# Voorwoord

Na zo’n acht weken geleden in het diepe te worden gegooid met dit project, kunnen we nu toch wel stellen dat het een geslaagd project is.  
Met niet al te veel kennis werd ons opgedragen een 3D spel te maken ter afsluiting van onze minor. Een samenwerkingsverband waarin niemand elkaar kende, en een opdracht die groot zo leek dat beginnen onmogelijk leek. Toch is het gelukt, een geslaagd spel met als titel “Medieval Invasion”. Na een brainstormsessie kwamen we tot de conclusie een origineel thema voor ons spel te willen. Een spel met als thema de middeleeuwen, in combinatie met een alien invasie. Tijdens het proces zijn we veel problemen tegen gekomen, met de hulp van de studentassistenten waren deze problemen echter altijd goed te overkomen. Dank daarvoor aan Julian Faber, Tim Rensen en Bert **ACHTERNAAM**.

# Inhoudsopgave

# Introductie

Dit rapport heeft betrekking op het in Java geschreven 3D spel Medieval Invasion. Dit spel is gemaakt ter afronding van de minor Software ontwikkelen en toepassen. De game is gebouwd op de basis van de Java files van “MazeRunner”, verkregen bij het vak Computer Graphics.

Voor het spel werden een aantal eisen gesteld. Zo moest het een 3D Java game betreffen met daarin een aantal aspecten. Hieronder vielen onder andere het programmeren van vijanden met enige intelligentie, het maken van een leveleditor waarin volledige levels gemaakt kunnen worden, een scoresysteem en een loginsysteem. Al deze eisen zijn op eigen wijze geïnterpreteerd en geïmplementeerd in het spel. Door het spel op een incrementele manier te bouwen is het spel gedurende het hele proces een werkend geheel gebleven. Hierdoor konden nieuwe methoden makkelijk geïmplementeerd en gelijk getest worden. Dit kwam de snelheid en overzichtelijkheid van het werk ten goede. Op deze manier werd gelijk duidelijk waar de eventuele ontwerpfouten in het spel zaten. Ook was op deze manier eenvoudig en snel te achterhalen waar eventueel performance-problemen van het spel zich bevonden. Dit in combinatie met GitHub voor het versiebeheer zorgde ervoor dat het ontwikkelen van het project een vloeiend geheel was.

# Game Design

Het originele thema van de game, een alien invasie van een middeleeuws kasteel, is uitgewerkt tot een First-Person actie game. Hierbij draait het voornamelijk om het doden van de aliens doormiddel van een zwaard of pistool. De speler heeft de mogelijkheid om de standaard campaign te spelen of om zelf levels te ontwikkelen en spelen. Bij de campaign is het uiteindelijke goal het vinden van de uitgang van ieder level, waardoor de campaign uitgespeeld kan worden. Bij eigen levels kan in ieder level een uitgang geplaatst worden. Hierbij wordt in de level editor de mogelijkheid geboden om een volgend level te selecteren of het einde van het verhaal aan te geven.

De uitgang van een level moet gevonden worden, maar ondertussen wordt de speler gehinderd door vijandelijke aliens. Hun doel is het elimineren van de speler. Doormiddel van pathfinding en swarm-intilligence zijn deze vijanden slimmer gemaakt. Het verslaan van vijanden levert punten op voor de speler. Upgrades, zoals nieuwe zwaarden en pistolen, kunnen het verslaan van vijanden makkelijker maken.

Score-multipliers verhogen de score bij het verslaan van een enemie. Hierdoor heeft de speler een extra uitdaging, versla voor ieder level de highscore. De highscores worden opgeslagen in de database en kunnen later teruggekeken worden. Levels kunnen sneller uitgespeeld worden door een speed-upgrade te pakken en hierdoor sneller door het level te bewegen. Voor het uitspelen van de levels moeten verschillende deuren geopend worden, Hierdoor zullen delen van levels meerdere malen gepasseerd moeten worden, waardoor het vrijwel onmogelijk wordt om alle vijanden te ontwijken.

Een level kan meerdere verdiepingen bevatten, waardoor het spel de extra dimensie beter benut. Een speler kan zich omhoog en omlaag verplaatsen doormiddel van hellingen in de levels. Tevens is er zwaartekracht toegevoegd, zodat de speler door gaten in het een verdieping naar een onderliggende verdieping kan vallen.

De campaign bestaat uit drie korte levels:

* Het eerste level bevindt de speler zich in de catacombe van het kasteel.
* Vervolgens bereikt de speler het plein van het kasteel waar veel vijanden verslagen moeten worden.
* In het laatste level moet een toren beklommen worden en een laatste vijand verslagen worden boven op de toren.

# GameObject: Enemy

In het spel is ervoor gekozen om 3 verschillende vijanden te implementeren. Er is een vliegende vijand, genaamd Bathos, een kleine predator en een iets grotere predator (“grote” predator ziet er hetzelfde uit als de “kleine” predator, maar is iets groter). Er is één enemy klasse die gameobject extends. In de leveleditor beaaplt een speler wat voor soort vijand hij wil en aan de hand van de uitput van de leveleditor wordt in de constructor de juiste vijand geladen met bijbehorende waardes. Verder worden de modellen van alle drie de vijanden vanuit .obj files geïmporteerd. De modellen van zowel bathos als predator zijn gedownload.

Het Bathos model is na het downloaden eerst omgezet naar de goede afmeteingen en vervolgens opgesplits in drie aparte modellen: lichaam en twee vleugels. Dit is zo gedaan zodat het animeren van de vleugels makkelijker verloopt. Het animeren van het gehele model vereist namelijk het inladen van meerdere modellen alleen voor Bathos. Door de vleugels apart in te laden is het mogelijk doormiddel van rotaties en translaties de vleugels te animeren zonder gebruik te maken van meerdere modellen.

De beide predator modellen hebben een soortgelijke behandeling gekregen. Het predator model is opegesplitst in 5 aparte modellen: twee armen, twee benen en een lichaam. Deze opsplitsing heeft dezelfde oorzaak als het Bathos model, namelijk animatie.

Om wat meer uitleg te geven over hoe het animeren verloopt wordt één van de armen van de (kleine) predator als voorbeeld genomen. Bij het inladen van het lichaam en de arm wordt de arm op de goede plek ingeladen, maar het middelpunt van de arm bevindt zich op hetzelfde punt als het middelpunt van het lichaam. Deze middelpunt( van beide) bevindt zich ter hoogte van de vloer direct onder het lichaam. Om de rototatie van de arm goed te laten verlopen wordt de arm eerst omlaag getransleerd vervolgens geroteerd en daarna weer terug getransleerd.

De andere ledematen en de vleugels worden op soort gelijke manier geanimeerd.

*Hitbox*

De hitboxes van alle drie de vijanden zijn heel simpel. Het zijn namelijk cilinder vormige hitboxes. In het xz-vlak is het een cirkel en in het y-vlak heeft een een minimum en maximum y-waarde. De straal van de cirkel is voor zowel Bathos als de kleine predator 1,0. Voor Bathos zal een kleinere straal er beter uitzien, maar omdat de snelheid van de kogels waarmee je kan schieten 2,0(dus elke update 2,0 aflegt) is ervoor gekozen om ook Bathos een straal van 1,0 (dus diameter van 2,0) te geven. De grotere predator heeft een iets grotere straal, namelijk 1,2.

De maximum y-waarde komt bij alle drie tot net boven hun hoofd. De minimum y-waarde bij de beide predators is op gelijke hoogte met de vloer. Bij Bathos, een vliegende vijand, komt het tot zijn benen.

*Healthbar*

De healthbar van de vijanden bestaat uit twee quads van verschillend kleur, waarbij de ene quad één pixel voor de andere quad staat. De afmetingen van de achterste quad zijn constant, maar de afmetingen van de voorste quad zijn afhankelijk van de health van de vijand. Deze afmeting wordt, aan de hand van een simpel bereking, aangepast naarmate de vijand minder health heeft.

Verder is de healthbar altijd naar de player toe gericht. Doormiddel van de vector die van de vijand naar de player wijst is bepaalt welke hoe de healthbar moet hebben in wereldcoördinaten zodat het altijd naar de player toe is gericht.

# GameObject: waepons

De player heeft, over het hele spel, beschikking tot twee typen wapens: een zwaard of een vuurwapen. (Aan het begin heeft hij alleen beschikking tot een zwaard, maar doormiddel van pickups kan hij beter zwaarden en een vuurwapen krijgen). Elke type wapen heeft een eigen klasse die GameObject extend (Sword.java en RangedWeapon.java). Wanneer de player een zwaard kiest heeft hij ook een schild vast. Deze extend ook GameObject een heeft ook een eigen klasse(shield.java).

*Zwaard*

Het zwaard wordt op hetzelfde punt geladen als de positie van de player. Verder wordt zijn positie en manier van display ten opzichte van de camera berekend door middel van bolcoördinaten met de player als middelpunt.

Het zwaard model(ook geladen vanuit een .obj file) bestaat uit een arm en een zwaard die vast wordt gehouden door de arm. Het zwaard model en arm model zijn beide gedownload en later in het programma Blender samengevoegd tot één model.

Om met het zwaard te slaan moet de speler op rechtermuisknop klikken. Wanneer dat is gedaan vindt er een slaan animatie plaats. Dit is een simpel combinatie van tranlaties en rotaties per frame. Wanneer het zwaard de laaste frame van de slaan animatie bereikt wordt gechecked of er een vijand is geraakt.

Het checken of een vijand geraakt wordt, wordt gedaan door de methode damage, die in Enemy.java zit, aan te roepen. Deze methode wordt voor elke vijand aangeroepen. Bij de aanroep worden de hipoint en de damage van het zwaard als argument meegegeven. Per vijand bekijkt de methode dan of de hitpoint overeen komt met de hitbox van de desbetreffende vijand. Als dat zo is wordt voor die vijand de health vermindert. Verder geeft de methode damage ook een boolean terug, maar deze heeft alleen functionaliteit voor vuurwapens.

Zoals hierboven vermeld kan de speler, doormiddel van pickups, een beter zwaard krijgen. Wanneer dat gebeurt wordt het model vervangen door een nieuw model. (Deze nieuwe model is opdezelfde manier gemaakt als het eerste zwaard). Verder wordt de damage van het zwaard verhoogd. Al het andere blijft opdezelfde manier werken.

*Vuurwapen*

Na het oppakken van een bepaald soort pickup krijgt de speler ook de beschikking tot een vuurwapen. De positie en display wordt zoals het zwaard gedaan aan de hand van bolcoördinaten met de player als middelpunt.

Het model bestaat uit een futuristisch raygun. Bij de raygun is er geen arm toegevoegd omdat de handgreep buiten beeld valt.

Om te schieten moet de speler rechtermuisknop klikken. Op dat moment wordt er een bullet object aangemaakt (bullet.java extend GameObject). Deze bullet wordt dan toegevoegd aan de arraylist van bullets gemaakt in MainClass.java. In de buller klasse wordt dan per kogel de positie en display bijgehouden. Verder wordt er per update gechecked of de bullet collision heeft met een vijand of een muur. Het checken van collision met vijand wordt gedaan doormiddel van de hierboven besproken damage methode: per update wordt de damage methode aangeroepen met als argumenten de positie van de kogel. Als deze methode schade toe brengt aan de vijand en dus ‘true’ teruggeeft, wordt de kogel uit de arraylist verwijdert. Collision met muur gaat op soortgelijke manier.

*schild*

Wanneer de player een zwaard heeft, heeft hij ook beschikking tot een schild. De positie en display hiervan wordt hetzelfde afgehandelt als zwaard en vuurwapen. Het model hiervan bestaat uit een arm die een schild vasthoudt. (de schild en arm waren eerst aparte modellen, maar zijn samengevoegd met behulp van Blender). Om de schild te gebruiken dient de speler zijn rechtermuisknop te klikken/vasthouden. Op dat moment wordt de schild wat directer voor de speler. Animatie hiervan is ook gedaan door translaties en rotaties.

Wanneer de schild wordt gebruikt, kan er geen schade aan de player worden toegebracht, maar de speler kan, zolang hij rechtermuisknop inhoudt, niet slaan met zijn zwaard.

# GameState

In het spel zitten acht verschillende gamestates.(Negen, als de STOP\_STATE wordt meegeteld. Maar het enige wat deze doet is een system.exit() gebruiken). De verschillende states zijn in een enumerated gezet. De GameStateManager houdt bij in welke state het spel zit. Doormiddel van de methode GameStateUpdate wordt van gamestate gewisseld en door gebruik te maken van een getState() kan de huidige state worden gevraagd.

Een probleem dat aan het begin plaats vond was dat bij het switchen tussen gamestates de init(GLAutoDrawable arg0)-methode van de nieuwe state niet werd aangeroepen. Dit is verholpen door handmatig de init-methode aan te roepen. Omdat deze methode een GLAutoDrawable nodig had, moest het aanroepen in de display/render-methode worden gedaan. Om ervoor te zorgen dat de init-methode maar één keer werd aangeroepen als je de state net binnen gaat is er in de GameStateManager, per state, een boolean toegevoegd. Voor elke zo een boolean is er ook een get- en set-methode toegevoegd. Aan de hand van de PAUSE\_STATE en MAINGAME\_STATE zal er uitgelegd worden wat er precies wordt gedaan.

Stel de speler zit in het pauze menu. Dan is de huidige state dePAUSE\_STATE. Verder staat de hierboven genoemde boolean van de PAUSE\_STATE (genaamd sPause) op ‘true’, aangezien de init van het pauze menu al is aangeroepen. (De boolean van MAINGAME\_STATE, sMainGame, staat nu op false wat de init-methode van maingame is niet uitgevoerd). Wanneer de speler nu op ‘resume’klikt, gaat hij naar de MAINGAME\_STATE. Op het moment van klikken wordt ten eerste aan de GameStateMANAGER doorgegeven dat de huidige state de MAINGAME\_STATE is. Verder wordt sPause op ‘false’ gezet (zodat als de speler weer naar het pauze menu gaat de init wel kan worden uitgevoerd). Omdat sMainGame op ‘false’staat wordt de init-methode van maingame uitgevoerd. Aan het eind van init wordt sMainGame op ‘true’gezet zodat het niet weer wordt uitgevoerd.

Een ander probleem dat plaats vond heeft te maken met in wat voor omgeving elke state zat(2D of 3D). Van alle verschillende states is de MAINGAME\_STATE de enige state in een 3D omgeving. Het spel begint in een 2D omgeving, namelijk de LOGIN\_STATE. Het switchen tussen 2D omgevingen of switchen van een 2D omgeving naar 3D omgeving ging altijd goed. Maar in eerste instantie gaf het switchen van een 3D omgeving naar een 2D omgeving wat problemen. Dat is later verholpen door aan het eind van de init-methode van elke 2D state gl.glDisable(GL.*GL\_DEPTH\_TEST*) en gl.glDisable(GL.*GL\_LIGHTING*) toe te voegen.

Een overzicht van alle states en hoe de speler tussen elke state kunt u vinden op het op gamestate diagram op de volgende pagina.

