TurboHiker Design

# Casper De Smet s0192221

General Overview

My turboHiker game library is built up using a version of the model-view-controller design pattern. In the main program an implementation of the ViewFactory is constructed, which is passed on to the world. This is the main controller. Then the run function is called on the world which starts the game.

I needed to provide a separation between the game logic and the game presentation. This was of course not easy to implement in the MVC pattern. For this I decided to provide a series of abstract classes that represent the view. These abstract classes need to be implemented to make the game presentation run. By doing this, the game logic does not need to know implementation details of the view, it simply calls the functions defined by the abstract classes.

Besides that, all entities needed to be created by using the abstract factory design pattern.

This is done by the AbstractTurboHikerFactory, which is implemented by the ViewFactory and ModelFactory. The ViewFactory, in turn, is an abstract class that needs to be implemented when implementing the presentation. The ViewFactory handles all construction, but relies on the concrete implementation of its abstract functions to get objects of the implemented PlayerHikerView, RacingHikerView…

There are four types of entities: Lane, Hiker, Obstacle and WorldComponent. The WorldModel and WorldView are WorldComponents. These entities all have abstract base classes, so that the factories can create them.

This all allows the World to interact with the full model and full view. This interaction is implemented by using Events. There is a single Event class, with an enum for all the different Event types. An event has data members for all event types, but only the relevant members for an event type are defined. Accessing any other data member will likely lead to undefined behaviour. Most classes have a handle event function, in which the relevant events will be caught and handled. Events can be raised by all components and are passed to the world, which decides how the pass the events to the relevant component.

The World contains a game and a number of HikerControllers. The game contains a thread that manages the full lifetime of a game. The full lifetime consists of a countdown of three seconds, followed by a maximum play time of 60 seconds. During this play time, every millisecond, an event is raised of type Tick. This updates the full game state. After a tick, the model passes its state to the view and the view gets refreshed. Then the world checks for collisions between hikers and obstacles and sends appropriate events to the model.

The HikerControllers all contain their own thread, making it possible that each hiker can move at his own speed. Every HikerController manages its own HikerModel, whose state gets propagated to the HikerViews every game Tick, by the World.

A doxyGen documentation is available in the project.