TicTacToe

Realisierung mittels minimax und alpha-beta pruning

Hauptziel dieses Projekts ist es einen Algorithmus, der einen unschlagbaren TicTacToe Gegner simuliert, zu entwickeln. Dieser Algorithmus soll auf Brettern der Größen 3x3, 4x4 und theoretisch auf auch noch größeren Brettern lauffähig sein. Im folgenden werden sowohl die Hauptkomponenten dieses Algorithmus erklärt als auch Vergleiche zu anderen Algorithmen gemacht in Bezug auf Leistung, Zuverlässigkeit und Umsetzung.

Konzept des perfekten Spielers:

Um die in den folgenden Abschnitten erklärte Algorithmen nachvollziehen zu können, muss zuerst einmal das Konzept des perfekten Spielers eingeführt werden. Ein perfekter Spieler ist prinzipiell einer, der entweder gewinnt oder ein Unentschieden erzwingt. Das heißt ein Spiel zwischen zwei perfekten Spielern kann nur mit einem Unentschieden enden.

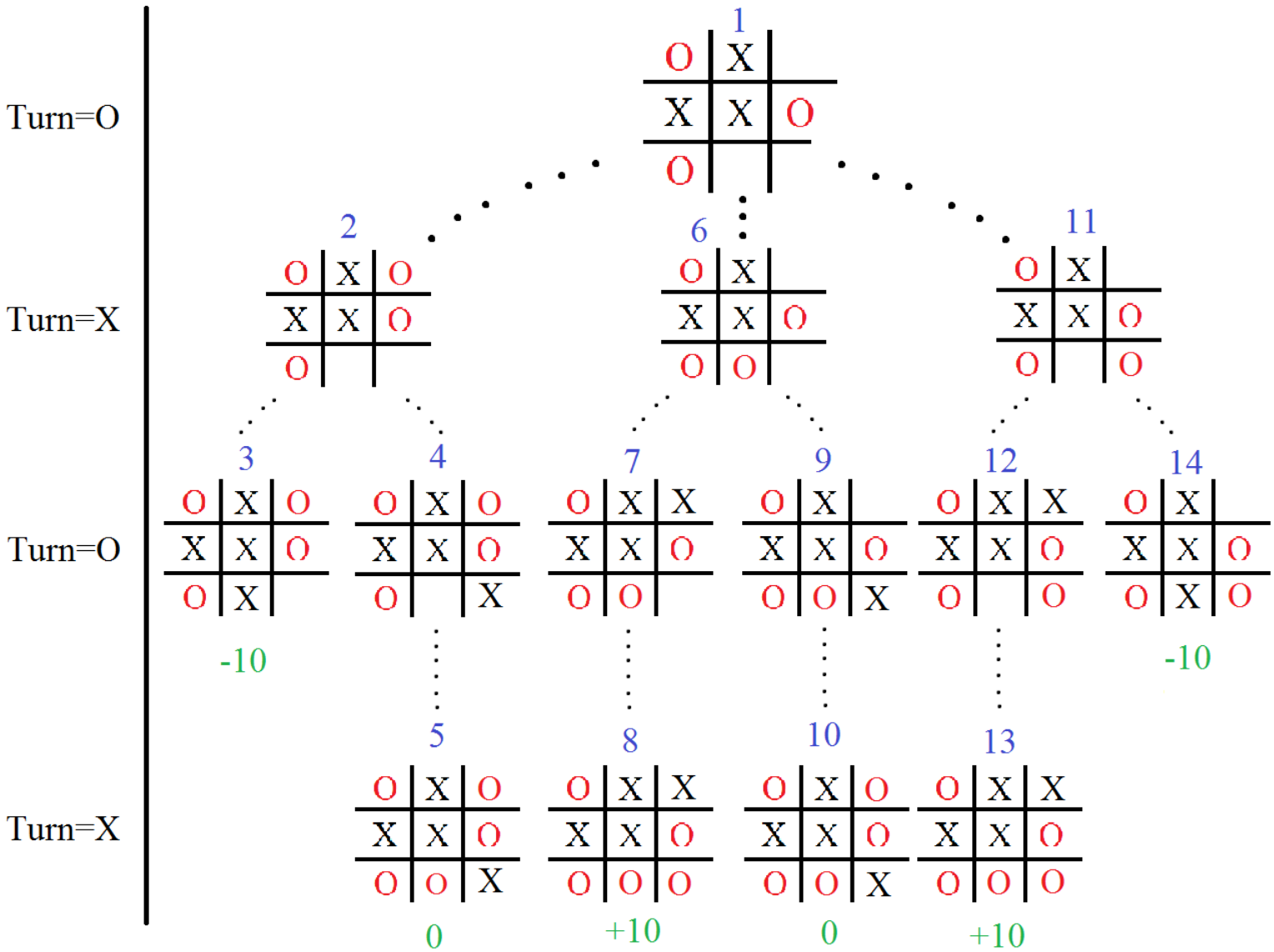
Prinzip von Minimax:

Für ein TicTacToe Spiel auf einem Brett der Größe 3x3 ist der Minimax-Algorithmus sehr gut geeignet. Zuerst aber muss über eine heuristische Auswertungsfunktion entschieden werden. In diesem program wurde eine einfache Funktion ausgesucht die einer von drei

Werte zurückgibt. +10 bei einem Computergewinn, -10 bei einem Spielergewinn und 0 bei einem Unentschieden.

Angesichts dieser Werte, werden die Interessen bzw. Ziele beider Spieler ersichtlich. Der Computer will eine größtmögliche Auswertung finden, deswegen wird er auch der ‚maximierende' Spieler genannt. Der wird immer den Pfad wählen, der zur größten Auswertung führt. Auf der anderen Seite wird der Spieler immer den Pfad wählen, der zur kleinstmöglichen Auswertung führt. Aus diesem Grund ist er auch der ‚minimierende' Spieler genannt. Das Prinzip von Minimax lässt sich sehr leicht verstehen anhand eines Beispiels.

Abb. [1]: Minimax-Baum zum Auffinden des besten Zuges



In Abbildung 1 ist ein Diagramm zu sehen, das den Suchverfahren nach dem besten Zug simmuliert. Die nummern die genau über den Brettern zu sehen sind, deuten auf den Zeitpunkt der „Erstellung“ dieses Brettzustands hin. In diesem Szenario ist der Computer dran. Er will nun wissen wo er seinen O legen sollte. Deswegen ruft er die Minimaxmethode auf. Ab diesem Zeitpunkt wird die Minimaxmethode jedes mögliches Spielszenario erstellen und erkunden. Nachdem jedes Szenario beendet ist wird es einen Wert(+10, -10, 0) zurückgeben der dann benutzt wird, um letztendlich entschieden werden kann welches Szenario oder welcher Spielablauf für den aufrufenden Spieler(in diesem Fall der Computer) optimal ist. Die Minimaxmethode wird dann die Zellennummer zurückgeben, die es dem Spieler dann letztendlich ermöglichen wird dieses Szenario in Wirklichkeit zu erleben.

Nun wird Schritt für Schritt den Ablauf für den linken Zweig des Baums erklärt:

1)Nachdem minimax für den Zustand 1 aufgerufen worden ist, wird zunächst geprüft, ob das Spiel schon zu Ende ist. Wenn dies nicht der Fall ist, wird ein int Feld mit den Zellennummern aller noch leeren Felder erstellt.

2)Das erste Element dieses Feldes wird gewählt und ein O entsprechend seinem Wert wird hingelegt. Somit wurde der Zustand 2 erstellt.

3)Für diesen Zustand wird die Minimaxmethode rekursiv aufgerufen und die Schritte 1 und 2 wiederholen sich bis ein Endezustand erreicht wurde(d.h. jemand gewonnen hat oder das Spiel unentschieden ist).

4)Beim Zustand 3 ist genau das passiert, da in diesem Zustand der Spieler X schon ein Gewinnmuster gefunden hat, nähmlich die Felder 1, 4 und 7.

5)Für diesen Zustand wird dann wieder die Minimaxmethode rekursiv aufgerufen und sofort geprüft, ob das Spiel zu Ende ist. Da der Zustand 3 ein Endezustand ist, ist die Bedingung wahr und mittels einer heuristischen Auswertungsmethode wird eine Auswertung zurück gegeben und im rekursiven Durchlauf von Zustand 2 als beste bisherige Auswertung gespeichert(hier wird die Auswertung sofort ohne Vergleiche mit anderen Auswertungen aus anderen Kinderknoten gespeichert, da sie einfach die erste Auswertung die beim Zustand 2 angekommen ist).

6)Danach wird fortgefahren, indem die anderen leeren Felder des Zustandes 2 erkundet werden. So wird der Zustand 4 erstellt.

7)Für diesen Zustand werden die Schritte wie zuvor wiederholt bis der Zustand 5 erreicht wurde. Da der Zustand 5 ein Endezustand ist, wird eine Auswertung dafür berechnet, die an den Zustand 4 übergeben wird. Die zu übergebende Auswertung ist hier 0, da es sich hier um ein Unentschieden handelt.

8)Der Zustand 4 wird die Auswertung an den Zustand 2 übergeben wo sie mit der da gespeicherten Auswertung verglichen wird.

9)Der Zustand 2 ist ein Minimierender Knoten. Anders gesagt, ist der X Spieler hier dran. Das heißt, dass er nach dem Zug sucht, der ihm die niedrigste Auswertung bietet. Und, da 0 größer ist als -10, wird diese neue Auswertung nicht angenommen. Infolgedessen, wird der Zustand 2 die Auswertung -10 und die Feld-oder Zellennummer 7 an den Zustand 1 übergeben.

10)Dieser ganze Ablauf wiederholt sich bis auch der Knoten 6 seine Auswertung hat. Nun kann der Knoten 1 die von den Knoten 2 und 6 kommenden Auswertungen -10 bzw. 0 vergleichen. Der Knoten 1 ist ein maximierender Knoten, da hier der Computer(der maximierende

Spieler) dran ist. Das heißt es wird die größte Auswertung gewählt, nähmlich die 0. Dann wird zwischen 0 und den vom Knoten 11 ankommenden Auswertung -10 verglichen und wiederum wird die 0 gewählt, weil sie größer ist. Dieses Ergebnis wird letztendlich zurückgegeben zusammen mit der Feldnummer(hier die 7), die zu diesem Ergebniss führen könnte.

Mit dem Ergebniss der Minimaxmethode weiß der Computer jetzt wo er seinen Zug machen soll, und zwar das Feld 7.

Minimax funktioniert perfekt mit einem 3x3 Brett. Die Ergebnisse werden blitzschnell zurückgegeben, was ein flüssiges Spielerlebnis ermöglicht. Die Probleme werden erst dann begegnet, wenn versucht wird ein 4x4 Spiel durchzuführen. Beim 3x3 Spiel werden rund 8! Knoten erstellt um den zweiten Zug des Spiels zu berechnen. Diese Zahl erhöht sich auf rund 15! bei einem 4x4 Spiel. Damit ein Rechner so viele Knoten erstellen und berechnen kann, würden mehrere Stunden gebraucht. Aus diesem Grund reicht Minimax alleine nicht.

Memoization:

Das Prinzip von Memoization basiert auf der Tatsache, dass ein einziger Spielzustand mehrfach auftreten kann innerhalb des Minimax-Baums. Memoization nutzt diese Tatsache aus und erstellt eine Liste, wo alle bisher erkundeten Spielzustände zusammen mit den für sie berechneten Auswertungen gespeichert werden. Nun, bevor ein neuer Spielzustand erstellt und erkundet wird, wird zuerst in die Liste geguckt, ob schon ein solcher Spielzustand in der Liste existiert. Falls diese Bedingung erfüllt ist, wird aufgehört dieser Spielzustand weiter zu erkunden und sofort die Auswertung zurückgegeben, die in der Liste gespeichert war.

Eine solche implementierung macht den zweiten Zug eines 4x4 Spiels berechenbar(ca. 24 Sekunden) aber immerhin nicht optimal.

Alpha-Beta pruning:

Alpha-Beta pruning ist die meist verwendete Verbesserung von Minimax. Wenn schon eine Minimaxmethode vorhanden ist, dann lässt sich die Methode sehr leicht so anzupassen, damit sie Alpha-Beta pruning verwendet. Dazu werden zwei Parameter Alpha bzw. Beta benutzt, die die Auswertung des bisher besten berechneten Zuges für den maximierenden bzw. minimierenden Spieler darstellen. Beim ersten Aufruf der Methode bekommen Alpha und Beta die Werte –unendlich bzw. +unendlich. Jeder danach erstellte Knoten erbt diese Werte von seinem Vaterknoten. Wenn eine Endezustand(Blattknoten) erreicht wurde und die Auswertung für diesen Zustand berechnet wurde, werden zunächst ein paar Schritte durchgefürt die etwas anders als beim Minimax erscheinen aber im Prinzip ist es fast die selbe Vorgehensweise. Falls der Knoten ein maximierender Knoten ist, wird geprüft, ob die aktuelle zurückgegebene Auswertung größer als Alpha ist. Wenn diese Bedingung stimmt, dann wird der Wert von Alpha auf den Wert dieser Auswertung gesetzt. Falls der Knoten aber ein minimierender Knoten ist, dann wird geprüft, ob die aktuelle zurückgegebene Auswertung kleiner als Beta ist. Wenn das gilt, dann nimmt Beta dieses Mal den Wert der Auswertung. Das besondere an Alpha-Beta pruning ist, dass ein Knoten wird nur dann Kinderknoten erzeugen, wenn sein Alpha-wert kleiner als sein Beta-wert ist. Wenn der Alpha-wert eines Knotens größer gleich seinem Beta-wert ist, wird sofort aufgehört Kinderknoten für diesen Knoten zu erzeugen auch wenn es noch leere unerkundete Felder im Zustand gibt. Diese Methode ermöglicht dem Program 4x4 Operationen in gerringer Zeit durchführen zu können, indem sie die Anzahl der zu erkundenden Knoten verringert.

Klassengliederung und Projektgestaltung:

Zur Realisierung eines lauffähigen, benutzerfreundlichen Spiels, wurden in diesem Program drei Klassen benötigt, die auf zwei Pakete ttt.engine bzw. ttt.ui verteilt sind. Das erste Paket ttt.engine, beschäftigt sich mit dem Innenleben des Spiels. Das heißt es enthält Klassen mit Methoden zum Auffinden des besten Zuges für den Computer, zum Setzen und Entfernen von Spielfiguren, zur einfachen Darstellung des Bretts auf der Konsole usw.. Abbildung 2 zeigt ein UML-Digramm zur Darstellung der Attribute und Methoden der im Paket ttt.engine befindlichen Klassen. In diesem Program war eine einzige Klasse namens State3 hinreichend, um dies zu erzielen.

Zur Realisierung einer freundlichen Benutzeroberfläche ist das Paket ttt.ui vorgesehen, das zwei Klassen ttt.ui.BoardPanel bzw. ttt.ui.BoardFrame enthält, die Swing Komponenten benutzen und implementieren, um das Spiel darstellen zu können. Abbildung 3 zeigt ein UML-Diagramm für die zwei Klassen. Eine ausführliche Erklärung der Aufgaben aller Methoden und Attribute aller Klassen des Projekts befindet sich in der entsprechenden HTML-Dokumentation.

Abb.[2]: UML-Diagramm für die Klasse ttt.engine.State3

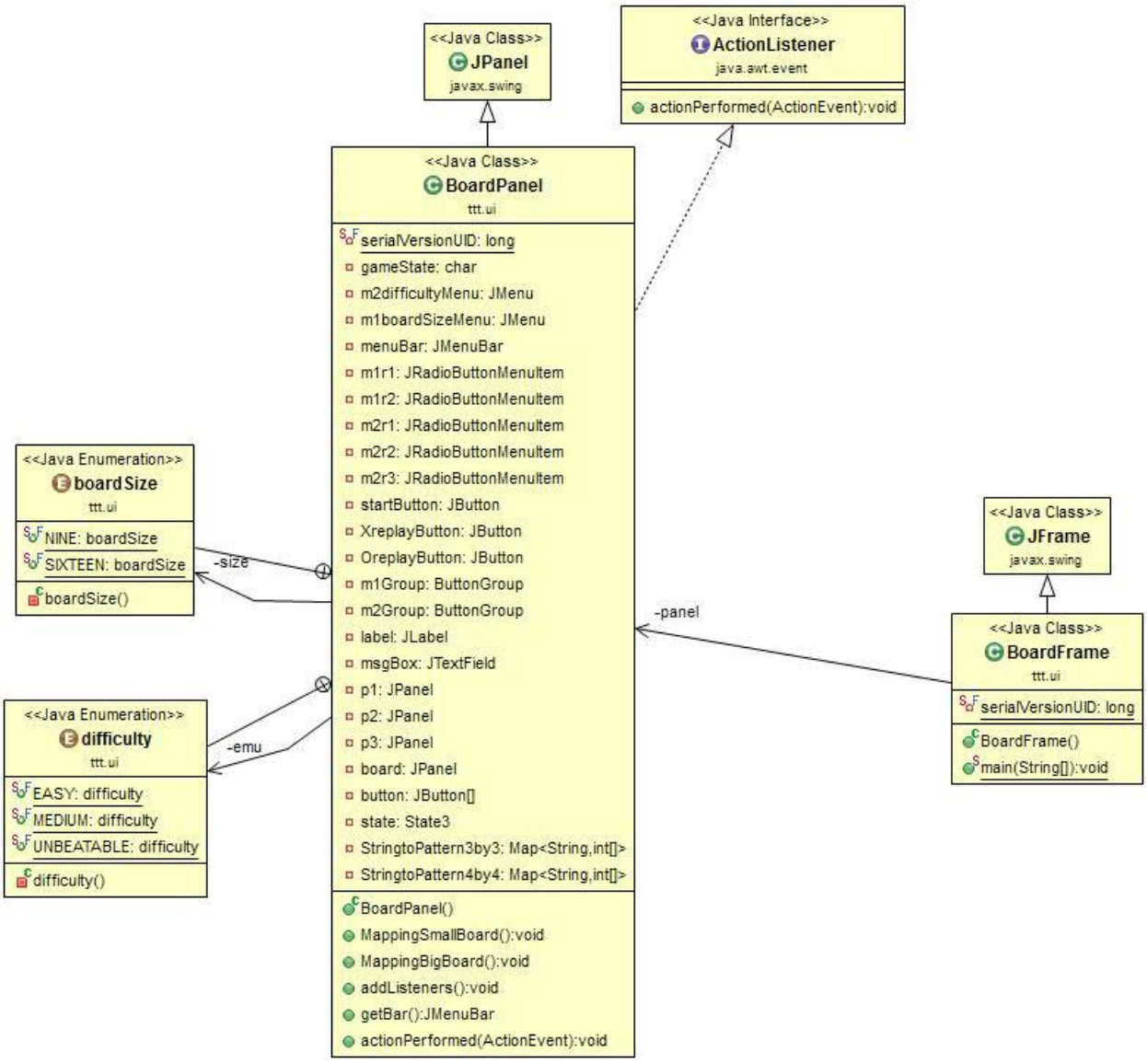


Abb.[3]: UML-Diagram der Klassen ttt.ui.BoardPanel und ttt.ui.BoardFrame

Zusätzliche Optionen:

Da es unmöglich ist gegen einen Minimax oder Alpha-Beta Gegner zu gewinnen, wird es dem Spieler nach ein paar Versuchen sinnlos erscheinen weiter zu spielen. Aus diesem Grund, hat der Spieler die Möglichkeit andere Schwierigkeitsgrade auszupropieren.

-Leicht: Auf dieser Stufe werden alle Züge des Computers zufällig gemacht. Es wird weder nach Siegen gestrebt noch Verluste vermieden. Diese ist die einfachste Form von Gegner und stellt dem Spieler gar keine Herausforderung dar. Die Anweisungen dieses Gegners befinden sich in der überschriebenen Methode actionPerformed der Klasse BoardPanel.

-Mittel: Gegen diesen Gegner ist ein Gewinn viel schwerer zu erzielen als beim leichten Gegner, jedoch ist einer immer noch erzielbar. Beim Start des Spiels wird zunächst geprüft, wer den ersten Zug machen will. Falls der Spieler gewählt hat die erste Runde zu nehmen, hängt der folgende Zug des Computers vom ersten Zug des Spielers ab; Falls der Spieler eine Ecke gewählt hat, wird der Computer die Mitte wählen. Anderseits, wenn der Speiler die Mitte besetzt hat, dann wird der Computer eine zufällige Ecke besetzen. Wenn keine dieser Bedingungen erfüllt ist wird der Computer wiederum eine zufällige Ecke wählen. Gleichermaßen, wenn der Computer den ersten Zug des Spiels hat, wird er wieder auch eine Ecke wählen, da es die optimale Anfangsposition ist. Nach den ersten zwei Zügen des Spiels wird der Computer jedes Mal einfach prüfen bevor er seinen Zug macht, ob er einen Gewinn in dieser Runde erzielen kann. Wenn dies stimmt, wird den Zug gewählt, der ihm den Gewinn ermöglicht. Wenn aber kein Gewinn erzielbar ist, dann wird

geprüft, ob der Gegner in der nächsten Runde gewinnen kann und, wenn ja dann wird das Feld, das zu diesem Gewinn führt blockiert. Wenn keine dieser Bedingungen stimmt, dann wird der Zug einfach an einem freien Feld gemacht. [1]

Für 4x4 Spiele sieht‘s etwas anders aus. Da es hier am Anfang des Spiels ziemlich egal ist, wo ein Spieler seine Spielfigure legt, wird der Computer nur zwei Anweisungen gegeben und zwar, wenn ein Gewinn in dieser Runde erzielbar ist, dann muss es genommen werden. Und zweitens, falls kein Gewinn verfügbar ist, dann muss geprüft werden, ob ein Gewinn des Gegners vorhanden ist und, wenn ja, dann muss dieser blockiert werden. Ansonsten, wird eine zufällige Position gewählt.

Unschlagbar(Minimax mit Alpha-Beta pruning): Die einzigen Erweiterungen oder Änderungen, die hier in der Klasse ttt.ui.BoardPanel unternommen werden, sind ästhetisch. Da Minimax sequenziell die Felder des Bretts prüft, wird immer das erste Feld des Bretts gewählt, wenn alle anderen Felder zum gleichen Ergebnis führen. Und da es am Anfang eines 4x4 Spiels wie schon erwähnt ziemlich egal ist wo die Züge gemacht werden, werden alle Spielfelder die gleiche Auswertung liefern. In Abbildung 4 ist das anhand eines Beispiels erklärt. In diesem Beispiel hat X das Spiel angefangen. Jeder danach von O durchgeführte Zug wird sequenziell gemacht bis X einen möglichen Gewinn in der nächsten Runde erzielen kann. Da wird der Computer das vierte Feld selbst besetzen, um einen gegnerischen Gewinn zu vermeiden. Aus diesem Grund hat der unschlagbare Gegner die Anweisungen in 4x4 Spielen zufällige Felder zu wählen solange sein Gegner weniger als n-1 Spielfigure auf dem Brett hat, wo n*n die Anzahl der Felder ist. Erst, wenn der Spieler größer gleich n-1 Spielfigure auf dem Brett hat, wird die Minimax-Methode ins Spiel kommen. Dadurch wird nichts beim Ablauf von Minimax beeinflußt, sondern es wird nur vermieden einfach

die nacheinander folgenden Felder zu wählen, was dem Spiel ein bisschen Realismus gibt.

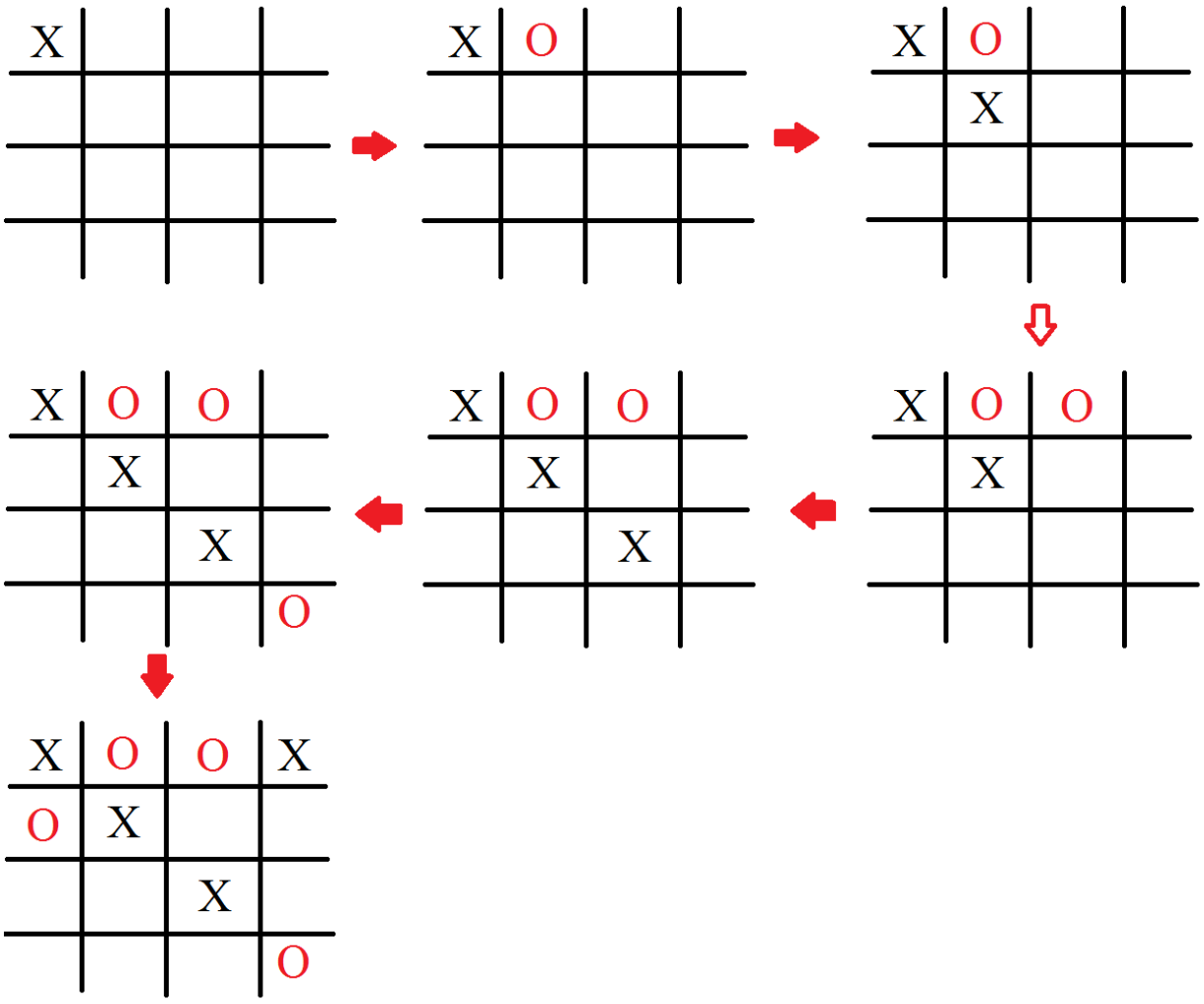


Abb.[4]: 4x4 sequenzieller Spielablauf gegen Minimax Alpha-Beta

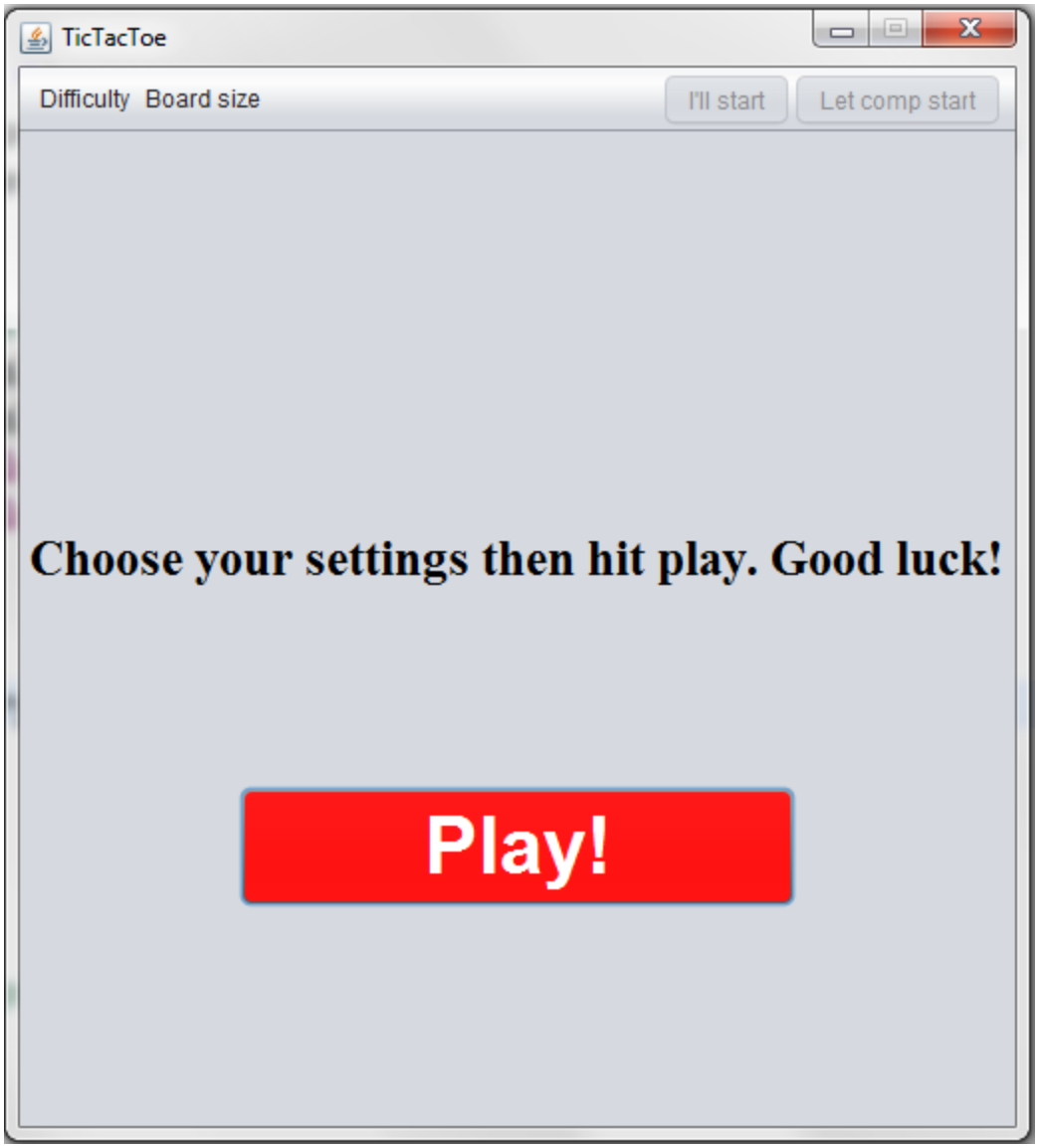
Mögliche Erweiterungen und Verbesserungen:

Nicht alle im Program verwendeten Datentypen sind optimal. Zum Beispiel ein zweidimensionales Feld wäre besser gewesen zur Darstellung des Bretts, da es viel flexibler ist was Prüf-Operationen angeht. Außerdem, hätte int anstatt char benutzt werden können, um die einzelnen Spielfigure darzustellen. Mit diesem Datentyp wären auch andere Prüf-Operationen wie zum Beispiel ein magisches Quadrat möglich gewesen. Schließlich, an vielen Stellen wurden Zeichenketten oder Chars als Rückgabewerte benutzt, zum Bespiel in den Prüf-Methoden(Methoden die den Zustand des Bretts prüfen), wo ein einfacher boolischer Wert angemessener gewesen wäre.

Ein schöner Bestandteil des Programs wäre eine Hinweise-Funktion gewesen. Damit wäre es dem Spieler möglich gewesen, den für ihn optimalen Zug mittels Minimax zu berechnen. Es hätte ein Auswahlfeld in der Menüleiste gegeben, das vom Spieler ausgewählt werden können hätte. Dies hätte dem Spieler dann die beste Möglichkeit für ihn angezeigt.

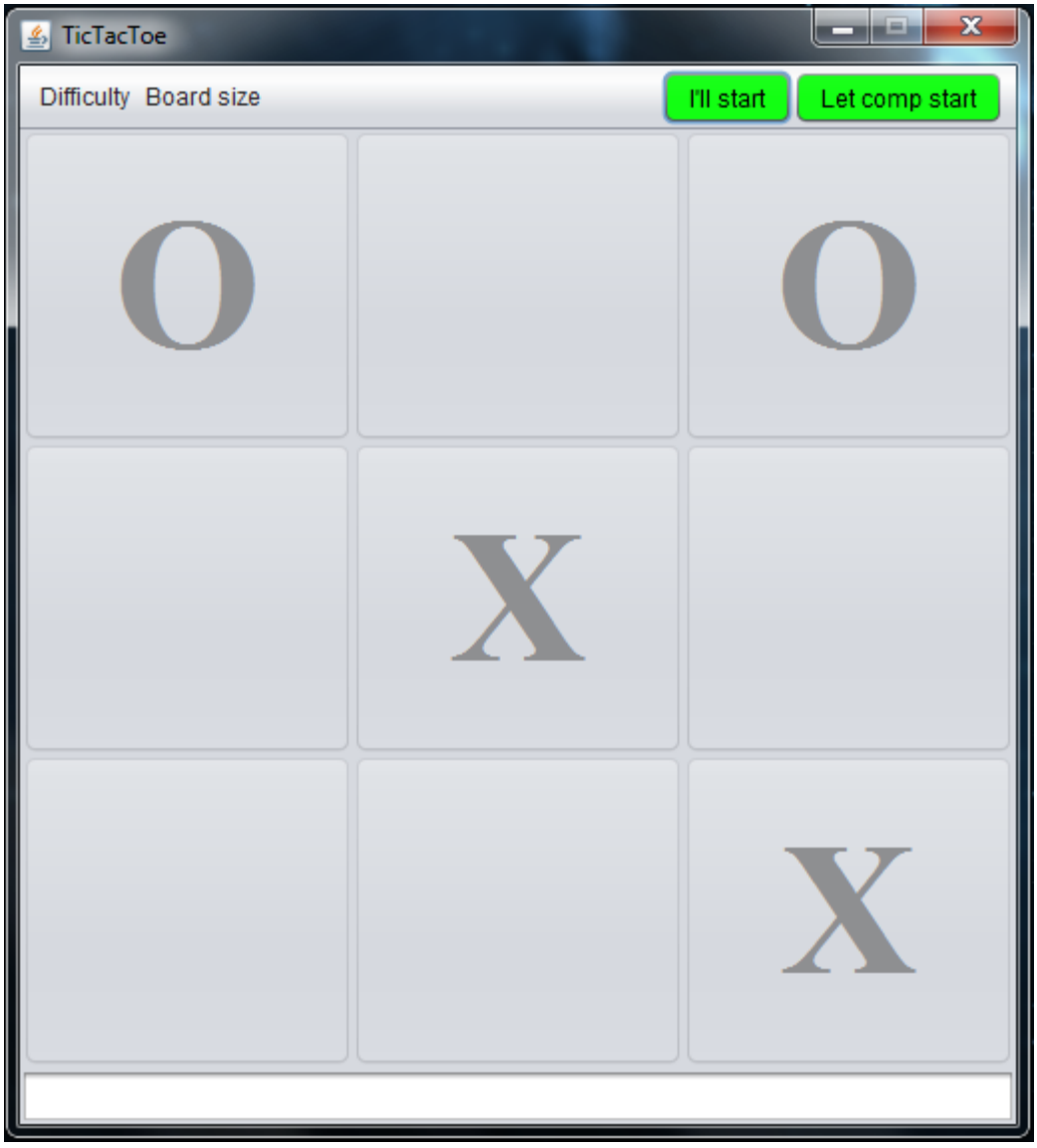
Anleitungen:

Beim Starten des Spiels wird das folgende Fenster geöffnet:



Hier können zunächst die gewünschten Einstellungen wie Schwierigkeitsgrad und Brettgröße gewählt. Danach kann mittels der ‚Play‘ Taste das Spiel gestartet werden.

Ein laufendes Spiel kann jede Zeit neugestartet werden mit den ‚I'll start' bzw. ‚Let comp start' Tasten die auch entscheiden, ob der Spieler bzw. der Computer das Spiel beginnt. Wenn während eines Spiels die Einstellungen geändert wurden, werden die Brett Tasten gesperrt, um unberechenbare Verhältnisse bzw. Ergebnisse des Spiels zu vermeiden. An dieser Stelle werden die zwei Neustart-Tasten grün aufleuchten, damit der Spieler weiß, wo jetzt angeklickt werden soll.



Beim klicken auf eine dieser Tasten, wird das Spiel neugestart.

Quellen und Hilfsmittel:

[1]Praktikum 5

Minimax:

http://neverstopbuilding.com/minimax

Minimax in Java:

https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaGame_TicTacToe_AI.html

http://www.codebytes.in/2014/11/alpha-beta-pruning-minimax-algorithm.html?showComment=1439918131665#c2259839133613795314

http://www.codebytes.in/2014/08/minimax-algorithm-tic-tac-toe-ai-in.html

Alpha-Beta pruning:
http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html

Alpha-Beta pruning für eine einfache heuristische Funktion:

https://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html

Verwendete Fremd-Software:

-**Eclipse Luna**

-**ObjectAid** plug-in für Eclipse

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

_____ / /