Project Space Invaders

Basil Rommens

January 19, 2020

About

We kregen voor het vak 'Gevorderd Programmeren' de taak om een versie van het befaamde spel 'Space Invaders' te maken. Dit moest in c++ met de graphics library SFML (2.4.2). Ik gebruik hiervoor c++14. Ook moesten we gebruik maken van design patterns, die het project meer flexibel en rigide maken.

1 Design

Om het design van het project kregen we hulpmiddelen en verplichting van de assistent. We gebruikten deze in het volledige project en lichten deze ook toe in de volgende paragrafen.

MVC-Pattern

Er werd ons gevraagd om dit project het MVC-pattern toe te passen. Dit was nodig om de drie hoofdzaken van de game apart te houden. Met name het model/data, de view/grafische weergave en de controller. Zo kunnen we de view veranderen als het later nodig is en als we veranderingen in het model aanbrengen deze geen effect zullen hebben op de view en de controller. We kunnen zo nodig de controller verbeteren zonder dat we invloeden ervan gaan zien in het model of in de view. Ik heb dit verwezenlijkt door deze drie delen eerst in zelfbeschrijvende namespaces te steken zodat ze makkelijk herkenbaar zijn. Het probleem dat we nog moeten oplossen is de communicatie tussen deze 3 delen. De oplossing hiervan wordt beschreven in de volgende paragraaf.

Observer Pattern

We realiseerden dit patroon door 2 klasses te maken, de Observer en Subject klasse. De Observer klasse is het object dat bekeken wordt en een notificatie zal ontvangen indien het Subject een notificatie verstuurt. Het is nodig dat de Subject klasse object dit Observer object heeft. We gebruiken dit patroon om de communicatie van de MVC-pattern mogelijk te maken. Ik doe dit door elke klasse van het MVC-pattern over te laten erven van minstens 1 van de 2 klassen. Zo creeer ik subklasses die zich hetzelfde zullen gedragen als de superklasse. Ik laten de Controller zal enkel van de Subject klasse overerven omdat deze klasse niets van notificaties moet ontvangen, ze moet ze enkel versturen. Ook zal de Controller klasse de event queue verwerken van de game. Het model daarentegen moet van beiden overerven. Dit gebeurd via de klasse Entity, die de superklasse is van het volledige model. Zo kan het model zowel notificaties van de Controller ontvangen, als notificaties naar de View klasse versturen. Daarom dat de view enkel moet overerven van de Observer klasse.

Event Queue

Dit patroon is niet expliciet door mij geimplementeerd, maar wordt wel gebruikt bij het verwerken van events. Deze events zijn het sluiten van de window en al de keyboard inputs van het spel. Ze worden in FIFO volgorde uitgevoerd. Enkel de toetsen die zijn geprogrammeerd worden herkend. We gebruiken dit patroon in de Controller klasse maar ook bij het displayen van een bericht, waar je het spel kan herstarten.

Game Loop

In elk spel kunnen we een game loop terugvinden. In dit geval vinden we die terug in de klasse Game. We gebruiken dit om elke 1/60te van een seconde de game te updaten. Om die tijd correct te laten verlopen hebben we 2 functies. De eerste is de start() member functie van de klasse Stopwatch om de start van een nieuwe loop aan te duiden. De andere is de wait() member functie van de Game klasse om te wachten vooraleer verder te gaan met een nieuwe loop iteratie. Daarnaast runnen we ook de Controller. Deze neemt al de input waar en verwerkt de event queue, en zal ook al de entities updaten. Tot slot vinden we Draw-¿view() terug. We gebruiken dit stuk code om de view te updaten.

Polymorfisme

Doorheen het project maakte ik gebruik van polymorfisme. We vinden dit terug in het model. Waar de Entity klasse zorgt voor het skelet van het model. Iedere klasse in het model erfde van minstens een van beide Observer Pattern klasses. We zien ook in de header file van de klasse dat we erven van een zogenaamde 'std::enable_shared_from_this⟨Entity⟩' klasse. Dit is een klasse die ons heeft toegelaten om shared this objecten van de huidige klasse makkelijk te construeren. We doen dit via een overgerfde functie.

Singleton Pattern

Naast het MVC en observer pattern waren we ook verplicht het singleton pattern te implementeren op 2 klassen. Het doel van dit patroon is om slechts 1 instantie van een klasseobject in het volledige programma te hebben. Ik heb dit gebruikt om 1 stopwatch object te hebben over het volledige spel. Zo kan ik op elk moment de tijd raadplegen indien het nodig is. Ook hadden we de transformation klasse die volgens dit patroon werd geimplementeerd. We implementeerden dit op deze manier, omdat ze zich altijd op dezelfde manier gedroeg ongeacht de waarden die er in werden doorgegeven.

Parser class

Ik implementeerde een parser klasse om de code meer flexibiliteit te geven. Zo kan de gebruiker de parser klasse herimplementeren. Zo nodig kunnen we een parser schrijven voor andere input formaten. Zonder dat er iets moet gebeuren

aan de andere code. Ik koos hiervoor zodat we in de klasses geen vreemde code hadden. De files die geparsed worden zijn json files, aangezien ze enorm eenvoudig zijn in gebruik. Om de json files te parsen heb ik gebruik gemaakt van de nlohmann::json parser.

Collision class

Om collision flexibiliteit te geven implementeerde ik deze in een aparte klasse. Zo kunnen er in die file de nodige aanpassingen gebeuren om collision detection te verbeteren.

2 Exception Handling

Dit deel hebben we toegevoegd aan het project zodat de user op de hoogte is van fouten die er gemaakt worden. Deze fouten kunnen sterk verschillen in aard. Zo kunnen we bij het parsen van de input files exceptions terugvinden. Zo worden er exceptions gethrowd vanaf dat 1 van de files niet kan worden geparsed. Sommige exceptions worden lokaal opgevangen terwijl anderen worden geforward. Zo kan het zijn dat we bij het inlezen van de enemies een foute enemy proberen in te lezen. Dit zal dan zorgen voor een niet gegenereerde enemy. Als dit gebeurd zal er een foutboodschap bij vermeld worden om de gebruiker hiervan op de hoogte te stellen.

3 Sprites

Sprites zijn gebaseerd op de standaard grootte van het scherm 800 x 600. Dus zullen op deze manier geschaald worden indien ze op een anders vormig rechthoekig scherm geladen worden.

4 Travis

Oorspronkelijk moesten we het project met travis laten werken, maar dit werd doorheen het jaar geschrapt. Ik had tegen dan een manier gevonden waarop het project gecompileerd kon worden op travis. Dus zit dit in het project. Aangezien dit mij niet zelf lukte heb ik een vermelding naar de persoon die deze travis file heeft doorgestuurd in de travis file gezet. U kunt in de de readme de build status van het project zien.

5 Documentation

Om het project een duidelijke documentatie van de API te geven heb ik gebruik gemaakt van doxygen. De documentatie zit standaard bij het project en kan ook gegenereerd worden. We vinden de methode om deze documentatie te openen terug in de readme.