Documentatie Doolhof v1.0

Gemaakt door: Emiel de Brouwer & Robbert Goey

Gebruiksaanwijzing Doolhof

Je speelt in een First Person View. Gebruik de vierpuntdruktoetsen op het toetsenbord om jezelf te bewegen: druk op “omhoog” om vooruit te lopen, druk op “links” of “rechts” om in die richting te keren en druk op “onder” om achteruit te lopen.

Vind de sleutel en gebruik deze op de deur om door te gaan naar de volgende kamer.

Op het scherm krijg je hints/berichten. Druk op enter om deze te verwijderen als je het hebt gelezen.

Requirements

De volgende requirements zijn in de game geimplementeerd:

- Technisch:
  - Visitor Pattern voor de Renderer
  - Visitor Pattern voor de Updater
  - Visitor Pattern voor de Collisions
  - Builder voor het samenstellen van de game elementen (uit een separaat ‘level’ bestand)
  - Factory voor het creëren van diverse zaken (mesh, text, gameobject, behaviours etc.). Wordt met name gebruikt door builder.
  - Singleton voor o.a. de tijd (netjes conform pattern)
  - Omgeving opgebouwd met een Scene Graph
  - Twee lichtpunten
- Game:
  - 4 Ruimtes
  - Verzamelt dingen (in dit geval sleutels) om deuren te openen
- HUD:
  - Tijd
  - (Aantal) sleutels
  - Scores
  - Hints

Gebruikte patterns

Visitor Pattern

We gebuiken het visitor pattern voor de renderer, de updater en de collisions. Dit zorgt ervoor dat alle render, update en collision operaties bij elkaar in een klasse zitten ipv verspreid over meerdere klasses. Ook een voordeel is mochten we een nieuwe visitor toe willen voegen dan hoeven hiervoor geen bestaande klasses worden aangepast.

We hebben ervoor gekozen om het objectmodel de traversal te laten doen in plaats van de visitor.

Het objectmodel zorgt ervoor dat kinderen ook gevisit worden, dus ook de accept methode van alle kinderen aanroepen. Ook de mesh, behaviour en collider van een object vallen hieronder.

Rendervisitor

Alle draw methoden uit de Visitable klasses worden vervangen door de visit methodes van de RenderVisitor. Behaviour en Collider hoeven niet te worden gerenderd dus die visit methodes doen niets.

Builder en factory

Voor de opbouw van de levels gebruiken wij het Builder en factory pattern. De builder leest een textfile uit (“level.txt”, standaard). Vervolgens gebruikt deze builder de factory om alle objecten aan te maken.

Singleton

Voor de tijd gebruiken we de Singleton Pattern. Hiermee zorgen we ervoor dat de toegang tot de tijd altijd via één object te laten gaan. Hierdoor is de tijd uniek.

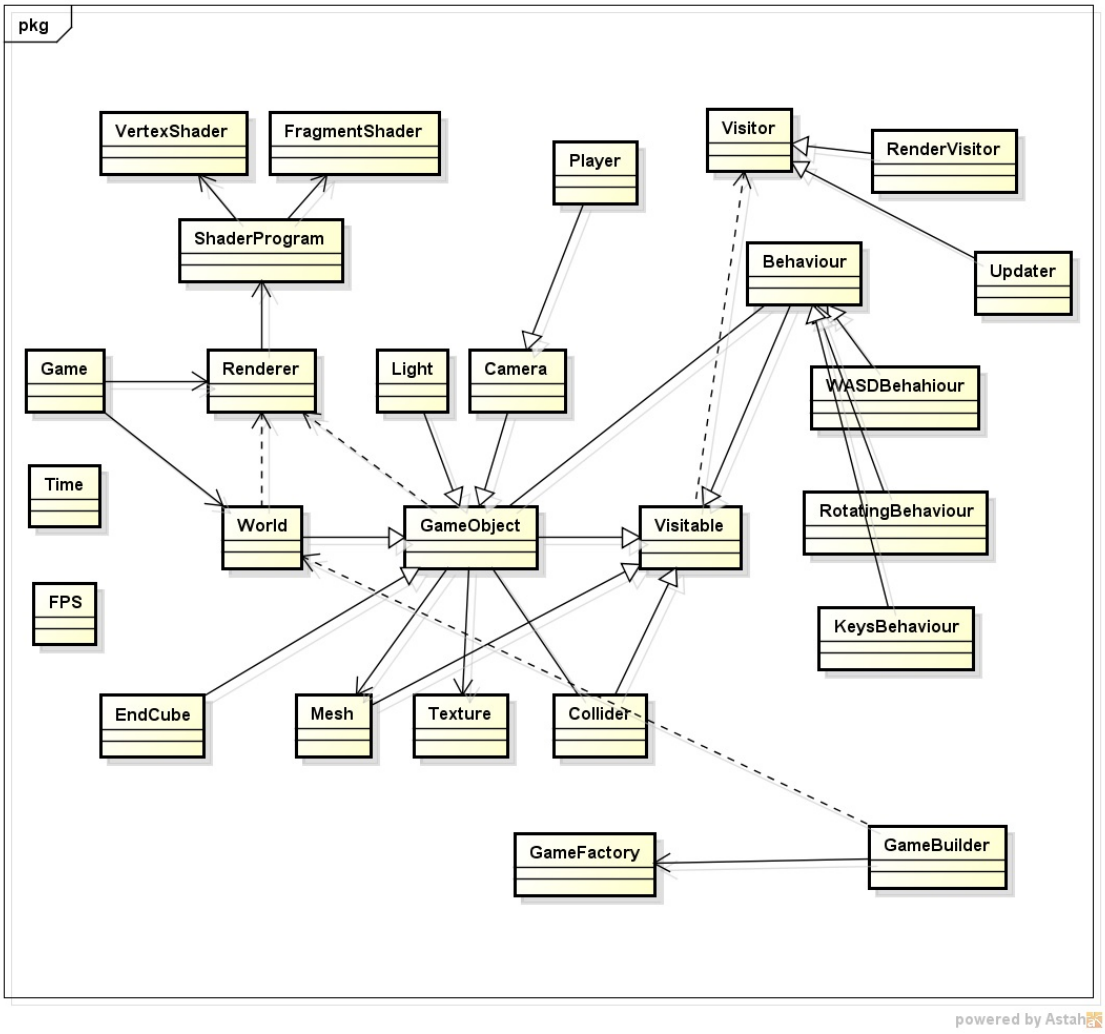
Composite

We gebruiken het composite pattern voor onze gameobjecten. Dit zorgt ervoor dat we eenvoudig gameobjecten kunnen nesten en geen onderscheid hoeven te maken tussen containers en “normale” objecten.

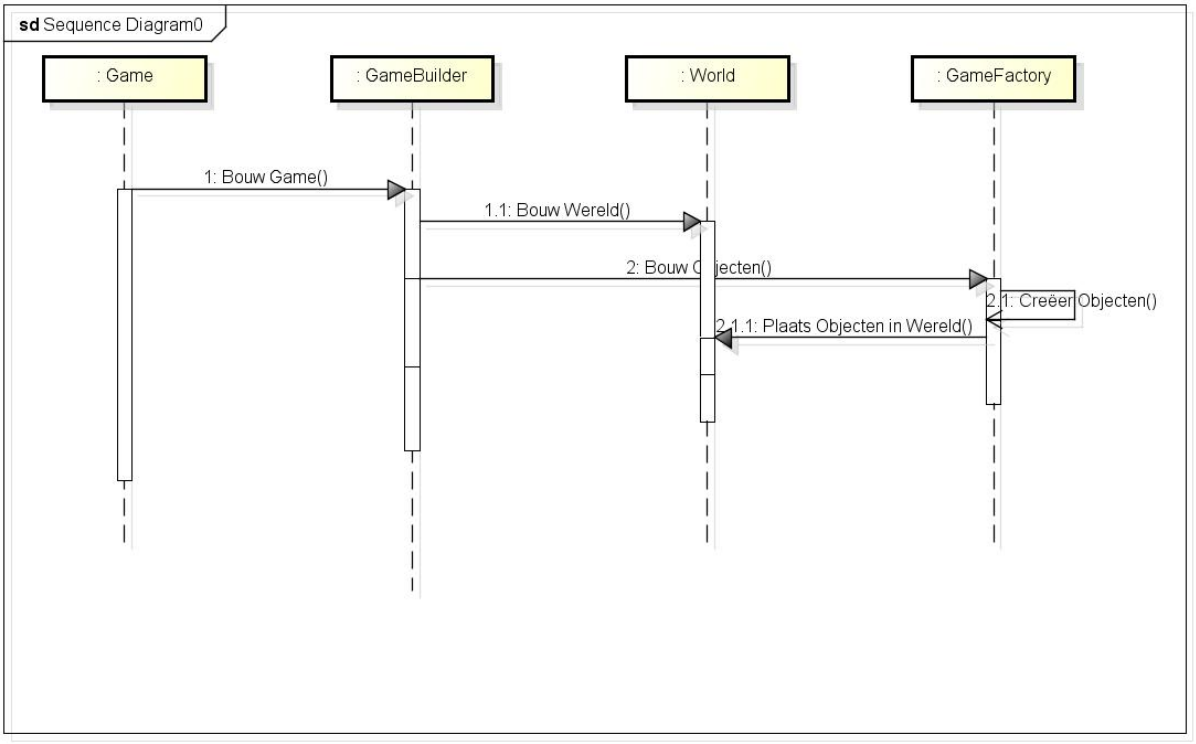
Ontwerp

Het ontwerp is vooral gewijzigd ten op zichte van de eerdere MicroGameEngine door het gebruik van bovenstaande design patterns. Verder hebben we een aantal wijzigingen/toevoegingen gedaan specifiek voor deze game. De Player klasse, wat erft van de camera omdat onze view First Person is. En de EndCube klasse, omdat onze game een keer moet stoppen. Onze scene graph is opgedeeld in Rooms, omdat verschillende ruimtes een requirement was en omdat het een hele handige opdeling is.

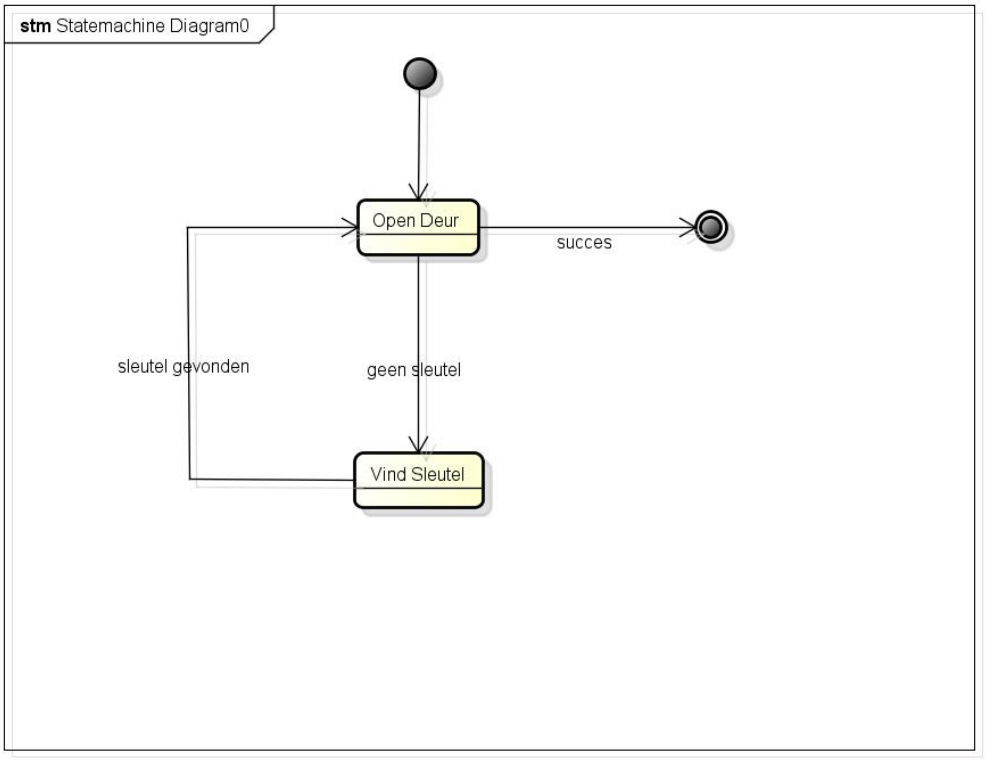
Class Diagram



Sequence Diagram



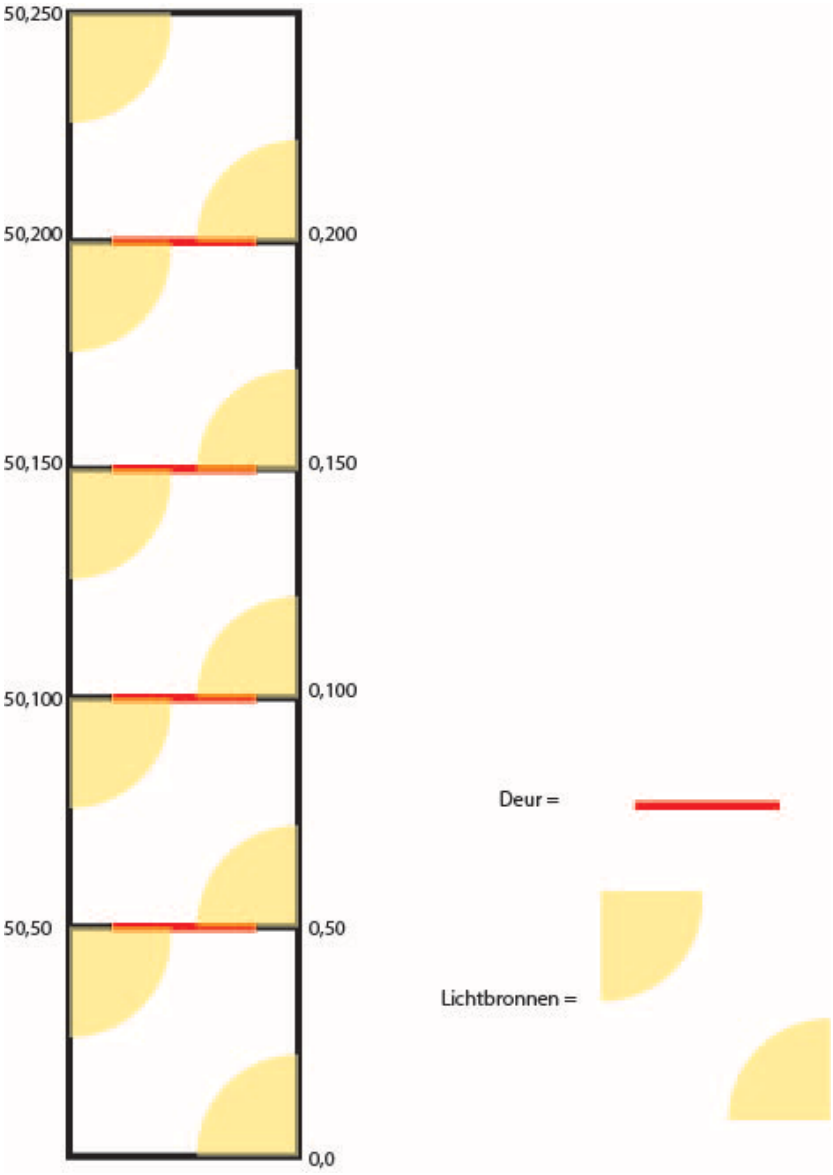
State Diagram



Scene Graph

- World
    - Ruimtes
        - Deur
            - ruimtes die de deur met elkaar verbindt
            - sleutel
        - Kubus
        - Kist
            - Sleutel
        - Speler
            - Sleutel(s)

Onze Scene Graph bestaat uit een World. In de World worden er ruimtes aangemaakt. Per ruimte worden deuren, kubussen en een kist aangemaakt. In de ruimte bevindt zich de speler. Zodra de speler de sleutel heeft gevonden kan het door de deur heen gaan en wordt de volgende ruimte met bijbehorende deuren, kubussen en een kist aangemaakt.



In het figuur hierboven zie je hoe we de wereld hebben opgebouwd:

- Aan het begin van de game wordt er een kamer gemaakt met 0,0 bij 50,50, hierbij zijn de coordinaten bij ons van rechts naar links.
- In de kamer zitten twee lichtbronnen diagonaal van elkaar, zodat alle kanten van de kamer belicht wordt.
- In de kamer zit een deur waar je alleen doorheen kan als je de sleutel in de kist hebt gevonden.
- Zodra je door de deur heen bent, verdwijnt de oude kamer en zit je vast in de nieuwe kamer.
- De nieuwe kamer is weer opgebouwd zoals in de vorige keer.
- In de laatste kamer hebben we de textures veranderd naar zwart.

HUD

De HUD laat de volgende teksten zien:

- De tijd
    - We gebruiken de tijd op dezelfde manier als hoe we dat gedaan hebben met de Computer Graphics taak met behulp van SFML. De tijd loopt meteen zodra het spel is opgestart. De tijd wordt ook gebruikt als score. Hoe minder lang je erover hebt gedaan hoe beter je score is.
- Sleutel in bezit
    - Via SFML tonen we heel simpel of de speler een sleutel in bezit heeft met een get-methode.
- Hints
    - De eerste hint in het spel is “hardcoded” erin gezet, dit in verband met het gebruik van \n. Overige hints/berichten kunnen via level.txt erin gezet worden als parameter van room. Let wel op: om de volgende hint/bericht te zien, moet er wel op enter gedrukt worden, anders blijft het hangen op de laatste hint/bericht.
- FPS
    - Voor debug mogelijkheden, kunnen we de Frames Per Second tonen, niet te verwarren met First Person Shooter ;)
- Huidige locatie van de speler
    - Voor debug mogelijkheden tonen we de locatie van de speler.

Specifieke oplossingen

Bouwen van de doolhoven

De doolhoven worden opgebouwd door kubussen. De ruimte bestaat uit 50x50. Door kubussen te maken van 10x10 zorgen we ervoor dat de speler sowieso nergens overheen kan kijken. Echter wordt de uitdaging om een doolhof te maken beperkten dan wanneer we gebruik maken van kleinere kubussen, maar hoe kleiner de kubus, hoe makkelijker de speler er overheen kan kijken.

Gebruik van object-files

Voor de deuren, kubussen en kisten maken we gebruik van object-files. Deze objecten scalen we naar behoefte. Hiervoor hebben we de load methode van Mesh aangepast om een vector mee te geven met de schaling x,y,z waardes.

Builder

Dankzij de Builder klasse kunnen we een txt-bestand inladen om snel en makkelijk objecten en tekst in de game te laden. De waarden moeten wel precies goed zijn, anders crasht het spel. Voor het creëren van de doolhoven wordt gebruik gemaakt van kubussen, zoals hierboven al beschreven is. Doordat je de coordinaten kent van de ruimte en de kubussen kun je deze zo creëren dat je er een leuke doolhof van kan maken. Verder zorgt het ervoor dat je ook meerdere ruimtes kan maken met bijbehorende onderdelen.

Problemen

Bouwen van de doolhoven

We hebben gemerkt dat bij het opbouwen van de doolhoven je de kubussen niet aan elkaar moet laten grenzen als de kubussen in twee verschillende ruimtes zitten. Het systeem denkt dan dat de kubussen in de niet-actieve ruimte erbij hoort in de actieve ruimte. Als je dan van actieve ruimte ruilt, dan zie je de kubus niet meer die er hoort te zijn. Dit probleem is op te lossen door +1 te geven aan de lokatie van de kubus voor de andere ruimte.

Een ander probleem is hierboven ook aangegeven: hoe kleiner de kubus, hoe makkelijker de speler er overheen kan kijken.

Collisions maar één keer behandelen

Collisions met een deur en een kist moeten maar één keer een event opleveren. Om dat op te lossen hebben de deur en de kist een boolean gekregen om de state bij te houden. Alleen bij deur was dit niet genoeg, je kunt immers door een deur proberen te gaan terwijl je de sleutel niet hebt. Dit hebben we opgelost door nog extra state bij te houden in de collission handler. Nadeel hiervan is dat je bij 2 deuren na elkaar geen event meer krijgt, maar dit komt niet heel snel voor.

Collision met de muren/kubussen

Uiteraard met de code die we hebben, kunnen we alleen de collisions registreren, maar we kunnen er nog niet voor zorgen dat de spelers dan ook niet daadwerkelijk door de muren heen kunnen lopen.

Textures

We weten niet hoe we de textures correct op bijvoorbeeld de deur kunnen. We hebben hier verder ook niet meer naar kunnen kijken vanwege tijdnood.

Suggesties

De game kan redelijk makkelijk uitgebreid worden met meer objecten. Ook zou het leuk zijn om een spring functie voor de speler te hebben, zodat het over bepaalde lage kubussen heen kan springen. Dit biedt nieuwe dynamische mogelijkheden voor de game. Uiteraard kan de game ook uitgebreid worden met meer kamers/levels. Variatie met deur-, en kamerlocaties maakt de game ook weer dynamischer.

Geluid

We hebben nog geluid geimplementeerd voor als je over de chest heen gaat en als je door de deur heen gaat.

Conclusie

De game kan op verschillende manieren worden geimplementeerd. Zelf hebben we gekozen, o.a. vanwege tijdnood, voor een “makkelijke” oplossing. Alle benodigde componenten hebben we ingebouwd om te zorgen dat we een werkende game krijgen met de requirements die we hadden.

We hebben hiermee geleerd dat we op deze manier levels redelijk eenvoudig kunnen opbouwen, nadat de implementatie (d.m.v. de verschillende design patterns) er staat.

Persoonlijke reflectie

Emiel

Ik vond het leuk om al die verschillende design patterns te leren en te gebruiken. Het maakt het gebruik van de engine veel makkelijker. Veel makkelijker uit te breiden zonder bestaande code aan te passen.

We hebben het werk aan het begin verdeeld ik heb designpatterns 1, 4, 5 (RenderVisitor, GameBuilder, GameFactory) gedaan Robbert 2, 3 en 6 (UpdateVisitor, CollisionVisitor, Singleton). Omdat GameBuilder en GameFactory samenwerken leek ons dit het handigst. Echter was dit wel iets meer werk voor mij. De verdere verdeling hebben we meer samengewerkt en gekeken wie wat wilde doen.

De samenwerking ging niet echt geweldig doordat we lange pauzes tussendoor hadden waarin we met andere dingen bezig waren. We moeten nu nog tot de deadline eraan werken, dit had anders niet gehoeven. Maar ik ben wel tevreden met het resultaat, het is nog een leuk spelletje geworden en de code erachter is ook “mooi” door het gebruik van de design patterns.

Robbert

Het lastige gedeelte vond ik het maken van de Design Patterns. Op papier ziet het er allemaal logisch uit, maar implementeren is lastiger. Ik heb veel kunnen leren door hoe Emiel dat heeft gedaan met de GameBuilder en GameFactory.

We hebben gemerkt dat het toch lastiger is om deze game te maken, dan bijvoorbeeld met de game voor Computer Graphics. Met de game van CG hadden we makkelijker taken kunnen verdelen en aan het einde alles samenvoegen. Met deze game voor Scene Graph & Patterns is het wat lastiger, omdat we beter op elkaar moeten afstemmen en er meer en lastigere taken zijn. Hierdoor was het handiger geweest dat we de belangrijkste onderdelen samen gingen doen, wat we ook gedaan hebben.

Afspreken ging moeizaam de laatste tijd, waardoor er grote gaten zaten in de tijd waarin we het werk konden doen. We hadden bijvoorbeeld het ontwerp al een tijdje af en het werk verdeeld, maar doordat we beide in de tussentijd andere dingen deden, vergaten we bijvoorbeeld ook alweer wat er nog gedaan moest worden. Ik moest mezelf en Emiel vaak aansporen om weer verder te gaan met deze taak, anders hadden we het nog steeds niet af gekregen.

Uiteindelijk hebben we de laatste paar dagen veel vooruitgang geboekt en hebben we zelfs een mooi werkend spel! Ik vind dit spel zelfs nog leuker dan het spel die we gemaakt hebben voor Computer Graphics :) We zijn er ook achter gekomen, dat met goede design pattern implementaties en een scene graph, je de rest van het spel best redelijk eenvoudig kan bouwen.