**Spielumgebung**

Um Zeit und Arbeit zu sparen sollte mit einer fertigen Implementierung aus dem Internet begonnen werden. Das entsprechende Programm sollte besten Falls leicht anpassbar sein und bereits eine einfach KI enthalten.

Nach ausgiebiger Recherche und kleinen Experimenten mit einigen der gefunden Programmen fiel die Auswahl auf das [hier](http://www.java-online.ch/gamegrid/gamegridEnglish/index.php?inhalt_mitte=gittergames/fourInARow.inc.php) verlinkte Programm. Um das Programm einsetzbar zumachen wurde einige Änderungen vorgenommen um die Spielfeldgröße über globale Variablen einstellen zu können. Außerdem sollte ein dritter Spielmodus eingeführt werden um zwei KIs gegen einander spielen lassen zu können. Die Umsetzung dieses Modus barg jedoch mehr Tücken als ursprünglich angenommen. Das größte Problem bestand darin, dass, die Reihenfolge der Spieler vertauscht wurde. D.h. immer wenn Spieler eins an der Reihe sein sollte wurde ein Stein von Spieler zwei Positionier und umgekehrt.

Da der Fehler nicht behoben werden konnte und nicht klar war in wie weit er sich auf den Lern Erfolg der KIs auswirken würde, fiel letzten Endes der Entschluss, die bisherige Arbeit zu verwerfen und eine eigene Version von Vier-Gewinnt zu programmieren.

Die Klasse *Game* enthält jeweils eine globale Variable für die Spielfeldbreite, eine für die Spielfeldhöhe und eine für die Anzahl von Steinen die nebeneinander gereiht werden müssen um zu gewinnen. Das Spielfeld selbst wird in einem zwei dimensionales Array *board* gespeichert auf das mit einem Getter zugegriffen werden kann. Um die Implementierung von Spiel und KIs zu erleichtern enthält die Spielklasse außerdem je eine Indikator-Variable um anzuzeigen ob das Spielbrett leer ist und ob das Spiel beendet wurde.

Neben einigen Hilfsmethoden mit denen u.a. geprüft wird, ob das Spielende erreicht ist, das Spielbrett zurücksetzt werden kann oder Steine auf dem Spielfeld platziert werden, enthält das Spiel vier unterschiedliche Methoden zum für die Durchführung von Spielen:

* *PlayGame*Hierbei handelt es sich um eine Methode die mit zwei übergebenen Spielern ein einzelnes Spiel durchführt und das Ergebnis auf der Konsole ausgibt.
* *generateDataset*Die Methode bekommt zwei Spieler übergeben und eine Anzahl an durchzuführenden Spielen übergeben und gibt nach entsprechend vielen Spielen ein Datenset zurück.
* *trainQPlayer*Die Methode bekommt zwei Spieler und die Anzahl zu spielender Spiele übergeben. Außerdem erhält die eine Information darüber, ob die Spieler abwechselnd beginnen sollen oder nicht.
* *playTurnament*Die Methode erhält die gleichen Eingabe-Paramter wie trainQPlayer. Auch die Funktionsweise ist im Wesentlichen gleich, mit dem Unterschied, dass falls ein Q-Player am Turnier teilnimmt eine Werte in seiner Datenbank mehr verändert werden. Diese Methode dient somit vor allem zur Beurteilung der Trainingsergebnisse. Auf der Konsole werden Spielzüge, Ergebnisse von einzelnen Spielen sowie summierte Kennziffern über das gesamte Turnier ausgegeben.

**NormalKI2**

Damit der KIs später automatisiert lernen können wird eine weiter KI benötigt, die bereits spielfähig ist. Das oben erwähnte 4-Gewinnt-Spiel enthält bereits eine gute KI, deren Algorithmus für die neue Spiel-Implementierung übernommen wurde.

Der enthaltene Algorithmus basiert auf einfachen if-then-Abfragen und Schleifen. Dabei wird zunächst geprüft, ob das Spielfeld leer ist. In diesem Fall wirf sie in die mittlere Spalte um eine möglichst gute Ausgangslage zu schaffen. Wenn bereits Steine auf dem Spielfeld liegen, prüft die KI als erstes ob sie selbst gewinnen kann. Falls nicht wird abgefragt ob der Gegner gewinnen kann und der eigene Stein entsprechend so geworfen, um dies zu verhindern.

Sofern keine dieser Fälle vorliegt, wird eine defensive Spielweise gewählt. Für ein klassisches Vier-Gewinnt-Spiel wird nun geprüft, welche Züge es dem Gegner erlauben würden drei bzw. zwei seine Steine nebeneinander zu platzieren. Alle entsprechenden Züge werden als Spalten in einem Array possibleSolutions zwischengespeichert. Da Züge die es ermöglichen drei Steine neben einander zu platzieren natürlich auch zwei Steine neben einander zur Folge haben, werden sie doppelt in das Array aufgenommen. Anschießend werden alle Züge aus possibleSolutions entfernt, die es dem Gegner in seinem nächsten Zug ermöglichen (über eine Diagonale oder Horizontale) zu gewinnen. Im nächsten Schritt werden alle verbleibenden Spalten, die nicht am Rand liegen ein weiteres Mal in das Array kopiert, um dann einen zufälligen Zug auszuwählen. Falls possibleSolutions leer ist wird nach einer Lösung gesucht, die in erster Linie erlaubt ist und dem Gegner nach Möglichkeit nicht ermöglicht, im nächsten Zug zu gewinnen.

Die Klasse NoramlKI2 enthält außerdem eine Variable epsilon, die festlegt wie viel Prozent der Spielzüge rein zufällig ausgewählt werden sollen. Diese Strategie ist bekannt als epsilon-greedy-Strategie und soll eine größere Varianz an Spielzügen ermöglichen.