Verslag Roadfighter

Senne Rosaer

27/08/2019

Het volledige spel begint vanuit de Game klasse. Deze zal alles initialiseren zoals de configuratie data, het graphische stuk en de objecten die nodig zijn in het spel.
Eerst zal de constructor van deze klasse de configuratie file, waarvoor we JSON gebruiken, parsen en alle informatie uit deze file in een ConfigData klasse steken. Deze klasse wordt gebruikt zodat de file niet meerdere keren geparst moet worden als verschillende objecten data nodig hebben uit dit bestand. Het voornamelijkste dat in dit bestand staat zijn de paden van bestanden die gebruikt worden zoals sprites en de level files maar ook veel nummers zoals bepaalde timers en afstanden worden daarin bijgehouden. De level files die ook vermeld staan in dit bestand zijn op deze manier opgebouwd:

```
{
        "Background" : ["../Sprites/road3.png","../Sprites/road3.png"],
        "enemies": ["rock","car2"]
}
```

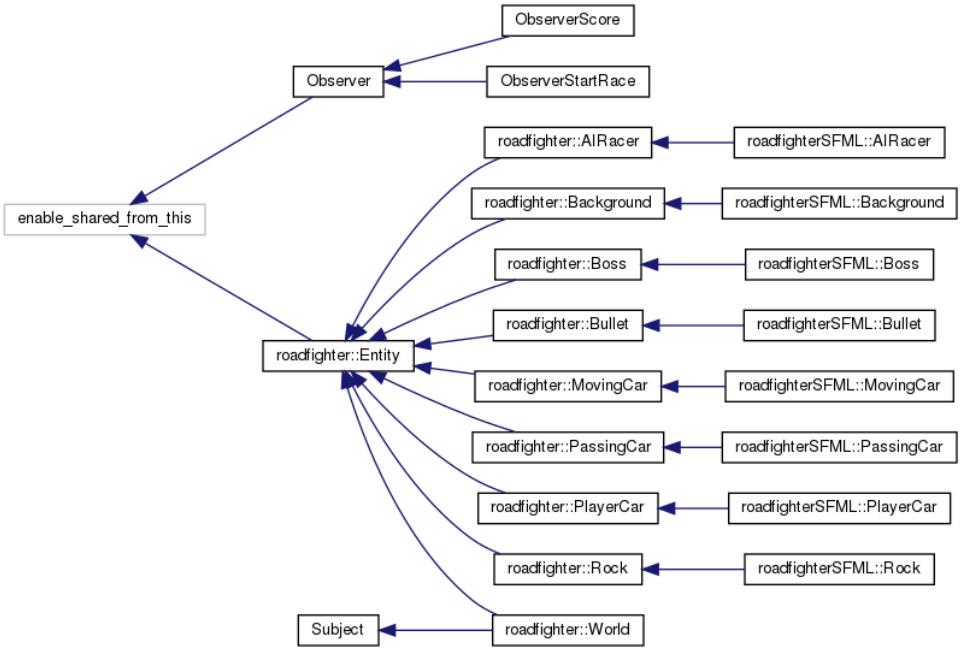
We kunnen dus aangeven welke sprites er gebruikt worden voor de weg en voor de finish (in dit geval zijn beiden hetzelfde aangezien het laatste level geen finish heeft maar een boss fight).
Er kan ook gespecifieerd worden welk obstakels er mogen voorkomen. We kunnen kiezen uit rock, car1 en car2. Car1 is de statische auto die voorbij rijdt en Car2 zal bewegen naar de speler als hij in de buurt komt. Het is op deze manier makkelijk voor elk level aan te passen welke obstakels mogen voorkomen.

In de constructor zullen ook nog de wereld en de verschillende staten van het spel geïnitizaliseerd worden.
Voor de rest loopt alles vanuit de run() functie van Game. Hier zal het window draaien met een vaste framerate door aan het einde van elke run de thread te laten slapen tot we alle tijd van een frame gebruikt hebben (33ms in dit geval voor 30 FPS). In een while loop gaat dan continu het spel uitgevoerd worden. Als we starten komen we op het menu waardat één van de drie levels geselecteerd kan worden waarvan het laatste level een boss fight bevat aan het einde. Eens een level gekozen is nemen we de benodigde informatie uit de level bestanden alles klaar te zetten. Eens we het laatste level afwerken gaan we door naar het scorebord. De menu en scorebord toestanden zullen ook in een aparte klasse afgewerkt worden.

Het menu is vrij statisch maar het scorebord is afhankelijk van de doorlopende scores van de speler. Elke keer als het laatste level en dus ook de baas verslagen is zal de score van de speler worden toegevoegd worden aan het scorebord indien mogelijk en in het rood weergegeven worden. Ondanks het feit dat de speler elk level met score nul begint worden de scores van elk level bijgehouden zodat van level één tot drie zonder stoppen doorspelen wel een invloed heeft op totale score. De score zal dus beïnvloedt worden door de afgelegde afstand (die elke keer wel hetzelfde is), aantal botsingen en aantal neergeschoten auto’s. Ik heb er wel voor gekozen om elke botsing hetzelfde aantal punten te laten afnemen omdat voor elk obstakel een botsing hetzelfde is ook al reageren ze anders met de omgeving.

Gedurende de loop van het spel zal de Game klasse continu informatie uitwisselen met de World zodat we zorgen dat de wereld bij elke uitvoering van de while loop zijn informatie aanpast. De data die we dan terug kunnen krijgen kan dan de aanvraag zijn om objecten aan te maken, een verandering van de toestand waarin het spel zich bevindt of basis informatie van de objecten om er zelf iets mee uit te voeren.

Voor objecten aan te maken is er gebruik gemaakt van abstract factory, de Game klasse is de enige die deze factory bezit dus alle objecten worden hierin gemaakt en zullen worden doorgegeven aan de wereld waar dan al de rest wordt uitgevoerd. Dit is omdat er bij de aanmaak van objecten wel nood is aan de graphische informatie voor de SFML kant van het spel en anders zou er in de wereld niet alleen logica staan. Het aanmaken van passerende auto's en stenen wordt dus wel geregeld vanuit Game door gebruik te maken van een timer en extra voorwaarden zoals de snelheid van de speler. Deze timers staan ook in de configuratie file waardoor ze makkelijk aangepast kunnen worden.



Het grootste deel van het programma bestaat uit alle objecten die zich in het spel bevinden en beschreven worden volgens het composition design pattern. Zo goed als elk object is een subklasse van Entity en is onderverdeeld in het logische en graphische gedeelte. Alle objecten reageren met elkaar of worden aangepast vanuit de World klasse; deze zal alles bijhouden als entities zodat we met dezelfde set functies alle benodigde acties kunnen uitvoeren.

virtual void	**draw** ()=0
virtual void	**update** ()=0
virtual void	**update** (int speed, std::shared_ptr< **roadfighter::Entity** > Player)=0
virtual int	**getSpeed** ()=0
virtual int	**Delete** ()=0
virtual void	**setDelete** (int del)=0
virtual bool	**Shoot** ()=0
virtual std::shared_ptr< **ObjBox** >	**getObjbox** ()=0

Dit zijn de enige functies die dus nodig zijn voor het grootste deel van de objecten aangezien de meesten een zeer gelijkaardige werking hebben. De enige twee klassen die in de World niet worden opgeslagen als een entity en dus niet volgens dezelfde regels werken zijn Boss en AIRacer. Aangezien deze compleet andere vereisten hebben zouden we te veel onnodige functies moeten toevoegen aan Entity vanwege de pure virtual functionaliteit. Hierdoor zouden veel klassen vol

staan met functies die niet gebruikt zullen worden. Zo heeft de computer gestuurde auto bijvoorbeeld constante informatie nodig over de andere objecten om te weten hoe hij daar op moet reageren en heeft de Boss meer acties nodig om aan te vallen.
De AI reageert dus op de objecten die boven hem komen door links of rechts te gaan. Er wordt enkel rekening gehouden met het voor hem dichtsbijzijnde obstakel dus een botsing is niet altijd uitsluitbaar. Aangezien deze computer soms sneller botst zal hij ook een hogere versnelling hebben zodat het een uitdaging blijft. Al deze informatie kan natuurlijk ook aangepast worden in de configuratie file.
Voor de gevect met de baas heb ik besloten het patroon vrij simpel te houden. De baas zal een constante beweging van links naar rechts uitvoeren zodat het geen statisch object is dat we te makkelijk kunnen raken. Hij zal ook op een bepaald interval stenen op willekeurige posities laten vallen die de speler moet ontwijken. Dit interval kan ook in de configuratie file aangepast worden om het gevecht makkelijker of moeilijker te maken.

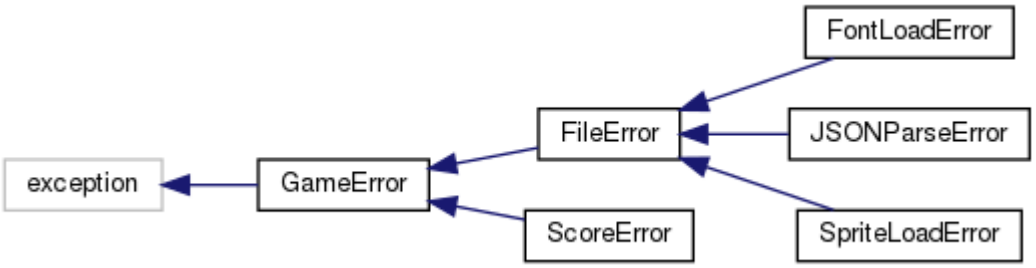
Andere objecten die wel een gelijkaardige functionaliteit hebben gebruiken dan de functies voorzien door Entity. De update functie zal elke frame uitgevoerd worden om bewegingen te bepalen en te zien of het object nog mag bestaan om dit daarna ook door te geven aan de graphische kant. De speler heeft dan weer iets andere functies die beïnvloedt worden door de toetsenbord input van de gebruiker. Een ander object dat andere functionaliteiten heeft is de achtergrond. Hierbij was er de mogelijkheid om SFML te gebruiken voor één zelfde sprite te laten doorlopen. Aangezien ik zo veel mogelijk zelf wou doen heb ik gebruik gemaakt van drie sprites achter elkaar die vanaf ze te veel bewogen zijn terug aan de bovenste geplakt worden. Dit zorgt voor hetzelfde effect dat we door blijven gaan. Aan elk einde van een level, wanneer er een bepaalde afstand is afgelegd, zal een vierde sprite die altijd boven het scherm staat mee bewegen om te simuleren da we een eindstreep voorbij komen. Deze gaan we een stuk voor dat we aan de eind afstand zijn beginnen bewegen zodat het ook echt op ons af komt en niet ineens er staat.

Zoals er gezien kan worden bij de hierachie gaat World als een kind beschouwd worden van Subject, dit is het object dat bekeken zal worden door de observers van het observer pattern. Dit is zowel voor de score als voor de timer aan het begin van elk level. Aangezien we niet willen dat we direct beginnen wordt er gekeken of we ons aan het begin van een level bevinden om dan eventueel een timer te laten aftellen. Ik heb ervoor gekozen dit ook met observer te doen aangezien zo op een makkelijkere manier zoals de score iets met de graphische kant kunnen doen van de wereld zonder het tussen de logica te steken.

De wereld regelt alles voor de objecten die hij bevat zoals het updaten, verwijderen, tekenen en botsingen. We controleren op botsingen door te zien of er tussen twee objecten hoeken over elkaar liggen. Afhankelijk van welke objecten een botsting hebben krijgen we ook een andere reactie. De speler of computer gestuurde auto verwijderen we niet na een botsing maar zullen een korte tijd moeten wachten voor ze terug verder kunnen. Ook de baas reageert anders want als deze geraakt is door een kogel gaan er levens af tot hij verslagen is en het spel gedaan is.
Bij elke botsing waarbij de speler betrokken is geven we wel informatie door aan de observers om de score te laten afnemen en als de speler met een kogel iets raakt om te de score te verhogen. In de wereld is er ook een controle zijn op wie er eerst aankomt, voor de computer aankomen resulteert in een bonus score.

Het gebruik van de singleton klassen Random en Transformation behoeven ook niet veel extra informatie. Alleen word Random meer gebruikt omdat er voor de auto's en stenen verschillende posities zijn om uit te kiezen en er ook voor de bossfight willekeurigheid nodig hebben voor de aanvallen.

Voor de rest is het enige waar nog veel keuzes bij gemaakt zijn de exception class.



Deze beginnen vanuit een base class GameError en zijn dan onderveeld in exceptions voor als er een probleem is met een bestand in te lezen of een gewone logica fout. De klasse FileError bevat ook het bestand waardoor er een fout veroorzaakt werd, aangezien dit geen fouten zijn die gehandeled kunnen worden door het programma zelf zullen we ze verder throwen en zorgen dat het programma eindigt. Een fout met de score kan wel opgelost worden, de score kan negatief worden door een botsing wanneer de score lager is dan wat er af gaat waardoor het negatief zal worden. Dit kan wel heel simpel opgelost worden door de score gewoon op nul te zetten. Ik vond dit de voornamelijkste dingen die fout konden lopen en dus ook in de exception class gezet konden worden.