Universiteit Hasselt

Trimesteroverschrijdend Project

Raycasting

Auteurs:
Nick Michiels
(0623764)
Kenneth Devloo
(0623746)

Begeleider
Tom Van Laerhoven

2de bachelor in de informatica

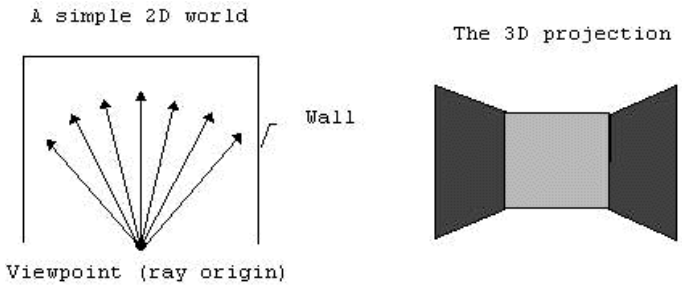
Academiejaar 2007-2008

Inhoudsopgave

1 Inleiding

1.1 Over raycasting

Spelletjes zoals Wolfenstein (de eerste versie) hebben een visueel systeem dat gebaseerd is op een raycasting engine. Zo'n engine maakt gebruik van een plattegrond waarop een aantal muren aangegeven worden. Door telkens 1 horizontale scan over het scherm uit te voeren, kan met de posities bepalen van de muren. Met behulp van enkele eenvoudige technieken kan zo de hele scene uitgetekend worden. Om het geheel iets mooier te maken kan weergave van de omgeving gebruik maken van textures. Tegenwoordig wordt deze techniek terug meer gebruikt in games voor gsm. Figuur 1 is een voorstelling van deze techniek.



Figuur 1: De rays worden vanuit viewpoint in een 2D wereld uitgestuurd en in 3D gemapped op een scherm.

1.2 Algemeen

We hebben gekozen voor de hogere programmeertaal C++ om dit project te maken. Deze keuze hebben we vooral gebaseerd op onze ervaring. Het gebruik van deze taal hebben we vrij goed onder de knie ten opzichte van andere talen. Ook de mogelijkheid om geheugenbeheer zelf af te handelen speelt in ons voordeel wegens het waarschijnlijk een geheugen-etend programma zou worden.

Verder heeft de mogelijkheid om Qt te gebruiken ons ook overtuigd. Qt is een zeer goed gedocumenteerde krachtige tool om een GUI te bouwen. Tevens biedt Qt ons de mogelijkheid om met een QImage onze projection plane (=scherm waar alles wordt uitgetekend) op een relatief snelle wijze te laten uittekenen.

Eerst wilden we in plaats van Qt een snellere tool gebruiken voor output, namelijk SDL. Maar wegens onze kennis hiervan vrij beperkt is en ons aangeraden is om Qt te gebruiken zijn we afgestapt van dit idee.

2 Onze implementatie

In een implementatie van raycasting is er veel ruimte voor uitbreiding. Naast het uittekenen van muren en vloer kunnen er nog veel extra's geïmplementeerd worden. Deze sectie beschrijft wat onze implementatie aankan en hoe we dit in grote lijnen hebben aangepakt. Voor muur, vloer, plafond en navigatie hebben we ons deels gebaseerd op tutorial [1].

2.1 Standaard raycasting

Zoals gezegd bestaat raycasting uit een aantal standaarden. In ons programma zijn ook deze standaarden opgenomen. Namelijk: muren, vloer, plafond, navigatie.

Hoe deze berekeningen in zijn werk gaan wordt uitgelegd in het raycasting algoritme 3.1.

2.2 Een wereld voorstellen

Een positie in een wereld wordt voorgesteld aan de hand van coördinaten zoals voorgesteld in Figuur 2. De wereld zelf bestaat uit vakjes die elk een grootte hebben van 64x64. Aan elk vakje worden eigenschappen gegeven. Zoals de muur, die voor noord, oost, zuid en west verschillend kan zijn, er kan een sprite bij horen, en er is ook telkens een vloer en plafond toegekend. Om uiteindelijk alles te kunnen berekenen worden ook nog eens 4 kwadranten vast gelegd en moet tijdens het raycasten worden rekening gehouden met botsingen op een horizontaal of verticaal vlak.

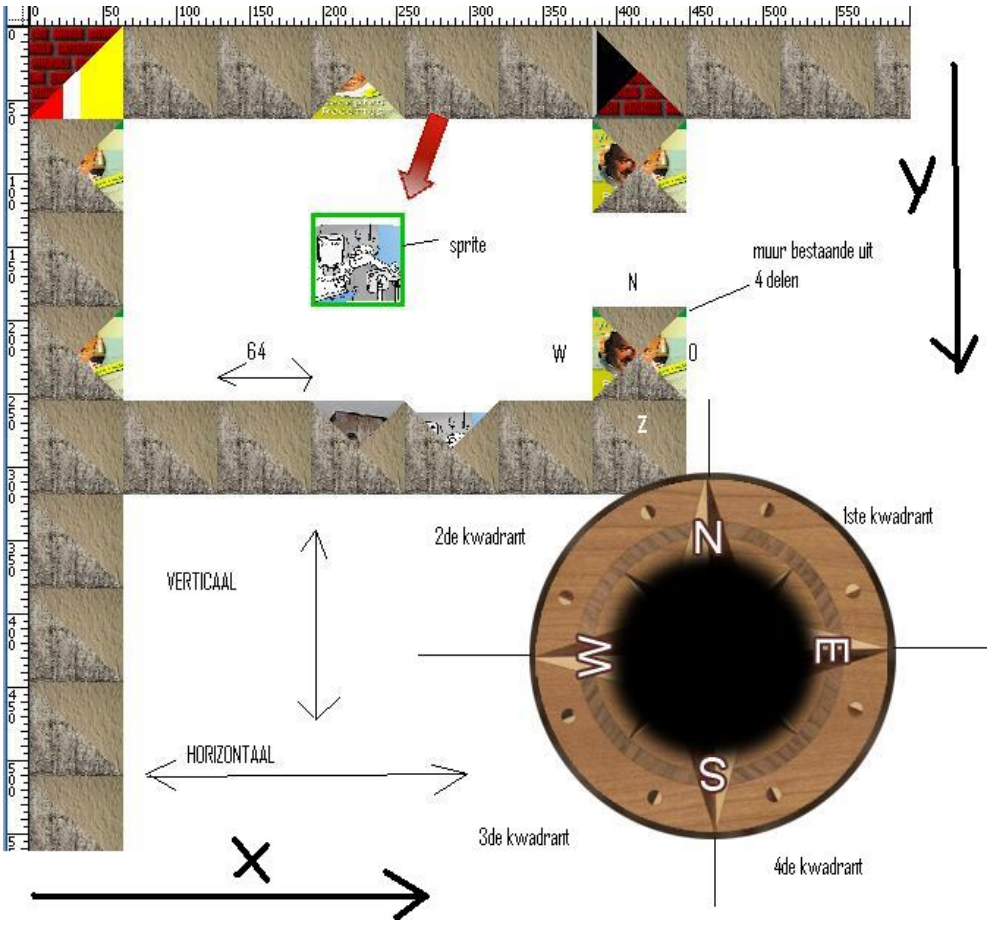
2.3 Triggers

Triggers zijn series van commandos die u kunt schrijven om een basisvorm van interactie met de wereld te hebben. Door het nummer van die trigger aan een vakje in de wereld toe te kennen kunt u die laten afgaan (door, afhankelijk van de eigenschappen van de trigger, over dat vakje te lopen of op 'T' te drukken). Een volledige trigger-editor is niet geïmplementeerd dus deze triggers moeten via een file ingegeven worden.

Meer informatie daarover is te vinden in de handleiding.

2.4 File input en output

Elke klasse die data bevat heeft zijn eigen *toString*- en *fromString*-functies om de inhoud van een klasse en zijn subklasses op te vragen als string (de klasse roept dus ook de *toString()*-functies aan van zijn subklasses). *FromString*-functies zijn om met een filestream data in te laden in deze klasses.



Figuur 2: Voorbeeld van een wereld.

Even wat uitleg over de uiteindelijke file gevormd voor world. Een voorbeeld kan u vinden in Figuur 3. We hebben ook een triggerfile, maar die wordt uitvoerig uitgelegd in de handleiding.

De eerste regel stelt de eigenschappen van de viewer voor. Viewer(x , y) staat voor de cordinaten waarop de viewer staat. FOV: x staat voor de field of view (kijkhoek), height: x staat voor de hoogte van de viewer, looking: x staat voor naar welke pixel in de hoogte van het scherm u kijkt en met DPP: x (distance projection plane) kunt u de hoogte van de muren wijzigen volgens de FOV.

Vervolgens komt per rij in de wereld, een rij in de file die elke cel in die rij zo gaat voorstellen: { wall? Northwall eastwall southwall westwall floor ceil trigger sprite isSprite walktrough }

Deze worden verder uitgelegd in de klasses en worden vooral gewijzigd in de editor.

De textures beginnen met de naam 'textures:' en vervolgens per nr het

```
Viewer:( 81 , 367 ) FOV: 66 ,viewAngle: 51 looking: 240 DPP: 985
red: -10 green: -10 blue: -10 trigger_everytime_on_square? 1 Height: 20 width: 30
0{1 26 26 26 26 2 3 0 0 0 1}{1 26 26 26 26 2 3 0 0 0 1}...
1{1 26 26 26 26 2 3 0 0 0 1}{0 0 0 0 0 26 26 0 0 0 1}...
...
textures:
0 images/white.png
1 images/redbrick.png
2 images/grid.jpg
3 images/wood.bmp
4 images/water.bmp
```

Figuur 3: Een voorbeeld van een file die een wereld voorstelt.

relatief of absoluut pad naar de afbeelding. De textuurlijst eindigt met -1 waarna de file eindigt op #. Verder dient ook vermeld te worden dat worldfiles de extentie .txt moeten hebben.

2.5 Editor

Enkel hardcoded wijzigingen kunnen brengen in een wereld vonden we ook maar niets. Met onze editor kunnen we *at runtime* onze wereld aanpassen. Elke cel in de wereld is voorgesteld in een kleine item waarop er aanpassingen kunnen gedaan worden. Zo kunnen we een cel definiëren als muur, vloer of plafond. Verder kunnen we textures manipuleren per cel, triggers toevoegen en sprites instellen. Dit alles hebben we geprobeerd zo overzichtelijk mogelijk weer te geven.

2.6 Sprites

Sprites zijn 2D-afbeeldingen die kunnen worden geplaatst in de wereld op een bepaald vakje. De afbeelding staat daar dan als object. In onze implementatie zullen de sprites altijd gericht zijn naar de viewer. Sprites kunnen deels doorzichtig zijn (.PNG afbeeldingen).

2.7 Walkthrough

Walkthrough kan ingesteld worden per vakje. Indien deze uit staat betekent dit dat u niet mag lopen over dat vakje. Dit is dus een doorzichtig alternatief voor een muur die kan gebruikt worden om delen van de wereld tijdelijk onbereikbaar te maken of te zorgen dat u niet door een sprite kan lopen.

2.8 Mipmaping

We hebben een eerder experimentele versie van mipmapping geïmplementeerd. Aan de hand van de afstand van de muur/vloer/sprite wordt een grotere/kleinere afbeelding gebruikt. Dit leverde geen optimaal resultaat op. Het aantal versies van afbeeldingen is dus terug gedrongen tot 4 om het gebruik van het ram-geheugen te beperken.

3 Algoritmes

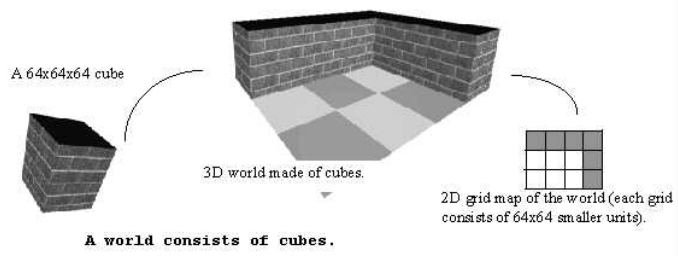
3.1 Raycasting

3.1.1 Belangrijke startgegevens

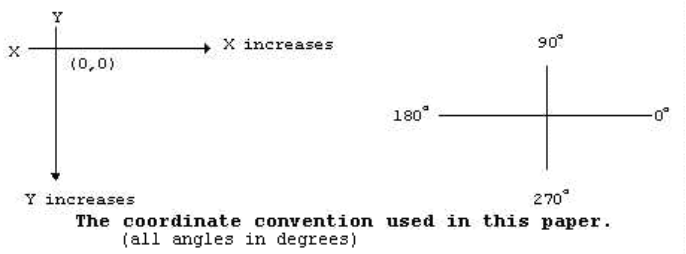
Alvorens te kunnen starten met het algoritme moeten er eerst een aantal afspraken worden gemaakt en een aantal startgegevens worden berekend.

Enkele conventies zijn onder andere de oriëntatie van de assen (zie Figuur 5) en de afmetingen per cel in de tabel. Eén cel neemt typisch 64 eenheden in bij de projectie (zie Figuur 4). Als eerste hebben we een projection plane nodig. Dit scherm waar de projectie op gebeurd is bij ons 640 pixels breed en 480 pixels hoog. Wegens de kijker een bepaalde kijkhoek heeft, hier 66 ° en de projectionplane deze kijkhoek inneemt kunnen we de afstand van kijker tot projection plane berekenen. Figuur 6 toont aan hoe we dit moeten doen. Voor een kijkhoek (ook Field Of View, FOV genoemd) en een projection plane van 640x480 geeft dit een afstand van 985.

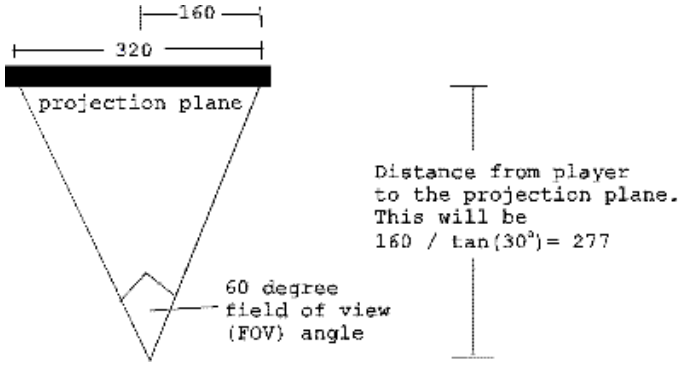
De kijker in de wereld heeft ook een aantal belangrijke gegevens. Zo heeft hij een x-positie en een y-positie: de coördinaten waar hij zicht bevindt. Deze bevat ook een hoek die de richting bepaalt waar hij in kijkt. De starthoogte van de kijker is 32, dit is de helft van de hoogte van de muur. De ogen van de kijker komen overeen met het midden van de projection plane, zo ook komt de kijkrichting recht uit in het middenpunt van de projection plane(Zie Figuur 7 en 8) Dit middenpunt is bij ons (320,240).



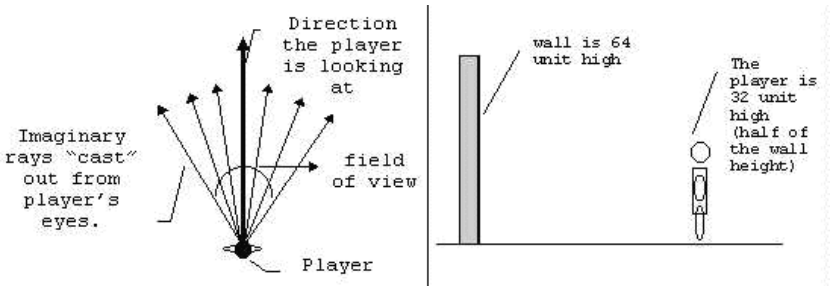
Figuur 4: Een cel in de tabel komt overeen met een kubus van 64x64x64.



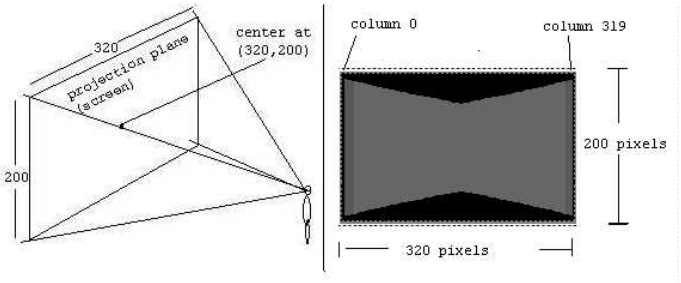
Figuur 5: Conventies in verband met het coördinatenstelsel.



Figuur 6: Met behulp van driehoeksmeetkunde kunnnen we de afstand tot de projection plane berekenen. In deze afbeeldingen zijn de afmetingen anders..



Figuur 7: De kijkerhoogte is de helft van de muurhoogte.



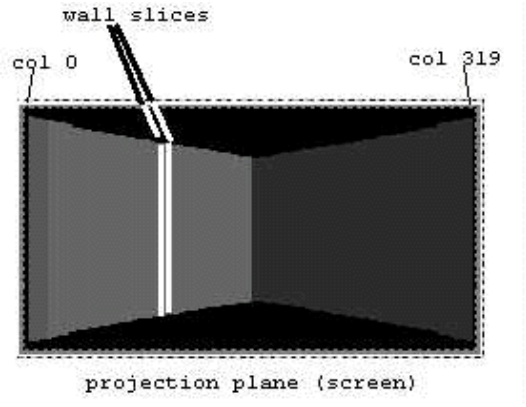
Figuur 8: De kijker kijkt altijd centraal naar het projection plane.

3.1.2 Raycasting

We weten nu dat het projection plane een kijkhoek van 66 ° moet innemen. En elke horizontale scan komt overeen met het zenden van 1 ray. De linkerkant van het scherm komt overeen met een hoek van kijkrichting+33 ° (let op: tegen de klok in worden de hoeken groter). Bij deze hoek wordt er telkens een hoek van 66 °/640 afgeteld tot we aan de rechterkant van de projection plane komen (Zie figuur 9). Volgend algoritme geeft aan hoe de rays worden uitgezonden en wat er met moet gebeuren:

1. Gebaseerd op de kijkershoek moet u 33 ° bijtellen (de helft van de FOV)
2. A Zendt een ray uit. Het is een lijn uitgezonden in de richting van de huidige hoek
   B Volg de ray over de verschillende cellen in de tabel totdat hij een muur raakt
   C Berekend de afstand tot de muur (= de lengte van de ray)
3. Trek van de hoek 66 °/640 af (de ray verplaatst zicht naar rechts)
4. Doe stap 2 en 3 totdat alle 640 rays zijn uitgezonden

Stap 2B van het algoritme is het moeilijkste. We gaan niet elke pixel moeten controleren of de ray een wall raakt of niet. Enkel op de intersecties van de tabel moeten we gaan controleren. Dit doen we in twee stappen: horizontale intersecties en verticale intersectien berekenen. Volgend algoritme laat zien hoe u de horizontale snijpunten berekent. Het algoritme maakt gebruik van Figuur 10.



Figuur 9: 1 horizontale scan komt overeen met een wall-slice en dit komt overeen met een uitgezonden ray.

1. Vind de coördinaten van het eerste horizontale snijpunt (punt A)

$$0^\circ < Ray < 180^\circ : A_y = floor(\frac{P_y}{64}) * (64) - 1$$

$$180^\circ < Ray < 360^\circ : A_y = floor(\frac{P_y}{64}) * (64) + 64$$

$$A_x = P_x + \frac{P_y - A_y}{\tan \alpha}$$

2. Vind Y_A: dit is 64 omdat het de hoogte is van 1 grid. Opgepast: Als de ray naar boven wijst is deze negatief, anders positief.

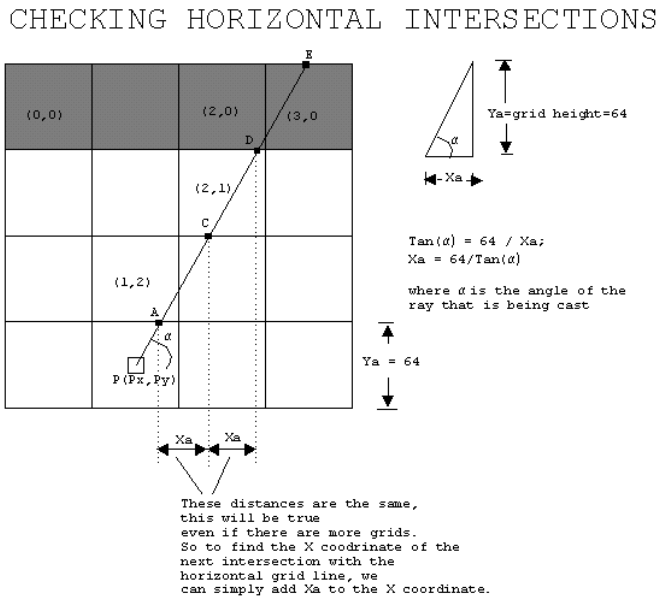
3. Vind X_A met de formule van op de tekening.

4. Kijk naar de grid op het snijpunt. Als er een muur is stop en bereken de afstand

5. Als er geen muur is: Bereken de coördinaten van het volgende horizontale snijpunt:

$$(X_{new}, Y_{new}) = (X_{old} + X_A, Y_{old} + Y_A)$$

   Ga naar stap 4.



Figuur 10: Algoritme voor horizontale snijpunten.

Voor de verticale snijpunten is het algoritme geheel analoog. Deze maakt gebruik van Figuur 11.

1. Vind de coördinaten van het eerste verticale snijpunt (punt B)

$$-90^\circ < Ray < 90^\circ : B_x = floor(\frac{P_x}{64}) * (64) + 64$$

$$90^\circ < Ray < 270^\circ : B_x = floor(\frac{P_x}{64}) * (64) - 1$$

$$B_y = P_y + (P_x - B_x) * \tan\alpha$$

2. Vind X_A: dit is 64 omdat het de hoogte is van 1 grid. Opgepast: Als de ray naar boven rechts wijst is deze positief, anders negatief.

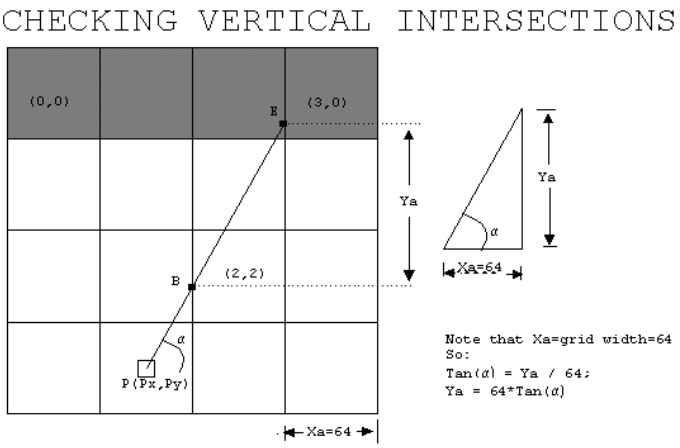
3. Vind Y_A met de formule van op de tekening.

4. Kijk naar de grid op het snijpunt. Als er een muur is stop en bereken de afstand.

5. Als er geen muur is: Bereken de coördinaten van het volgende verticale snijpunt:

$$(X_{new}, Y_{new}) = (X_{old} + X_A, Y_{old} + Y_A)$$

   Ga naar stap 4.



Figuur 11: Algoritme voor verticale snijpunten.

Beide algoritmes worden altijd uitgerekend. De muur met de korste afstand wordt uitgetekend. Nu weten we hoe de snijpunten moeten worden berekend en rest er ons enkel nog de afstand tot een snijpunt te berekenen. Hiervoor gebruiken we de formule met de goniometrische functies uit Figuur 12. We kunnen best deze met de goniometrische functies gebruiken omdat deze waardes uit een tabel komen en dus op voorhand uitgerekend zijn.

Als we met deze berekende afstanden de projectie uittekenen krijgen we een effect zoals in Figuur 13. Dit heet het zogenaamde fishbowl-effect. Het is een afwijking te danken aan het feit dat het menselijke oog bol is. De oplossing hiervoor is beschreven in Figuur 14.

Voor alle 640 rays kunnen we nu de wall slice gaan projecteren op het projection plane. Hiervoor moeten we de hoogte van de geprojecteerde wall slice weten:

$$\text{geprojecteerde slice height} = \frac{\text{werkelijke slice height}}{\text{afstand tot de slice}})*\text{afstand tot de projection plane}$$

We hebben enkele constanten namelijk: werkelijke slice height is 64 en de afstand tot de projection plane is 985. Door deze waardes in te vullen kunnen we de geprojecteerde slice height uitrekenen. De slice die u nu gaat uittekenen op de projection plane moet verticaal in het midden staan. Een grafische voorstelling hiervan vindt u op Figuur 15.

3.1.3 Floor Casting

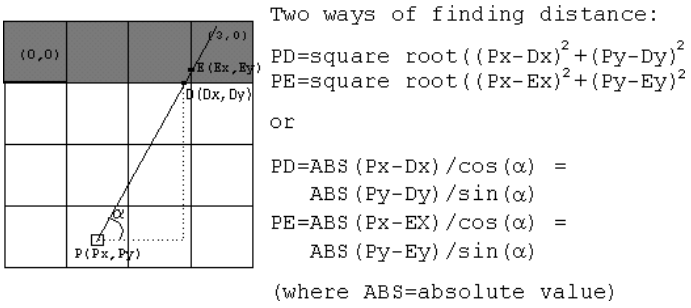
Het volgende algoritme laat zien hoe we een vloer moeten tekenen en hoe u voor 1 horizontale scan de vloer moet casten.

* Start op de onderkant van de geprojecteerde wall slice.

1. Neem de pixel.
2. Teken een ray van de pixel tot het oog van de kijker.
3. Verleng de lijn in de andere richting tot dat hij de vloer snijdt. Hoe u dit moet berekenen ziet u in Figuur 16.
4. Het punt waar de lijn de vloer snijdt, punt P, is het punt van de texture dat wordt geraakt door de ray.
5. Neem de pixel value van dat punt van op die texture en teken het op het scherm.

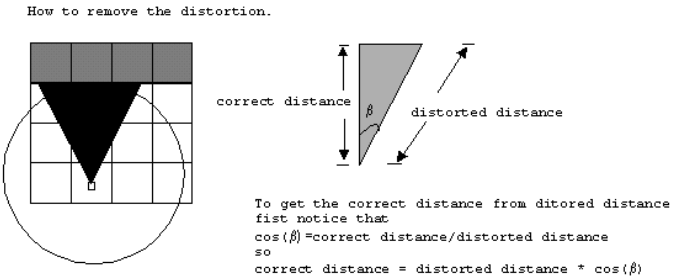
* Herhaal stappen 1 tot 5 totdat de onderkant van het scherm wordt bereikt.

Om punt P te berekenen gaan we een x-offset en y-offset berekenen die we bij de huidige positie van de kijker kunnen optellen om positie P te bekomen. In Figuur 17 hebben we een hoek β die de hoek van de ray relatief met de kijkhoek voorstelt. We kunnen nu een rechthoekige driehoek vormen tussen de ray, de lijn loodrecht op de kijkrichting en de lijn evenwijdig met de kijkrichting. Door gelijkvormigheid van hoeken weten we dat de ray de linkse zijde van de driehoek ook onder een hoek β snijdt. Met behulp van driehoeksmeetkunde kunnen we de x-offset en de y-offset bepalen. De y-offset komt overeen met de rechte afstand tot punt P.

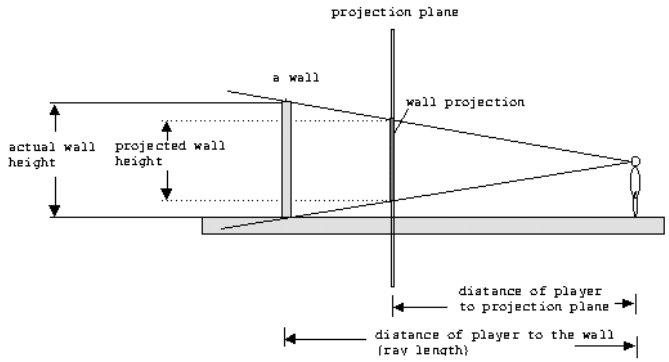


Figuur 12: Algoritme om de afstand tot een snijpunt te berekenen.

Figuur 13: Een afwijking naargelang we verder van de kijkrichting gaan. Het zogenaamde Fishbowl-effect.



Figuur 14: Een oplossing voor het fishbowl-effect.



Figuur 15: Projectie van een wall slice.

3.1.4 Ceiling Casting

Het uittekenen van het plafond is geheel analoog met het uittekenen van de vloer. Dit gaan we dus ook niet meer behandelen.

3.2 Navigatie

3.2.1 Vooruit en achteruit bewegen

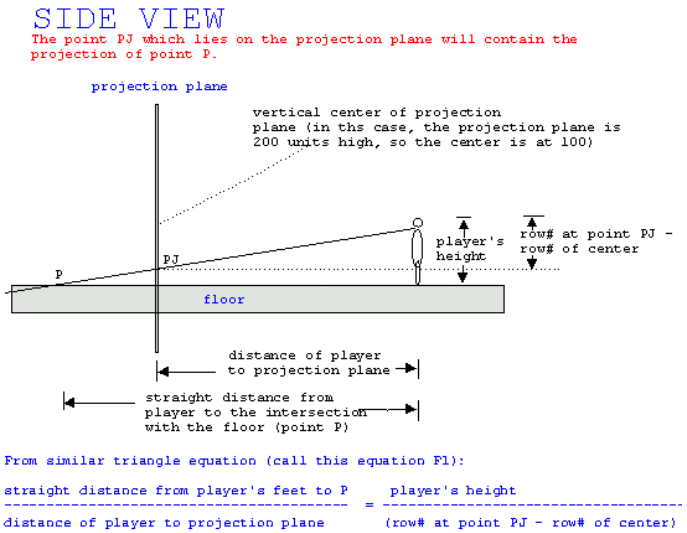
Om vooruit en achteruit te bewegen hebben we een movespeed nodig. De movespeed is de afstand die vooruit wordt bewogen in de viewrichting. Om nu werkelijk de viewer te verplaatsen in die richting hebben we een offset nodig in de x richting en een offset in de y richting. Met behulp van driehoeksmeetkunde zoals aangegeven in Figuur 18 kunnen we deze twee zijden berekenen. De hoek aangegeven op de tekening komt overeen met de kijkhoek van de viewer.

3.2.2 Draaien

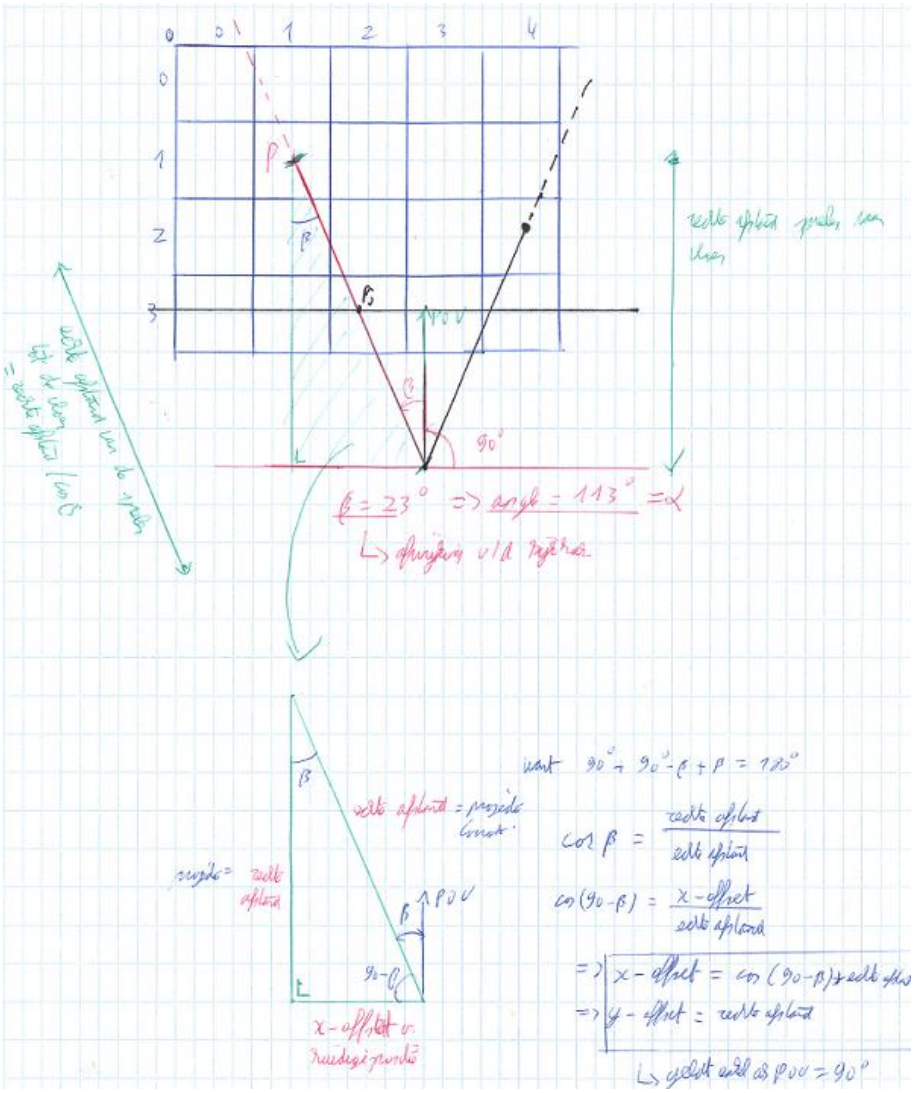
Om te draaien hoeven we enkel de kijkhoek van de viewer een aantal graden naar links of rechts te draaien.

3.2.3 Naar boven en beneden kijken

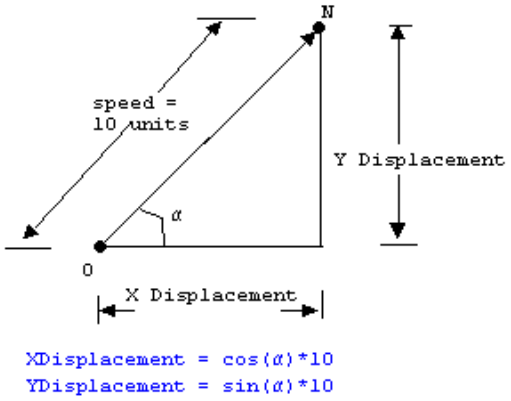
Onze projection plane is 480 pixels hoog. Normaal staat het verticaal centrum exact in het midden. Dus op een hoogte van 240 pixels. Door deze



Figuur 16: Floor Casting.

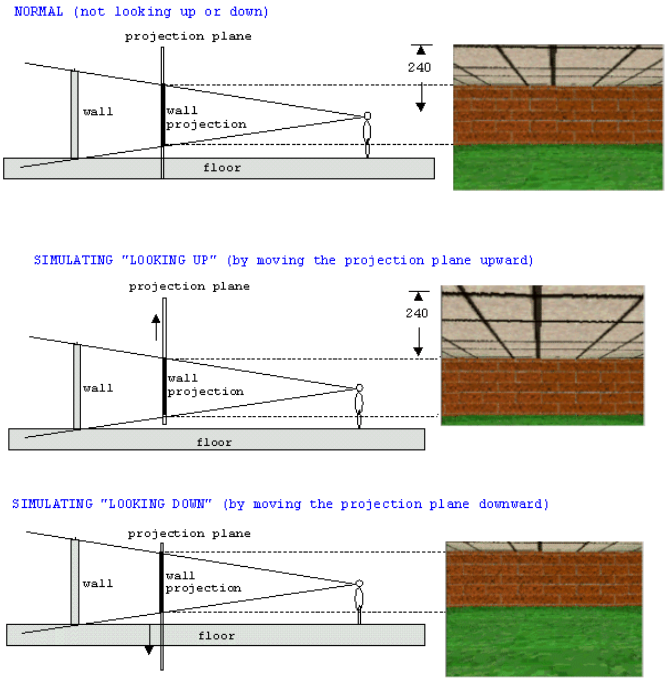


Figuur 17: Berekenen van de afstand tot de snijding met de vloer.



Figuur 18: Berekeningen nodig om vooruit en achteruit te bewegen.

hoogte te veranderen krijgen we het effect van *naar boven* en *naar beneden kijken*. Om naar boven te kijken moet het verticale centrum groter zijn als 240 (dus de projection plane naar boven plaatsen ten opzichte van de viewer). Het omgekeerde moet er gebeuren om naar beneden te kijken. Figuur 19 stelt dit visueel voor.



Figuur 19: Berekeningen nodig om vooruit en achteruit te bewegen.

3.3 Sprites

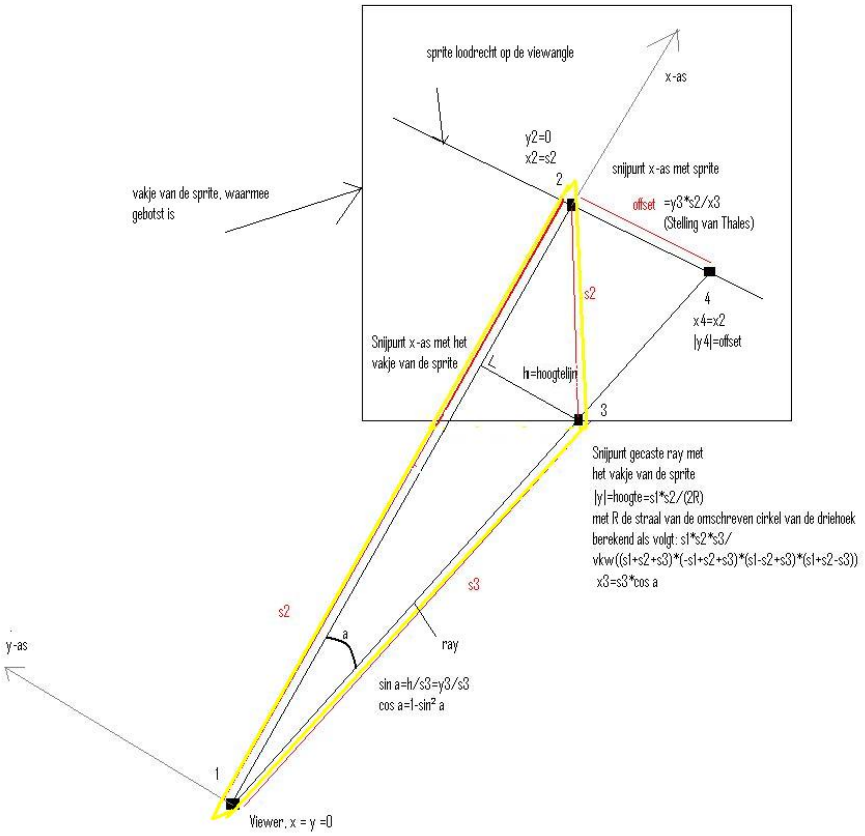
Sprites worden uitgetekend per verticale lijn op het scherm. Als er tijdens het raycasten een botsing is met een vakje van een sprite, dan moet worden uitgerekend welke verticale lijn van de afbeelding op die plaats moet worden uitgetekend. Dat gaat als volgt (zie ook Figuur 20):

- Om de juiste lijn te bepalen werken we in een nieuw assenstelsel zoals op Figuur 20.
- Punt 1, 2 en 3 kent u van het oude assenstelsel, zo kunt u de afstanden s_1,s_2 en s_3 bepalen met de afstandsformule: $\sqrt{(y_1 - y_2)^2 + (x_1 - x_2)^2}$.
- Eerst berekenen we de nieuwe x- en y-waarden van punt 3.
- Y-waarde: $y = \frac{s_1 * s_2}{2 * R}$ (formule voor de hoogtelijn h) met R de lengte van de straal van de omschreven cirkel van de gele driehoek in de figuur.
- $R = \frac{s_1 * s_2 * s_3}{\sqrt{s1+s2+s3}} * (-s_1 + s_2 + s_3) * (s_1 - s_2 + s_3) * (s_1 + s_2 - s_3)$
- Met de waarde van y berekend, kan over gegaan worden op de waarde van x:
- Dat kan door de sinus en cosinus van de aangeduide hoek op de figuur te berekenen, dan is $x_3 = s_3 * \cos a$.
- De uiteindelijke offset wordt dan $\frac{y_3 * s_2}{x_3}$.

Na het berekenen van de offset moet enkel nog bepaald worden of deze nu positief of negatief is (zie Figuur 21). Dit gebeurt met enkele berekeningen en maximaal 8 tests. Dit gebeurt door de hoek van de rechte van punt 1 en 2 te vergelijken met de ray-angle. Aangezien de eerste hoek niet perfect te berekenen valt met cosinusen$(0 \rightarrow 180)$ en sinusen$(-90 \rightarrow 90)$ worden ze beide apart berekend.

- Eerst kijken we naar de hoek berekend met behulp van de sinus.
- Vervolgens naar de hoek van de viewer
- Met behulp van de hoek, berekend met de cosinus; kunnen we uitsluitsel geven de hoek waarin dit lijntje sprite ligt. En kunnen we bepalen of we de rechter- of linker-kant van een sprite nodig hebben.
- Indien we de linkerhelft van de sprite willen: offset $= -$offset $+ 32$, rechterhelft: offset $+= 32$
- Een voorbeeld is in Figuur 21 gegeven.

Het algoritme wordt beëindigd door de afbeelding op te vragen, en een lijn uit te tekenen volgens de berekende offset



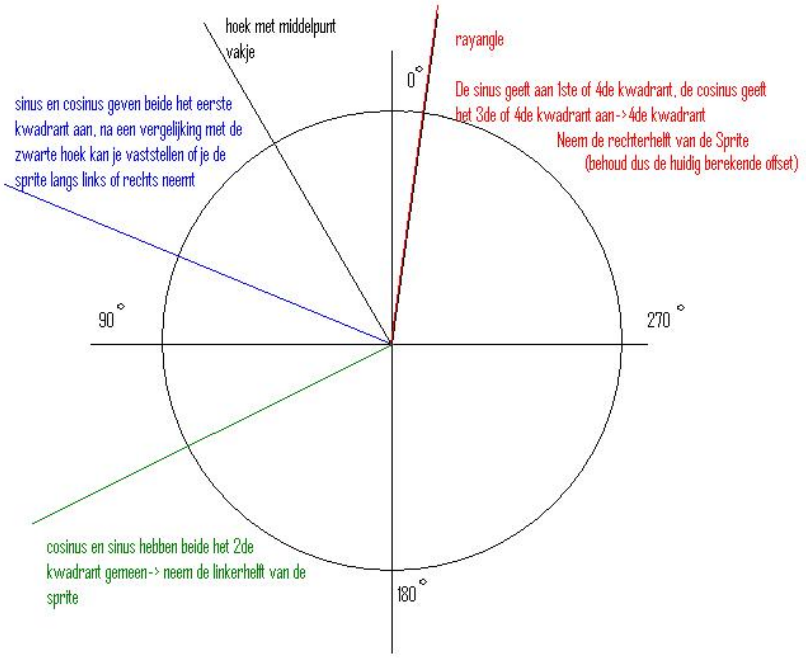
Figuur 20: Intersecties van de sprites berekenen.

4 Overzicht code

In dit onderdeel bespreken we de algemene structuur van ons programma. We gaan kort aanhalen waarvoor onze klasses dienen, maar we gaan niet dieper in op de specifieke implementatie. Voor gebruikte algoritmes verwijzen we u terug naar sectie 3.

4.1 Screen

Screen is onze klasse die de projection plane voorstelt. Hij heeft een functie gedefinieerd die punten kan uittekenen op het scherm. Doordat raycasting pixel per pixel werkt is het niet nodig dat deze meer functionaliteit aankan. Zoals uit de code blijkt zijn er hier enkele static-functies gedeclareerd. Hiervoor hebben we gekozen omdat we zo gemakkelijk vanuit elke klasse een pixel kunnen uittekenen naar het scherm.



Figuur 21: Zijden van de sprites.

4.2 World

Deze klasse stelt de huidige wereld voor waarin de viewer rond loopt. Hier wordt de grootte van de wereld bij gehouden en een matrix aan cellen (structs) die data per vakje in de wereld bijhouden. In elke cel wordt bijgehouden:

- welke vloer/plafond er op een bepaalde plaats komt, (textuurnummer)
- per windrichting welk soort muur er staat, (textuurnummer)
- het nummer van de sprite, (textuurnummer)
- het nummer van de trigger,
- en booleans om sprites , muren en de mogelijkheid om met de viewer op een vakje te staan uit en aan te zetten.

De wereld bevat ook functies om van afmeting te veranderen en een standaardwereld op te stellen. Alles wordt bijgehouden in deze struct omdat zo bij uitbreiding (bijvoorbeeld sprites) gemakkelijk de waarde van een sprite kan toegevoegd worden per vakje.

4.3 Textures

Hier worden alle texture die tot een wereld behoren opgeslaan. Bij het inladen van een textuur worden ook versies van de textuur gemaakt met vaste afmetingen (256x256 px, 64x64 px en 16x16 px). Deze worden gebruikt voor een eenvoudige versie van mipmapping. De 64x64 px afbeelding is ook nodig om afbeeldingen met een correcte grootte in de editor te gebruiken. De klasse heeft ook de mogelijkheid om een correcte afbeelding terug te geven afhankelijk van hoe ver deze staat in de wereld. Dit is enkel toegevoegd om de code in de engine te beperken en overzichtelijker te maken.

Al deze texturen worden opgeslaan in een vector die structs bijhoudt met afbeeldingen van verschillende groottes en een padnaam om later alles op te slaan. We hebben een vector gebruikt omdat deze random acces aan de elementen ondersteunt en dus bij grote lijsten een stuk sneller werkt dan bijvoorbeeld een map.

4.4 File

Deze klasse bevat enkel functies die de wereld- en triggerdata in strings verzamelen en wegschrijven naar files, of deze data terug inladen met een file.

4.5 Viewer

Voor te bepalen wat we moeten kunnen uittekenen op het scherm hebben we natuurlijk een camera nodig. Deze zit bij ons in de klasse **Viewer**. Een viewer in de wereld heeft enkele specifieke eigeschappen. Zo heeft hij een **positie (x,y)** nodig die uitdrukt waar hij zich in de wereld bevindt. Verder heeft hij ook een **kijkhoek** nodig die de richting bepaalt waar hij in kijkt. Als laatste zijn er nog enkele algoritme-specifieke waardens nodig zoals: **FOV (Field Of View)** en de **afstand tot de projection plane**. Als u niet meer goed weet waarvoor deze laatste termen staan kan u ze opfrissen in algortime 3.1.

4.6 Navigation

Om met behulp van de viewer doorheen de wereld te lopen hebben we een **navigatie**-klasse nodig. Hoe de navigatie mogelijk wordt gemaakt hebben we al reeds uitgelegd in 3.2.

In deze klasse definiëren we 3 snelheden, namelijk een **movespeed**, **rotatespeed**, **strafespeed** en **lookupspeed**. Deze zijn gekoppeld aan de hoofdtimer van het programma. Dit wil zeggen dat elke keer het beeld vernieuwd wordt, de viewer getracht wordt om met een movespeed vooruit te bewegen, met een rotatespeed te draaien, met een strafespeed zijwaarts te bewegen en met een lookupspeed naar boven of beneden te kijken . Het

is belangrijk dat standaard deze snelheden op 0 staan en pas als er een key wordt ingedrukt deze op een negatieve of positieve waarde wordt gezet.

Aan deze klasse hebben we ook nog extra functionaliteit toegevoegd. Het is dus mogelijk om **mousetracking** aan te zetten. Indien deze aan staat gaat elke mousemove worden opgevangen en met een offset van het middenpunt berekend hoe snel moet worden gedraaid in die richting.

4.7 Engine

De **engine** is de klasse die al de berekeningen doet in verband met raycasting. In de constructor krijgt hij gegevens van de **world**, **navigation**, **viewer** en **GUI** mee. Navigation heeft hij nodig omdat hij deze samen met de rest moet vernieuwen. Ook wordt hier bij het vernieuwen van een pagina getest of er een trigger mag afgaan of niet. De **UI** is nodig voor de triggers omdat deze moeten weten op welke objecten ze kunnen manipuleren. De overige twee pointers worden gebruikt voor het raycasting algoritme.

Belangrijk in deze klasse is de **timer**. Deze zorgt ervoor dat op een vast tijdsinterval de **teken-functie** wordt aangeroepen. Zonder deze timer zou het beeld niet mooi evenredig bewegen.

De **teken-functie** gaat ongeveer het hele algoritme uit 3.1 toepassen. Met andere woorden hij gaat een horizontale scan van links naar rechts doen en met gegevens uit de world en viewer bepalen hoe ver de botsing is plaats gevonden. Een belangrijke nuance op het algoritme is onze onderverdeling in **4 kwadranten** met daarbij de rechte hoeken 0 °, 90 °, 180 ° en 270 °. Deze onderverdeling was nodig om problemen met tekens van goniometrische functies en onbestaande waardes te vermijden. Via deze kwadranten kan ook worden gezien welke kant van een muur moet worden uitgetekend en dus welke texture er moet worden op gemapped.

Bij het testen of er op een snijding een muur is of niet wordt er nu ook getest of er een **sprite** is. Zo ja wordt ook de sprite uitgetekend met het algoritme 3.3

De teken-functie ondersteunt ook het uittekenen van de **vloer** en het **plafond**. Per wall-slice wordt er een functie aangeroepen die de muur uittekent volgens het algoritme van 3.1.4

4.8 Triggers

In de Triggers-klasse worden alle elementen van het type trigger bijgehouden. Deze worden in een **map** geplaatst.

We houden pointers bij omdat zo de triggers snel kunnen worden door gegeven.

4.9 Trigger

Hier wordt een takenlijst bijgehouden die moet worden uitgevoerd indien de trigger geactiveerd wordt. De trigger houdt ook informatie bij over welke vereisten nodig zijn om een trigger te kunnen starten. Deze vereisten zijn de volgende:

- leftAngle: De uiterst linkse hoek waarin de viewer mag staan als de trigger wordt geactiveerd.
- rightAngle: De uiterst rechtse hoek waarin de viewer mag staan.
- Key: Op true mag een trigger enkel geactiveerd worden als T wordt ingedrukt, anders wordt de trigger geactiveerd van het moment dat de viewer op een vakje met het triggernummer staat en aan de rest van de vereisten voldoet.
- Triggered: Deze variabele houdt bij of de trigger al eens is af gegaan of niet.
- allowTrigger: allowTrigger houdt bij of een trigger meer dan 1 keer mag aangeroepen worden.

Om een trigger te starten moet de functie startTasks aangeroepen worden. Deze krijgt pointers naar een viewer mee, naar een world, en naar een raycasting UI die worden opgeslaan in de trigger. Er wordt op alle vereisten gecontroleerd. Indien daaraan voldaan wordt kan de trigger opgestart worden in een thread (De Trigger is de thread zelf.) en kan het raycasten verder gezet worden samen met het uitvoeren van de trigger. Het uitvoeren van de trigger gebeurt door met een iterator de verschillende taskpointers binnen de klasse Tasks te doorlopen, hun execute te doorlopen en telkens een struct triggerdata mee te geven zodat deze task zijn opdracht kan uitvoeren.

4.10 Tasks

De klasse Tasks beheert de tasks. Hier worden de pointers naar een task toegevoegd en verwijderd. Een task wordt bijgehouden met een pointer naar elementen van het type Task binnen een map.

4.11 Task

Task is de basisklasse voor alle opdrachten die kunnen worden uitgevoerd op de wereld/viewer/ui. Deze verplicht elke afgeleide klasse om een executefunctie te hebben die een opdracht uitvoert en een toString functie om data over die afgeleide klasse op te slaan in een string. FromString-functies hebben de afgeleide klasses als constructor.

Van deze klasse zijn dus nog ongeveer 30 andere klasses afgeleid om aparte opdrachten uit te voeren in de wereld. De functies van deze afgeleide klasses komen aan bod in de handleiding.

De keuze om afgeleide klasses te nemen is om telkens enkel de variabelen op te slaan die nodig zijn, en om op een eenvoudige en snelle manier een onderscheid te maken tussen de verschillende opdrachten en manieren van opslaan en inladen.

4.12 Editor

De editor gebruikt gegevens van World en Viewer om de wereld voor te stellen. Hij is zo opgebouwd dat u interactief alles kunt aanpassen met behulp van **drie soorten views**, namelijk één voor de **vloer**, één voor het **plafond** en één voor de **muren**. Elke view afzonderlijk heeft zijn eigen functionaliteit. Zo dient de view voor de vloer specifiek om de textures van de vloeren aan te passen, de view voor het plafond specifiek om textures van het plafond aan te passen en de view voor de muren om de textures van de muren aan te passen.

Bij het inladen van een view gebeuren er altijd dezelfde stappen:

- Er wordt gegeken welke view er is opgevraagd.
- De editor wordt **gecleaned** en opnieuw opgevuld met **items**. Een item komt overeen met een cel in de World.
- Naargelang het type view worden de items anders ingevuld.
- In het geval van de views vloer en plafond worden de textures van de desbetreffende waarde ingeladen.
- In het geval van de view muur wordt een item opgedeeld in vier deelvakken en deze worden opgevuld met de vier textures die bij een muur horen.
- Vervolgens wordt er voor elke item gekeken of er een **sprite** aanwezig is. Zo ja, wordt dit bij de muur-view aangegeven door de texture van de sprite in te laden en bij de andere view door een vierkantje in het midden van de item te zetten.
- Als laatste wordt er gecontroleerd voor elke item of deze **walkthrough** is of niet. Als u niet door een vakje kan lopen wordt er rond de item een rode rand getekend. In het geval het ook een sprite is wordt het vakje in de vloer- en plafond-view rood ingekleurd.

Bij het uittekenen van de wereld moeten we ook de huidige positie en kijkrichting van de **viewer** uittekenen. Dit wordt in de vorm van een **pijl** voorgesteld en is een soortgelijk item als de andere items.

Het is mogelijk om op deze klasse een schaalparameter in te stellen. Via deze parameter kan de wereld uitgetekend worden op een schaal van 0.1 tot 1.

Naast de weergave van de wereld is er ook enkele interactieve functionaliteit nodig om deze te kunnen aanpassen. Hiervoor hebben we onderandere een pointer naar het **geselecteerde item** nodig. Deze pointer wordt opgevuld bij het klikken op een item van de wereld door de desbetreffende item. Met deze geselecteerde item kunnen we in de hoofdklasse werken als we textures of andere opties van een item willen aanpassen.

Naast de mogelijkheid om een item te selecteren is het ook mogelijk om een item te kopiëren en te plakken. Bij het kopiëren van een item worden de celgegevens van een item in een hulpvariabele gestopt en bij het plakken in de geselecteerde item terug ingeladen.

4.13 EditQGraphicsItem

In de bespreking van de Editor hebben we al aangehaald dat elke cel van de wereld wordt voorgesteld door een **item**. De klasse EditQGraphicsItem definieert zo'n **item**.

Een EditQGraphicsItem is niet meer als een uitbreiding van een **QGraphicsPixmapItem**. Deze uitbreiding is nodig omdat een item enkele signalen moet kunnen geven aan de editor:

- Als er op deze item wordt geklikt moet hij de editor laten weten dat er een **nieuw item is geselecteerd**.
- Ook moet de item laten weten als er een **drop** gebeurd is. Dit kan een drop zijn van een andere item of de pijl-item. Hij moet voor elke soort drop het juiste signaal doorzenden en de precieze coördinaten.

Het onderscheid tussen de twee drops kan hij maken door het feit dat er bij een drop van de pijl de letter '**a**' meekrijgt in de mimedata en bij een andere item niet.

4.14 ArrowItem

Zoals eerder vermeld wordt de viewer apart voorgesteld met een pijl-item. Deze item is ook een kleine uitbreiding van een **QGraphicsPixmapItem**. Het enige dat deze klasse moet kunnen is een drag starten en de mimedata '**a**' meegeven.

4.15 QEditTexture

Rechts in de editor vind u enkele textures terug die voor het geselecteerde item laat zien welke textures er voor zijn ingesteld. Elk van deze voorstelling is een oject van deze klasse.

In de klasse moet worden bijgehouden welke texture het voorstelt. Er zijn 7 mogelijkheden: vloer, plafond, sprite, linkermuur, rechtermuur, bovenmuur en ondermuur. Dit is nodig omdat een object van deze klasse aan de window wordt meegegeven die een texture gaat aanpassen. Zo weet dit window welke texture hij moet aanpassen.

Het object dat hij moet meegeven wordt bepaald door op welk object in de editor is geklikt. Dus in deze klasse wordt er een signaal teruggegeven met het huidige object indien er op geklikt is.

4.16 MiniMap

Voor de minimap hebben we een aparte klasse gemaakt die aparte tekenfuncties ondersteunt. Hij is ongeveer hetzelfde opgebouwd als de Sceen-klasse. Maar de afmetingen zijn anders en er is extra functionaliteit voorzien. De volgende stappen geven aan hoe de minimap wordt opgebouwd:

- Elke **pixel** van de minimap wordt afgegegaan van linksboven naar rechtsonder.
- Voor elke pixel wordt er in verhouding (op basis van een **schaal** die ingesteld kan worden) gekeken of er voor die pixel een muur is of niet. Er wordt aangenomen dat het middelpunt van de minimap de plaats van de viewer is.
- De kleur van de pixel wordt bepaald door het soort van **object (wall, out of bound, sprite, walkthrough)** er in de wereld staat.
- De afstand van deze pixel tot het middelpunt wordt berekenend en naargelang verder van het middelpunt wordt de pixel donkerder getekend.
- De hele minimap wordt gedraaid zodat de **kijkrichting** van de viewer altijd naar boven is gericht.
- Als laatste worden er nog twee lijnen met het Bresenham's line algoritme getekend die de **Field Of View** aangeeft.

4.17 QTexturesWindow

QTexturesWindow is een apart venster die de gebruiker een texture laat kiezen voor een bepaald object. Zoals eerder vermeld krijgt deze klasse via het hoofdprogramma een signaal binnen als er op een object van type **QEditTexture** is geklikt. Met behulp van een **spinbox** kunt u in dit venster een getalletje kiezen. Een getal komt overeen met een texture. Als u een getal hebt gekozen en op 'ok' hebt gedrukt wordt de overeenkomstige item opnieuw uitgetekend met zijn nieuwe waardes. In dat venster is het

ook mogelijk om **textures toe te voegen**. De user kan een afbeelding uit zijn usermappen selecteren en inladen in het programma. Waarbij het programma voor die afbeelding een nieuw getal gaat definiëren dat kan geselecteerd worden in de spinbox.

4.18 TriggersWindow

TriggersWindow is een venster met een tekstvak waar de triggerfile wordt ingeladen. In dit tekstvak kan u **at-runtime** de triggers tekstueel aanpassen. Als de aanpassingen worden gesaved zijn de triggers onmiddellijk beschikbaar in het verdere verloop van het programma. Deze klasse bevat dus enkele knoppen die verbonden staan met functies uit de **File**-klasse om rechtstreeks de wijzigingen weg te schrijven en in te laden in de triggers.

4.19 RaycastingQT

Dit is de hoofdklasse die de **GUI** van het programma voorstelt. Hij koppelt alle delen van het programma en interpreteert communicatie tussen de verschillene klasses. Verder zorgt hij ervoor dat de raycasting start en is vooral de basis van de editor.

5 Gekozen datastructuren

QImage Om de projection plane voor te stellen hebben we een QImage gekozen. Deze kan volgens zijn documentatie snel een pixel uittekenen. Aangezien bij raycasting er pixel per pixel wordt uitgetekend was deze keuze voor ons snel gemaakt.

QGraphicsView Voor de editor hebben we gekozen voor een QGraphicsView. Het was bij ons noodzakelijk om voor elk item afzonderlijk acties te kunnen definiëren. We hebben ook afgewogen om 1 grote afbeelding te maken waarop we alles uit zouden tekenen. Maar dan moesten we voor elke aanpassing van een cel de hele afbeelding hertekenen en bij het klikken op het scherm via een offset gaan bepalen welke wereld-cel hiermee overeen kwam. Het nadeel is dat we nu ook wel wat performantieverlies hebben doordat we bij het veranderen van view of schaal de hele QGraphicsview moeten legen en alle items opnieuw moeten aanmaken. Bij een grote wereld kan dit problemen geven.

Map Voor de triggers en de tasks hebben we de datastructuur map gebruikt omdat zoekopdrachten niet veel nodig zijn en de map ook iets veiliger werkt dan bijvoorbeeld een vector. (Het programma crasht niet bij een eventuele zoekopdracht die buiten de size ligt.)

Triggers Voor triggers hebben we zelf onze datastructuren geschreven. Triggers bestaat uit een lijst van trigger-objecten. Elk trigger-object heeft een takenlijst, de zogenaamde Tasks. Deze bevat op zich weer een lijst van pointers naar een Task-object. We hebben gekozen om een tasks voor te stellen met een abstracte klasse Task. Zo kunnen we met behulp van polymorfisme allerlei soorten tasks bij definiëren. Om een task uit te voeren wordt de pure virtuele functie **Execute** aangeroepen.

World De wereld is een datastructuur bestaande uit een tweedimensionale tabel van structs. Elke struct stelt een vakje voor in de wereld met alle bijbehorende eigeschappen.

6 Problemen

Tijdens het programmeren van dit project zijn we op heel wat problemen gebotst. In Figuur 22 kan u enkele voorbeelden zien van problemen. Hier een opsomming van de grootste problemen:

1. De world-klasse moest regelmatig worden uitgebreid om de uitbreidingen van de raycaster te volgen.

2. De hoogte en breedte van de wereld was in het begin hardcoded gebruikt in alle klasses. Hierdoor hebben we deze nog vrij laat moeten abstraheren.

3. Bij het gebruik van de goniometrische functies moesten we op de tekens letten. Ons coördinatensysteem kwam hier niet met overeen. Om dit op te lossen hebben we de hoeken onderverdeeld in vier kwadranten met daarbij vier rechte hoeken. Voor al deze mogelijkheden hebben we de goniometrische functies apart behandeld en manueel gezien naar het teken.

4. Voor de gemakkelijkheid hebben we de muren eerst behandeld als blokjes van 64x64x64. Maar de uiteindelijke bedoeling was om een blokje in te delen in een linker-, rechter-, boven- en ondermuur. Dit gaf heel wat problemen in ons raycasting-algoritme en we zouden nogmaals moesten onderverderdelen in de vier kwadranten. Ook voor de minimap wisten we dan niet meer hoe deze uit te tekenen omdat de muren maar 1 pixel breed zouden zijn. We hebben het dus maar gelaten bij blokjes van 64.

5. Op de één of andere manier wou bij de correctie van het fishbowl-effect de goniometrische formule niet werken. Toen we deze hebben aangepast naar de stelling van Pythagoras berekende hij het correct.

6. Waar ook goed op moest worden gelet was het gebruik van radialen i.p.v. graden.

7. De vloer en het plafond hebben we vrij snel kunnen uittekenen. Maar bij het naar boven en naar onder kijken leek deze in en uit de muren te schuiven.

8. Het draaien van de minimap en het kompas hebben we geïmplementeerd met een transformatiematrix op de afbeeldingen zelf. Hoe deze transformatiematrix moest worden opgesteld stond vrij vaag in de documentatie van Qt. We hebben dus heel wat moesten testen voordat we dit werkende kregen.

9. Sprites werden eerst uitgetekend door aan de hand van de wereldcoördinaten zelf. Door constante problemen hiermee is dit vervangen door het gegeven algoritme.

10. Textures werden eerst opgeslaan in een map, dit leverde ons een enorm trage raycaster op. De oplossing was om een vector te gebruiken.

11. Sommige berekeningen kunnen door afronding wel eens net buiten de wereld liggen, hierdoor werd een foute texture opgevraagd, wat niet goed samen ging met een vector. Extra beveiliging in de textuurklasse was het antwoord.

12. Onze eenvoudige versie van mipmapping bleek niet een gewenst resultaat te geven. Vandaar dat nu, om de raycaster niet te vertragen, slechts enkele kleine afbeeldingen worden gebruikt om pixels 'in de verte' uit te tekenen.

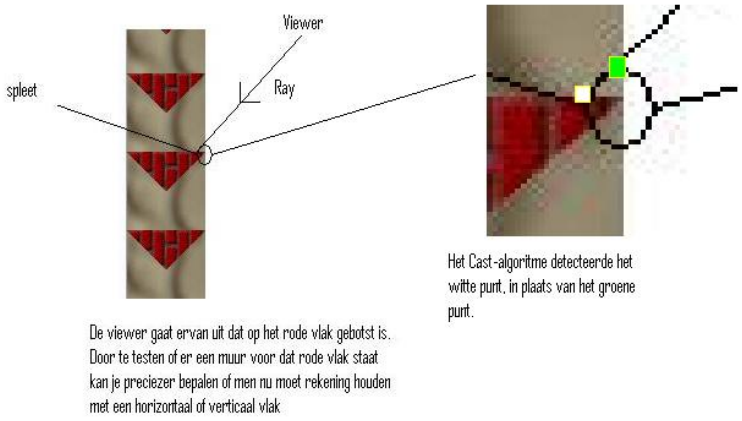
13. De minimap neemt 5% cpu-gebruik in, deze vertraging is verholpen door de minimap in een aparte thread te laten maken.

14. Triggers werkten eerst met Win32-threads. Na de invoering van een timer om de raycaster vanzelf te laten refreshen gaf dit problemen met toegang tot variabelen. Het gebruik van Qthreads loste dit op.

15. Tussen 2 muren in was er soms een kleine glitch door afronding. Het stukje aan de rand van die muur werd toen niet mee getekend, in plaats daarvan werd wel de muur daarachter uitgetekend. De spleten in een buitenhoek zijn verholpen door de horizontale- en verticale afstand te bepalen zonder het fishbowl effect mee te rekenen. Een oplossing voor 2 muren die naast elkaar staan is hieronder te zien. Het probleem was hier weer dat een horizontale botsing voorrang kon krijgen op een verticale botsing van de muur die er na kwam en omgekeerd. (Doordat deze dicht bij elkaar liggen.) Dit is opgelost telkens te kijken of er geen muur achter de zichtbare muur kwam (a.d.h.v het kwadrant om zo te

zien te weten waar die muur erachter precies kon liggen) en deze fout indien nodig te corrigeren. Figuur 23 illustreert dit probleem.

Figuur 22: De eerste 4 figuren geeft de evolutie weer van problemen met sprites, de laatste figuur geeft een voorbeeld van de spleten.

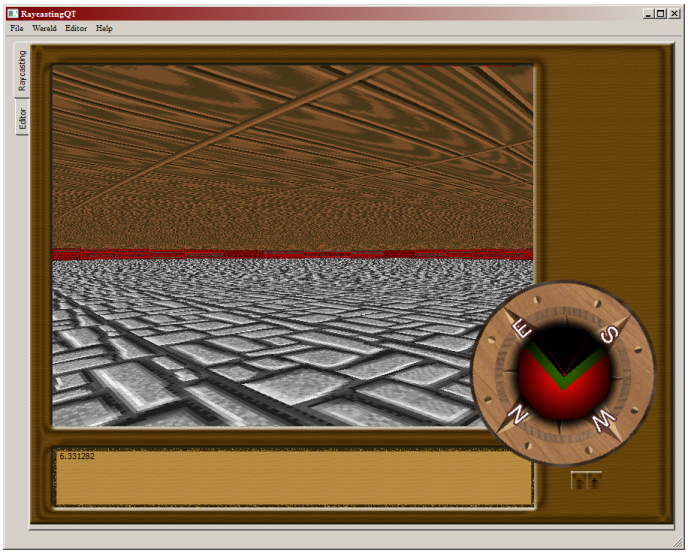


Figuur 23: Illustratie van de kloven in de hoeken.

7 Handleiding

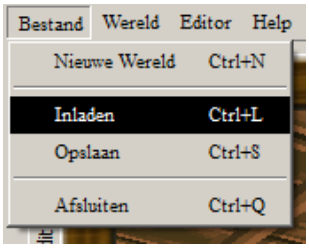
7.1 Hoe ga ik van start?

Bij het opstarten van het programma komt u direct in een basiswereld terecht. Het zou moeten opstarten zoals in Figuur 24.



Figuur 24: Bij het opstarten zit u direct in de basiswereld.

Een nieuwe wereld opstarten kan u bovenaan in de menubalk via **Bestand→Nieuwe Wereld**. Hier wordt er in een popup-window gevraagd naar de afmetingen van de wereld en vervolgens een nieuwe wereld gecreëerd met standaard-afbeeldingen. In het menu **Bestand** is het ook mogelijk om een wereld op te slaan en in te laden. Een wereld verkleinen of vergroten kan in de menubalk via **Editor→Nieuwe Grootte**.



Figuur 25: Menu Bestand waar u een wereld kan opslaan, inladen en aanmaken.

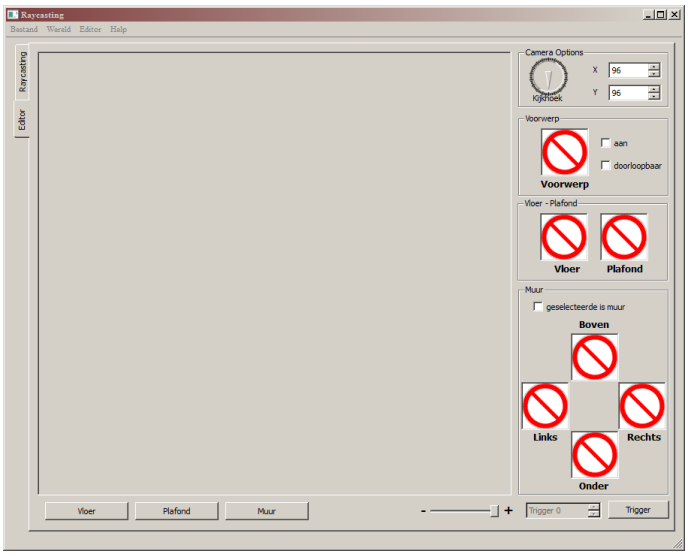
7.2 Bewegen in de wereld

Standaard staat mousetracking af. Dit wil zeggen dat er enkel met de toetsen van het keyboard kan worden rondgelopen. De bewegingen van de viewer worden bediend met de pijltjestoetsen. Door op **spatie** te drukken kan u de **mousetracking** aan- of afzetten. Bij het aanzetten van mousetracking wordt het gewoon draaien van de pijltjes links en rechts omgewisseld met een **strafe-beweging**. Naar boven kijken kan u door op de toets '**R**' te drukken en naar onder kijken met de toets '**F**'. Als laatste is het nog mogelijk te lopen met de toets '**E**'. Als u de minimap wil inzoomen en uitzoomen kan u op de + of −-knop drukken onder de minimap.

7.3 Editor

Om verder te gaan met de handleiding kan u best de file '**voorbeeld.txt**' inladen die zich in de map '**maps**' van de installatiefolder bevindt. De textures voor deze voorbeeldwereld hebben we gevonden op [6] en [7].

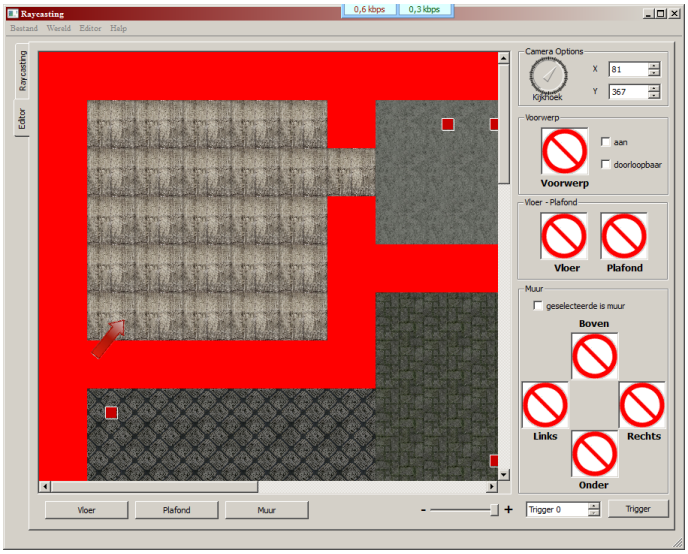
Links in het venster ziet u dat het programma is opgedeeld in 2 tabbladen. Als u op de tab **Editor** drukt komt u in de editor terecht. Uw beeld zou er dan als in Figuur 26 moeten uitzien. Zoals u ziet is er nog geen wereld weergegeven in de editor.



Figuur 26: De editor zonder dat er een wereld is weergegeven.

Om 1 van de drie views te kiezen in de editor moet u linksonderaan de pushbutton '**Vloer**', '**Plafond**' of '**Muur**' kiezen (zie Figuur 27). Als u nu bijvoorbeeld op de knop '**vloer**' drukt zou u normaal een beeld moeten krijgen zoals in figuur 28.

Figuur 27: Knoppen om een view in de editor te laden.



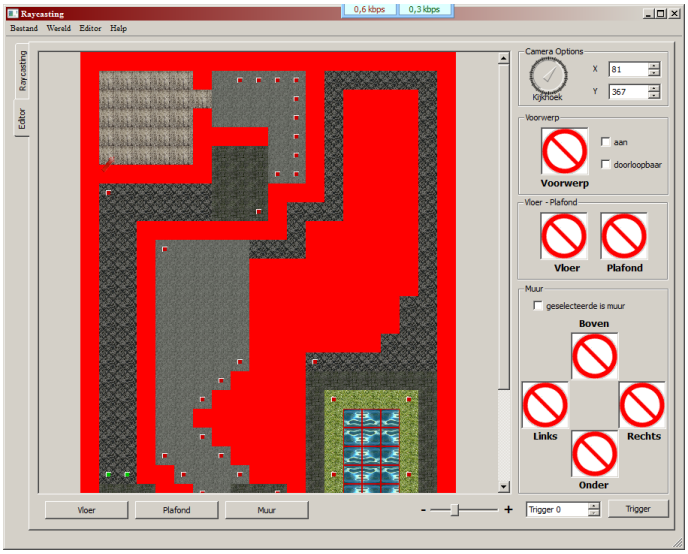
Figuur 28: Voorbeeld als de vloer-view is ingeladen.

Het is mogelijk om de map te laten in- en uitzoomen. Door met de slidebar te bewegen linksonder in het editscherm (Zie Figuur 29). Als u een beetje zou uitzoomen op de huidige map zou u iets moeten krijgen zoals in Figuur 30.

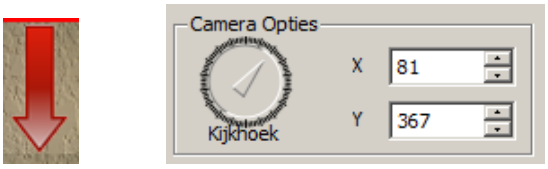
Figuur 29: Slidebar om in en uit te zoomen.

Nu kunnen we beginnen met wijzigingen aan te brengen. De viewer is in de editor voorgesteld als een **pijl**. Rechtsboven in de editor zijn er de camera opties. Hier kan u textueel de coördinaten van de speler aanpassen. Door middel van de wijzer in het klokje te draaien kan u de kijkrichting van de viewer aanpassen. Het is ook mogelijk om de viewer te verslepen door de pijl te **drag en droppen**. (Zie Figuur 31)

Om een vakje te selecteren moet u gewoon op een item drukken. U ziet rechts in de editor de overeenkomstige textures opvullen (Zie Figuur 32). Als er bij de texture geen verbodsteken staat wil dit zeggen dat u deze kan aanpassen. Wilt u bijvoorbeeld de vloer aanpassen kan u op deze texture



Figuur 30: Vloer-view van de editor die een beetje is uitgezoomd.



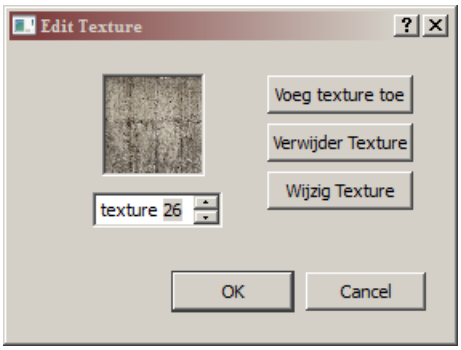
Figuur 31: De viewer wordt voorgesteld met een pijl.

klikken. Door op een texture te klikken opent er een nieuw venster. In dit venster, geïllustreerd in Figuur 33 kan u nieuwe textures vanuit file toevoegen en door op ok te drukken 1 selecteren. Het is de bedoeling dat u met behulp van de **spinbox** de juist texture selecteert. Dit werkt voor alle textures op dezelfde manier.

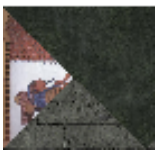
Een vakje veranderen naar een muur is ook vrij snel gedaan. U hoeft enkel de checkbox '**geselecteerde is muur**' aan te vinken. U zult zien dat in de huidige view het vakje nu rood wordt (wat wil zeggen dat het een muur is). Als u nu naar de muur-view gaat hebt u een beter overzicht over de textures op de muren. Een item dat voor een muur staat is ingedeeld in vier delen, elk deel is opgevuld met zijn overeenkomstige texture. Als u wat speelt met de textures van de muren zal al snel duidelijk worden waarom het zo is uitgetekend. Een voorbeeld van zo'n vakje is geïllustreerd in Figuur 34.

Zoals u misschien al hebt opgemerkt staan er op sommige plaatsen rode of groene vierkantjes in bepaalde items. Deze willen zeggen dat er een sprite aanwezig is. Een groene betekent dat u door de sprite kan lopen, bij een rode gaat dit niet. In de 'muur'-view zijn de sprites aangegeven als volle

Figuur 32: Instellingen voor een item.



Figuur 33: Venster om textures aan te passen item.



Figuur 34: Deze item stelt een muur voor item.

textures met een groene of rode rand rond. (Zie Figuur 35 en 36 Ook de textures van de sprites kunnen rechts op een identieke manier als de rest worden aangepast.

Om een sprite doorloopbaar of niet doorloopbaar te maken kan u de checkbox '**doorloopbaar**' aan- of afvinken. Om een sprite aan te zetten op een bepaald vakje moet u de checkbox '**aan**' bij '**voorwerp**' aanvinken.

De doorloopbaarheid kan u ook op een gewone item definiëren. U gaat zien dat er dan een kleine rode rand rond het vakje verschijnt.

Figuur 35: De twee verschillende voorstellingen van een doorloopbare sprite.

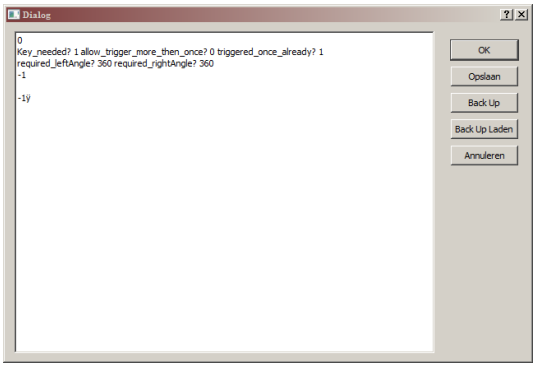
Figuur 36: De twee verschillende voorstellingen van een niet doorloopbare sprite.

In de editor is het ook mogelijk om de triggers aan te passen. Door rechtsonderaan in de spinbox het getalletje naast '**trigger**' te veranderen definieer voor de geselecteerde item de trigger met deze key. Als u nu zelf triggers wil definiëren of veranderen kan u op de '**trigger-knop**' drukken. Deze opent een nieuw venster met in een tekstvak de huidige triggers ingeladen. Hoe u de triggers moet aanmaken wordt uitvoerig uitgelegd in de volgende sectie. In Figuur 37 kan u het venster van de triggers zien. Door op save te drukken worden de triggers in de trigger-file opgeslagen en direct beschikbaar in het programma. Als u wat wil experimenteren met triggers is het mogelijk om op de '**backup**'-knop te drukken. Deze laat toe een veiligheidskopie te bewaren die later terug kan worden ingeladen als de gebruiker de triggerfile helemaal heeft dooreen gesmeten.

7.4 Triggers

7.4.1 algemeen

Triggers moeten dus langs een file ingelezen worden. De filenaam van de triggers is de filenaam van de wereld (incl de '.txt'- extentie) + een *'.tri'-extentie.*Een voorbeeld-triggerfile kan u zien in Figuur 38:



Figuur 37: Het venster waar u triggers kan aanpassen en definiëren.

```
0
key? 1 allow_trigger_more_then_once? 0 triggered_once_already? 1
required_leftAngle? 360 required_rightAngle? 360
-1
1
key? 0 allow_trigger_more_then_once? 1 triggered_once_already? 0
required_leftAngle? 0 required_rightAngle? 360
2 ChangeFloors to_texture 4 ,coordLU( 8 , 8 ) ,coordRU( 30 , 30 )
4 ChangeCeils to_texture 4 ,coordLU( 8 , 8 ) ,coordRU( 30 , 30 )
0 commentaar
3 Wait tijd 1000
6 SetImage to 18 start( 300 , 300 ) , offset( 100 , 100 )
-1
2
key? 1 allow_trigger_more_then_once? 1 triggered_once_already? 0
required_leftAngle? 90 required_rightAngle? 180
1 MoveViewer relative 1 to_pos( 200 , 65 ) ,viewAngle: 45 ,height: 10 ,looking: 240
,DistanceProjectionPlane: 985 , FOV 66
4 AddText
You pressed the evil T!
~
6 SetImage to 19 start( 300 , 300 ) , offset( 250 , 250 )
7 Wait duur 2000
8 RemoveImage
-1
-1
```

Figuur 38: Dit is een voorbeeld van een trigger-file.

Een eerste belangrijk punt is dat enkel de variabelen en commandonamen hier van belang zijn, de rest wordt mee gegenereerd bij het opslaan. Het is wel verplicht om spaties te hanteren waar ze gebruikt zijn en evenveel tokens tussen de variabelen te steken als in de file, vandaar dat het handig

is om te blijven bij de standaardtekst.

In deze file worden 3 triggers opgeslaan. Elke trigger begint met een bepaald nummer. De eerste trigger is de standaardtrigger en krijgt nummer 0 mee. Deze trigger is altijd aanwezig en moet niet worden ingegeven. U kan deze langs een file wel overschrijven door ze mee te geven. In de file heeft deze trigger de volgende eigenschappen:

- key? 1 : De toets 'T' is nodig om deze trigger te activeren, als u 0 ingeeft zal de trigger afgaan vanaf u over een bijbehorend vakje loopt.
- allow_trigger_more_then_once? 0 : Bepaalt of een trigger meer dan 1 keer mag afgaan. 0 betekent dat dat niet mag, 1 betekent dat dat wel mag.
- triggered_once_already? 1: Bepaalt of de trigger al eens is af gegaan of niet. O betekent nog niet, 1 betekent dat de trigger al eens is af gegaan.
- required_leftAngle? 360 :Bepaalt de uiterst linkse hoek waarin de viewer moet staan voordat de trigger af kan gaan.
- required_rightAngle? 360 :Bepaalt de uiterst rechtse hoek waarin de viewer moet staan voordat de trigger af kan gaan.

In deze eerste trigger zitten geen commando's, de trigger kan dan ook afgesloten worden (met een getal strikt kleiner dan 0).

De tweede trigger (hier als 1 genummerd) heeft opnieuw zijn parameters zoals hierboven uitgelegd en bevat wel commando's. Deze zijn genummerd, onafhankelijk van de volgorde waarin u ze mee geeft zullen commando's toch uitgevoerd worden volgens het nummer dat u ervoor zet. Indien u het nummer 0 mee geeft wordt die regel beschouwd als commentaar (max. 499 karakters). Na het nummer geef u telkens de naam mee van het commando die u wilt uitvoeren gevolgd door zijn parameters. Opgelet, bij een fout commando wordt daar het inlezen van de file gestopt. De precieze betekennis van de commando's wordt hieronder uitgelegd. Na de commando's dient de trigger ook afgesloten te zijn met een getal kleiner dan 0.

7.4.2 De commando's

Elk commando heeft dus een naam. Het begin van de naam zegt al iets over de functie ervan:

- Set∗: Bepaalt meestal of iets aan of uit wordt gezet. Dus dat iets is bvb wel of niet zichtbaar, hoewel er al data kan zijn van dat iets.
- Flip∗: Als iets aan staat gaat het hiermee uit en omgekeerd. U switcht dus tussen aan en uit.

- Change∗: Zet de waarde van iets (meestal van een textuur). Hiermee wordt niets aan gezet, dat gebeurt langs set of flip.
- Nog andere commandos die appart moeten behandeld worden.

Een lijst van alle mogelijke commando's met ingevulde waardes:

1. ChangeFloor to_texture 3 ,coord(2 , 4)
2. ChangeFloors to_texture 9 ,coordLU(2 , 2) ,coordRU(7 , 22)
3. ChangeCeil to_texture 3 ,coord(2 , 4)
4. ChangeCeils to_texture 9 ,coordLU(2 , 2) ,coordRU(7 , 22)
5. ChangeWall tex_n: 54 tex_e: 23 tex_s: 54 tex_w: 12 ,coord(4 , 4)
6. ChangeWalls tex_n: 54 tex_e: 23 tex_s: 54 tex_w: 12 ,coordLU(4 , 22) ,coordRU(4 , 30)
7. ChangeSprite to_texture 11 ,coord(4 , 4)
8. ChangeSprites to_texture 15 ,coordLU(22 , 2) ,coordRU(4 , 30)
9. ChangeTrigger trigger: 345 ,coord(5 , 5)
10. ChangeTriggers trigger: 153 ,coordLU(4 , 2) ,coordRU(22 , 30)
11. FlipWall coord(3 , 5)
12. FlipWalls coordLU(3 , 6) ,coordRU(7 , 12)
13. FlipSprite coord(3 , 5)
14. FlipSprites coordLU(3 , 6) ,coordRU(7 , 12)
15. FlipWalktrough coord(3 , 5)
16. FlipWalktroughs coordLU(3 , 6) ,coordRU(7 , 12)
17. SetWall to 1 coord(6 , 3)
18. SetWalls to 0 coordLU(6 , 3) ,coordRU(44 , 12)
19. SetSprite to 1 coord(6 , 3)
20. SetSprites to 0 coordLU(6 , 3) ,coordRU(44 , 12)
21. SetWalktrough to 1 coord(6 , 3)
22. SetWalktroughs to 0 coordLU(6 , 3) ,coordRU(44 , 12)
23. SetColors Red: 100 Blue: 100 Green: 20

24. General coord(12 , 13), north: 4 east: 7 south 11 west: 1 ,floor: 2 ,ceil: 2 ,trigger: 9 ,sprite: 11
25. Generals coordLU(12 , 5) ,coordRU(15 , 2), north: 4 east: 7 south 12 west: 1 ,floor: 2 ,ceil: 2 ,trigger: 9 ,sprite: 11
26. SetImage to 12 startcoords(680 , 20) ,offset(140 , 270)
27. RemoveImage
28. AddText Dit is een vb-tekstje

    met enters.

    ~
29. Wait time: 4000
30. MoveViewer relative? 0 to_pos(567 , 876) ,viewAngle: 30 ,looking: 260 ,DistanceProjectionPlane: 500 ,FieldOfView: 66
31. TriggerMultipleSet to 1 coord(6 , 3)
32. TriggerMultipleFlip coord(3 , 5)
33. 1 If SPRITE 1 1 EQUAL 2 RemoveImage
34. Music music/naam.wav

Veel commando's verschillen maar van elkaar in enkel of meervoud, bvb ChangWall en ChangeWalls, ChangeWall betekent een verandering in 1 vakje, ChangeWalls betekent een verandering in meerdere vakjes en is er slechts voor het gemak.

Het verschil in parameters is als volgt Coord(x , y) voor triggers die werken op 1 vakje en coordLU(x_1 , y_1) ,coordRU(x_2 , y_2) voor triggers voor meerdere vakjes. Coord(x , y) is dus de cordinaat van het vakje. Verder betekent coordLU(x_1 , y_1) ,coordRU(x_2 , y_2) anders geschreven coordLinks(Left)boven(up)(x_1 , y_1) ,coordRechts(right)onder(Under)(x_2 , y_2) dat de trigger invloed zal hebben van x_1 tot x_2 en van y_1 tot y_2.

De meeste triggers bevatten ook to_texture x, die de textuur uit de wereldfile bepaalt waarin iets gewijzigd wordt (naar textuur x).

North, east, south en west staan voor de zijden van een muur als een commando deze kan aanpassen en dienen ook een textuurnr mee te krijgen.

Hiermee is de functie van de meeste triggers normaal al duidelijk: ∗Wall(s), ∗Wallktrough(s),∗Sprite(s), ∗Ceil(s), ∗Floor(s) en ∗Trigger(s) passen dus de muur, doorloopbaatheid van een vakje/meerdere vakjes, de sprite, plafond, vloer en trigger aan in een bepaald vakje. Van Triggers zijn enkel ChangeCommando's aanwezig, en van walktrough(die bepaald of u over een vakje mag lopen) enkel flip- en set- commando's.

Een algemene trigger is general, deze zal (zonder iets aan of uit te zetten) de muur, plafond, vloer, trigger en sprite aanpassen.

Verder zijn er ook de volgende triggers:

- SetColors die een extra kleurlaag op het scherm kan brengen.
- SetImage die een textuurnummer meekrijgt van een afbeelding die op het scherm geplaatst wordt. Samen met de startcordinaten ervan en een maximum grootte van de afbeelding (offset).
- RemoveImage die die afbeelding opnieuw verwijdert.
- AddText die een tekst onderaan het venster plaatst, het commando wordt afgesloten met een ’; op een nieuwe lijn.
- Wait die een pause tussen 2 commandos in kan brengen, in het voorbeeld is die pause 4 seconden.
- MoveViewer die alle eigenschappen van de viewer aanpast, dus naar een positie, een hoek, kijkhoogte, DPP en FOV. Relative 1 staat voor het feit dat u relatief werkt ten opzichte van de huidige settings, met Relative 0 geeft u absolute waarden mee.
- TriggerMultipleSet en TriggerMultipleFlip: Indien deze aan worden gezet gaat deze trigger in actie treden elke frame dat u op een vakje staat waaraan deze trigger is toegekend. Indien deze op 0 staat, gaat de trigger slecht 1 keer per vakje in actie treden bij het betreden.
- De If-test werkt als volgt: U begint met het te controleren onderdeel (SPRITE, NOORD,ZUID,OOST,WEST,TRIGGER,POSX,POSY,ANGLE(Met de laatste 3 ervan eigenschappen van de viewer.)), gevolgd door de grid-coördinaten (Indien u op een viewereigenschap controleert,geeft u dummywaardes mee.), gevolg door een operator(EQUAL, BIG, SMALL (=,>,<)), gevolgd door de waarde waarmee moet vergeleken worden. Vervolgens schrijft u de taak neer die moet uitevoerd worden indien de test slaagt.

Nog enkele bijzonderheden aan de tasks:

- Texturen/afbeeldingen dienen altijd een nr ≥ 0 mee te krijgen.
- Behalve bij General(s) waar u -1 kan meegeven voor waardes die u niet wilt veranderen.
- Aan MoveViewer kan u ook de waardes -1 meegeven als er sommige eigenschappen niet mogen veranderen.

Het editen van de triggers kan in notepad of in de triggereditor (met gelijke mogelijkheden als notepad).

7.5 Extra aanpassingen die kunnen

In de file staat standaar de DPP ingesteld op 985. Met deze waarde kan u de projectie aanpassen van de wereld aanpassen. Als u hem bijvoorbeeld op 476 zet gaan de muren groter worden uitgetekend. Dit oogt voor sommige personen beter. De FOV is ook aanpasbaar. Deze gaat de wereld iets breder uittekenen.

Een mooie uitdaging om de kracht van de triggers aan te tonen is om een damspel na te bootsen in een wereld. Wat mogelijk zou moeten zijn met de iftesten.

8 Niet opgeloste problemen

1. In sommige kwadranten kunnen er nog stukken vloer en plafond uit de muren steken.
2. Bij het gebruik van triggers kan het programma nog heel zelden een break geven. De oorzaak hiervan hebben we nog niet gevonden.
3. Sommige doorzichtige sprites worden van dichtbij slecht afgebeeld.
4. Het trigger-window waar u in het programma triggers kunt aanpassen gaat soms tekens achteraan toevoegen waardoor de triggers niet meer fatoenlijk willen inladen.

9 Taakverdeling

In de code staat specifiek per klasse aangegeven wie de hoofdauteur is. Doorheen het project zijn er verschillende aanpassingen gebeurd in elkaars code. Indien dit is gebeurd en de moeite waard was hebben we dit bij die specifieke functie beschreven.

9.1 Kenneth

Vloer, Plafond, Triggers, Sprites, Optimaliseren, Files, Mist, Navigatie, World, ...

9.2 Nick

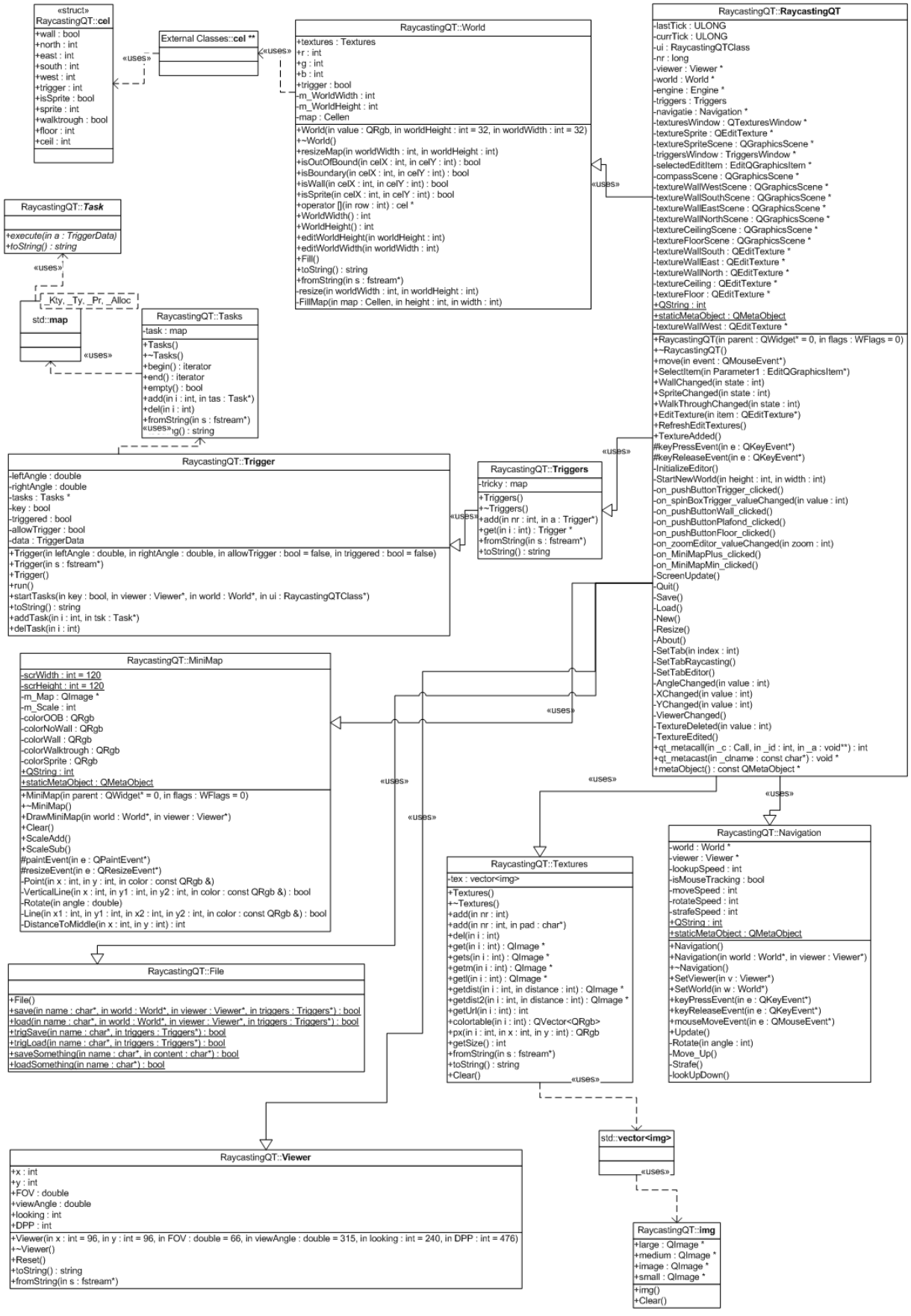
Basisengine, Editor, GUI, Minimap, Navigatie, Viewer, Screen, World ...

10 Logboek

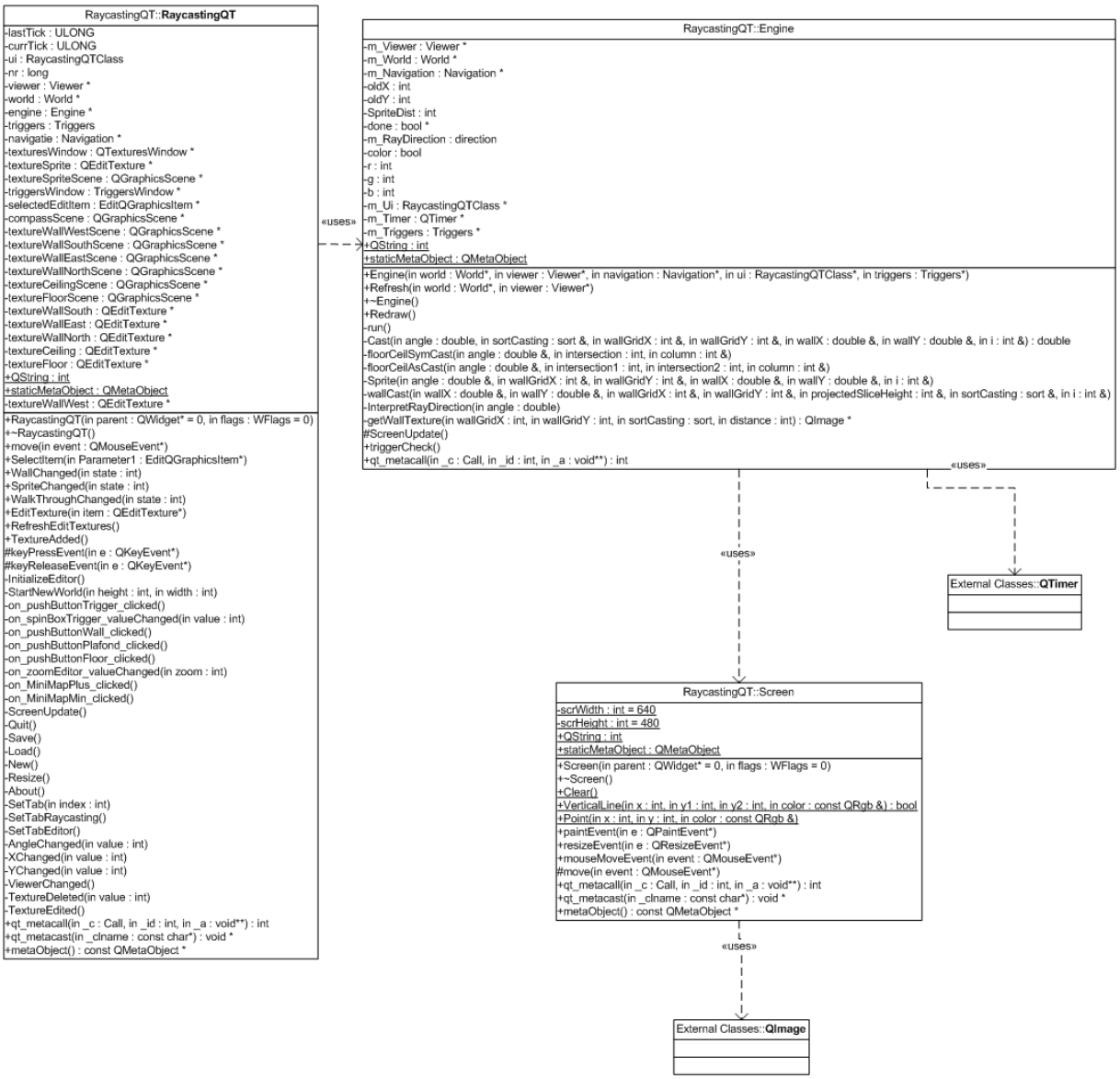
Het logboek is achteraan het verslag toegevoegd.

11 UML

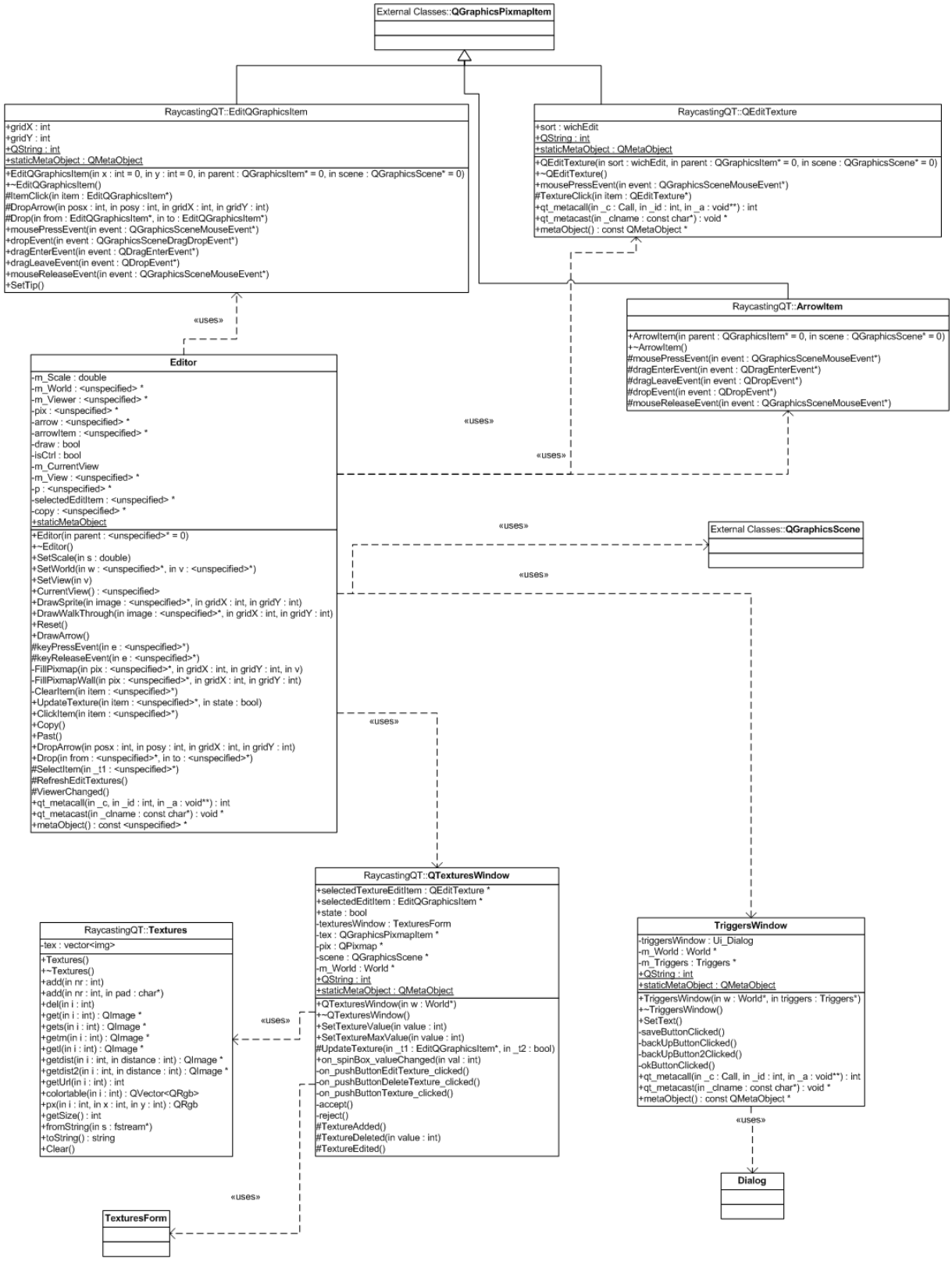
De UML kan u vinden in de map doc/UML. Hij bestaat uit 5 delen. In de *umlBasic.png* wordt de basiskoppeling uitgelegd hoe onze gegevens zijn opgebouwd. De hoofdklasse gebruikt de Engine voor de berekeningen. De uml van de engine staat in *umlEngine.png*. Zoals eerder vermeld gebruiken de triggers een lijst van tasks. Omdat we zoveel verschillende tasks hebben staat de uml hiervan apart in de *umlTasks1.png* en *umlTasks2.png*. Als laatste is er nog een *umlEditor.png* die de uitwerking van de klasse Editor voor zich neemt. U kan de uml's ook bekijken in Figuur 39, 40, 41, 42, en 43.



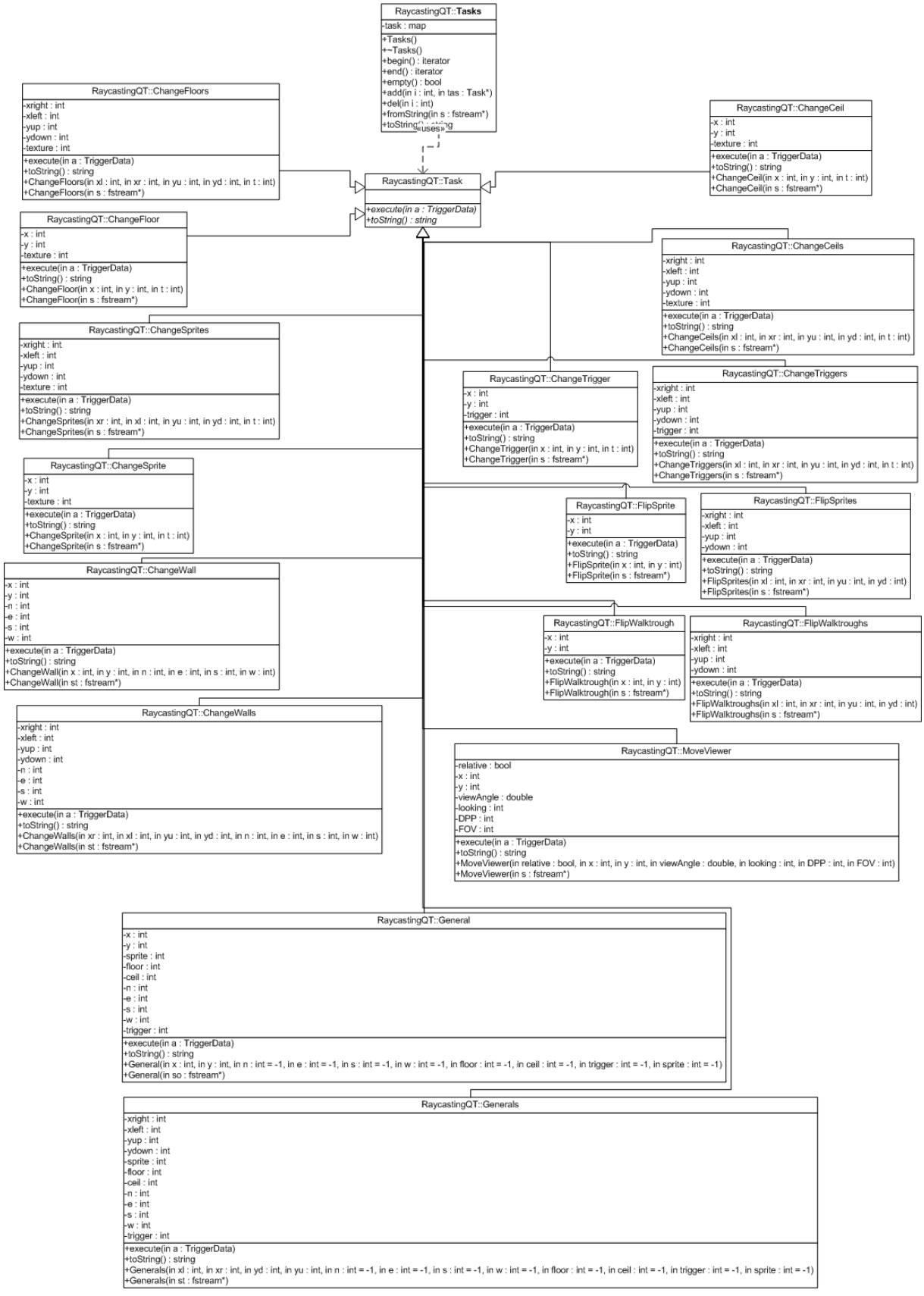
Figuur 39: Basis opbouw weergegeven in een uml-diagram.



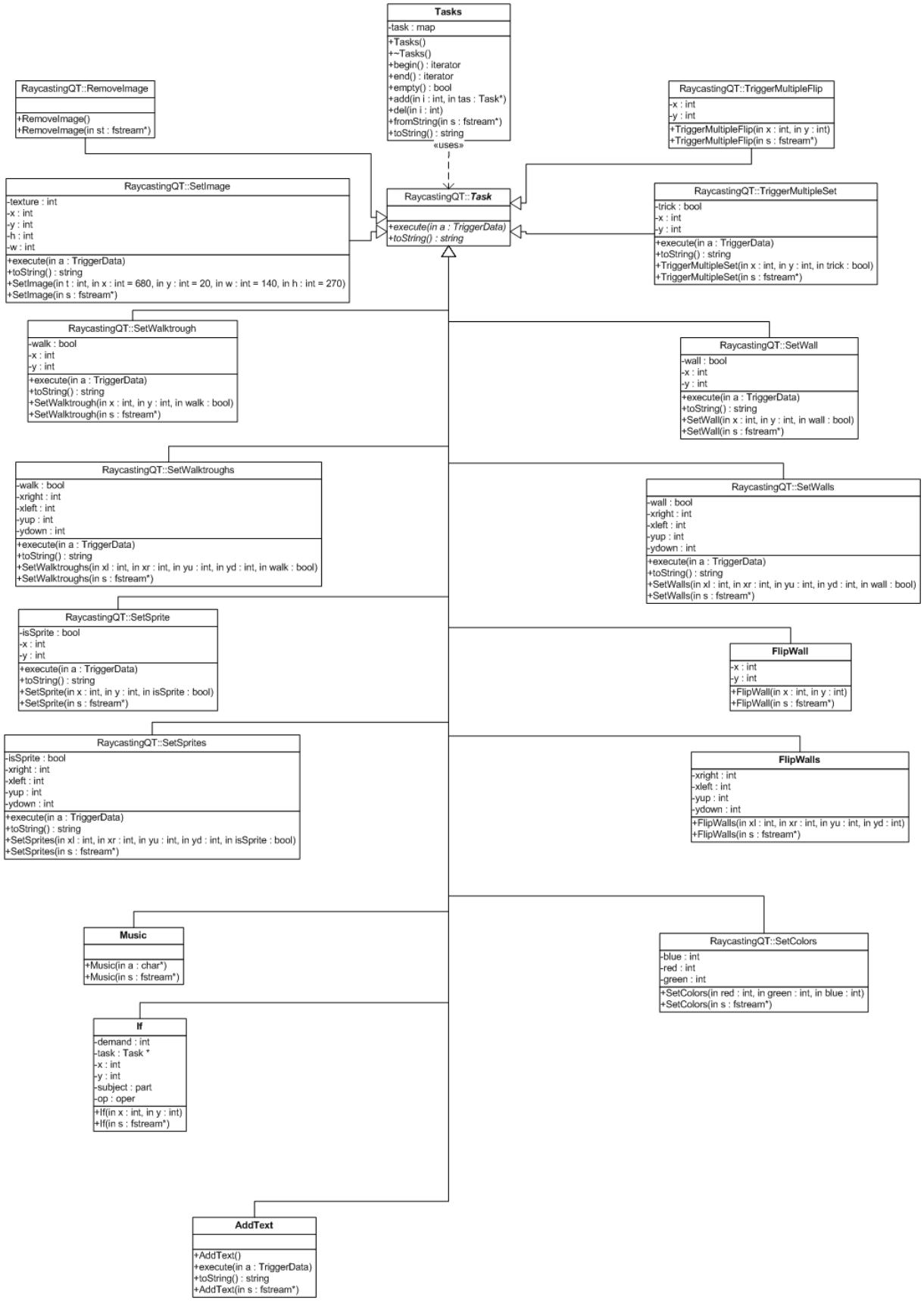
Figuur 40: De engine in een uml-diagram.



Figuur 41: De opbouw van de editor in een uml-diagram.



Figuur 42: Lijst van taken in een uml-diagram - deel 1.



Figuur 43: Lijst van taken in een uml-diagram - deel 1.

Referenties

[1] F. Permadi, tutorial, World Wide Web, http://www.permadi.com/tutorial/raycast/, 1996.

[2] TrollTech, documentation, World Wide Web, http://doc.trolltech.com/4.3/, 2008.

[3] L. Vandevenne, Lode's Computer Graphics Tutorial, World Wide Web, http://student.kuleuven.be/ m0216922/CG/, 12 augustus 2007

[4] Wikipedia, Hoogtelijnen, World Wide Web, http://nl.wikipedia.org/wiki/Hoogtelijn_(meetkunde)

[5] Wikipedia, Cirkel, World Wide Web, http://nl.wikipedia.org/wiki/Omgeschreven_cirkel

[6] CG Textures, Textures, World Wide Web, http://www.cgtextures.com/

[7] Hazel.H, Egyptian Textures, World Wide Web, http://www.hazelwhorley.com/textures.html