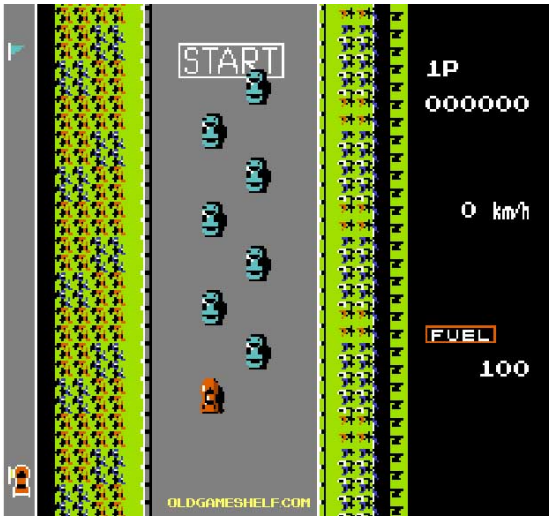
C++ Project 2018-2019: Road Fighter



Goal of this project:

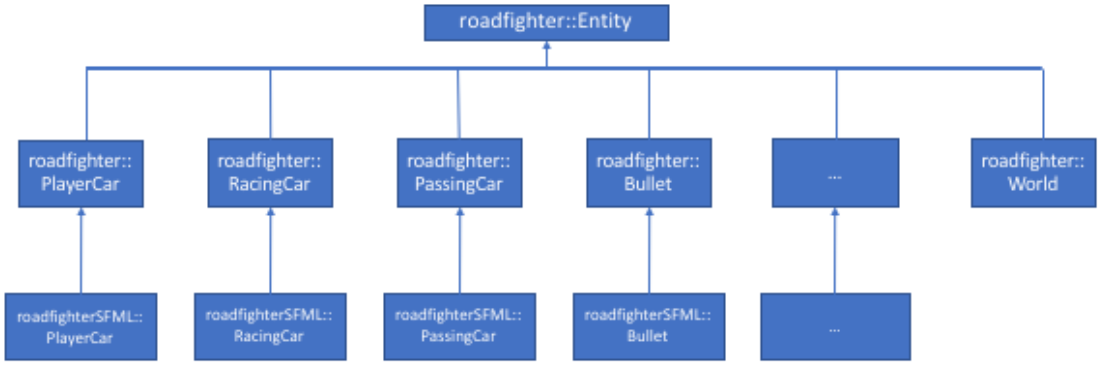
Write a Road Fighter - like game in C++ using SFML and (try to) follow the guidelines listed below.

Basic game feature requirements:

- The player can move the player car from left to right, up and down the road.
- At least one type of computer controlled racing cars that race with the player car to the finish line.
- At least two other types of vehicles that the player car can pass. Hitting different types of cars should have different effects.
- The player can shoot bullets to other cars to slow them down or destroy them.
- The live score of the player is determined by multiple events: order of arrival, crashes, destroyed cars, …
- Cars are only allowed to drive on the road.

Game logic library:

- Provide a clear **separation between game logic and game presentation**. Do this by encapsulating all game objects and logic, except their presentation and user-interaction (i.e. keyboard), in a **self-contained (static or shared) library**. (Look at C++ “prog2” code for examples of creating libraries. CMake can easily create static & shared libraries) The goal of this separation is to provide a very simple way of writing a completely new visual presentation based on the same game logic and structure. This way you might have two completely different looking games, based on the same game library. Use a separate namespace (f.i.: **roadfighter**) for this library.
- Design a hierarchy of game entities and their interactions (collision control, position control, ...). Be as creative & fancy as you like. A basic entity hierarchy would, for instance, be implemented as in this figure:



- Obligated logic featues:
    - Provide an entity that represents the game world (**roadfighter::World**). The world contains a list of child entities (cfr: **composition** design pattern) and delegates visualisation requests to its children. The world might also do other game related things such as collision control.
    - Use one class to represent **the game**: an object that interacts with world by calling the world's methods to enable the game logic.
    - Use the **abstract factory** pattern to create entities (such as the player, other cars, bullets, …) throughout the game. Look at the prog2 code to have an idea of how a abstract factory works.
    - Create two singleton classes: a **Random** class that generates random numbers and a **Transformation** class that converts pixel values to the visible 2D game world space of **[-4,4] x [-3, 3]**. The game logic library will only use the [-4,4] x [-3,3] coordinate system and will know nothing about the actual pixel resolution of the screen.
    - Provide a live scoring component that gets updated by using the **observer pattern**. A high score view should be present to show all-time best scores.
- Use of **smart pointers** throughout the whole project is **obligated**.
- To simplify the **collision detection**, work with rectangular shapes only. Don't focus too much on collision detection. Use the easiest, but **working** solution you can implement.
- Be creative but implement at least the basic entities found in the figure above. **Make it work first; then make it fancy. Not** the other way around.

Visual & interactive implementation:

- Use the SFML library (http://www.sfml-dev.org/) in your **visual** and **interactive** implementation of game's objects. Also in this case, use a separate namespace to encapsulate SFML implementations of your game objects (f.i.: **roadfighterSFML**).
- In your visual implementation, you can use your own images for the entities or you can find some online.
- Not every computer has the same speed and yet your player and ball should move with the same speed on all computers. Keep the time between two ticks so you can regulate the speed of your entities. Use the **C++ functionality** for this (use of **SFML::Clock is not allowed**).

Practical:

- A basic **working** implementation of the library with basic game logic (and more importantly: following clear design considerations) & visual representation in SFML are sufficient to get a passing grade.

- Apply proper **code commenting & documentation** of your API. This is implied to get a passing grade.
- Your code must be appropriately formatted using the formatting rules of prog2.
- The **CLion Code Inspection** feature will be used to assess code quality.
- Apply a **logical structure of the code & code base**, proper class construction that proves you understand C++. This is obviously implied to get a passing grade.
- Use **CMAKE** as your build system.
- Extensions are **up to you,** but put them **in your report** to be sure I notice them:
    - Multiple levels
    - Extra obstacles
    - …
- **Attention**: projects that do not compile or work (e.g., compiling error or segmentation faults when starting the application) automatically imply "student **failed** this part". The computers in G026 are used as the reference platform: your code should compile and work on those computers.
- The project has to be made and handed in **individually**. Of course, you can discuss design/ problems/solutions with other students as much as you like. Projects will be **tested on plagiarism**. If tested positive, the grade will be zero or worse.
- Describe your design and some of the choices you made in **a report** (ca. 2 A4 pages). This report has to reflect that you really thought about the choices you made for your design. Additionally, provide a **README** which explains the rules and controls of your game.
- Implement features **incrementally**! Make a **private Github repository** (using your student Github account) to frequently commit your code increments and **enable Travis** for your project. **Invite gdaneels for this Github repository.** The final version of your project should show a successful build on Travis.
- Good luck! If you have any questions, remarks or comments: glenn.daneels@uantwerpen.be or ask me during the lab sessions.
- Deadline: To be announced. (Somewhere in **January 2019**)
- Submit the final project and report **on Blackboard and by mail and by Github.**