Übungszettel 6 Gruppe 100

**Aufgabe 1: Testen der AdvancedAI**

Siehe testAufgabe1.TudAdvancedAITest.java

**setUp()**

Vor jedem Test werden Spielfelder für das Testen initialisiert, außerdem enthält die Testklasse als Attribut eine Test-AI.

**adaptiveTreeDepth()**

Als erstes testen wir die Methode AdvancedAI.getMasDepth(Board). Dafür werden der Methode verschieden große Spielfelder(Boards) mit jeweils 5,11,23 und 59 freien Kanten übergeben, um alle Fälle zu überprüfen.

**selectsFirstBest()**

Als nächstes testen wir ob die essentielle Funktion AdvancedAI.getNextTurn(Board) bzw. AdvancedAI. supportPlayer(Board, String) (Da get nextTurn nichts tut außer supportPlayer aufzurufen) in einem Spielfeld die erste Kante der besten Kanten wählt, in dem wir sie auf ein leeres Feld aufrufen, damit wir sehen ob Großteile des Codes arbeiten. Bei einem leeren Feld sollte hier 1 gewählt werden.

**selectsBestSimple()**

Nun testen wir, ob die AI in der Lage ist, die Kante zu finden, die ihr einen Punkt gibt für das triviale Beispiel:

[1, 3, 4] also

\* - \* 2 \*

| | 5

\* 6 \* 7 \*

8 9 10

\* 11 \* 12 \*

Hier sollte 6 gewählt werden. Damit wird die Funktion des Baumes in den ersten Blättern getestet.

**selectsBestAdvanced()**

Betrachte folgendes Beispiel:

{ 1, 2, 3, 5, 8, 10, 12 }

\* - \* - \*

| 4 |

\* 6 \* 7 \*

| 9 |

\* 11 \* - \*

Wir testen nun, ob die AI in der Lage ist, die Kante zu finden, mit der sie im nächsten Zug keinen Punkt verliert. Damit wird also getestet ob der Entscheidungsbaum auch in der 2. Ebene richtig funktioniert.

Hier ist dann auch schon 100% des Codes durchgeführt worden, um sicherzustellen, dass auch ein weiteres Vorausschauen der AI funktioniert und die in moderatem Zeitaufwand geschieht, betrachten wir die folgenden, komplexeren Testfälle:

{ 1, 2, 3, 5, 8, 10, 12 } also

\* - \* - \* - \*

| 5 6 7

\* 8 \* - \* - \*

| 12 13 14

\* - \* - \* - \*

18 19 20 21

\* - \* - \* - \*

Nur die Auswahl einer der Kanten 18-21 führt zum Sieg, da die AI die erste der besten Kanten wählt, sollte hier 18 das Resultat von AdvancedAi.getNextTurn(Board, String) sein, hier muss die AI mindestens 4 Züge des Gegners im Voraus berechnen, der Zeitaufwand wird auf 10 sec gekappt.

**advancedStrategyTestLarge()**

Zuletzt betrachten wir ein ähnliches Beispiel, in einem größeren Feld:

{ 1, 2, 3, 4, 5, 11, 12, 13, 14, 18, 19, 20, 21, 27, 28, 29, 30, 31, 37, 38, 39, 40 } also

\* - \* - \* - \* - \*

| 6 7 8 9

\* 10 \* - \* - \* - \*

| 15 16 17 |

\* - \* - \* - \* 22 \*

23 24 25 26 |

\* - \* - \* - \* - \*

32 33 34 35 36

\* - \* - \* - \* - \*

Hier führen nur Markierungen 32-36 zum Sieg, die AI muss mindesten 5 Züge des Gegners im voraus berechnen, wobei 18 Kanten zu prüfen sind (Dementsprechend nur eine Baumtiefe von 4 gewährleistet ist). Dementsprechend schlägt die Implementierung für diesen Test fehl, dafür ist der Rechenaufwand moderat und es wird wie zu erwarten Kante „6“ gewählt, da für Tiefe 4 alle Kanten gleichen Wert haben.