Schriftliche Hausarbeit zur Abschlussprüfung der erweiternden Studien für Lehrer im Fach Informatik

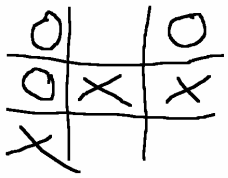
VII. Weiterbildungskurs in Zusammenarbeit mit der Fernuniversität Hagen

Eingereicht dem Amt für Lehrerausbildung -Außenstelle Gießen -

Vorsitzender des Prüfungsausschusses: Jungermann

TicTacToe

Ein Spiel begleitet den Informatikunterricht der Sekundarstufe II



Verfasser: Alexandra Horn

Gutachter: Dr. Rolf Zimmermann

Inhalt

0. Einleitung

Ausgangspunkt für die schriftliche Hausarbeit ist die Idee, dass im Projektunterricht der Sekundarstufe II des Informatikunterrichts ein Produkt immer im Mittelpunkt stehen kann und dieses Erzeugnis im Laufe der Halbjahre verbessert bzw. erweitert werden soll. Der Vorteil ist, dass ein bereits einmal erstelltes Produkt nicht in Vergessenheit gerät, sondern immer wieder aufgegriffen wird. Ein Nachteil kann darin liegen, wenn die Schüler ihren Programmtext nicht gut kommentieren. Was zur Folge hat, dass sie nach einigen Monaten ihr eigenes Werk nicht mehr verstehen. Wobei selbst dann ein großer Lernfortschritt eintreten würde, denn manch ein Schüler lernt vermutlich nur auf diese Weise, dass es tatsächlich Sinn macht, Programme sorgfältig zu planen und zu dokumentieren. Als Inhalt für dieses Vorgehen eignet sich ein Spiel, welches unter anderem

- zunächst als reine Bedienoberfläche zu programmieren ist,
- erweiterbar als Netzspiel ist,
- nicht zu einfach und nicht zu komplex ist, so dass jeder Schüler genug Möglichkeiten zur Programmierung findet und schließlich
- durch Strategien ein Spiel „Mensch gegen Computer“ ermöglicht.

Welche Voraussetzungen ein Spiel noch haben muss, um sinnvoll im Informatikunterricht eingesetzt werden zu können, und warum dies auf das von mir gewählte Spiel TicTacToe zutrifft, werde ich im Weiteren erläutern.

Ich habe mich entschieden, die Programme in Java zu erstellen. Natürlich wären sie auch mit Hilfe einer anderen objektorientierten Sprache programmierbar gewesen. Gegenüber der integrierten Entwicklungsumgebung von Delphi bietet Java meiner Ansicht nach besonders die folgende Vorteile:

- im Quellcode steht nur das, was auch programmiert wurde (allerdings kann man auch für Java schon integrierte Entwicklungsumgebungen bekommen, wenn man dies wünscht),
- die objektorientierten Dogmen werden streng eingehalten, so dass die Prinzipien der objektorientierten Programmierung umgesetzt werden können,
- der Java-Editor von Herrn Röhner, der gut für den Unterricht geeignet ist, ist kostenlos[1],
- Java ist plattformunabhängig, so dass die erstellten Anwendungen unter jedem Betriebssystem laufen.

[1] http://www.bildung.hessen.de/abereich/inform/skii/material/java/editor.htm

1. Das Spiel

1.1. Die Spielbeschreibung von TicTacToe

TicTacToe ist ein bekanntes Positionsspiel, zu dem man gewöhnlich nur einen Stift und ein Stück Papier benötigt. Es gehört zu den ältesten Spielen, dies belegen Funde von griechischen Vasen und ägyptischen Grabmalereien. Charles Babbage[2] hat sechs unterschiedliche Tic-Tac-Toe-Maschinen entwickelt.[3]

Das Spielfeld des Partnerspiels besteht aus neun Feldern, die quadratisch (3x3) angeordnet sind. Die Spieler[4] markieren abwechselnd eines der noch freien Felder mit einem Kreuz (X) bzw. Kringel (O). Gewonnen hat, wer drei X bzw. O in einer Zeile, Spalte oder Diagonale – auch Mühle genannt - gekennzeichnet hat. Sind alle Felder belegt und keiner der Spieler hat gewonnen, so endet das Spiel unentschieden.

Demnach gehört dieses Spiel nach Rüdiger Baumann[5] zu den endlichen deterministischen Zweipersonen-Nullsummenspielen mit alternierendem Zugrecht und vollständiger Information. Es werden die sieben Forderungen an ein solches Spiel erfüllt:

1. Es spielen genau zwei Spieler miteinander.
2. Es gibt nur endlich viele Spielstellungen, wobei die Anfangsposition das leere Spielfeld ist.
3. Je nach gegebener Spielstellung stehen dem Spieler verschiedene Züge offen. Nach einem durchgeführten Zug erhält man eine neue Spielstellung.
4. Es wird abwechselnd gezogen.
5. Spätestens nach neun Zügen hat ein Spieler gewonnen, oder das Spiel endet unentschieden.
6. Es gibt keine Zufallselemente, d. h. nur die Spieler selbst entscheiden über den Spielverlauf.
7. Beide Spieler sehen immer das Spielfeld.[6]

1.2. Mögliche Spielvarianten für den Unterricht

Das Spiel bietet gerade dadurch, dass die Spielregeln und der Spielaufbau einfach sind, viele Möglichkeiten es im Informatikunterricht einzusetzen. Um eine erste lauffähige Variante zu

[2] Charles Babbage (1792 – 1871) gilt als Vater des Maschinenrechners, da er bereits 1834 eine Rechenmaschine mit Speicher konstruierte (vgl. Knopp, das neue Computerlexikon, S. 30)
[3] vgl. Baumann, Strategiespiele, S. 43
[4] Ich verzichte auf die doppelte Nennung von Spieler und Spielerin, Lehrer und Lehrerin, Schüler und Schülerin etc. in meiner Arbeit. Es sind selbstverständlich immer sowohl männliche als auch weibliche Personen gemeint.
[5] vgl. Baumann, Strategiespiele, S. 7
[6] vgl. Baumann, Strategiespiele, S. 7

programmieren, benötigen die Schüler Grundkenntnisse der objektorientierten Programmierung. Diese Variante lässt sich jedoch für bessere Schüler schnell erweitern, so dass alle Schüler zu einer spielbaren Lösung kommen können und keiner sich langweilen muss, weil er schneller fertig ist. Außerdem bietet das Spiel die Möglichkeit, dass alle Schüler am gleichen Spiel arbeiten, wenn zuvor entsprechende Absprachen getroffen worden sind. Auf die Einsatzmöglichkeiten werde ich im 3. Kapitel eingehen.

Als 1. Vorversion bietet es sich an ein 3 x 3- Spielfeld zu programmieren, in dem jedes Feld durch Anklicken in ein X umgewandelt werden kann. In der 2. Vorversion sollen dann zwei Spieler abwechselnd das Spielfeld bedienen müssen, so dass jeweils ein X bzw. O gesetzt wird. Bereits bei den Vorversionen sollte darauf geachtet werden, dass ein Konzept erstellt wird, so dass die Versionen leichter erweitert werden können. Hierzu gehört, dass z. B. die Bedienoberfläche in einer Extraklasse realisiert wird. Eventuell kann auch auf die Vorversionen verzichtet werden, allerdings bieten sie eine gute Grundlage, um über Schnittstellen und Klasseneinteilung zu diskutieren.

Die 1. Version des Spiels erhält man, wenn die 2. Vorversion insofern erweitert wird, dass der Computer die Rolle des Spielleiters übernimmt. Diese 1. Version kann nun durch einige Zusätze verfeinert werden, z. B.

- Namensabfrage der Spieler: Die Spieler können namentlich zum Zug aufgefordert werden. Diese Aufforderung zum Zug kann auch ohne Namensabfrage erfolgen, indem Spieler_X oder Spieler_O aufgefordert werden, ein Feld zu markieren.
- Optisch verbessertes Spielfeld: z. B. Bilder anstelle der X und O verwendet werden usw.
- Erweiterung des Spielfeldes um die 3. Dimension
- die Ausdehnung als Netzwerkspiel
- Abändern der Spielregeln, z. B. im zweidimensionalen Fall darf jeder maximal drei Felder markieren (ähnlich dem Mühlespiel) oder im dreidimensionalen Fall wird die Anzahl aller Mühlen gezählt und es gewinnt derjenige, der die meisten Mühlen erstellt hat.

Die 2. Version des Spiels erfordert von den Schülern Einblicke in das Wissensgebiet der Künstlichen Intelligenz, nämlich in den Bereich der Spieltheorie. Diese Version des Spiels kann die Möglichkeit der Wahl bieten, ob zwei Spieler oder ein Spieler mit dem Computer spielen möchten. Hierzu ist es erforderlich, dass eine gewisse Spielstrategie programmiert wird.

Erweiterbar ist diese Version durch unterschiedliche Schwierigkeitsgrade. Interessant ist sicherlich, wenn die Schüler arbeitsteilig verschiedene Spieltheorien aufarbeiten und in einem Programm umsetzen. Anschließend kann man die Computer gegeneinander antreten lassen, um zu einer Bewertung der verschiedenen Strategien zu kommen.
Eine Frage, die sich stellt, ist, ob ein Computer lernfähig ist und somit erkennen kann, dass das Spiel bei gleichwertigen Spielpartnern sinnlos ist, weil es immer unentschieden ausgehen wird. Dieser Fragestellung kann auf mehreren Wegen nachgegangen werden, die bisherige Version ohne Spielstrategie könnte entweder so erweitert werden, dass der Computer gespielte Spielzüge abspeichert und dabei bewertet, oder indem man versucht mit Hilfe einer logischen Programmiersprache das Spiel auszuwerten.
Wie diese Auflistung zeigt, kann das Spiel TicTacToe als wiederkehrendes Element in die Projektarbeit einbezogen werden. Alte Versionen können unter neuen Aspekten weiterbearbeitet werden.

2. Möglichkeiten der Programmierung

In diesem Kapitel werde ich einzelne Bestandteile der Programme erläutern. Der gesamte Quellcode aller Versionen befindet sich im Anhang und auf der beigefügten CD, so dass die Programme geladen und ausprobiert werden können. Der Quellcode ist mit vielen Kommentaren versehen, so dass es nicht all zu schwer fallen dürfte, sich dort einzuarbeiten. Dies hat auch den Zweck, dass man ein bestehendes Programm benutzen kann, um es weiter mit den Schülern zu bearbeiten. Zusätzlich sollen in diesem Kapitel die Ideen, die hinter der mitgelieferten Lösung stecken, erläutert werden.
Programmbeispiele findet man auch in Büchern, im Internet oder von der Firma Sun, jedoch sind die meisten so schlecht kommentiert, dass es größtenteils unmöglich ist, die Lösungsidee nachzuvollziehen. Aus diesem Grunde werde ich auch einige Ansätze aufzeigen, die mir zunächst sinnvoll erschienen, sich aber dann im Verlauf der Programmierung als untauglich oder unpraktisch erwiesen.

2.1 Programmierungen der Vorversionen

Sinnvoll und wünschenswert ist in der heutigen Zeit sicherlich eine Spielprogrammierung, die interaktiv ist, d. h. der Spieler soll mit der Maus ein freies Feld „anklicken“ können. Nahe

liegend ist eine Darstellung des Spielfeldes durch Buttons[7], die bei Spielbeginn mit einem Leerzeichen beschriftet sind. Zur besseren Strukturierung des Programms wird der ActionListener für die Button in einer eigenen Klasse *„ButtonListener“* realisiert.

Es ergibt sich das Problem, dass das Spielfeld mit 3 x 3 - Feldern erzeugt werden muss. Es liegt zunächst auf der Hand, dies mit einem zweidimensionalen Array mit 3 x 3 - Einträgen zu realisieren.

```
// 1. Vorversion: Erstellung eines Spielfeldes mit 3*3 Feldern
public class TicTacToe01 extends Applet {
  Button Feld[][] = new Button[3][3];
  public void init () {
    setLayout (new GridLayout(3,3));
    for (int n=0;n<3;n++) {
      for (int m=0;m<3;m++) {
        Feld[n][m] = new Button(" ");
        add(Feld[n][m]);
      }
    }
  }
}
```

Für die weitere Arbeit mit dem Programm erscheint mir dieser Lösungsansatz jedoch zu umständlich, weil sehr oft mit diesem Array gearbeitet werden muss bzw. weitere Arrays benötigt werden. Daher ist es günstiger, mit einem eindimensionalen Array zu arbeiten und dieses von 0 bis 8 durchzunummerieren, so dass in der ersten Zeile die Felder 0 bis 2 liegen usw. Diese Nummerierung muss nur bei der Programmierung berücksichtigt werden. Sie spielt für den Anwender keine Rolle.

Somit ergibt sich folgende Nummerierung der Felder:

0	1	2
3	4	5
6	7	8

Die Realisierung sieht daher wie folgt aus: Button[] spielbu = new Button[9]. Man erhält ein Feld mit 9 Buttons, auf die man über den Index zugreifen kann. Denkbar ist in diesem Stadium auch, dass die Schüler jeden Button einzeln realisieren. Es ist aber sehr schnell einzusehen, dass dieser Weg viel zu unübersichtlich und mit viel Tipparbeit verbunden ist. Vorstellbar ist auch,

[7] Sicherlich kann man auch von Knöpfen reden, allerdings bin ich davon überzeugt, dass sich der Begriff Button schon soweit in den deutschen Sprachgebrauch im Informatikunterricht eingebürgert hat, dass es eher komisch wirken würde, wenn man von Knöpfen reden würde. Zumal der Gebrauch der Sprache Java auch die Benutzung des englischen Vokabulars mit sich bringt. Entsprechendes gilt für andere Begriffe wie z. B. Array.

dass auf die Button verzichtet wird und mit einer graphischen Darstellung und dem Mouselistener gearbeitet wird.
Zusätzlich soll in dieser Vorversion durch das Anklicken eines Buttons ein X erscheinen. In der dritten Vorversion entsteht durch abwechselndes Anklicken ein X bzw. O und ein Button soll hinzugefügt werden, der ein neues Spiel startet.

2.2. Programmierung der 1. Version

In der 1. Version sollen zwei Personen am gleichen Rechner miteinander spielen können. Der Computer übernimmt die Aufgabe eines Spielleiters. Dies bedeutet, dass er nur erlaubte Spielzüge zulassen darf und nach jedem Zug überprüfen muss, ob ein Spieler bereits gewonnen hat oder das Spiel unentschieden ausgegangen ist, da neun Züge gemacht worden sind. Der Computer muss also überprüfen, ob das angeklickte Spielfeld noch leer ist, bevor der Zug ausgeführt wird.
Um dies zu realisieren, muss die 3. Vorversion um eine Methode *„gewonnen()“* ergänzt werden, die erkennt, ob bereits eine Mühle existiert. Damit das Programm feststellen kann, ob ein Remis vorliegt, genügt es eine Variable *„zuganzahl“* einzufügen, die die Anzahl der durchgeführten Züge speichert. Hat kein Spieler gewonnen und die *„zuganzahl“* wird auf neun gesetzt, so endet das Spiel unentschieden.
Die Methode *„gewonnen()“* hat einen booleschen Rückgabewert, der nur dann auf ‚true’ gesetzt wird, wenn der Spieler, der den letzten Zug vollzogen hat, gewonnen hat. Zunächst scheint es, dass die Überprüfung des zweidimensionalen Arrays (siehe 1. Vorversion) einfacher zu programmieren gewesen wäre. Allerdings reicht es nicht, die Beschriftungen der Button zu vergleichen, da diese Überprüfung sofort nach dem ersten Zug ‚true’ zurückgibt, weil in mehreren Zeilen, Spalten und Diagonalen die gleiche Beschriftung „ “ vorhanden ist. Realisiert ist diese nicht funktionierende Variante in Version 1.0.

```
//überprüfen der Spalten
 for (int i=0; i<3; i++) {
   if((spielbu[i].getLabel() == spielbu[i+3].getLabel()) &&
      (spielbu[i].getLabel() == spielbu[i+6].getLabel()))
     {return true; }
```

Denkbar ist auch ein Vergleich der Buttonbeschriftungen, bei dem zusätzlich folgende Abfrage erfolgt: spielbu[i].getLabel() == ‚X’ bzw. spielbu[i].getLabel() == ‚O’, allerdings wird der Quellcode unnötig lang und unübersichtlich. Wichtiger ist jedoch, dass auch im weiteren Verlauf diese Programmierung zu Schwierigkeiten führen wird und keine einfachen

Berechnungen möglich sind. Daher muss neben der Beschriftung der Button ein zweites Array *„spielfeld“* angelegt werden, um dort Zahlenwerte zu speichern, über die eine Berechnung erfolgt. Dies habe ich in Version 1.1 realisiert.

```
public int[] spielfeld = new int[9];

//Codierte Spielfeldeinträge
public static final int SPIELER_X = 1,
                        SPIELER_O = -1,
                        FREI = 0;
```

Die Gewinnberechnung ist mit dieser Codierung eindeutig, da ein Spieler nur dann gewonnen hat, wenn die Summe über eine Reihe[8] 3 bzw. -3 ergibt.

Generell muss man sich entscheiden, ob man das Spiel als Applet oder als Fensterapplikation programmieren will. Unter der Voraussetzung, dass die Schüler anhand des Buches „Praktische Informatik mit Java“ von Reinhold Ley objektorientierte Programmierung und Java erlernt haben, kann man davon ausgehen, dass sie zunächst das Spiel als Applet realisieren werden. Der Vorteil liegt darin, dass eine Einbindung in eine Homepage somit sehr einfach möglich ist. Die Fensterapplikation bietet jedoch auch einige Vorteile, z. B. kann man Menüs anlegen, wie sie aus Anwendungen bekannt sind, so dass ein Programm optisch „professioneller“ wirkt. Der wesentliche Unterschied zwischen einem Java-Applet, welches über das Internet verbreitet werden kann, und einer Java-Applikation liegt darin, dass die Java-Applikation ein eigenständiges Programm ist, welches auch auf die lokale Festplatte zugreifen kann. Dies spielt jedoch für die Spielprogrammierung von TicTacToe nur eine untergeordnete Rolle, d. h. für die eigentliche Programmierung ist es reine Geschmackssache, welche Form der Programmierer wählt. Entscheidend ist die Wahl nur dann, wenn man Spielstände abspeichern will. Dazu muss auf die Festplatte zugegriffen werden, so dass dies nicht über ein Applet realisiert werden kann (vgl. Version 1.2., bei der diese Punkte der Auswahlliste nicht funktionieren, während in Version 1.3. Spielstände geladen und abgespeichert werden können).

Für eine bessere Spielsteuerung ist es sinnvoll einen zusätzlichen Button einzufügen, mit dem ein neues Spiel gestartet werden kann. Hierfür ist dann die Methode *„neuesSpiel()“* zu schreiben, die den Anfangszustand des Spiels herstellt und festlegt, welcher Spieler beginnt.

[8] Im Folgenden ist mit Reihe immer eine Zeile, Spalte oder Diagonale gemeint.

```
public void neuesSpiel() {
  for (int i=0; i<9; i++) {
    spielbu[i].setLabel(" ");
  }
  spielerAmZug = 1;  //in dieser Version automatisches Setzen
                     //später frei wählbar
  zuganzahl = 0;
  meldung.setText("Los geht's");
```

Für die Optik, die eine nicht zu unterschätzende Rolle spielt (wer mag schon mit einem Spiel spielen, das nicht gut aussieht), ist es von Vorteil einen Bereich für die Überschrift vorzusehen, in dem der Name des Spiels erscheint, und einen Bereich, in dem das Programm Meldungen an die Spieler geben kann. Diese Meldungen können sich darauf beziehen, welcher Spieler am Zug ist, ob ein Spieler gewonnen hat oder zur Kontrolle des Programms, ob ein Programmierfehler aufgetreten ist. Diese Bereiche können in Java durch Panel verwirklicht werden.

```
oben = new Panel();
oben.add(ueberschrift);
oben.setBackground(Color.blue);
```

Als Beispiel sei hier das Panel für die Überschrift *„oben“* herausgegriffen. Ihm wird die extra definierte *„ueberschrift“* hinzugefügt und die Hintergrundfarbe wird auf blau gesetzt. Die verschiedenen Bereiche müssen schließlich mit Hilfe des Layout-Managers richtig angeordnet werden. Diese optischen Möglichkeiten habe ich bereits in den Vorversionen 0.2 und 0.3 verwirklicht, dies ist jedoch nicht notwendig.

Zur umfassenden Aufgabe eines Spielleiters gehört auch, dass ein Spieler auf Wunsch die Spielregeln und Informationen über den Autor abrufen kann. Dies habe ich in der Version 1.2. durch eine Auswahlliste realisiert, die mit Hilfe der Klasse *„LauscherListe2“* arbeitet.

```
hilfe = new List(2, false);
  hilfe.add("Anleitung");
  hilfe.add("Info");
  hilfe.select(0);
  hilfe.addActionListener(L2);
```

Zusätzlich ist eine Auswahlliste eingefügt worden, mit der Spielstände abgespeichert bzw. geladen werden können und das Applet geschlossen werden kann. Die Klasse *„LauscherListe1“* ermöglicht das Schreiben und Lesen von einer Datei. Das Spiel kann auch über die Auswahlliste neu gestartet werden, wobei ich den Button für benutzerfreundlicher halte. Wie bereits zuvor erwähnt, führt diese Programmierung zu Schwierigkeiten, da der Rechner auf die lokale Festplatte zugreifen muss, daher habe ich in Version 1.3. auf eine

Fenster-Applikation umgestellt, so dass diese Funktionen gegeben sind. Um ein Applet in eine Applikation umzuwandeln, muss die Methode *„init()“* in den Konstruktor überführt werden. Die Auswahllisten werden in Menüs abgeändert.

```
//Menü mit Auswahlpunkten deklarieren
MenuBar mb = new MenuBar ();
  Menu spielsteuerung = new Menu("Spiel");
  neu = new MenuItem("Neues Spiel starten");
  laden = new MenuItem("Alten Spielstand laden");
  speichern = new MenuItem("Spielstand abspeichern");
  beenden = new MenuItem("Spiel beenden");

  Menu hilfe = new Menu("Hilfe");
  regel = new MenuItem("Anleitung");
  info = new MenuItem("Info");
```

Die nun vorliegende Version erfüllt die Funktionen, die Baumann für die Spielleiter-Version fordert, denn der Computer übernimmt nur die Verwaltungs- bzw. Steuerungsfunktionen, bietet jedoch noch keine Strategie an.

Folgende Funktionen werden erfüllt:

- die Spielregeln werden auf Wunsch angezeigt,
- die Startposition und der beginnende Spieler werden zu Beginn einer Partie festgelegt,
- es wird kontrolliert, ob die Spielregeln beim Ziehen eingehalten werden und der aktuelle Spielstand wird angezeigt,
- der Ausgang einer Partie wird ausgegeben,
- es ist möglich eine neue Partie zu starten oder das Programm zu beenden.[9]

Dies bedeutet, dass bisher nur die Benutzerschnittstelle erstellt worden ist, d. h. Klassen und Methoden, die sich auf die Bildschirmgestaltung beziehen. Das eigentlich reizvolle ist sicherlich die Programmierung einer geeigneten Spielstrategie, um gegen den Computer spielen zu können. Darauf werde ich im nächsten Abschnitt eingehen.

[9] vgl. Baumann, Strategiespiele S. 9

2.3. Programmierung der 2. Version

Es gibt unterschiedliche Ansätze, mit deren Hilfe man zu einer Spielstrategie gelangen kann. Diese Ansätze werde ich kurz skizzieren und dabei auch auf Vor- und Nachteile bei der Anwendung für das vorliegende Spiel TicTacToe eingehen.

Unter einer Spielstrategie versteht man „einen vollständigen Verhaltensplan, das heißt durch die Wahl einer Strategie legt ein Spieler für jede im Verlauf der Partie mögliche Position seinen jeweils nächsten Zug fest.“[10]

2.3.1. Änderungen an der Benutzerschnittstelle

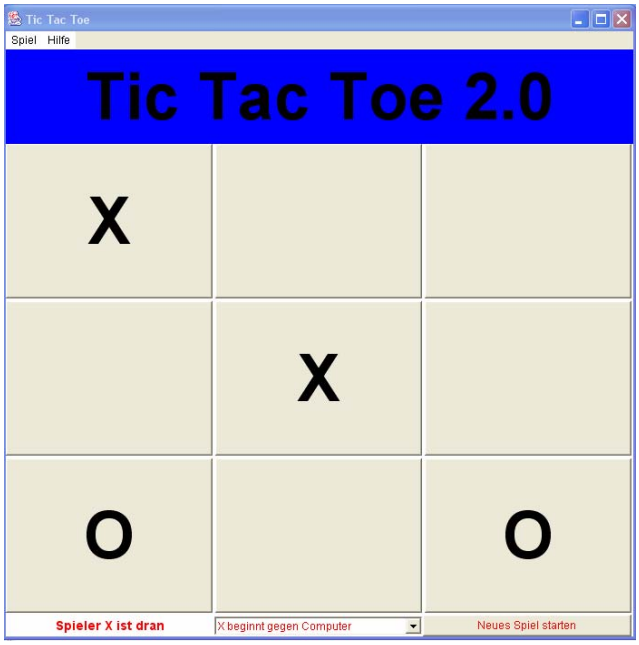
Vorab sind auch bezüglich der Bedienoberfläche einige Änderungen von Nöten. Anfangs muss der Spieler die Alternative haben, ob er gegen den Computer oder gegen einen Mitspieler antreten will und wer beginnen soll. Nachdem der Spieler diese Wahl getroffen hat, muss zunächst eine neue Partie gestartet werden. Die Wahl wird mit Hilfe eines Choice-Menü *„spielerauswahl“* realisiert.

Entscheidende Änderungen sind jedoch bei der Zugdurchführung vorzunehmen, da der Computer keinen Button drücken kann. Die Methode *„actionPerformed“* der *ButtonLauscher*-Klasse muss abgeändert werden. Zunächst wird eine Variable *„computer“* eingeführt, die auf ‚1’ gesetzt wird, wenn gegen den Computer gespielt werden soll. In der Methode wird also eine Fallunterscheidung vorgenommen. Für den Fall, dass kein Computer beteiligt ist, ändert sich nichts. Soll gegen den Computer gespielt werden, muss nach einem Zug des Spielers direkt der Computerzug aufgerufen werden, es sei denn, das Spiel ist beendet. Die Meldung „Computer ist dran“ erscheint nur, wenn ein Fehler im Programmablauf aufgetreten ist, ansonsten ist diese Einblendung aufgrund der Schnelligkeit nicht zu sehen.

```
for (int i=0; i<9; i++) {
  if (gedruekterBu == spielbu[i]) {
    if (zuganzahl <9) {
      if (spielbuBeschriftung == " ") {
      if (spielerAmZug == 1) {
         setzen(i, SPIELER_X);
         meldung.setText("Computer ist dran");
      }   //spielerAmZug == 1
      if (gewonnen()==-2) {
        computerZieht();
      }
        sieg();
```

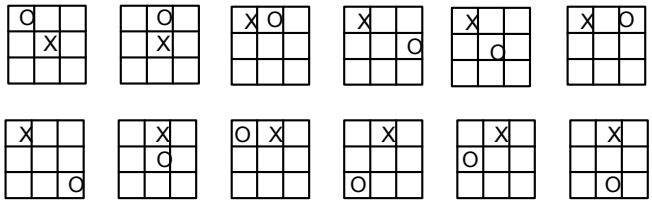
[10] Schrage, Spielstrategie, S.17

Wird eine Partie neu gestartet und der Computer soll beginnen, so muss bereits ein Computerzug in der Methode *„neuesSpiel()“* aufgerufen werden.
Wie man am Quellcode erkennt, muss noch eine Methode *„computerZieht()“* ergänzt werden. Mit Hilfe dieser Methode wird die entsprechende Strategie aufgerufen.



2.3.2. Mathematische Vorüberlegung

Das Spielfeld besteht aus neun Feldern, die drei unterschiedliche Zustände annehmen können: leer, vom Spieler_X belegt oder vom Spieler_O belegt. Somit gibt es $3^9 = 19683$ verschiedene Spielpositionen. Viele dieser Zustände entsprechen jedoch nicht den Spielregeln, denn ein Spieler kann maximal nur fünf Felder belegen, ein Spieler kann nur <u>eine</u> Mühle setzen, da dann das Spiel endet usw. Hinzukommt, dass einige Spielpositionen symmetrisch sind, so dass sie nicht doppelt betrachtet werden müssen.
Als Eröffnungszug sind drei verschiedene Züge möglich: besetzen der Mitte, einer Ecke oder der Seitenmitte. Nach dem zweiten Zug ergeben sich 12 unterschiedliche Spielpositionen:



X hat angefangen, O zieht als 2. Spieler

2.3.3. Die erste einfache Spielstrategie

Bei der Programmierung der diversen Spielstrategien kann auf das Vererbungsprinzip zurückgegriffen werden, denn – wie man im weiteren Verlauf erkennen kann – werden die alten Spielstrategien immer wieder verfeinert. Grundlage für alle weiteren Spielstrategien ist ein Computer, der nur Zufallszüge durchführt, wobei man hier kaum von einer Strategie sprechen kann. Die Klasse *„computer0“* enthält zwei Methoden, die Methode *„ziehen(TicTacToe20 ttt)“*, die die Feldbelegung vornimmt, und die Methode *„zufallszahl()“*, die mit Hilfe der mathematischen Funktion Math.random() eine Zufallszahl zwischen null und acht ermittelt.

Eine sicherlich einfache - wenn auch nicht sonderlich erfolgreiche - Spielstrategie besteht darin, dass der Computer überprüft, ob sein Gegner bereits zwei gleiche Symbole in einer Reihe besitzt und das verbliebene Feld noch frei ist, so dass der Gegner mit seinem nächsten Zug gewinnen kann. In diesem Fall blockiert der Computer dieses Feld, ansonsten wählt er per Zufall ein freies Feld aus. Diese Strategie führt dazu, dass der Computer nicht mehr so schnell verlieren kann, es sei denn der Gegner baut eine Falle.[11] Um dem Computer zusätzlich die Möglichkeit zu geben, auch zu gewinnen (wenn der Gegner es zulässt), bietet es sich an, das Programm so weiterzubearbeiten, dass der Computer erkennt, wenn er mit dem nächsten Zug gewinnen kann und dann das entsprechende Feld auswählt.

Die Programmierung dieser Strategie ist nicht sonderlich schwierig. Die Klasse *„computer1“* erbt von der Klasse *„computer0“*, allerdings muss die Methode *„ziehen(TicTacToe20 ttt)“* überschrieben werden.

```
public void ziehen (TicTacToe20 ttt){
  if ((feld = fastgewonnen(ttt)) != -1) {} //Sieg möglich
  else if ((feld = fastverloren(ttt)) != -1) {}
    //Verlieren möglich, also blockieren
    else {
      do {zufallszahl();}
      while (ttt.spielfeld[feld] != ttt.FREI);
    }
  ttt.spielbu[feld].setLabel("O");
  ttt.spielfeld[feld] = ttt.SPIELER_O;
};
```

Zwei weitere Methoden *„fastgewonnen(TicTacToe20)“* und *„fastverloren(TicTacToe20)“* werden hinzugefügt, um zu überprüfen, ob der Computer mit seinem nächsten Zug gewinnen kann bzw. ob der Gegner im nächsten Zug einen Gewinnzug setzen kann. Falls dies der Fall ist,

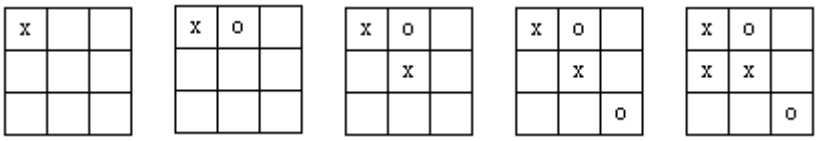
[11] Unter einer Falle verstehe ich, wenn es zwei mögliche volle Reihen gibt, so dass nur eine blockiert werden kann.

wird die entsprechende Feldnummer von der Funktion *„ziehen(TicTacToe20 ttt)“* belegt. Tritt keiner der beiden oberen Fälle ein, zieht der Computer zufällig.

2.3.4. Eine Spielstrategie aufgrund der Gewichtung der Spielfelder

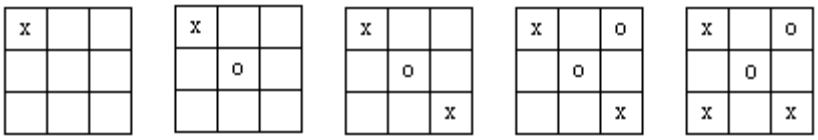
Beim Spiel können folgende kritische Situationen auftreten.

Falle

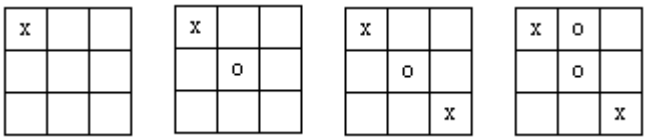


Nach dem letzten Zug können zwei Mühlen entstehen und der Gegner kann nur eine verhindern. (…) Um eine Falle zu blockieren, muss der [1.] Stein in die Ecke gesetzt werden.

Doppelfalle



Bei der Doppelfalle, wie sie in der dritten Stellung aufgebaut wurde, versagt die obige Strategie. Hier darf der [2.] Stein nicht in die Ecke gesetzt werden. Wie die unteren Bilder zeigen, muss der Stein hier in die Mitte gesetzt werden.



Bei einer Doppelfalle darf nicht das Eckfeld besetzt werden.[12]

Betrachtet man sich diese kritischen Situationen genauer, so kann man feststellen (was auch so vielen Schülern klar sein dürfte), dass das mittlere Feld besonders gut geeignet ist, um Mühlen zu erstellen, die Eckfelder wichtiger sind als die restlichen Felder. Man kann somit zu folgender Spielfeldgewichtung kommen:

2	1	2
1	3	1
2	1	2

Der bisherige zufällige Zug kann also dadurch ersetzt werden, dass der Computer zufällig ein Feld mit höchster Priorität auswählt, d. h. wenn möglich in die Mitte setzt, ansonsten eine freie

[12] www.gymnasium-wertingen.de/deutsch/fachbereiche/informatik/material/wahlkurs-java

Ecke auswählt und nur als letzte Möglichkeit in die Mitte einer äußeren Reihe setzt. Erfolgsversprechender ist es jedoch, wenn zwei Fälle unterschieden werden:

- ist beim ersten Zug des Computers Feld 4 mit höchster Priorität frei, so wird dies belegt, und im nächsten Zug wird ein Feld mit Priorität 1 ausgewählt (Vermeidung der Doppelfalle),
- ist beim ersten Zug des Computers Feld 4 belegt, so wird eine Ecke belegt.

Die Klasse *„computer2“* erbt wiederum von der Klasse *„computer1“,* denn im Falle einer möglichen Mühle soll diese natürlich gebildet bzw. verhindert werden. Die Methode *„ziehen(TicTacToe20 ttt)“* muss daher wieder überschrieben werden. Die Methoden *„ersterZug(TicTacToe20 ttt)“* und *„zweiterZug(TicTacToe20 ttt)“* müssen ergänzt werden. In diesen beiden Methoden kommt die *do-while-Schleife* zum Einsatz:

```
do {
  do { feld = zufallszahl();
  } while( feld == 1 | feld == 3 | feld == 5 | feld == 7 );
} while (ttt.spielfeld[feld]!=ttt.FREI);
```

Betrachtet man die innere der beiden Schleifen, so wird mindestens einmal eine Zufallszahl bestimmt und im Zweifel so oft, bis ein Eckfeld oder die Mitte „gezogen“ worden ist. Mit der äußeren Schleife wird verhindert, dass der Computer die Mitte belegt, da diese bereits vom Spieler_X belegt worden ist.

Um es etwas interessanter zu gestalten kann man eine Zufallskomponente einfügen, so dass nur in 80 % der Fälle bei einem freien Feld 4 dies ausgewählt wird. Allerdings führt das an dieser Stelle auch zu weit.

2.3.5. Gewinn – und Verlustpositionen

Im Verlauf einer Partie hat jeder Spieler je nach Spielsituation unterschiedlich viele Zugmöglichkeiten zur Verfügung, von denen manche günstiger andere ungünstiger für den weiteren Fortgang der Partie sind. Um alle Spielpositionen beurteilen zu können, muss man sich zunächst einen Überblick über sämtliche möglichen Spielverläufe[13] verschaffen. Dies gelingt am ehesten mit einem gerichteten Graph, in dem jede Spielposition als Knoten auftritt. Ein Weg durch den Graphen von der Anfangsposition bis zur Endposition ist somit eine Partie.[14]

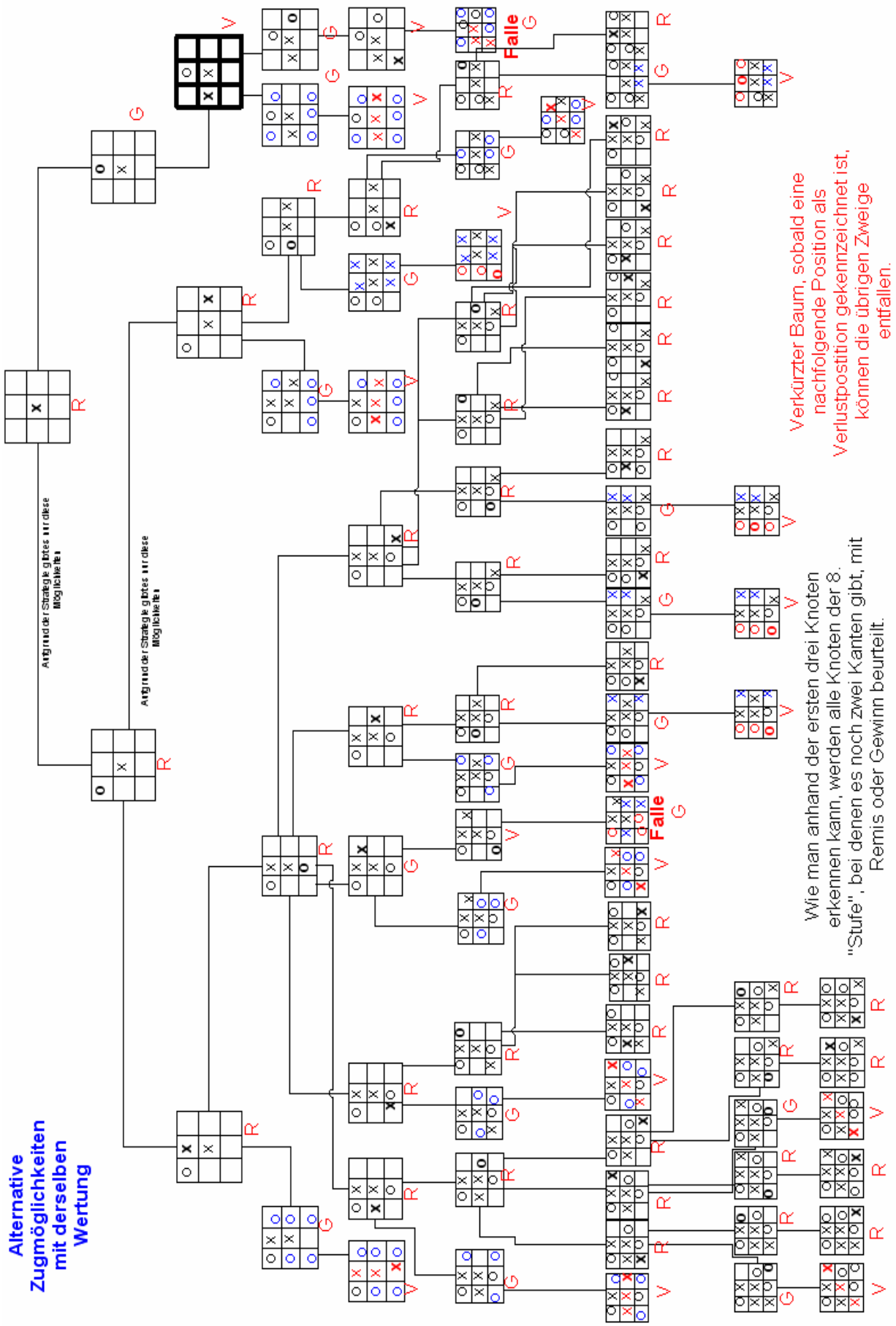
Insgesamt gelten für den Positionsgraphen folgende Regeln:

(1) Jede Position ist entweder eine Gewinn–, eine Verlust- oder Remisposition.

[13] Jeder erlaubte Spielverlauf, der von der Anfangsposition bis zur Endposition verläuft, heißt Partie.
[14] Vgl. Schrage, Strategiespiele, S. 13

(2.1.) Die Endposition, deren Erreichen bedeutet, dass der am Zug befindliche Spieler gewonnen hat, ist eine Gewinnposition.
(2.2.) Die Endposition, deren Erreichen bedeutet, dass der am Zug befindliche Spieler verloren hat, ist eine Verlustposition.
(3.1.) Jede Position, die mindestens eine Verlustposition zum unmittelbaren Nachfolger hat, ist eine Gewinnposition.
(3.2.) Jede Position, die nur Gewinnpositionen als unmittelbare Nachfolger hat, ist eine Verlustposition.
(3.3.) Jede Position, die mindestens eine Remisposition, aber keine Verlustposition zum unmittelbaren Nachfolger hat, ist eine Remisposition.[15]
Auf dieser Spielanalyse kann dann eine Spielstrategie aufgebaut werden, wobei zu beachten ist, dass sich die Wertung einer Position immer auf den Spieler bezieht, der gerade am Zug ist.[16]
Wer nur ansatzweise versucht einen Spielbaum für TicTacToe zu erstellen, wird sehr schnell feststellen, dass dies nicht so ohne weiteres möglich ist und man sehr schnell alleine aufgrund des Platzmangels scheitert. Geht man jedoch davon aus, dass der Computer nach vorherigem Abschnitt die Position der Mitte bevorzugt, so kann man einen Teilbaum zeichnen. (siehe nächste Seite).
Die Auswertung des Teilbaums zeigt, dass der Wert des Spiels eine Remisposition ist. Allerdings lässt sich auch ablesen, dass der beginnende Spieler (im Baum Spieler_X) für den Fall in eine Gewinnposition kommt, wenn sein Gegner den ersten Stein in ein Seitenmittenfeld (Feld 1, Feld 3, Feld 5 oder Feld 7) setzt. Man kann also die Strategie insofern erweitern, dass für oben beschriebenen Fall der Computer in seinem zweiten Zug ein Seitenmittenfeld auswählt, welches jedoch nicht in einer Reihe mit dem vom Gegner gewählten Seitenmittenfeld liegt (siehe markierten Knoten). Blockiert der Gegner die mögliche Mühle, muss die Computerstrategie so weiter programmiert sein, dass nun eine Falle entsteht. Hierzu muss der Computer ein Eckfeld auswählen, so dass zwei mögliche Mühlen entstehen.

[15] Baumann, Strategiespiele, S. 33
[16] Sobald ein Spieler einen erlaubten Zug getätigt hat und somit gesetzt hat, ist der Gegner am Zug!


Alternative Zugmöglichkeiten mit derselben Wertung
Falle
Falle
Verkürzter Baum, sobald eine nachfolgende Position als Verlustpostition gekennzeichnet ist, können die übrigen Zweige entfallen.
Wie man anhand der ersten drei Knoten erkennen kann, werden alle Knoten der 8. "Stufe", bei denen es noch zwei Kanten gibt, mit Remis oder Gewinn beurteilt.

Für den Fall, dass der Gegner im ersten Zug ein Eckfeld belegt, ist es sinnvoll, dass der Computer im zweiten Zug ein Seitenmittenfeld auswählt, welches nicht neben dem gegnerischen Eckfeld liegt. So lässt sich nämlich vermeiden, dass der Gegner die im Spielbaum enthaltene Falle aufbauen kann. Entsprechend kann man die Strategie verfeinern, wenn der Gegner beginnt und Feld 4 markiert. Die Computerstrategie gibt vorerst vor, dass nun eine Ecke markiert werden muss. Betrachtet man nun den Spielbaum (Computer ist in diesem Fall Spieler_O), so sieht man, dass es insgesamt nur zwei Möglichkeiten gibt zu gewinnen. Es ist klar, dass egal welchen Zug der Gegner als nächsten tut, die mögliche Mühle blockiert werden muss. Dies geschieht bereits nach der Spielstrategie.

Im linken Teilbaum kann diese Blockade zugleich zur Falle für den Gegner führen. Auch beim Betrachten aller weiteren Fälle, bei denen Spieler_O gewinnen kann, zeigt ein Blick auf den Weg zu dieser Endposition, dass die entsprechenden Züge bereits von der vorhandenen Spielstrategie abgedeckt werden, so dass sich für diese Situation keine Neuerungen ergeben.

Auch ein Teilbaum bietet einige Hinweise, wie die Spielstrategie verbessert werden kann. Es gibt jedoch keine Möglichkeit den Computer so zu programmieren, dass von Beginn an klar ist, dass er gewinnen wird. Denn der Wert des Spiels ist Remis. Somit existiert keine Gewinnstrategie, „die den Gewinn einer Partie sichert, ungeachtet dessen, was der Gegner unternimmt.“[17]

Um die Strategie einfacher, besser und übersichtlicher programmieren zu können, greife ich auf die Codierung von Baumann zurück. Er verschlüsselt die Felder folgendermaßen: „das leere Feld durch eine 0, jedes von X besetzte Feld mit einer 1 und jedes O-Feld mit einer 4. Dann sind X-Mühlen an der Reihensumme 3 und O-Mühlen an der Reihensumme 12 leicht erkennbar.“[18] Diese Verschlüsselung hat den Vorteil, dass die Reihensumme darüber eindeutig Auskunft gibt, wie viele Felder von Spieler_X bzw. Spieler_O belegt worden sind. Diese Veränderungen müssen auch in der Klasse *„TicTacToe20“* beachtet werden. Die Änderungen speichere ich daher in einer neuen Version *„TicTacToe21“* in einem neuen Ordner, sämtliche Computerstrategien müssen auch in diesen Ordner kopiert werden und entsprechend leicht abgeändert werden.

Zunächst ist also ein zusätzliches Array *„Ranzahl“* notwendig, in dem jeweils gespeichert wird, wie viele Reihen auf dem Spielfeld mit der entsprechenden Reihensumme existieren.

[17] Schrage, Strategiespiele, S. 17
[18] Baumann, Strategiespiele, S. 45

Ranzahl[0] = 8 und *Ranzahl[i]* = 0 für i = 1,..12 ist also die Anfangsposition, da noch kein Feld belegt ist. Das Array ist somit mit folgenden Ziffern gefüllt: (8,0,0,0,0,0,0,0,0,0,0,0,0) und kann in der Methode *„neuesSpiel()“* entsprechend erzeugt werden. Erforderlich ist dies nicht, da nach jedem Zug das Feld ohne einen Rückgriff auf die bisherigen Werte neu berechnet wird. Die Methode *„gewonnen()“* kann – muss aber nicht - entsprechend abgeändert werden. Die Methode wird durch die Änderungen wesentlich kürzer, so dass sich die Änderungen rentieren. Ein Sieg liegt dann vor, wenn im Array *„Ranzahl“* eine 3 bzw. 12 notiert wird.

```
public int gewonnen() {
  if (Ranzahl[3] == 1) return 1;
  else if (Ranzahl[12] == 1) return 4;
    else if(zuganzahl == 9) return 2;
      else return -2;
} //Ende gewonnen()
```

Zusätzlich wird eine Methode *„berechnen()“* benötigt, die nach jedem durchgeführten Spielzug die Reihensummen bestimmt und die entsprechende Anzahl im oben genannten Array einträgt.

„Computer3“ erbt natürlich wieder von *„Computer2“,* wobei auch hier einige Änderungen nötig sind. Die Methode *„zweiterZug(TicTacToe21 ttt)“* muss so abgeändert werden, dass für den Fall, dass der Computer begonnen hat (also Mitte belegt hat) und Spieler_X ein Seitenmittenfeld belegt hat, die Reihensumme acht ergibt. Um im nächsten Zug die Falle aufzubauen, muss ein Feld belegt werden, so dass Ranzahl[8] = 2 vorliegt. Dies wird in der Methode *„dritterZug(TicTacToe21)“* realisiert. Ebenso wie in der Methode *„zweiterZug(TicTacToe21)“* wird hier ein möglicher Zug simuliert, kontrolliert, ob eine Falle entstanden ist, und der Zug zurückgenommen. Dies ist nötig, da der eigentliche Zug in der Methode *„ziehen(TicTacToe21)“* durchgeführt wird.

```
public void dritterZug(TicTacToe21 ttt) {
  do {
    do {
      do { feld = zufallszahl();
      } while( feld == 1 | feld == 3 | feld == 5 | feld == 7 );
    } while (ttt.spielfeld[feld]!=ttt.FREI);
  ttt.spielfeld[feld] = ttt.SPIELER_O;
  ttt.berechnen();
  achterReihe = ttt.Ranzahl[8];
  ttt.spielfeld[feld] = ttt.FREI;
  ttt.berechnen();
  } while (achterReihe != 2);
}
```

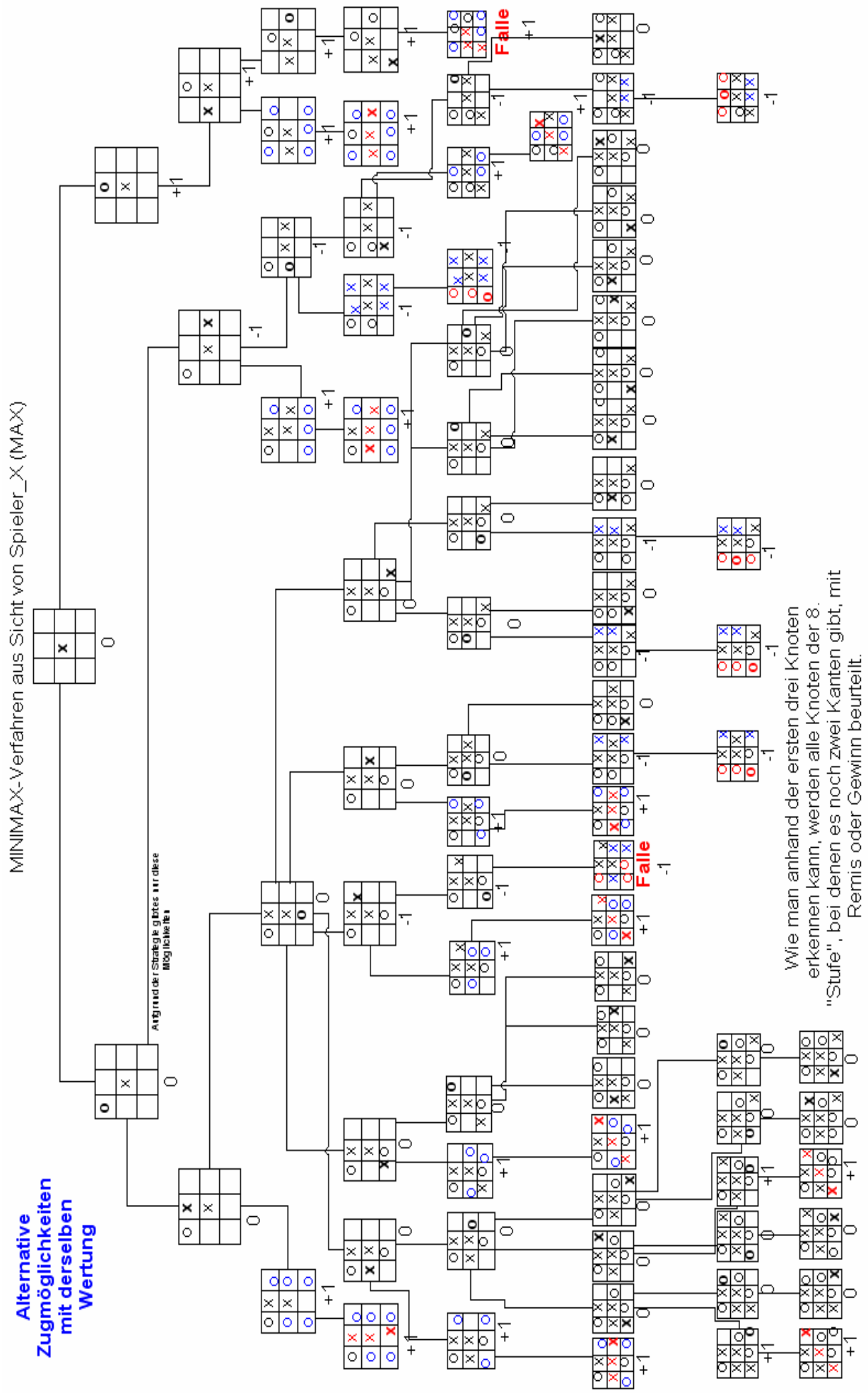
In der Methode *„zweiterZug(TicTacToe21 ttt)“* muss zudem noch eine Fallunterscheidung eingebaut werden, denn bisher wird nur der Fall behandelt, dass der Gegner ein

Seitenmittenfeld belegt hat. Ergänzt werden muss die Möglichkeit, dass ein Eckfeld belegt wurde. In diesem Fall muss ein Seitenmittenfeld ausgewählt werden, welches nicht neben dem Eckfeld des Gegners liegt. Die Reihensumme darf also nur einmal fünf ergeben.
Um diese Methode aufzurufen, muss die Methode *„ziehen(TicTacToe21)“* ebenfalls überschrieben werden.

2.3.6. MINIMAX-Prinzip oder Negmaxverfahren

Beim MINIMAX_Prinzip werden die Spielstände aus Sicht eines Spielers beurteilt. Man beginnt mit der Bewertung der Endzustände, was nicht schwierig ist, weil immer klar ist, wer gewonnen bzw. verloren hat. Liegt ein Sieg von Spieler_X vor, wird die Position mit ‚+1’ bewertet. Liegt eine Niederlage vor, wird als Wertung ‚-1’ eingetragen und ein Unentschieden wird mit ‚0’ gewertet. Um die im Graphen darüber liegenden nichtterminalen Knoten bewerten zu können, müssen zunächst die beiden Spieler unterschieden werden, da eine gemeinsame Gewinnfunktion verwendet werden soll. Spieler MAX versucht, die Gewinnfunktion zu maximieren, Spieler MIN versucht, diese zu minimieren. Die Bewertung der nichtterminalen Knoten erfolgt nach folgenden Regeln:
„- wenn der MAX-Spieler am Zug ist […], ordnet man dem betreffenden Knoten die günstigste (d.h. die quantitativ höchste) Bewertung aller Nachfolgerknoten zu.
- wenn der MIN-Spieler am Zug ist […], erhält der betreffende Knoten die ungünstigste (d.h. quantitativ niedrigste) Bewertung der Nachfolgeknoten.“[19]
Die Bewertung der Wurzel liefert schließlich den Wert des Spiels. Eventuell ist es nicht möglich, einen kompletten Baum zu einem Spiel zu zeichnen, in diesem Fall muss man auf ein modifiziertes MINIMAX-Prinzip zurückgreifen, bei dem der Baum nur bis zu einer bestimmten Tiefe gezeichnet wird und die Endknoten mit Hilfe einer „adäquaten heuristischen Bewertungsfunktion“[20] bewertet werden.
Der Unterschied der MINIMAX-Bewertung und der Bewertung nach Gewinn- und Verlustpositionen liegt also darin, dass beim MINIMAX-Verfahren die Bewertung aus der Sicht eines Spielers erfolgt, während die Positionsbewertung sich immer auf den gerade am Zug befindlichen Spieler bezieht. Wie man jedoch am entsprechenden Spielbaum (nächste Seite) erkennen kann, liefert dieses Verfahren keine neuen Erkenntnisse, denn man kommt zur gleichen Beurteilung. Auf eine entsprechende Programmierung habe ich daher verzichtet.

[19] Helbig, Künstliche Intelligenz und automatische Wissensvermittlung, S. 161
[20] Helbig, Künstliche Intelligenz und automatische Wissensvermittlung, S. 164


MINIMAX-Verfahren aus Sicht von Spieler_X (MAX)
Alternative Zugmöglichkeiten mit derselben Wertung
Falle
Falle
Wie man anhand der ersten drei Knoten erkennen kann, werden alle Knoten der 8. "Stufe", bei denen es noch zwei Kanten gibt, mit Remis oder Gewinn beurteilt.

Die Vorteile des MINIMAX-Verfahren liegen darin, dass man eine gemeinsame Gewinnfunktion hat. Diese kann man jedoch auch nutzen, ohne die Bewertung entsprechend umzudrehen, indem man auf das Negmax-Verfahren zurückgreift. Dieses ist mit dem MINIMAX-Verfahren äquivalent. Bei diesem Verfahren werden die Positionen aus der Sicht des jeweils am Zug befindlichen Spielers beurteilt, allerdings wird der optimale nächste Zug dadurch ermittelt, dass das „Maximum der negierten Werte aller unmittelbaren Nachfolger dieser Position“[21] ermittelt wird, sprich das Minimum.

2.3.7. Heuristische Suche

„Man bezeichnet eine empirisch gestützte und sehr stark auf Intuition beruhende Methode, die es gestattet, die Lösung in einem Problemraum effektiver zu finden und den dafür notwendigen Suchaufwand deutlich einzuschränken, als *Heuristik*. Ein Verfahren, in dem solche Heuristiken eingesetzt werden, heißt dementsprechend *heuristisches Verfahren*.“[22]

Betrachtet man den Spielbaum von Seite 22 so kann man feststellen, dass nach dem fünften durchgeführten Zug eine Abschätzung des weiteren Spielverlaufs möglich ist, denn je nach Spielposition ist bereits an dieser Stelle klar, dass einer der beiden Spieler bereits keine Chance mehr hat zu gewinnen, da er keine Mühle mehr bauen kann. Der aufgezeigte Lösungsansatz beruht auf der Idee von Rüdiger Baumann[23], der die Idee jedoch mit Pascal umgesetzt hat, so dass sich bei der Programmierung mit Java neue Aspekte ergeben.

Nun kann man sich an die Überlegung einer heuristischen Bewertungsfunktion heranwagen.

Wie oben begründet, soll die heuristische Bewertungsfunktion nach dem fünften durchgeführten Zug verwendet werden. Aussagekräftig sind die Anzahl der Reihen X(i) mit genau i = 0, 1, 2, 3 Kreuzen (X) und keinem Kringel (O) und parallel dazu die Anzahl der Reihen O(i) mit genau i = 0, 1, 2, 3 Kringeln und keinem Kreuz. Diese Werte können aus dem Array *„Ranzahl“* leicht ermittelt werden, denn es gilt folgendes:

		Bezogen auf diese Reihe
- kein Kreuz, kein Kringel	-> Reihensumme = 0	alles noch möglich
- 1 Kreuz, kein Kringel	-> Reihensumme = 1	Sieg nur für X noch möglich
- 2 Kreuze, kein Kringel	-> Reihensumme = 2	Sieg nur für X noch möglich
- 3 Kreuze, kein Kringel	-> Reihensumme = 3	Sieg für X

[21] Baumann, Strategiespiele, S. 41
[22] Helbig, Künstliche Intelligenz und automatische Wissensvermittlung, S. 151
[23] nachzulesen bei: Baumann, Strategiespiele, S. 45 ff

- 1 Kringel, kein Kreuz -> Reihensumme = 4 Sieg nur für O noch möglich
- 2 Kringel, kein Kreuz -> Reihensumme = 8 Sieg nur für O noch möglich
- 3 Kringel, kein Kreuz -> Reihensumme = 12 Sieg für O
- 1 Kreuz, 1 Kringel -> Reihensumme = 5 Remis
- 1 Kreuz, 2 Kringel -> Reihensumme = 9 Remis
- 2 Kreuze, 1 Kringel -> Reihensumme = 6 Remis

Daraus ergibt sich X(i) = Ranzahl[i] und O(i) = Ranzahl[4*i] für i = 0,1,2,3.

Für Spieler_X ist eine Reihe mit zwei Kreuzen natürlich mehr wert als eine Reihe mit nur einem Kreuz und entsprechend negativ ist es für ihn, wenn Spieler_O ein oder sogar zwei Kringel in einer Reihe ohne Kreuz hat. Noch schlechter ist es, wenn Spieler_O sogar drei Kringel in einer Reihe hat, dann hat Spieler_X nämlich verloren. Aus diesen Überlegungen ergibt sich folgender Ansatz, wenn Spieler_X am Zug ist:

$-a * O(3) + b * X(2) – c * O(2) + d * X(1) – e * O(1)$, mit $a > b > c > d > e$

„Da eine eigene Mühle mehr als alles andere wert ist, geben wir ihr einen vergleichsweise hohen Wert, etwa a = 31. Hat ein Spieler eine Mühle erzielt, kann der Gegenspieler höchstens zwei Zweierreihen erreicht haben, daher muss a > 2b gelten, weshalb wir b = 15 setzen. Besitzt ein Spieler eine Zweierreihe, so kann der Gegner nicht mehr als drei Einserreihen besitzen, also gilt: b > 2e und c > 2d, was durch die Setzung c = 7, d = 3 und e =1 erreicht werden kann.“[24]

Somit folgt:

$$-31 * O(3) + 15 * X(2) – 7 * O(2) + 3 * X(1) – 1 * O(1)$$

Analog ergibt sich, wenn Spieler_O am Zug ist:

$$-31 * X(3) + 15 * O(2) – 7 * X(2) + 3 * O(1) – 1 * X(1)$$

Als gemeinsame Bewertungsfunktion kann man also folgende Funktion festhalten:

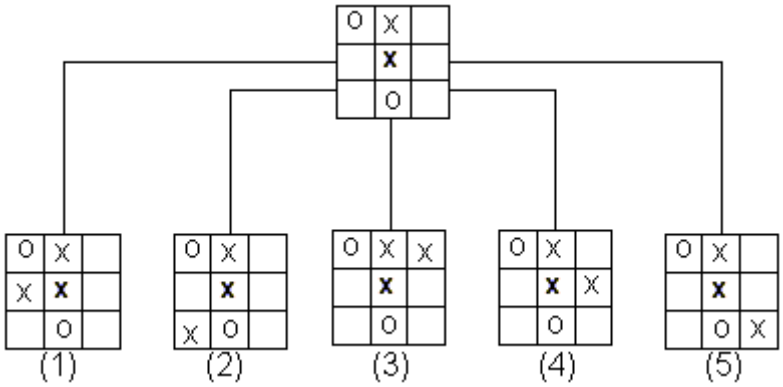
$$B(x) = -31 * Ranzahl[3(5-x)] + 15 * Ranzahl[2x] – 7 * Ranzahl[2(5-x)] + 3 * Ranzahl[x] – Ranzahl[5-x].$$

Ist Spieler_X am Zug setze x = 1, ist Spieler_O am Zug setze x = 4.

Beispiel 1:

Spieler_X ist am Zug und muss sich für einen der fünf möglichen Züge entscheiden. Nach dem jeweiligen Zug wäre Spieler_O an der Reihe, so dass B(4) berechnet werden muss, d.h. B(4) = -31 * Ranzahl[3] + 15 * Ranzahl[8] – 7 * Ranzahl[2] + 3 * Ranzahl[4] – Ranzahl[1]

[24] Baumann, Strategiespiele, S. 48f



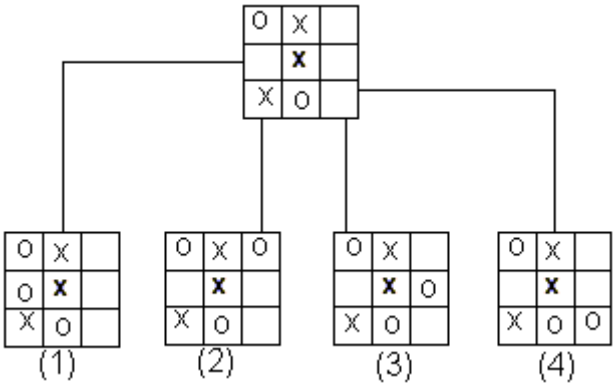
Wie man sofort erkennen kann, ist Ranzahl[3] = Ranzahl[8] = 0 für alle fünf Fälle.

Fallnummer	Ranzahl[2]	Ranzahl[4]	Ranzahl[1]	B(4)
(1)	1	1	1	-5
(2)	1	0	1	**-8**
(3)	1	2	2	-3
(4)	1	2	2	-3
(5)	0	1	3	0

Spieler_X wählt also die zweite Zugmöglichkeit aus, da hier der größte negierte Wert (Minimum) berechnet wurde.

Beispiel 2:

Spieler_O ist am Zug und muss sich für einen der vier möglichen Züge entscheiden. Nach dem jeweiligen Zug wäre Spieler_X an der Reihe, so dass B(1) berechnet werden muss, d.h. B(1) = -31 * Ranzahl[12] + 15 * Ranzahl[2] – 7 * Ranzahl[8] + 3 * Ranzahl[1] – Ranzahl[4]



Wie man sofort feststellen kann, ist Ranzahl[12] = Ranzahl[8] = Ranzahl[1] = 0 für alle vier Fälle.

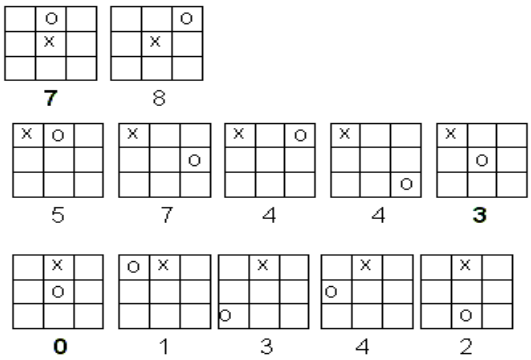
Fallnummer	Ranzahl[2]	Ranzahl[1]	Ranzahl[4]	B(1)
(1)	1	0	0	15
(2)	0	1	1	**2**
(3)	1	0	1	14
(4)	1	1	1	17

Spieler_O wählt ebenfalls die zweite Zugmöglichkeit aus, da hier der kleinste Wert berechnet wurde. Jeder andere Zug ist auch nicht sinnvoll, da Spieler_O die Mühle blockieren muss.

Diese Bewertungsfunktion kann jetzt auf jeder Stufe des Baumes angesetzt werden, d. h. nicht nur nach fünf durchgeführten Zügen. Dies hat zur Folge, dass sich zwei Möglichkeiten der Programmierung ergeben. Entweder erbt *„computer4“* von *„computer3“* oder er wird unabhängig von den übrigen Spielstrategien programmiert.

Wenn *„computer4“* erbt, muss die Methode *„dritterZug(TicTacToe21)“* ebenso wie die Methode *„ziehen()“* überschrieben werden. In die Methode *„dritterZug(TicTacToe21)“* muss die Bewertungsfunktion integriert werden und alle folgenden Züge dürfen in der Methode nicht mehr zufällig aufgerufen werden, sondern müssen ebenfalls über die Bewertungsfunktion bestimmt werden.

Andernfalls ist es denkbar, dass der Computer alle Züge über die Bewertungsfunktion ausführt, so dass die Vererbung nicht notwenig und nicht sinnvoll ist. Betrachtet man sich die ersten Möglichkeiten, so führt diese Vorgehensweise zur gleichen Strategie. Denn belegt der Computer in seinem ersten möglichen Zug die Mitte, so gilt Ranzahl[0] = Ranzahl[4] = 4 und Ranzahl[i] = 0 für i = 1, 2, 3, 5, 6, ... , 12, d. h. B(1) = -4. Bei der Belegung der Seitenmitte ergibt sich B(1) = -2 und bei der Belegung eines Eckfeldes B(1) = -3. Beginnt der Computer wird er also das mittlere Feld belegen. Eröffnet Spieler_X das Spiel, so können folgende Situationen eintreten:



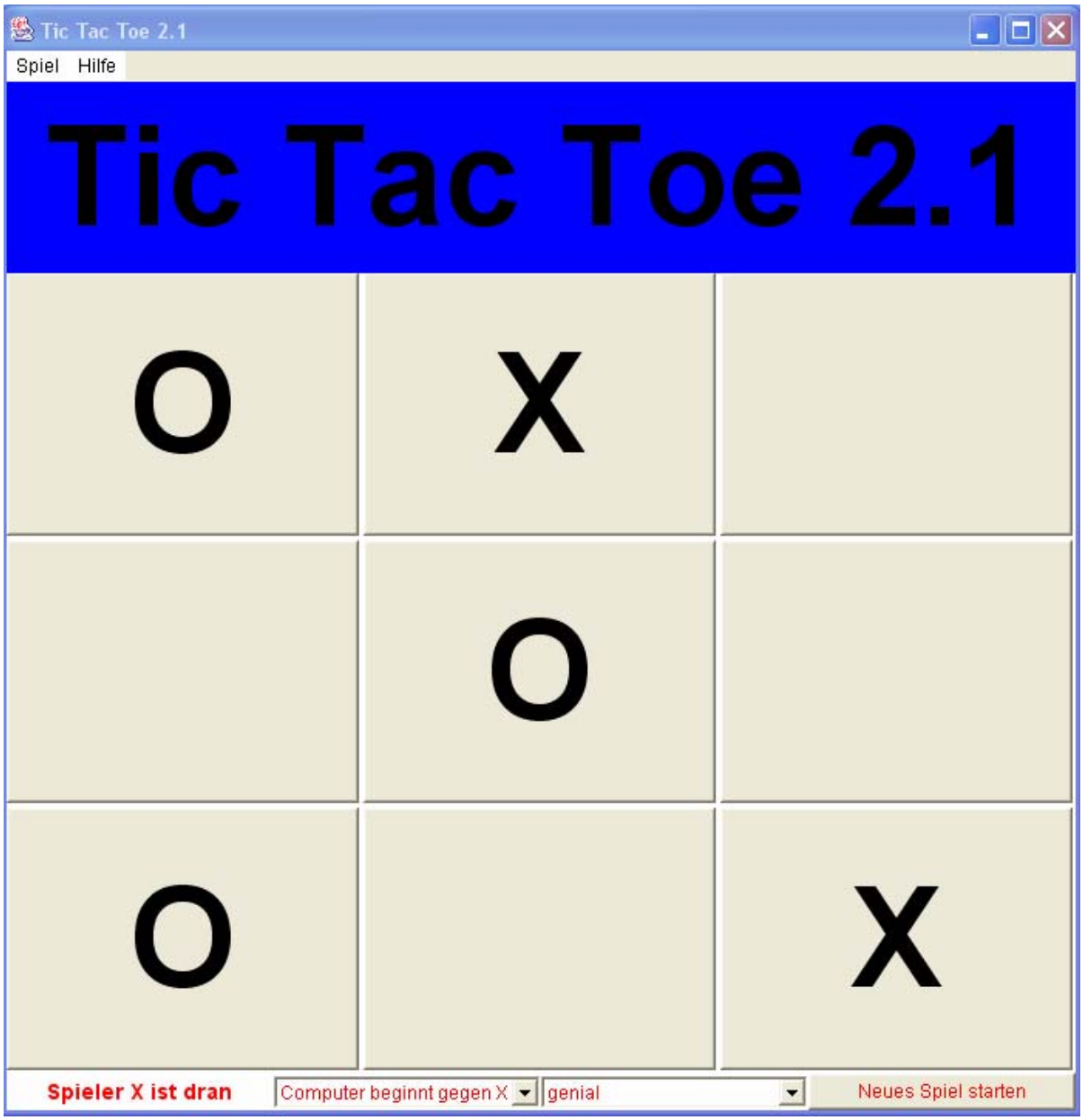
Man kann also erkennen, dass auch in diesem Fall die Mitte belegt wird, falls das Feld noch frei ist und ansonsten eine Ecke ausgewählt wird. Es ist also egal, für welche der beiden Arten man sich entscheidet, die Strategie ist die gleiche. Für eine übersichtliche Programmierung empfiehlt sich in beiden Fällen, eine Methode *„bewerten()“* einzufügen, die das Ergebnis der Bewertungsfunktion zurückgibt. Man braucht nicht auf die allgemeine Bewertungsfunktion zurückzugreifen, da der Computer immer einen möglichen Zug simuliert und die entstandene Spielsituation aus Sicht von Spieler_X bewertet. Daher genügt es, B(1) zu berechnen.

Realisiert habe ich die Fassung, bei der sämtliche Züge über die Bewertungsfunktion ausgeführt werden. In der Methode *„ziehen(TicTacToe21 ttt)“* wird mit Hilfe der Methode *„bestesFeld(TicTacToe21)“* das Feld bestimmt, welches für den Computer in der jeweiligen Spielsituation am günstigsten ist. In dieser Methode werden alle möglichen Spielzüge simuliert und mittels der Methode *„bewerten(TicTacToe21 ttt)“* einschätzt. Die Bewertungen werden in dem Feld *„bewertungen[]“* abgespeichert und anschließend verglichen. Da Math.min(a,b) immer nur von zwei Werten das Minimum bestimmen kann, muss mittels einer Schleife jede Bewertung des Feldes mit dem aktuellen Minimum verglichen werden.

```
public int bestesFeld(TicTacToe21 ttt) {
  int b, min;
  int minfeld;
  int bewertung[];
  minfeld = 9;// Dies Feld existiert nicht, denn es wird in jedem Fall ein
              // existierendes Feld eingesetzt, nötig für Compiler
  b=100;      // Muss hoch initialisiert werden, damit die Bewertungsfunktion
              // in jedem Fall unter diesem Wert liegt
  bewertung = new int[9];
  // alle möglichen Züge müssen simuliert und bewertet werden
  for ( int j=0 ; j<9 ; j++ ) {
  if (ttt.spielfeld[j] == ttt.FREI) {
    ttt.spielfeld[j] = ttt.SPIELER_O;
    ttt.berechnen();
    bewertung[j] = bewerten(ttt);
    ttt.spielfeld[j] = ttt.FREI;
  }
  else {bewertung[j]=100;} // besetzte Felder müssen hoch bewertet werden
  }

  //Vergleich der Bewertungen
  for (int i=0; i<9; i++) {
    if( bewertung[i]<b) { minfeld = i;}
    min= (int)(Math.min(bewertung[i],b)); // Minimum von bewertung[i] und b
    b = min;                              // b auf das aktuelle Minimum setzen
  }
  return minfeld;  // Feld mit der kleinsten Bewertung wird zurückgegeben
}
```

Um nun die verschiedenen Strategien überprüfen und benutzen zu können, empfiehlt es sich, eine Wahlmöglichkeit für die Stärke des Computers einzubauen. Es kann nun zwischen folgenden Schwierigkeitsgraden gewählt werden: sehr einfach (*„computer0“)*, einfach (*„computer1“),* mittel (*„computer2“*), schwer (*„computer3“*) und genial (*„computer4“*). Diese Möglichkeit ist in Version 2.1 verwirklicht. Ich habe mich dafür entschieden, diese ebenfalls mit Hilfe eines Choice-Menü umzusetzen.



3. Der Einsatz im Unterricht

3.1. Projekte im Informatikunterricht

Es würde den Rahmen dieser Arbeit sprengen, wenn ich ausführlich auf den Einsatz von Projekten im Informatikunterricht eingehen würde. Daher sei an dieser Stelle die Rechtfertigung von Informatikprojekten kurz nach R. Schulz-Zander zitiert.

„Die inzwischen wohl weitgehend anerkannte Forderung, Projekte im Informatikunterricht durchzuführen, stützt sich auf allgemeine pädagogische Konzeptionen und darüber hinaus auf folgende Argumentationen:

- Verschiedene in der Informatik entwickelte Methoden und Verfahren […] werden für den Anwender dieser Methoden und Verfahren erst ab einer gewissen Komplexitätsstufe notwendig und damit erst einsichtig …
- Die Bedeutung des Einsatzes der Datenverarbeitung in gesellschaftlichen Bereichen kann erst dann vom Schüler eingeschätzt werden, wenn die Probleme eine größere Komplexität haben und damit einer größeren Realitätsnähe aufweisen.
- Die Schüler lernen die in den Anwendungsbereichen üblichen arbeitsteilige Entwicklung von DV-Systemen in Projekten und dort eingesetzte Formen der Arbeitsorganisation ansatzweise kennen.“[25]

3.2. Spiele im Projektunterricht der Informatik?

Schüler finden es zunächst interessant und motivierend, wenn ein Spiel als Projektthema vorgeschlagen wird. Dies allein darf jedoch noch nicht dazu führen, dass ein Spiel ausgewählt wird. „Jeder Spielvorschlag sollte besonders sorgfältig geprüft werden, ob sich mit ihm die übergeordneten Ziele der Projektarbeit verwirklichen lassen.“[26] Zusätzlich ist es von Vorteil, wenn sich die Schüler mit dem ausgewählten Spiel auch Grundsätze der Spielprogrammierung aneignen können.[27] Allgemein eignen sich Strategiespiele zum Studium von Lern- und Problemlösungsmethoden, da sie komplexe Situationen mit nur wenigen Regeln schaffen, „die nicht weniger Einsicht, Kreativität und Expertenwissen verlangen als Probleme, die uns auf den Gebieten der Wirtschaft, der Verwaltung oder der Wissenschaft begegnen. Im Gegensatz zu

[25] R. Schulz-Zander in LOGIN 1981, Heft 1, S. 25, zitiert aus: Lehmann, Projektarbeit im Informatikunterricht, S. 204f
[26] Lehmann, Projektarbeit im Informatikunterricht, S.186
[27] vgl. Lehmann, Projektarbeit im Informatikunterricht, S. 186

diesen Gebieten geben Spiele Gelegenheit, die erarbeiteten Strategien durch genau definierte Leistungskriterien zu beurteilen.“[28]

Wie man dem vorherigen Kapitel entnehmen kann, erfüllt das ausgewählte Spiel TicTacToe diese Anforderungen, denn es genügt den Ansprüchen bezüglich der allgemeinen Programmstruktur strategischer Spiele (vgl. Seite 4 dieser Arbeit).

3.3. Wann kann man das Spiel TicTacToe einsetzen?

3.3.1. Ein Projekt am Ende der Jahrgangsstufe 11

Die Einsatzmöglichkeiten sind sehr vielfältig, so dass der Lehrer je nach Kurszusammensetzung, Zeit und Kenntnisstand der Schüler einen eigenen Weg wählen muss. Die ersten Versuche können sicherlich bereits am Ende der 11.2 unternommen werden. Sicherlich muss die Aufgabenstellung immer dem Kursniveau angepasst werden. Um die Schüler auf die systematische Erstellung umfangreicher Softwaresysteme vorzubereiten, können den Schülern die beiden Vorversionen als Quellcode ausgehändigt werden. Diese Programme dienen dann nach ihrer Analyse als Grundlage für die Diskussion über die Datenstruktur, d.h. über Klassen, Methoden und Schnittstellen.

Da aufgrund des hessischen Lehrplans gerade am Ende der 11.2 nur wenig Zeit für ein Projekt bleibt, kann zur Zeitersparnis ein Kurzprojekt durchgeführt werden, bei dem die Themenwahl nicht durch die Schüler erfolgt, sondern vom Lehrer das Thema vorgegeben wird. Zeit lässt sich auch einsparen, indem die Aufgabenstellung sehr konkret mit genauer Angabe bzgl. der gewünschten Klassen und Methoden an die einzelnen Gruppen gegeben wird, so dass die Problemanalyse entfällt. Alternativ kann auch ein Struktogramm vorgegeben werden. Wünschenswert ist es, dass keine zu starken Vorgaben gemacht werden und die Schnittstellen, benötigte Klassen und Methoden mit den Schülern gemeinsam erarbeitet werden. Da die Schüler allerdings wenig Erfahrung mit einer solchen Projektarbeit mitbringen, halte ich es für sinnvoller, wenn im Rahmen der ersten Projektarbeit exaktere Vorgaben gemacht werden, um den Erfolg nicht zu gefährden.

Ein erstes erfolgreiches Projekt steigert die Motivation der Schüler beim nächsten Projekt, und noch wichtiger: ein missglücktes Projekt kann die Schüler so negativ beeinflussen, dass dies länger Folgen haben kann. Zudem hat diese Vorgehensweise den weiteren Vorteil, dass die

[28] Baumann, Didaktik der Informatik, S. 368

Schüler anhand der Aufgabenstellung erkennen können, wie das Ergebnis einer Problemanalyse aussehen kann. Dies führt beim nächsten Projekt, bei dem die Schüler diesen Schritt selbständig durchführen sollen zu mehr Sicherheit.

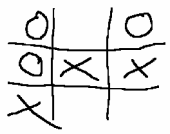
Ob die Schüler nun an einem Spiel gemeinsam oder jede Gruppe für sich an einem Spielvorschlag arbeitet, muss sicherlich danach beurteilt werden, wie viel Zeit zur Verfügung steht bzw. wie die Gruppe zusammengesetzt ist. Denkbar ist z.B., dass die Schüler in sehr wenigen Stunden die Methoden arbeitsteilig bearbeiten, wenn die Zeit für ein eigentliches Projekt nicht mehr ausreicht, so können die Schüler auf diese Weise mindestens die Ansätze der Projektarbeit kennen lernen. Hat man noch einige Stunden Zeit, so ist es für die Schüler gewiss einfacher, wenn sie in einer Kleingruppe ein eigenes Programm erstellen und keine Absprachen mit den anderen Gruppen nötig sind. Zudem bittet dieser Ansatz den besseren Schülern die Möglichkeit, ihre eigenen Ideen umzusetzen und weitere Aspekte in die Programmierung mit einfließen zu lassen.

Insgesamt werden viele verschiedene Bereiche aus den verbindlichen Unterrichtsinhalten des hessischen Lehrplans für die Bearbeitung benötigt, so dass in diesem Projekt der gesamte Stoff des Kurses „Grundlagen der Programmierung“ wiederholt werden kann.

Eine stark vorstrukturierte Aufgabenstellung könnte folgendermaßen aussehen:

Projekt: Entwurf und Implementierung des Spiels TicTacToe

Beim Spiel "TicTacToe“ spielen zwei Spieler abwechselnd gegeneinander. Der erste setzt sein Kreuz in eines der neun Felder, die quadratisch angeordnet sind, worauf der Gegner einen Kringel platziert. Ziel ist es, drei gleiche Symbole in einer senkrechten, waagrechten oder diagonalen Reihe anzuordnen. Sind alle Felder belegt und kein Spieler hat eine Mühle erzeugt, so endet das Spiel unentschieden.



Für die Programmierung eines TicTacToe – Spiels wird eine Klasse benötigt, die das Spielfeld erzeugt, die Aufgaben eines Spielleiters übernimmt und von Frame abgeleitet wird. Folgende Bezeichnungen für Konstanten und Variablen sollen verwendet werden:

```
//Objekte und Variablen deklarieren
Label ueberschrift, meldung;
Panel oben, mitte, unten;
Button neuesSpiel;
int spielerAmZug, zuganzahl;
Font spielfont;
ButtonLauscher spielLauscher = new ButtonLauscher();
LauscherListe1 L1 = new LauscherListe1();
LauscherListe2 L2 = new LauscherListe2();
private MenuItem neu, laden, speichern, beenden;
private MenuItem regel, info;

//Button tragen den gleichen Namen, werden aber von 0 bis 9
//durchnummeriert
//Anordnung im Spielfeld
//0  1  2
//3  4  5
//6  7  8
public Button[] spielbu;          //Array für Button-Spielfeld
public int[] spielfeld;           //Array für codierte Einträge

//Codierte Spielfeldeinträge
public static final int SPIELER_X = 1,     //die Gewinnberechnung ist mit
                        SPIELER_O = -1,    //dieser Wertebelegung eindeutig
                        FREI = 0;
```

*Die Spielfeldklasse soll neben **dem Konstruktor** folgende **Klassen** enthalten:*

- Class FensterLauscher extends WindowAdapter

  zum Schließen des Fensters
- public class ButtonLauscher implements ActionListener

  führt einen Spielzug durch, wenn ein Button angeklickt wird
- public class LauscherListe1 implements ActionListener

  führt den entsprechenden Vorgang durch, wenn ein MenüItem gedrückt wird (neu, laden, speichern, beenden)
- public class LauscherListe2 implements ActionListener

  führt den entsprechenden Vorgang durch, wenn ein MenüItem gedrückt wird (regel, info)

*und folgende **Methoden**:*

- public void neuesSpiel()

  eine neue Partie wird gestartet
- public int gewonnen()

  überprüft, ob einer der Spieler in einer Zeile, Spalte oder Diagonale eine Mühle hat
- public static void main (String args[])

Die Ergebnisse der jeweiligen Phasen des Projektverlaufs sind zu dokumentieren.

Eventuell können dann entsprechende Teilaufgaben von einzelnen Gruppen übernommen werden.

3.3.2. Ein Projekt oder eine Hausarbeit in der Jahrgangsstufe 12

Zu den fakultativen Unterrichtsinhalten des Kurses „Objektorientierte Modellierung“ im Leistungskurs gehört nach dem hessischen Lehrplan auch die Bewertung von Spielsituationen.[29] Eine Analyse unterschiedlicher Spielstrategien ist für die Schüler interessant. Es bietet sich an, die Schüler an unterschiedlichen Spielstrategien arbeiten zu lassen, nachdem die ersten grundsätzlichen Ideen gemeinsam besprochen worden sind. Zu diesen grundsätzlichen Ideen gehören die Auswahl eines Zufallfeldes und das Besetzen eines Feldes, wenn man dadurch gewinnen bzw. nicht verlieren kann. Interessant ist es dann, die Ergebnisse der diversen Herangehensweisen zu diskutieren. Im Laufe der Diskussion werden die Schüler schnell Gemeinsamkeiten zwischen Negmax- und MiniMax-Verfahren und der Betrachtung von Gewinn- und Verlustpositionen erkennen. Sie werden aber auch die Vor- und Nachteile herausstellen können.

Da es ausreichend theoretische Materialien gibt, an Hand derer die Schüler sich das Hintergrundwissen erarbeiten können, ist dies ein geeigneter Bereich für eine Hausarbeit. Je nach Größe des Kurses ist es auch denkbar, dass nicht alle Schüler an Spielstrategien für TicTacToe arbeiten, sondern ein anderes Spiel bearbeiten oder die Auswirkungen einer veränderten Spielregel analysieren. Mögliche Abwandlungen der Spielregel können darin bestehen, dass

- das Spielfeld vergrößert wird, z.B. 4*4- Felder,
- nur drei Symbole eines Spielers gleichzeitig gesetzt sein dürfen (vgl. Mühlespiel), d. h. nachdem ein Spieler dreimal gesetzt hat, muss er zunächst ein gesetztes Feld löschen, bevor er ein neues Feld belegen darf
- die Anzahl der Mühlen über Sieg oder Niederlage entscheidet (Dies macht nur Sinn, wenn das Spielfeld vergrößert oder auf die 3. Dimension erweitert worden ist)

Als Grundlage bietet es sich an, den Schülern neben den in dieser Arbeit enthaltenen Spielbaum aus der Reihe „Arbeitsheft Informatik“ den Band Strategiespiele zur Verfügung zu stellen. Die Programme sind hier in Pascal erstellt, so dass die Schüler eine gute Hilfestellung aber keine fertige objektorientierte Lösung vorfinden. Man kann die Frage stellen, ob es sinnvoll ist, den

[29] vgl. Lehrplan Informatik, S. 14

Schülern einen Teilspielbaum zur Verfügung zu stellen. In Anbetracht des Umfangs halte ich es für zweckmäßig, allerdings ohne Bewertungen, so dass die Schüler diese selbst vornehmen müssen. Auch die weitere Auswertung und Umsetzung in das Programm muss von den Schülern noch geleistet werden. Den entsprechenden „leeren“ Teilspielbaum füge ich im Anhang bei.
Eine entsprechend offene Aufgabenstellung könnte wie folgt aussehen:

Projekt: Entwurf und Implementierung des Spiels TicTacToe.

Das bereits erstellte Computerspiel soll so überarbeitet werden, dass der Computer die Rolle des Spielgegners übernimmt. Die nötige Spielstrategie leiten Sie bitte nach dem MiniMax-Verfahren ab.

Die Ergebnisse der jeweiligen Phasen des Projektverlaufs sind zu dokumentieren und die Ergebnisse sowie eine kurze Einführung in das MiniMax-Verfahrens zu präsentieren.

Entsprechend ist im Rahmen der Client-Server-Programmierung eine Erweiterung als Netzwerkspiel denkbar. Eine Lösung hierzu habe ich im Anhang beigefügt. Es handelt sich um ein Resultat eines Java-Kurses des Gymnasiums Korschenbroich, welches zusätzlich den Spielern die Gelegenheit bietet, mit den Mitspielern zu chatten.
Die Erweiterung des Spiels um die 3. Dimension kann sicherlich frühestens in der 12. Jahrgangsstufe vorgenommen werden. Reizvoll ist die Programmierung sicherlich, besonders wenn die Darstellung auch räumlich wahrgenommen werden kann. Denn es ist auch möglich drei zweidimensionale Spiele nebeneinander anzuzeigen und in der Programmierung zu verknüpfen. Dies erfordert dann jedoch vom Benutzer viel räumliches Vorstellungsvermögen. Die Ermittlung einer Strategie ist sehr schwierig und soll daher an dieser Stelle nicht vertieft werden.

3.3.3. Ein Projekt in der Jahrgangsstufe 13

Im Rahmen des Wahlthemas „Prolog als Sprache der Künstlichen Intelligenz“ kann thematisiert werden, ob ein Computer lernen kann. Dieses Thema ist sehr aktuell und ist nicht nur für die Informatik interessant, sondern ist auch in Bezug auf ethische Fragen wichtig. Umgesetzt werden kann diese Idee dann auf verschiedene Weisen

Als Aufhänger kann ein Ausschnitt aus dem Film „WarGames“[30] verwendet werden. In diesem Film dringt ein Schüler in das Computersystem des Pentagon ein und verursacht dadurch beinahe einen dritten Weltkrieg. Dieser Krieg wird nur dadurch verhindert, dass der Computer über das Spiel TicTacToe eine günstige Strategie sucht. Da das Spiel jedoch den Wert Remis hat, erkennt der Computer, dass das Spiel keinen Sinn macht und bricht das Szenario ab.
Eine Lösung kann entweder über eine logische Programmiersprache, z. B. Prolog oder auch mit Hilfe des erstellten Programms in Java gesucht werden.
Die Schüler können versuchen, die einzelnen Spielpositionen während einer Partie mit einer Bewertung abzuspeichern. Die Bewertung soll immer dann positiv ausfallen, wenn die Partie gewonnen wurde. Ein Spielstand kann jedoch mal zum Sieg, mal zum Remis oder sogar zum Verlust der Partie führen, so dass die Häufigkeit berücksichtigt werden muss. Es ist denkbar, dass bei jedem anschließenden Sieg des Computers die Bewertung um eins erhöht, bei einem Remis nichts geschieht und bei einem Sieg des Computergegners um eins erniedrigt wird. Die Zugauswahl erfolgt dann, nachdem von allen nachfolgenden Zügen die Bewertung abgefragt wurde. Es wird der Zug gewählt, der die höchste Bewertung hat. Bei gleicher Bewertung entscheidet der Zufall. Ob diese Vorgehensweise zum Ziel führt, d. h. der Computer lernt und somit immer besser spielt, ist fraglich. Es ist denkbar, dass der Computer manche guten Züge nicht kennen lernt, da er sich zuvor nach dem Zufallsprinzip für einen schlechteren entschieden hat. Durch „unglückliche“ Umstände ist es möglich, dass die Bewertung dieses Zuges zunächst steigt, so dass der Computer immer diesen bevorzugen wird, bis seine Bewertung dem eigentlich besseren Zug wieder gleicht. Diverse dieser Züge sind denkbar. Damit der Computer erfolgreich lernen kann, sind daher sehr viele Partien nötig.

[30] Der Spielfilm „WarGames – Kriegsspiele“ von 1983 mit Matthew Broderick ist ein spannender Computerthriller aus einer Zeit als Hacker und Internet noch nicht zum allgemeinen Vokabular gehörten.

Literaturverzeichnis

Baumann, Rüdeger: Didaktik der Informatik. 2. vollst. Neu bearb. Aufl. Stuttgart: Klett, 1996.

Baumann, Rüdeger: Strategiespiele. Einführung in kombinatorische Suchverfahren und in die modulare Programmentwicklung. Arbeitshefte Informatik. Stuttgart: Klett, 2000.

Helbig, Hermann: Künstliche Intelligenz und automatische Wissensverarbeitung. 2. stark bearb. Aufl. Berlin: Verlag Technik, 1996.

Kopp, Dr. Bernhard: Das neue Computerlexikon. Lizenzausgabe für die Büchergilde Gutenberg. Frankfurt am Main: Büchergilde, 1992.

Lehmann, Eberhard: Projektarbeit im Informatikunterricht. Entwicklung von Softwarepaketen und Realisierung in PASCAL. Stuttgart: Teubner, 1985.

Lehrplan Informatik. Gymnasialer Bildungsgang Jahrgangstufe 11 bis 13. Hg. vom Hessischen Kultusministerium. Stand: 03.06.2002

Schrage, Georg. Baumann, Rüdeger : Strategiespiele. Computerorientierte Einführung in Algorithmen der Spieltheorie. München: Oldenbourg, 1984.

Internetseiten (auch im Anhang als Kopie)

- www.gymnasium-wertingen.de/deutsch/fachbereiche/informatik/material/wahlkurs_java/Modellbildung/TicTacToeStrategie.html

- www.fh-niederrhein.de/~gkorsch/javakurs/wn4/wn4.htm

Versicherung

Ich versichere, diese Arbeit selbständig verfasst, keine anderen als die angegebenen Hilfsmittel verwendet und die Stellen, die in anderen Werken im Wortlaut, als Graphik oder dem Sinne nach entnommen sind, mit Quellenangaben kenntlich gemacht zu haben.