Space Invaders
Project Gevorderd Programmeren 2015 - 2016

Stijn Janssens
20142002

januari 2016

1 Introductie

Ik verwijs om te beginnen door naar mijn README.md bestand in de project root voor alle details ivm met gameplay en installatie van het spel. Voorts verwijs ik ook door naar het html/index.html bestand voor de doxygen pagina van mijn project. Op deze pagina kan u alle nodige inheritance diagrams terugvinden alsook de documentatie van de .h files.

2 Design

Ik ben uiteraard vertrokken vanuit het model-view-controller design welke ik hier zal toelichten.

2.1 View

Mijn view bestaat uit de view klasse, dit is de klasse die zorgt voor de correcte rendering van alle resources alsook voor het afspelen van de correcte geluiden op het juiste moment.
In deze namespace zitten ook de observers. De observers bestaan uit een audio-observer en een visual-observer. Elke entity in het spel is gelinked met beide observers. Wanneer een significante wijziging gebeurt in een entity zal deze de functie notify(si :: model::Event) oproepen met een Event (enum) als argument om duidelijk te maken wat er is gebeurt. De observer zal dan zijn view updaten door op basis van de event de correcte methode in de view aan te roepen.

De enige koppeling tussen view en model gebeurt bij het creëren van een nieuwe entity, waarna deze (via de observer uiteraard) wordt gemapped naar een correcte sprite in de view.
Ik heb gekozen voor een fullscreen view. Dit heb ik mogelijk gemaakt door vanuit zeer grote sprites te vertrekken en deze te downscalen tot de correcte grootte.

Dit zorgt voor hetzelfde spelbeeld en identieke gameplay op elke mogelijke resolutie.

2.2 Controller

De controller is de klasse die alle user input afhandelt. Binnenin de controller namespace zitten de utility klasses Stopwatch en Randomgenerator (Singleton patroon).
De controller is de interface tussen de gebruiker en het spel. Hij regelt de overgangen van de verschillende gamestates (intro, main, ending).
De volledige gameloop wordt gerunned in de controller. De controller kijkt naar user input, de controller geeft door aan de world (zie infra) of er een element moet geupdate worden (want alle tijdsafhankelijke zaken worden berekend in de controller en bijgehouden in de stopwatch klasse, i.e. enemies die bewegen, schieten, ...) vervolgens zal de controller de opdracht geven aan de view om te renderen.
De randomgenerator wordt gebruikt bij alle functies die random moeten voorkomen (enemy die schiet, moment waarop hij schiet,...)
De stopwatch bevat een reeks timepoints (enum) die kunnen geupdate worden of verkregen worden, deze houden bijvoorbeeld het laatste moment bij waarop de enemies bewogen hebben.

2.3 Model

Het model bevat enkel de logica en de huidige game state van het spel. Bij het parsen van een level door de world (nadat het geclicked was in het menuscherm) wordt het model opgezet.
De world krijgt opdracht van de controller als er nieuwe entities moeten worden gecreëerd en houdt al deze elementen bij in overeenkomstige vectoren. Binnenin de world klasse zitten overkoepelende berekeningen zoals collision detection, berekenen van de eerste rij vijanden (enkel zij kunnen schieten), het moven van de volledige groep vijanden etc.
De game-entities van het spel bevatten vooral de inheritance structuur van de verschillende objecten alsook entity-specifieke eigenschappen. Een playership en een bullet hebben bijvoorbeeld een verschillende move methode, etc. Dit zit in de entity-implementatie zelf.

Ik verwijs hierbij zeker door naar de doxygen file voor bijhorende inheritance diagrammen.

2.4 Levels

Voor de levels heb ik het populaire XML-formaat gebruikt (parsen gebeurt via de tinyxml library)
Dit gaf me de mogelijkheid om object-gebaseerde levelspecificaties te doen. De

levels zijn te vinden in de levels map en zijn naar hartelust aanpasbaar. Momenteel zijn ze naar mijn mening gerangschikt van makkelijk naar moeilijk.

2.5 Gameplay en extra features

Voor de controls verwijs ik nogmaals naar de README.md file.

Ik heb de basisfunctionaliteit geimplementeerd en heb op de meeste plaatsen gewerkt met aanpasbare waarden.
Dit wil zeggen: In elke level xml kan gespecifieerd worden wat de rate of fire is van de speler, van de enemies, hoe snel alles kan bewegen, hoeveel levens ze hebben, ... Dit geeft een grote flexibiliteit naar gameplay van mijn spel. Het is afwijkend van het origineel waar er maar 1 kogel van de speler tegelijk op het scherm kan zijn.
Voorts heb ik ook statische schilden toegevoegd (waaronder de speler kan schuilen) met variabele levens. Beide partijen kunnen deze kapot schieten!

Mijn belangrijkste extra feature werd echter mogelijk gemaakt door het gebruik van rate of fire.
Ik heb 2 verschillende powerups geimplementeerd (deze spawnen op random momenten op random plaatsen, de frequentie van spawning, de tijd dat ze op het scherm blijven en de tijd dat je ze kan gebruiken zijn allemaal instelbaar in de xml.)

De shootpowerup zorgt voor een korte verhoging van de rate of fire van de speler (hoeveelheid instelbaar in de xml)

De shieldpowerup zorgt voor een tijdelijk krachtveld dat rond de speler wordt geplaatst en dat meebeweegt als de speler zich verplaatst (krachtveldlevens zijn ook instelbaar in de xml).

De powerups "refreshen" als u er een nieuwe oppikt. Dit wil zeggen dat wanneer een shootpowerup actief is en u tijdens deze periode een nieuwe shootpowerup pakt, de powerup pas uitgewerkt zal zijn als de tweede powerup verloopt.
Beide powerups kunnen gecombineerd worden.
Dit geeft uiteraard een zeer grote uitbreiding naar gameplay van het spel toe.

Voorts heb ik twee verschillende eindes voorzien aan het spel (een goed en een iets minder goed...) aan u om ze beide te ontdekken.

Rest mij u enkel nog te vertellen om zeker het geluid aan te zetten.
May the Force be with you!

Stijn Janssens, januari 2016