

Fachhochschule für Technik und Wirtschaft Mittweida

Fachbereich Mathematik/Physik/Informatik

Schriftliche Ausarbeitung zum Thema:

Rundenbasiertes Rollenspiel mit einer 3-dimensionalen Spielansicht auf Basis eines Brettspiels

– unter Verwendung der Standard -3D-API Java3D für Java 6 –

Wintersemester 2008/2009

vorgelegt bei

Prof. Dr.-Ing. Rainer Gaudlitz

von

Lars Denkewitz
IF05wP1

1. Inhalt

2.	Motivation	3
3.	Aufgabenstellung	3
4.	Problemanalyse.....	4
a.	Karte laden / Spiel erstellen.....	4
b.	Spielereingaben steuern.....	4
c.	Bearbeitung und Überprüfung der Spieleraktionen.....	5
d.	Aktualisierung des Spielfeldes und der restlichen GUI.....	5
5.	Lösungskonzeption.....	6
a.	Karte laden / erstellen.....	6
b.	Spielsteuerung	8
c.	Spielereingaben und Grafische Benutzeroberfläche.....	8
6.	Programmentwurf.....	9
7.	Implementierung.....	11
8.	Erweiterungsmöglichkeiten und Fazit	14
9.	Literaturverzeichnis.....	15
10.	Abbildungsverzeichnis.....	15
11.	Selbstständigkeitserklärung.....	15

2. Motivation

Die Motivation entstand mit der Ideenfindung einer geeigneten Aufgabenstellung für diese Projektarbeit. Dabei hat sich die Idee durchgesetzt, ein Computerspiel zu entwickeln. Da dies jedoch in Alleinarbeit entstehen würde, mussten im Umfang und Inhalt schon im Vorfeld klare Grenzen gezogen werden.

Anhand des interessanten Szenarios und dem damit verbundenen Wegfall der Inhaltssuche, wurde als Basis des Spiels ein bereits existierendes Brettspiel gewählt. Dieses Brettspiel namens „Hero Quest“ ist ein Fantasy-Rollenspiel aus dem Jahr 1989. Hierbei tritt eine kleine Heldengruppe im typischen Rollenspielstil gegen eine relativ überschaubare Anzahl von Gegnern an.

Bezüglich dieser Basis konnten sich gegenüber dem Originalspiel Vereinfachungen überlegt werden, um die Entwicklungszeit in einem gewissen Rahmen zu halten. Dies führte zur Aufgabenstellung, wie sie im Folgenden beschrieben ist.

3. Aufgabenstellung

Mit der Vorlage eines Brettspiels soll ein Computerspiel entwickelt werden, welches folgende Funktionen, sowie Vereinfachungen gegenüber dem Original besitzt:

- Ein Mehr-Spieler-Spiel ohne künstliche Intelligenz (Computergegner), welches vorerst auf zwei Spieler begrenzt ist
- Spielziel ist lediglich das besiegen aller gegnerischen Figuren
- Verzicht auf Wegfindung, Rollenspielaspekte wie Inventar oder Erfahrung der Figuren, sowie auf die Speicherung von Spielständen
- Verwendung von einfachen geometrischen Grundformen anstatt komplexer 3D-Modelle
- Durch einfaches Kartenformat können nicht sämtliche Inhalte des Originals auf dem Spielfeld dargestellt werden
- Einfach zu handhabende jedoch etwas umständliche Figurensteuerung durch vorheriges Drehen in gewünschte Blickrichtung

Des Weiteren ist zu erwähnen, dass dieser Beleg im Rahmen des Fachs Grafikprogrammierung stattfindet und somit das Spielfeld in 3D mittels Java3D dargestellt wird.

4. Problemanalyse

Da die Komplexität eines solchen Spiels dem einer einfachen Applikation übersteigt, ist die Problemanalyse ein sehr zeitaufwändiger Teil der Entwicklung. Auch bestand bisher keinerlei Erfahrung in der Entwicklung eines komplexeren Computerspiels. Mit der Entscheidung, Spielprinzip und Grundinhalte vom Original zu übernehmen, wurde der Abschnitt der Ideen- und Inhaltsfindung um Einiges verkürzt. Auch die Rundenbasiertheit des Spiels wird die Entwicklung gegenüber Echtzeit vereinfachen.

Folgende Funktionen bilden den Hauptbestandteil des Programms.

a. Karte laden / Spiel erstellen

Bei der Erstellung eines neuen Spiels muss eine existierende Kartenvorlage im eigenen definierten Kartenformat vorhanden sein. Aus dieser Karte, welche in Form von drei Textdateien (*.gmp, *.omp, *.fmp)¹ vorliegt, werden alle Informationen eingelesen, die zum Aufbau des Spielfeldes nötig sind. Zu diesen Informationen zählen u.a. Felder, Figuren, Objekte wie Tische oder Türen, Startposition der Helden oder auch der Kartennname.

Auf Basis dieser Daten wird das Spielfeld in Form eines Canvas3D erstellt und die Startpositionen der Figuren und Objekte wird eingenommen. Zudem werden alle Grundinformationen zurückgesetzt, wie zum Beispiel Rundenanzahl, Bewegungspunkte und Lebenspunkte der Figuren. Diese müssen von der GUI (Graphical User Interface) angezeigt werden, abhängig davon welche Figur in Runde 1 startet.

b. Spielereingaben steuern

Die Interaktion mit der Spielfigur ist ein wichtiger Bestandteil des Spiels. Diese kann ihre Blickrichtung ändern, sich in diese bewegen oder eine feindliche Figur in Blickrichtung angreifen. Weitere Eingabemöglichkeiten bestehen, indem ein Spieler zwischen seinen Figuren wechselt kann, sowie seine Runde beendet.

Die Spielereingaben müssen sowohl im Modell, als auch im 3D-Spielfeld bearbeitet und aktualisiert werden.

Zum Betrachten des Spielfeldes ist auch eine Interaktion mit diesem möglich, welche jedoch nicht durch die eigentliche Spielsteuerung gehandhabt wird, sondern direkt von der Java3D Klasse. Hierbei stehen das Zoomen, Rotieren der X-Achse sowie die Translation des Spielfeldes zur Auswahl.

1) - Stern (*) entspricht dem jeweiligen Kartennamen, z.B. „The Castle.omp“

c. Bearbeitung und Überprüfung der Spieleraktionen

Vor jeder Bewegung wird deren Zulässigkeit überprüft. Hierbei muss beachtet werden, dass sich die Figur nicht durch Wände bewegt, über andere Figuren geht, oder den Spielfeldrand überschreitet. Auch darf sie sich nicht mehr bewegen, wenn deren Bewegungspunkte aufgebraucht sind sowie, den originalen Regeln konform, sie sich bereits vor einem Angriff bewegt hat.

Ähnliche Abfragen sind auch bei einem Angriff zu tätigen. Befindet sich auf dem Zielfeld eine Figur und ist diese feindlich gesonnen, blickt die angreifende Figur über den Spielfeldrand hinaus, wodurch das Zielfeld nicht existent wäre und kann die Figur abhängig ihrer Punkte noch angreifen?

Ein weiterer, nicht zu vernachlässigender Punkt war die Entdeckung neuer Bereiche durch Heldenfiguren. Diese können unentdeckte Räume und Flure betreten und die darin enthaltenen Figuren und andere Objekte auf der Karte sichtbar machen. Diese neu entdeckten Objekte müssen korrekt auf das Spielfeld übertragen werden und falls es sich um Figuren handelt, dem jeweiligen Spieler neu hinzu gewiesen werden, welche dann in kommender Runde für diesen nutzbar sind.

Bei Beendigung einer Runde wird der nächste Spieler ermittelt, die Bewegungspunkte von dessen Figuren ausgewürfelt und nach einem Durchlauf aller Spieler die Rundenzahl erhöht.

d. Aktualisierung des Spielfeldes und der restlichen GUI

Zu allen Aktionen des Spielers gehört stets eine Aktualisierung der GUI und wenn nötig auch die des Spielfeldes.

Spielfeldaktualisierungen können statt finden, wenn Interaktionen mit den Figuren ausgeführt werden. Dazu gehören das Drehen, Bewegen, Besiegen einer Feindfigur oder dem Wechsel zwischen verschiedenen Figuren.

Die Entdeckung neuer Bereiche und das damit verbundene Aufdecken neuer Objekte und Figuren führt ebenfalls zu einer Aktualisierung des Spielfeldes.

Die Informationsfelder der GUI werden wie beim Spielfeld mit jeder Figurinteraktion aktualisiert. Die Attribute der Figur ändern sich, die aktuelle Figur wechselt oder der nächste Spieler kommt an die Reihe, was wiederum ein Figurenwechsel nach sich zieht.

5. Lösungskonzeption

Aus den in der Problemanalyse genannten Hauptfunktionen des Spiels kann eine Zusammenfassung der Komponenten erstellt werden:

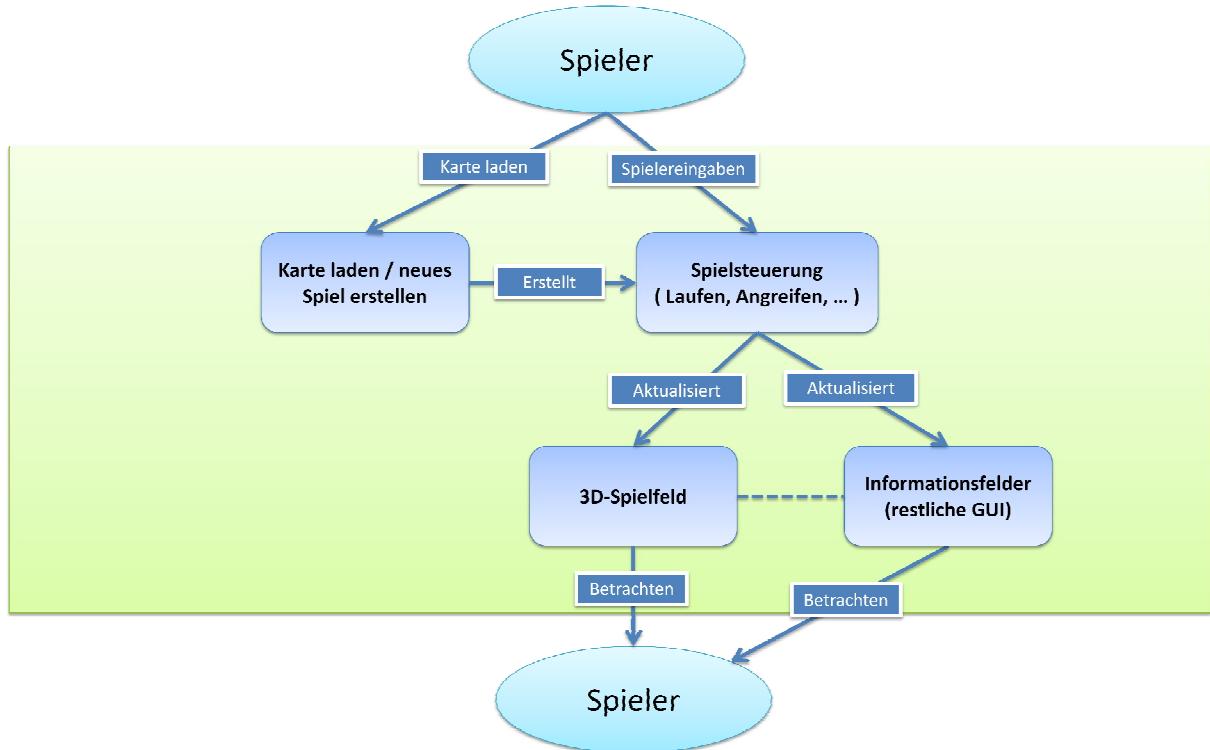


Abbildung 1 - Hauptfunktionen

Im Folgenden wird nun die Herangehensweise zu diesen Komponenten aufgezeigt.

a. Karte laden / erstellen

Wie bereits in der Aufgabenstellung erwähnt, wird kein komplexes Kartenformat zum abspeichern der Karte verwendet. Stattdessen wird ein Format auf Textbasis genutzt. Dies hat zwar zur Folge, dass nur grundsätzliche Informationen gespeichert werden können, dem gegenüber steht jedoch die Einfachheit im Umgang und Aufbau dieses Formates.

Die Karte wird somit in Drei verschiedene Ebenen untergliedert und diese werden in unterschiedliche Dateien, unterschieden durch deren Endung, abgelegt.

Zum Einlesen dieser Textdateien muss die verwendete Klasse die jeweiligen ASCII-Zeichen, welche in den Kartendateien verwendet werden können, fest hinterlegt werden. Somit sind die Möglichkeiten der zu verwendenden Objekte auf einer Karte vom Programmierer vorgegeben und können nicht (z.B. durch eine XML) extern erweitert werden. Auch die Attributwerte der Figuren sind in jener Klasse festgelegt.

Hier die drei verschiedenen Ebenen:

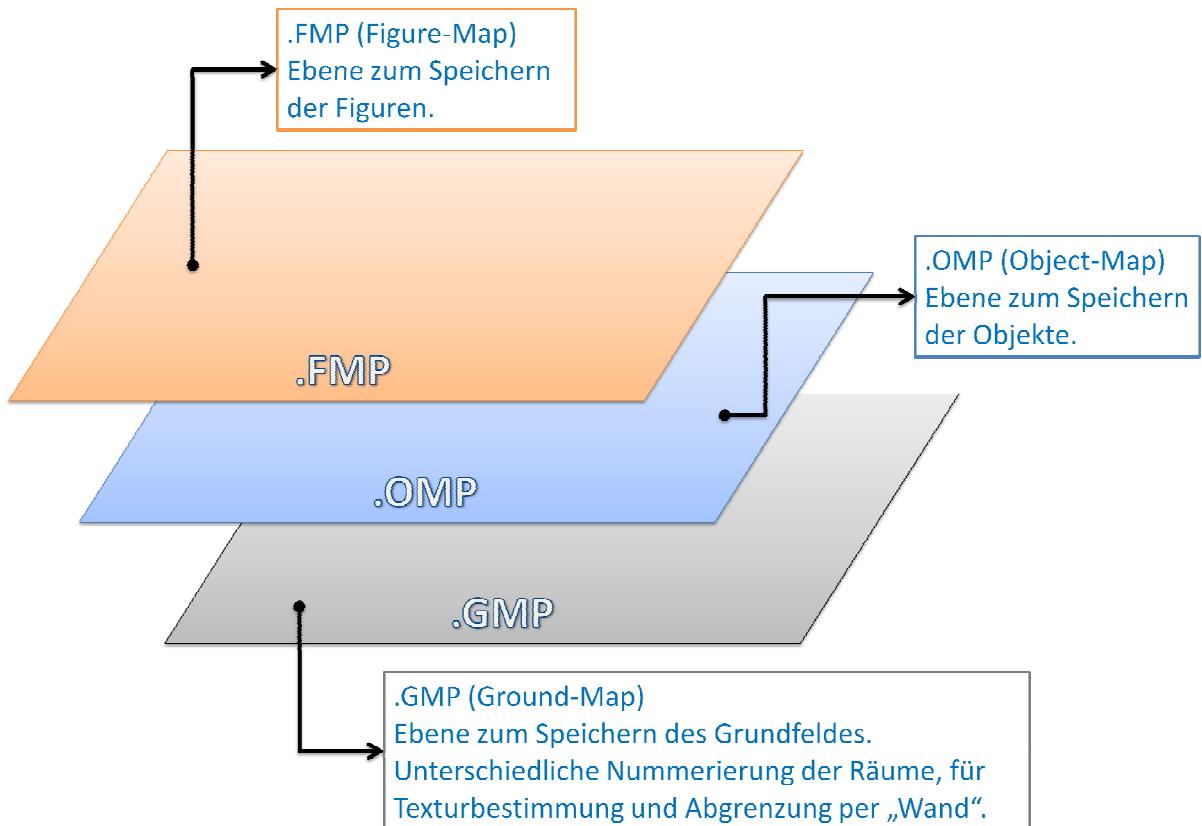


Abbildung 2 - Kartenebenen

Der Aufbau dieser Textdateien sind der angefügten Spielanleitung zu entnehmen, dadurch kann jeder Nutzer seine eigenen Karten erstellen, nach den wie eben genannt, fest vorgegebenen Objekten, Figuren und deren Attributen.

Zum Einlesen der Karte wird jede der drei Dateien zeilenweise durchgegangen und, falls die Zeichen innerhalb von doppelten Anführungszeichen („...“) stehen, die entsprechenden Informationen eingelesen. Dabei gibt die Zeile und Spalte des eingelesenen Zeichens auch die X- / und Y-Position auf dem Spielfeld an.

Verwendet werden die Modelle „field“, „mapItem“ sowie „figure“ welche den jeweiligen Kartenebenen zugewiesen werden können. Die einzige Ausnahme besteht bei „field“, da bei jedem hinzugefügten Objekt oder jeder Figur auch das entsprechende Feld aktualisiert werden muss.

Nachdem die Modelle erfolgreich eingelesen wurden, werden diese teilweise an die Spielsteuerung übergeben, um von dort aus wiederum an die entsprechenden

Komponenten weiter geleitet zu werden bzw. direkt verwendet werden. Dem Spielfeld muss die Liste der Felder sowie die am Anfang des Spiels entdeckten Figuren und Objekte, was wiederrum durch die Spielsteuerung ermittelt wird, übergeben und „gezeichnet“ werden. Die Spielsteuerung ermittelt die Startbedingungen, übergibt die nötigen Informationen an die GUI und es beginnt die erste Runde.

Die Idee dieses Kartenformates ist im Standard „tile based maps“ zu finden, welcher vor noch einigen Jahren von vielen Spielen (auch kommerziellen Titeln) verwendet wurde.

b. Spielsteuerung

In der Spielsteuerung verwendet und bearbeitet das Datenmodell des Spiels. Sie arbeitet alle Aktionen, die von Benutzer ausgelöst wurden ab und lässt die entsprechenden Aktualisierungen der GUI ausführen.

Die Instanz der Grundkarte, welche wie beschrieben zu Anfang des Spiels eingelesen wird, wird als Informationsbasis verwendet, welche bei Entdeckungen neuer Bereiche des Spielfelds und somit die Entdeckung neuer Figuren und Objekte genutzt wird. Dabei werden die entsprechenden Objekte aus der Karteninstanz in die eigenen Modellinstanzen der Spielsteuerung übernommen. Handelt es sich um Figuren, werden diese außerdem aus der Karteninstanz entfernt um eine Redundanz sowie die einfache Kontrolle über noch verfügbare Figuren zu gewährleisten.

c. Spielereingaben und Grafische Benutzeroberfläche

Spielereingaben werden in hauptsächlich einfacher Weise in Form von Knöpfen (buttons) realisiert. Je nach Aktion, werden Überprüfungen durch die Spielsteuerung angestellt und abhängig deren Ergebnisse die notwendigen Modelle sowie GUI-Komponenten aktualisiert.

Dem 3D-Spielfeld werden die überprüften, aktualisierten Informationen zugesteuert, jedoch muss es diese selber weiterverarbeiten und darauf reagieren.

Wie schon erwähnt, existiert eine Kontrolleinheit für die GUI. Diese übernimmt die Steuerung sämtlicher GUI-Komponenten und stellt das Bindeglied zwischen der Spielsteuerung und der GUI dar.

6. Programmumentwurf

Im folgenden groben Klassendiagramm sind die verwendeten Klassen dargestellt, eingeordnet in die jeweiligen Architekturschichten. Hierbei ist zu erkennen, dass die GUI-Kontrolleinheit „ViewCtrl“ die Bearbeitung aller sich in der Präsentationsschicht befindlichen GUI-Komponenten übernimmt. Die Kontrolle des Datenmodells und der GUI-Kontrolleinheit obliegt der Klasse „GameCtrl“.

Da vorerst keine Möglichkeiten vorgesehen sind, um Spielstände o.ä. zu speichern, beinhaltet die Persistenzschicht keinerlei Komponenten.

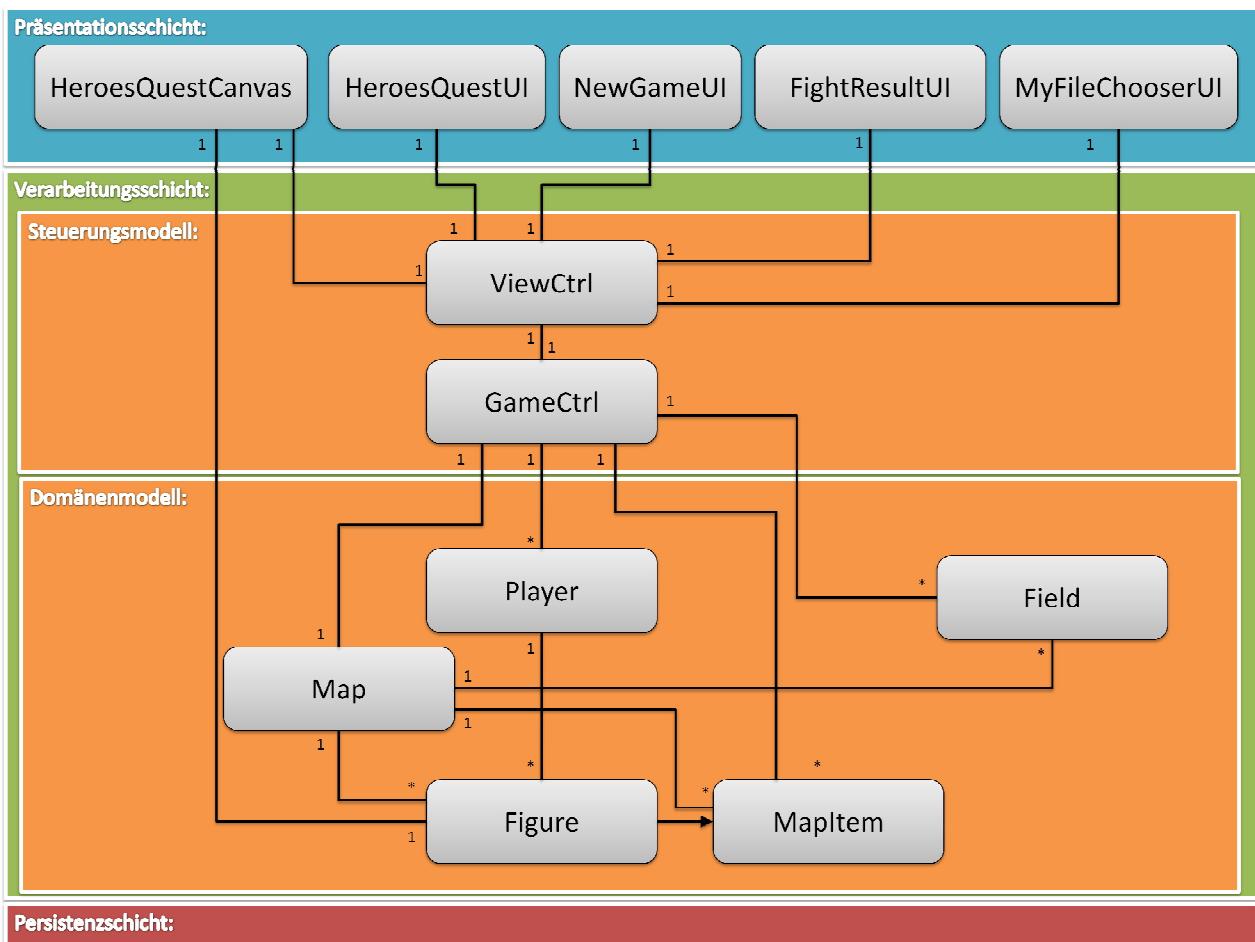


Abbildung 3 - Grobes Klassendiagramm

Der in der Klasse HeroesQuestCanvas verwendete SceneGraph ist im Folgenden dargestellt. Hierbei befindet sich wie üblich, direkt an der ersten BranchGroup „root“ die TransformGroup „tgMouse“ um die komplette Scene zu betrachten.

An der TransformGoup für die Maussteuerung befindet sich die TransformGroup „tgFloor“, welche das Spielfeld darstellt und neben einer Anzahl von TransformGroups „tgField“ auch als Elternknoten für die BranchGroups für Figuren, Objekte (mapItems) sowie Türen fungiert. An diesen befinden sich wiederrum TransformGroups für Positionseinstellungen und weiteres. Figuren haben darüberhinaus noch eine TransformGroup „tgRot“ für die Rotation bzw. Blickrichtung der jeweiligen Figur.

Als Blätter dieser Vier Zweige dienen Grundformen wie Box oder Cylinder.

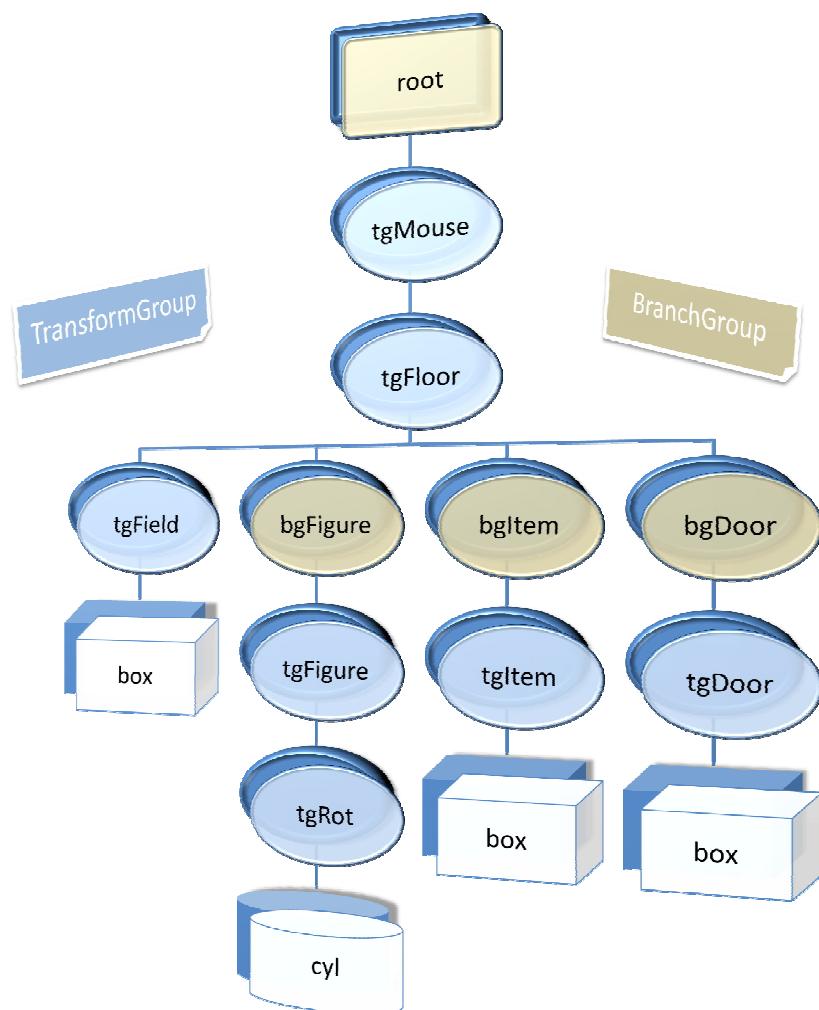


Abbildung 4 - SceneGraph

7. Implementierung

Die Implementierung wurde auf dem in der Planungsphase erstellten Klassendiagramms aufgebaut und umgesetzt. Dabei haben sich noch einige kleine Änderungen ergeben wodurch die Architektur entstand, wie sie in Abbildung 3 dargestellt ist. Dabei wurden die Klassen nach dem MVC-Modell eingeordnet:

- Datenmodell: Field, MapItem, Figure, Player, Map
- Präsentation: HeroesQuestUI, HeroesQuestCanvas, NewGameUI, FightResultUI, MyFileChooserUI
- Steuerung: GameCtrl, ViewCtrl

Jeder Klasse kann folgende Hauptfunktionen zugeordnet werden:

- Field
 - Wird zum Abspeichern von Bodeninformationen verwendet (X-Achse, Y-Achse, BodenNummer, Name der Texturdatei)
 - Zudem werden zusätzliche Informationen gespeichert (entdeckt, belegt)
 - Für die Verwaltung von Türen besitzen 2 angrenzende Felder zusätzlich ein Flag (*hasDoor*), um einen Übergang in verschiedene Räume zu ermöglichen (und eine Tür dazwischen darzustellen)
- MapItem
 - Diese Klasse stellt ein Objekt dar, welches sich auf dem Spielfeld befindet (z.B. Tisch, Schrank,...)
 - Zudem stellt sie die Elternklasse für Figuren dar, da sie sich die gleichen Grundeigenschaften teilen
 - Gespeichert werden Grundinformationen zu statischen Objekten (X, Y, Textur, X-Ausrichtung, Y-Ausrichtung, Größe in Feldern in X und Y)
- Figure (extends MapItem)
 - Stellt im Grunde ein dynamisches MapItem mit zusätzlichen Figur typischen Attributen dar (Lebenspunkte, Bewegungspunkte-/würfel, Angriffspunkte-/würfel, Verteidigungswürfel)
 - Um die Figuren eines Spielers zu unterscheiden erhalten sie noch einen Namen
 - Zur einfacheren Unterscheidung der Parteien besitzen Figuren ein Flag *isHero*, welches bei Figuren vom Heldenspieler(n) gesetzt ist
 - Um die Regel einzuhalten, dass sich Figuren, welche sich bereits vor dem Angriff bewegten, nach dem Kampf kein weiteres Mal bewegen dürfen, erhalten die ein Flag *hasMoved*
- Player
 - Diese Klasse dient dem Abspeichern aller notwendigen Informationen eines Spielers (Liste der Figuren vom Spieler und Name des Spielers)

- Map
 - Die Klasse wird zum Abspeichern der originalen Karte verwendet und besitzt somit den kompletten Kartenninhalt (Kartename, Felder, Figuren, MapItems)
 - Speichert zudem zusätzliche Information wie Spielernamen und Spielfeldgröße
 - Die Klasse ist für das Einlesen der Kartendateien zuständig (*loadMap()*)
 - Nach dieser vorliegenden Karte wird das Spielfeld durch Aufdecken neuer Gebiete sukzessive aufgebaut -> Map-Objekt wird als Kartenbasis verwendet
 - Besitzt in Form von statischen Klassenattributen alle Informationen zu Inhalten, die auf dem Spielfeld vorkommen können, also Bezeichnungen der Objekte, Figuren und Felder in den jeweiligen Kartendateien sowie deren Attribute
 - Füllt beim Einlesen der Karte Initial alle Figurenattribute
 - Um neue Inhalte für Karten hinzuzufügen, muss diese Klasse erweitert werden
- HeroesQuestUI (extends JFrame)
 - Hauptspielfeld
 - Dient der Steuerung der Spielfiguren und enthält das Spielfeld in Form von HeroesQuestCanvas
 - Stellt neben dem Spielfeld sämtliche weitere Informationen dar
- HeroesQuestCanvas (extends JPanel)
 - Java3D Klasse welche alle Inhalte des Canvas3D's sowie alle Methoden zu dessen Steuerung enthält
 - Stellt das (visuelle) Spielfeld dar
- NewGameUI (extends JDialog)
 - Wird für das Erstellen / Laden eines neuen Spiels verwendet
 - Funktionen sind das Auswählen der Karte sowie Eingabe der Spielernamen
- FightResultUI (extends JDialog)
 - Wird nach jedem Kampf aufgerufen
 - Stellt das Ergebnis eines Kampfes dar (geworfene Würfel, Lebenspunkt, ...)
- MyFileChooserUI
 - JFileChooser mit eigen erstelltem Filter (*MyFileFilter extends FileFilter*)
 - Verwendet, um bei neuem Spiel die Drei Kartenebenen auszuwählen, um diese zu laden
- ViewCtrl (implements ActionListener)
 - Steuerungsklasse aller UI-Elemente
 - Dient als ActionListener sowie als Bindeglied zwischen GUI und Spielsteuerung
- GameCtrl
 - Spielsteuerung
 - Enthält/steuert das Datenmodell des Spiels
 - Behandelt und Kontrolliert die Aktionen des Spiels
 - Alle Aktionen, die eine Aktualisierung der GUI nach sich ziehen, werden an die Klasse ViewCtrl weitergeleitet
 - Bedient sich der Klasse Map, um neue Inhalte der aktuellen Karte hinzuzufügen

Die Spielsteuerung **GameCtrl** besitzt die extern ausführbare Methode **public static void main()**, welche ein neues Objekt dieser Klasse erstellt und die Methode zum Starten des Spiels ausführt. Nun wird das Hauptfenster **HeroesQuestUI** dargestellt, womit der Nutzer ein neues Spiel im Menü laden kann. Dazu wird der Dialog **NewGameUI** geöffnet, mit welchem man in Verbindung mit dem eigen erstellten FileChooser, **MyFileChooserUI** eine Karte auswählen kann. Nach Abschließen dieser Aktion ließt die Klasse **Map** alle Drei Dateien Zeichen für Zeichen und fügt die Information der Liste der entsprechenden Modelle hinzu. Nach Abschluss des Lesevorgangs beinhaltet das **Map**-Objekt, welches die Klasse **GameCtrl** besitzt, eine Liste aller Felder (**Field**), Figuren (**Figure**) und Objekte (**MapItem**), sowie Spielernamen, Kartename und Ausbreitung der Karte in X-/Y-Achse.

Die Spielsteuerung holt sich nun von der Klasse **Map** die Heldenfiguren, setzt diese auf deren Startposition und erkundet bei jedem Setzen den jeweiligen Raum oder auch Flur. Diese Figuren werden in die Figurenliste des jeweiligen Spielers hinzugefügt und aus der Liste vom **Map**-Objekt entfernt. Derzeit ist die Nutzung von nur Zwei Spielern (Gut und Böse) möglich, jedoch sind die Methoden und Funktionen der Spielsteuerung so angelegt, dass sie dynamisch mit einer größeren Anzahl von Spielern umgehen können sollten.

Nach dem setzen aller Heldenfiguren und der entdeckten Objekte startet die erste Runde. Der Heldenspieler beginnt und die erste Figur in dessen Figurenliste wird aktiv gesetzt. Bei jeder zulässigen Interaktion des Spielers, welche eine Änderung im Spiel ergibt, wird die GUI aktualisiert. Diese Änderungen werden von **GameCtrl** berechnet und im Modell vorgenommen und anschließend an **ViewCtrl** weitergeleitet, welche die Änderungen an den entsprechenden GUI-Elementen vornimmt. Bei Wechsel der aktiven Figur wird das Objekt dieser in der Figurenliste vom Spieler gespeichert, um veränderte Attribute zu übernehmen und dann je nach Richtung die nächste oder vorherige Figur in der Liste des aktiven Spielers ermittelt und aktiv gesetzt.

Wird die Runde beendet, ermittelt die Spielsteuerung den nächsten Spieler durch Inkrementieren des dafür verwendeten Zählers und der nächste Spieler aus der Liste von **Player** bzw. der erste wird ausgewählt und dessen erste Figur aktiv gesetzt. Zudem werden bei jedem Rundenwechsel, wie auch zu Spielbeginn alle Bewegungspunkte der verfügbaren Figuren ausgewürfelt. Besitzt ein Spieler momentan keine Figuren, nicht möglich bei Heldenspieler(n), setzt er solange aus, bis wieder Figuren von ihm entdeckt wurden.

Um Figuren zu bewegen oder gegnerische anzugreifen, muss die jeweilige Blickrichtung gewählt werden und die entsprechende Aktion ausgeführt werden. Auch hier wird vor dem ausführen der Aktion von **GameCtrl** die Zulässigkeit dieser Operation überprüft, ob keine Grenzen von Listen überschritten werden usw.

Das Spiel besitzt keine Abbruchbedingung, was jedoch nicht für notwendig empfunden wurde. Der Spieler, der zuerst alle Figuren verloren hat, ist besiegt, was durch eine Meldung mittels **JOptionPane** angezeigt wird. Überprüft wird dabei nach jeden Angriff, bei dem eine Figur besiegt wird, ob der angegriffene Spieler noch verfügbare Figuren besitzt. Falls das nicht der Fall ist, wird im **Map**-Objekt nachgeschaut, ob sich dort noch Figuren des Spielers befinden, welche bisher nicht entdeckt wurden.

8. Erweiterungsmöglichkeiten und Fazit

Durch die begrenzt zur Verfügung stehende Entwicklungszeit für dieses Projekt und der Tatsache, dass es sich um eine 1-Mann-Arbeit handelt, mussten wie schon anfangs erwähnt, Vereinfachungen gegenüber dem originalen Brettspiel in Kauf genommen werden. Auch mussten Abstriche in einigen anderen Punkten des Spiels, wie z.B. des Bedienkomforts gemacht werden. Durch diese notwendigen Vereinfachungen zusammen mit weiteren Verbesserungsmöglichkeiten, können einige mögliche Erweiterungen vorgenommen werden:

- Reichhaltigkeit der Karteninhalte entsprechend dem Original oder darüber hinaus aufstocken
- Rollenspielaspekte hinzufügen, wie dem Erhalt von Erfahrung oder neuer Ausrüstung
- Mit Einführung von Rollenspielaspekten wäre die Möglichkeit der Speicherung von Spielständen und/oder Charakteren sehr sinnvoll
- Verbessertes Kartenformat, was eine leichtere Generierung sowie die Speicherung komplexerer Karteninformationen ermöglicht
- Bessere Objekt- und Figurendarstellung durch Verwendung von komplexen 3D-Modellen mittels *File Loader*
- Einführen von Animationen bei Bewegungen, Kämpfen oder Neuerkundungen
- Mehr Interaktionsmöglichkeiten mit dem Spielfeld, durch auswählen der gewünschten Figur oder dem Zielfeld
- Wegfindungssystem, um mehrere Felder automatisch überschreiten zu können
- Künstliche Intelligenz für Computergegner und Einspielerparien
- Netzwerkkompatibilität
- Soundausgabe

Die aktuelle Version des Spiels erinnert noch stark an ein Grundgerüst, welches zu erweitern gilt. Jedoch ist es innerhalb der Zwei monatigen Entwicklungszeit, welche nur durch ein zusätzliches Semester ermöglicht werden konnte, zu einem durchaus spielbaren Programm geworden. Durch den Mangel an Entwicklungszeit ist natürlich auch die Testphase für Stabilität, Fehlerbehebung und dem Ausbalancieren eines ausgeglichenen Schwierigkeitsgrades viel zu kurz gekommen.

Der Rahmen der Belegarbeit wurde mit diesem Projekt bei weitem überschritten. Jedoch war es nicht nur das Ziel, den Beleg abzuschließen, sondern auch ein etwas komplexeres Spiel zu entwickeln, welches einige Punkte der Spieleentwicklung beinhaltet, wie z.B. das finden eines geeigneten Kartentyps etc. Es hat auf der einen Seite Spaß gemacht, war jedoch auch eine zuweilen recht große Herausforderung, da bisher weder Erfahrung in diesem Bereich bestand, noch die Kommunikation zur Problemlösung mit einem Team/Teammitglied gegeben war.

9. Literaturverzeichnis

Köhler, P. D.-D.-I. (2008). Vorlesungs- und Praktikumsstoff - Grafiksysteme. Mittweida.

Steinke, L. (2003). *Spieleprogrammierung*. Landsberg: bhw.

Strasser, C. S. (kein Datum). Java 3D - Ein Überblick der API.

SUN. (kein Datum). *Java 3D API*. Von
<http://java.sun.com/javase/technologies/desktop/java3d/> abgerufen

SUN. (kein Datum). *Java SE APIs*. Von <http://java.sun.com/javase/reference/api.jsp>
abgerufen

Ullensboom, C. (2009). *Java ist auch eine Insel (8. Auflage)*. Galileo Computing.

10. Abbildungsverzeichnis

Abbildung 1 - Hauptfunktionen	6
Abbildung 2 - Kartenebenen.....	7
Abbildung 3 - Grobes Klassendiagramm	9
Abbildung 4 - SceneGraph.....	10

11. Selbstständigkeitserklärung

Mit dieser Unterschrift erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt haben.

Lars Denkewitz