Advanced Programming – Road Fighter

Design Report

Luuk van Sloun
University of Antwerp
2018-2019

Implemented Features

Composition Design Pattern

The Composition Design Pattern is implemented with the World at the top of the chain. The World contains the player car, the opponents and the other entities. These entities are influenced by the World, as the World calls the actions on them. Examples would be the moving of the player car, the collisions between entities and of course the addition of new entities.

Observer Design Pattern

The Observer Design Pattern is used to control the (high) score system. The subject that is being observed is the player itself. The player's score changes due to certain events, such as overtaking an opponent and shooting a passing car. With every increase or decrease the subject will notify the attached observers, who then all receive the update at once. The observers in this case are the game object and the separately created high score class (see Extensions).

Roadfighter Library

A self-contained shared library called "roadfighter" was created to separate the logic aspect of the game from the visual aspect. This library contains every section needed for the logic part, such as the Singleton Classes, the Entity Factory and every logic derivative of the Entity Class.

Abstract Factories

Abstract Factories were implemented with at the top level the Entity Factory, containing pure virtual functions for the creation of all the entities (player car, opponents, etc.). The SFML Factory was then derived from this to create the actual entities. The entities used are stored as Entity pointers, while being SFML Entity pointers, to make sure both logic and game presentation aspects are reachable by 1 single pointer while still being separated from each other.

Singletons

The Singleton implementations in this project are the Random Class and the Transformation Class. The Random Class works together with the random number generator and is used to determine the placement on the x-axis when a new passing car spawns. The random x-coordinate of course lies between the outer edges of the road, as cars aren't allowed to drive off-road.

The Transformation Class uses the [-4,4] x [-3,3] coordinate system to determine the on screen pixel coordinates. The desired coordinates are passed in as a pair of floats, as well as the screen size. The screen size that is entered is the View size. In that way the coordinates are calculated by the actual window size and not by the initial size it started with. The Transformation Class is mostly used by the SFML objects, as they need the exact pixel coordinates to position the sprite.

The implementation of the Singletons has been done with static references. This way of creating a Singleton requires minimal effort combined with a safe and solid solution. The use of smart pointers was considered at some point, but due to the lack of permission to access a private constructor when using smart pointers, the decision to use static references was easily made.

The following link helped inspire this implementation of a Singleton:

https://stackoverflow.com/questions/1008019/c-singleton-design-pattern

Frames Per Second

To secure the same experience on any computer, a functionality was made to make sure every machine would run the game at approximately 60 frames per second. This might vary a few frames, but no more than a few is guaranteed. As soon as the while loop, which checks if the window is still open and if the game is still running, starts from the top, the end time of that “tick” is calculated by taking the current time at that moment and adding 16 milliseconds, which results in about 60 frames per second. At the end of every tick the current time is checked and if the calculations were faster than the tick should take, it simply waits until the end time is reached. This provides a stable execution of the game on every system.

Extensions

High Score System

The high score system uses the observer system with the high score class containing the top 10 scores. At the start of the game, these high scores will be read from the designated high score file and stored in a vector. As an extra feature to the game there’s an implementation of a live high score view. Not only will your live score be shown on the screen throughout the entire game, but your score will be shown in the all time top 10 as soon as it surpasses a score in that same top 10. This feature has been added to create an extra form of tension while playing, as you get to see the amount of points still needed to make it to a certain spot in the top 10.

Splash screens

Along with the use of sprites and a scrolling background to provide a more visually appealing game, the addition of splash screens was used. Splash screens are single screens that act as simple menus or just as screens to show info. The first implementation is the start screen, on which the player has to press enter to start the game. The second implementation can be found at the end of the game, where two screens are provided: one for when the player crosses the finish line (which grants a fuel bonus) and another for when the player has taken too long to reach the finish line, which will result in a stand still due to running out of fuel.

Fuel

The addition of fuel comes from the original game. Movement requires fuel and fuel levels decrease, even when standing still. This forces the player to keep moving towards the finish line.

Shooting

To make sure simply spraying bullets isn't allowed, a limit of 30 bullets was added. This way the player has to keep in mind that, in case of a lot of cars close to each other further on in the game, bullets might be needed to pass without crashing.

Animation

A simple, but visually satisfying explosion animation was added to give the player the feeling that crashing or shooting a car results in something more than just a slowdown.