TPE POO: RUSHHOUR

Programación Orientada a Objetos | Primer Cuatrimestre 2016 | ITBA

Integrantes: Juan Manuel Alonso, Eduardo Fittipaldi, Facundo Varela, Emiliano Vazquez

Sistema y clases

Utilizamos Model-View-Controller como patrón de diseño para tener una división comprensible del juego. El modelo está compuesto por las clases Board, Block y Player. Un bloque (Block) consiste de una orientación (horizontal o vertical), una longitud (no menor a 1) y una posición principal en el tablero. A partir de ésta última y de su longitud, bloque ocupará la cantidad de casilleros correspondientes. Por ejemplo, si la posición es 0 en *x* y 0 en *y* (esquina superior izquierda), su orientación horizontal y su longitud 2, el tablero sabrá que ocupa (0,0) y (1,0). El auto rojo (Player) es un bloque único en el tablero, de orientación siempre horizontal y en la misma fila que la salida. El tablero (Board), que es cuadrado, tiene un tamaño (size) determinado por la partida que se desee jugar, una instancia del auto rojo (Player), una salida (exit) por donde el auto rojo deberá salir para ganar, un Set de Blocks para almacenar todos sus bloques (autos) y un mapa de posición (Point) a bloque utilizado para la interpretación del mouse respecto del tablero.

La vista (view) como su nombre lo indica es la representación gráfica del modelo (BlockView, BoardView, PlayerView) y está compuesta por clases advocadas al UI (MenuButton, EditorInput, Color, ViewConstants), la clase ScreenManager, que se encarga de actualizar lo que el se observa por pantalla (menús, juego, etc.) y paneles (~Pane) que representan el estado actual del juego, ya sea un menú, un juego o el editor de nivel.

El controlador (controller) consta de la clase Loader que carga un tablero desde un archivo, clases de estado (~State) , clases de manipulación (~Handler) y un administrador de estado del juego (GameStateManager) que cuenta con una pila de estados. Por ejemplo: al iniciar, el GameStateManager agrega a la pila un MainMenuState, que, como todo State, tiene un GameStateManager (el que lo instanció), un manipulador (MainMenuHandler) y un panel (MainMenuPane). Luego el GameStateManager le indica al ScreenManager que debe poner el panel del estado que agregó (MainMenuPane). Al finalizar, comienza el AnimationTimer del manipulador, en este caso, MainMenuHandler, el cual cuenta con el método handler (como el resto de los Handlers) que se ejecuta a la velocidad del AnimationTimer. Según el accionar del usuario, MainMenuHandler le pedirá al GameStateManager que agregue o quite estados.

Decisiones importantes

En cuento a la complejidad del modelo, cabe destacar la simpleza del mismo. Decidimos mantener dicha simpleza para preservar la integridad del mismo. Agregar más clases solo perjudicaría el diseño del modelo.

Para compensar la falta de complejidad del modelo, decidimos extender el módulo controller, agregando funcionalidad al juego. El primer agregado es el modo de 2 jugadores, el cual permite a 2 jugadores competir entre si para ver quién es el que resuelve un juego en menos movimientos. El segundo agregado al proyecto es la presencia de un editor de niveles, el cual permite al usuario crear niveles propios que luego puede cargar usando el botón Load en el menú principal (aclaración: no se valida si los niveles creados por el usuario tienen solución).

Otra decisión para recalcar es que no se utiliza un único framework. Elegimos JavaFX como framework principal y la inclusión de la clase Point pero perteneciente al framework de Java AWT ya que ésta permite un uso más simple en lo que refiere al guardado de un tablero (es serializable) mientras que la clase Point de JavaFX no.