

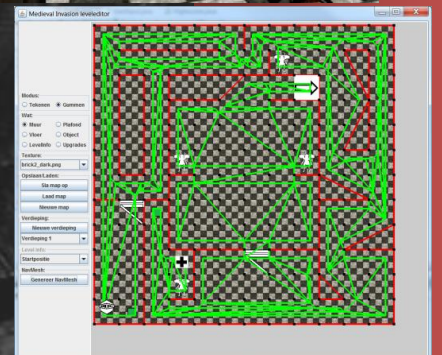
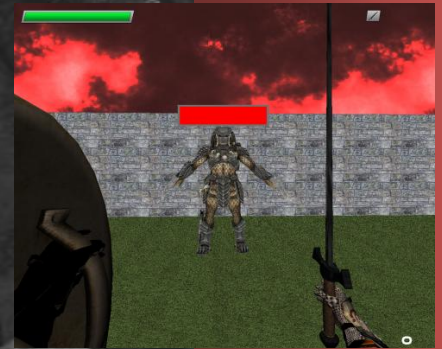
# 2014

TU Delft

MINORPROJECT  
SOFTWARE ONTWERPEN  
EN TOEPASSEN

Hassan Al Mahmoedi  
Ruben Koeze  
Guido Remmerswaal  
Johnny Wang

GROUP 10



# [MEDIEVAL INVASION]

Student assistenten: Bert Bosch, Julian Faber en Tim Rensen

## 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 Bosch.

# CONTENTS

Voorwoord.....	2
Contents .....	3
Introductie .....	5
1 Game Design .....	6
2 UML documentatie .....	7
2.1 GameStates.....	7
2.2 Use cases .....	9
Het spel starten .....	9
Het spel uitspelen .....	10
Een deur openen .....	10
Vijanden vermoorden .....	11
2.3 gameClass diagram .....	12
2.4 Applicatie Architectuur diagram.....	13
3 Beschrijving van algoritmes .....	14
3.1 Leveleditor .....	15
Modus.....	15
Wat .....	15
Texture.....	15
Opslaan/laden .....	16
Verdieping .....	16
Navmesh .....	16
3.2 Maze Opbouw.....	17
3.3 3D Model-Loader .....	17
3.4 Collision detection .....	18
Detectie op muren .....	18

Detectie op vloer .....	19
Detectie op objects .....	19
3.5 GameObject: Enemy .....	20
Hitbox .....	20
Healthbar .....	21
3.6 Enemy: PathFinding .....	21
feromonen .....	21
Navigation Mesh .....	22
3.7 Enemy: intelligence .....	25
3.8 GameObject: weapons .....	26
Zwaard .....	26
Vuurwapen .....	27
schild .....	27
4 Testing .....	28
5 Conclusie .....	30
6 Evaluaties .....	31
6.1 Evaluatie HassaN al Mahmoedi .....	31
6.2 Evaluatie Ruben Koeze .....	31
6.3 Evaluatie Guido Remmerswaal .....	32
6.4 Evaluatie Johnny Wang .....	33
7 Literatuur .....	34

## 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.



# 1 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-intelligence 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 enemy. 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 is de speler opgesloten in een doolhof op de toren van het kasteel en moet de uitgang van dit doolhof gevonden worden.

## 2 UML DOCUMENTATIE

In dit hoofdstuk wordt de UML documentatie behandeld.

### 2.1 GAMESTATES

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 de PAUSE\_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.

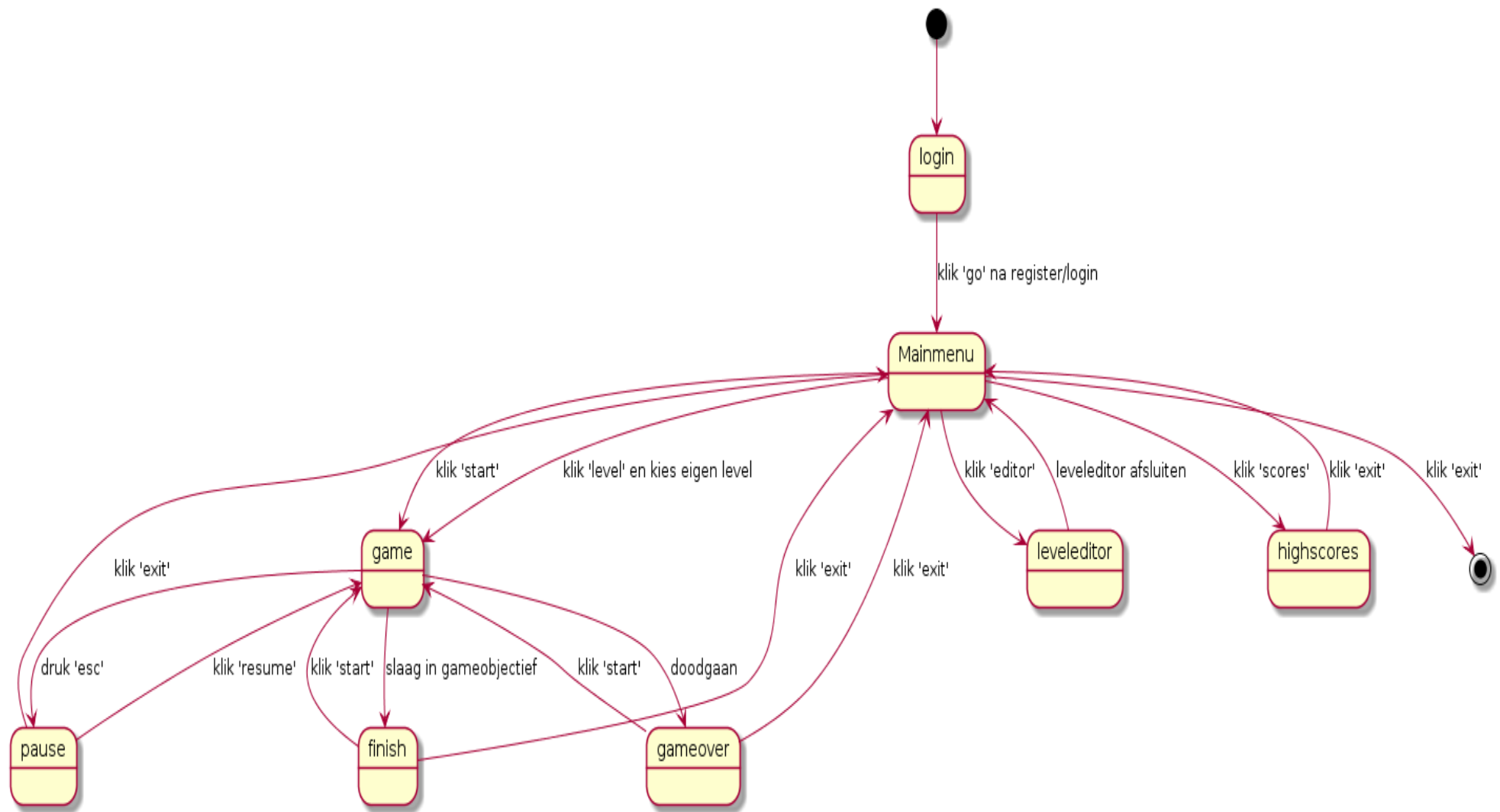


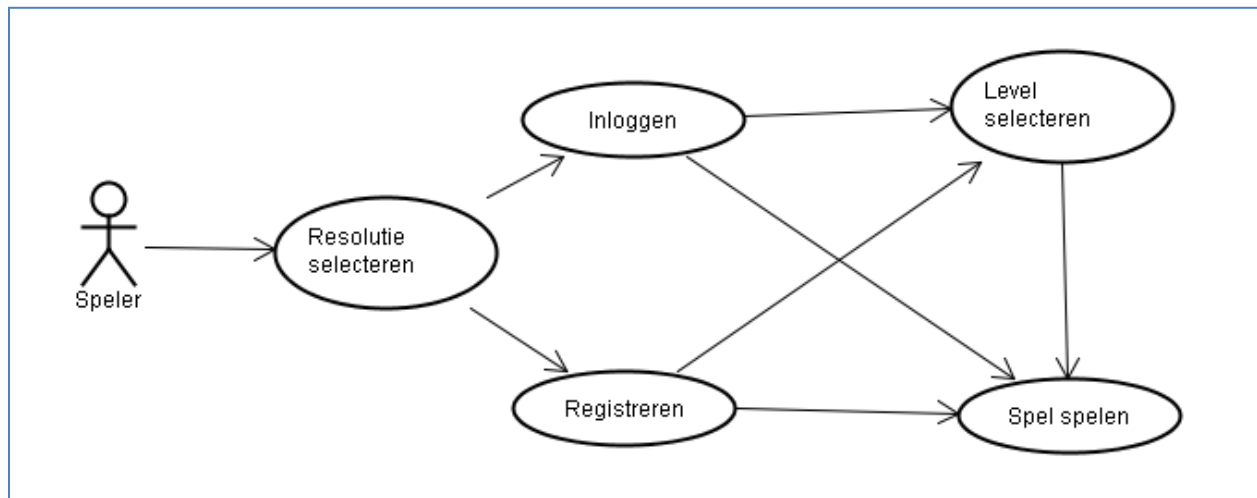
Fig.2.1 States Diagram



## 2.2 USE CASES

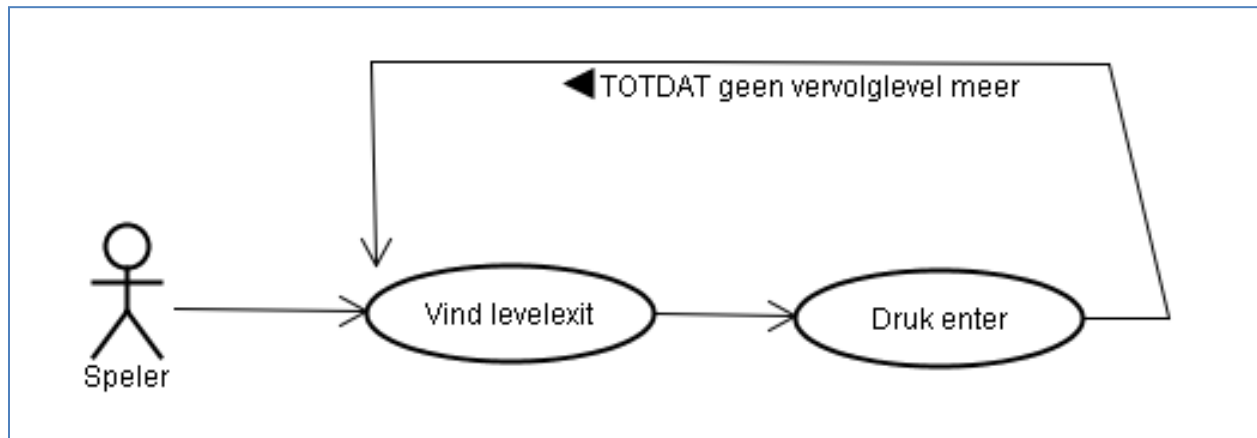
Hieronder de use cases van enkele game onderdelen.

### HET SPEL STARTEN



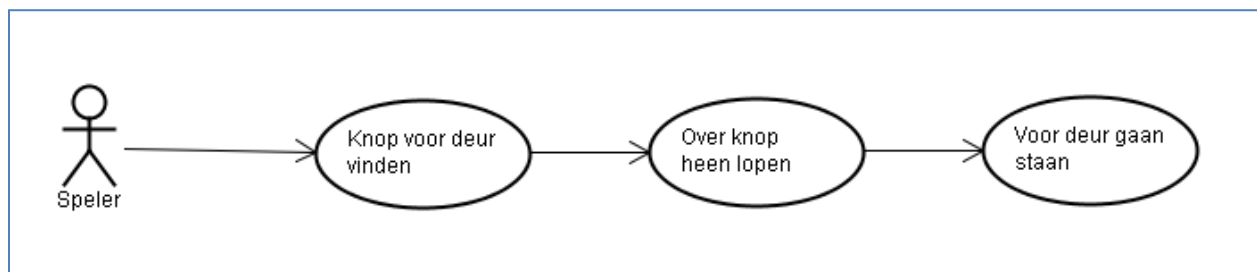
Use case	Het spel starten
<b>Beschrijving</b>	Het spel spelen vanaf het opstarten
<b>Actoren</b>	Speler (primair)
<b>Aannames</b>	
<b>Stappen</b>	1. Resolutie selecteren #2. Inloggen <b>of</b> registreren 3. if (ander level gewenst) THEN 3.1 Level selecteren 4. Spel starten

## HET SPEL UITSPELEN



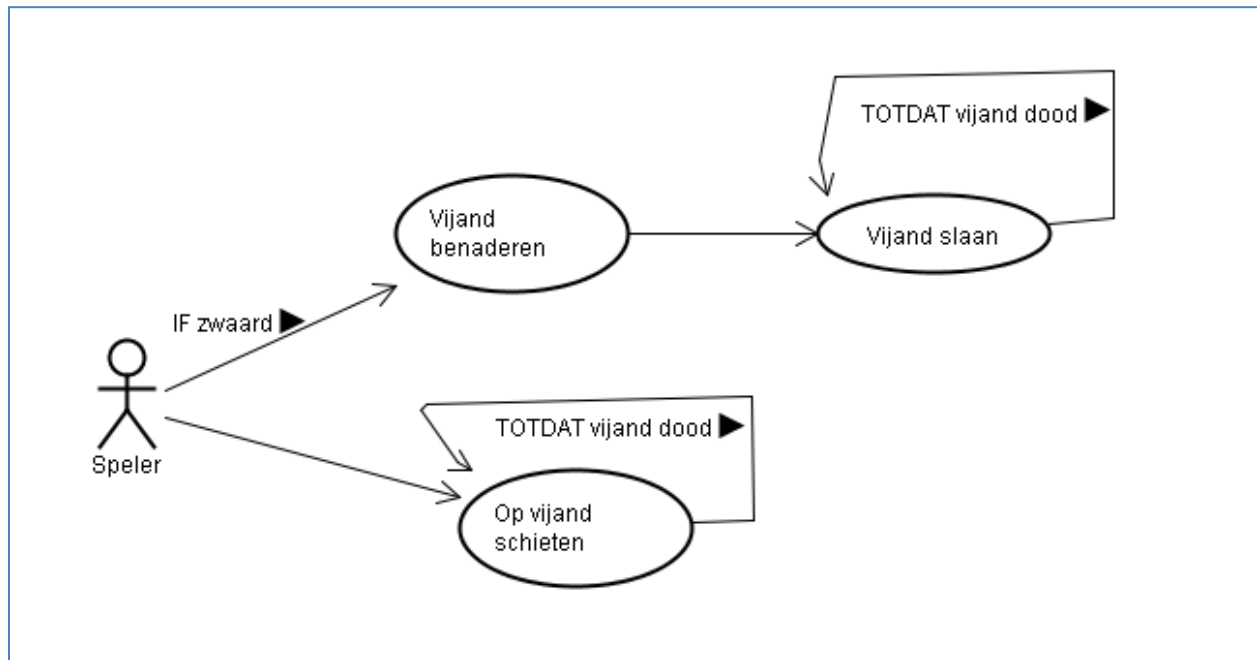
Use case	Het spel uitspelen
<b>Beschrijving</b>	Het spel uitspelen vanaf begin level
<b>Actoren</b>	Speler (primair)
<b>Aannames</b>	
<b>Stappen</b>	1. REPEAT 1.1 Vind levelexit 1.2 Druk op enter UNTIL geen vervolglevel

## EEN DEUR OPENEN



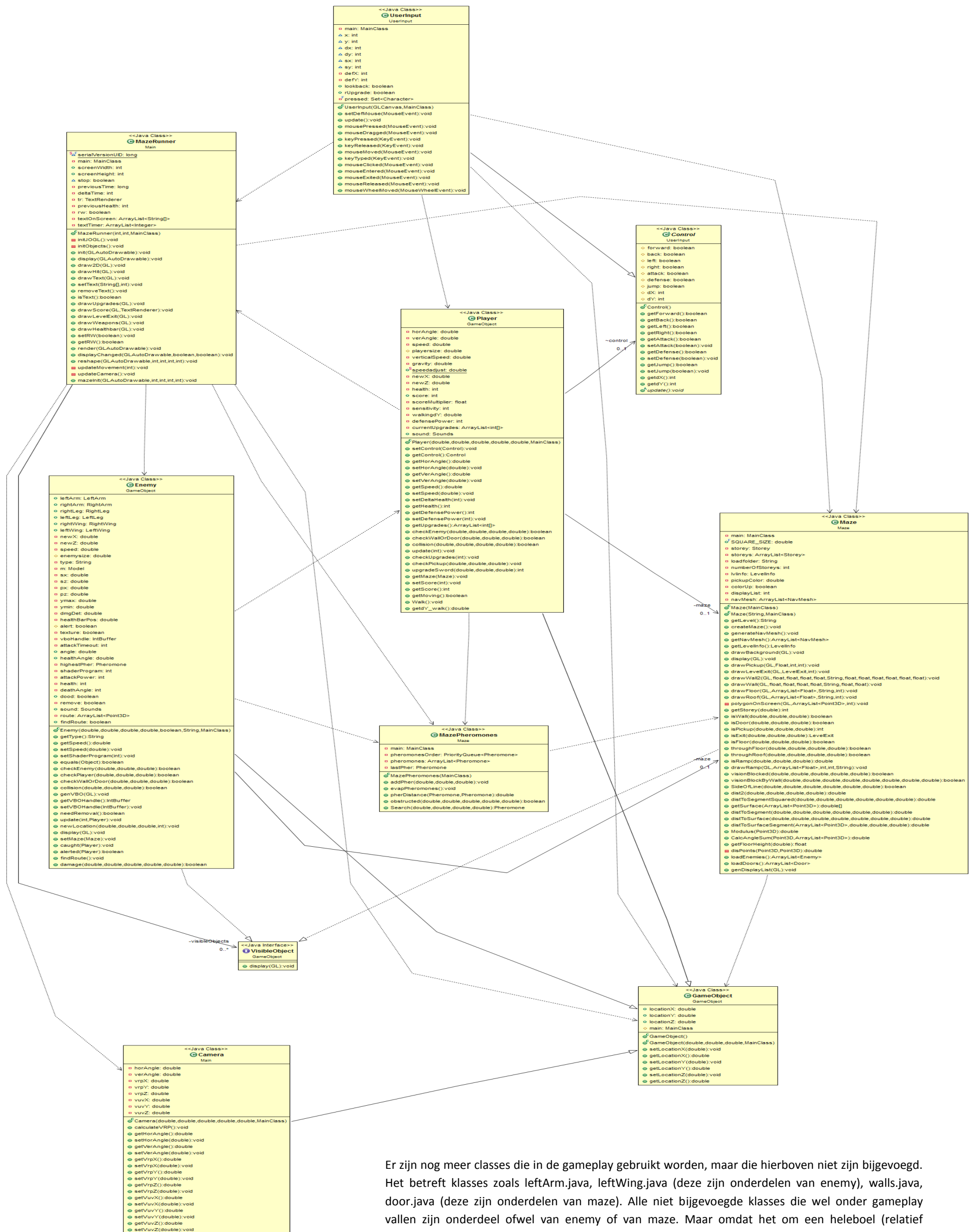
Use case	Een deur openen
<b>Beschrijving</b>	Het openen van een deur in het spel
<b>Actoren</b>	Speler (primair)
<b>Aannames</b>	
<b>Stappen</b>	1. Knop voor deur vinden 2. Over knop heen lopen 3. Voor deur gaan staan

## VIJANDEN VERMOORDEN



Use case	Vijand vermoorden	
<b>Beschrijving</b>	Het vermoorden van een vijand in het spel	
<b>Actoren</b>	Speler Vijand	(primair)
<b>Aannames</b>		
<b>Stappen</b>	1. IF zwaard THEN 1.1 Vijand benaderen REPEAT 1.1.1 Vijand slaan UNTIL vijand dood ELSE REPEAT 1.2 Op vijand schieten UNTIL vijand dood	

## 2.3 GAMECLASS DIAGRAM







## 2.4 APPLICATIE ARCHITECTUUR DIAGRAM

### 3 BESCHRIJVING VAN ALGORITMES

De code van het spel bevat natuurlijk enkele methoden en algoritmes die wellicht dienen te worden toegelicht. In dit hoofdstuk worden enkele belangrijke algoritmes kort beschreven om te verduidelijken hoe het spel daadwerkelijk werkt.

In het vervolg zullen de volgende plaatjes gebruikt worden om meer uitleg te geven:

	<b>Speler</b> (Ridder)
	<b>Vijand</b> (Predator Youngblood)
	<b>Object met <i>size</i></b> (Collision area – H3.3)
	<b>Uit te voeren verplaatsing</b>



### 3.1 LEVEEDITOR

Voor het eenvoudig ontwerpen van levels is een leveeditor gemaakt. Deze leveeditor kan worden gestart vanuit het startmenu. Eenmaal in de editor kan een volledig nieuw level gecreëerd worden of kan een bestaand level aangepast worden. De leveeditor bestaat uit een menu met opties aan de linkerkant, en een bovenaanzicht van het level aan de rechterkant. Door opties aan de linkerkant te selecteren, kunnen de verschillende aspecten van het level aan de rechterkant gemanipuleerd worden. Hieronder zal kort worden besproken wat de verschillende opties in het menu doen.

#### MODUS

In dit menu kan gekozen worden voor de teken of gum modus. Wanneer de tekenmodus geselecteerd is zal elk punt wat aangeklikt wordt, opgeslagen worden. Hierover zullen vervolgens dingen getekend worden. Wanneer de gum-modus aangevinkt is kunnen aspecten uit het level verwijderd worden. Als deze modus aangevinkt staat zal het gene wat verwijderd wordt wanneer er geklikt wordt groen worden. Dit kan door de muis over het level te bewegen.

#### WAT

Hierin kan geselecteerd wat er moet worden getekend of gegumd, afhankelijk van wat de geselecteerde modus is. Muur, plafond en vloer spreken voor zichzelf. Wanneer object geselecteerd is kan uit een lijst onderaan het menu geselecteerd worden wat voor object getekend moet worden. Hier kunnen de verschillende vijanden gekozen worden, maar ook de ramp, deur en levelexit kunnen hiermee getekend worden. Met de levelinfo optie kan de beginpositie van de speler geselecteerd worden. Ten slotte kan met de upgrades knop een scala aan pick-ups getekend worden. Welke getekend wordt is wederom afhankelijk van de optie geselecteerd in het menu onderaan alle opties.

#### TEXTURE

In dit menu zijn alle textures zichtbaar. Deze hebben betrekking op bijna alles wat getekend wordt. Wanneer een muur getekend wordt, zal deze de texture krijgen die in dit menu geselecteerd staat.

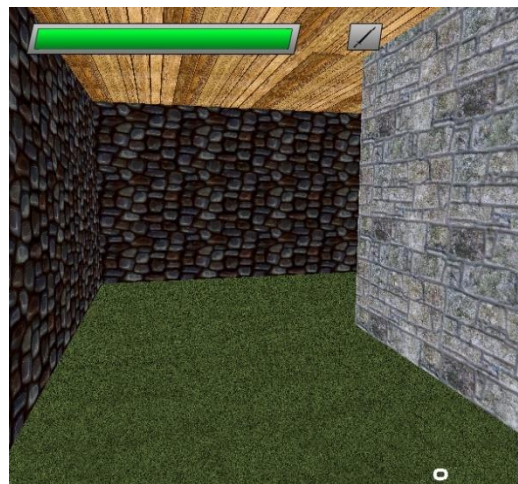


Fig. 3.1 Verschil in textures in het spel





### 3.2 MAZE OPBOUW

De maze is opgebouwd uit muren, deuren, vloeren, plafonds, ramps en pickups. Om een maze te bouwen moet de speler een level maken in de leveleditor. In de leveleditor kan de speler begin en eindpunten aangeven van elk onderdeel dat hij in de maze wil. Deze informatie(begin- en eindpunt, soort object) wordt dan in een .txt file opgeslagen. Bij het inladen van de gemaakte level wordt deze informatie gelezen. De objecten worden per soort in een arraylist gezet (dus eenlist met muren, een list met vloeren enz). Al deze objecten worden als polygonen getekend. Aangezien de in de file waarin de objecten worden opgeslagen alleen begin- en eindpunten staan vind er eerst een berekening plaats. Aan de hand van de muur zal er worden uitgelegd wat er gebeurt (overige objecten worden op soortgelijke manier getekend).

Wanneer een muurobject in de Muurlijst zit moet hij dus worden getekend. Er is een methode(drawmuur2) die aan de hand van een begin- en eindpunt een muur tekent van één pixel breedte(hoogte staat vast, dus in combinatie daarmee tekent hij een polygon met 4 hoekpunten). Verder is er een methode, genaamd drawmuur, die aan de hand van het begin- en eindpunt van de muur bepaalt in welke richting de muur dikker moet worden gemaakt en daarmee de acht verschillende hoekpunten van de muur bepaalt. Vervolgens wordt in de drawmuur-methode zes keer drawmuur2 aangeroepen met combinaties van deze acht punten. Op deze manier worden de 6 zijdes van een muur getekend (de binnenkant is dus leeg).

Zoals vermeld worden vloeren, plafonds en ramps op dezelfde manier getekend. Een pickup tekenen is nog simpeler. Om een pickup te tekenen wordt gewoon een kubus getekend op de aangegeven positie.

### 3.3 3D MODEL-LOADER

Voor het laden van 3D modellen zijn de tutorials van Oskar Verhoek[6] geraadpleegd. Deze tutorial leverde de klasse obj-loader. Deze klasse kon 3D modellen in obj-formaat inladen in JOGL. Hierbij waren echter nog geen textures toegevoegd. De modellen werden door deze klasse ingeladen tot een displaylist, een vooraf gegenereerde lijst van JOGL operaties. Hierdoor wordt het model eenmaal ingeladen aan het begin van het programma, waardoor het spel minder traag wordt.

Het toevoegen van textures aan een displaylist is echter lastig. Het inladen van de modellen is daarom overgezet van een displaylist naar een vertexbufferobject. Bij deze manier van tekenen worden de coördinaten van de polygons van het model aan het begin van de game ingeladen. Bij ieder frame worden op deze coördinaten de polygons getekend. Deze manier van tekenen is langzamer dan het gebruik van een displaylist, maar heeft tevens de mogelijkheid om textures op een 3D object te tekenen. Voor het tekenen van textures op een 3D object worden naast de coördinaten van de polygons, ook de texture-coördinaten in een buffer opgeslagen. JOGL heeft nu de mogelijkheid om de polygons met textures te tekenen.

Wegens het gebrek aan tijd is er gebruik gemaakt van modellen gevonden op TF3DM [7]. Deze modellen zijn doormiddel van Blender[8] bewerkt om aan de eisen van deze game te voldoen.

### 3.4 COLLISION DETECTION

Zowel de speler als de vijanden in het spel bezitten een collision detection methode. Dit is een detectie die tijdens het lopen rekening houdt andere objecten, zodat voorkomen wordt dat twee objecten door elkaar staan. De methode controleert, voordat er gelopen wordt, of het object op de volgende locatie kan staan. Bijvoorbeeld: indien de speler in een muur wil gaan staan, wordt deze actie herkent als niet mogelijk, wordt de loopactie niet uitgevoerd en blijft de speler dus op de huidige positie staan.

Aan de speler en vijanden is een parameter *size* toegekend. Deze staat voor de straal om het object waar andere objecten zich niet in kunnen bevinden. De collision detection neemt deze parameter als basis voor het controleren op andere objecten. De detectie is opgebouwd uit drie aparte controles, die elk net iets anders werkt.

#### DETECTIE OP MUREN

Op de kaart zijn de muren eigenlijk lijnen met een aangegeven dikte. Wanneer het object wil lopen wordt er gekeken of de nieuwe positie niet te dicht op de muren zit. Indien dit wel het geval is wordt de loopactie niet uitgevoerd.

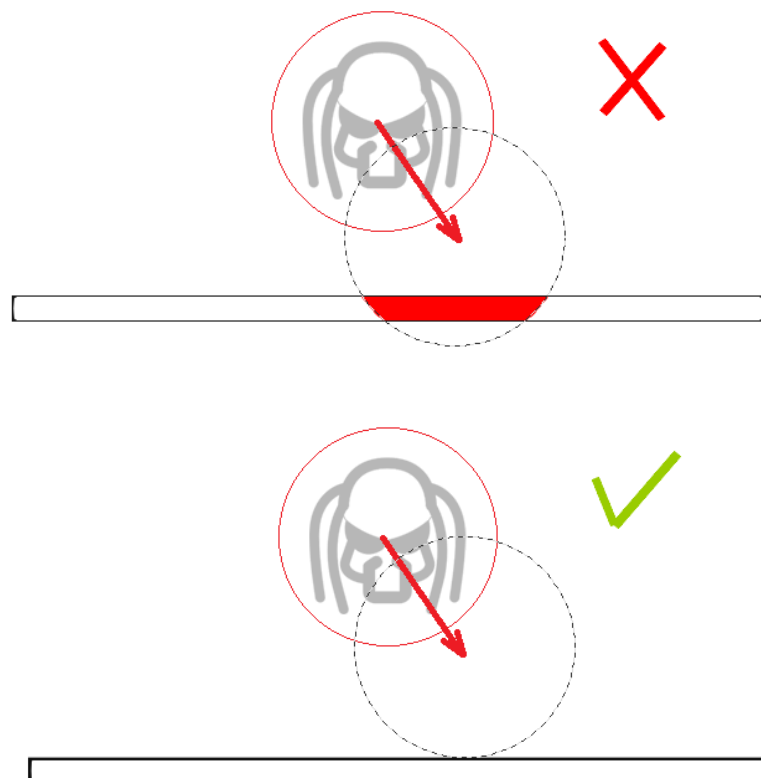


Fig. 3.3 Visualisatie van muurdetectie

## DETECTIE OP VLOER

De speler ondervindt ten alle tijden zwaartekracht. Een versnelling van de verplaatsing in de negatieve Y-richting (hij wordt naar beneden getrokken). Indien de speler door de gravitatie een positie toegekend krijgt waarbij hij door de vloer zakt, dan wordt hij in plaats daarvan op normale hoogte gezet.

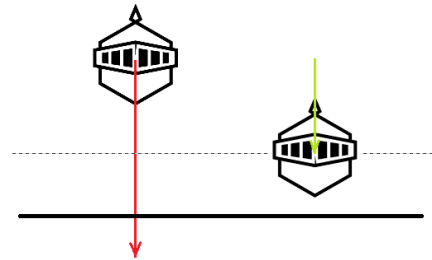


Fig. 3.4 Visualisatie van vloer detectie en correctie van locatie.

De vijanden hebben echter geen vloerdetectie. Dit omdat er per verdieping vijanden gedefinieerd worden en het niet de bedoeling is dat de vijanden je achterna gaan naar een andere verdieping.

## DETECTIE OP OBJECTS

De speler en vijanden kunnen elkaar detecteren. Dit geldt voor detectie tussen speler en vijand, zowel als vijanden onderling. De detectie hier werkt echter net iets anders dan de eerder beschreven muurdetectie.

Voor de detectie van de objecten wordt de onderlinge afstand tussen de twee objecten bepaald. Deze afstand wordt dan vergeleken met de sizes van de beide betreffende objecten. Blijkt de afstand kleiner te zijn dan de som van de objectsizes, dan betekent dit dat de twee objecten met elkaar botsen en is de actie niet mogelijk.

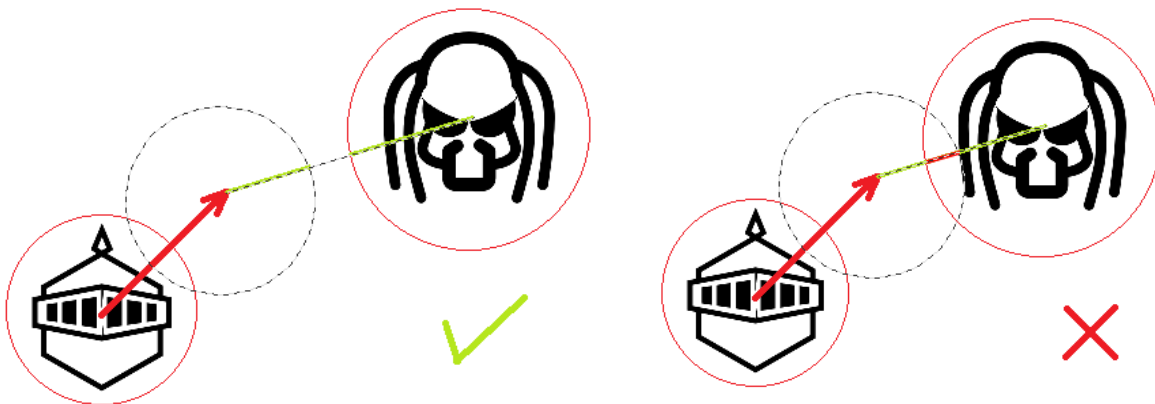


Fig. 3.5 Visualisatie van object detectie

Voordat een beweging van een object wordt uitgevoerd wordt er gecontroleerd op deze drie detecties. Indien aan alle drie voldaan wordt wordt de beweging uitgevoerd. Op deze manier kunnen de objecten zich verplaatsen door de maze, zonder door andere objecten, muren of vloeren te lopen.

### 3.5 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 beaapt 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 afmetingen en vervolgens opgesplitst 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 opgesplitst 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 rotatie 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 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.



Fig. 3.6 Visualisatie van cilindervormige hitbox

## 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 berekening, 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 wereld-coördinaten zodat het altijd naar de player toe is gericht.



Fig. 3.7 De healthbar zit altijd boven de vijand

## 3.6 ENEMY: PATHFINDING

Voor enige intelligentie van de vijanden is het vinden van routes noodzakelijk. Het vinden van routes tussen de speler en een vijand wordt op twee verschillende manieren geregeld. Voor lange afstanden is een navigatie mesh gebruikt, een lijst van driehoeken waarop de vijanden een route kan zoeken. Voor korte afstanden is gebruik gemaakt van feromonen.

## FEROMONEN

Voor het vinden van een route op korte afstanden wordt gebruik gemaakt van feromonen. De speler laat gedurende het gehele spel feromonen achter. Dit zijn geuren die over tijd vervagen en uiteindelijk verdwijnen. De vijanden kunnen deze geuren opvangen en hierdoor achter de speler aanlopen. Op korte afstanden kunnen hiermee vloeiende routes naar de speler gevonden worden. De vijand loopt achter de speler aan en zal zijn route vrijwel kopiëren.

Het kiezen van feromonen wordt op de volgende manier afgehandeld. In het eerste deel van **Fout! erwijzingsbron niet gevonden.** zijn drie feromonen en een vijand te zien. De feromonen wordt op volgorde afgewerkt, waarbij begonnen wordt met feromonen met de sterkste geur. De lijst wordt dus afgewerkt van meest recent naar minst recent neergelegd. In het voorbeeld wordt feromoon 1 als eerste afgemerkt. Aangezien dit feromoon niet zichtbaar is voor de vijand, een muur blokkeert het zicht, gaat het algoritme verder met feromoon 2. Voor deze feromoon geldt het zelfde als voor feromoon 1. Feromoon 3 is het eerste zichtbare feromoon, waarna het algoritme stopt. De vijand beweegt zicht voort naar feromoon 3, waarna het algoritme herhaald wordt.

## NAVIGATION MESH

22

.....

**Fout! Verwijzingsbron niet gevonden..10.**



.....

de vijand zich kan bevinden. Het resultaat is te zien in Figuur 3.11.



.....

resultaat is te zien in Figuur 3.12.



## VINDEN VAN BUREN (POLY2TRI)

De driehoeken van de navmesh moeten gekoppeld worden als buren. Hiervoor worden de aanliggende driehoeken toegevoegd aan de buren van iedere driehoek. Dit kan gedaan worden door alle driehoeken met overeenkomende zijden aan elkaar te koppelen. In het voorbeeld op Figuur 3.12 moet driehoek 1 gekoppeld worden aan 2 en 3.

De gevonden driehoeken in de klasse navmeshgeneration worden nu door gegeven aan de klasse navmesh. In deze klasse wordt het vinden van routes verzorgt.

## VINDEN VAN EEN DRIEHOEK ROUTE MET DE NAVMESH

Voor het vinden van de route is opgedeeld in twee delen. Eerst wordt er doormiddel van een A\* pathfinding algoritme over de driehoeken een route gezocht. Dit resulteert in een route bestaand uit driehoeken. Ook hier is een simpele tutorial gevolgd [4]. Deze tutorial maakt echter gebruik van vierkant polygons in plaats van driehoeken. De implementatie van het algoritme is hierdoor zelf geschreven aan de hand van de tutorial.

Het A\* pathfinding algoritme gaat alle driehoeken af tot er een route is gevonden. In het voorbeeld op **Fout! Verwijzingsbron niet gevonden.** wordt een route gezocht tussen driehoek 1 en 2 (het begin en eind punt van de route liggen in deze driehoeken). Het zoeken wordt begonnen bij driehoek 1. Aangezien driehoek 3 de enige buurman van driehoek 1 is zal het algoritme meteen verder gaan met driehoek 3. Driehoek 1 wordt als ouder van driehoek 3 opgeslagen.

Bij driehoek 3 moet er echter een keuze gemaakt worden tussen twee driehoeken. Er wordt gekozen voor de driehoek waarbij de te verwachte afstand het kleinst is. De verwachte afstand  $F$  is in dit geval gedefinieerd als  $F = G + H$ . Hierin is  $G$  de afgelegde afstand om de driehoek te bereiken en  $H$  de afstand hemelsbreed tussen het midden van de driehoek en het einde van de route. In het voorbeeld is dit waarschijnlijk driehoek 4. Driehoek 3 wordt als ouder van driehoek 4 opgeslagen.

Het zoeken van driehoeken met de beste verwachte afstand wordt nu herhaalt totdat het eindpunt bereikt wordt. Waarschijnlijk wordt bij de volgende iteratie driehoek 5 gevonden (een driehoek op de juist route) en driehoek 3 wordt als ouder van driehoek 5 opgeslagen. Bij het zoeken van driehoek 2 worden waarschijnlijk de rode en blauwe driehoek gebruikt. De blauwe driehoeken zijn uiteindelijk de kortste route. Door vanaf de laatste driehoek (driehoek 2) alle ouders op te zoeken kan de route terug gevonden worden.

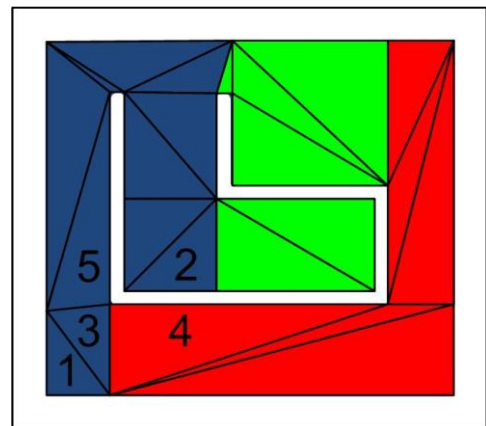


Fig. 3.13 Driehoek route



A blue square containing a white L-shaped path. A green line starts at the bottom left, goes up to the top left, then diagonally down to the center, and finally horizontally to the right, ending at the center of the L-shape.

### Fig. 3.14 Uiteindelijke route

### 3.7 ENEMY: INTELLIGENCE

Voor de vijanden van dit spel is een kleine hoeveelheid intelligentie toegevoegd. Het doel van de vijand is het doden van de speler. Hiervoor heeft de vijand de mogelijkheid om naar de speler toe te lopen. De vijand loopt op de speler af doormiddel van de feromonen als de speler eenmaal in de visie van de vijand is geweest. De visie is een maximale afstand van de vijand waarbinnen hij de speler of andere vijanden waarneemt. De vijand zal de speler nu blijven volgen, totdat de vijand sterft.

De vijand heeft tevens de mogelijkheid om te communiceren met andere vijanden (swarm intelligence). Als de speler in de visie komt van de vijand, roept deze naar omliggende vijanden. Er kan alleen gecommuniceerd worden met vijanden binnen zijn visie. De vijand geeft de locatie van de speler door aan de andere vijanden en zij zullen via de navmesh een route zoeken naar de speler. Een vijand heeft de mogelijkheid om met andere vijanden te communiceren door een muur, zolang het maar in zijn visie blijft.

Een vijand, die de locatie van de speler heeft verkregen van een andere vijand, zal via de navmesh zoeken naar de speler. Als de speler binnen de visie van de vijand komt zal de vijand overschakelen op bewegen via feromonen en eventueel andere vijanden informeren.

### 3.8 GAMEOBJECT: WEAPONS

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 hitpoint 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.

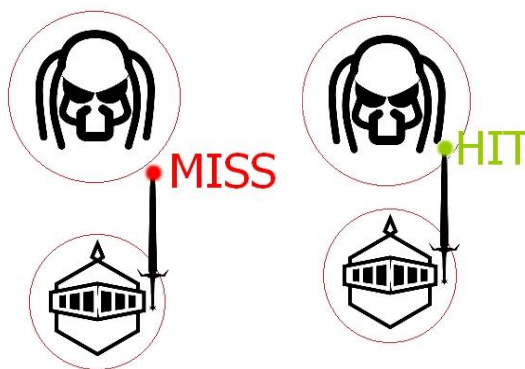


Fig. 3.15 Visualisatie van vijand aanvallen

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.



Fig. 3.16 Zwaard upgrades

## 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 bullet 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 verwijderd. Collision met muur gaat op soortgelijke manier.



Fig. 3.17 Raygun gericht op een vijandig doelwit

## 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.



Fig. 3.18 Gebruik schilden om de predator tegen te houden



## 4 TESTING

Er zijn twee unit test geschreven voor dit project. De testen zijn uitgevoerd op Maze en Storey, de twee klassen die samen een level opbouwen. Het testen van deze klassen is gebeurt aan de hand van een testlevel: JUnitTestLevel. Het is voor het testen noodzakelijk om over dit level te beschikken. Het coverage report van deze klasse is te zien in Figuur 1.

De eerste unit test is geschreven voor de klasse Maze. Aan de hand van het coverage report zijn vrijwel alle branches getest. De methodes waarin de maze drawing gebeurt zijn niet getest. Hierdoor is de coverage maar 44%. Als de drawing methodes niet meegeteld worden resulteert dit in een coverage van 85,6%. Door coördinaten in het spel op te zoeken zijn zoveel mogelijk verschillende testen uitgevoerd op bijvoorbeeld de collision.

### Maze

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
drawPickup(GL, Point2D, float, int, int)	<div></div>	0%	<div></div>	0%	9	9	72	72	1	1
drawBackground(GL)	<div></div>	0%	<div></div>	n/a	1	1	59	59	1	1
drawWall(GL, float, float, float, float, String, float, float)	<div></div>	0%	<div></div>	0%	6	6	30	30	1	1
drawRamp(GL, ArrayList, int, int, String)	<div></div>	0%	<div></div>	0%	4	4	43	43	1	1
drawFloor(GL, ArrayList, String, int)	<div></div>	0%	<div></div>	0%	3	3	39	39	1	1
genDisplayList(GL)	<div></div>	0%	<div></div>	0%	8	8	24	24	1	1
polygonOnScreen(GL, ArrayList, int)	<div></div>	0%	<div></div>	n/a	1	1	20	20	1	1
drawWall2(GL, float, float, float, float, String, float, float, float, float, float)	<div></div>	0%	<div></div>	0%	2	2	31	31	1	1
drawLevelExit(GL, LevelExit, int)	<div></div>	0%	<div></div>	n/a	1	1	20	20	1	1
drawRoof(GL, ArrayList, String, int)	<div></div>	0%	<div></div>	0%	2	2	10	10	1	1
display(GL)	<div></div>	0%	<div></div>	0%	4	4	10	10	1	1
isDoor(double, double, double)	<div></div>	0%	<div></div>	0%	5	5	7	7	1	1
createMaze()	<div></div>	91%	<div></div>	83%	1	4	2	16	0	1
visionBlockByWall(double, double, double, double, double, double, double, double)	<div></div>	97%	<div></div>	75%	1	3	1	7	0	1
isRamp(double, double, double)	<div></div>	100%	<div></div>	100%	0	11	0	26	0	1
getSurface(ArrayList)	<div></div>	100%	<div></div>	n/a	0	1	0	8	0	1
CalcAngleSum(Point3D, ArrayList)	<div></div>	100%	<div></div>	100%	0	3	0	19	0	1
loadDoors()	<div></div>	100%	<div></div>	100%	0	4	0	20	0	1
throughRoof(double, double, double, double)	<div></div>	100%	<div></div>	93%	1	8	0	16	0	1
distToSurfaceSegment(ArrayList, double, double, double)	<div></div>	100%	<div></div>	75%	1	3	0	13	0	1
isFloor(double, double, double)	<div></div>	100%	<div></div>	100%	0	8	0	16	0	1
throughFloor(double, double, double, double)	<div></div>	100%	<div></div>	100%	0	8	0	16	0	1
loadEnemies()	<div></div>	100%	<div></div>	100%	0	4	0	12	0	1
isPickup(double, double, double)	<div></div>	100%	<div></div>	100%	0	4	0	10	0	1
isExit(double, double, double)	<div></div>	100%	<div></div>	100%	0	10	0	13	0	1
isWall(double, double, double)	<div></div>	100%	<div></div>	100%	0	6	0	10	0	1
visionBlocked(double, double, double, double, double)	<div></div>	100%	<div></div>	100%	0	6	0	10	0	1
distToSegmentSquared(double, double, double, double, double, double)	<div></div>	100%	<div></div>	100%	0	4	0	6	0	1
getFloorHeight(double)	<div></div>	100%	<div></div>	100%	0	4	0	5	0	1
getStorey(double)	<div></div>	100%	<div></div>	100%	0	4	0	5	0	1
disPoints(Point3D, Point3D)	<div></div>	100%	<div></div>	n/a	0	1	0	3	0	1
distToSurface(double, double, double, double, double, double, double)	<div></div>	100%	<div></div>	n/a	0	1	0	4	0	1
Maze(String, MainClass)	<div></div>	100%	<div></div>	n/a	0	1	0	10	0	1
Maze(MainClass)	<div></div>	100%	<div></div>	n/a	0	1	0	9	0	1
SideOffline(double, double, double, double, double, double, double)	<div></div>	100%	<div></div>	100%	0	2	0	1	0	1
Modulus(Point3D)	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getLevel()	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
dist2(double, double, double, double)	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
distToSegment(double, double, double, double, double, double)	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
generateNavMesh()	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
getNavMesh()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getStoreys()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getLevelInfo()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
Total	2.681 of 4.782	44%	72 of 224	68%	50	155	368	625	12	43

Figuur 1: Maze Coverage Report

De klasse Storey is voornamelijk belangrijk voor het inlezen en wegschrijven van Levels. Eerst wordt hier getest of de Read functie op de juiste manier werkt. Hiervoor wordt het JUnitTestLevel in geladen. Alle get en set methodes zijn hierna getest aan het ingeladen level. Bij de WriteToFile wordt getest door dit level naar een file weg te schrijven. Er wordt hier één branche gemist op het moment dat de test voor een tweede maal gedraaid wordt. De savefile is dan al aangemaakt waardoor de test de branche “MisssingFile” mist. Hierdoor is er maar een branche coverage van 50% in Figuur 2. Als de test echter voor het eerst uitgevoerd wordt of de map Floor 1 uit JUnitTestLevel2 leeggemaakt wordt deze branche wel getest.

### Storey

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
WriteToFile(String)	<div></div>	98%	<div></div>	50%	1	2	1	16	0	1
Read(String)	<div></div>	100%	<div></div>	n/a	0	1	0	24	0	1
Storey(int, int, int, int)	<div></div>	100%	<div></div>	n/a	0	1	0	11	0	1
Storey()	<div></div>	100%	<div></div>	n/a	0	1	0	11	0	1
Storey(int, int, int, int, WallList, FloorList, RoofList, ObjectList, PickupList)	<div></div>	100%	<div></div>	n/a	0	1	0	11	0	1
setSizeX(int)	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
setSizeY(int)	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
setFloorHeight(int)	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
setRoofHeight(int)	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
setWalls(WallList)	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
setFloors(FloorList)	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
setRoofs(RoofList)	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
setObjects(ObjectList)	<div></div>	100%	<div></div>	n/a	0	1	0	2	0	1
getSizeX()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getSizeY()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getFloorHeight()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getRoofHeight()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getWallList()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getFloorList()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getObjectList()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getPickupList()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
getRoofList()	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
Total	3 of 465	99%	1 of 2	50%	1	23	1	98	0	22

Figuur 2: Storey Coverage Report

Zowel de use cases als de state transitions zijn handmatig in game getest. Dit is gedaan door het spel in te gaan en verschillende situaties te testen. Ook kon er met eclipse variabele waarden gecontroleerd door meerdere keren `System.out.print` weer te geven in het console-tablad van eclipse. In game werd er voor de use cases vooral gekeken of de gewenste resultaten gerealiseerd werden. Vragen werden gesteld, zoals:

1. Is de gebruiker ingelogd in het systeem na het inlogscherf?
2. Staan de nieuwe gegevens in de database na registratie in het inlogscherf?
3. Gaat er health af van de enemy na hitten?
4. Gaat er wel alleen health af van de enemy als de speler dichtbij genoeg staat?
5. ....

Zo hebben we de situaties getest tot we de content waren met het resultaat.

Voor het testen van de state transitions echter, werd in het bijzonder rekening gehouden met de gebruiker input. Wanneer de gebruiker in het pauze scherm zit, moet deze niet gebruik kunnen maken van de mainmenu knoppen die wellicht op de achtergrond nog actief zijn. Rekening houdende hiermee werd met een soortgelijke testmethode aan die van use cases, ook de state transitions getest.

## 5 CONCLUSIE

Medieval Invasion is een in acht weken gecreëerd 3D game in Java. Het spel werkt zoals het hoort. Toch zijn bugs niet onoverkomelijk. In het spel en de leveleditor, wanneer het intensief gebruikt wordt, zitten toch nog aardig wat fouten. Dit heeft te maken met de doelen die gesteld zijn aan het begin van het project. Sommige waren wellicht wat te ambitieus. Dit zorgt voor een enorme tijdsdruk wanneer het project zijn einde nadert. Toch kan gesteld worden dat de game een succes is. Ook al is acht weken kort, er kan toch met trots naar de game gekeken worden. De game verhoudt zich niet met de spellen van nu, maar kan toch plezierig zijn om een keer te spelen.

## 6 EVALUATIES

In dit hoofdstuk geeft elk van de ontwikkelaars van het spel een korte evaluatie. Ze zullen vertellen wat hun bijdragen waren aan het project en hoe zij het hele proces van game development hebben ervaren.

### 6.1 EVALUATIE HASSAN AL MAHMOEDI

Mijn eerste taak tijdens het project was om samen met Johnny Wang een gamestate manager te maken. Omdat we allemaal weinig programmeer ervaring hadden verliep dat in eerste instantie heel stroef. Door veel google werk, wat hulp van de assistenten en wat trial en error was het uiteindelijk toch gelukt. Daarna bespraken we gaande weg steeds af wie wat moest doen. Hierdoor kwam het erop neer dat ik me, in het vervolg, vooral bezig heb gehouden met de verschillende gameobjecten zoals enemy, player, door en de verschillende wapens en animaties ervan. Voor de animaties van de wapens en vijanden heb ik ook wat werk moeten doen in het programma Blender.

Tijdens het project zijn we verschillende problemen tegengekomen. Hetgene waar ik het langst vast zat, was, zoals hierboven vermeld, het maken van de gamestate manager.

Een ander probleem dat we tegenkwamen was het inladen van 3D modellen. Na veel werk van Guido Remmerswaal was het uiteindelijk gelukt om .obj files in te laden. Toen ik daarna de vijanden wilde animeren kwam we er achter dat .obj modellen niet te animeren zijn. Omdat het te veel tijd zou kosten om een andere object loader te maken hebben we daar uiteindelijk omheen gwerkt.

Verder ben ik weinig problemen tegengekomen. Natuurlijk lukte alles niet in één keer, maar dat had meer met het feit te maken dat we weinig programmeer ervaring hadden.

Hetgeen waar ik me helaas aan heb gestoord is dat er voor dit project bij de vakbeschrijving alleen OOP programmeren als voorkennis stond terwijl als eis voor het project gesteld werd dat je misntens één element uit elk ander minorkvak moest implementeren. Aangezien ik een vrije minor volg en alleen OOP programmeren van de overige minor vakken heb gedaan vond ik het wel raar.

Over het algemeen denk ik dat het project best goed is verlopen. Ik vond dat ik in een goed groep zat waarin iedereen zijn taken goed en op tijd heeft gedaan. Ook ik heb veel programmeer ervaring opgedaan.

### 6.2 EVALUATIE RUBEN KOEZE

In eerste instantie was mijn taak om met Guido samen de leveleditor te maken. Dit ging vrij voorspoedig, en al snel, toen de basis van de leveleditor stond ben ik overgegaan naar het opbouwen van de maze. Hierna zijn alle taken door elkaar heen gaan lopen. Aspecten van de game die gedaan moesten worden werden gemaakt door degene die klaar was met hetgene waar hij daarvoor mee bezig was. Zo heb ik bijgedragen aan de leveleditor, het opbouwen van het level, het visuele ontwerp van de menu's, het visuele ontwerp van de hulpmiddelen in de game, de kern van het spel en het loginsysteem.

In het begin leek de opdracht onmogelijk, ik had geen idee waar ik moest beginnen. Toen we eenmaal op gang kwamen viel het uiteindelijk wel mee. Over sommige aspecten van de game moest goed worden nagedacht, en hebben we dan ook brainstormsessies gehad. Zo was de vraag hoe we de muren zouden opbouwen, hoe vijanden zouden reageren, wat de manier van aanvallen zouden zijn en ga zo maar door. Toch zijn we overal goed uitgekomen, vooral door goed met elkaar te overleggen en gezamenlijk een oplossing te kiezen.

Al met al denk ik dat ik veel heb opgestoken van dit project. Mijn Java programmeerkunsten zijn in ieder geval een stuk beter geworden. Ook het werken in een groep is voor mij nog nooit zo intensief geweest als hiervoor. Toch ben ik niet helemaal te spreken over hoe het project in elkaar zit. Je wordt toch wel erg in het diepe gegooid wanneer het project start. Ook ben je volledig afhankelijk van de studentassistenten, die hun best doen, maar zeker niet alles weten. Ook viel het me op dat ik de docent maar twee keer in de volle acht weken op toch 16 uur werkgroep heb gezien.

### 6.3 EVALUATIE GUIDO REMMERSWAAL

Over de gehele game ben ik samen met Ruben verantwoordelijk geweest voor de LevelEditor. Aan het begin van het project hebben we de basis van de LevelEditor opgebouwd en gedurende het project hebben we dit up-to-date gehouden. Na de basis van de LevelEditor heb ik mij voornamelijk bezig gehouden met het inladen van 3D modellen. Hiervoor heb ik een tutorial gevolgd en dit hierna zelf uitgebreid.

Aan het einde van het project heb ik mij gericht op het uitbreiden van het pathfinding algoritme. Johnny had een feromonen algoritme geïmplementeerd, maar met dit algoritme was het niet mogelijk om swarm-intelligence toe te voegen. Voor de pathfinding heb ik een NavigationMesh gebruikt. Na het schrijven van een A\* pathfinding algoritme heb ik als laatste nog een kleine hoeveelheid swarm-intelligence toegevoegd.

In VWO 6 heb ik een 3D game in elkaar gezet met een game-engine. Hierdoor had ik enig idee wat onze mogelijkheden voor deze game waren. Het moeilijke aan dit project was het ontwikkelen van een eigen game-engine. Bij inladen van 3D modellen, wat in een game-engine automatisch gaat, heb ik de meeste problemen gevonden. Na de tutorial te hebben omgezet van LWJGL (een game-engine voor JOGL) naar JOGL, bleek de manier van inladen geen textures te gebruiken. Hiervoor moest ik de VertexBufferObject methode uitbreiden, waardoor ik het inladen van modellen opnieuw kon schrijven. Ook het genereren van de NavigationMesh heeft veel problemen opgeleverd. Voordat Clipper en Poly2Tri gebruikte kan worden moeten er nog een aantal testen uitgevoerd worden. Het schrijven van een methode om te bepalen of een polygon in een ander polygon ligt heeft hierbij de grootste problemen gegeven.

Het ontwikkelen van een eigen game-engine heeft mij veel geleerd en meer waardering gegeven voor de games van dit moment. Het nadeel van dit project is dat er nog twee vakken gevolgd moeten worden tijdens de periode, maar dit moet ook direct in het de game geïmplementeerd worden. De opdracht is daarna niet volledig duidelijk over de eisen van de game en ook de studentassistenten weten niet precies wat noodzakelijk is. Hierdoor wordt het project een minder overzichtelijk.



## 6.4 EVALUATIE JOHNNY WANG

Aan het begin van het project werden de projectgroepjes aangeraden te beginnen met het bouwen van de level editor en het opzetten van de gamestates. Samen met Hassan Al Mahmoedi gingen we ervoor zorgen dat er een goede overgang zou plaatsvinden tussen de verschillende menu- en gameschermen. Toen deze opzet gedaan was verdeelden we de overige taken over de vier groepsleden. Eerst heb ik geprogrammeerd dat vijanden je konden volgen. Daarna werd dit uitgebreid met een artificial intelligence voor de vijanden door de computational intelligence methode van ant pheromones toe te passen. Toen de vijanden je eenmaal konden volgen werd het nodig collision detection tussen objecten te implementeren, dus ben ik hier aan verder gaan werken. Gedurende het hele project en vooral tegen het eind ben ik vooral nog bezig geweest met het uitbreiden van de gamestates, de verschillende menuschermen goed werkend te krijgen en deze goed mee te laten schalen bij het aanpassen van het window formaat. Dit laatste heeft me heel wat problemen meegeleverd.

Gedurende het hele project zijn we tegen best veel problemen aangekomen. Het grootste probleem waar ik mee zat was het meeschalen van het scherm bij het aanpassen van het window formaat, het reshaper. Omdat GLEventListener automatisch je canvas mee liet schalen, maar de buttonlocaties absoluut bleven en handmatig aangepast moesten worden, kwamen de weergegeven schermen en de userinput vaak niet overeen. Uiteindelijk leverde dit teveel problemen op, waarna wij ervoor hebben gekozen dat de speler aan het begin van het spel een schermresolutie moet kiezen waarop hij het spel wil spelen. Verder waren er een verscheidenheid aan andere problemen waarvan ik zonder twijfel van kan zeggen dat mijn collega ontwikkelaars daar al over hebben verteld.

Voorafgaand aan de minor had ik al enigszins basiskennis van programmeren. Ik was al bekend met Matlab, Delphi en PHP. Java was geheel nieuw voor mij en dus was het project dan ook een zekere uitdaging. Ik moet echter ook zeggen dat we echt in het diepe werden gegooid bij dit project. De begeleiding rondom de grote lijnen van dit project was beperkt, waardoor we veel moesten vragen aan de student assistenten en veel zelf moesten opzoeken op internet.

Desalniettemin heb ik veel kennis opgedaan rondom het programmeren van een project en ben ik zeker niet ontevreden met het bereikte resultaat.

## 7 LITERATUUR

- [1] Christoph Romstöck, *Generating 2D Navmeshes*, Bezocht op 8-1-2014, URL:  
[http://www.gamedev.net/page/resources/\\_/technical/artificial-intelligence/generating-2d-navmeshes-r3393](http://www.gamedev.net/page/resources/_/technical/artificial-intelligence/generating-2d-navmeshes-r3393)
- [2] Angus Johnson, *Clipper*, Bezocht op 8-1-2014, URL:  
<http://www.angusj.com/delphi/clipper.php>
- [3] *Poly2Tri*, Bezocht op 8-1-2014, URL:  
<https://code.google.com/p/poly2tri/>
- [4] Patrick Lester, A\* *Pathfinding for Beginners*, Bezocht op 9-1-2014, URL:  
<http://www.policyalmanac.org/games/aStarTutorial.htm>
- [5] Ash Hamnett, *Funnel Algorithm*, Bezocht op 10-1-2014, URL:  
<http://ahamnett.blogspot.nl/2012/10/funnel-algorithm.html>
- [6] Oskar Verhoek, *Obj-Loader*, Bezocht op 27-11-2013, URL:  
<https://github.com/OskarVeerhoek/YouTube-tutorials/tree/master/src>
- [7] TF3DM, Bezocht op 27-11-2013, URL:  
<http://tf3dm.com/>
- [8] Blender, Bezocht op 27-11-2013, URL:  
<http://www.blender.org/>