Real-Time-Strategy Game

Pflichtenheft

Version 1.3

Gruppe 8: Gregor Plebanek, Dominik Ridder, Marvin Winkens

Datum: 15.12.2015

Inhaltsverzeichnis

Inhaltsverzeichnis 0
1. Zielbestimmungen 1
1.1. Musskriterien 1
1.2. Wunschkriterien 1
1.3. Abgrenzungskriterien 1
2. Produkteinsatz 1
2.1. Anwendungsbereiche 1
2.2. Zielgruppen 1
2.3. Betriebsbedingungen 1
3. Produktübersicht 2
4. Produktfunktionen 3
5. Produktdaten 5
6. Produktleistungen 6
7. Qualitätsanforderungen 6
8. Bedienoberflächen 6
9. Nichtfunktionale Anforderungen 6
10. Technische Produktumgebung 7
10.1. Software 7
10.2. Hardware 7
10.3. Orgware 7
11. Spezielle Anforderungen an die Entwicklungsumgebung 7
11.1. Software 7
11.2. Hardware 7
11.3. Orgware 7
12. Entwicklungsschnittstellen 7
13. Gliederung in Teilprodukte 7
14. Ergänzungen 7

1. Zielbestimmungen

Es soll eine Desktop-Anwendung entwickelt werden, worin der Nutzer strategische Gefechte mit unterschiedlichen Einheiten gegen einer oder mehreren KI's in Echtzeit ausfechten kann.

1.1. Musskriterien

Ein Hauptmenü zum Starten der folgenden Fenster:

- Einzelspieler
- Spiel laden
- Achievements
- Highscore
- Einstellungen

Im Einzelspielerbereich kann der Spieler ein Szenario, beinhält auch ein Tutorial, mit weiteren Spezifikation auswählen und starten. Der Benutzer kein ein Spiel mittels einen Namen laden, um an einem gespeicherten Punkt wieder weiterzuspielen. Im menüpunkt Achievements kann der Benutzer seine bisherigen erlangten Erfolge ansehen. In den Highscores kann man die Statistik der Top 5 Spiele sehen. Bei den Einstellungen soll man Soundeinstellungen und Tastenbelegungen ändern können.

1.2. Wunschkriterien

1.3. Abgrenzungskriterien

Das Spiel soll keine 3D Models verwenden.

2. Produkteinsatz

2.1. Anwendungsbereiche

Das Produkt wird nur im privaten Bereich ein Nutzen finden.

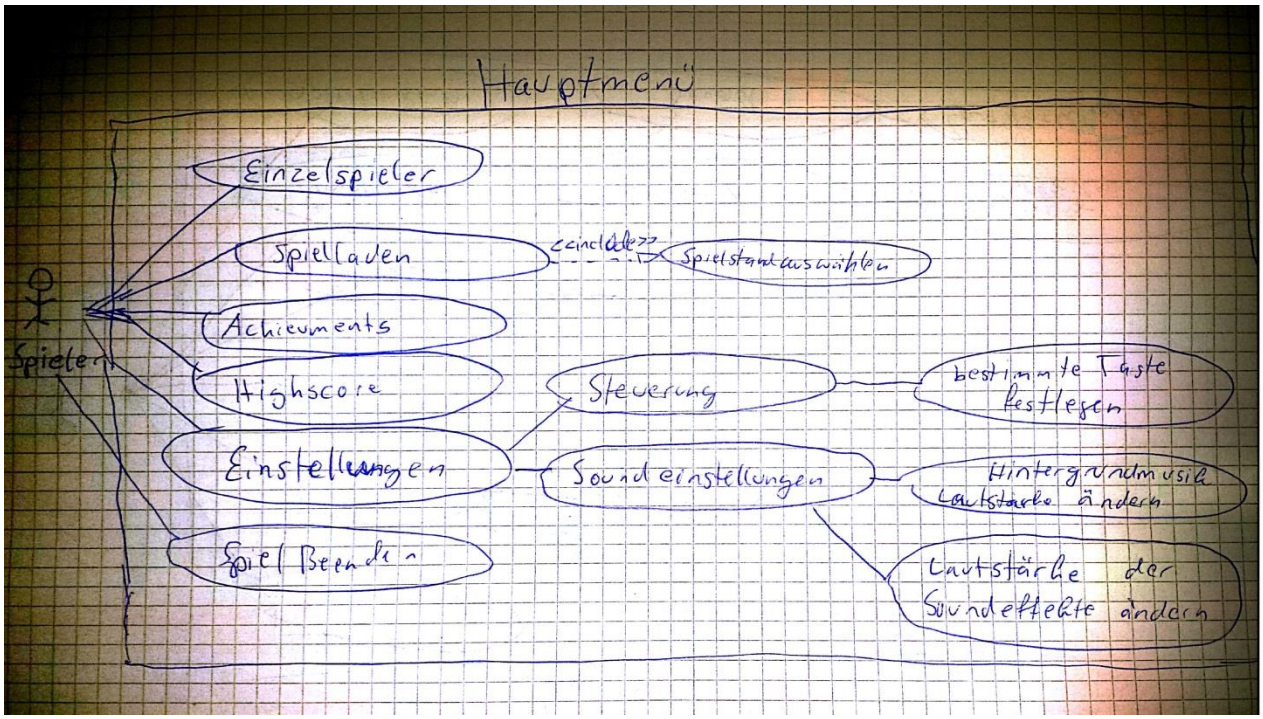
2.2. Zielgruppen

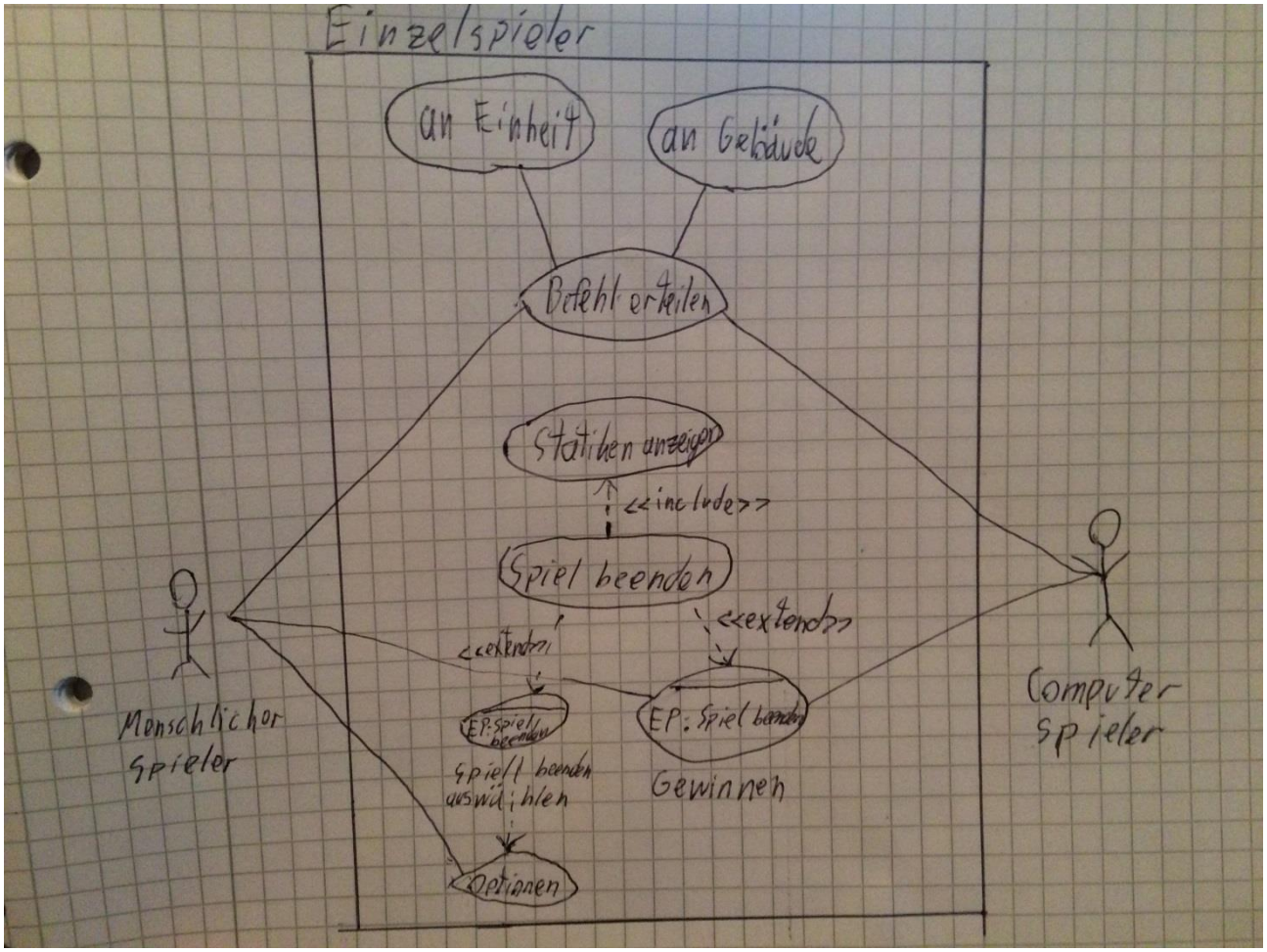
Menschen/Spieler ab 12 Jahren.

2.3. Betriebsbedingungen

Eine Windows Betriebssystem ab XP oder ein Linux Betriebssystem mit einer installierten Java Version.

3. Produktübersicht





Im Einzelspieler Modus können die Spieler (sowohl Menschlich, als auch Computer gesteuert) befehle an Einheiten und Gebäuden erteilen. Wenn ein Spieler Gewinnt sollte das Spiel Beendet werden. Außerdem soll der Menschliche Spieler auch die Möglichkeit haben, über Optionen, dass Spiel direkt zu Beenden. Nach dem Spiel soll außerdem eine Statistik über das gerade abgeschlossene Spiel angezeigt werden.

4. Produktfunktionen

/F100/ Hauptmenü

/F110/ Einzelspieler

/F111/

Hier wählt man ein Spiel aus verschiedenen Szenarien und anderen Spezifikationen und kommt so zu Start eines neuem Spiels.

/F112/

Es gibt 3 verschiedene Spielmoden

- Belagerung ()
- Last-Man-Standing ()
- King of the Hill ()

Außerdem gibt es noch ein Tutorial, worin dem Spieler die Steuerung und andere Grundlagen erklärt wird.

/F113/

In einem Spiel spielt man gegen eine Künstliche Intelligenz(KI)

/F114/

Die KI hat 2 unterschiedliche Schwierigkeiten haben.

/F120/ Spiel laden

/F121/

Ein Spielstand wird mittels Namen (was einer Datei entspricht) geladen.

/F130/ Achievements

/F131/

Hier sieht der Benutzer alle seine Erfolge die er bisher beim Spieler erlangt hat.

/F140/ Highscore

/F141/

Hier werden die Top 5 Spiele angezeigt.

/F150/ Einstellungen

/F151/

Sound Einstellungen können dort vorgenommen werden können.

/F152/

Tastenbelegungen können hier geändert werden.

/F160/ Spiel Beenden

/F200/ Einheiten

/F210/

Bei Einheiten unterscheidet man zwischen Fern- und Nahkämpfer

/F220/

Einheiten besitzen sogenannte Strukturpunkte(HP), wenn diese unter 1 fallen stirbt eine Einheit

/F230/
Einheiten haben zu ihren Standardangriff auch noch speziellere Fähigkeiten.
/F240/
Entsprechend zum Angriffs- und Verteidigungstyp haben die Einheiten einen Wert den deren Angriff bzw. deren Verteidigung wiederspiegelt
/F250/
Der Angriff teilt sich unter einen minimalen Angriff und einen maximalen.

/F300/ Gebäude
/F310/
Für das Errichten von Gebäuden und Erstellen von Einheiten werden die Rohstoffe Holz, Stein, Eisen, Gold und Zeit benötigt.
/F320/
Bestimmte Gebäude können ebenfalls an einem Kampf teilnehmen z.B. ein Turm.
/F330/
Alle Gebäude haben den Verteidigungstyp Schwer.
/F340/
Mit Investitionen können Gebäude verbessert werden und bieten dann mehr Produktionsmöglichkeiten.
/F350/
Eine Verbesserung hat nur auf neu erstelle Gegenstände/Einheiten Einfluss und nicht auf schon erstellte.
/F360/
Gebäude die zur Ausbildung von Truppen dienen haben ein Gegenstandsinventar. In Abhängigkeit von den Inhalten des Inventars können unterschiedliche Einheiten ausgebildet werden.
/F370/
Produktionsgebäude, wenn ein Arbeiter darin arbeitet, produzieren periodisch einen ausgewählten Gegenstand.
/F380/
Es soll Gebäude geben, die die maximale Anzahl von Einheiten die eine Stadt beherbergen kann vergrößert.

/F400/ Im Spiel
/F410/
Der Spieler und die KI müssen im Spiel Gebäude errichten, Einheiten erstellen und anderen Einheiten angreifen können.
/F420/
Die Spieler (auch KI) starten mit einem Hauptgebäude und 4 Arbeiter, die Gebäude reparieren und bauen, Rohstoffe sammeln und in Produktionsgebäuden Gegenstände prodozieren können.
/F430/
Im Spiel soll es möglich sein, ein Spiel pausierendes Menü zu öffnen.
/F440/
Nach dem Beenden oder Abbrechen eines Spiels soll eine Statistik angezeigt werden

/F500/ Kampf-System
/F510/

Zusammenhang zwischen den Angriffs- und Verteidigungstypen:

- Stumpf ist wirkungsvoller gegen Schwer, aber weniger wirkungsvoll gegen Leicht
- Spitz ist wirkungsvoller gegen Unverteidigt und Magisch, aber weniger wirkungsvoll gegen Leicht und Schwer
- Klinge ist wirkungsvoller gegen Unverteidigt und Leicht aber weniger wirkungsvoll gegen Schwer
- Alle magischen Angriffe sind gegen Schwer wirkungsvoller und gegen Unverteidigt weniger wirkungsvoll
- Feuer ist wirkungsvoller gegen Wasser, aber weniger wirkungsvoll gegen Luft
- Wasser ist wirkungsvoller gegen Erde, aber weniger wirkungsvoll gegen Feuer
- Erde ist wirkungsvoller gegen Luft, weniger wirkungsvoll gegen Wasser
- Luft ist wirkungsvoller gegen Feuer, weniger wirkungsvoll gegen Erde
- Licht ist wirkungsvoller gegen Erde, Feuer, Leere
- Leere ist wirkungsvoller gegen Wasser, Luft, Licht

/F600/ Gegenstände

/F610/

Gegenstände werden in Produktionsgebäuden produziert und in anderen Gelagert.

/F620/

Bestimmte Gegenstände können gefunden werden und müssen in speziellen Gebäuden aktiviert werden.

5. Produktdaten

/D10/ Einheiten

/D11/

Einheiten besitzen verschiedene Angriffstypen die wie folgt heißen:

- Stumpf
- Spitz
- Klinge
- Magie
  - Feuer
  - Luft
  - Wasser
  - Erde
  - Licht
  - Leere

Einheiten besitzen ebenfalls unterschiedliche Verteidigungstypen die z.B.

- Unverteidigt
- Leicht
- Schwer
- Magie
  - Feuer

- Luft
- Wasser
- Erde
- Licht
- Leere

/D20/

Das Speichern von Maps/Spielstaenden erfolgt durch das folgende Schema:

- Eine Zeile faengt mit einem Bezeichner an (z.B. dem Klassenamen)
- gefolgt von einem „@“ Zeichen
- gefolgt von einer Zeichenkette, die von der jeweiligen Klassen selbst ausgewertet werden soll.

- Durch eine Vielzahl von Zeilen lassen sich somit alle notwendigen Informationen abbilden.

- Zum Abspeichern soll die Zeile von der jeweiligen Klasse selbst erzeugt werden.

/D30/

Die Highscore/Achievements werden nach dem in D20 erklaerten vorgehen gespeichert werden.

/D40/

Einstellungen vom Spieler sollen in einer CSV (Comma-separated values) Datei abgespeichert werden

6. Produktleistungen

/L10/

Alle Grafiken sollen in einem Fantasy-Setting umgesetzt werden. Zudem sollte das Spiel fluessig laufen.

7. Qualitätsanforderungen

Das RTS-Game soll den Anforderungen eines Echtzeit Spiels erfüllen.
Entsprechend müssen Befehle die der Spieler ausführt mit keiner Verzögerung stattfinden.

Weitere Spielmoden sollten mit wenig Aufwand implementierbar sein.

Fähigkeiten von Einheiten sollen mit wenig Aufwand implementierbar sein.

8. Bedienoberflächen

Mittels einer GUI findet die Kommunikation zwischen dem User und dem Spiel statt.

9. Nichtfunktionale Anforderungen

10. Technische Produktumgebung

10.1. Software

Das Produkt wird in der Programmiersprache Java entwickelt.

10.2. Hardware

Zum Spielen wird eine Maus, Tastatur und ein Bildschirm benötigt.
Für ein besseres Spielerlebnis empfehlen wir ein Audioausgabegerät ist aber optional.

10.3. Orgware

Für die Orgware werden keine speziellen Anforderungen benötigt.

11. Spezielle Anforderungen an die Entwicklungsumgebung

11.1. Software

Eclipse als integrierte Entwicklungsumgebung
Git als Versionsverwaltung

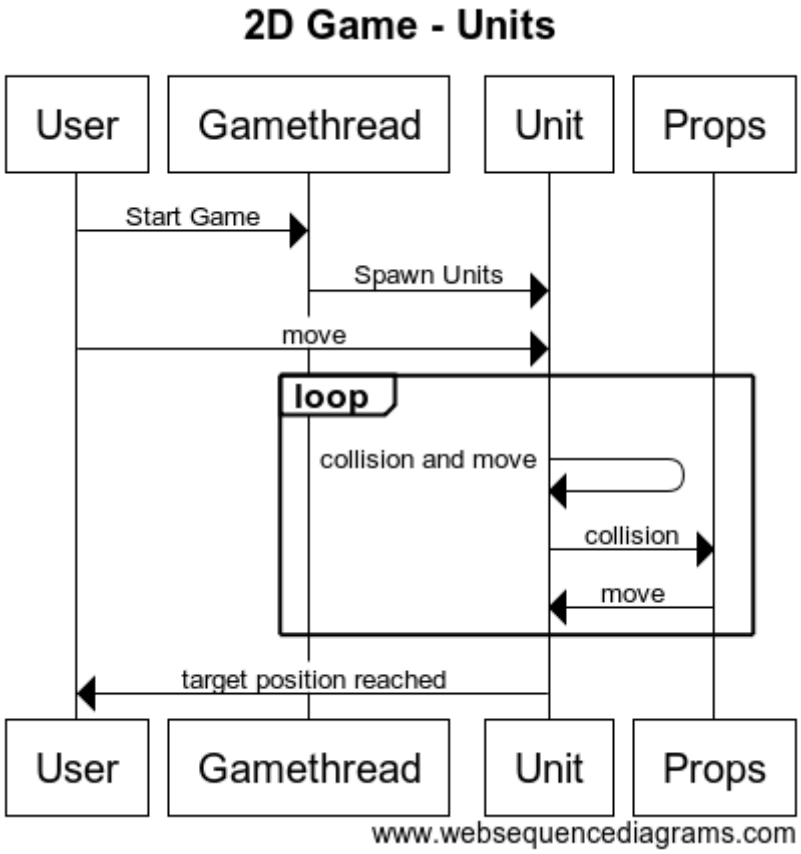
11.2. Hardware

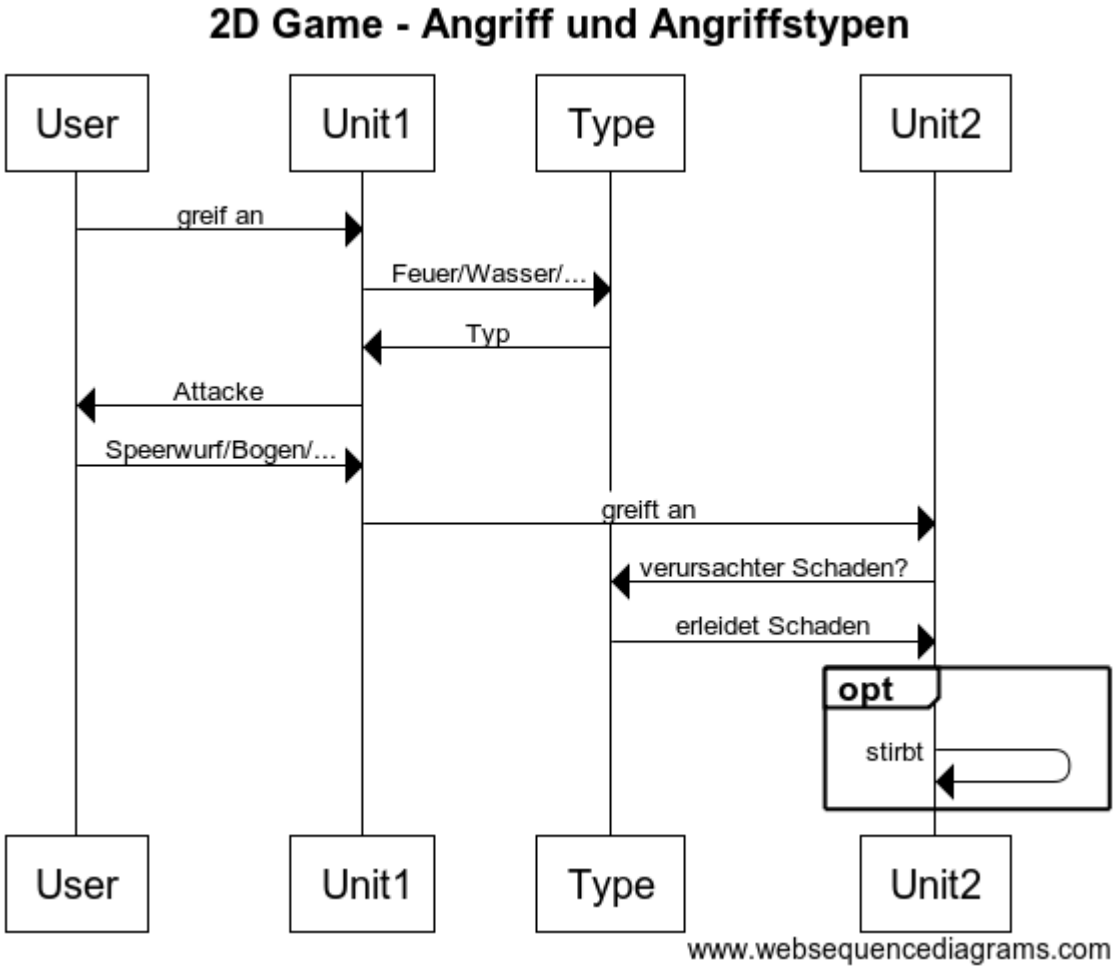
11.3. Orgware

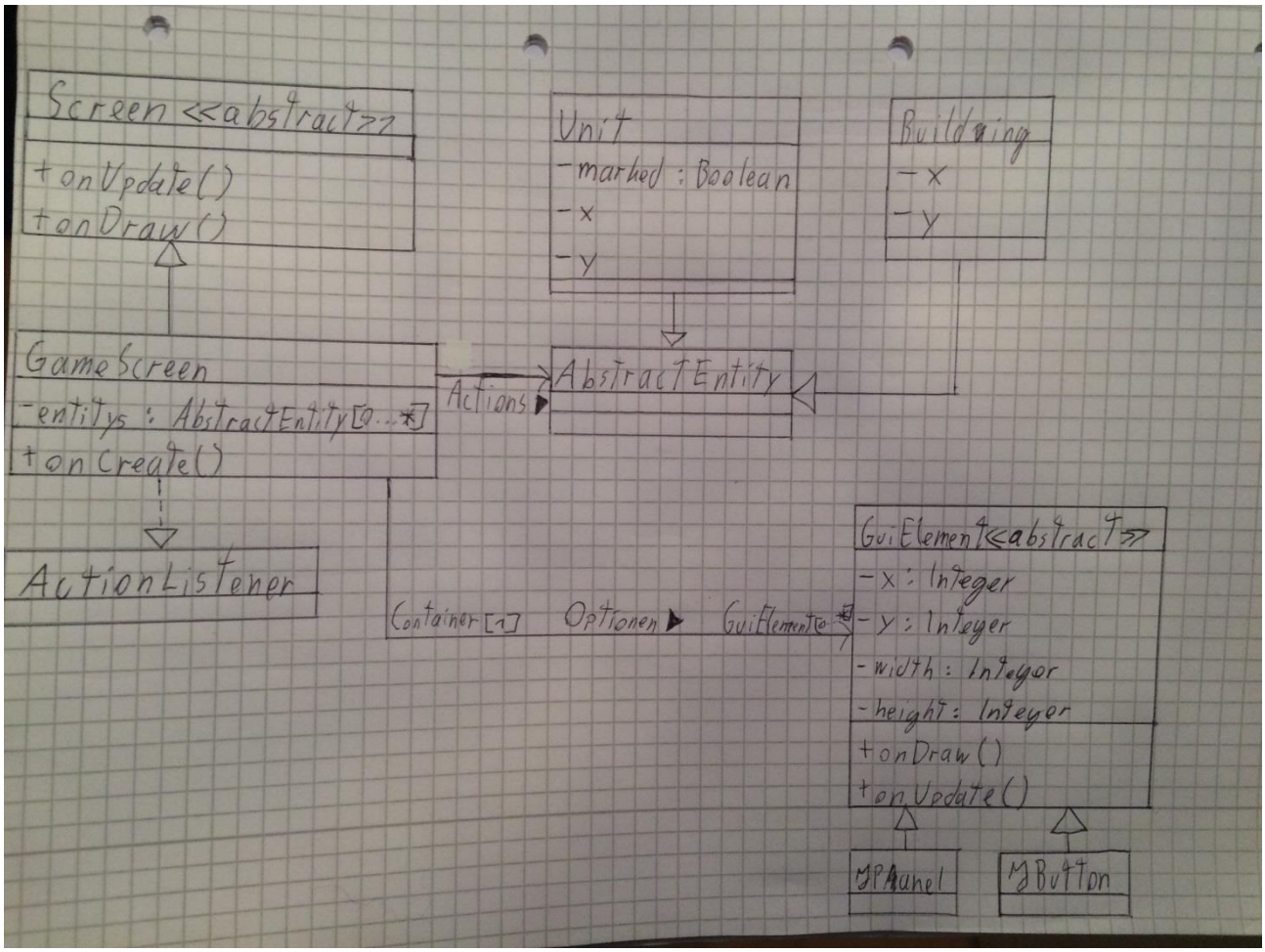
12. Entwicklungsschnittstellen

13. Gliederung in Teilprodukte

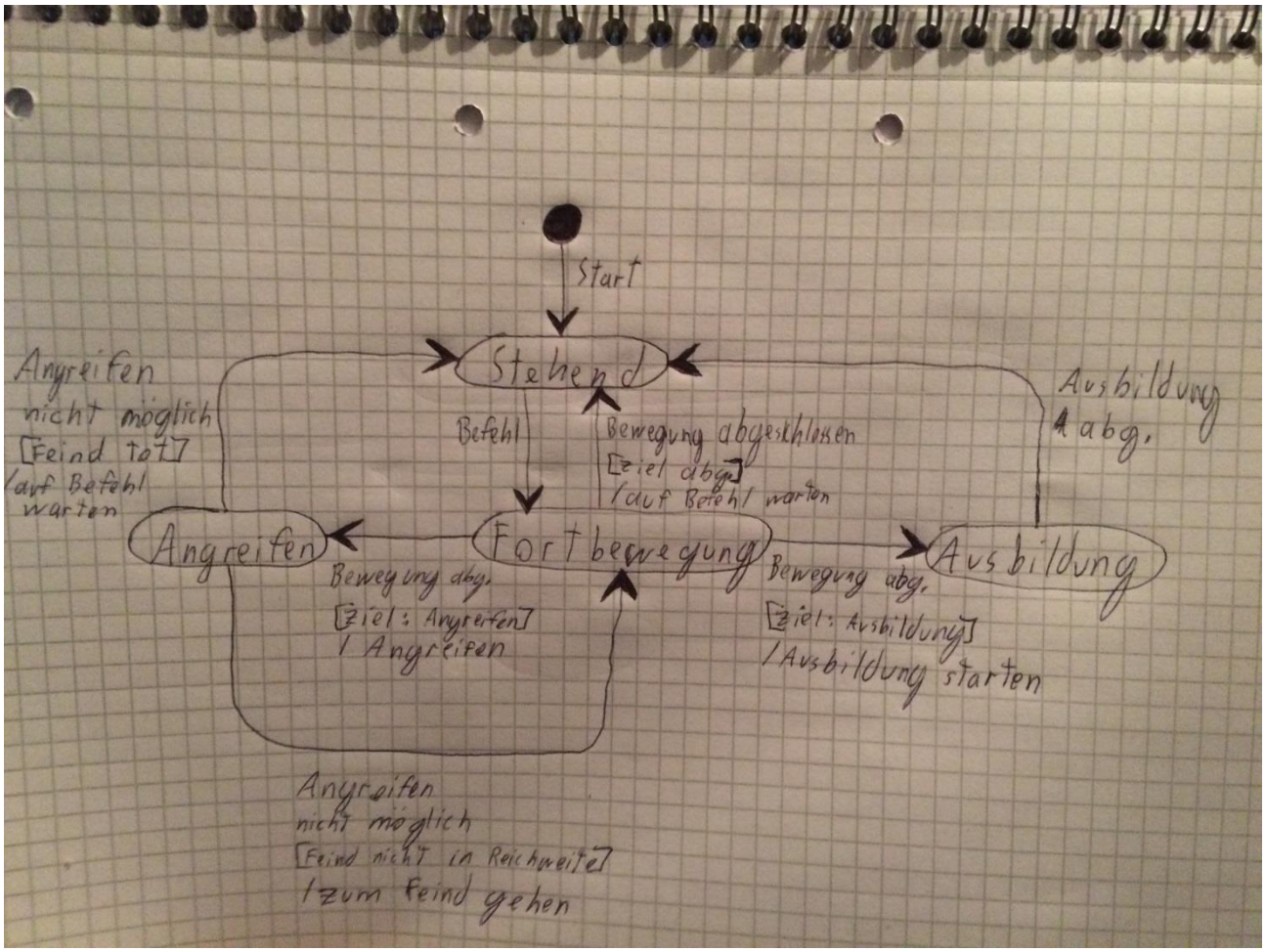
14. Ergänzungen



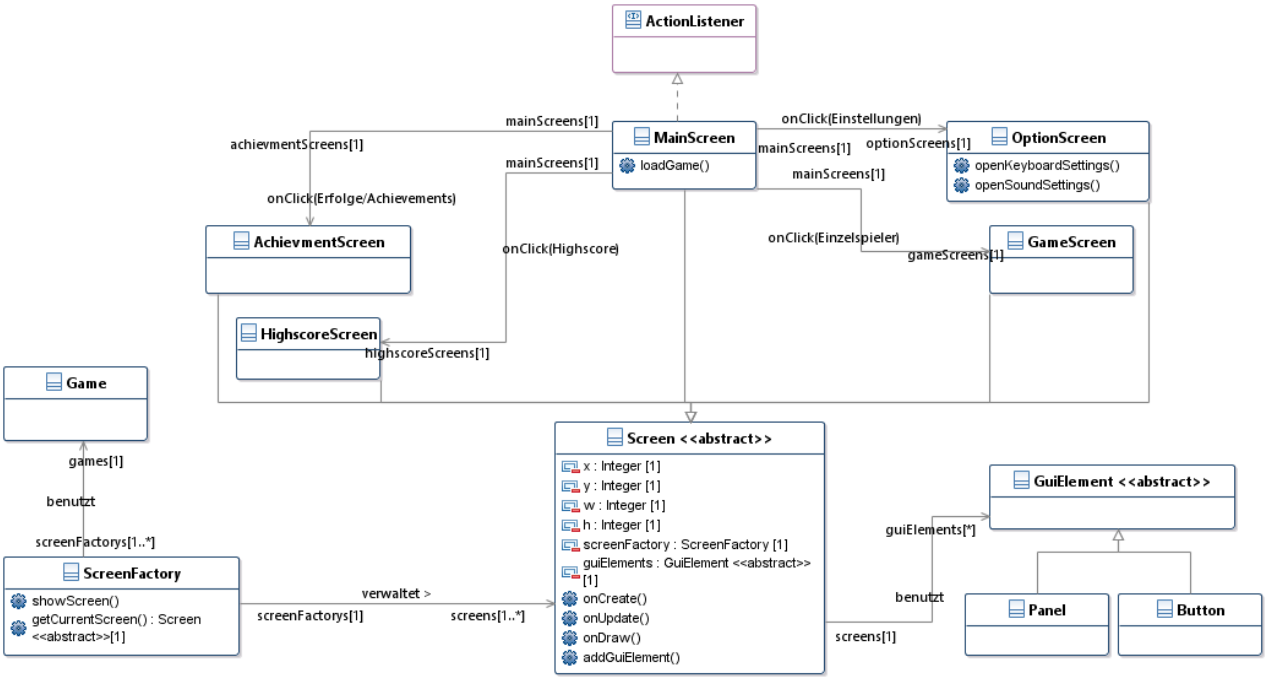

2D Game - Angriff und Angriffstypen
User
Unit1
Type
Unit2
greif an
Feuer/Wasser/...
Typ
Attacke
Speerwurf/Bogen/...
greift an
verursachter Schaden?
erleidet Schaden
opt
stirbt
User
Unit1
Type
Unit2
www.websequencediagrams.com



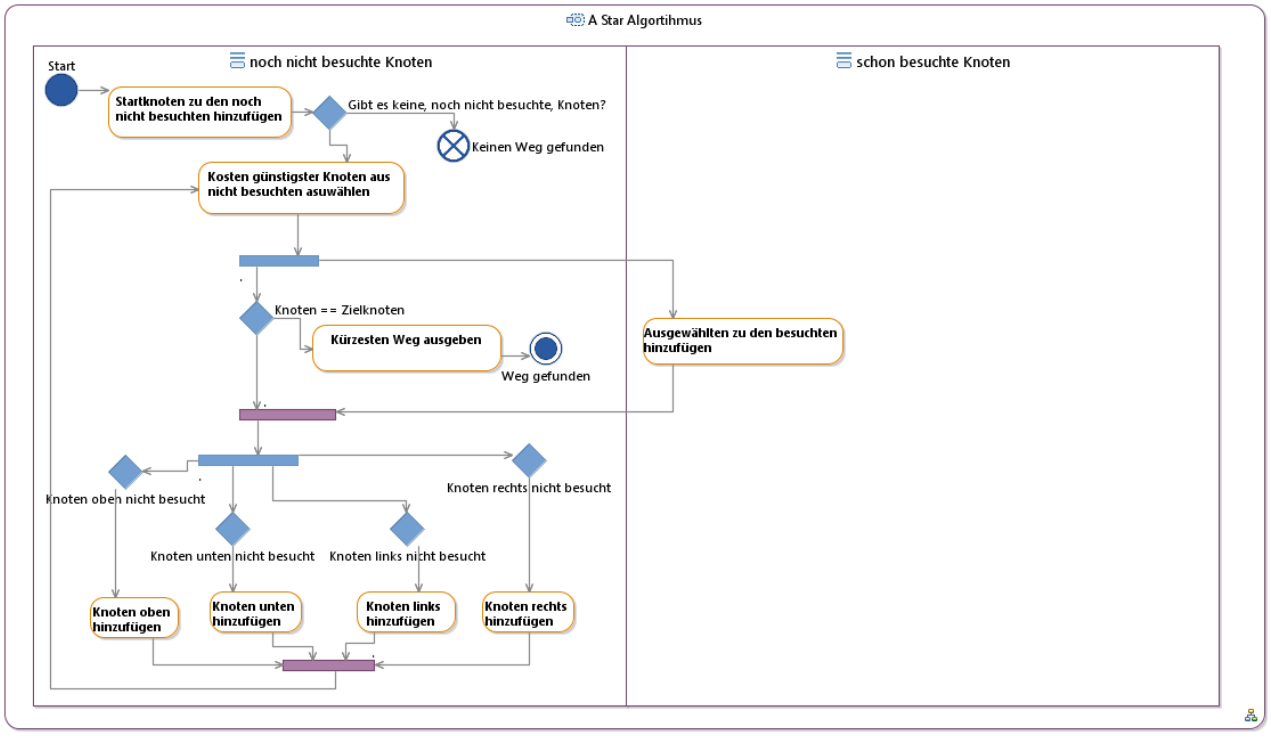
UML Klassendiagramm zum Anwendungsdiagramm „Einzelspieler“



Zustandsdiagramm zu einer Einheit/Unit



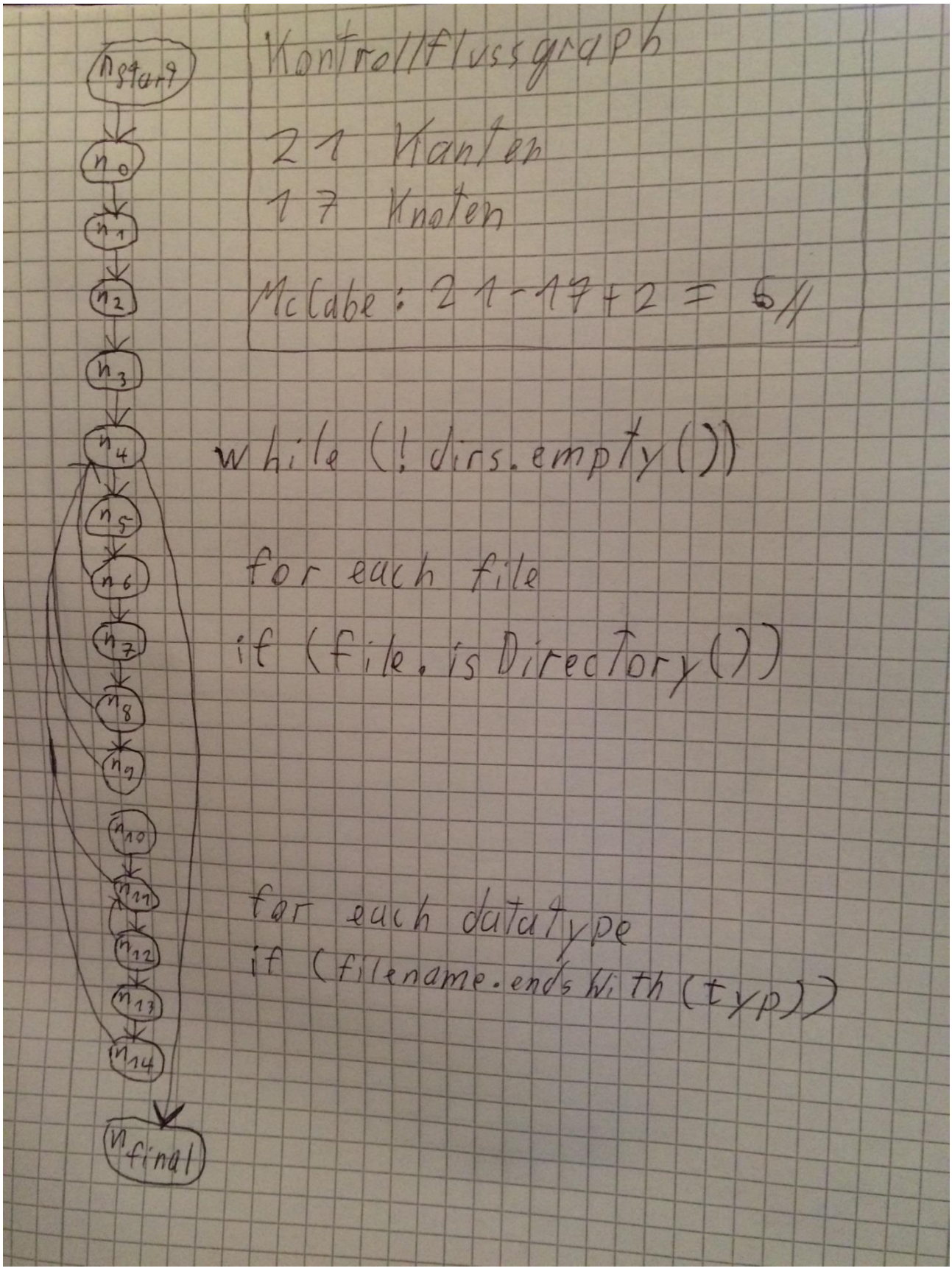
UML Klassendiagramm zum Anwendungsdiagramm „Hauptmenü“



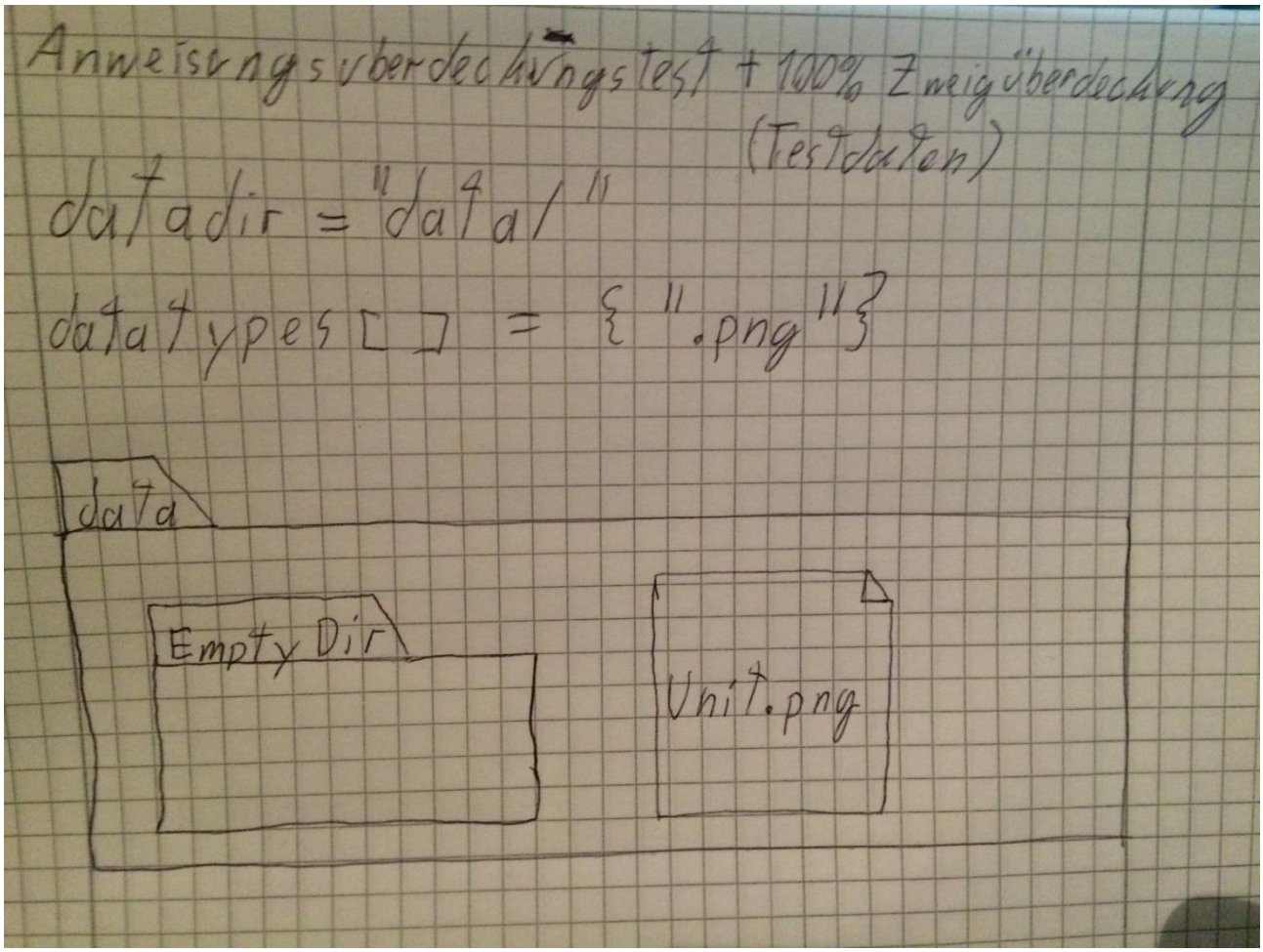
Die Methode ImageManager.loadRelativNames wird im Folgenden getestet.

Quellcode der Methode:

```
	/**
	 * This static Method should be used to have an easier access to the Data
	 * that provided by a Directory.
	 *
	 * @param datadir
	 *            The Directory, to search for Files.
	 * @param datatypes
	 *            An accepted array of File extensions/endings.
	 * @return A HashMap with the Mapping from relative Names to Path;
	 */
	public static HashMap<String, String> loadRelativNames(String datadir,
			String datatypes[]) {
		HashMap<String, String> relativNames = new HashMap<String, String>();
		Stack<File> dirs = new Stack<File>();
		File nextfile = null;

		dirs.add(new File(datadir));

		while (!dirs.empty()) {
			nextfile = dirs.pop();
			for (File file : nextfile.listFiles()) {
				if (file.isDirectory()) { // Adding dirs to search
					if (!file.getName().endsWith(".git")) {
						dirs.push(file);
					}
				} else { // is File
					String filename = file.getName();

					for (String typ : datatypes) { // File is an image?

						if (filename.endsWith(typ)) {
							relativNames.put(filename,
							file.getAbsolutePath()); // remember

							// File

							// name
							break;
						}
					}
				}
			}
		}

		return relativNames;
	}
```


Kontrollflussgraph
21 Kanten
17 Knoten
McCabe: 21-17+2 = 6
while (!dirs.empty())
for each file
if (file.isDirectory())
for each datatype
if (filename.endsWith(typ))
nstart
n0
n1
n2
n3
n4
n5
n6
n7
n8
n9
n10
n11
n12
n13
n14
nfinal

Testfaelle:



JUnit Testcase:

```
package tests;

import static org.junit.Assert.*;

import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;

import org.junit.Test;
import org.junit.runner.notification.Failure;
import org.junit.runner.*;

import dataManagement.ImageManager;

public class ManagerTest {
	public static void main(String[] agrs) {
		Result rc = new Result();

		rc = JUnitCore.runClasses(ManagerTest.class);

		System.out.printf("%d tests were executed, %d of them with failures\n",
```

```
                        rc.getRunCount(), rc.getFailureCount());

            if (!rc.wasSuccessful()) {
                  List<Failure> fList = rc.getFailures();
                  for (Failure f : fList) {
                        System.out.println(f.getTestHeader());
                        System.out.println(f.getMessage());
                  }// end of for (Failure f: fList)
            }// end of if (!rc.wasSuccessful)
      }// end of method "main(String[] args)"

      @Test
      public void testRelativNames() {
            File dir = new File("data2/");
            File empty = new File("data2/EmptyDir");

            dir.mkdir();
            empty.mkdir();

            File unit = new File("data2/Unit.png");
            try {
                  File tmp = File.createTempFile("Unit", "png", dir);
                  tmp.renameTo(unit);
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }


            String datadir = dir.getName();
            String datatypes[] = { ".png" };

            HashMap<String, String> result = ImageManager.loadRelativNames(datadir,
                        datatypes);

            assertTrue(result.values().size() == 1);
            assertEquals(result.get("Unit.png"), (unit.getAbsolutePath()));

            unit.delete();
            empty.delete();
            dir.delete();
      }
}
```

Ergebnis auf der Konsole:

```
1 tests were executed, 0 of them with failures
```