Fachhochschule Nordwestschweiz
Hochschule für Technik

Algorithmen und Datenstrukturen 2

Prof. Dr. C. Stamm christoph.stamm@fhnw.ch Tel.: 056 462 47 44

Weihnachtsübung

Ausgabe: KW 51

Abgabe: KW 3

1 Mühle

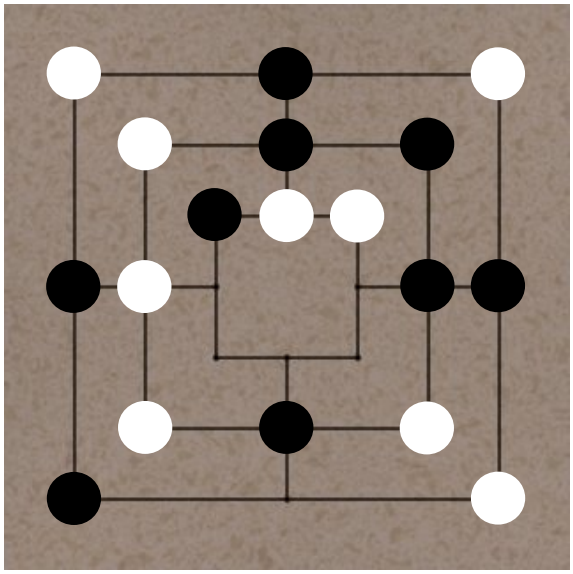
Die vorliegende Gruppenarbeit dient als Abschlussarbeit der beiden Module „Algorithmen und Datenstrukturen 2“ und „Objektorientiertes Programmieren 2“. Sie wird Ihnen ermöglichen, das Gelernte aus der objektorientierten Programmierung und der Welt der Algorithmen und Datenstrukturen einzubringen. Sie wird wohlwollend bewertet, sofern offensichtlich ist, dass Sie sich wirklich mit der Materie auseinandergesetzt und allen Code selbständig erstellt haben. Plagiate oder Plagiatversuche werden geahndet und erschweren somit in hohem Masse den erfolgreichen Abschluss des Moduls.

2 Aufgaben

Das Hauptthema dieser Übung sind Spielbäume in einem Zweipersonenbrettspiel. Sie programmieren in einer Gruppe von drei Personen Teile des Zweipersonen-Brettspiels Mühle. Ihre Implementierung soll über eine grafische Benutzeroberfläche verfügen, korrekte Spielzüge ausführen und einen Computerspieler simulieren können.

Die grossen Arbeitsbereiche für diese Gruppenarbeit sind ab Abschnitt 2.3 aufgeführt. Nicht alle sind gleich schwierig und benötigen gleich viel Einsatz. Bilden Sie Ihre Teams so, dass alle Teammitglieder einen signifikanten Beitrag leisten können.

2.1 Spielregeln



Für das Spiel benötigt man 2x9 Spielsteine in zwei verschiedenen Farben (z.B. weiss und schwarz). Das Spiel gliedert sich in drei Phasen: Zunächst ist das Spielfeld leer und die beiden Spieler dürfen abwechselnd einen Stein auf einen freien Schnittpunkt der Linien setzen. Üblicherweise beginnt Weiss. In der zweiten Phase darf abwechselnd jeweils ein Stein entlang der eingezeichneten Linien um ein Feld weiter ziehen. Sobald ein Spieler nur noch drei Steine besitzt, darf er „springen“; er darf einen eigenen Stein dann frei auf dem Spielfeld bewegen.

Ziel des Spiels ist es, durch geschickte Positionierung der eigenen Steine, eine „Mühle“ zu bilden, d.h. drei eigene Steine in einer Reihe (entlang der eingezeichneten Linien) anzuordnen. Gelingt dies einem Spieler, darf er einen gegnerischen Stein „schlagen“, d.h. vom Spielfeld entfernen. Gelingt es ihm, gleich zwei Mühlen auf einmal zu schliessen, so darf er dennoch nur einen gegnerischen Stein entfernen. Dabei ist wichtig, dass sich der zu entfernende Stein nicht in einer gegnerischen Mühle befindet. Von dieser Regel gibt es eine Ausnahme: besitzt der Gegner nur noch drei Steine, so werden alle drei Steine auf einmal geschlagen, unabhängig davon ob sie in einer Mühle angeordnet sind oder nicht.

Das Spiel ist zu Ende, sobald ein Spieler keine Steine mehr besitzt oder keinen legalen Zug mehr ausführen kann. Dieser Spieler hat die Partie verloren. Das Spiel endet unentschieden, sobald sich beide Spieler destruktiv verhalten und den gleichen Spielzustand gewollt wiederholen.

Diese Spielregeln müssen in Java umgesetzt werden. Sie sorgen dafür, dass sowohl der menschliche Spieler als auch der Computerspieler nur legale Züge ausführen.

2.2 Software-Aufbau

In Abbildung 1 sehen Sie einen möglichen Aufbau des Spielprogramms (ohne `GameClient`). Sie sehen die klare Trennung zwischen Benutzeroberfläche (GUI, violett), der Steuerung des Spielablaufs (`Controller`, blau) und der Modellierung des Spiels (rot). Benutzereingaben werden von der Benutzeroberfläche entgegengenommen, an den Spielkontroller übermittelt, dort auf Korrektheit überprüft und dann ans Spielmodell übergeben um den Spielzustand zu aktualisieren und eine Reaktion zu bewirken. Die Reaktion wird dann wieder via Spielkontroller ans GUI übermittelt und visualisiert. Im Spielmodell gehen wir von einem Spielbaum aus, welcher alle möglichen Spielzustände abbilden kann.

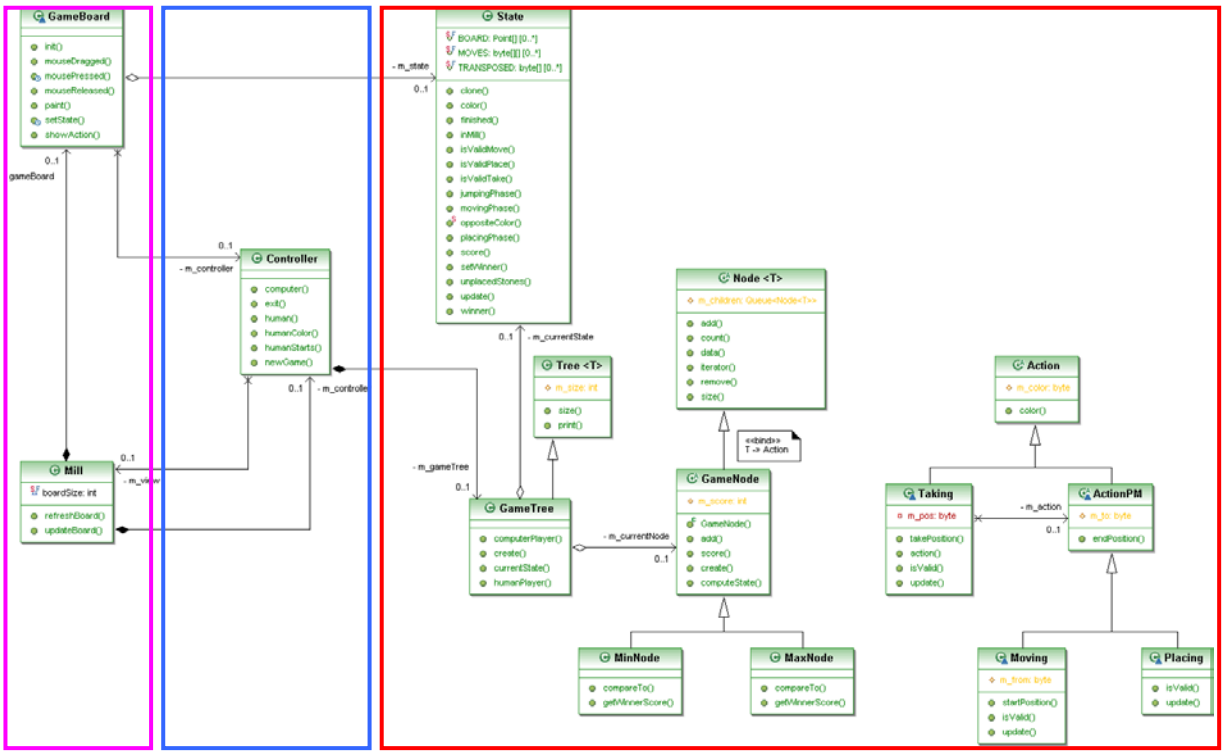


Abbildung 1: UML-Diagramm von Mühle mit GUI (violett), Controller (blau) und Modell (rot)

Eine komplette Implementierung des ganzen Spiels inklusive graphischer Benutzeroberfläche würde den Rahmen der Übung wohl sprengen. Aus diesem Grund erhalten Sie ein Grundgerüst der Software und konzentrieren sich auf die Implementierung der fehlenden Teile. Die fehlenden Teile sind nachfolgend aufgelistet.

2.3 GUI (aufwändig)

Bei der Gestaltung des GUIs sind Sie weitgehend frei. Selbstverständlich muss das Spielbrett und der aktuelle Zustand des Spiels dargestellt werden. In der ersten Spielphase werden die Steine durch Klicken auf die Positionen gelegt. Dabei sollten noch zu legende Steine ausserhalb des Spielfelds ersichtlich sein. In der zweiten Phase werden die Steine entlang der eingezeichneten Linien verschoben.

Beim Platzieren der Steine des Computerspielers ist es ratsam, den veränderten Stein über eine längere Zeit visuell hervorzuheben (z.B. blinkend, nachglühend), um dem menschlichen Spieler genügend Zeit zu geben, die neue Spielsituation zu erfassen.

Folgende Aktivitäten müssen durchgeführt werden können:

- Spielfeld und Spielsteine im Spielfeld und ausserhalb darstellen
- Spielsteine setzen, bewegen und schlagen (entfernen)

Achten Sie in erster Linie auf eine einfache, funktionale Benutzerschnittstelle. Gehen Sie nach der Methode des schrittweisen Verfeinerns vor, so dass Sie möglichst schnell eine funktionstaugliche Variante haben, die nachträglich noch verbessert werden kann.

2.4 Spielbaum (schwierig)

In einem Spielbaum werden die gespielten und die möglichen Folgezüge aufgelistet. Ein solcher Spielbaum ist ein Vielwegbaum, weil meist viele verschiedene Züge zur Auswahl stehen. Die Levels des Vielwegbaums können eindeutig dem Computerspieler bzw. dem menschlichen Spieler zugeordnet werden. Wenn wir von einer Spielsituation ausgehen, wo der Computerspieler an der Reihe ist, so listet er alle erlaubten Züge als Kinder des aktuellen Knotens auf. Für jeden dieser Blattknoten listet er anschliessend die möglichen Züge des menschlichen Spielers auf. All diese neuen Blattknoten liegen wiederum auf dem gleichen Level. Danach wiederholt sich das gleiche Prozedere für die möglichen Züge des Computerspielers usw., bis einzelne Spielzüge wegfallen, weil das Spiel zu Ende ist.

Üblicherweise wird nicht der ganze Spielbaum aufgebaut, weil er viel zu gross ist und es somit viel zu lange dauern würde, ihn aufzubauen (vor etlichen Jahren hat ein Doktorand der ETH Zürich den kompletten Spielbaum von Mühle in seiner mehrjährigen Dissertation aufgebaut). Der Baum wird also nur partiell aufgebaut, z.B. nur bis zu einer gewissen Tiefe oder an interessanten Stellen tiefer als an anderen. Auf jeden Fall müssen alle Blattknoten (d.h. die repräsentierten Spielzustände) bewertet werden, um anzugeben, ob es sich dabei um für den Computerspieler günstige oder ungünstige Spielsituationen handelt. Dazu dient eine separat entwickelte Bewertungsfunktion (siehe unten).

Die Klasse `GameTree` mit der zugehörigen Klasse `GameNode` sollen die Interfaces `IGameTree` bzw. `IGameNode` implementieren und dabei folgende Funktionalitäten anbieten:

- Spielbaum erzeugen, nachführen und partiell erweitern
- den nächsten Zug des Computerspielers bestimmen
- evtl. Branch-and-Bound-Idee realisieren (z.B. durch Min-Max-Prinzip)

Sie dürfen die Klasse `GameTree` auf der mitgelieferten Klasse *Tree* mit dem generischen Datentyp `Action` aufbauen. Anstatt in jedem Knoten des Baumes den entsprechenden Spielzustand mit einer Instanz der Klasse `State` abzuspeichern, ist es sinnvoller und speichereffizienter, nur die inkrementellen Veränderungen zwischen den Spielzuständen, d.h. die ausgeführten Aktionen (`Actions`) abzuspeichern. Um den Spielzustand in einem Blattknoten zu erhalten, ist es dann erforderlich, den aktuellen Spielzustand zu kopieren (`State.clone()`) und die entsprechenden `Actions` in der richtigen Reihenfolge auf den kopierten Spielzustand anzuwenden (`State.update(…)`).

2.5 Bewertungsfunktion (raffiniert)

Wie bereits angetönt, dient die Bewertungsfunktion zur Bewertung eines Spielzustandes (Blattknoten im Spielbaum). Es werden zum Beispiel die eigenen und die gegnerischen Steine gezählt, deren Lage analysiert und das Gewinnpotential abgeschätzt. Diese Bewertungsfunktion dient also dazu, um gute von schlechten Spielzuständen zu unterscheiden. Üblicherweise gibt die Bewertungsfunktion einen ganzzahligen Wert zurück, wobei ein hoher positiver Wert ausdrückt, dass der weisse Spieler im Vorteil ist, und ein hoher negativer Wert für Vorteile beim schwarzen Spieler steht. Je höher der Absolutwert, desto grösser der Unterschied zwischen den beiden Spielern in der jeweiligen Situation. Ist zum Beispiel ein Blattknoten ein Siegeszustand für den weissen Spieler, so sollte die Bewertungsfunktion den grösstmöglichen positiven Wert zurückgeben.

In einem Spiel, wo der vollständige Spielbaum aufgebaut werden kann, muss die Bewertungsfunktion lediglich feststellen, in welchem Blattknoten welcher Spieler gewonnen hat. Diese Aufgabe ist meistens einfach. Die Aufgabe wird jedoch bedeutend schwieriger, wenn auch nicht Endspielzustände analysiert werden müssen, weil es zu lange dauern würde und zu viel Speicherplatz benötigte, den ganzen Spielbaum aufzubauen. Umgekehrt heisst das, je besser Ihre Bewertungsfunktion arbeitet, desto geringer darf die Höhe des entwickelten Spielbaums sein und desto weniger Blattknoten müssen bewertet werden.

Die Bewertungsfunktion `score()` ist Teil der Klasse `State`. Die Klasse `State` repräsentiert einen Spielzustand vollständig. Die Differenz zwischen zwei benachbarten Spielzuständen wird durch eine Instanz der Klasse `Action` repräsentiert.

Links

So gewinnt man Mühle: http://hubbie.de/mu_html/mu1.html

Mühlespiel-Applet: http://mitglied.lycos.de/sandkasten/index.html

Branch-and-Bound: http://de.wikipedia.org/wiki/Branch-and-Bound

Min-Max-Prinzip: http://www-lehre.inf.uos.de/~ainf/2007/skript/node82.html