Rapport Gradius

Het model:
Bij het model heb ik twee grote delen. De entities zijn alle dingen die in de wereld kunnen voorkomen en dan de wereld en het level zelf.
De entities heb ik dan weer opgedeeld in bullets, obstakels en schepen. De reden hiervoor is dat schepen allemaal kunnen schieten en naar alle kanten kunnen bewegen. In deze categorie zitten mijn AI's en het schip waar de speler zelf mee speelt. Voorlopig heb ik 2 AI's. De Follower gaat continu kijken waar de speler zich bevindt en gaat zo proberen tegen hem te botsen. De Shooter beweegt en schiet aan de hand van random getalletjes. Bullets bewegen enkel naar links of naar rechts, afhankelijk van wie de bullet geschoten heeft. Obstakels bewegen enkel van rechts naar links in de wereld. Hier heb ik twee soorten, de Border die twee levens aftrekt als je ertegen botst en dan nog de gewone obstakels waar je maar 1 leven door verliest. Het grootste verschil tussen de Border en het Obstakels is dat als het Obstacle het einde van de wereld bereikt (links) die zal verdwijnen terwijl de Border zichzelf terug zal verplaatsen naar het begin van de wereld (rechts). Zo zorg ik ervoor dat er niet continu entities aangemaakt en verwijderd moeten worden.
De wereld is eigenlijk een soort container voor alle benodigdheden voor een spelwereld te hebben, zo bevat die bijvoorbeeld de entities en het level. Het level is niet meer dan een queue van timestamps. Een timestamp is een bepaalde tijd wanneer er bepaalde dingen moeten spawnen. Dit is een zeer eenvoudige manier om levels te kunnen opstellen. De queue zorgt ervoor dat je makkelijk de timestamps die geweest zijn kan weg doen, zodat ze allemaal in de juiste volgorde gebruikt kunnen worden.
De wereld gaat er ook voor zorgen dat alle entities geüpdatet worden en kijkt steeds of er entities botsen.

De view:
De view bevat alle representaties van de entities van de wereld. Die representaties zijn voor elke entity hetzelfde buiten voor het Playership, omdat die een paar extra functies heeft. Zo zullen de spelers zijn levens op het scherm afgebeeld worden en als hij geraakt is, kan de speler voor 0.6 seconden niet geraakt worden. Tijdens deze 0.6 seconden zal de texture van het Playership veranderen om aan te geven dat die niet geraakt kan worden. Omdat het Model alleen iets afweet van zijn coördinatensysteem moeten alle getallen van het model omgezet worden naar de grote van het window. De transformatie gebeurt met de Transfomatie klasse. De entities kunnen ook een animatie bevatten en niet enkel een texture. De manier voor de animatie te tonen heb ik gebaseerd op een youtube filmpje waarin een persoon de basis van sfml uitlegt. Het komt er in het kort op neer, dat je een afbeelding neemt met allemaal kleinere afbeeldingen. Zo maak je steeds een texture met een stukje van die afbeelding en ga je na een bepaalde tijd naar een ander stukje van de afbeelding. Hierdoor lijkt het dat de texture beweegd/veranderd. Het filmpje kan u vinden via deze link:
https://www.youtube.com/watch?v=kAZVbPF6N4Q

Gebruik van MVC:
Mijn controller bevat mijn view en mijn model. De controller is eigenlijk degene die het spel doet draaien. Zo zorgt de controller ervoor dat het model een wereld en de view een window (voor de wereld in te tonen) aanmaakt. Hierin bevindt zich de whileloop die het spel laat werken. In die whileloop laat de controller de wereld continu updaten en kijkt het telkens na of het spel nog niet gewonnen of verloren

is. Soms slaat hij een loop over, de reden hiervoor is dat er een tijd bijgehouden wordt zodat het spel even snel zou spelen op een snelle computer als een trage. De tijd wordt berekend en bijgehouden door een Stopwatch klasse. Buiten het updaten en doorgeven van input van de user aan de wereld, manipuleert de controller niets aan het model of de view. Zo is hij dus erg onafhankelijk van beide.
De view en het model communiceren met elkaar aan de hand van een ObserverPattern. Als er iets in het model aangepast wordt, zal deze een event aanmaken. Dat event wordt dan doorgestuurd naar de view, die het direct zal afhandelen. De events die ik gemaakt heb, heb ik gebaseerd op de events van de sfml library. Dit is erg handig om extra dingen toe te voegen aan de wereld en zo blijven ook het model en de view erg gescheiden, wat natuurlijk de bedoeling is van MVC. De view zal nooit weten welk soort entity hij toont, enkel dat het een entity van de wereld is. De entities van het model bevatten wel een string dat het type van de entity omschrijft, maar die wordt enkel gebruikt om te weten welke texture er gemaakt moet worden voor die entity. De reden dat ik niet gewoon het pad naar de texture meegeef, is omdat ik niets, dat met iets grafisch te maken heeft, in het model wou steken.
Het ObserverPattern heb ik gebaseerd op een topic van stackoverflow, te vinden via deze link: https://stackoverflow.com/questions/2225162/observer-design-pattern-in-c

Probleem singleton en smartpointers:
Bij de Singleton klasse heb ik geen gebruik gemaakt van smartpointers. Ik heb de Singleton klasse zo gemaakt dat er ook maar werkelijk 1 object van die klasse kan bestaan. Als je dat wilt bereiken moet je de constructor private maken. Dat is een probleempje voor smartpointers. Als je een smartpointer aanmaakt gaat die standaard de default constructor van die klasse aanroepen, maar omdat die nu net private staat, kan die daar dus niet aan en is er een error. Hierdoor gebruik ik enkel hier raw pointers. Bij het maken van de Singleton klasse heb ik me gebaseerd op een topic van stackoverflow, te vinden via deze link: https://stackoverflow.com/questions/11452760/singleton-c-template-class

Inlezen van levelfiles:
Als het spel opgestart wordt, zal er eerst aan de speler gevraagd worden welk level hij wil spelen via een CLI. Aan de hand van de speler zijn/haar antwoord zal het juiste level bestand ingelezen worden.
Mijn levels zijn json bestanden. Aan het begin van het spel lees ik heel het bestand in 1 keer in. Zo kan het level niet gemanipuleerd worden tijdens het spelen.
De level bestanden zijn vrij eenvoudig opgesteld. Je hebt dus telkens een time die de tijd van een timestamp voorstelt en dan een lijst van entities die moeten spawnen op die tijd. De entities hebben een type en een lijst van getallen. Deze getallen stellen de x en y coördinaten, breedte en hoogte voor.
Al de timestamps die gemaakt worden, worden dan op de level queue gepushed van de wereld. Het gebruik hiervan is reeds uitgelegd in het stukje ‘het model’.