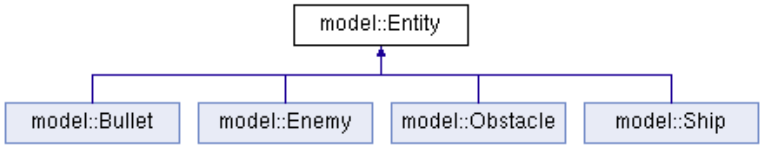
# Rapport Gradius

Igor Schittekat
s0160651

## 1 Structuur

Mijn programma is onderverdeeld in 3 delen, namelijk het Model, de View en de Controller. Dit onderscheid is duidelijk te zien in de folderstructuur. In de code is het te merken aan de namespaces.
Het model bestaat uit een overkoepelende klasse Level en een hiërarchie van entities, zoals hier te zien is:




De view bestaat uit een klasse Window en een klasse EntityObserver.
De controller bestaat uit de klasse Game. Het spel wordt volledig door de controller bestuurd, omdat de controller alle userinput ontvangt. De controller laat het model zichzelf updaten, en vervolgens de view zichzelf afbeelden op het scherm.
Naast het model, de view en de controller heb ik een namespace util toegevoegd, die de overige klassen overkoepelt. Deze klassen zijn de Stopwatch, die de fps regelt, de Transformation, die de omzetting maakt tussen levelposities en pixelcoördinaten, de Singleton, die ervoor zorgt dat er maar één Stopwatch en Transformation bestaat, de GradiusException die exceptions opvangt en de klasse Vec2, een 2D vector die gebruikt wordt voor levelposities en pixelcoördinaten.

## 2 Aanmaken van een spel

Om een spel aan te maken, wordt eerst de Game aangemaakt. Deze gaat op zijn beurt het Window en vervolgens het eerste Level aanmaken.
Het Window beschikt over een sh::RenderWindow, alle presentaties van entities(EntityObservers), alsook de te gebruiken textures. De EntityObservers houden niets meer bij dan de sprite die weer wordt gegeven, alsook de locatie waar deze sprite weergegeven moet worden.
Het Level houdt het schip bij, alsook een vector van al de andere entities die in het level voorkomen. Het schip wordt apart bijgehouden, omdat het door de user wordt gecontrolleerd. Verder wordt ook een pointer naar het Window opgeslagen, omdat mijn level een subject is, en wordt geobserveerd door het Window (Hierover later meer).
Iedere entity heeft een positie, een hoogte en breedte, en een eigen snelheid. Iedere entity is ook een subject, en heeft dus een pointer naar EntityObservers.

## 3 Spelen

Wanneer het spel wordt gestart, wordt eerst nagegaan of er userinput is. In dit geval wordt de bijbehorende actie uitgevoerd. Vervolgens wordt het model geupdate, en de view weergegeven. Dan wordt er nagegaan of het spel al is afgelopen. Indien de speler gewonnen is, wordt het volgende level ingeladen, indien hij verliest wordt het spel afgesloten. De speler wint op het moment dat hij alle obstacles gepasseerd is, en wanneer alle enemies verslagen zijn. Wanneer het spel nog niet afgelopen is, wordt alles herhaald. Om een stabiele fps te behouden, wordt gewacht tot er al $1/60^{ste}$ van een seconde verstreken is sinds het begin van de loop.

## 3.1 Updaten van het level

Wanneer het level wordt geupdate, wordt iedere individuele entity in het level geupdate. Bullets en obstacles bewegen gewoon in de richting waarnaar ze vliegen, enemies bewegen ook, en indien ze kunnen schieten, schieten ze. Wanneer dit gebeurt, wordt in het level een nieuwe bullet aangemaakt, en wordt in het Window een nieuwe representatie van deze bullet aangemaakt. Deze interactie tussen model en view gebeurt door middel van het observer pattern. De nieuw aangemaakte entities worden daarna toegevoegd aan de lijst met alle entities.
Wanneer er een bullet of obstacle van het scherm verdwijnt, wordt het verwijdert uit de lijst met alle entities. Verder wordt ook de representatie van deze entity verwijdert uit de view. Wanneer een enemy van het scherm verdwijnt, verscijnt deze terug aan de andere kant. Dit maakt het spel interessanter, daar de enemy echt vernietigd moet zijn door de speler, alvorens deze verwijdert wordt. Uiteindelijk wordt nagegaan welke entities met elkaar botsen (AABB collision detection), en worden volgende acties ondernomen:

- Ship met vijandig bullet: verwijder bullet, verminder levens van het ship met 1.
- Ship met enemy: verminder levens van het ship met 1.
- Ship met randobstakel: verminder levens van het ship met 2.
- Ship met normaal obstakel: verminder levens van het ship met 1.
- Enemy met friendly bullet: verwijder bullet en enemy.

## 3.2 Weergeven van het Window

Eerst wordt de achtergrond en het aantal levens weergegeven op het scherm, gevolgd door de representatie van iedere entity.

## 3.3 Toepassing van het observer pattern

Er zijn 2 plaatsen waar het observer pattern wordt toegepast, namelijk tussen het Level en het Window, en tussen de individuele Entities en de EntityObservers.
Zowel de klasse Level als de klasse Entity zijn de subjects, en hebben de functies addObserver en notify. addObserver voegt een observer toe aan de lijst van observers, en notify roept de functie update op op al de observers uit diezelfde lijst.
De klassen Window en EntityObserver zijn de observers, en hebben de functie update. update krijgt een Entity pointer mee, alsook een enumerator met de info wat er geupdate is.
Deze enumerator kan 4 verschillende waarden bevatten, alsook een niets zeggende 'NONE'. Volgende acties worden uitgevoerd wanneer desbetreffende waarden meegegeven worden:

- In Window:
  - CREATED: Een nieuwe EntityObserver wordt aangemaakten toegevoegd aan de lijst met EntityObservers (zowel in Window, als in bijbehorende Entity). Daarna wordt de EntityObserver geüpdate met notification CREATED.
  - DELETED: Er wordt voor iedere entity in de lijst van EntityObservers nagegaan of deze nog bestaan, en indien dit niet het geval is, uit de lijst verwijdert. Dit gebeurt door middel van weak pointers.
  - HIT: Dit wordt gebruikt om het aantal levens van het Ship te bepalen, zodat het correct aantal hartjes op het scherm weergegeven worden.
- In EntityObserver:
  - CREATED: De coördinaten en de grootte van de Entity wordt berekend door middel van de Transformation klasse, en de sprite wordt daaraan aangepast.
  - MOVED: De coördinaten worden opnieuw berekend en aangepast.

### 3.4 UserInput

De speler kan het Ship besturen en laten schieten. Dit gebeurt in de controller, in het begin van de loop. Wanneer het beweegt, wordt de functie Ship::move opgeroepen. Het schip wordt dan in de door de gebruiker bepaalde richting bewogen. Wanneer de gebruiker het Ship laat schieten, wordt nagegaan of het weldegelijk mag schieten. Dit is een kleine delay die tussen de verschillende kogels zit, zodat hij ze niet iedere frame kan laten afvuren. Wanneer het Ship mag schieten, wordt een nieuwe friendly Bullet aangemaakt op de postite van het Ship.

### 3.5 Types van Enemies

Er zijn 2 verschillende soorten Enemies:

- Flying Enemies: Deze vliegen doorheen het scherm, van boven naar beneden en omgekeerd.
- Shooting Enemies: Deze vliegen van rechts naar links, en schieten Bullets af wanneer ze kunnen schieten (weer met een kleine delay tussen).