Abgabe von Kilinc, Abdülkerim und Henrich, Nicolas

Dokumentation TicTacToe

Vorüberlegungen

In der Vorlesung hatten wir die Reduzierung des TicTacToe-Spielbaums durch Ausnutzen von Symmetrien des Spielbretts skizziert. Daraufhin haben wir händisch versucht diesen Spielbaum auf einem Blatt darzustellen. Hierfür haben wir uns Schicht für Schicht die Boards des Spielbaumes aufgemalt und versucht, durch Drehen und Spiegeln in verschiedene Richtungen visuell gleiche Boards zu identifizieren und diese dann aus dem Baum zu streichen. Das Ergebnis dieses fertigen Spielbaums ist im Anhang zu finden. Durch dieses Vorgehen hatten wir im Anschluss bereits einige Methoden gefunden, um die sich durch Symmetrien doppelnden Boards zu identifizieren.

Uns ist aufgefallen, dass in der ersten Ebene, das heißt bei dem ersten Zug der Baum von 9 möglichen Boards auf 3 Boards verkürzt werden kann. Diese sind:

```
['X', '_', '_'] ['_', 'X', '_'] ['_', '_', '_']
['_', '_', '_'] ['_', '_', '_'] ['_', 'X', '_']
['_', '_', '_'] ['_', '_', '_'] ['_', '_', '_']
```

Auf diese drei Felder kommt man, indem man das Board beispielsweise über verschiedene Achsen spiegelt oder um ein Vielfaches von 90° dreht.
Durch Ausnutzen von Spiegelungen und Rotationen kann so der Spielbaum weiter reduziert werden. Das der Baum, den wir als Ergebnis durch unser visuelles Kürzen erzeugt haben ist als Dokument angehängt.
Stellen wir uns vor, dass jedes Feld im TicTacToe Spielbrett eine bestimmte Nummer zugewiesen bekommt. Das Ausgangsspielbrett sehe wie folgt aus:

1	2	3
4	5	6
7	8	9

Die erste Symmetrie-Art, die Rotation, die wir nun betrachten wollen, hat drei bestimmte Fälle: 90°, 180°, 270°. Das Ausgangsspielbrett hat in den jeweiligen Fällen die bestimmte Spielsituation:

Bei 90°:

7	4	1
8	5	2
9	6	3

Bei 180°:

9	8	7
6	5	4
3	2	1

und Bei 270°:

3	6	9
2	5	8
1	4	7

Die zweite Symmetrie-Art, die Spiegelung, hat auch vier bestimmte Fälle: vertikal, horizontal, diagonal von oben links nach unten rechts und diagonal von oben rechts nach unten links. Das Ausgangsspielbrett hat in den jeweiligen Fällen die bestimmte Spielsituation:

Abgabe von Kilinc, Abdülkerim und Henrich, Nicolas

Vertikal: horizontal: diagonal (o. links > u. rechts): diagonal (o. rechts > u. links):

3	2	1
6	5	4
9	8	7

7	8	9
4	5	6
1	2	3

1	4	7
2	5	8
3	6	9

9	6	3
8	5	2
7	4	1

Anhand dieser Überlegungen sind wir zu der Implementierung übergegangen. Dabei haben wir diese Symmetriearten ausgenutzt, was im Folgenden näher erläutert wird.

Implementierung des reduzierten Spielbaums und Dokumentation der wesentlichen Eigenschaften:

Als Codevorlage wurde das in Moodle bereitgestellte Python Script Spielbaum TicTacToe 3 verwendet. Außerdem wurden die beiden Skripte TicTacToeMinimax und TicTacToeAlphaBeta verwendet.

Das Aufbauen des Spielbaumes in dem bereitgestellten Script erfolgt in der setUpTree() Methode. Hierbei werden neue Boards (Variable x) erstellt und in einer for-Schleife dem Baum hinzugefügt. An dieser Stelle haben wir eine if Prüfung eingefügt, um festzustellen, ob das erstellte Board bereits existiert. Für diese Prüfung haben wir die has_symmetry() Methode erstellt. Diese sollte im ersten Schritt prüfen, ob es bereits einen Knoten gab, der unter Berücksichtigung der Symmetrien das gleiche Board enthält.
Die Eingabeparameter der has_symmetry() Methode sind der Spielbaum (self) und das zu prüfende Board (board). Im ersten Schritt werden nun alle möglichen symmetrischen Darstellungen des Boards erzeugt. Da wir festgestellt hatten, dass manche Symmetrien nur durch mehrfaches Drehen und Spiegeln erreicht werden können, haben wir jede neue Symmetrie auch auf die bereits bestehenden angewandt. So hatten wir am Ende 64 Boards. Hierfür wurde dieser Code erstellt (Auszugbeispiel für horizontale und vertikale Spiegelung):

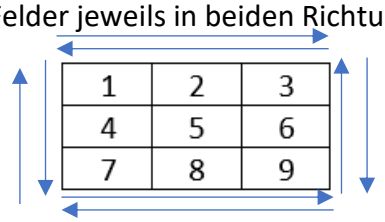
```
boards = [data.board]
# horizontal gespiegelt
boards.append([boards[0][2], boards[0][1], boards[0][0]])
# vertikal gespiegelt
brd = []
for i in boards:
    brd.append([[i[0][2], i[0][1], i[0][0]],
                [i[1][2], i[1][1], i[1][0]],
                [i[2][2], i[2][1], i[2][0]]])
for i in brd:
    boards.append(i)
```

Im Codebeispiel wird zunächst das Board horizontal gespiegelt (durch Tauschen der einzelnen Zeilen) und im nächsten Schritt vertikal gespiegelt (durch tauschen der einzelnen Board-Positionen). Da das horizontale Spiegeln die einzige Symmetrie ist, die in einer Zeile übersichtlich implementiert werden kann, haben wir die Restlichen Symmetrien auf die gleiche Weise wir die vertikale Spiegelung implementiert.

Das Ergebnis dieser Methode waren 765 Knoten in dem Spielbaum. Dies ist die minimale Anzahl an Boards, die nach Betrachtung der Symmetrien noch vorhanden sein kann[1].
Durch einen Aufruf der Methode ohne Board ist uns aufgefallen, das die has_symmetry() Methode einige Boards doppelt erstellt hat und auf diese Weise unnötig viel Zeit benötigte. Um diese Zeit zu verringern haben wir durch das hinzufügen einer Schleife am Ende der Methode die Doppelt erstellten Boards entfernt:

```
new_boards = []
for i in boards:
    if i not in new_boards:
        new_boards.append(i)
boards = new_boards
```

An dieser Stelle haben wir uns die verbleibenden Boards angesehen und uns ist ein Muster aufgefallen. Es blieben nur 8 Boards, die sich, dadurch auszeichneten das sie auf jeder Kante des Spielfeldes die ersten drei Felder jeweils in beiden Richtungen darstellten:



Diese 8 Boards können durch horizontales Spiegeln, eine 90° Drehung und Spiegeln über eine Achse von der oberen rechten Ecke zu der unteren Linken Ecke erstellt werden.
Daraufhin haben wir den Code noch etwas aufgeräumt(die jeweils 2 for-Schleifen zu jeweils einer zusammengefasst):

```
# 90° Drehung
b_len = len(boards)
for i in range(b_len):
    boards.append([[boards[i][2][0], boards[i][1][0], boards[i][0][0]],
                   [boards[i][2][1], boards[i][1][1], boards[i][0][1]],
                   [boards[i][2][2], boards[i][1][2], boards[i][0][2]]])
```

Dies diente jedoch lediglich der Übersicht und hat die Laufzeit sogar wieder marginal erhöht. Ein letzter Versuch den Code zu optimieren war das manuelle (hart codierte) erstellen der 8 Boards, doch dies erhöhte die Laufzeit wieder, weshalb wir uns gegen diese Lösung entschieden.

In der folgenden Tabelle sind die Optimierungen mit den jeweiligen Laufzeiten dargestellt. Es wurden jeweils 10 Bäume erstellt, die Zeiten gemessen. Dargestellt ist jeweils der Durchschnitt, der Längste und der Kürzeste Lauf:

Version	Zeit(durchschnitt)	Zeit(max)	Zeit(min)
Ohne Symmetrie Betrachtung	7,1330s	7,7917s	6,4415s
Mit erster Version (64 Prüfungen)	2,772s	2,864s	2,727s
Mit kürzen der 64 Symmetrien	0,4286s	0,4485s	0,4240s
Nur 3 Schritte (lediglich 8 Boards erstellt)	0,3769s	0,3925s	0,3721s
Code aufgeräumt (übersichtlicher)	0,3788s	0,3977s	0,3752s
Hartcodiert (Verschlechterung)	0,3935s	0,4179s	0,3887s

[1] https://oeis.org/A008907

Abgabe von Kilinc, Abdülkerim und Henrich, Nicolas

Eine Erweiterung, die wir noch „Optional“ hinzugefügt haben, ist die Rückgabe des gespiegelten Knotens. In der setUpTree() Methode haben wir eine Boolean Variable „short“ erstellt die zwei if-Abfragen steuert über die der Knoten zu dem es ein Symmetrisches Pendant gibt zwar erstellt wird, jedoch die ihm nachfolgenden nicht und stattdessen dem Knoten die Knotennummer seines Pendants als data.symmetry_id übergeben wird. So kann auf Wunsch (short=True) nachfollzogen werden welche Knoten jeweils die Pendants bilden. Dies geht jedoch wieder zulasten der Laufzeit, da die symmetrisch bereits vorhandenen Knoten dennoch erstellt werden müssen.

Bei der Bearbeitung des Aufgabenteil b ist uns aufgefallen das die beiden bereitgestellten Methoden durch simples Kopieren in die vorhandene TicTacToeGame funktionieren. Wir haben diese jedoch darüber hinaus noch optimieren können, da die vorhandene getValue() Methode von Ihrem Inhalt her bereits einen Maximin/Minimax[2] darstellt. Damit die Methode auch mit den Boards, die durch die Symmetrien Betrachtung nicht erstellt werden korrekt behandelt, haben wir die has_symmetry() Methode um den Aufruf der getValue() Methode geschachtelt:

```
def maximin_neu(self, board):
    return
self.getValue(self.gametree.get_node(self.has_symmetry_ultra_short(board)))
```

Zudem wurde um die korrekte Betrachtung des Minimax zu ermöglichen die Variable Symbol sowohl in die minimax_neu() Methode als auch in die getValue Methode implementiert. In der getValue() Methode ersetzt sie lediglich die vorher hart codierten Sympole (X und O). Die neu Implementierten Methoden sind Laufzeittechnisch erheblich effektiver als die bisher existierenden, da sie den reduzierten Spielbeum nutzen. So braucht der neue Maximin ca. 52 Microsekunden wohingegen der ursprüngliche 3523691 Microsekunden brauchte.

Bei der Implementierung des Alpha-Beta Search konnte im Wesentlichen genau gleich vorgegangen werden. Für diesen musste jedoch der Alpha und Beta Wert noch in die getValue() Methode implementiert werden, weshalb wir eine neue Methode (getValue_alpha_beta()) erstellten. Diese haben wir umstrukturiert und nach der isWinningBoard Abfrage zunächst das aktuelle Symbol geprüft. Daraufhin wird dann für jeden Child-Knoten erneut die getValue_alpha_beta() Methode aufgerufen und sobald der abgefragte Wert den „passiven“ Wert dann über oder untersteigt wird kein weiteres Child mehr überprüft und die for-Schleife die durch die einzelnen Child Elemente geht durch den break aufrug beendet.
Auf diese Weise konnte der Alpha Beta Search in 12 Microsekunden durchgeführt werden wo der ursprüngliche noch 149669 Microsekunden benötigte.

Wir haben gemeinsam an den allen Aufgaben gearbeitet. Das meiste des Codes wurde auf diese Weise in der Vorlesungszeit erstellt und das Übrige zu hause. Bei der Dokumentation hat jeder 2 Seiten verfasst.

[2] Minimax-Theorem