19-9-2017

Gijs, Marouan, Kevin, Remar, Herman, Kelvin

S31T

Onderzoeksrapport

GridScape Online

Inhoudsopgave

[Samenvatting 3](#_Toc496617550)

[Inleiding 4](#_Toc496617551)

[Onderzoeksvragen 5](#_Toc496617552)

[Hoofdvraag 5](#_Toc496617553)

[Hoe maken we een gedistribueerde spel in java? 5](#_Toc496617554)

[Deelvragen 5](#_Toc496617555)

[Wat voor technieken zijn er voor het distribueerd maken van de applicatie? 5](#_Toc496617556)

[Welk game framework kunnen we gebruiken voor onze applicatie? 5](#_Toc496617557)

[Hoe maken we het spel gebruiksvriendelijk? 5](#_Toc496617558)

[Hoe kunnen we het beste informatie opslaan zoals een map? 5](#_Toc496617559)

[Onderzoeksaanpak 6](#_Toc496617560)

[Bieb 6](#_Toc496617561)

[Lab 6](#_Toc496617562)

[Showroom 6](#_Toc496617563)

[Onderzoeksresultaten 7](#_Toc496617564)

[Wat voor technieken zijn er voor het distribueerd maken van de applicatie? 7](#_Toc496617565)

[RMI 7](#_Toc496617566)

[Sockets 10](#_Toc496617567)

[Welk game framework kunnen we gebruiken voor onze applicatie? 12](#_Toc496617568)

[Hoe maken we het spel gebruiksvriendelijk? 13](#_Toc496617569)

[Hoe kunnen we het beste informatie opslaan zoals een map? 15](#_Toc496617570)

[Conclusies 17](#_Toc496617571)

[Wat voor technieken zijn er voor het distribueerd maken van de applicatie? 17](#_Toc496617572)

[Welke game framework kunnen we gebruiken voor onze applicatie? 17](#_Toc496617573)

[Hoe maken we het spel gebruiksvriendelijk? 17](#_Toc496617574)

[Hoe kunnen we het beste informatie opslaan zoals een map? 17](#_Toc496617575)

[Literatuurlijst 18](#_Toc496617576)

[Bronnen Sockets 18](#_Toc496617577)

[Bronnen Libraries 18](#_Toc496617578)

[Bronnen Mappen 18](#_Toc496617579)

# Samenvatting

In dit document bevindt zich de onderzoeksresultaten van de onderzoeken die zijn gepleegd. Met onze hoofdvraag proberen we te beantwoorden hoe we multiplayer gaan implementeren in onze game.

# 

# Inleiding

Voor ons proftaak project heeft de opdrachtgever de opdracht gegeven om een gedistribueerde applicatie te maken. We hebben al besloten om voor deze opdracht een spel te maken met multiplayer mogelijkheden. Om dit te realiseren zijn er veel mogelijkheden en technieken beschikbaar. Om te onderzoeken wat het beste is om te gebruiken gaan we een grootschalig onderzoek uitvoeren.

# Onderzoeksvragen

## Hoofdvraag

### Hoe maken we een gedistribueerde spel in java?

Om een multiplayer spel te maken is het natuurlijk nodig voor de applicatie om met elkaar te communiceren. Hier zijn verschillende mogelijkheden voor en het is nodig om deze nader te onderzoeken. Ook is het belangrijk om te onderzoeken hoe we de applicatie zelf maken en wat voor technieken we kunnen gebruiken om dit gebruiksvriendelijk te doen.

## Deelvragen

### Wat voor technieken zijn er voor het distribueerd maken van de applicatie?

Het spel zal uiteindelijk meerdere applicaties met elkaar verbinden. Er zijn verschillende manieren om dit te doen.

### Welk game framework kunnen we gebruiken voor onze applicatie?

Om een spel te maken kan het veel tijd besparen om een al bestaande game framework te gebruiken. Gelukkig bestaan er al veel game frameworks voor java, maar welke kunnen we het beste gebruiken?

### Hoe maken we het spel gebruiksvriendelijk?

Het is belangrijk dat het spel gemakkelijk door de gebruiker te gebruiken is, hij moet bijvoorbeeld zonder al te veel technische kennis kunnen connecten met een andere speler.

### Hoe kunnen we het beste informatie opslaan zoals een map?

In ons spel kunnen spelers kiezen uit verschillende maps. Uiteindelijk is het de bedoeling dat spelers zelf deze maps kunnen maken. Hoe zouden we dit het beste kunnen realiseren

# Onderzoeksaanpak

Om het onderzoek uit te voeren zijn er verschillende strategieën mogelijk. Hier gaan we uitleggen welke strategieën en methodes we in dit onderzoek gaan gebruiken.

## Bieb

Er is veel informatie beschikbaar op het internet, om hier relevante informatie uit te halen maken we gebruiken van een zoekmachine zoals Google. Als we een webpagina hebben gevonden met informatie is het ook belangrijk om de authenticiteit van deze informatie te verifiëren. We zullen meerdere bronnen onderzoeken en deze bronnen in ons onderzoek citeren. Een paar bronnen die we kunnen gebruiken zijn bijvoorbeeld Wikipedia, StackExchange en de Java documentatie.

## Lab

Met de lab strategie meten we de informatie die we met de andere strategieën hebben gevonden. We kunnen bijvoorbeeld testen hoe snel we een map kunnen opslaan of hoe snel een gebruiker het spel kan begrijpen. Verder kijken we naar de informatie van andere strategieën en die meten met elkaar.

## Showroom

Wanneer we uiteindelijk meerdere oplossingen hebben gevonden moeten we hieruit een oplossing kiezen. Om deze oplossingen met elkaar te vergelijken kunnen we bijvoorbeeld een prototype maken. Hiermee kunnen we de verschillende oplossingen beter tegen elkaar meten en de beste oplossing kiezen.

# Onderzoeksresultaten

## Wat voor technieken zijn er voor het distribueerd maken van de applicatie?

### RMI

#### Remote Method Invocation

RMI staat voor Remote Method Invocation. Dit is een manier om Java-objecten met elkaar te laten communiceren, dit kan gebruikt worden voor Java-objecten op verschillende computers. Oftewel een object dat bestaat in een JVM kan toegang krijgen tot een object dat op een andere JVM draait.  
RMI wordt daarom gebruikt voor gedistribueerde applicaties. De package “java.rmi” wordt hiervoor gebruikt.

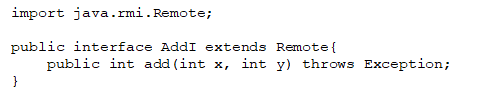
#### Stub en Skeleton

Voor RMI worden twee objecten gebruikt voor de communicatie tussen de applicaties: de “stub” en de “skeleton”. De stub en de skeleton worden gebruikt als de client en de server kant.   
De Stub fungeert als een gateway voor de client. Alle uitgaande verzoeken worden hier doorheen gestuurd. Wanneer een methode op het stub object wordt aangeroepen zal er een verbinding worden gemaakt met de JVM. Vervolgens schrijft en verzendt deze de meegegeven parameters naar de JVM, wacht het op een resultaat, leest de retourwaarde (of de exceptie als deze meegegeven wordt) en geeft de waarde terug.   
De skeleton fungeert als een gateway voor de server. In plaats van de uitkomende verzoeken worden voor de server de inkomende verzoeken hier doorheen gestuurd. Deze leest de parameters van de methode, roept de methode vervolgens op en schrijft en verzendt het resultaat.  
Dit is simpelweg de communicatie tussen de client en de server en dus op deze manier kan er toegang verkregen worden over objecten over verschillende JVM’s.

#### Stappen voor het schrijven van een RMI programma

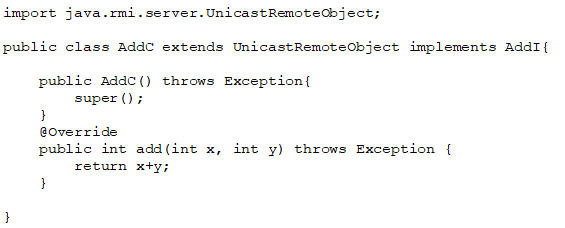
Er zijn zes stappen die gebruikt worden om een RMI applicatie te maken.

1. Het creëren van een remote interface

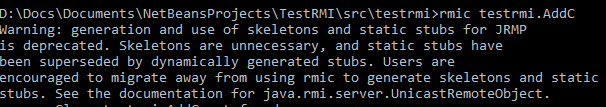
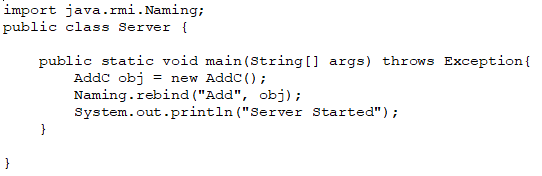
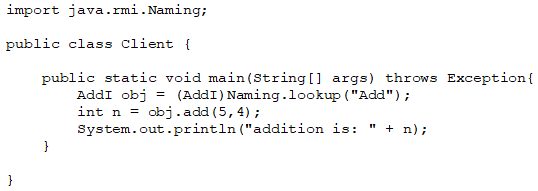


De remote interface extends Remote. Ieder object dat dient als een remote object moet een remote interface implementeren. Oftewel alleen een interface dat java.rmi.Remote extend zijn beschikbaar voor in dit geval een andere JVM. Zie het voorbeeld hierboven.

1. De implementatie van de remote interface



Hierbij wordt de extend van de UnicastRemoteObject gebruikt. Deze implementeert de remote interface. De UnicastRemoteObject wordt gebruikt voor het exporteren van het remote object en om de stub te verkrijgen om met het remote object te kunnen communiceren.

1. Het creëren van de stub en de skeleton.  
     
   Dit werd in eerste instantie op bovenstaande manier uitgevoerd, maar is nu niet meer nodig zolang er een poort beschikbaar is voor het exporteren van het remote object.
2. Start het registratie service.  
     
   Met rmiregistry wordt een remote object geregistreerd op een poort. Dit gebeurt op de achtergrond. Het is een “bootstrap naming-service” die de remote objecten binden aan een naam. De clients kunnen dan de remote objecten opzoeken en gebruik maken van remote method invocations.
3. Het creëren en uitvoeren van de server.  
     
   Een remote object wordt hier aangemaakt. En vervolgens wordt met rebind() een naam verbonden aan dit object.
4. Het creëren en uitvoeren van de client.  
     
    In dit geval kan er niet direct een object worden aangemaakt van de AddC klasse (die we in bovenstaande voorbeelden hebben gemaakt). Dit komt omdat deze hoort bij de server kant. De interface kan wel gebruikt worden. Hiervoor vragen we de RMI om de naam te geven die we hier aan gegeven hebben bij stap vijf (in het voorbeeld de naam “Add”). Typecasting is nodig omdat we een interface gebruiken en geen klasse. Vervolgens worden parameters aan de methode add() meegegeven.

Nu kan de server worden gestart:

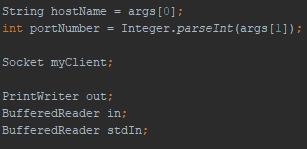


En vervolgens de client:

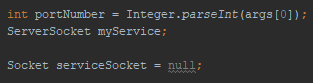


De methode die in eerste instantie onbekend was voor de client wordt nu uitgevoerd met de parameters 4 en 5 en correct uitgevoerd.

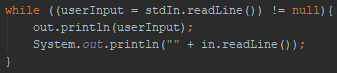
### Sockets

Om een gedistribueerde applicatie te maken voor de proftaak zijn er verschillende mogelijkheden. Een manier om computers met elkaar te verbinden is sockets. Als je sockets wil gebruiken heb je een client en een server nodig. De client heeft de hostnaam en poort nummer waar de server op luistert. Ook heb je input en output nodig, in dit voorbeeld gebruik ik Printwriter en BufferedReader maar je kan ook andere methodes gebruiken. 

De server heeft naast de input en output een serversocket, deze luistert of een client een socket wil openen en vult dan een socket.



Wanneer er een connectie is gemaakt is het mogelijk om de input en output te initializeren. Dit kan door de input en output stream van de socket te gebruiken. Dit is zowel in de client als in de server hetzelfde. 

Daarna is het mogelijk om in de client of server een string in de output te stoppen en daarvoor in de input naar te luisteren!

De vier stappen die de client dus moet doorgaan zijn.

1. Open een socket
2. Open de input en output stream naar de socket.
3. Lees en schrijf naar de socket volgens de protocol van de server.
4. Sluit de sockets.

#### Datagram / Stream communication

Als je sockets gebruikt zijn er twee soort methodes om deze te gebruiken, Datagram communication (UDP) en Stream communication (TCP). In het voorbeeld wordt een TCP connectie gebruikt. Door deze methode te gebruiken weet je zeker dat de packets aankomen. Terwijl je bij UDP dat niet zeker weet.

#### Voordelen en Nadelen van Sockets

Het gebruik van sockets is het laagste level dat je kan programmeren in java. Dit betekent dat je complete controle hebt over de data die je verstuurt. Maar deze moet je zelf dus ook afhandelen. Hiermee kun je dus met elke applicatie praten in tegenstelling tot RMI, wat alleen met andere java applicaties werkt. Sockets zijn over het algemeen ook sneller dan RMI.

## 

## 

## Welk game framework kunnen we gebruiken voor onze applicatie?

Er zijn veel verschillende game frameworks en libraries beschikbaar voor Java. Een aantal populaire zijn:

* LibGDX
* Ligthweight Java Game Library 3 (LWJGL3)
* jMonkeyEngine

Dit zijn verreweg de beste libraries beschikbaar voor Java. Maar omdat wij van plan zijn een 2D game te maken, valt jMonkeyEngine af. jMonkeyEngine is speciaal gemaakt voor 3D games en is voor ons dus niet nuttig. LibGDX en JWJGL3 blijven dus over.

Beide libraries supporten OpenGL en 2D graphics, precies wat we nodig hebben. Het verschil tussen de twee is dat LibGDX gebaseerd is op LWJGL3.

LWJGL3 is een low-level framework. Je kan er veel mee leren over OpenGL. Maar laat de naam je niet voor de gek houden. Hij is misschien wel “Lightweight”, het is een hele moeilijke framework om te leren. LibGDX doet al het moeilijke werk dat je met LWJGL3 moet doen voor je.

Omdat wij beginnend programmeurs zijn als het gaat om Java en ook niet veel ervaring hebben met OpenGL en game frameworks, is het voor ons het beste om met de simpele route te gaan en LibGDX te kiezen.

LibGDX Homepage: <https://libgdx.badlogicgames.com/index.html>

## 

## 

## Hoe maken we het spel gebruiksvriendelijk?

We kunnen dit opdelen in 2 delen, namelijk: UI en Functionaliteit.

Eerst kijken we naar UI, hiervoor gebruiken wij het boek Designed for Use geschreven door Lukas Mathis (uitvoering van 2011). Uit dit boek kunnen wij de volgende info halen:

Gebruik korte duidelijke maar informele tekst.

Hou de UI simpel, niet teveel info op 1 scherm.

We kunnen bij eventuele stukken tekst ook een leesbaarheids test doen.

Een leesbare font.

De font grootte moet net groter zijn dan een gemiddeld leesboek.

De schermen lopen in logische volgorde (de gebruiker heeft hier vaak al een verwachting bij)

Veld onderzoek voor het UI, we kijken hierbij hoe bij vergelijkbare games het UI eruit ziet:

Battle for Wesnoth



Duidelijke tekst, grote knoppen. Voor de hand liggende statistieken (stats die worden verwacht door spelers van dit type spel).

Fire Emblem (de game waar onze game op gebasserd is)



Overzichtelijke UI, de lettergrote, de indeling en de kleuren maken het zeer leesbaar.

Nu kijken we naar functionaliteit. Hiervoor gebruiken we de boeken Designed for Use (2011) en Multiplayer Game Types (2015), en verschillende game communities op internet.

Lijst met gebruiksvriendelijke functionaliteiten:

Scheldwoorden censureren in de chat.

Specifieke gebruikers kunnen blokkeren uit de chat.

Anti cheat systeem.

Het makkelijk maken om een game te starten.

## 

## 

## Hoe kunnen we het beste informatie opslaan zoals een map?

Het is belangrijk dat we de mappen die we gebruiken makkelijk kunnen maken en aanpassen. Een populaire techniek bij het creëren van mappen in een 2d game zijn Tile Maps. Tile Maps gebruiken een spritesheet waar alle tiles in een afbeelding zitten. Wanneer er een tegel getekend moet worden word er alleen een deel van de sprite sheet getekend.

Om de mappen makkelijk te kunnen aanpassen kunnen we een tilemap editor gebruiken.

In mijn onderzoek ben ik op een tilemap editor gestuit genaamd Tiled. Tiled support vierkante en isometrische tiles. Tiled kan mappen exporteren als TMX (Tile Map XML) en JSON. De mappen zouden we via serializable in de database kunnen zetten.

Het gegenereerde text bestand die ontstaat met het maken van een map met de map editor kunnen we opslaan in de database. :

import java.io.File;  
import java.io.FileInputStream;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
  
public class InsertTextFileToMySQL {  
  
 public static Connection getConnection() throws Exception {  
 String driver = "org.gjt.mm.mysql.Driver";  
 String url = "jdbc:mysql://localhost/databaseName";  
 String username = "root";  
 String password = "root";  
  
 Class.forName(driver);  
 Connection conn = DriverManager.getConnection(url, username, password);  
 return conn;  
 }  
  
 public static void main(String[] args)throws Exception {  
 String id = "001";  
 String fileName = "fileName.txt";  
   
 FileInputStream fis = null;  
 PreparedStatement pstmt = null;  
 Connection conn = null;  
 try {  
 conn = getConnection();  
 conn.setAutoCommit(false);  
 File file = new File(fileName);  
 fis = new FileInputStream(file);  
 pstmt = conn.prepareStatement("insert into DataFiles(id, fileName, fileBody) values (?, ?, ?)");  
 pstmt.setString(1, id);  
 pstmt.setString(2, fileName);  
 pstmt.setAsciiStream(3, fis, (int) file.length());  
 pstmt.executeUpdate();  
 conn.commit();  
 } catch (Exception e) {  
 System.err.println("Error: " + e.getMessage());  
 e.printStackTrace();  
 } finally {  
 pstmt.close();  
 fis.close();  
 conn.close();  
 }  
 }  
}

# Conclusies

## Wat voor technieken zijn er voor het distribueerd maken van de applicatie?

Sockets is voor ons de betere keuze. Dit omdat het sneller is dan RMI en met sockets kun je low-level programmeren waardoor je meer controle hebt in vergelijk met RMI. Verder zijn sockets ook te gebruiken met applicaties die niet geschreven zijn in java maar dat is niet relevant voor ons project.

## Welke game framework kunnen we gebruiken voor onze applicatie?

LWJGL3 en LibGDX zijn in principe hetzelfde, maar LibGDX pakt alles wat moeilijk is aan LWJGL3 en doet het voor je. Daarom kiezen wij voor LibGDX.

## Hoe maken we het spel gebruiksvriendelijk?

microtransactions

## Hoe kunnen we het beste informatie opslaan zoals een map?

De mappen opslaan als TMX omdat het makkelijk aanpasbaar is en de standaard is voor het programma Tiled. Verder is er ook een library die het makkelijk maakt om dit bestandstype te gebruiken.

# Literatuurlijst

## Bronnen Sockets

<https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>

<https://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html>

<https://www.thecodingforums.com/threads/rmi-vs-sockets-vs.515288/>

## Bronnen Libraries

<https://libgdx.badlogicgames.com/index.html>

<http://jmonkeyengine.org/>

<https://www.lwjgl.org/>

## Bronnen Mappen

<https://developer.mozilla.org/en-US/docs/Games/Techniques/Tilemaps>

<http://doc.mapeditor.org/en/latest/>

<https://github.com/bjorn/tiled/tree/master/util/java/libtiled-java>

<https://www.youtube.com/watch?v=rWzINXeC0lY>

<http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html>