.

Detalles de implementación

Jerarquía de excepciones

Debes implementar al menos las siguientes clases para manejar excepciones:

PacManError: Hereda de *logic_error* y sirve como superclase de todas las demás excepciones que definiremos. Debe proporcionar por tanto la funcionalidad común necesaria. Reutiliza el constructor y método `what` de `logic_error` para el almacenamiento y uso del mensaje de la excepción.

SDLError: Hereda de `PacManError` y se utiliza para todos los errores relacionados con la inicialización y uso de SDL. Utiliza las funciones `SDL_GetError`, `IMG_GetError` y `TTF_GetError` para obtener un mensaje detallado sobre el error de SDL que se almacenará en la excepción.

FileNotFoundError: Hereda de `PacManError` y se utiliza para todos los errores provocados al no encontrarse un fichero que el programa trata de abrir. El mensaje del error debe incluir el nombre del fichero en cuestión.

FileFormatError: Hereda de `PacManError` y se utiliza para los errores provocados en la lectura de los ficheros de nivel y de partida guardada por formato incorrecto. Debes detectar al menos 5 errores diferentes de formato. Algunos ejemplos: tamaño incorrecto del mapa, valor de celda incorrecto, tipo no válido de fantasma, etc.

Estados del juego y Menús

El juego usará una maquina 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 dos atributos: el escenario del estado del juego (`list<GameObject*>`) y el puntero al juego. Implementa además los métodos `update`, `render` y `handleEvent`.

PlayState: Implementa el estado del juego del Pacman propiamente dicho. Incluye por tanto gran parte de los atributos y funcionalidad que antes teníamos en la clase `Game`. La antigua lista de personajes del juego (`list<GameCharacter*>`) ahora sería el escenario (`list<GameObject*>`) heredado de `GameState`. Debe incluirse ahora también al objeto `GameMap`. Sugerencia: Pon el Pacman el primero, el mapa el segundo y los fantasmas detrás. Por otro lado, el cambio de nivel lo desencadena 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 y reaccionan a eventos. Implementa por tanto la clase `MenuButton` como subclase de `GameObject` con atributos para su(s) textura(s), y para la función a ejecutar en caso de ser pulsado (de tipo `CallBack*`), que se invocará desde el método `handleEvent`. El tipo de la función será: `typedef void CallBack(Game* game)`. Se darán más detalles en clase (ver también el capítulo 5 del libro de SDL).

Cambios en la jerarquía de clases

Realiza además los siguientes cambios en la jerarquía de clases para adecuarla a las nuevas funcionalidades:

- La clase `Game` quedaría 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.
- Añade una nueva clase abstracta de nombre `PacManObject` como superclase de `GameCharacter` y `GameMap`, y como subclase de `GameObject`. Debe declarar al menos un atributo para guardar un puntero al estado del juego (`PlayState*`) y los métodos `loadFromFile` y `saveToFile` (que ya no pertenecerían a `GameObject`).
- Añade a la clase `GameObject` el método `bool handleEvent(SDL_Event& e)` con cuerpo vacío. La idea es que el manejo de eventos se vaya propagando desde el `handleEvents` de la clase `Game` por los objetos del juego mediante llamadas al método `handleEvent`.

Pautas generales

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 y se eviten las repeticiones de código.
- Asegúrate de que el programa no deje basura (usa para ello el mecanismo de detección de basura explicado en el CV).
- Todos los atributos deben ser privados o protegidos excepto quizás algunas constantes del juego en caso de que se definan como atributos estáticos.
- Define las constantes que sean necesarias (no deberían aparecer literales numéricos arbitrarios 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. Preferiblemente usa nombres en inglés.

Entrega

En la tarea del campus virtual *Entrega de la práctica 3* y dentro de la fecha límite (17 de enero), cada uno de los miembros del grupo, debe subir un fichero comprimido (.zip) que contenga los archivos de código .h y .cpp del proyecto, las imágenes que se utilicen, y, 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. 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 sobre la implementación. Las entrevistas se realizarán en las dos sesiones de laboratorio siguientes a la fecha de entrega, o si fuese necesario en horario de tutorías.