Documentatie Doolhof v1.0

Gemaakt door: Emiel de Brouwer & Robbert Goey

# test

# 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.

# 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)
  + 2 andere Patterns. (Observer en Composite)
  + Omgeving opgebouwd met een Scene Graph
  + Oberserver Pattern voor geluid
  + 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 collissions. Dit zorgt ervoor dat alle render, update en collission 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

bla

## Factory

Bla.

## Singleton

bla

## Observer

bla

## Composite

bla

# Ontwerp

Toelichting

## Class Diagram

plaatje

## Sequence Diagram

plaatje

## State Diagram

Plaatje

## Interaction Diagram

plaatje

# Toelichting Patterns

Bla.

# Scene Graph

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

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.

# 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.

# 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.

# Suggesties

Bla.

# Conclusie

Bla.

# Persoonlijke reflectie

## Emiel

Bla.

## Robbert

Bla.