WKM/MIM Lösungsverfahren für Strategiespiele
(WS 2021/22)

-

Visualisierung der Algorithmen: alpha-beta (Web)

Felix Wegener & Chiara Knipprath

Inhalt

Motivation

Im Laufe des Medieninformatik Studiums wird ein Student zwangsläufig mit Themengebieten wie Theoretischer Informatik, Suchbäumen und Algorithmen in Berührung kommen. Diese grundlegenden Themen sind essenziell für das informationstechnische Verständnis und helfen in vielerlei Hinsicht, das logische Denken zu fördern und den Horizont zu erweitern. Dabei kann jener, doch teilweise sehr anspruchsvolle Themenzweig durchaus herausfordernd für viele sein und ist relativ theorielastig. Umso mehr hilft es vielen Menschen Dinge zu visualisieren, um Prozesse besser zu verstehen und leichter nachvollziehen zu können. So sagen viele Physiologen, dass der Mensch circa 70% seiner Sinneseindrücke über das visuelle System wahrnimmt (vgl. Schnerr 2020).

Darüber hinaus sprechen noch viele weitere Faktoren dafür, dass Menschen über ihre visuelle Wahrnehmung Informationen leichter und nachhaltiger verarbeiten können. Der sogenannte *Picture Superiority Effect* sagt aus, „[v]on einem Text, den man nur liest oder hört, sind drei Tage später noch etwa 10 Prozent präsent. Von einem Text, dessen Inhalt auch bildlich rüber gebracht wird, sind es nach drei Tagen über 60 Prozent“ (Schnerr 2020). Diese Zahlen unterstreichen die Bedeutung einer visuellen Präsentation von wichtigen Themengebieten wie der Algorithmik und Suchbäumen.

Eben diese Fakten können dazu genutzt werden, davon zu profitieren und den Studierenden das Verständnis zu erleichtern. So können unteranderem Suchalgorithmen leicht dargestellt werden, um die Theorie plastisch zu visualisieren und den theoretischen Informationsgehalt leichter zu verstehen.

Daher hat es sich dieses Projekt zur Aufgabe gemacht, zunächst eine klare Aufgabenstellung zu definieren und mit dem klaren Ziel der Visualisierung eines Suchalgorithmus, die Funktionsweise und Idee einer künstlichen Intelligenz anhand eines Strategiespiels zu visualisieren.

Aufgabenstellung

Für das vorliegende Projekt im Kontext “Lösungsverfahren für Strategiespiele”, konnte eine klare Aufgabenstellung definiert werden. In der Ausarbeitung des Projektes wurde sich primär mit der Visualisierung von Algorithmen beschäftigt, sodass dieses Thema im Kontext der Strategiespiele eingeordnet werden musste. Nach eingehender Recherche und Planung, wurde sich darauf verständigt die Visualisierung anhand des beliebten Strategiespieles Tic-Tac-Toe vorzunehmen. In der Regel wird für jenes Spiel der Minimax-Algorithmus verwendet, sodass die Wahl auf eben jenen fiel.

Um den Suchalgorithmus performanter zu gestalten und sich von der anderen Gruppe des gleichen Themas abzuheben, wurde sich in der Ausarbeitung speziell auf die Visualisierung der Variante Alpha/Beta des Minimax-Algorithmus konzentriert. Dieser kann als eine Erweiterung des klassischen Algorithmus gesehen werden und unterscheidet sich in gewisser Hinsicht.

Als Basis der Applikation konnte sich auf den Kontext Web verständigt werden, sodass die Applikation über den Browser läuft und dort abrufbar ist.

Die Idee sollte es sein, eine Tic-Tac-Toe-Anwendung im Kontext Web zu realisieren, bei der der Nutzer gegen eine KI spielen kann, die mit dem Alpha/Beta-Algorithmus arbeitet. Parallel soll der Suchbaum der KI visualisiert werden, sodass der Nutzer den Arbeitsprozess der KI verfolgen und besser nachvollziehen kann.

Recherche

Um ein besseres Verständnis über die Mächtigkeit und die Funktionsweise des Alpha/Beta-Algorithmus zu erhalten, musste sich tiefgehend mit dem Thema beschäftigt werden, auch vor dem Hintergrund, dass bisher wenige Berührungspunkte damit bestanden und die Erfahrung auf diesem Themengebiet nicht besonders groß war.

Inspiration

Als Basis der Recherche wurde die Auseinandersetzung mit dem Konstrukt des Alpha-Beta-Algorithmus und die grundlegende Implementation einer künstlichen Intelligenz in TypeScript definiert. Beschäftigt man sich mit der klassischen Implementation von künstlicher Intelligenz, so ist in der Regel die Programmiersprache Python die erste Wahl vieler Programmierer. Da bisher keine Kenntnisse im Bereich Python erworben werden konnten, wurde allerdings auf TypeScript zurückgegriffen, was als eine minimale Hürde identifiziert werden konnte.

Im Web existieren unzählige Arbeiten und Projekte, die sich einem solchen Thema ebenfalls annehmen und in Form von Bäumen darstellen, wie ein Algorithmus arbeitet. Wenige Beispiel gehen jedoch über die einfache Visualisierung eines Suchbaumes als Bilddatei hinaus. Dennoch konnte eine beispielhafte Webanwendung gefunden werden, die die Arbeit eines MiniMax-Algorithmus im Web am Beispiel Tic-Tac-Toe visualisiert und dem Nutzer ermöglicht, parallel gegen die KI zu spielen (Siehe Abbildung 1).

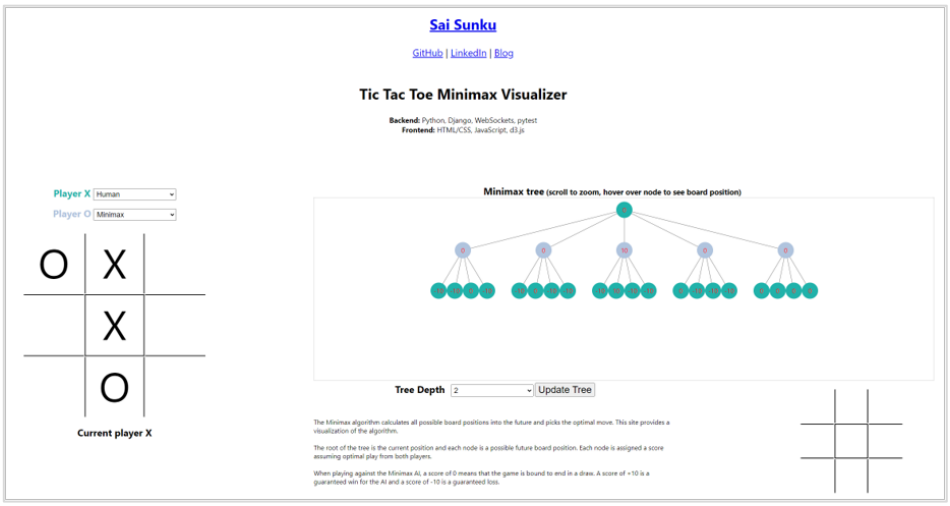

Abbildung 1 Visualisierung eines MiniMax-Suchbaums Quelle: https://minimax-visualizer.herokuapp.com/

Jenes Beispiel wurde bei der Konzeption der Anwendung und ihrer Funktionsweise als konkrete Inspiration genutzt und konnte hilfreiche Erkenntnisse in der Visualisierung und Funktionsweise der Benutzeroberfläche liefern.

Dennoch konnte darüber hinaus noch eine weitere Erkenntnis gewonnen werden. Alle gefunden Beispiele visualisieren einen Suchbaum, der in seinem Grundaufbau jedes Mal nahezu identisch dargestellt ist und sich ein wenigen, minimalen Aspekten unterscheidet. Dies hat den Ansporn geweckt, den konventionelle Weg zu verlassen und eine unkonventionelle, aber mindestens genauso ansprechende und verständliche Lösung zu implementieren.

Visualisierung von Suchbäumen

Mit diesem Gedanken im Hinterkopf wurde bei dem Framework Quasar ein Objekt gefunden, mit dem die Visualisierung des Baumes auf eine andere Art umgesetzt werden konnte. Das Framework wurde dazu verwendet eine ansprechende Oberfläche möglichst unkompliziert und schnell aufzubauen, um den Fokus auf dem Algorithmus sowie auf dem Baum zu haben.
Das Objekt, welches für die Visualisierung des Baumes genutzt wurde, ist der Quasar tree[1]. Dieser ermöglicht einen Aufbau des Suchbaumes auf eine andere Weise.

[1] https://quasar.dev/vue-components/tree

Wie in Abbildung 2 zu sehen, ist der Quasar tree ähnlich einem Baum aufgebaut. Der größte Unterschied ist die Richtung, in der die Elemente angezeigt werden bzw. in die sie wachsen können. In diesem Beispiel besitzt das tree Element eine Überkategorie mit jeweils drei Unterkategorien, welche ebenfalls drei Unterkategorien besitzen.

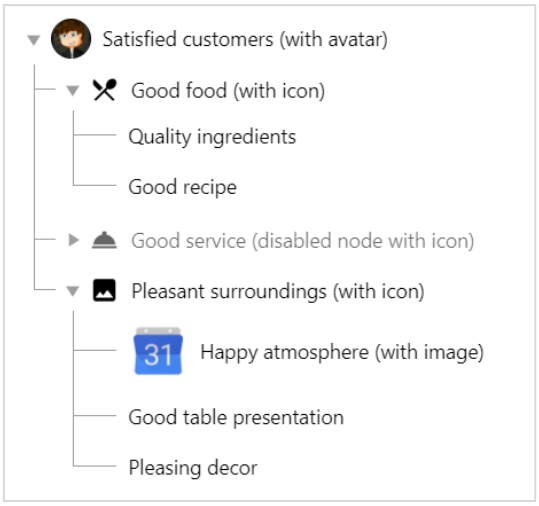

Abbildung 2 Quasar Tree Standardaussehen Quelle: https://quasar.dev/vue-components/tree

An diesem Beispiel wird deutlich, dass solche Bäume schnell sehr groß und unübersichtlich werden können. Solch ein Baum für das Spiel Tic-Tac-Toe würde mit acht Unterpunkten für den ersten Spielzug anfangen und diese hätten dann auch wieder Unterpunkte und so weiter. Insgesamt würde es so mindestens 4 und maximal 8 Ebenen geben. Die Quasar tree Komponente hat den Vorteil, dass sie eine eingebaute „accordion“-Funktion besitzt, welche es ermöglicht Unterkategorien beliebig ein und auszublenden. So kann der Nutzer selbst entscheiden, welchen Pfad er näher untersuchen will. Des Weiteren kann auch durch eine Eigenschaft dem tree-Objekt mitgeteilt werden, ob ein bestimmter Pfad zu Beginn ausgeklappt werden soll. Durch diese Eigenschaft namens „:default-expand-all = true“ werden zu Beginn alle Zweige des trees ausgeklappt. Als weitere Möglichkeit kann die Quasar tree Komponente ein Array, mit den Keys der Abzweigungen, die ausgeklappt werden sollen, als Eigenschaft mitgeben.

Eine weitere nützliche Eigenschaft des Quasar tree-Objektes ist die einfache Nutzung von Icons. Dadurch ist es möglich, ohne großen Aufwand verschiedene Icons am Anfang der Abzweigungen einzufügen, um eine bessere Gesamtübersicht zu gewährleisten. In diesem Projekt wurde sich auf vier verschiedene Icons geeinigt. Einen Stern, um den Pfad mit einem potenziellen Sieg für den Algorithmus hervorzuheben. Ein Kreuz, um ein mögliches Verlieren auf dem Pfad darzustellen. Ein neutrales Icon, um ein mögliches Unentschieden zu verdeutlichen sowie ein Stopp-Icon, um einen übergangenen Pfad zu makieren.

Vorgehensweise

Implementation der KI

Auf Basis der gesammelten Rechercheergebnisse wurde nach Erstellung des Projektes, mit der Implementation einer Tic-Tac-Toe Anwendung begonnen und mit der Konzeption einer künstlichen Intelligenz angefangen. Diese stellt die Grundlage dar, auf deren Basis die Visualisierung der Hintergrundprozesse geschehen kann und wurde somit sehr zeitnah in Arbeit genommen.

Zunächst wurde eine simple Tic-Tac-Toe Oberfläche umgesetzt, auf der zwei Nutzer gegeneinander spielen können, ohne dass die Möglichkeit bestünde, gegen eine KI anzutreten. Dies war der

Grundstein, da bereits zu diesem Zeitpunkt die Spielregeln, die Gewinnüberprüfung und das Spielbrett angelegt werden konnten. Nachdem dem das klassische Spiel implementiert wurde, sollten weitere Funktionen eingefügt werden, welche dem Nutzer als hilfreiche Features zur Verfügung stehen. Zu diesen gehört die „Undo-Funktion“ sowie die individuelle Einstellung der Zeichen („X“ oder „O“). Das Revidieren von gespielten Zügen ist ein wichtiges Feature und steht dem Spieler auch im Spiel gegen die KI zur Verfügung. Denn dadurch kann der Spieler seine Züge beliebig verändern und sieht danach die jeweilige „Antwort“ der KI. Danach begann die eigentliche Arbeit des Projektes.

Es wurde eine KI in Form eines Alpha-Beta-Algorithmus implementiert und die Möglichkeit integriert, die Suchtiefe über ein späteres Bedienelement durch die Eingabe des Nutzers festzulegen.

```
if (isMaximizing) {
  let bestScore = -1000;
  for (let i = 0; i < 10; i++) {
    for (let j = 0; j < 5; j++) {
      // Is the spot available?
      if (board[i][j] == '' && typeof depth === 'number') {
        board[i][j] = ai;

        // eslint-disable-next-line @typescript-eslint/no-unsafe-assignment
        let score = minimax(board, depth - 1, false, alpha, beta);

        board[i][j] = '';

        bestScore = Math.max(score, bestScore);
        // check for alpha
        alpha = Math.max(alpha, bestScore);
        // Check for alpha beta pruning
        if (beta <= alpha) {
          break;
        }
      }
    }
  }
  return bestScore;
```

Abbildung 3 Alpha-Beta-Algorithmus Quelle: Eigene Darstellung

Abbildung 3 zeigt dabei einen Ausschnitt der Code-Basis, die sich im grundlegenden Aufbau von der Theorie kaum unterscheidet.

In der Funktion makeComputerMove() die, wie der Name bereits impliziert, einen Zug der KI simuliert, wird ein anhand des Algorithmus eine Wertigkeit jedes Zuges errechnet und der Zug mit dem Bestem gespeichert und als finaler Zug ausgewählt.
Ein Score ist die Punktzahl, die ein Zug bekommt, aufgrund seiner Wichtigkeit im Erreichen des Spielendes. Die Funktion arbeitet dabei rekursiv und ruft sich innerhalb ihres Codes selbst auf. Dies ist notwendig, um in der Suchtiefe des Baumes voranzuschreiten und die Wertigkeit von Zügen beider Spieler zu ermitteln. Ist die finale Suchtiefe erreicht, so steht der finale Zug anhand seiner Wertigkeit fest. Erfasst die KI bei einem der Zweige, dass dieser auch mit fortschreitender Suchtiefe, nicht die Wertigkeit eines bereits erfassten Zweiges erreichen kann, so stellt dies eine Abbruchbedingung dar und dieser Zweig wird nicht weiter durchlaufen. Dies spart Ressourcen und erhöht die Zuggeschwindigkeit der KI.

Nachdem die Implementation fertiggestellt und überprüft wurde, wurde die Bedienoberfläche entsprechend um die neuen Funktionalitäten erweitert.

Suchbaum aufbauen

Die Scores der verschiedenen Züge sind die Hauptinformationen, welche in den Suchbaum eingebaut werden mussten. Um dies umzusetzen wurden die einzelnen Scores in einem Array gespeichert.

Da in der Applikation mit TypeScript gearbeitet wurde, war es entscheidend ein eigenständiges Interface zu implementieren, um eine genaue Definition der Objekte zu haben, die in das Array eingefügt werden sollen. Die Herausforderung beim tree-Objekt war, auf die speziellen, von Quasar integrierten Eigenschaften einzugehen, da diese essenziell für die Funktionalität sind. In Abbildung 4 ist das finale tree-Objekt zu sehen, welches als zulässiges Objekt gilt, mit dem die tree-Komponente arbeiten kann.

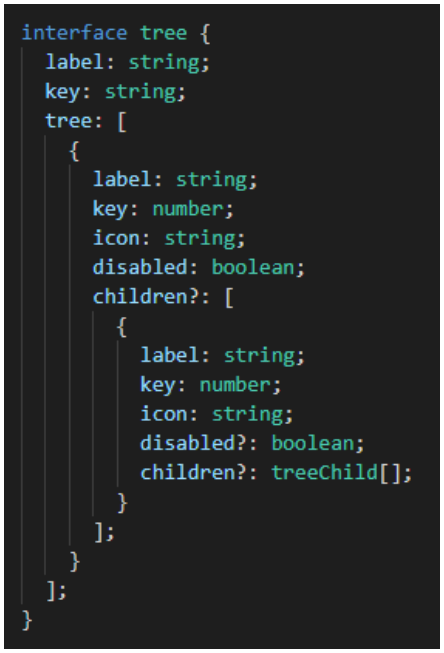

Abbildung 4 tree- Interface Quelle: Eigene Darstellung

Das erste *label* und der *key*, dienen lediglich zur besseren Sortierung der einzelnen Quasar trees. Das tree-Objekt, welches danach zu sehen ist, ist mit den als Baum angezeigten Daten gefüllt. Die darauffolgenden „*children*"-Objekte sind jeweils optional und bilden die einzelnen Unterpunkte des Baumes ab. Um einen in der Theorie endlos großen Baum zu erzeugen, muss mit einem weiteren Objekt gearbeitet werden, dem „*treeChild*[]". Der Aufbau dessen ist in Abbildung 5 zu sehen. Solch ein Objekt zu haben ermöglicht es, unendlich viele Instanzen davon in das tree-Objekt zu setzen.

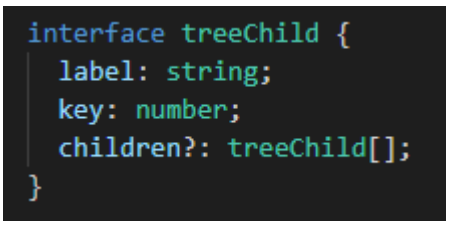

Abbildung 5 treeChild- Interface Quelle: Eigene Darstellung

Diese Vorarbeit musste geleistet werden, um eine genaue Definition des Objektes zu haben, mit der der finale Array befüllt wurde, welcher später die einzelnen Bäume enthielt. Die nächste Herausforderung war die einzelnen tree-Objekte richtig zu befüllen, um sie dann in den Array abspeichern zu können.
Das Problem war, dass die Scores, welche aus der Algorithmus Funktion kommen, nicht exakt zuzuordnen sind. Das heißt, es ist nicht möglich das tree-Objekt schon innerhalb des Algorithmus zu füllen, da dieser nie genau weiß in welchem Zweig er sich gerade befindet. Es musste eine andere Lösung gefunden werden. Die einzelnen Scores, welche die Funktion ausgibt, werden für jeden Zug in der makeComputerMove() Funktion in einem temporären Array gespeichert. Dieser wird dann am Ende des Zuges durchgegangen und das tree-Objekt richtig befüllt. Das fertige tree-Objekt dieses

Zuges wird dann in den finalen Array geladen und für den nächsten Zug wiederholt. Das fertige Array kann dann von der Quasar tree Komponente ausgewertet werden und wird korrekt angezeigt. Die fertige Darstellung ist in Abbildung 6 zu sehen.

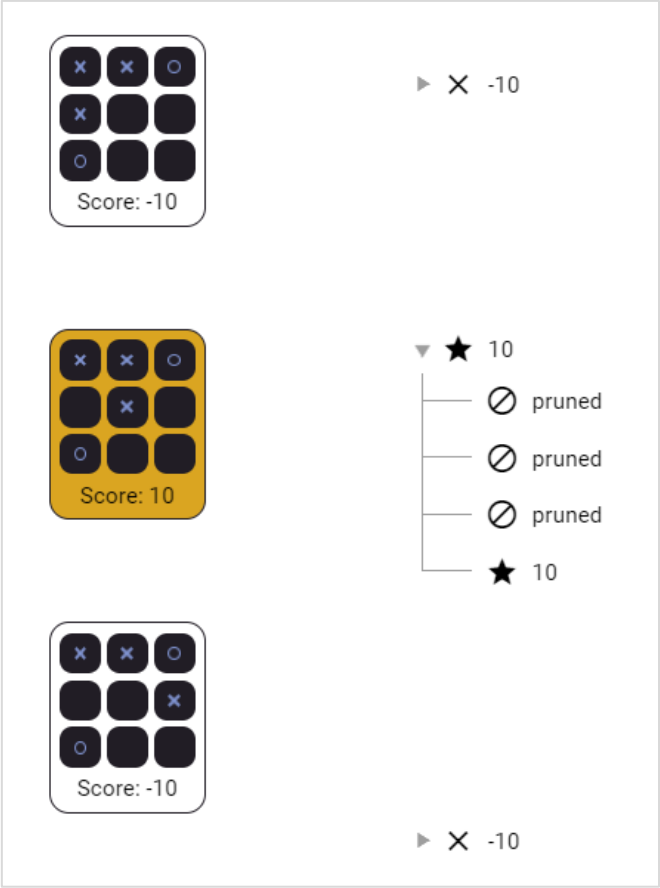

Abbildung 6 Ausschnitt Browserdarstellung Quelle: Eigene Darstellung

Ebenfalls in Abbildung 6 zu sehen, sind die verschiedenen Boards auf der linken Seite des Bildes. Nachdem der Suchbaum fertiggestellt wurde, wurde beschlossen diese ebenfalls einzufügen, um eine optisch ansprechendere Darstellung des final ausgewählten Zuges zu bieten. So sieht der Nutzer auf den ersten Blick den Baum, für den sich der Algorithmus letztendlich entschieden hat und kann diesen näher untersuchen.

Finale Anpassungen

Nach Fertigstellung des Suchbaumes, war es an der Zeit, die Oberfläche optisch anzupassen und dem Nutzer ein positives visuelles Erlebnis zu liefern. Dazu wurden unteranderem kleine Animationen eingefügt, welche den Aufbau des Baumes besser darstellen sollen. Davor ist jeder Zweig der KI bei jedem Zug gleichschnell erschienen, teilweise sogar gleichzeitig mit dem Zug des Menschen. Um darzustellen, dass die KI „nachdenkt“ wurde einerseits zwischen den Zügen eine Verzögerung eingebaut und andererseits eine entsprechende Animation eingefügt. Die Animation wurde jedoch im Laufe des Projektes etwas abgeschwächt, da sie als störend im Spielfluss empfunden wurde.

Des Weiteren wurden optische Kleinigkeiten, wie die vorher festgelegten Icons in den Suchbaum eingebaut sowie die richtige Skalierung eingestellt.

Ergebnis

Das Ergebnis dieses Projektes ist ein modernes Tic-Tac-Toe-Browserspiel inklusive funktionierender KI. Die Zielsetzungen konnten umgesetzt werden und aufkommende Herausforderungen wurden bewältigt. Durch das Framework Vue, in Kombination mit Quasar, wurde eine nutzerfreundliche und leicht verständliche Oberfläche geschaffen. Die unkonventionelle Umsetzung des Suchbaumes, abseits von der bekannten Baumstruktur, sorgt für ein neues visuelles Erlebnis. Ebenso ist es dadurch möglich eine viel größere Suchtiefe abzubilden, da Verzweigungen einfach eingeklappt werden können.

Rein aus Interesse und dem Spaß an der Implementierung der Visualisierung, wurde beschlossen eine weitere Art des Tic-Tac-Toe-Spiels umzusetzen. Das Ergebnis dieses Versuches ist ein Spezial-Tic-Tac-Toe mit einem 5x10 Feld, bei der mit einer 5er Reihe (diagonal, vertikal oder horizontal) gewonnen wird. Dadurch, dass das originale Tic-Tac-Toe schon implementiert wurde, konnte der Großteil dieses Codes übernommen werden. Auch der Algorithmus konnte adaptiv in diese spezielle Variante integriert werden und musste sich nur minimalen Änderungen unterziehen. Lediglich bei der Visualisierung wurde sich darauf geeinigt, diese einfacher zu halten und lediglich die erste Ebene des Baumes anzuzeigen. Ein Tic-Tac-Toe-Brett von solch einer Größe hat zu viele mögliche Spielzüge, die in den Baum mitaufgenommen werden müssten.

Letztendlich konnte jedoch auch dieses Zusatzprojekt erfolgreich abgeschlossen werden und der Nutzer hat die Möglichkeit zwischen den verschiedenen Spielmodi zu wechseln.

Erkenntnisgewinn

Der größte Erkenntnisgewinn aus diesem Projekt ist eindeutig das Verständnis der Algorithmen. Vorher war aus dem Studium zwar bekannt, dass es diese Algorithmen gibt und auch das theoretisch Arbeiten damit wurde nähergebracht, jedoch fehlte die praxisnahe Anwendung. Dadurch, dass sich in diesem Projekt auch mit der Implementierung und Visualisierung auseinandergesetzt werden musste, musste viel tiefer in die Materie eingetaucht werden. Weniger die Implementierung, aber gerade die Visualisierung war ein zeitaufwändiges und recht komplexes Unterfangen. Dies hat Interesse geweckt genauer zu verstehen, wie es eventuell in anderen Algorithmen oder KI's funktionieren kann und ob dort andere Wege neue Möglichkeiten der Visualisierung eröffnen. Aufschlussreich war es ebenfalls zu erfahren, dass der implementierte Algorithmus, bis auf wenige, minimale Änderungen, nahezu identisch für beide Versionen des Spiels implementiert werden konnte.
Weniger gut hat die Arbeit mit TypeScript für solch ein spezielles Szenario funktioniert. Einerseits ist es praktisch immer eine strikte Typisierung aller Objekte und Variablen zu haben, da Fehler viel schneller gefunden werden können und somit präventiv Fehler beseitigt werden. Andererseits hat dies erschwert, die Quasar tree Komponente mit den richtigen Daten zu befüllen. So musste Interface implementiert werden, welches nur umständlich um mehrere Objekte erweitert werden konnte. Auch das Einspeisen der richtigen Werte in das tree-Objekt war dadurch wesentlich komplizierter.

Ausblick

In Zukunft wäre eine Implementation des Algorithmus in andere Spiele erdenklich. Tic-Tac-Toe hatte für dieses Projekt den Vorteil, dass die Anzahl an möglichen Zügen recht limitiert ist. Somit konnte sich primär um den Algorithmus und dessen Visualisierung gekümmert werden. Wenngleich der Baum bei anderen Spielen sehr viel größer werden würde, wäre durch die Vorarbeit dieses Projektes eine Möglichkeit gegeben, ihn auf eine praktische Art und Weise darzustellen.

Zudem bietet das Projekt die Möglichkeit die Website mit weiteren Features auszustatten, welche dem Nutzer mehr Mittel bieten seine Erfahrung mit dem Algorithmus zu optimieren. So können unteranderem mehr Einstellungsmöglichkeiten für die Suchtiefe, das individuelle An- und Ausschalten des Alpha/Beta-Prunings oder die Visualisierung des gesamten Suchbaumes beim Special-Tic-Tac-Toe, die Nutzungserfahrung weiter verbessern.

Literaturnachweise

- Schnerr, Bettina (2020): Visualisierung: Wie Bilder beim Lernen helfen | Bildungsmagazin - eduwo.ch, Bildungsmagazin von eduwo.ch, [online] https://eduwo.ch/bildungsmagazin/visualisierung-wie-bilder-beim-lernen-helfen/ [abgerufen am 19.03.2022].