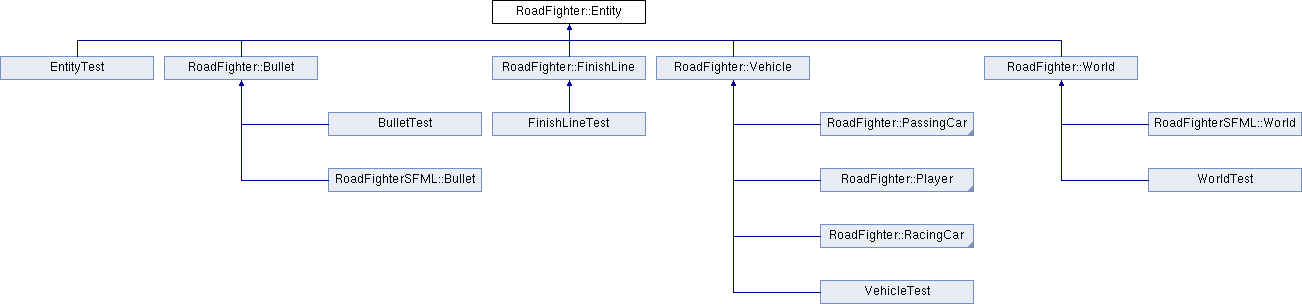
Freek De Sagher

S0171876

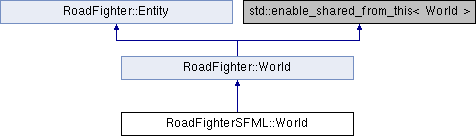
**Rapport examenopdracht:**

**Road Fighter**

**Algemene structuur**



De datastructuur heb ik ongeveer volledig overgenomen uit de opgave. De grootste base klasse is Entity. Hieruit worden de klassen Bullet, FinishLine, Vehicle en World afgeleid. Vehicle is op zijn beurt een base klasse waaruit de voertuigen Player, PassingCar en RacingCar zijn afgeleid.



De klassen Bullet, World, FinishLine, Player, PassingCar en RacingCar hebben op hun beurt een afgeleide klasse in de RoadFighterSFML namespace. Deze klassen zullen ervoor zorgen dat alle entiteiten correct weergegeven kunnen worden op het scherm.

De klassen uit de game logica hebben ook een afgeleide klasse die bedoeld is om de testen uit te voeren. Deze test klassen bieden geen extra functionaliteit, maar implementeren alle virtuele functies met een functie die niets doet. Een klasse met pure virtual functions is namelijk niet te instantiëren.

Voor het testing framework koos ik Googletest. Dit framework hebben we reeds gebruikt in vorige vakken en was daarom de beste oplossing.

**Player**

De player klasse stelt de speler voor. Deze is afgeleid van de klasse Vehicle.

Een speler kan in de vier richtingen (omhoog, omlaag, links, rechts) bewegen en versnellen en vertragen. Deze klasse heeft ook een reeks Observers. Deze observeren de afstand die de speler afgelegd heeft, de score van de speler en de snelheid van de speler. De score en de snelheid worden ook weergegeven op het scherm. De Player klasse houdt zelf ook de afgelegde afstand bij om dit zo gemakkelijker aan de World te kunnen communiceren.

**Passing Car en Racing Car**

Deze twee klassen zijn eveneens van Vehicle afgeleid. Een passing car kan enkel naar beneden bewegen. Een racing car kan de vier richtingen uit. Een passing car zal bij elke update gewoon verder naar beneden bewegen. Een racing car is iets complexer. Deze zal versnellen tot deze zijn minimumsnelheid bereikt en zal daarna random beslissen om te versnellen of vertragen. De racing car kijkt ook voor zich om te controleren of er geen andere wagen in de buurt is en zal deze soms proberen ontwijken.

Als een passing car geraakt is door een kogel zal deze vernietigd worden, een racing car zijn maximum snelheid gaat omlaag.

**Bullet**

Een kogel wordt geïnstantieerd op basis van de positie van de speler, en de hoogte van de speler. Op die manier is het onmogelijk dat de bullet de speler raakt, en dus moet collision tussen deze twee objecten niet gecontroleerd worden.

**Finish Line**

De finish line is ook een entity. Op die manier is het gemakkelijk deze aan de wereld toe te voegen zonder al te veel overhead.

**Verschil World en Game**

Dit zijn de twee belangrijkste klassen van het project. De World is het hart van het spel. Deze houdt alle entiteiten van het spel bij, is in staat nieuwe entiteiten aan te maken, handelt collision af en update alles waarover deze controle heeft.

Intuïtief zou men denken dat Game de klasse is die alles doet. Deze klasse zijn taak is echter veel minder complex als die van de World. Het enige wat de Game klasse doet is een state van de game bijhouden. (deze is op het moment SCOREBOARD of PLAYING). Op basis van deze state zal de game ofwel de update functie van de World aanspreken, of de update functie van het scorebord. De Game klasse is de klasse die de game loop bevat en bepaald welke grote delen van het geheel deze moet aanspreken.

De Game klasse zal ook de events zoals wanneer het scherm focus verliest en krijgt, exit functie, keyboard input… afhandelen.

**Entity Factory**

Deze klasse bevat een aantal functies om alle entiteiten die in de wereld kunnen zijn aan te maken. De entity factory zal correcte entiteiten aanmaken bv. Aan een speler worden 3 observers vastgemaakt.

**Random en Transformation**

Dit zijn twee Singleton klassen. Random zal random integers en random doubles kunnen genereren. Als seed voor de random getallen heb ik voor de tijd gekozen. Deze verandert constant en geeft dus ideale willekeurige getallen. Random device als seed gaf vaak hetzelfde resultaat en was dus niet ideaal.

De Transformation klasse zet posities uit de game logic om in scherm posities. De Transformation klasse houdt de grootte en breedte van het scherm bij. Deze worden gezet vanaf het window aangemaakt wordt aan de hand van een setter functie.

**Position**

Deze klasse wordt gebruikt om posities voor te stellen. Er staan geen randvoorwaarden op een positie (bv. X mag maximaal 4 zijn of Y mag minimaal -5 zijn). Op die manier is het mogelijk om buiten het bereik het game logic coördinaatstelsel objecten te positioneren en is het niet nodig om een aparte Position klasse te maken voor de SFML implementatie.

**Clock**

Deze klasse zorgt ervoor dat we een game tick kunnen specifiëren in de game loop. Daarnaast kan deze klasse ook een timer voorstellen, wat gemakkelijk is voor bijvoorbeeld een delay op het spawnen van Passing Cars te zetten.

Deze twee features waren perfect combineerbaar in één klasse en zitten daarom ook samen onder de Clock klasse.

**Scoreboard**

Het scorebord is een redelijk groot subonderdeel van het project en zit volledig verwerkt in de Scoreboard klasse. Deze leest een .ini bestand in waar de scores in opgeslagen zijn aan de hand van een externe parser. ([mINI parser](https://github.com/pulzed/mINI))

Indien een high score behaald is, dan zal het scorebord dit weergeven aan de gebruiker en vraagt deze om diens naam in te geven. De naam en score worden opgeslagen en teruggeschreven naar de file van zodra de destructor van het scorebord opgeroepen wordt. Zo zijn we zeker dat deze operatie slechts één keer gebeurt.

**Controls**

Op dit moment zijn de controls als volgt: pijltjestoetsen voor te bewegen (in de 4 richtingen), spatiebalk om te versnellen (loslaten om te vertragen), de -x- knop voor te schieten.

De SFML implementatie van de wereld bevat een functie om een keymap te initialiseren. Op die manier is het gemakkelijk om de controls vlot aan te passen in de code. Deze implementatie maakt het gemakkelijker om een feature toe te voegen aan de game waarmee de gebruiker zelf zijn controls aanpast zonder de source code aan te moeten passen. Ikzelf heb er voor gekozen deze feature niet te implementeren op deze moment omdat deze veel tijd in beslag zou nemen. Elke SFML key zou een tekst representatie moeten krijgen. Voor een extensie was dit zijn tijd niet waard.

**Extensies**

Er zijn ook enkele kleine extensies toegevoegd aan de opgave. Om gebruik te maken van de extensies moet het programma opgeroepen worden met de volgende mogelijke programma argumenten:

* -debug
* -windowed
* -reset-scores

De debug optie doet het volgende:

Wanneer debug modus geactiveerd wordt, dan zullen de colliders van alle entiteiten in de wereld zichtbaar worden aan de gebruiker. De grenzen van de wereld worden ook aangegeven aan de hand van een rode lijn.

De collider van de speler heeft een blauwe kleur, de colliders van de andere wagens zijn groen. Deze optie was een ideale tool om fouten op te sporen uit de colliders en collision detection.

Naast het weergeven van de colliders zal de debug optie er ook voor zorgen dat het minder lang duurt vooraleer eindstreep bereikt wordt. Op die manier kunnen de grootste features (inclusief scorebord) sneller getest worden.

De windowed optie doet het volgende:

Default zal het spel fullscreen opstarten. Om dit te vermijden kan deze optie gebruikt worden. Het scherm zal dan opstarten in een kleiner venster met een vaste grootte.

De reset scores doet het volgende:

Het scorebord zal alle scores bijhouden. Bij het installeren van het spel zal het scorebord enkele default namen en scores bevatten. Om het scorebord naar deze scores te resetten is deze optie noodzakelijk.

**Mogelijk probleem (en oplossing)**

Soms kan het zijn dat de map die de resources bevat niet correct wordt gekopieerd naar de plaats van de executable (in CLion bijvoorbeeld). De sprites zullen dan niet getekend worden op het scherm. Dit probleem is oplosbaar door gewoon het CMake project opnieuw in te laden. De map zou automatisch gekopieerd moeten worden naar de plaats van de executable.