Softwarepraktikum SS 2018

Assignment 3

Group - 3

Ramil Sabirov	369500	ramil.sabirov@rwth-aachen.de
Joel Choi	345575	joel.choi@rwth-aachen.de
Eric Remigius	366895	eric.remigius@rwth-aachen.de

Task 1

Unsere Implementation des Minimax-Algorithmus ist eine sogennante Paranoid-Suche und nutzt im wesentlichen drei Methoden und zusätzlich die Bewertungsfunktion. Diese wären:

- Starting Player
- Min Player
- Max Player

Der Callgraph für den Algorithmus mit den Methoden ist in der Abbildung 1 zu sehen. Starting Player ist der Einstiegspunkt der rekursiven Baumsuche. Dieser

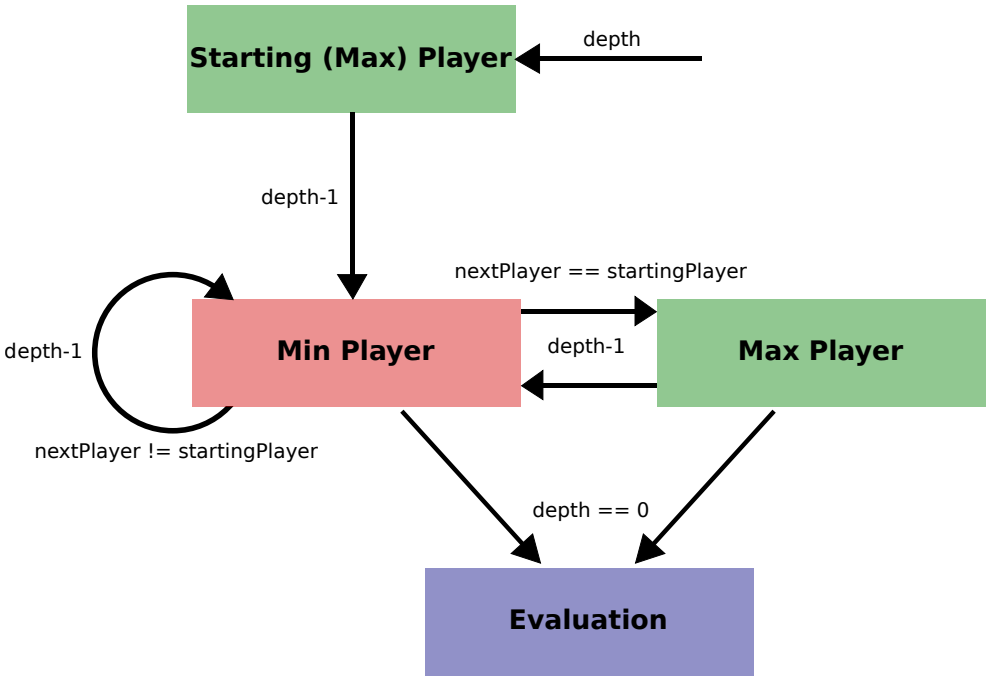

Figure 1: Callgraph des Minimax-Algorithmus

erhält als Parameter das Tiefenlimit und spielt die Rolle des Maximalspielers, also er versucht den besten Zug zu finden, welcher den Stellungswert maximiert. Er unterscheidet sich vom Max-Spieler in tieferen Rekursionsebenen, denn er aktualisiert den besten Zug (welcher am Ende zurückgegeben werden soll), sollte ein neu erforschter Teilbaum eine Verbesserung gegenüber den alten Ergebnissen liefern.

Ansonsten rufen sich Min- und Max-Spieler gegenseitig auf, mit jeweils dekrementierter Restrechentiefe. Dabei versucht der Max-Spieler den Stellungswert zu maximieren, wohingegen der Min-Spieler minimiert. Da wir nicht zwingend ein Spiel mit zwei Spielern vorliegen haben und der Paranoid Strategie folgen, gibt es nur einen Max-Spieler und unter Umständen mehrere Min-Spieler. Folgt also nach dem Min-Spieler nicht der Max-Spieler ($nextPlayer! = startingPlayer$), dann ruft der Min-Spieler einen weiteren Min-Spieler (und damit dieselbe Methode mit dekrementierter Tiefe) auf.
Ist die Resttiefe ausgeschöpft ($depth == 0$), dann bricht die Rekursion ab und die Abbruchstellung wird evaluiert, ausgehend von aktueller Evaluationsfunktion.

Task 2

Der Alpha-Beta Algorithmus ist nur eine kleine Anpassung des Minimax-Algorithmus. Mit zwei zusätzlichen Parametern und einer Abfrage mehr pro Aufruf des Min- oder Max-Spielers. Dementsprechend ist die Logik und der Callgraph gleich. Wir haben bereits eine Schnittstelle für das Sortieren der Züge eingebaut, was für zukünftige Assignments wahrscheinlich eine Rolle spielen wird. An dieser Stelle wollen wir dies jedoch *zunächst* nicht weiter thematisieren.

Um das *Alpha-Beta-Pruning* deaktivieren zu können, haben wir einen Commandline Parser geschrieben. Um diesen zu nutzen, muss man nur eine *CliOption* erstellen und diese dem Parser hinzufügen. Nach einem Aufruf der *parse(String[] args*-Methode, kann man die angegebenen Werte der Optionen aus den Objekten der Optionen auslesen. Sollte eine Option nicht gesetzt sein und einen weiteren Parameter haben (z.B. -s servername), wird ein Default-Wert zurückgegeben. Wenn nicht alle, als *mandatory* markierten Optionen richtig gesetzt werden, gibt der Parser *false* zurück und das Programm kann beendet werden. Eine Hilfeseite wird automatisch ausgegeben.

Task 3

Um die Zeiten zu messen, haben wir eine einfache Stoppuhr Klasse implementiert. Diese hat eine Auflösung im Bereich von Nanosekunden.

Für die eigentliche Performance Messung haben wir die *PerfLogger* Klasse entworfen. Mit dieser Klasse, können wir über die *start-* und *stopTotal()*-Methoden die Zeit messen, die insgesamt für einen Zug gebraucht wird. mit den *start-* und *stopInner()/stopLeaf()*-Methoden können wir Zeit messen, die einzelne Knoten brauchen. Dabei wird zwischen inneren Knoten und Blättern unterschieden. Blätter sind hierbei alle Knoten, auf denen die Evaluierungsfunktion ausgeführt wird. Außerdem inkrementieren diese Methoden einen Zähler, sodass wir wissen, wie viele Knoten welcher Art besucht wurden. Auch wird die gemessene Zeit gespeichert. So lässt sich später dann die maximale, minimale und durchschnittliche Dauer berechnen.

3.1 Performance Vergleich

Wir haben auf einer Map 1000 mal in Folge den besten Move berechnen lassen, ohne dass sich die Map verändert, und die Performance Daten für jeden Zug gemessen. Die durchschnittlichen Ergebnisse sind in den Folgenden Diagrammen abgebildet:

Wenn nicht in die Tiefe gerechnet wird, unterscheiden sich die beiden Algorithmen wenig. Der *MinMax*-Algorithmus ist sogar minimal schneller,wie Abbildungen 2 und 3 zeigen.

Ab einer Tiefe von 2 sind jedoch deutliche Unterschiede bemerkbar. Wie wir in Abbildung 4 sehen, braucht *Alpha-Beta-Pruning* wesentlich weniger Zeit als der *MinMax*-Algorithmus. In Abbildung 5 sehen wir auch den Grund dafür: Der *Alpha-Beta*-Algorithmus besucht sehr viel weniger Blätter.

Sobald bis Tiefe 3 oder weiter gerechnet wird, verdeutlicht sich das Bild, dass sich schon auf Tiefe 2 zeigte. Allerdings besuchter der *Alpha-Beta*-Algorithmus nun auch weniger innere Knoten, wie Abbildungen 6 und 7 belegen.

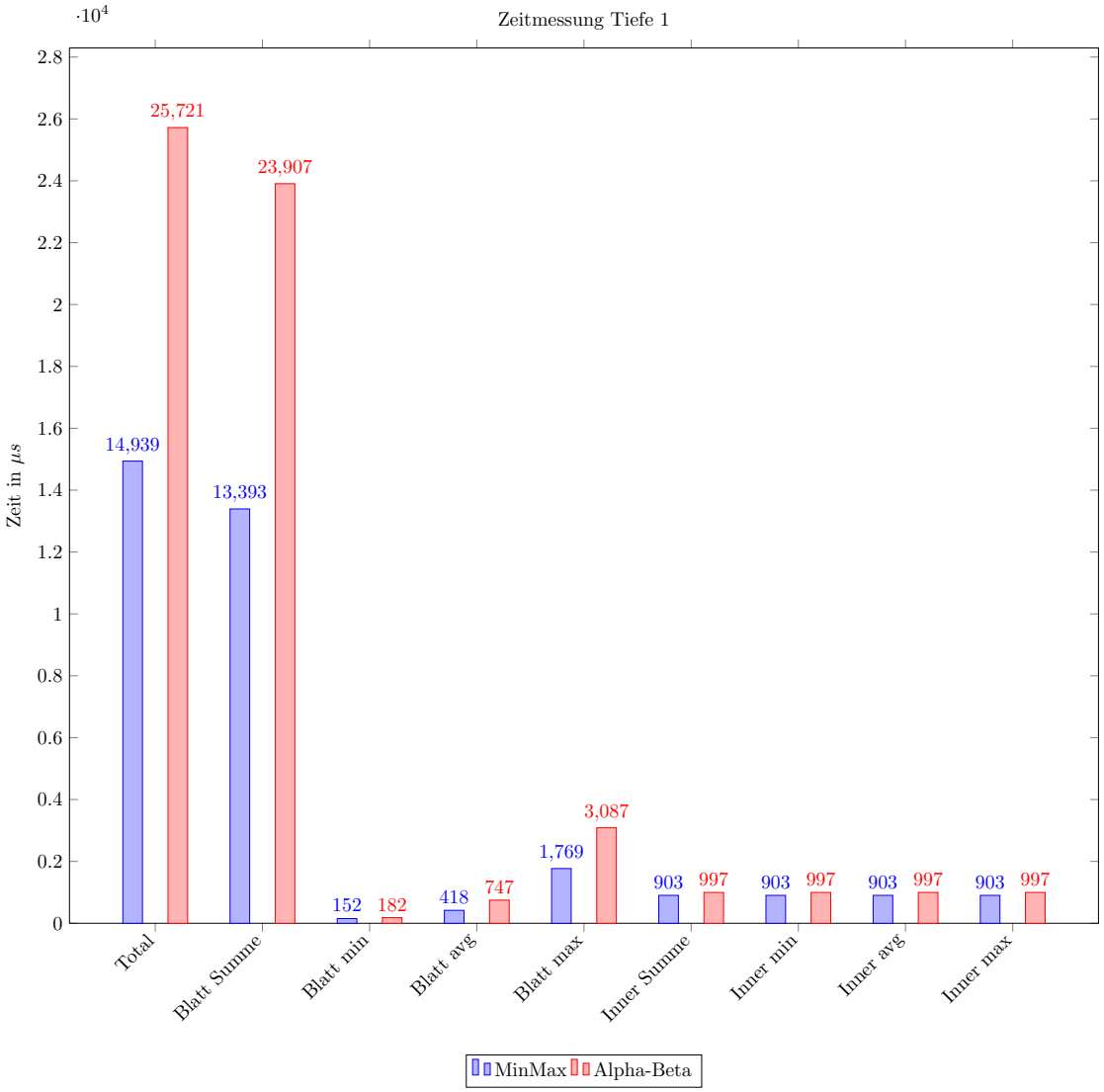

Figure 2: Gemessene Zeiten bei Rechnung bis Tiefe 1

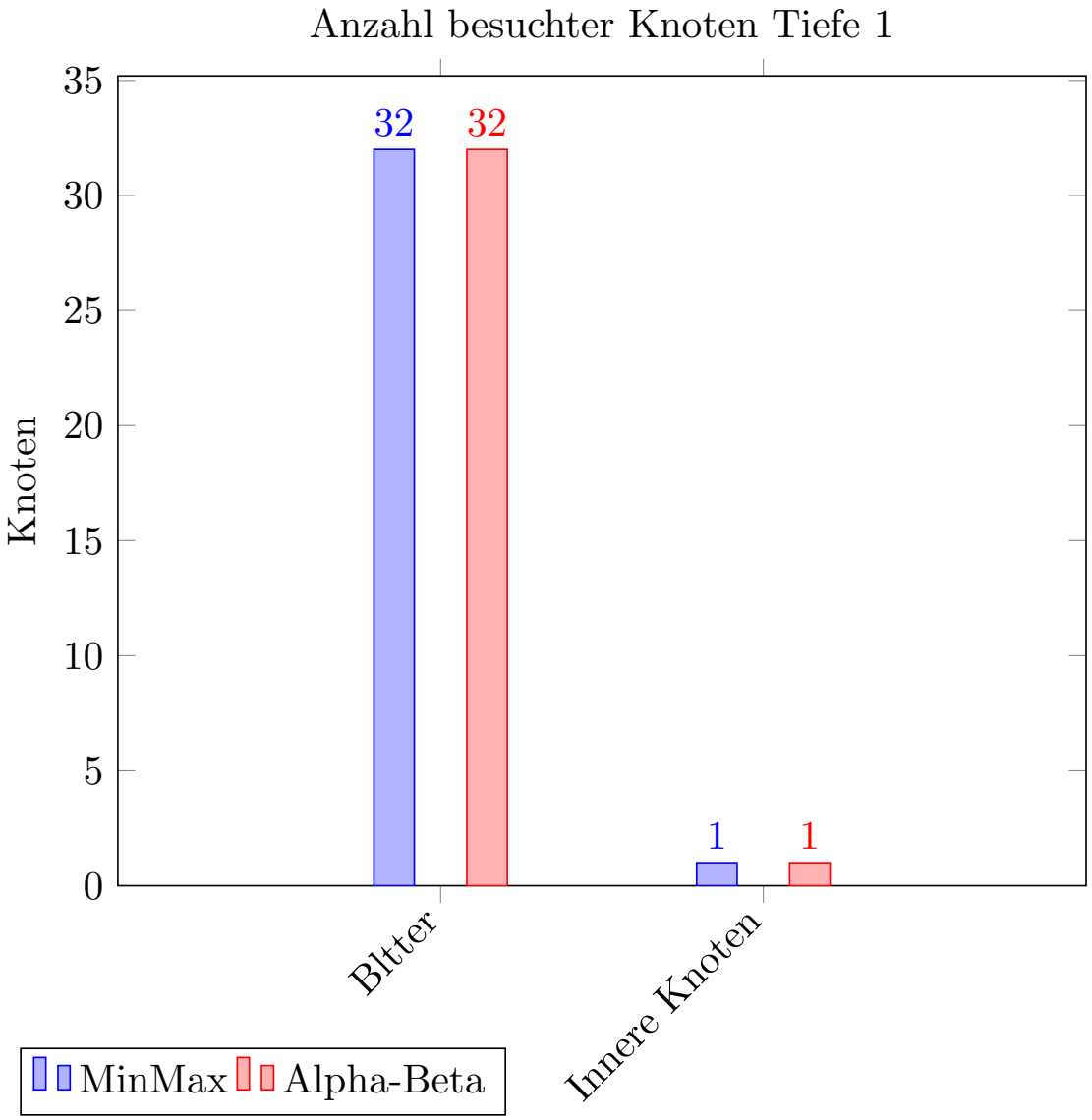

Figure 3: Anzahl besuchter Knoten bei Rechnung bis Tiefe 1

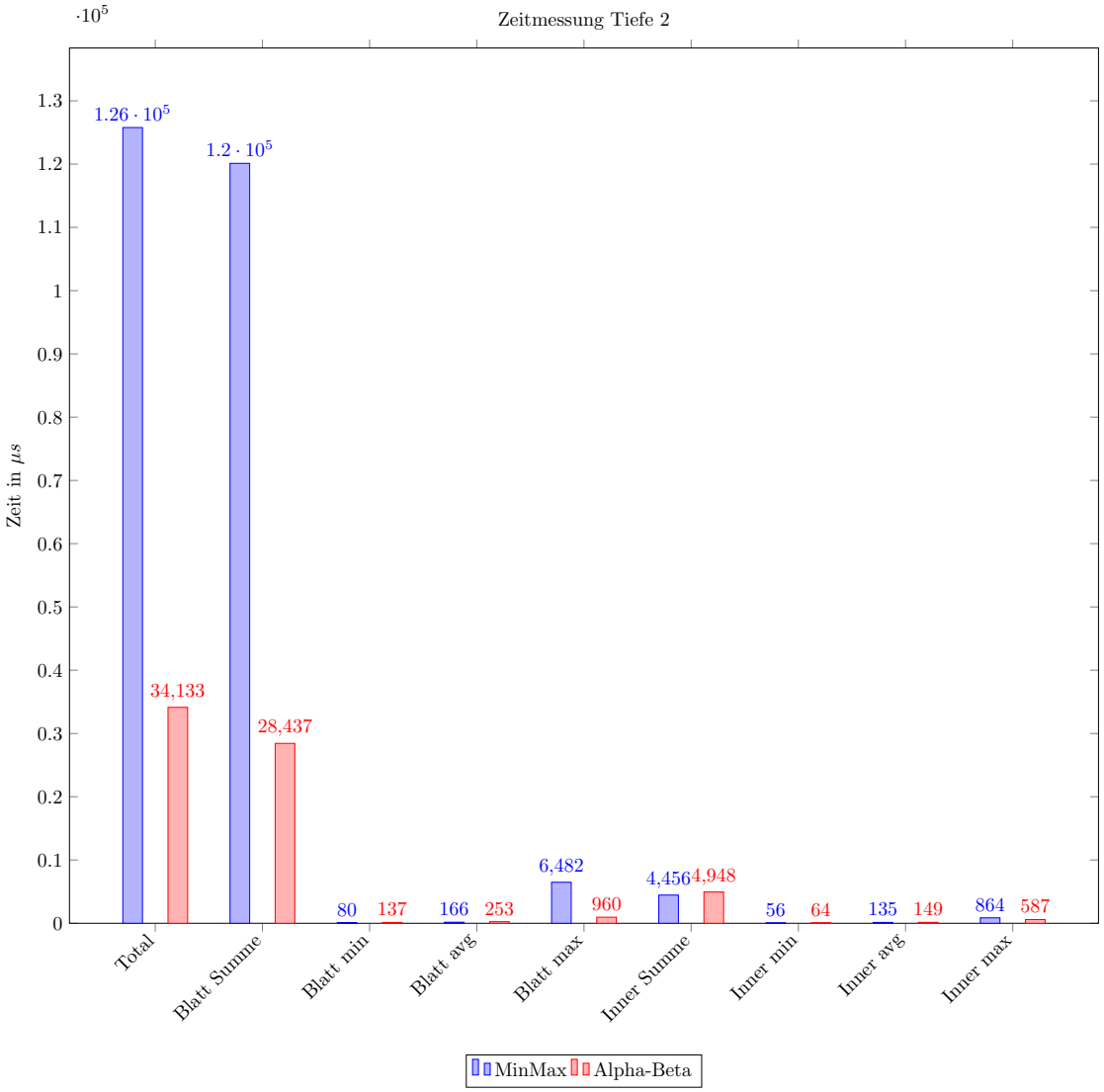

Figure 4: Gemessene Zeiten bei Rechnung bis Tiefe 2

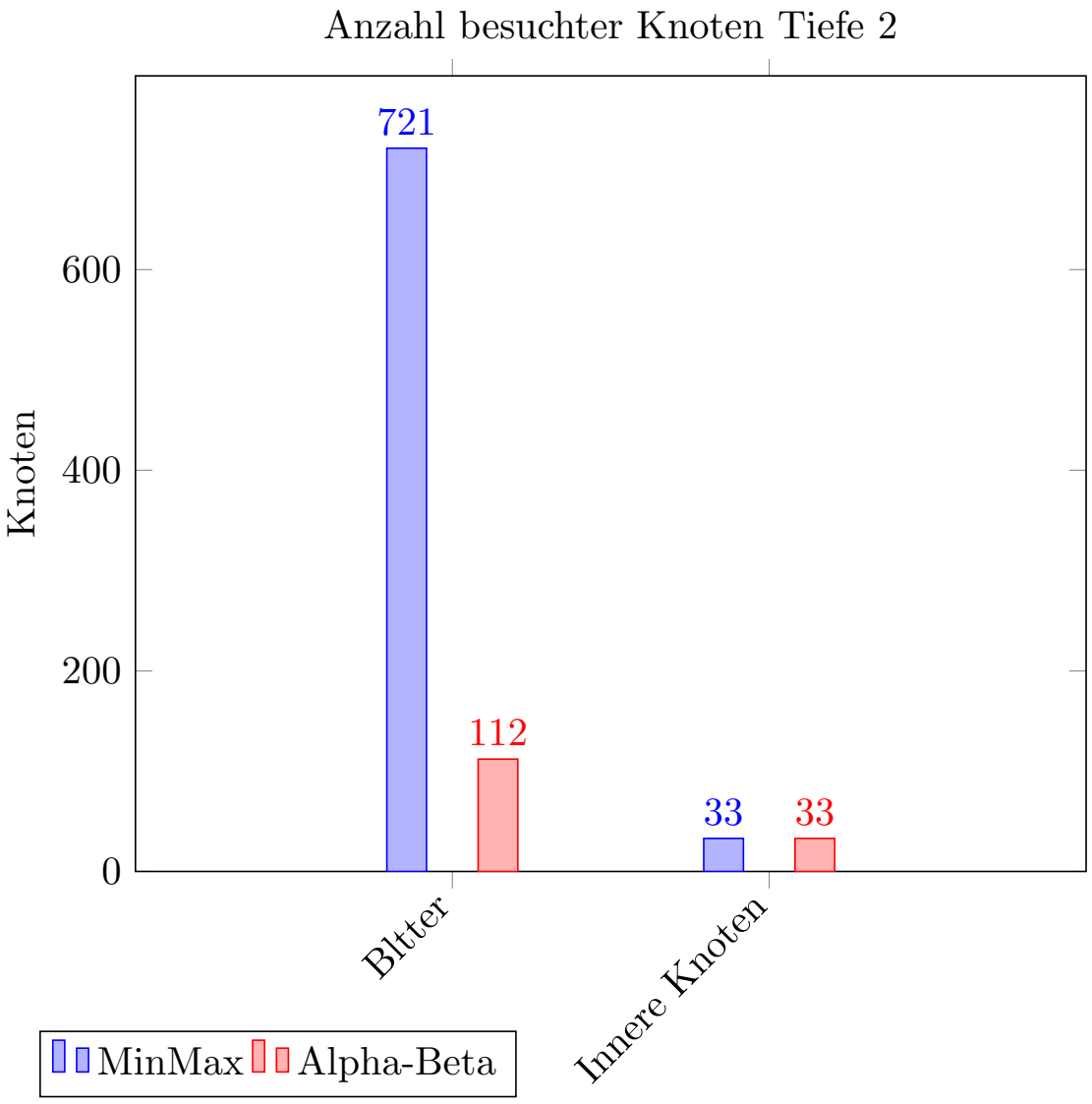

Figure 5: Anzahl besuchter Knoten bei Rechnung bis Tiefe 2

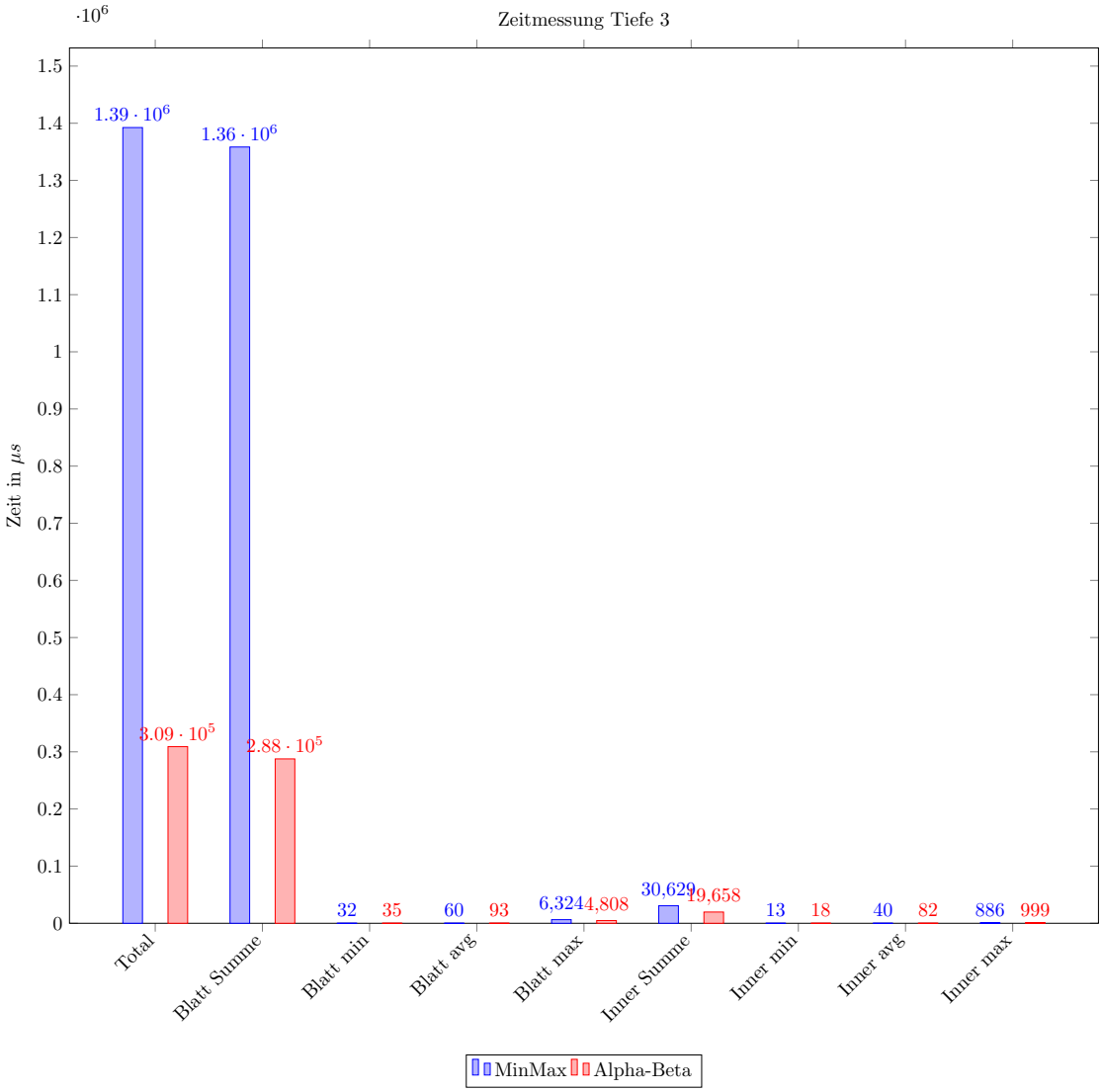

Figure 6: Gemessene Zeiten bei Rechnung bis Tiefe 3

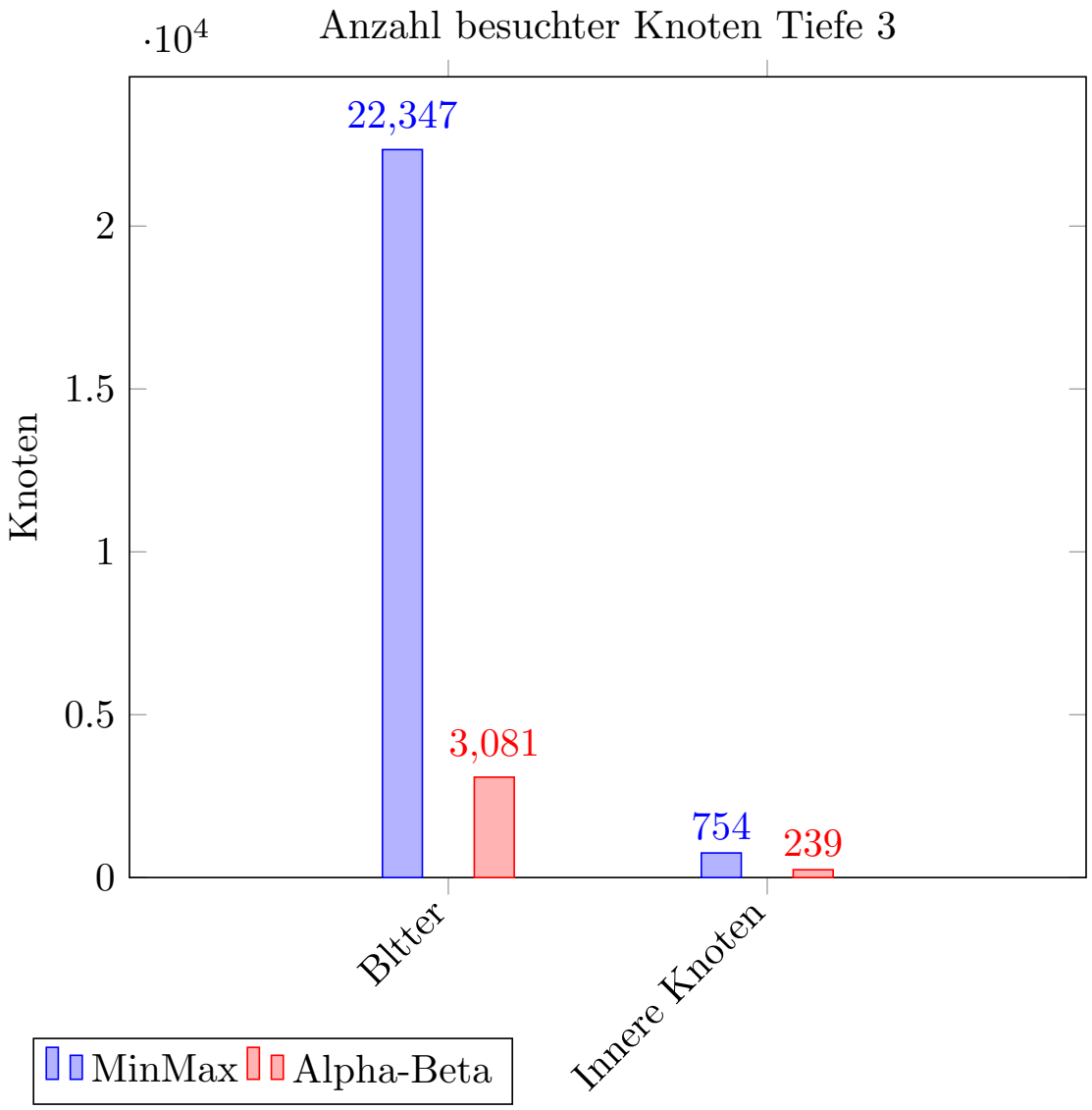

Figure 7: Anzahl besuchter Knoten bei Rechnung bis Tiefe 3