

Tyrian

RUBEN VAN ASSCHE - S0122623

In dit rapport zal ik enkele van mijn beslissingen die ik maakte tijdens het programmeren van het spel Tyrian voor het van Gevorderd Programmeren toelichten.

0. Structuur

Zoals gevraagd is de game opgedeeld in 2 stukken: een library met de logica van de game en een deel gekoppeld met SFML voor een grafische weergave.

1. MVC

Ook al is de code opgedeeld in 2 stukken, er zit toch een MVC(Model View Controller) structuur in. In het library gedeelte zitten de controllers en de models en in het grafische gedeelte zitten de views. Er zijn 2 controllers aanwezig namelijk de game controller die samenwerkt met een world model en ervoor zorgt dat de game werkt. En een HAL9000 controller waar alle artificial intelligence in zit zoals het vuren van kogels uit vijandelijke schepen, het verplaatsen van vijandelijke schepen en het controleren van collision control. Wanneer de game controller aangemaakt wordt, zal er via compositie automatisch een HAL9000 controller aangemaakt worden waarmee de game controller samenwerkt. Tevens wordt er in de game controller een world gecreëerd, dit is een model waarin alle entities in een wereld worden opgeslagen zoals de schepen, kogels en achtergronden. Deze kogels en schepen zitten allemaal als pointers opgeslagen in lists(we hebben praktisch nooit een index van een schip nodig en doorlopen bijna altijd alles dus daarom koos ik een lijst inplaats van een vector) en zijn beschikbaar gesteld aan de beide controllers.

De models zijn allemaal afgeleid vanaf een base entity die wat basis functionaliteit(locatie, grootte, botst met een andere entity check) bevat voor een entity zoals een grootte, een locatie en een visibility. Het Schip entity stelt een schip voor wat een Gun entity bij het aanmaken meekrijgt. Zo'n gun heeft een bulletfactory voor het aanmaken van kogels. Eén speciale entity is de Tile entity, deze regelt de achtergrond van de game en zal bijhouden hoe ver we al in de wereld zijn gevlogen en wat we allemaal zien.

De views zitten in de grafische voorstelling en krijgen bij het aanmaken een pointer mee naar één van de entities welke ze voorstellen. Dit gebeurt via de bridge(zie verder). Elke View leidt ook wat functionaliteit af van zijn basisklasse View.

Ik heb uiteindelijk gekozen voor MVC omdat indien het goed geprogrammeerd is ervoor zorgt dat het heel makkelijk is om extra functionaliteit toe te voegen en omdat het de code vrij overzichtelijk houdt.

2. Libraries

In de Library zitten ook enkele libraries, deze zorgen ervoor dat bepaalde taken die de controller uitvoert en enkele speciale typen objecten die nodig zijn aanwezig zijn. De filelodder zal ervoor zorgen dat alle nodige xml files worden ingelezen.

De bridge is een volledig virtuele klasse en dient geïmplementeerd te worden in het grafische gedeelte, ze bevat 4 functies: schip toevoegen/ verwijderen en kogel toevoegen/ verwijderen. Bij het toevoegen van een schip/kogel zal in het grafische gedeelte een pointer gegeven worden naar de entity in world waar een View voor moet gebouwd worden. Vanaf dat punt kan de bridge dan de logica van het grafische gedeelte gebruiken om de View op te bouwen. Bij het verwijderen van een schip/kogel wordt opnieuw een pointer meegegeven naar een entity in world die verwijderd moet worden, in dit geval steken alle pointers naar de Views in de bridge klasse zelf maar het hangt af van de grafische implementatie hoe dit wordt afgehandeld(het verwijderen van het View object).

De bridge zorgt ervoor dat we de models en controllers die in de library zitten en de views die in het grafische gedeelte zitten toch nog kunnen aan elkaar koppelen.

De direction is een encapsulatie van een richting(op, neer, links, rechts), de Vector klasse stelt een Vector(lineair, geen lijst waarden) voor met alle functies die erop vereist zijn. En zal dienen voor vectoren aan te geven en punten in het library gedeelte. Als laatste is er Size, dit is een encapsulatie van een grootte(breedte en hoogte) welke handig is om de grootte van Entities aan te geven.

Naast deze klassen zitten er in de external map nog enkele klassen die ikzelf niet heb geschreven. Deze zorgen voor: een testframework voor het kleine aantal tests die ik heb geschreven, een systeem om te kijken welke bestanden er in een map aanwezig zijn.

3. Factories

Er zijn 3 soorten factories: een bulletfactory die kogels genereert, een shipfactory die schepen bouwt en een gunfactory die geweren bouwt voor schepen. Elke factory krijgt een pointer naar de world entity mee om zo ineens de kogels en schepen in de juiste lijst te plaatsen. De plaatsen waar de factories worden opgeroepen verschillen nogal: de shipfactory wordt opgeroepen in game bij het laden van een deel van een level dat gespeeld moet worden, de gunfactory wordt in de shipfactory opgeroepen en geeft elk schip direct een gun. En als laatste wordt voor elke gun gebouwd door de gunfactory een bulletfactory gebouwd.

Ook al werd er in de opdracht vermeld dat we ene abstract factory moesten gebruiken, heb ik uiteindelijk toch gekozen voor de iets simpelere versie van een factory omdat een abstract factory heel wat extra complexiteit zou toevoegen voor iets dat vrij simpel is, en als we het uitbreiden(bv. meer soorten schepen toevoegen) toch nog simpel blijft.

4. Stages

Zijn een deel van het grafische gedeelte en worden aangemaakt in de main, dit zijn scenes waarin de game zich bevind namelijk eerst een openingsscherm daarna een selectie van level scherm en dan uiteindelijk de game zelf. Wanneer de game stopt wordt er ook nog stage getoond die een bericht geeft of de speler al dan wel of niet gewonnen is. Stages maken het makkelijk om de verschillende grafische stukken van de game op te delen en zo ervoor zorgen dat alles leesbaar blijft.

5. Helpers

Dit zijn enkele klassen in het grafische gedeelte die ervoor zorgen dat we de bridge kunnen slaan tussen het grafische gedeelte en de library(de Bridge). De stage base class bevat een window class waarin een SFML window zit geëncapsuleerd omdat er enkele verschillen tussen de coördinaat systemen van SFML en mijn library zitten waarmee we niet steeds willen lastig gevallen worden en dus zorgt deze klasse ervoor dat we verder kunnen werken zoals met een normaal SFML window werken. De Button klasse is een knop die getekend kan worden door SFML met enkele speciale eigenschappen zoals kleur en lijndikte. De Assets klasse is een singleton die doorheen het gehele grafische gedeelte wordt gebruikt om resources in te laden, door slechts één keer een resource in te laden zorgen we ervoor dat ons geheugen niet te veel belast wordt. De input klasse is een singleton die SFML events ontvangt en waarmee dan gecheckt kan worden welk event er gebeurt is. Als laatste is er een stopwatch klasse die tussen de ticks van een game de tijd bijhoudt en deze doorgeeft aan HAL9000 om zo te kunnen berekenen in hoeverre ships zich mogen verplaatsen. Zo zorgen we ervoor dat de game op elk platform even snel werkt.

6. Levels

Levels bevinden zich als XML files in de Levels map. De XML files bevatten ten eerste een naam van de level en een moeilijkheidsgraad(tussen 1 en 5). Vervolgens bevatten ze enkele stages waar telkens een lading schepen op het scherm worden gezet.

7. Nieuw in deze versie

Multiplayer

Kan geselecteerd worden in het beginscherm. T.o.v. de vorige versie is er een extra object toegevoegd in World waarin alle spelers zitten zodat deze verder in het spel kunnen worden gebruikt.

Besturing 2de speler

i → naar boven
l → naar rechts
j → naar links
k → naar onder
h → schieten

HighScores

Worden bijgehouden in een xml bestand in de data map, telkens wanneer een speler verliest of wint worden deze geüpdatet. De klasse die dit beheert is via een singleton pattern geschreven en bevat slechts één functie: add(voor toevoegen van nieuwe score). Bij het constructen van een object van deze klasse wordt de xml file ingelezen en bij het destructen van het object zal het highscores.xml bestand geschreven worden.

Levels

Maakt nu gebruik van Boost Property Tree inplaats van pugixml.

Preferences

Alles van magic Numbers : levels, configuratie van het spel(hoogte, schip van de speler, paden naar asset mappen), schepen(grootte, texture informatie, ...), kogels(grootte, texture informatie, ...) worden gelezen uit de bestanden in de Data map.

Constness & Memory management

Aanzienlijke verbeteringen

Bugfixes

Achtergrond die niet mooi werd gerenderd