GRUPPE 7: FRENCH CONNECTION

Das Handbuch

FS 2010

Gabriele Döhring
Alexander Haesen
Sven Mergner
Arthur Toenz

PROGRAMMIEREN II PROJEKT

Inhaltsverzeichnis

1. Installation ... 3
2. Spielbeschreibung ... 3
3. Ziel des Spiels ... 3
4. Steuerung ... 3
5. Die Figuren ... 4
4.1 Klassische Version ... 5
4.2 Studenten Version ... 6
6. Mögliche Aktionen ... 8
7. Der Kampf ... 8
8. Das Spiel starten ... 8
9. Verwendete Bibliotheken ... 13
10. Struktur des Programms ... 14
11. Software-Qualitätsmanagement (Testing) ... 15
12. Beispielsession ... 16

1. Installation

Das Programm wurde komplett in 2 Jar-Files gepackt. Man kann den Server also starten, indem man doppelt auf das Jar-File klickt und anschließend den Port grafisch über die GUI einstellt. Man kann den **Server** aber auch über die Konsole mit **java -jar Server.jar server <listenport>** starten. Die Clients können ebenfalls durch doppelklick auf das Jar-File gestartet werden. Die Einstellungen werden dann wieder grafisch über die GUI vorgenommen. Man kann die **Clients** aber auch über die Konsole mit **java -jar Client.jar client <serverip> <serverport>** starten, man muss jedoch anschließend auf "Send" klicken.

2. Spielbeschreibung

Bei Dungeon Twister handelt es sich um ein rundenbasiertes Server-Client-Spiel in Form eines Labyrinths für 2 bis 4 Spieler. Das Labyrinth besteht aus Hallen, die vor Spielbeginn zufällig angeordnet werden. Dabei besteht ein Spielfeld für 2 Spieler aus 8 Hallen und ein Spielfeld für 4 Spieler aus 16 Hallen. Es gibt insgesamt 16 verschiedene Spielfiguren, die in zwei Versionen unterteilt sind und zwar in die Klassische und die Studenten Version. Jede Version beinhaltet 8 verschieden aussehende Figuren. Jede von den 8 Figuren beherrscht eine andere Fähigkeit. Jeder Spieler wählt zu Beginn des Spiels eine Version aus und stellt dann ein Team aus 4 Spielfiguren zusammen. Dungeon Twister besitzt einen im Spiel integrierten Chat, der den Spielern ermöglicht vor und während dem Spiel Nachrichten auszutauschen.

3. Ziel des Spiels

Ziel des Spiels ist es 5 Siegespunkte (SP) zu erreichen. Der Spieler erhält einen Siegespunkt, wenn eine Figur erfolgreich das Ende des Labyrinths erreicht, oder wenn eine Figur des Spielers die Figur eines anderen Spielers im Kampf besiegt.

4. Steuerung

Um Dungeon Twister zu steuern muss nur mit der linken Maustaste auf die gewünschte Aktion geklickt werden.

Steuerung innerhalb des Spiels:

Figur in die entsprechende Richtung bewegen

Halle drehen

Bricht Zug des derzeitigen Spielers ab
Nächster Spieler ist am Zug

Beendet das Spiel


Speichert das Spiel


Zeigt die Statistik für das Spiel an


Durch eine Wand gehen


Feuerballstab benutzen


Ein Gitter zerstören


Figur heilen


Gitter aufschließen


Gitter zuschließen


Regenerieren


Labyrinth mit einer Figur verlassen


Kämpfen

In der Lobby:


Credits


Highscore

5. Die Figuren

Die Figuren können unterschiedlich viele Felder laufen, zudem sind die Figuren unterschiedlich stark (Kampfkraft) und sie haben unterschiedliche Fähigkeiten. Alle Figuren können die Halle um 90° drehen, wenn sie auf einer Drehplattform stehen. Dabei wird die vorgegebene Pfeilrichtung auf der Plattform eingehalten. Die Halle mit der gleichen Zahl bzw. Farbe dreht sich automatisch mit.

4.1 Klassische Version

Kleriker – Priest

Maximale Anzahl Felder: 4
Kampfkraft: 2

Der Kleriker hat die Fähigkeit andere Spielfiguren zu heilen, wenn sie verletzt sind. Sich selbst kann der Kleriker allerdings nicht heilen.

Goblin

Maximale Anzahl Felder: 4
Kampfkraft: 1

Der Goblin hat keine spezielle Fähigkeit, jedoch wenn es dem Spieler gelingt den Goblin unversehrt aus dem Labyrinth zu bringen, dann bekommt der Spieler 2 Siegespunkte.

Krieger – Warrior

Maximale Anzahl Felder: 3
Kampfkraft: 3

Der Krieger kann die im Labyrinth befindlichen Gitter zerstören, wodurch die anderen Figuren ungehindert ihren Weg durch das Labyrinth fortsetzen können.

Magier – Magician

Maximale Anzahl Felder: 4
Kampfkraft: 1

Der Magier besitzt die Fähigkeit über Fallgruben und feindliche Figuren zu schweben. Allerdings kann er weder auf einer Fallgrube noch auf einer feindlichen Figur stehen bleiben. Der Magier ist die einzige Figur, die den Feuerballstab benutzen kann.

Mechanork

Maximale Anzahl Felder: 3
Kampfkraft: 2

Der Mechanork kann, wie die anderen Figuren auch, eine Halle um 90° drehen, wenn er auf einer Drehplattform steht, allerdings beachtet er dabei nicht die vorgegebene Richtung. D.h. der Spieler kann auswählen in welche Richtung die Halle gedreht werden soll.

Wandläuferin – Acrobate

Maximale Anzahl Felder: 4
Kampfkraft: 1

Die Wandläuferin kann, wie der Name schon sagt, durch Wände laufen. Allerdings kann sie nicht durch geschlossene Gitter hindurch.

Troll

Maximale Anzahl Felder: 2
Kampfkraft: 4

Der Troll kann sich regenerieren, d.h. er kann sich selbst heilen. Allerdings nicht in der Runde, in der er verletzt wurde. In der Runde in der der Troll regeneriert kann er keine andere Aktion mehr ausführen. Wird der Troll von einer Feuerkugel getroffen, dann wird er sofort zerstört.

Diebin – Burglar

Maximale Anzahl Felder: 5
Kampfkraft: 2

Die Diebin kann über Gruben laufen und auf ihnen stehen bleiben. Solange sie auf einer Grube steht, können Figuren ihres Teams über sie hinweg laufen. Wird die Diebin auf einer Grube verletzt, dann verliert sie diese Fähigkeit, stürzt ab und stirbt. Die Diebin kann das Schloss eines Gitters knacken und ein offenes Gitter wieder zuschließen, solange es nicht vom Krieger zerstört wurde.

4.2 Studenten Version

Medizin Studentin – Medic (= Kleriker)

Maximale Anzahl Felder: 4
Kampfkraft: 2

Die Medizin Studentin hat es sich zur Aufgabe gemacht anderen Menschen zu helfen. Während ihres Studiums hat sie gelernt wie man die Verletzungen anderer Menschen heilen kann und diese Fähigkeit macht sie sich im Labyrinth zu nutze. Allerdings kann sie ihre eigenen Verletzungen nicht versorgen.

Geschichtsstudent - History Student (= Goblin)

Maximale Anzahl Felder: 4
Kampfkraft: 1

Der Geschichtsstudent interessiert sich nicht für das Geschehen um ihn herum. Er steckt mit der Nase nur in seinen Büchern und schwelgt in der Vergangenheit. Er ist somit keine große Hilfe für sein Team.

Sport Student – Sports Student (= Krieger)

Maximale Anzahl Felder: 3
Kampfkraft: 3

Da der Sport Student während seines Studiums verschiedene Kampfsportarten, wie zum Beispiel Kickboxen, erlernt hat, kann er die im Labyrinth befindlichen Gitter zerstören und so seinem Team neue Wege eröffnen.

Nano Student – Nano Student (= Magier)

Maximale Anzahl Felder: 4
Kampfkraft: 1

Der Nano Student hat während seines Studiums kleine Nanoroboter entwickelt, die es ihm ermöglichen für kurze Zeit zu schweben. Im Labyrinth benutzt er die Nanoroboter um über Fallgruben und Gegner zu schweben. Außerdem kann er den Feuerballstab benutzen.

Informatik Student – Computational Science Student (= Mechanork)

Maximale Anzahl Felder: 3
Kampfkraft: 2

Der Informatik Student hat während seines Informatik Studiums in der Programmieren II Vorlesung ein Programm geschrieben, womit er die Halle des Labyrinths in eine beliebige Richtung drehen kann. Er muss also nicht auf die vorgegebene Pfeilrichtung der Drehplattformen achten.

Physik Student - Physics Student (= Wandläuferin)

Maximale Anzahl Felder: 4
Kampfkraft: 1

Der Physik Student kann sich, dank einer Erfindung seines Professors, durch die Wände des Labyrinths beamen. Nur durch die Gitter des Labyrinths kommt er damit nicht.

Chemie Student – Chemistry Student (= Troll)

Maximale Anzahl Felder: 2
Kampfkraft: 4

Der Chemie Student hat während seines Studiums in einem Labor einen Trank entwickelt, mit dem er sich selbst heilen kann. Allerdings wirkt der Trank nicht sofort und benötigt deswegen eine Ruhephase von einer Runde.

Archäologie Studentin – Archeology Student (= Diebin)

Maximale Anzahl Felder: 5
Kampfkraft: 2

Die Archäologie Studentin nimmt sich Indiana Jones und Lara Croft als Vorbild. Sie kann die Gitter des Labyrinths mit dem ausgegrabenen Generalschlüssel öffnen und schließen. Sie kann sich über die Gruben des Labyrinths schwingen und über ihnen verharren, damit andere ihres Teams auf die andere Seite gelangen können.

6. Mögliche Aktionen

Der Spieler kann während einer Runde maximal 3 Aktionen ausführen:

- Eine der Figuren bewegen (Figuren können unterschiedlich viele Felder laufen)
- Eine Halle erkunden
- Eine Halle drehen
- Eine Spezialfähigkeit einer Figur nutzen
- Kämpfen

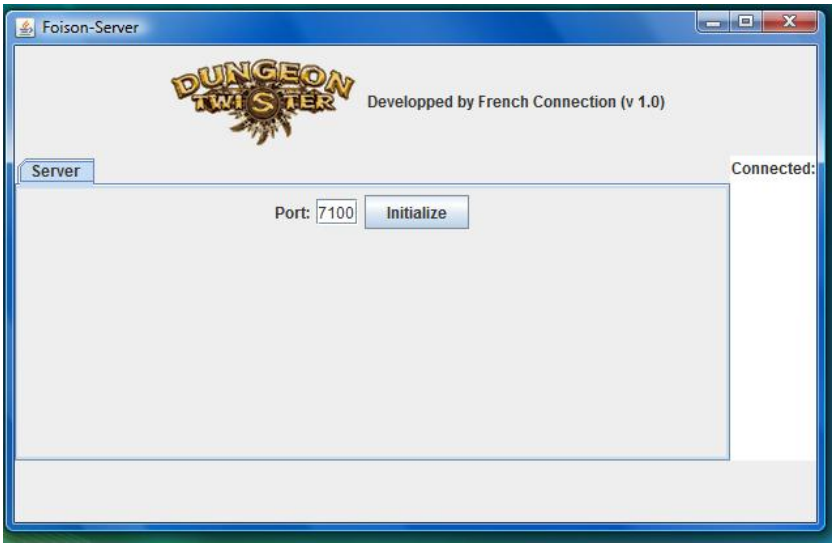
7. Der Kampf

Wenn sich zwei Figuren aus gegnerischen Teams gegenüber stehen kommt es zu einem Kampf. Dieser läuft dann folgendermaßen ab:

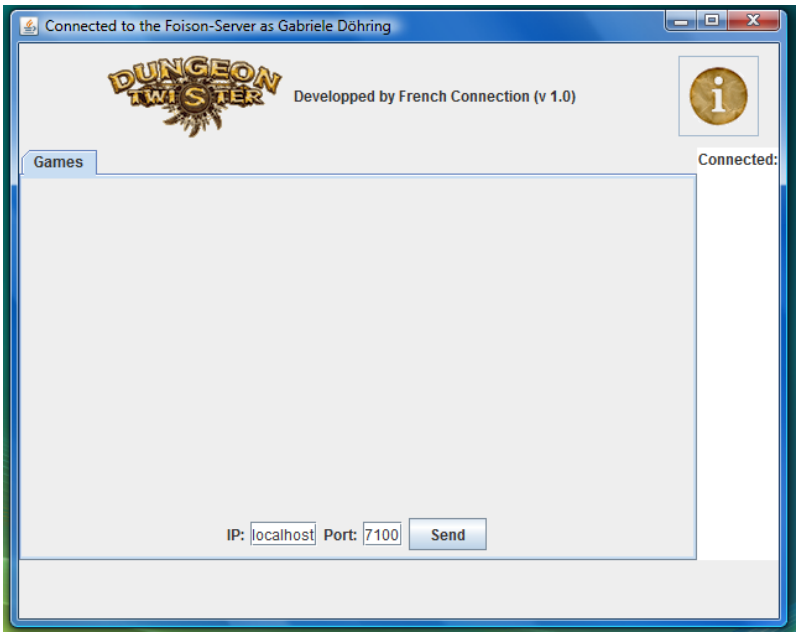
Jeder Spieler wählt eine der verschiedenen Kampfkarten aus. Die Punkte der Kampfkarten werden dann zu den Kampfpunkten der jeweiligen Figur hinzuaddiert und anschließend mit der Summe des anderen Spielers verglichen. Der Spieler mit der niedrigeren Punktzahl verliert den Kampf und der Figur werden Punkte abgezogen. Die benutzte Kampfkarte kann anschließend nicht mehr verwendet werden.

8. Das Spiel starten

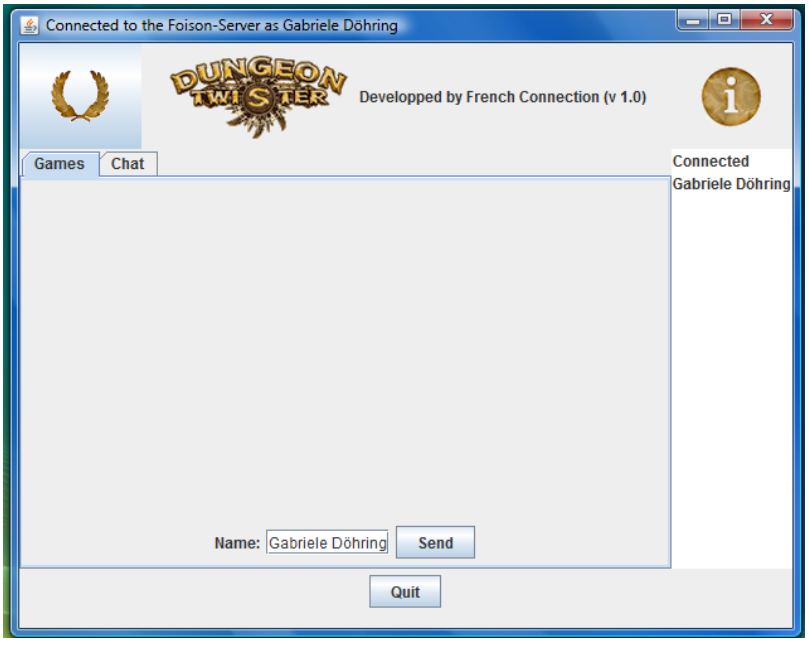
Zuerst muss der Server gestartet werden. Hierfür klickt man entweder doppelt auf das Jar-File und anschließend auf den Button „Initialize“, oder man startet den Server über die Konsole mit: java -jar Server.jar server 7100 (siehe Installation).



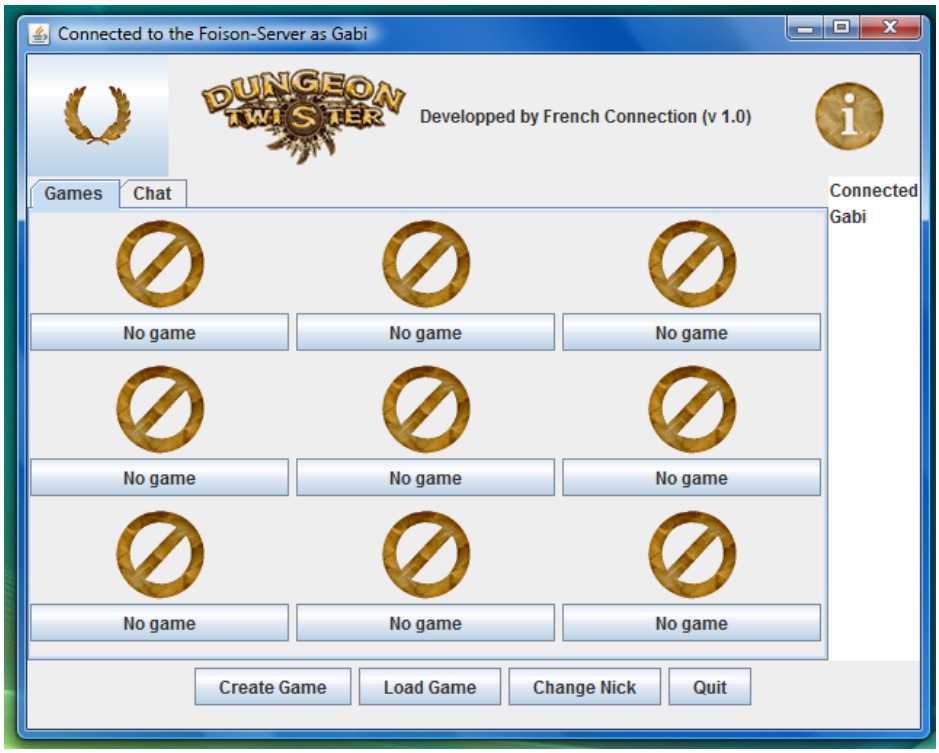
Danach wird der Client gestartet entweder ebenfalls durch doppelklick auf das Jar-File oder über die Konsole mit: java -jar Client.jar client localhost 7100 (siehe Installation. Anschließend wird auf den Button „Send“ geklickt.



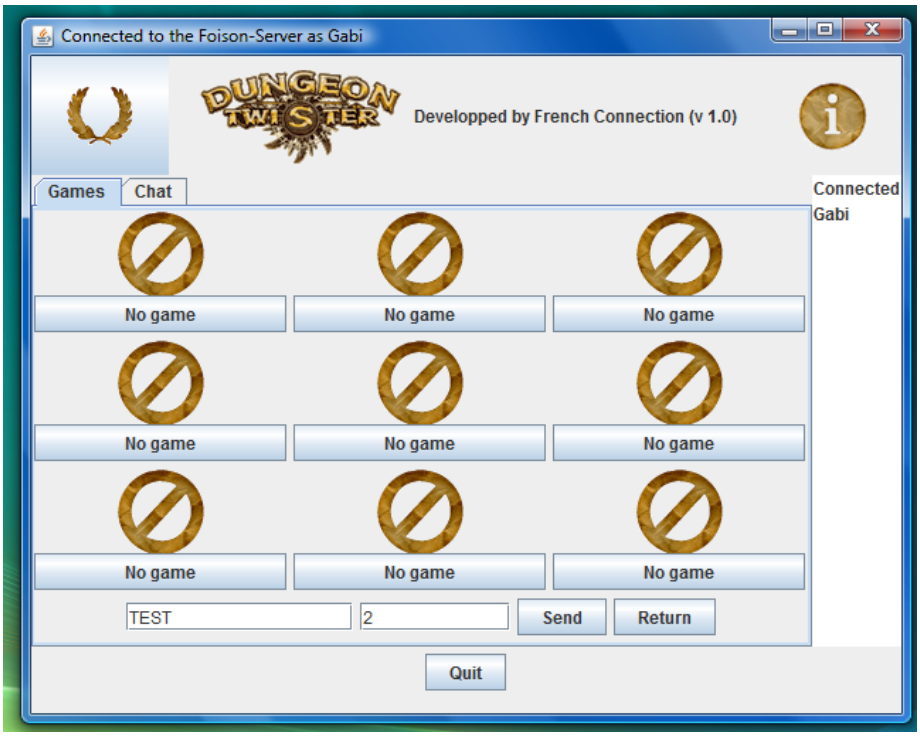
Nun wird dem Spieler ein Username vorgeschlagen, diesen kann er akzeptieren oder ändern. Danach wird wieder auf „Send“ geklickt.



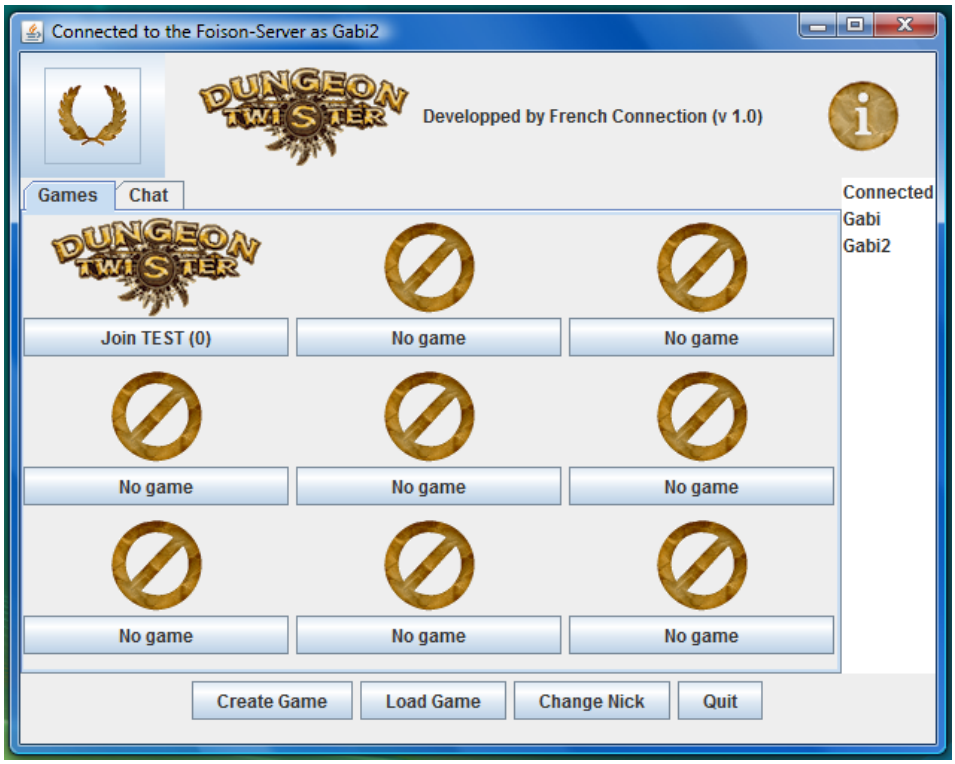
Anschließend kann der Spieler entweder ein neues Spiel öffnen(Button „Create Game“), ein Spiel laden (Button „Load Game“), seinen Nicknamen ändern (Button „Change Nick“) oder den Client wieder beenden (Button „Quit“). Durch Klicken auf die Registerkarte „Chat“ kann der Spieler im Lobby Chat mit anderen Spielern Nachrichten austauschen.



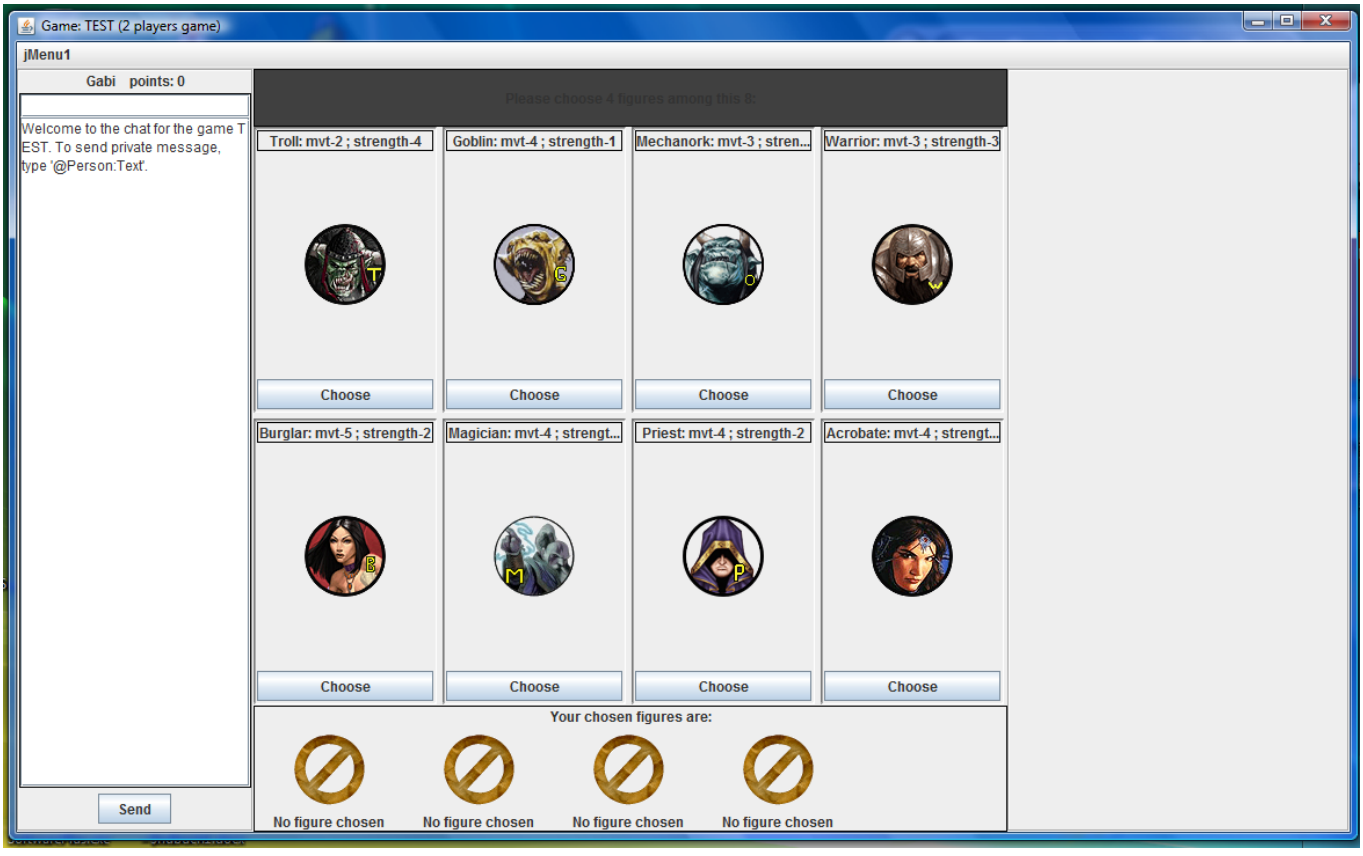
Wenn der Spieler ein neues Spiel eröffnen will, so muss der Spieler, nachdem er auf den „Create Game“ Button gedrückt hat, einen Namen für das Spiel (im Beispiel: „TEST“) und die Anzahl der Spieler eingeben (im Beispiel: „2“) und anschließend auf „Send“ klicken.

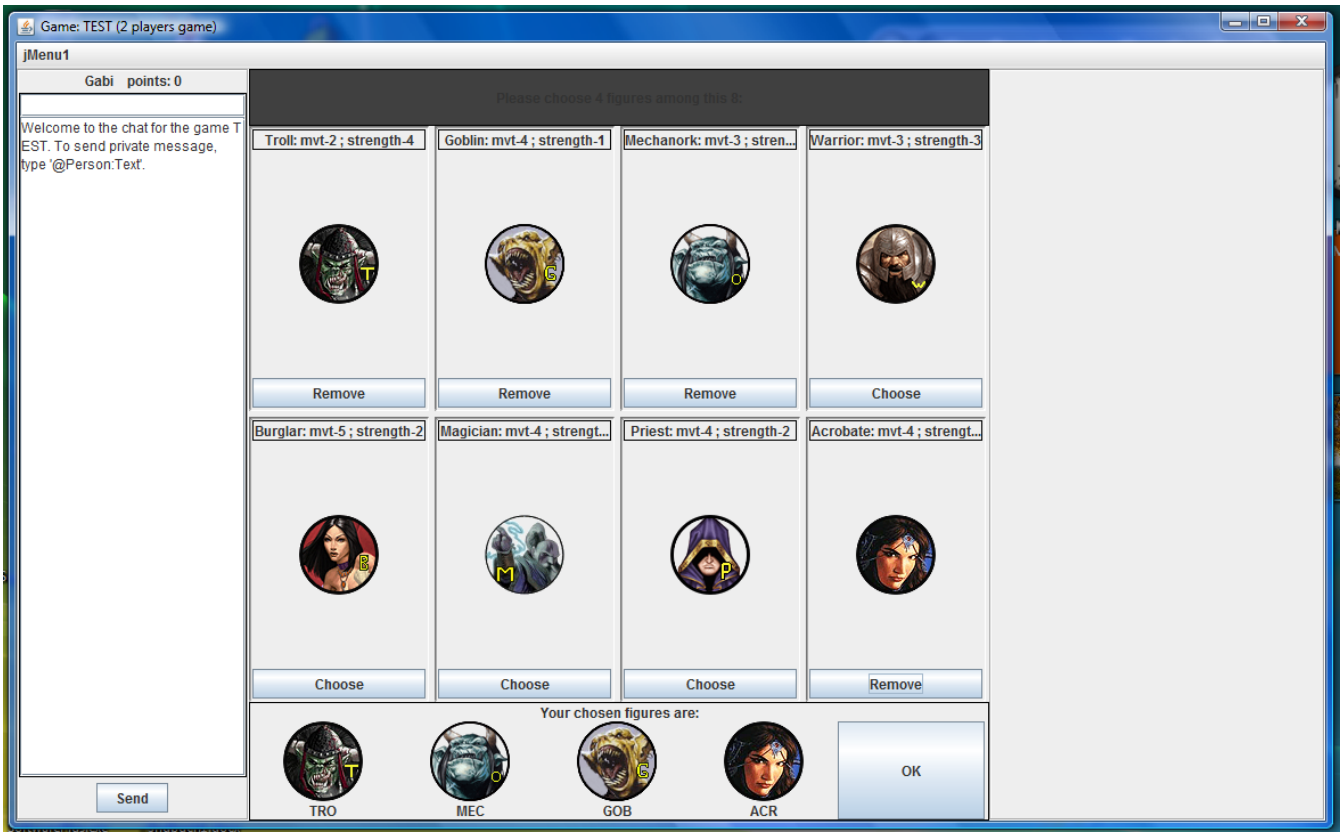


Danach wird gewartet, bis genügend Spieler dem Spiel beigetreten sind, bevor das Spiel gestartet wird (im Beispiel: neuer Spieler kam hinzu und tritt durch klicken auf den Button „Join TEST“ dem Spiel bei).

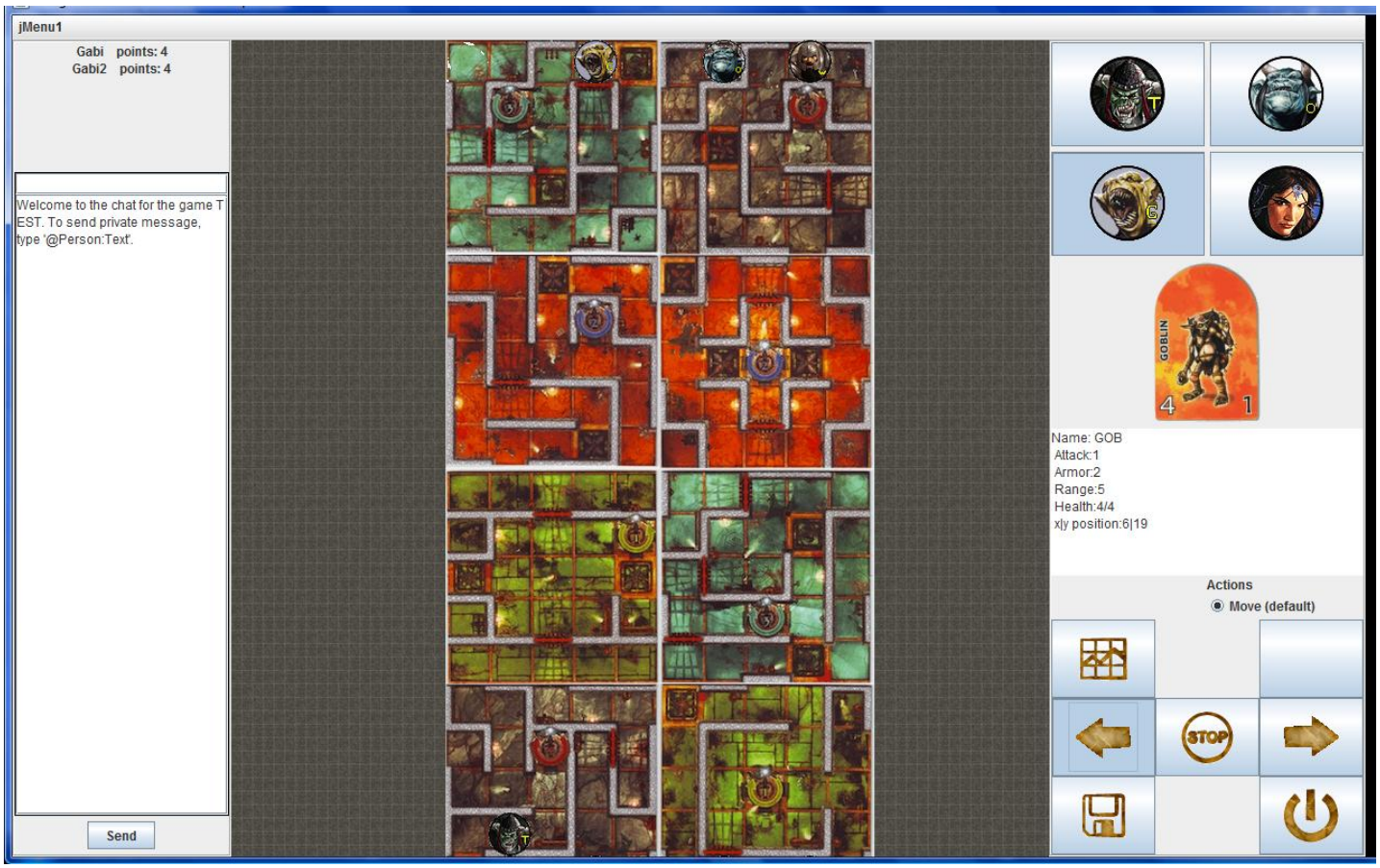


Nun stellt jeder Spieler ein Team aus 4 Figuren zusammen. Dabei kann der Spieler zwischen 2 Versionen wählen: der Klassischen und der Studenten Version (im Beispiel: die Klassische Version). Durch klicken auf den Button „Choose“ wird eine Figur ausgewählt. Ist eine Figur ausgewählt, so wird aus dem „Choose“ Button ein „Remove“ Button, womit der Spieler die Figur wieder abwählen kann, sollte er sich falsch entschieden haben.





Wenn der Spieler alle 4 Figuren ausgewählt hat, dann erscheint ein Button mit der Aufschrift „OK“. Nach klicken auf den „OK“ Button wird die Karte des Labyrinths aufgebaut und das Spiel kann beginnen.



9. Verwendete Bibliotheken

Verwendet wurden die Standard Bibliotheken aus der JRE System Library [JavaSE-1.6]:

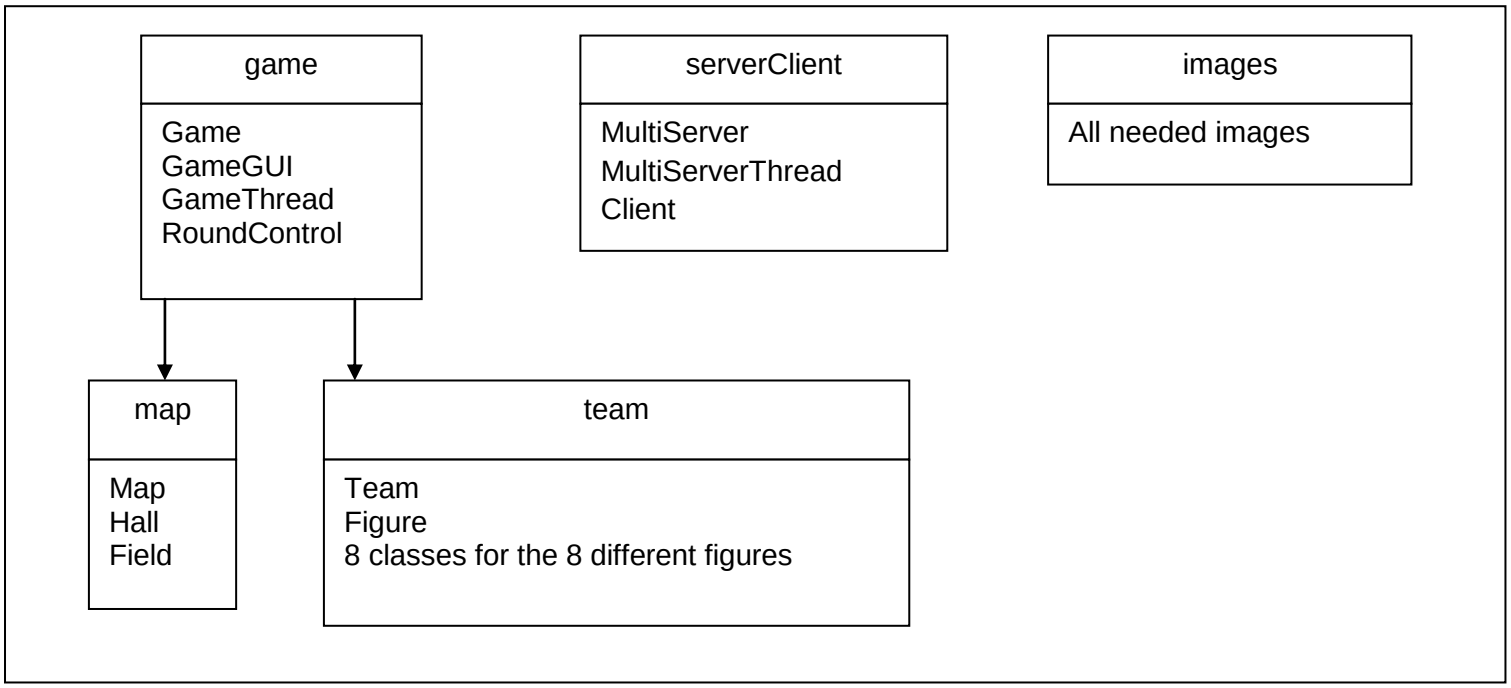
```
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.Writer;


import java.util.Properties;
import java.util.StringTokenizer;
import java.util.Vector;

import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;

import java.awt.Frame;
import java.awt.FileDialog;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;


import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.AbstractButton;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
```

```
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingConstants;
import javax.swing.WindowConstants;
import javax.swing.border.BevelBorder;
import javax.swing.border.LineBorder;
import javax.swing.border.SoftBevelBorder;
```

Für den JUnit Test (aus JUnit4):

```
import org.junit.Test;
import junit.framework.TestCase;
```

10. Struktur des Programms



Das Paket „game“ ist für die Regelüberprüfung („Game“ und „RoundControl“), die Spielverwaltung („Game“) und die grafische Darstellung („GameGUI“) zuständig.

Die Unterpakete „map“ und „team“ stellen die Funktionalität der „Map“ mit deren Hallen und Feldern und die Funktionalität der einzelnen Figuren („team.Figure“) zur Verfügung.

Dabei hat jede Figur eine eigene Klasse, welche die passenden Methoden der Klasse „Figure“ erbt.

Die Klasse „team.Team“ verwaltet die Daten, die für das Team als Ganzes, sprich den Spieler, von Bedeutung sind; beispielsweise die Anzahl eigener Figuren, die das Labyrinth bereits durchquert haben oder die erreichten Siegespunkte.

11. Software-Qualitätsmanagement (Testing)

Wir haben den UnitTest in der Klasse unitTest geschrieben, die sich im Package Test befindet. Sie enthält 9 Tests bezüglich der Figuren und deren Aktionen.

Der Test beginnt mit der Methode setUp, die den Test aufbaut. Dort werden 4 verschiedene Figuren initialisiert, auf eine Map gesetzt und verschiedenen Teams zugewiesen. Dann folgt die Abfrage, ob der Name der Figur wirklich dem Namen entspricht, den diese haben sollte. Dies ist wichtig, da die Abfrage, ob diese Figur eine bestimmte Aktion durchführen kann vom Namen der Figur abhängt. Anschließend wird die Position der Figur getestet. Dies ist wichtig, damit man weiß, ob eine Figur richtig auf die Map gesetzt wird wenn man den Konstruktor der Figur mit der x- und y-Position aufruft.Dann wird die Bewegung einer Figur getestet. Wir haben dazu den Goblin extra auf ein Feld gesetzt von dem aus er nach rechts laufen kann, jedoch nicht nach links. Dann werden zwei Abfragen getestet. Zum einen, ob er nach rechts laufen kann (AssertTrue) und zum anderen, ob er nicht nach links laufen kann (AssertFalse). Der Goblin steht auch vor einer Falle, über die er drüber springen kann, deswegen testen wir auch, ob er dies auch kann.

Diese Tests werden alle auf Methoden angewendet, die sich in der Klasse Figure befinden (canMoveLeft(), canMoveRight(), canJumpDown()). Diese Methoden greifen auf die Map zu und da diese Methoden fehlerfrei getestet wurden, wissen wir dass diese Klassen problemlos aufeinander zugreifen können.

Wir testen nun noch das Gitter öffnen einer Diebin. Die Diebin haben wir neben ein geschlossenes Gitter gesetzt und es wird erst mal getestet, ob dieses Gitter auch wirklich geschlossen ist. Danach wird die Methode openBars() durch die Diebin ausgeführt. Und anschließend wird getestet, ob das Gitter auch wirklich geöffnet wurde. Dies lief auch problemlos, also wissen wir, dass die Klasse Figure mit der Klasse Game interagieren kann.

Jetzt wird noch ein Kampf getestet, dazu haben wir einen Troll neben den Goblin gesetzt. Auf den Troll wird nun die Methode fight() angewendet und als Parameter werden die angegriffene Figur (der Goblin) und die zwei Kampfkarten übergeben. Diese sind so gewählt, dass der Troll den Kampf gewinnt und dass der Goblin stirbt. Es dann wird getestet ob der Goblin auch wirklich tot ist (mit Hilfe eines Boolean in Figure: boolean isDead), welcher nur dann auf true gesetzt wird wenn diese Figur 0 Lebenspunkte hat. Dieser Test läuft auch fehlerfrei, wir können daraus also schließen, dass Figuren auch zwischen sich selbst interagieren können.

Jetzt wird mit der Wandläuferin die Methode beam() getestet. Die Wandläuferin wird neben eine Wand gesetzt und dann wird getestet, ob sie sich durch diese Beamen kann. Dieser Test läuft auch einwandfrei.

Zuletzt wird noch die Methode heal() mit dem Priester getestet. Wir starten noch einmal einen Kampf zwischen Troll und Goblin, aber diesmal so, dass der Troll zwei Lebenspunkt verliert. Er ist nicht tot, aber er kann sich nun nicht mehr bewegen. Der Priester steht neben dem Troll und es wird die Methode heal aufgerufen. Es wird anschließend getestet, ob die Lebenspunkte des Trolls wieder auf Max stehen. Dies ist der Fall, wir können also behaupten, dass die Klasse Figur einwandfrei mit den anderen Klassen interagiert. Variablen werden korrekt gesetzt, nachdem die korrekten

Methoden aufgerufen werden. Der Test hat natürlich keine 100 Prozentigen Coeverage der Klasse Figur, aber die meisten Regelüberprüfungen sind einwandfrei getestet worden.

12. Beispielsession

Siehe LIVE DEMO