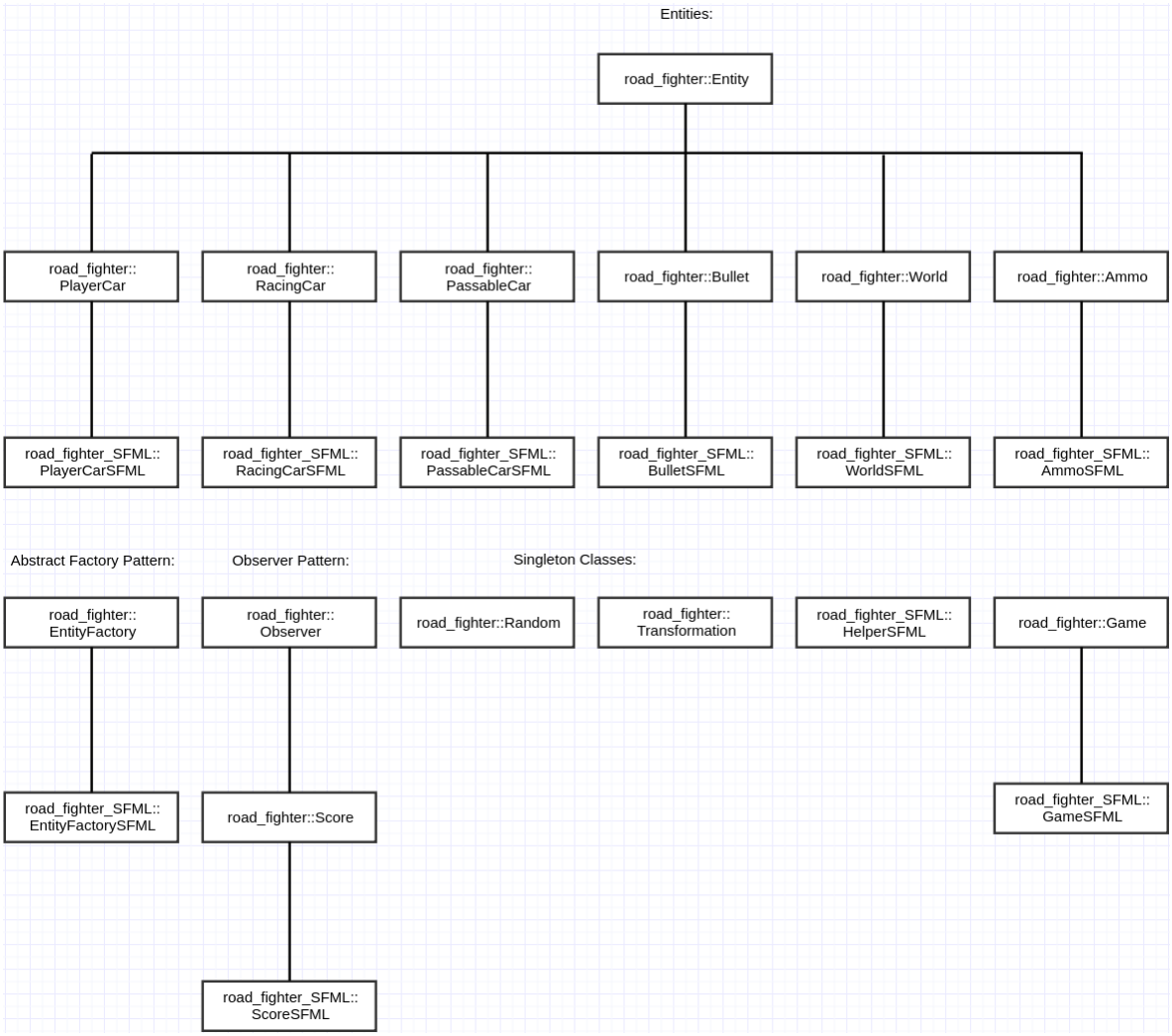
Name: Sam Legrand Student number: 20171079

Report: C++ Project — Road Fighter

Class Hierarchy:



Namespace road_fighter:

Contains mainly objects present in the example hierarchy provided in the project assignment, although it has been expanded with a couple of additional ones. The first one is ‘Ammo’ which refers to an entity which the player is able to pick up. It replenishes bullets, which a player can only hold a limited amount of (20). Furthermore, there is an Abstract Factory ‘EntityFactory’ which, as the name suggests, is meant to create Entities to place in our world. Next to that, the observer pattern is used for the score component and the singleton pattern for the ‘Random’ and ‘Transformation’ classes. It is important that the ‘Transformation’ instance’s resolution gets set, as it is used in the conversion of the coordinates. The ‘Random’ class uses an mt19937 random engine combined with a uniform real distribution to generate a random number between two bounds.

Name: Sam Legrand Student number: 20171079

Namespace road_fighter_SFML:

Contains all visual implementations for the entities. I also added a visual implementation for ‘World’. It uses two sprites which it repeatedly scrolls through to make it seem as if you are just driving through a level. It is also used to draw the finish line. The ‘GameSFML’ is just used for some additional input handling. Finally, there is a ‘HelperSFML’ singleton class which can be used to scale sprites and set a sprite’s position, given the world space dimensions or coordinates. I found that this was necessary to avoid code duplication in the entity classes.

Implementation details for logic:

The Entity class contains predefined bounds for the road in which objects should stay. These bounds are used in the movement implementation of PlayerCar. The constructor of Ammo, PassableCar and DrivingCar use these bounds to determine their random location on creation (using the Random class).

Movement for PlayerCar is pretty straight forward, a speed is used which can be increased or lowered by user input, when no input is detected it gradually moves towards 0. PassableCar objects have a constant speed, while RacingCars have variations in their speed (using Random).

The Score component can read and write a list of scores from and to a file, using file streams. At the GameEnd state, it updates this list and determines if the current score was a high score (meaning the highest score on the list), all scores lower than the highest 10 will be eliminated.

The World class contains by far the most functionality, it delegates all input, movement and display requests to its children. The Game class is responsible to add these children to World by using an EntityFactory. It also notifies the Score component of a GameEnd event (so it can display the high score view) and of any score changes that need to happen.

A large part of World is collision control, where it checks for every pair of entities if they’re colliding and executes the appropriate actions. A ‘type’ variable containing a string is used to distinguish entity types, this way all types of PassableCars can be contained within the same class. It also adds extensibility (a new PassableCar type is implemented by just adding another string).

I chose to keep the PlayerCar unique_ptr separate from the list of entities, as there can only be one. This also enabled me to call PlayerCar functions without a cast and ensures the list doesn’t have to be searched for the PlayerCar every time it’s used.

Name: Sam Legrand Student number: 20171079

Implementation details for visual:

Upon creation of an Entity, its texture will be loaded into a sprite and the sprite will be properly scaled. The draw calls will then display this sprite at the position determined by the logic library (after conversion using Transformation). I chose to keep a pointer to the RenderWindow in every entity so World could just call the draw function for every entity.

The main game loop was kept simple. It only contains the RenderWindow and the GameSFML object. Frame rate has been limited to 60FPS using the chrono steady_clock.