OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Entwicklung einer künstlichen Intelligenz für Brettspiele und deren Anbindung an eine Touch-Hardware über mobiele Endgeräte

An der Fakultät für Informatik und Mathematik der
Ostbayerischen Technischen Hochschule Regensburg
im Studiengang
Techniche Informatik

eingereichte

Bachelorarbeit

zur Erlangung des akademischen Grades des
Bachelor of Science (B.Sc.)

Vorgelegt von: Korbinian Federholzner
Matrikelnummer: 3114621

Erstgutachter: Prof. Dr. Carsten Kern
Zweitgutachter: Prof. Dr. Daniel Jobst

Abgabedatum: 31.08.2020

Erklärung zur Bachelorarbeit

1. Mir ist bekannt, dass dieses Exemplar der Abschlussarbeit als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.
2. Ich erkläre hiermit, dass ich diese Abschlussarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Regensburg, den 19. September 2020

_____

Korbinian Federholzner

Zusammenfassung

In der folgenden Arbeit wird ...

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

I Abkürzungsverzeichnis

1 Einleitung

Das Thema dieser Arbeit ist die Implementierung einer künstlichen Intelligenz für diverse Brettspiele. Dabei soll die resultierende Software auf einem von einem Raspberry Pi gesteuerten Touch-Bildschirm laufen, durch welchen ein Benutzer die künstliche Intelligenz herausfordern kann. Außerdem gibt es die Möglichkeit, sich mit der Hardware über ein mobiles Endgerät wie einem Smartphone zu verbinden, um die KI oder den Spieler, der den Touch-Bildschirm bedient, herauszufodern.

1.1 Motivation

Das Feld der künstlichen Intelligenz ist momentan eines der sich am schnellsten entwickelnden Felder der Informatik. Dabei spielt die Spieltheorie schon seit Anfang eine große Rolle. So bieten klassische Brettspiele wie z. B. Schach, Dame oder Mühle nicht nur eine klar definierte Abstraktion von Problemen der realen Welt, sondern sie können auch von dem Großteil der Bevölkerung verstanden und gespielt werden. Das Meistern einer dieser Brettspiele wird auch oft mit hohem Grad an Intelligenz gleichgesetzt. Viele Algorithmen, die in der Spieltheorie entwickelt wurden, haben sich auch erfolgreich auf andere Felder der Informatik übertragen lassen. Ebenso hat sich die Art, wie Brettspiele gespielt werden, durch das Verwenden von künstlicher Intelligenz auch verändert, da die KI Züge in Betracht zieht, die auf den ersten Blick recht ungewöhnlich und nachteilhaft aussehen, sich aber als extrem stark herausstellen. Diese Arbeit versucht eine künstliche Intelligenz für Brettspiele, wie Dame, zu implementieren.

1.2 Aufgabenstellung

Im Rahmen der Bachelorarbeit soll eine künstliche Intelligenz entwickelt werden, welche Brettspiele, wie Dame, spielen kann. Gegeben ist eine Software, bei welcher man in der Lage ist, das Spiel ReversiXT (Reversi Extreme) gegen eine KI, sowie sich selbst zu spielen. Diese Software soll um einen Game Server, die KI und das neue Spiel in der GUI, erweitert werden. Außerdem soll ein Benutzer in der Lage sein, sich mit seinem Smarthphone mit der Hardware zu verbinden, um neue, sowie die alte KI herausfordern zu können.

1.3 Struktur dieser Arbeit

Die Arbeit ist folgendermaßen aufgebaut:
Kapitel 2 befasst sich mit den Grundlagen zu Dame, sowie den verwendeten künstliche Intelligenz Algorithmen. Dabei werden zu Dame auch die Grundregeln der verwendeten Variante erklärt. Bei den KI Algorithmen handelt es sich um die Algorithmen, die in der Arbeit verwendet und miteinander verglichen werden.
In Kapitel 3 werden die Anforderungen, welche gefordert sind, vorgestellt.

2 Grundlagen

Dieses Kapitel gibt einen Überblick über die theoretischen Grundlagen, die für das Verständnis dieser Arbeit notwendig sind. Zunächst werden die Grundregeln des behandelten Brettspieles vorgestellt. Darauf folgt eine Erklärung der künstlichen Intelligenz Algorithmen, welche für folgenden Kapitel von großer Relevanz sind. [Rus12]

2.1 Dame

Dame ist eines der ältesten Brettspiele, wobei erste Varianten 3000 v. Chr. im irakischen Ur entdeckt wurden. In der heutigen Zeit werden verschiedene Varianten desselben Spieles weltweit gespielt. So wird in vielen englischsprachigen Ländern eine andere Version gespielt als im Rest der Welt, was auch als Internationale Dame bekannt ist. Unterschiede sind z. B. bei diesen Varianten, dass bei internationaler Dame, Damen beliebig viele Felder in alle Richtungen springen dürfen, in anderen Varianten jedoch nur 1 Feld. Da die Regeln der internationalen Dame weiter verbreitet sind und auch im Vereinssport praktiziert werden, wird sich diese Arbeit im folgenden auf diese Regeln fokussieren. [Dra]

2.1.1 Internationale Dame

In dieser Variante des Spieles wird auf einem 10x10 Brett mit Schachbrett-Muster gespielt. Siehe Abbildung 1. Die Spielsteine sind scheibenförmig und in zwei Farben vorhanden, meist schwarz und weiß und dürfen nur auf den dunklen Feldern des Schachbrettes bewegt werden. Es gibt zwei Arten von Spielsteinen. Normale Spielsteine, welche nur in Richtung des Gegners bewegt werden, aber Rückwerts schlagen dürfen, und Damen, welche in alle Richtungen beliebig viele Felder fahren und schlagen dürfen. Allgemein herrscht Schlagzwang, was bedeutet, dass falls ein Spieler die Möglichkeit hat zu schlagen, er auch schlagen muss. Ein normaler Spielstein wird zur Dame, falls er in die hinterste Reihe des Gegners kommt. Ziel des Spieles ist es, entweder alle Steine des Gegners zu schlagen, oder den Gegner in eine Situation zu zwingen, in der er keine Züge mehr machen kann. [Int]

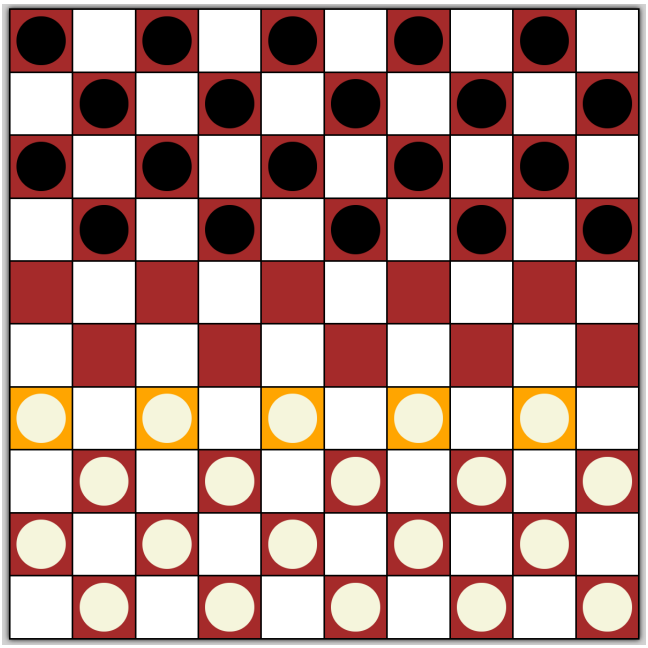

Abbildung 1: Das 10x10 Spielfeld aus der Anwendung

2.2 Minimax

Der Minimax Algorithmus wird verwendet, um einen optimalen Spielzug in Spielen mit perfekter Information, bei zwei Spielern zu finden. Dazu wird eine Baumstruktur verwendet, welche den Zustand des Spielbrettes als Knoten hat, siehe Abbildung 2. Alle Züge, die von einer Stellung aus möglich sind, werden in den Kindknoten des jeweiligen Knotens gespeichert. Der Wurzelknoten beschreibt den momentanen Zustand des Spieles bei dem der Algorithmus aufgerufen wird. Die Blattknoten am Ende des Baumes entsprechen entweder einer Stellung in der das Spiel beendet wurde, oder der Stellung bei einer Tiefe, bei der der Algorithmus aufgehört hat zu suchen. Die Angabe einer Tiefe ist nötig, da Spiele wie Schach oder Dame einen extrem großen Suchbaum zur Folge hätten und das Suchen eines Endzustandes in diesen sehr viel Zeit beansprucht. [MSC82]

Zum Beispiel kann die Abbildung 2 durch den Baum aus der Grafik 3, mit einer Suchtiefe von vier, dargestellt werden. Dabei wird jeder Terminal-Knoten, ein Knoten bei dem das Spiel vorbei ist oder die Endtiefe erreicht wurde (im Beispiel Grün makiert), mit einer Bewertungsfunktion bewertet:

- Normale Figur: +1 für Weiß und -1 für Schwarz
- Dame: +3 für Weiß und -3 für Schwarz
- Spielende: ∞ für Weiß und $-\infty$ für Schwarz

Ein Dreieck mit der lange Seite nach unten, steht für eine Maximierung der Kindknotenwerte, das andere Dreieck für eine Minimierung. Die Bewertungen in den Endzuständen werden nach oben durchgereicht und je nachdem, ob der Elternknoten ein Maximierer oder ein Minimierer

ist, bekommt er einen neuen Wert zugewiesen. Im Beispiel kann man sehen, dass egal welche Züge gewählt werden es immer zu einem Materialverlust von Weiß kommt. Würde Weiß die zweite Option wählen, so ist das Spiel nach dem Nächsten Zug von Schwarz schon entschieden und Weiß verliert.

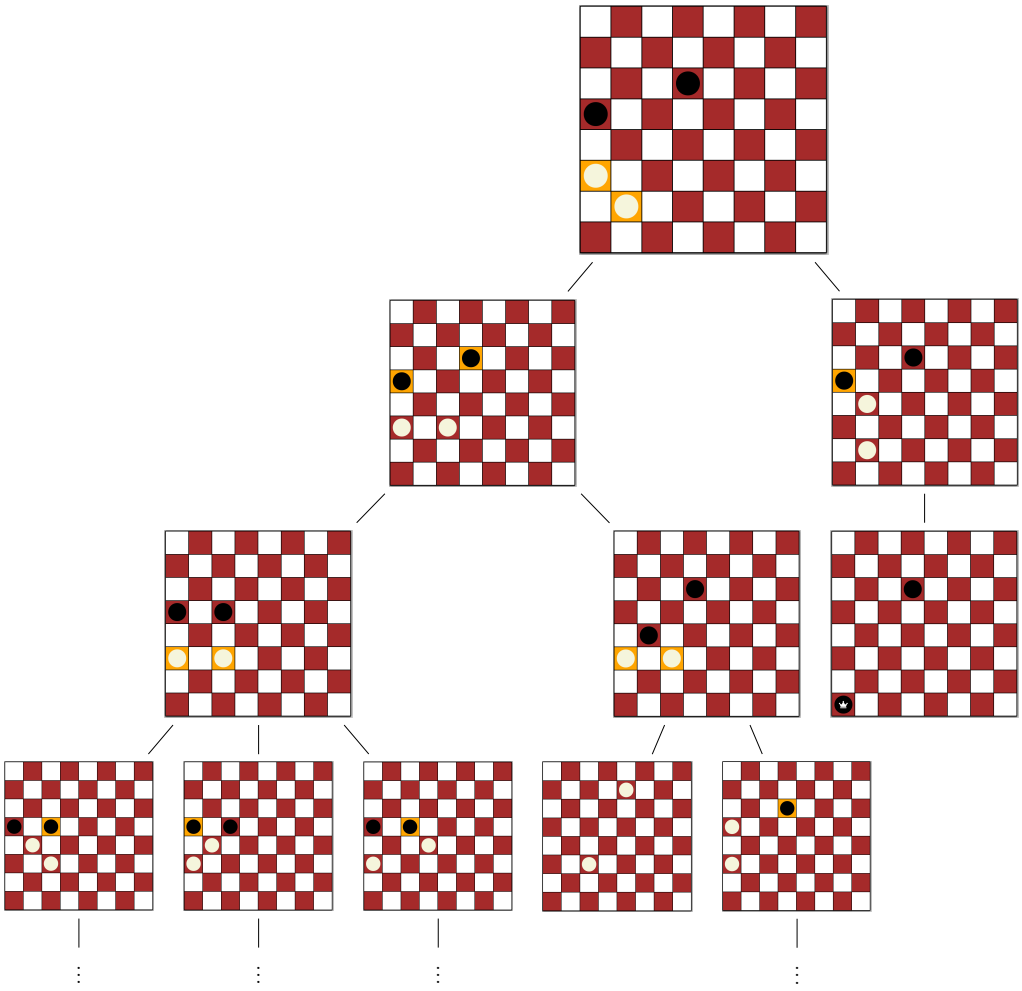

Abbildung 2: Minimax-Baum mit Spielfeldzustand als Knoten

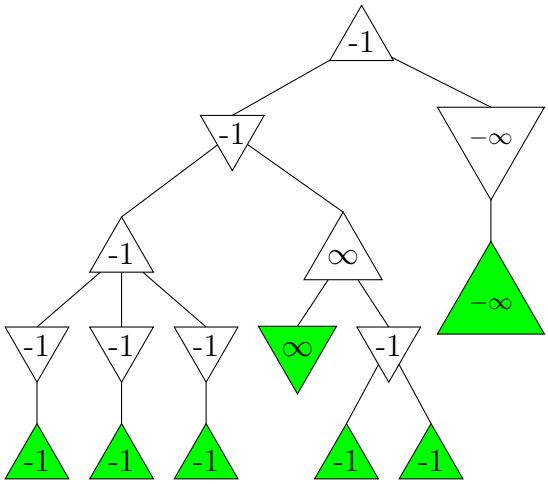

Abbildung 3: Minimax-Baum mit Bewertung der Stellungen

2.3 Iterative Deepening

In komplexeren Spielen wie Go, Schach oder Dame ist es wegen des Rechenaufwands sehr schwer, den kompletten Baum von Minimax bis zu den Endzuständen aufzubauen. Deswegen wird in diesen Spielen der Baum nur bis zu einer gewissen Tiefe aufgebaut. Da es aber bei einer gleichen Tiefe für verschiedene Stellungen zu unterschiedlichen Dauern der Suche kommen kann, ist es problematisch eine fixe Suchtiefe anzugeben, vor allem wenn mit Zeitlimits gearbeitet wird. Iterative Deepening hilft hierbei, der Ablauf des Algorithmus ist wie folgt: Zuerst führe Minimax für eine Tiefe von eins aus. Danach, verwerfe alle generierten Knoten des Baumes und starte erneut von Anfang, aber dieses Mal bis zu einer Tiefe von zwei. Dieses verwerfen und neu starteten wird so oft wiederholt bis ein Zeitlimit erreicht wird, siehe Abbildung 4. Der letzte aufgebaute Baum, bevor neugestartet wird, wird zwischengespeichert und falls Minimax bis zum Ablauf des Zeitlimits nicht fertig ist, wird die momentane Berechnung verworfen und der letzte gespeicherte Baum verwendet. Ein Nachteil dieser Methode ist, dass der Rechenaufwand der ersten Tiefen verschwendet wird, da diese Ergebnisse verworfen werden. Jedoch beeinflusst diese verschwendete Rechenzeit nicht die asymptotische Laufzeit des Algorithmus, da die meiste Arbeit in der untersten Tiefe der Suche gebraucht wird. [Kor85].

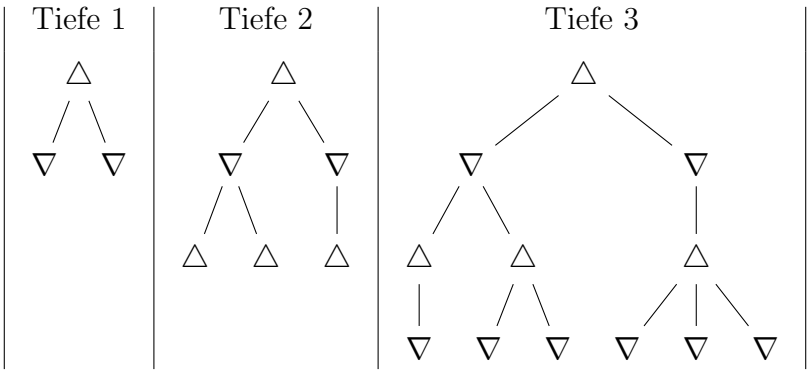

Abbildung 4: Ablauf des Iterativen Deepenings

2.4 Alpha-Beta Pruning

Das Alpha-Beta Pruning ist eine Optimierung zum Minimax Algorithmus. Die Idee des Algorithmus ist, dass manche Zweige des Suchbaums nicht untersucht werden müssen, da für den anderen Spieler diese Züge nicht in Frage kommen. Hierbei ist α der Wert für den Spieler, für den die niedrigen Werte besser sind und β für den anderen Spieler. Für jeden Knoten, je nachdem, ob er ein maximierender oder ein minimierender Knoten ist, wird überprüft, ob ein Kind-Knoten, welcher einen neuen Wert erhalten hat, nicht mehr vom Knoten beachtet werden muss. Der Vorteil des Alpha-Beta Prunings zu Minimax ist, dass der verbrauchte Speicher weniger wird, da vom Baum Zweige nicht beachtet werden müssen. Was wiederum zur Folge hat das die Ausführungszeit des Algorithmus schneller ist und gleichzeitig auch dasselbe Ergbnis wie Minimax zur Folge hat. [SPS]

Wenn man Abbildung 2 und 3 als Beispiel nimmt und den Alpha-Beta Pruning Algorithmus anwendet, so kann der Zweig mit dem Wert ∞ ignoriert werden, siehe Abbildung 5. Der gelbe Knoten bekommt eine -1 durch seinen linken Zweig vorübergehend zugewiesen. Da der Knoten des Rechten Zweiges ein Maximierer ist, also immer den Wert des Kindknotens mit dem höchsten Wert nimmt, und dieser bereits einen Knoten mit dem Wert ∞ gefunden hat, wird sein Wert definitiv mindestens ∞ sein. Der restliche rechte Zweig des gelben Knotens kann nun ignoriert werden, da der linke Zweig mit -1 definitiv kleiner sein wird.

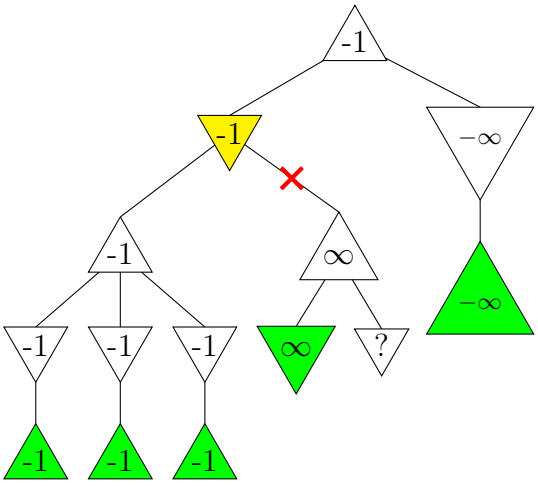

Abbildung 5: Gewinn durch Alpha-Beta Pruning

2.5 Zugsortierung

Zugsortierung ist eine Erweiterung zur Alpha-Beta Suche. Da Alpha-Beta Pruning abhängig von der Reihenfolge, in der die Zustände untersucht werden, ist es sinnvoll die Nachfolger zu wählen, welche die besten Werte erbringen. Den besten Nachfolger findet man, in dem man eine weitere Bewertungsfunktion einbaut, die nicht so genau wie die Bewertungsfunktion an den Terminalknoten sein muss. Wenn ein Knoten also alle möglichen Nachfolgezüge als Kinder bekommt, werden auf diese die vereinfachte Bewertungsfunktion angewandt und je nach Ergebnis der Funktion werden die Nachfolger sortiert. Dadurch, dass der beste Zug nun sehr weit am Anfang steht, ist es sehr wahrscheinlich, dass die anderen Züge durch Alpha-Beta Pruning ignoriert werden. [Rus12]

Auf der Linken Seite der Abbildung 6 kann man erkennen, dass Alpha-Beta Pruning keinen Effekt auf die beiden gelben Knoten hätte. Ändert man jedoch die Reihenfolge der Knoten, so kann der Knoten mit dem Wert 42 ignoriert werden. Bei der Bewertungsfunktion könnte man die bedrohte Figuren als Faktor haben, um auf dieses Ergbnis zu kommen.

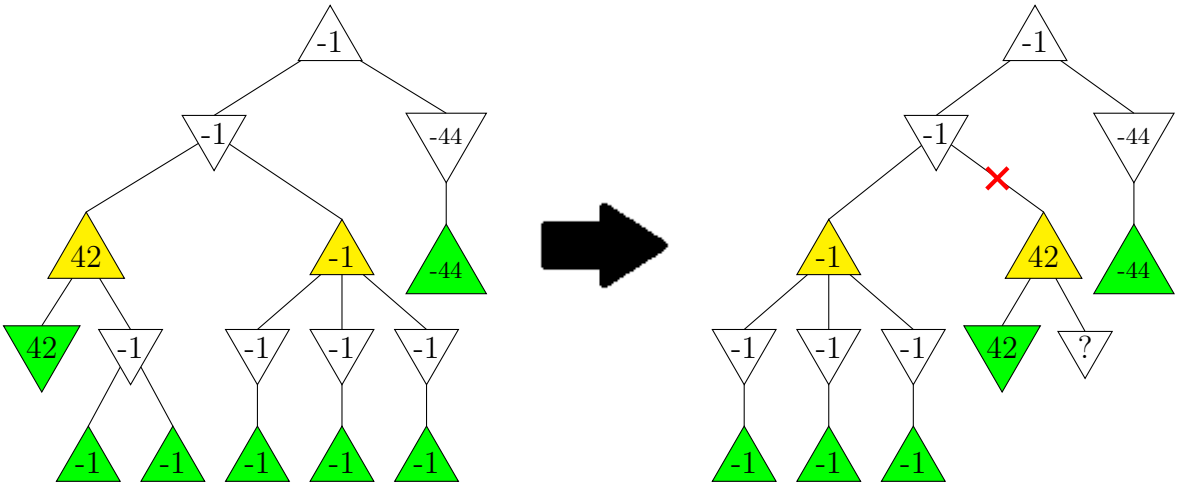

Abbildung 6: Beispiel einer Zugsortierung

2.6 Monte Carlo Tree Search (MCTS)

Der Monte Carlo Tree Search Algorithmus, ist ein heuristischer Algorithmus, bei welchem von einem Zustand eines Spieles zufällig endlich viele Simulationen durchgeführt werden. Die Simulation endet, wenn ein Ergebnis des simulierten Spieles feststeht. Das Wiederholen der Simulationen aus verschiedenen Knoten hat zur Folge, dass das Ergebnis immer genauer wird. Am Ende wird der Knoten gewählt, bei dem die Simulationen die besten Ergebnisse für den momentanen Spieler gezeigt haben. Ein Vorteil des MCTS-Algorithmus gegenüber Minimax ist, dass erst am Ende eines Durchlaufs eine Bewertungsfunktion benötigt wird. Allgemin besteht der Algorithmus aus vier Schritten:

- *Selektion*: Versucht wird, einen Zustand zu finden der noch erweiterbar ist, also einen Zustand zu finden, der kein Endzustand ist und noch nicht besuchte Züge hat.
- *Expansion*: Der Spielbaum wird zufällig um einen noch nicht besuchten Zug erweitert.
- *Simulation*: Von dem gewählten Knoten aus wird nun ein Spiel zufällig bis zum Ende simuliert.
- *Backpropagation*: Das Ergebnis der Simulation wird den vorhergehenden Knoten mitgeteilt und diese werden mit diesem aktualisiert.

Da man im Normalfall nicht beliebig viel Zeit hat, alle Möglichkeiten zu simulieren, versucht man die limitierte Zeit so gut wie möglich zu nutzen, um die richten Knoten zum Expandieren zu wählen. Dazu wird der *upper confidence bound for trees* (UCT) verwendet. Die UCT Formel lautet:

$$w + c\sqrt{\frac{\log N}{n}} \tag{1}$$

Wobei w die prozentuale Anzahl an Gewinnen des Knotens, N die Anzahl der gesamten Expansionen und n die Expansionen nur an dem betrachteten Knoten sind. Die Aufgabe der UCT Formel ist das Erreichen von zwei im Konflikt stehenden Zielen. Das erste Ziel ist es die Knoten die bisher die höchsten Chancen auf den Gewinn haben, tiefer zu simulieren, um eine bessere Genauigkeit des besten Zuges zu haben. Das zweite Ziel ist, Knoten die noch nicht sehr oft besucht worden sind genauer zu untersuchen, da diese vielversprechender sein könnten als gedacht. Für die Balancierung der beiden Ziele gibt es den Parameter c [MP19].

Als Beispiel wird die zuvor verwendete Stellung aus 2 benutzt, aus welcher der MCTS-Baum in Abbildung 7 entsteht. „B“ steht für die Siege aus Schwarzer und „W“ für Siege aus Weißen Sicht, nach der Beendigung von 33 Simulationen. Weiß entscheidet sich in diesem Baum für den gelben Knoten, da dieser bei Betrachtung der Gewichtung von Siege eine höhere Gewinnchance für ihn hat.

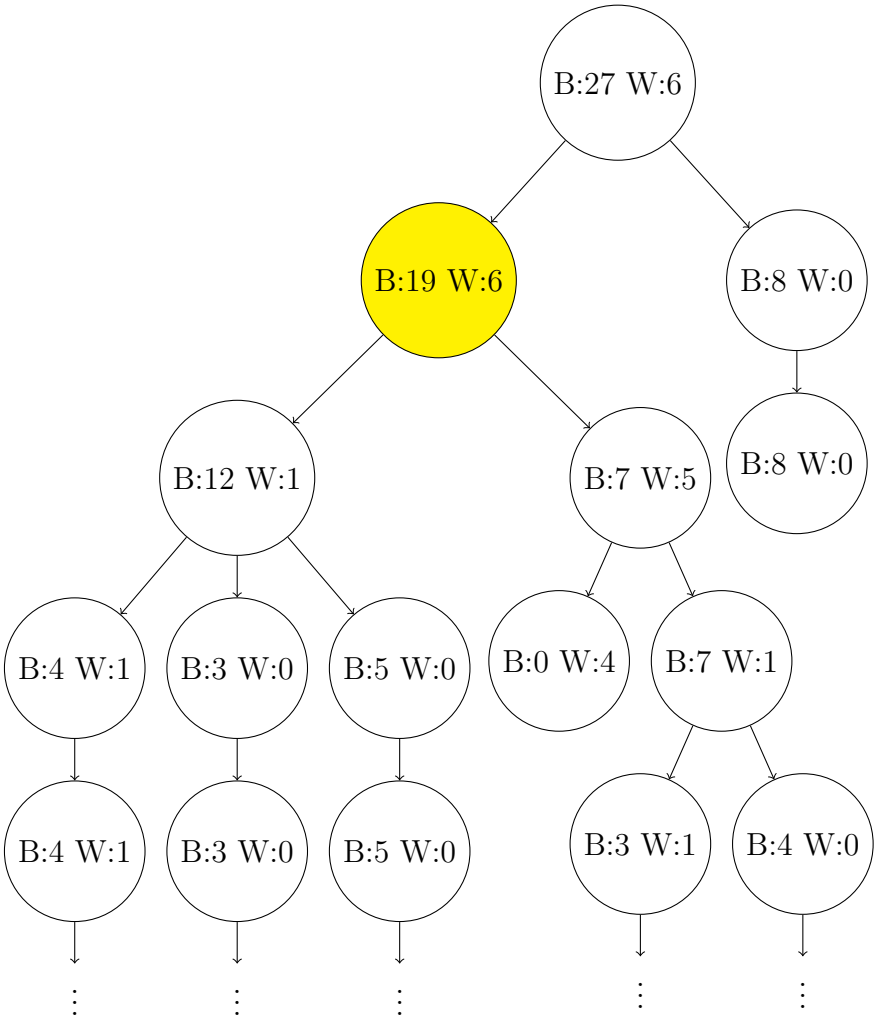

Abbildung 7: Minimax-Baum mit Bewertung der Stellungen

3 Architektur der Software

Dieses Kapitel setzt sich mit der Softwarearchitktur auseinander.

3.1 Überblick

Um für eine bessere Modularität und Erweiterbarkeit zu sorgen, ist die Software auf drei Teile aufgeteilt, siehe Abbildung 8:

- Gameserver
- ReversiXT GUI
- Gameclient

Der Gameserver hält die Spielelogik und den Zustand des Spieles. Er prüft gespielte Züge auf ihre Gülgitkeit und entscheidet ob ein Spiel entschieden ist. Er ist außerdem für die Spielzeitverwaltung und die Reihenfolge, in der Spieler an der Reihe sind verantwortlich. Die ReversiXT Graphische Oberfläche (GUI) ist das Bedienelement für den Endnutzer, um Spiele gegen verschiedene KI's oder andere Nutzer starten zu können. Die künstliche Intelligenz Logik befindet sich im Gameclient, er berechnet abhängig vom Spielfeld und Algorithmus den nächsten Zug den die KI für den besten hält.

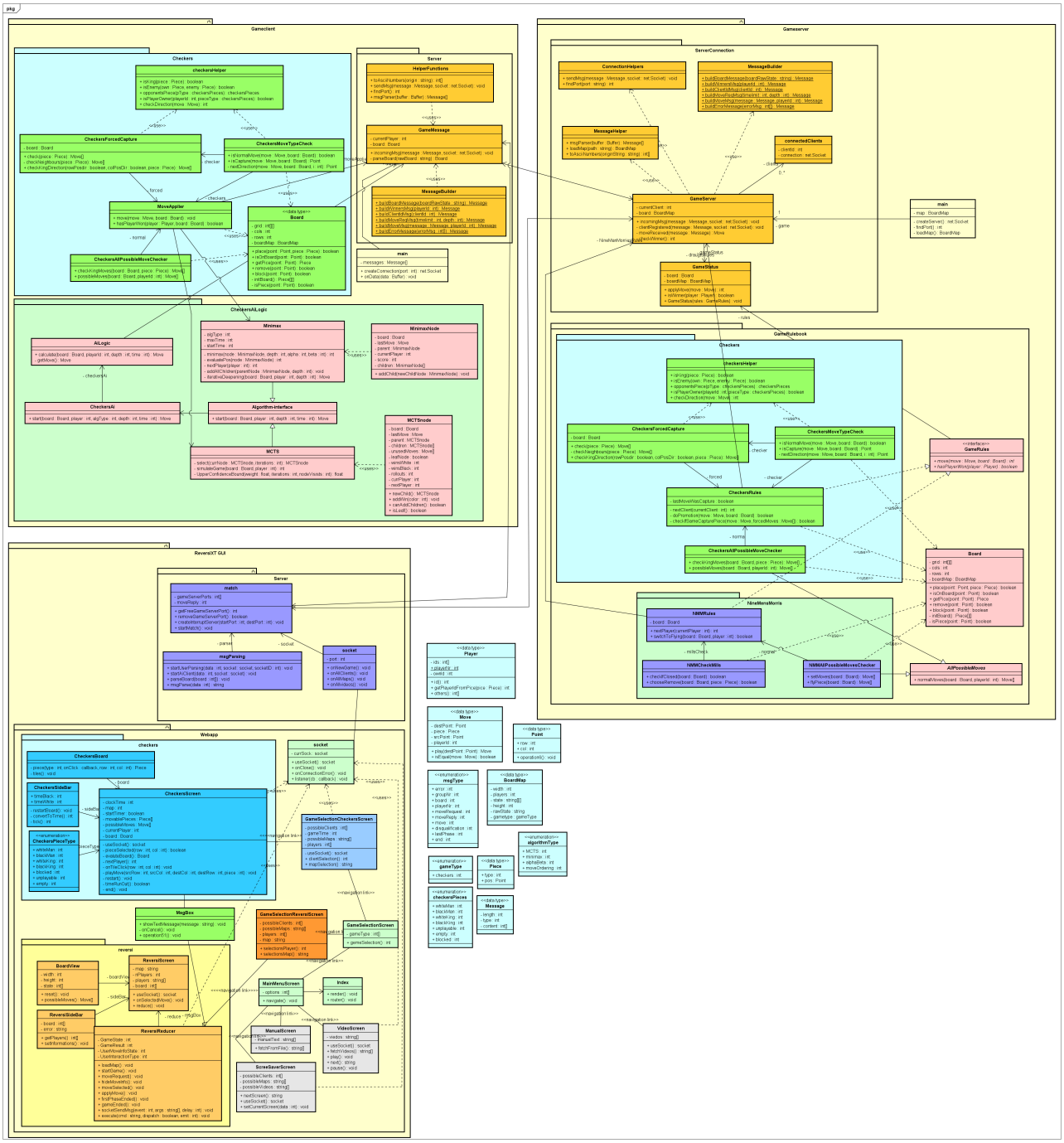

Abbildung 8: Die gesamte Anwendung im UML Klassendiagramm

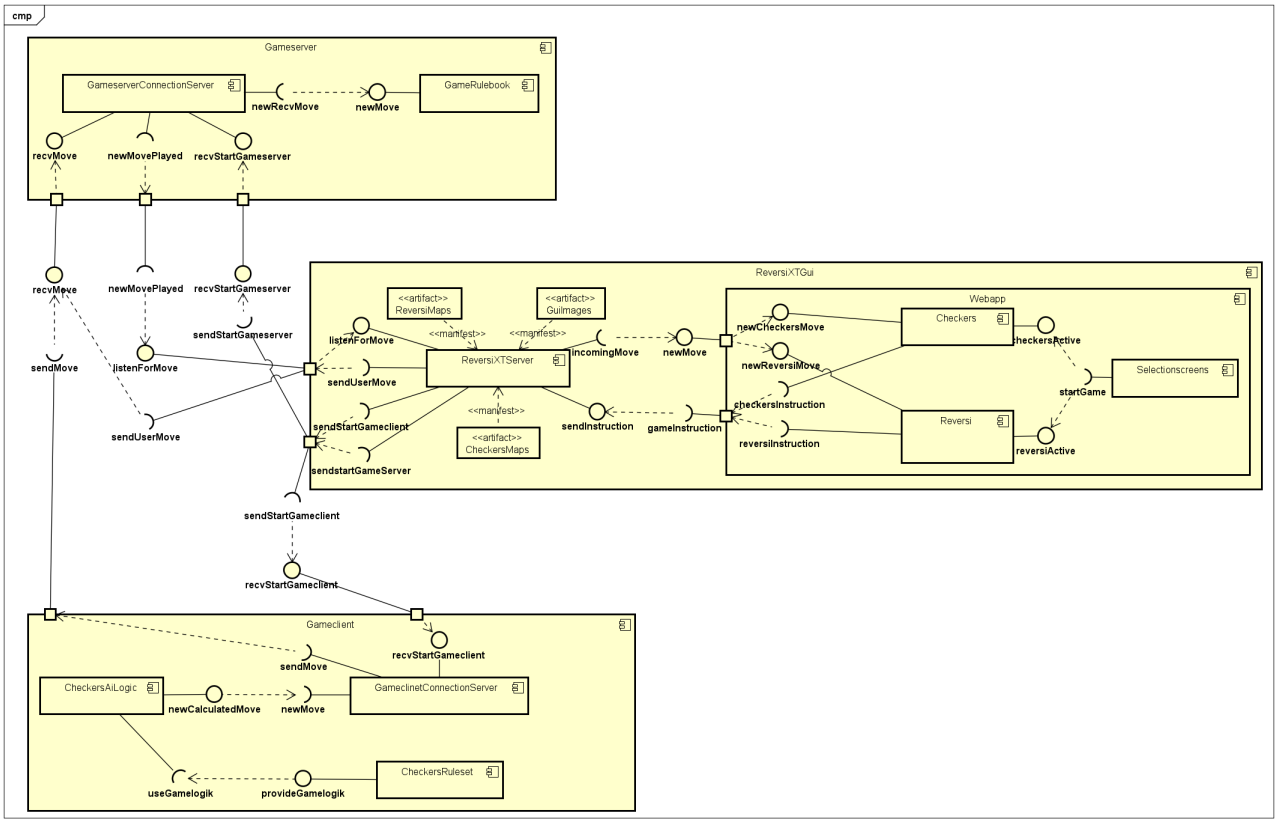

Abbildung 9: Die gesamte Anwendung im UML Komponentendiagram

3.2 Kommunikation der Komponenten

Da die Applikation mit mehreren Spielmodi ausgestattet ist, ändert sich die Kommunikation und welche Komponente gestartet werden muss, abhängig vom Modus. Die verfügbaren Spielmodi sind:

- Benutzer gegen Benutzer
- Benutzer gegen KI
- KI gegen KI

Egal welche Option gewählt wird, der Gameserver wird immer gestartet, da dieser die Züge überprüft und Sieg oder Niederlage auswertet. Bei Benutzer gegen Benutzer wird die KI Komponente der Software nicht gestartet, es kommuniziert die GUI-Komponente direkt mit Gameserver. Wird sich für zwei KI Clients die gegeneinander spielen entschieden, so werden auch zwei gestartet. Die Kommunikation findet nur mehr von Gameserver mit den beiden KI-Clients statt, jedoch hat die GUI-Komponente eine Man-in-the-Middle-Funktion wodurch sie die Kommunikation abhört und die gespielten Züge darstellen kann.

Abbildung 10 stellt ein UML Sequenzdiagramm dar, bei welchem ein Benutzer gegen die KI spielt. Zuerst startet der Benutzer über die GUI mit dem Spielmodus ein Spiel, dadurch wird der

Gameserver gestartet, sowie ein Gameclient. Würde man stattdessen KI gegen KI als Parameter mitsenden würden zwei Gameclients gestartet werden. Der Gameserver wartet wärenddessen bis zwei Spieler bei ihm registriert haben. Ist die Registrierung abgeschlossen, werden Züge abwechselnd von den Clients angefordert, bis ein Spieler gewonnen hat.

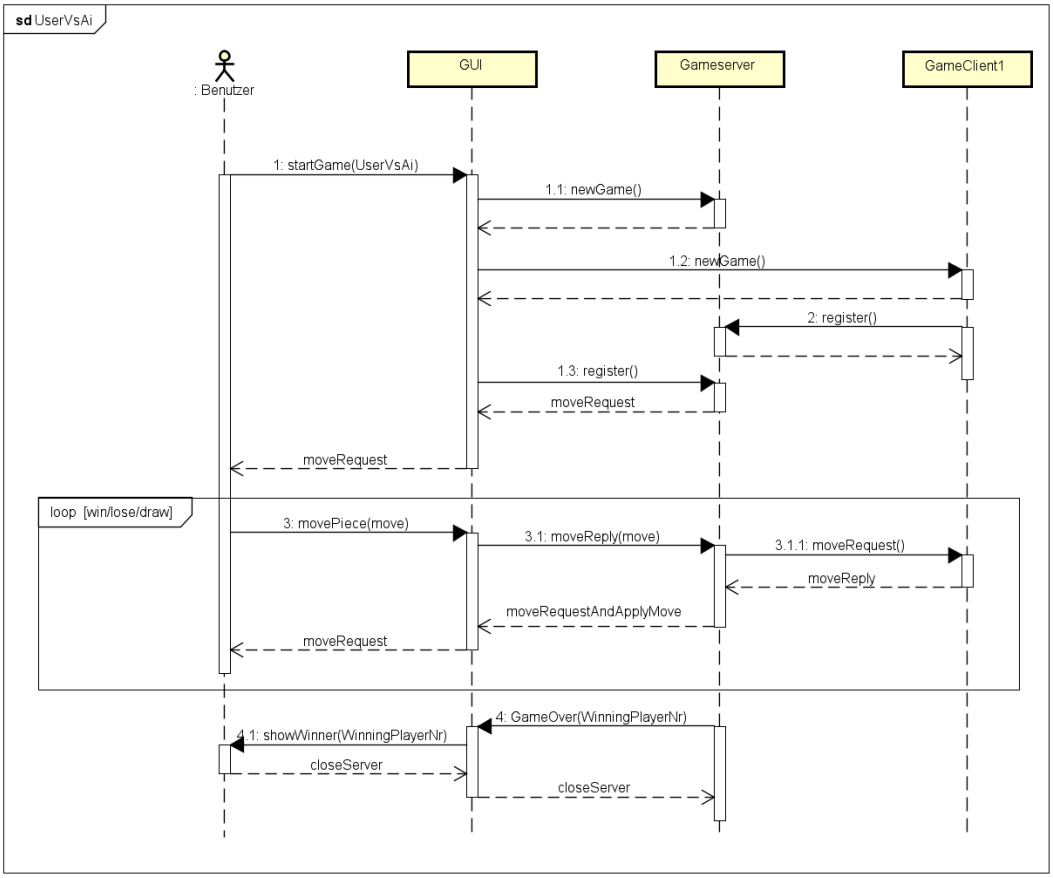

Abbildung 10: Das UML Sequenzdiagramm der Kommunikation von KI gegen Benutzer

3.3 Dame Spiellogik

Die Spiellogik für Dame ist in mehrfach in der Anwendung vorfindbar. Zum einen um Züge im KI-Client zu testen und zum anderen für den Gameserver, zur Überprüfungen und Aktualisierung der Züge. In den Abbildungen 13 und 11 befindet sich die Spiellogik im Paket „Checkers“. Diese Redundanz ist wichitg, da es sich beim Server und Client um eigenständige Prozesse handelt und diese nicht die gleiche Code-Basis haben. Eine Möglichkeit diese Redundanz zu vermeiden wäre, die Validierung in den Gameserver zu verschieben und dem Gameclient zu zwingen, dass jeder Zug von Gameserver validert werden muss. Dies erhöht aber die Netzwerkkommunikation und verlangsamt das finden eines Zuges, durch den daraus entstehenden Overhead.
Die Hauptklassen der Spiellogik sind, die Board-, Move- und Player-Klasse. Dabei hält die Board-Klasse die Information über das aktuelle Spielbrett, sowie Methoden um Spielsteine auf

dem Spielbrett zu entfernen, oder zu bewegen. Die Move-Klasse dient zum repräsentieren eines Zuges, sie besteht aus dem Uhrsprungspunk, den Zielpunkt, dem Spieler der den Zug getätigt hat und um welchem Spielstein es sich handelt.
Die Klassen CheckersForceCapture und CheckersAllPossibleMovesChecker, sind Klassen zum finden von aller möglichen Züge die aus einer Stellung heraus gespielt werden können. Dabei ist CheckersForceCapture verantwortlich für Züge bei denen ein Spieler gezwungen wird eine Figur zu schlagen und CheckersAllPossibleMovesChecker für alle Züge die keinen Schlagzwang haben. Die Aufteilung in zwei Klassen hat den Hintergrund, dass falls es sich um einen Schlagzwang handelt, braucht die AllPossibleMoves-Klasse nicht erzeugt werden um Züge zu finden.
Um einen von der Netzwerkkommunikation eingehenden Zug zu überprüfen, werden die Metheoden der CheckersMoveTypeCheck-Klasse verwendet. Sie vergleichen einen gegebenen Zug, den zum Beispiel der Benuzter getätigt hat, mit den Zügen die aus CheckersForceCapture und CheckersAllPossibleMovesChecker hervorgehen. Dieses Vergleichen ist sehr wichitg für den Gameserver, da alle Züge die von den Clients kommen valide sein müssen. Im Gameclient wird das Vergleichen nur als Extravalidierung verwendet um den gesendeten Zug aus dem Gameserver nochmal zu überprüfen und so Fehler zu minimieren.
Das Anwenden eines Spielzuges auf das Spielbrett wird mittels der MoveApplier-Klasse im Gameclient und mit der CheckersRules-Klasse im Gameserver getätigt. Beide verfolgen im Prinzip das selbe Ziel, der Unterschied wird im Implementierungskapitel 5.5 erklärt.

3.4 Gameserver

Als Gameserver wird der Teil der Software betrachtet, welcher für das einhalten der Spielregeln und die Verwaltung der Spielfeldzustandes, verantwortlich ist.

3.4.1 Softwareaufbau des Gameservers

Die Software des Gameservers ist in zwei Pakete aufgeteilt, in das Gamerulbook- und Serverconnection-Paket. Der Serverconnection Teil ist für die Verbindung und das Dekodieren der Nachrichten der Clients verantwortlich. Nachdem ein Client verbunden ist und seine Nachrichten schickt, wird diese dekodiert und falls das Protokoll korrekt eingehalten wird, an den Gamerulbook Teil weitergeleitet. Das Gamerulbook Paket verwaltet den Spielzustand, sowie die Spielregeln des laufenden Spieles. Es ist über das GameRules Interface mit dem Serverconnection Paket verbunden. Ein beliebiges Spiel kann mittels dieses Interfaces als eigenes Paket festgelegt werden, was zu einer Möglichkeit führt den Gameserver um beliebig viele Spiele erweitern zu können. Über den weiteren Inhalt des „Checkers“ Paketes wird in Kapitel 3.3 berichtet.

Abbildung 11 zeigt den Aufbau als UML-Klassendiagramm, dabei sind Dame und Mühle as Spiele über das GameRules Interface als Pakete Beispielhaft gezeigt.

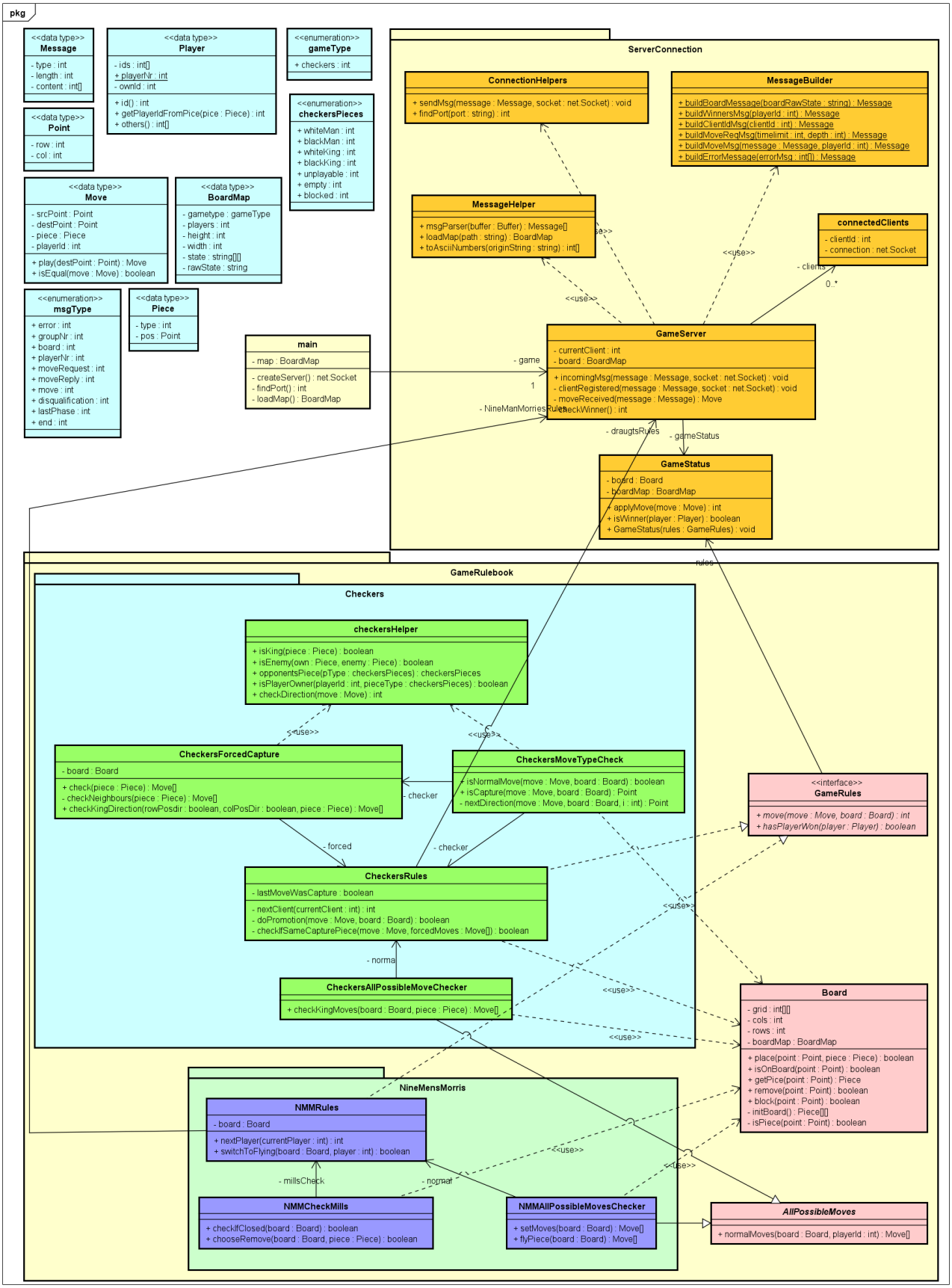

Abbildung 11: Das UML Klassendiagramm des Gameservers

3.5 ReversiXT GUI

Die ReversiXT Graphische Oberfläche (GUI) ist der Teil der Software der für die eigentliche Userinteraktion gedacht ist. User Können über die GUI spiele starten, bei welchen sie gegen KI's oder andere User spielen können.

3.5.1 Softwareaufbau der ReversiXT GUI

Die ReversiXT GUI besteht aus zwei Teilen, dem Server und dem Webapp-Paket, siehe 12. Dabei behandelt der Server, die Verbindung zu den anderen Komponenten. Er startet den Gameserver und je nachdem wie viele KI's als Spieler gewählt werden, startet er auch die Gameclients dafür. Sind die anderen Komponenten gestartet, kommunziert der Server der ReversiXT GUI nur noch mit Gameserver und benutzt dabei das Protokoll welches dieser vorschreibt. Die Webapp behinhaltet das Benutzer Interface, über welches die Software für den Endnutzer bedient werden kann. Über das Menü kann kann die Gameselection augerufen werden, bei welcher die verfügbaren Spiele ausgewählt werden können. Wird ein Spiel ausgewählt, gibt es ein weiteres Menü über dass Spielspezifische Parameter einstellbar sind. Die Spiele sind als eigene Pakete eingegliedert und behinhalten, je nach Spiel ein Spielbrett und Möglichkeiten mittels Touch-Berührungen Züge auszuführen. Durch diese Untergliederung in weitere Pakete kann die Software leicht um weitere Spiele erweitert werden.

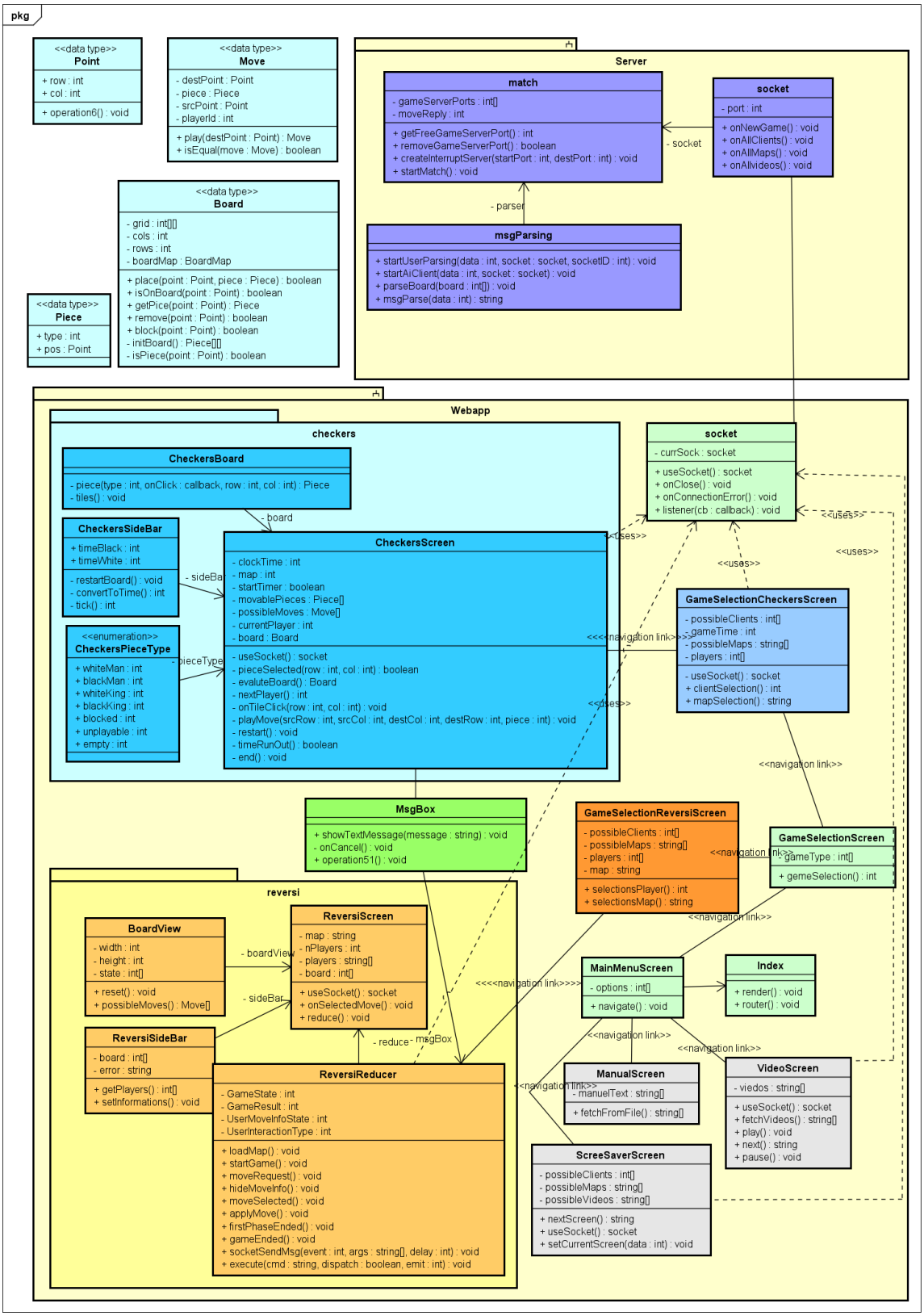

Abbildung 12: Das UML Klassendiagramm der ReversiXT GUI

3.6 KI Client

Der KI Client beinhaltet die Logik der Künstlichen Intelligenz der Applikation. Wird ein Spiel gegen einen KI Client gestartet, so wird dieser gestartet und agiert als Gegenspieler zum User. Der User kann auch ein Spiel bei welchem zwei KI's gegeneinander spielen starten, wodurch zwei KI Clients gestartet werden.

3.6.1 Softwareaufbau des KI Clients

Der KI Client besteht aus drei Paketen, der KI-Logik, der Spiellogik und der Serververbindungslogik, siehe 13. Die Spilelogik, beeinhaltet ein Momentanzustand des Spielbrettes, sowie Möglichkeiten dieses nach belieben zu modifizieren und ist in Kapietel 3.3 erklärt. Das KI-Paket benutzt diese Logik um Züge auf auszuwerten und den bestmöglichen Zug zu wählen. Um verschiedene KI-Algorithmen verwenden zu können wird ein Interface bereit gestellt, wird die Applikation um einen weiteren KI-Algorithmus erweitert, so muss dieser das Interface verwenden. Wird eine Auswertung des Nächsten besten Zuges abeschlossen, so sendet das KI-Paket sein Ergebnis an die das Server-Paket. Dieses ist verantwortlich für das Versenden und kodieren der Nachrichten. Nachrichten werden vom KI-Client zum Gameserver geschickt und müssen dabei das Protokoll von diesem einhalten.

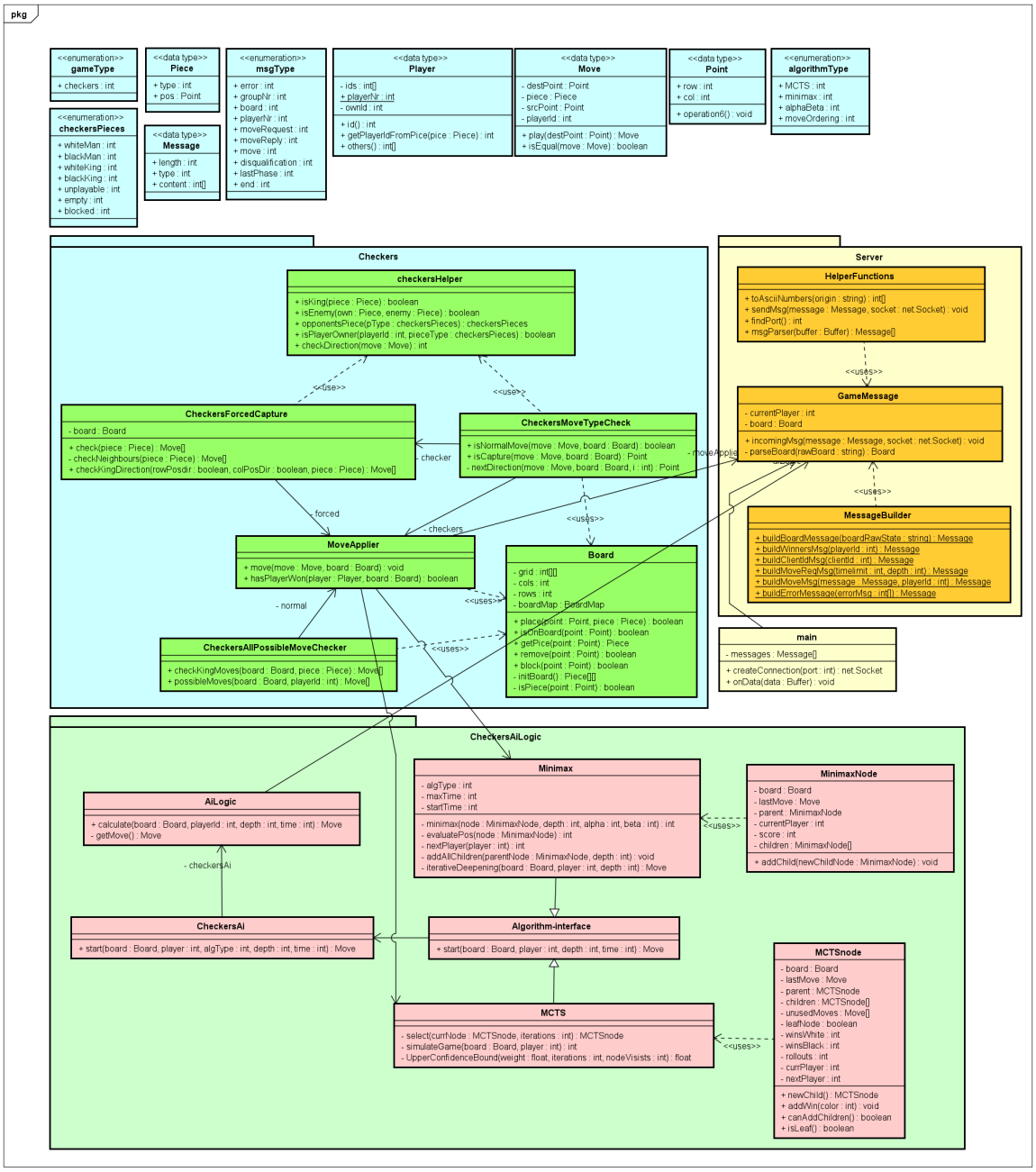

Abbildung 13: Das UML Klassendiagramm des KI Clients

4 Hardware

Dieses Kapitel handelt von der verwendeten Hardware für welche die Software erstellt worden ist. Die Ursprüngliche Idee des Projektes ist, einen Vorzeigegegenstand für Messen, wie den Tag der Informatik, zu haben. Um dies zu realisieren, besteht das Projekt aus einen Touchscreen, welcher in einem Tischförmigen Gerüst verankert und mit einen Raspberry Pi verbunden ist.

4.1 Raspberry Pi

Die Software wird auf einen Raspberry Pi Model 3 ausgeführt. Bei dem Betriebssystemen handelt es sich hierbei um Raspbian, ein auf Debian Basiertes Linux Betriebssystem. [Ras] Da die Software auch mit anderen Betriebssystem kompatibel ist, könnte man den Pi durch ein beliebig anderes Gerät ersetzen, dass entweder Linux oder Windows unterstüzt. Der Raspberry Pi ist jedoch anhand seiner Kosten und Größe perfekt für das Projekt, da er in das Gerüst passt.

4.2 Touch Monitor

Die Bedienung der Benutzerobefläche erfolgt hauptsächlich mit dem Touchscreen, jedoch kann die Software auch auf normalen Monitoren ausgeführt werden. Der Touch Monitor ist ...Eingaben können entweder durch Berührung des Touchscreens mit dem Finger, oder mit einer klassischen Computermaus ausgeführt werden. Für Texteingabe, kann entweder das integrierte Softwarekeyboard, oder eine normale Tastatur verwendet werden.

5 Implementierung

5.1 Programmiersprachen und Frameworks

Da die Software aus Mehreren individuellen Applikationen besteht, werden auch verschiedenste Programmiersprachen und Frameworks for diese Verwendet. Im Folgenden werden alle Programmiersprachen und Frameworks die Verwendung finden erklärt.

5.1.1 React.js

Um die Software so kompatibel wie möglich zu gestalten, basiert die Software auf Webtechnologie. Dies bedeutet, jedes Endgerät, welches einen neuren Internetbrowser unterstüzt kann die Software zu einem Teil ausführen. Da eine Webseite in reinen Javascript, HTML und CSS zu schreiben viel Aufwand benötigt, wird ein Framework wie React.js verwendet. Bei React handelt es sich um eine Javascript Library zum erstellen von Benutzeroberflächen. Die Vorteile von React zu reinem Javascript sind:

- Einfach Dynamische Websiten zu entwickeln
- Wiederverwertbare Komponenten
- Methoden und Markup gehören zusammen
- Funktionale Programmierung mit pure functions

Vorallem das erleichterte Erstellen von Dynamischen Websiten ist Vorteilhaft, da die Benutzerobefläche wärend ein Spiel gespielt wird, dynamisch gehalten werden muss. Im klassischen Javascript muss das Document Object Model (DOM), mühsam verändert werden. Diese DOM-Manipulationen sind sehr Fehleranfällig und können Memoryleaks verursachen. React sitzt hingegen zwischen Komponenten und dem DOM und übernimmt die komplette DOM-Manipulation. Das Verwenden von Komponenten ist auch Vorteilhaft, da zum Beispiel einzelne Spielfiguren als Komponenten festgelegt werden können und so Code mehrfach eingespart werden kann. [Rea].

5.1.2 Node.js

Der Gameserver, der Gameclient und die Serverlogik der GUI, sind mit Node.js implementiert. Node.js ist eine asynchrone ereignisgesteueretes Javascript-Laufzeitumgebung, welche Javascript-Code außerhalb des Webbrowsers ausführen kann. Node.js wird hauptsächlich zur Programmierung von Netzwerkanwendungen, wie Webservern verwendet. Da einen Großteil der Anwendung die Kommunikation der einzelnen Komponenten ausmacht, ist Node.js perfekt für dieses Szenario. Ein weiterer Vorteil von Node ist, dass Node.js unabhängig vom Betriebssystem ist. Dies bedeutet, kann die V8 Javascript-Laufzeitumgebung auf dem Betriebssystem ausgeführt werden, so kann man auch die Node-Applikation dort zum laufen bringen. [Noda]

5.1.3 Typescript

Node.js wir eigentlich in reinem Javascript geschrieben, jedoch leidet Javascript daraunter, dass es keine Typisierung hat und sich somit leicht Fehler einschleichen können. Typescript ist eine auf Javascript basierende Programmiersprache, welche statische Typisierung unterstützt. Ähnlich wie bei C und C++ ist valider Javascript-Code auch valider Typescript code wodurch Typescript eine Obermenge von Javascript ist. Nachdem Typescript-Code geschrieben wird, kann er in reines Javascript kompiliert werden. Der Gameserver und der Gameclient sind in Typescript geschrieben um Fehler durch die Typsicherheit zu verhindern.

5.1.4 C/C++ Node Addons

Da Javascript eine im Vergleich sehr langsame Programmiersprache ist, wäre es sinnvoll den künstliche Intelligenz Teil der Applikation, welcher für die Berechnung der nächsten Züge verantwortlich ist, in einer Performanteren Sprache welche auch Memorymanagement bietet, wie C oder C++ zu schreiben. Node.js bietet hierfür die Möglichkeit an Module mittles C++ zu implementieren und von deren Geschwindigkeit zu profitieren [Nodb]. Ein C++ Node Addon wird zu Maschinencode kompiliert, welcher von der normalen Node-Anwendung, über eine Javascript-Funktion ausgeführt werden kann. Der Vorteil hierbei zum reinen C++ ist, dass der Code in der V8 Laufzeitumgebung ausgeführt wird, was ihn Plattformunabhängig macht. [Cno]

5.2 Kommunikation

in 3.2 ist bereits auf die Kommunikationsstruktur geschildert, im folgenden wird die genauere Implementierung beschrieben

5.2.1 Verwendete Technolgie

Da der Gameserver, der Gameclient und die GUI eigenständige Prozesse sind, die nach bedarf gestartet und getstoppt werden, brauchen sie eine Möglichkeit Nachrichten auszutauschen. Dafür werden die von Node.js bereitgestellten TCP-Sockets verwendet. Der Vorteil von TCP-Sockets gegenüber zum Beispiel UDP, ist dass aufrechterhalten einer Verbindung. So muss sich der Client oder die GUI nur einmal zum Server verbinden und kann im Laufe eines gesamten Spieles diese Verbindung nutzen. Ist ein Spiel zu Ende, kann die TCP-Verbindung gekappt und die Sockets gescholossen werden. Da die GUI in zwei Softwarepakete getrennt ist siehe 12, der React.js Webapp und einem Node.js Server, verwenden diese Pakete auch eine eigen Netzwerkverbindung. Hierfür wird die Javascript library Socket.io verwendet, welche auf basis von Websockets basiert.

5.2.2 Kommunikationsprotokoll des Gameservers

Im Abschnitt 3.2 wird eine Kommunikation zwischen dem Gameserver und seinen Clients beschrieben, bei welcher der Server nur Anfragen eines festgelegten Protokolles Akzeptiert. Das Protokoll für den Server orientiert sich stark nach dem Protokoll des Reversi-Gameservers und ist nur in einigen stellen erweitert worden.

Typ (8-Bit-Integer)	Länge der Nachricht n (32-Bit-Integer)	Nachricht (n Bytes)

Abbildung 14: Der Aufbau einer Nachricht

in Abbildung 14 ist der Aufbau einer Nachricht dargestellt. Es gibt ingesamt neun verschiedene Nachrichtentypen, dazu gehören neues Spiel starten, Zugaufforderung, Spiel Ende und viele weitere. Die meisten Nachrichten von einen gewissen Typ haben immer die selbe Länge, es gibt jedoch Ausnahmen, wie die Spielbrettnachricht, bei welcher ein 10x10 oder 8x8 Damenspielbrett gewählt werden kann. Die Nachricht selber entspricht dann den eigentlichen Daten, die übertragen werden, wie zum Beispiel, den Feldern welche besetzt sind beim Spielfeld, oder das Ursprungsfeld und das Zielfeld bei einer Zugantwort.

5.3 GUI Dame Erweiterung

Da die Anwendung vor der Erweiterung um das Spiel Dame nur Reversi als Spiel hatte, musste ein Extra Menü implementiert werden, über welches man Dame auswählen kann. Des weiterem wird ein weiteres Menü benötigt, bei welchem der Algorithmus und die Spielbrettgröße gewählt werden kann, siehe Abbildung 15. Dieses Menü zeigt eine Auswahl für Spielfeldgrößen,

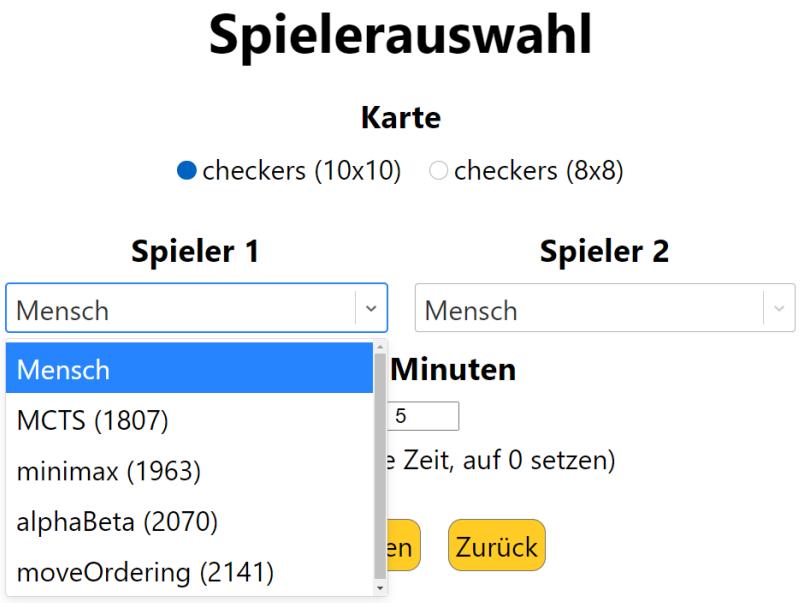

Abbildung 15: Das Auswahlmenü für ein Damespiel

5.4 Verwendete KI-Algorithmen

In Kapitel 2 wurden einige künstliche Intelligenz Algorithemen beschrieben, welche auch im Dame-Spiel der Applikation implementiert sind. Die Applikation bietet vier Algorithemen zur Auswahl an, gegen die der Spieler antreten kann, siehe 15. Diese Algorithemen sind: Minimax, Alpha-Beta-Pruning, Zugsortierung und MCTS. Aus Abbildung 13 kann man entnehmen, dass

es zwei Hauptalgorithmen in dem Paket CheckersAiLogik gibt, Minimax und MCTS. Diese Aufteilung kann vorgenommen werden, da Alpha-Beta-Pruning und Zugsortierung auf Minimax basieren.

5.4.1 Minimax

Der Minimax-Algorithmus wird im wie in 2.2 erklärt, durch einen Baum repräsentiert. Dadurch, dass jeder Knoten einen Spielbrettzustand repräsentiert und durch den Algorithmus einen Wert zugewiesen bekommen, hat die Applikation eine MinimaxNode-Klasse. Diese Klasse ist hauptsächlich für das Speichern der Informationen zuständig, welche dann einfach von der eigentlichen Minimax-Klasse zur Berechnung verwendet werden können. Damit ein vollständiger Baum aufgebaut werden kann, enthält die Minimax-Node-Klasse ein Array mit Zeigern auf allen Kindknoten, was eine Vorwärtstraversierung des Baumes erlaubt. Um eine Berechnung zu starten, braucht die start-Funktion ein Spielbrett, den Spieler der an der Reihe ist, eine Tiefe bis zu der berechnet werden soll und oder ein Zeitlimit. Ist ein Zeitlimit angegeben, so benutzt die start-Funktion Iterative Deepening, siehe 2.3, dadurch kann die Berechnung so gut wie Möglich die komplette Zeit ausnutzen. Ist kein Zeitlimit angegeben, so wird spezifizierte Tiefe dem Minimax übergeben und dieser rechnet so lang bis er diese erreicht. Wird Minimax mit Iterative Deepening gestartet, so wird ein Zeitwert zum Anfang gespeichert. Dieser Zeitwert wird wärend der Berechnung von Minimax mit dem momentanen Zeitwert verglichen, ist die Differenz größer oder gleich des Zeitslimits, wird ein mittels einer C++ Exception die Momentane Berechnung abgebrochen und nachdem der zurückkehren in den Catch-Block der letzte komplett berechnete Zug verwendet. Wird Minimax gestartet, holt sich der Algorithmus mittels der Verwendung der Funktion AddAllChildren, alle Kinderspielbretter. Diese definieren sich aus allen möglichen Zügen die von einer Stellung aus möglich sind, siehe Kapitel 3.3. Der Vorgang wird rekursiv wiederholt, bis die geforderte Tiefe erreicht wird und die Bewertungsfunktion aufgerufen wird. Die Ergenisse werden dann rekursiv rückwärts in den MinimaxNodes des Baumes eingetragen.

Um mit Minimax den besten Zug berechnen zu können wird eine gute Bewertungsfunktion benötigt, die verwendete hat folgende Eigenschaften:

- Normale Figur: +1 für Weiß und -1 für Schwarz
- Dame: +3 für Weiß und -3 für Schwarz
- Spielende: +40 für Weiß und -40 für Schwarz

Erweiterungsmöglichkeiten wären, Bedrohte Figuren, sowie Figuren in der Spielmitte und in den Grundreihen mit in die Berechnung miteinfließen zu lassen. Man muss ich aber immer vor Augen halten, dass eine komplexere Bewertungsfunktion mehr Berechnungszeit benötigt, aber bessere Ergebnisse liefern könnte.

5.4.2 Minimax Erweiterungen Alpha-Beta-Pruning und Zugsortierung

Dadurch, dass Alpha-Beta-Pruning und Zugsortierung Erweiterungen von Minimax sind, können diese beiden Algorithmen mit in die Minimax-Klasse gepackt und über Übergabeparameter des Konstruktors an- und ausgeschalten werden. Dadurch hat man weniger Code, kann aber trozdem noch verschieden Algorithmen mit unterschieder Stärke im Menü wählen. Da Minimax rekursiv implementiert ist, kann Alpha-Beta-Pruning sehr elegent mit wenigen Zeilen Code geschrieben werden. In Listing 1 sieht man die Einfache Implementierung des Alpha-Beta-Prunings im Maximierer Teil von Minimax. Zuerst wird überprüft ob es sich um reinen Minimax handlet, oder um Alpha-Beta oder Zugsortierung. Danach wird der Alpha Wert mit dem Maximum aus dem alten Alpha-Wert und dem durch die Bewertungsfunktion berechneten eval Wert berechnet. Das Codestück im Minimierer hätte an dieser Stelle ein Min anstelle des Max. Das Pruning finden für den fall dass Beta kleiner gleich Alpha ist statt, dadurch wird der Teil dieses Zweiges nicht weiter untersucht.

```
1 if(algType != MINIMAX) {
2     alpha = std::max(alpha, eval);
3     if (beta <= alpha) {
4         break;
5     }
6 }
```

Listing 1: Code der Implementierung des Alpha-Beta-Prunings

Um eine Zugsortierung im Minimax zu erreichen, wird im Teil von AddAllChildren des Minimax die Bewertungsfunktion der Zugsortierung aufgerufen. Dadurch dass es sich um Array von Zeigern handelt, welche auf die Kindknoten zeigen muss nur dieses Array unter Berücksichtigung der Bewertungsfunktion sortiert werden, was durch die Hilfe der Standartbibliotek sehr einfach zu implementieren ist. Die Verbessung wird nun durch das Aufrufen des Alpha-Beta-Prunings erreicht, siehe Kapitel 2.5.

5.4.3 MCTS

Der MCTS-Algorithmus ist ein auf Simulation basierender Algorithmus, er ist in Kapietel 2.6 erklärt worden. Ähnlich wie beim Minimax-Algorithmus verwendet der MCTS-Algorithmus auch einen Baum mit Knoten welche den Spielbrettzustand halten. Der Unterschied liegt hierbei bei den Informationen, die ein Konten hält. Eine MCTSnode hält die Anzahl an verlorenen und gewonnen Spielen, die aus der Simulation hervorgehen. Jeder Knoten hat ein Array mit Zeigern auf seine Kinderknoten, dadruch entseht die bekannte Baumstruktur. Im Gegensatz zur MinimaxNode, werden nicht bei einer Iteration alle Kinder eines Knoten mit AddAllChildren hinzugefügt, sondern mittels des UCTS-Ergbnisse, wird der beste Kandidat gewählt. Für den Parameter c wird zwei gewählt, da in vielen gelungen anderen Spielen die Zahl zwei zu guten erfolgen geführt hat. Da der MCTS nicht unendlich lange simulieren soll, braucht er ein Zeitlimit oder eine Anzahl an Simulationen nach welchen er die Simulation abbrechen und das Ergebnis zurückliefern soll. Die Zeitüberprüfung wird ähnlich wie bei Minimax gehalten, ist der Zeitpunkt

zum abbruch gekommen wird eine C++ Exception geworfen, was zum Stop der Simulation führt. Um die Simulation Ausführen zu können werden die Methoden, welche in 3.3 beschrieben sind verwendet. Das Ergebnis des MCTS ist der Kindknoten, welcher den besten Wert nach den Simulationen hat, dieser wird auch vom Algorihtmus zurückgegeben.

5.5 Dame Spiellogik

Im Kapitel 5.5 ist bereits der Aufbau der Spiellogik erklärt. Die Besonderheit bei der Implementierung ist, dass sich der Gameserver und der Gameclient in der Implementierung des MoveAppliers und CheckersRules unterscheiden. Der Grundliegende Unterschied ist, dass beim Berechenen eines neuen Zuges durch den Gameclient sehr oft die Methode „move“ aufgerufen werden muss. Da diese Züge nicht von anden Komponenten der Anwendung kommen, müssen sie nicht auf Korrektheit überprüft werden. Eine der Hautpaufgaben des Gameservers ist die Überprüfung und Validierung der eingehenden Züge, diese Validierung kostet aber auch Zeit welche die Berechnung im Gameclient verlangsamen würden und daher im Gameclient bei der Berechnung weggelassen wird.

6 Testing und Simulation

6.1 Simulation

Da die Software mehrere KI-Algorithemen beeinhaltet und diese unterschiedlich stark sind, werden Simulationen verwendet um die erwartete Stärke der Algorithemen zu bestätigen. Im laufe der Simulationen spielt jede KI gegen die anderen KI's. Um die Ergebnisse in Form von Stärke festzuhalten, bekommt jeder Algorithmus ahnhand seiner Performance eine ELO-Zahl zugewisen.

6.1.1 ELO

Das Elo System ist eine Kennzahl für die relative Spielstärke die ein Spieler in einem Nullsummenspiel hat. ELO wird hauptsächlich in Schach und Go verwendet, findet aber auch immer mehr Anwendung in anderen Sportarten wie Tischtennis oder auch in Computerspielen. Bei der Berechnung von Elo wird ein Spiel von zwei Spielern untersucht, dabei ist die Elozahl der Spieler die erwartete Spielstärke. Demnach wird von einem Spieler mit höherer Elo-Zahl als sein Gegener vielmehr ein Gewinn erwartet. Dementsprechend verhält sich auch die Eloänderung nach dem Spiel. Verliert ein Spieler A mit einer höheren Elo gegen einen Spieler B mit gerigerer Elo, so verliert A viel Punkte und B bekommt viel Punkte dazu. Gewinnt jeoch A, so bekommt er nur sehr wenig Punkte hinzu und B verliert nur sehr wenig.
Die in der Applikation verwendete Elo Zahl wird wie folgt berechnet:

$$R_1 = 10^{\frac{E_1}{400}} \tag{2}$$

$$R_2 = 10^{\frac{E_2}{400}} \tag{3}$$

$$F_1 = \frac{R_1}{R_1 + R_2} \tag{4}$$

$$F_2 = \frac{R_2}{R_1 + R_2} \tag{5}$$

$$E_{neu1} = E_1 + k \cdot (S_1 - F_1) \tag{6}$$

$$E_{neu2} = E_2 + k \cdot (S_2 - F_1) \tag{7}$$

Dabei entspricht:

E = Elo Zahl
F = Warscheinlichkeit das der Spieler gewinnt
k = K-Faktor
S = Sieg oder Niederlage (1 oder 0)

Der K-Faktor gibt eine Aussage darüber wie stark sich ein Spiel auf die Eloänderung auswirkt. In der Applikation wurde ein K-Faktor von 30 verwendet, da dieser auch in Turnierschach für Spieler unter 2100 Elo verwendet wird.

6.1.2 Aufbau der Simulationsoftware

Für die Simulation wurde eine Eigene Software geschrieben, welche unabhängig von der ReversiXT GUI arbeitet. Es werden lediglich der Gameserver und der Gameclient benötigt. Um eine Simulation zu starten muss eine Anzahl durchläufen angeben werden, je nach dem wie viele Spiele ausgeführt werden sollen. Startet man ein Spiel wird der Gameserver und zwei Gameclients mit zufällig sich unterscheidenden Algorithmen gestartet. Wird ein Spiel abgeschlossen wird das Ergbnis in einer Datei gespeichert und die Elo aktualisiert.

6.1.3 Interpretation der Ergebnisse

Die vier implementierten Algorithemen der Applikation sind:

- MCTS
- Minimax
- Alpha-Beta Pruning
- Alpha-Beta mit Zugsortierung

Man könnte erwarten dass Alpha-Beta mit Zugsortierung die höchste Elozahl, Alpha-Beta Pruning die zweithöchste, Minimax die dritthöchste und MCTS die geringste Elo- Zahl erhalten würden. Diese Annahme lässt sich durch die Simulation bestätigen.

6.2 Testing

Um die Stabilität der Software, zu verbessern, sind Tests für die anfälligen Softwarekomponenten erstellt worden, durch welche gröbere Fehler vermieden werden können. Die GUI ist weniger Fehleranfällig, da in ihr lediglich Informationen dargestellt werden. Die Kritischen Softwareteile sind der Gameserver und der Gameclient, in welchen das Dame Spiel ausgeführt und neue Züge berechnet werden. Ein Fehler dort wäre verherend, da die Software nicht in einen Korrekten Zustand zurückgeführt werden kann.

6.2.1 Gameserver

Im Gameserver ist der Zustand des Spieles und die Überprüfung der gespielten Züge. Diese Überprüfung darf auch in Spezialfällen nicht fehlschlagen, so muss z.B. ein Spieler in Schlagzwang immer das Schlagen einer Figur ausführen, oder ein Spieler kann nicht noch einmal schlagen, wenn er mit einem Zug bei dem geschlagen worden ist, eine Dame erhält. Um dies zu überprüfen, gibt es Tests, die Spezielle Spielsituationen als Eingabe bekommen und prüfen ob die Software den Zug als Gültig/Ungültig anerkennt. Die Situationen sind entweder Spezialfälle die aus den Damenregeln hervorgehen, oder Standartstellungen bei denen eigentlich kein Fehler auftreten dürfte. Die Testsoftware verwendet das Testing Framework Chai, welches zum testen von Node Anwendungen verwendet wird. Dabei werden mehrere Testfällte beschrieben, diese werden dann über asserts oder expects von Chai ausgewertet. Diese Auswertung verläuft automatisiert, dass heißt wird die Software neu von Typescript in Javascript kompiliert, werden auch die Testfälle ausgeführt.

6.2.2 Gameclient

Das Testen des Gameclients ist sehr wichtig, da kleinste Fehler zu einer enormen Verschlechterung der Performance des KI-Algorithmuses führen. Um den Gameclient zu testen, verwendet man ähnlich wie beim Gameserver Szenarien, in denen es schwirig ist den besten Zug zu finden, oder in sehr einfachen Situationen, bei dem der KI-Algorithmus nicht sehr lagen brauchen sollte um den Perfekten Zug ausfindig zu machen. Da die KI-Logik des Gameclients in C++ implementiert ist, ist die Testsoftware manuel geschrieben. Was bedeutet, dass es handelt sich um eine unabhängige Applikation, welche einen Gameclient startet und diesem mit Spielsituation und Zugaufforderung füttert. Der Client gibt der Testsoftware, dann den nächsten Zug zurück. Das Ergbnis wird mit dem gewünschten Ergbnis verglichen, um zu bestimmen ob ein Fehler oder eine Korrekte Ausgabe getätigt werden muss. In diesem Prinzip startet die Testsoftware den Gameclient mehrfach für verschiedenste Testfällte und wertet diese Ergbnisse aus.

7 Fazit und Ausblick

Literaturverzeichnis

[Kor85] Richard E. Korf. *Depth-First Iterative-Deepening: An Optimal Admissible Tree Search*. Techn. Ber. University of Columbia, 1985.

[MP19] Kevin Ferguson Max Pumperla. *Deep Learning and the Game of Go*. Manning, 2019.

[MSC82] T. A. Marsland M. S. Campbell. *A Comparison of Minimax Tree Search Algorithms*. Techn. Ber. University of Alberta, 1982.

[Rus12] Stuart Russell. *Künstliche Intelligenz ein moderner Ansatz*. Pearson, 2012.

[SPS] M. Sridevi Shubhendra Pal Singhal. *Comparative study of performance of parallel Alpha Beta Pruning for different architectures*. Techn. Ber. National Institute of Technology, Tiruchirappalli.

Quellenverzeichnis

[Cno] *C++Node*. Node C++ Addons. URL: https://nodejs.org/api/addons.html.

[Dra] *Mindsports*. Die Geschichte von Dame. URL: http://www.mindsports.nl/index.php/arena/draughts/478-the-history-of-draughts.

[Int] *World Draughts Federation*. official FMJD rules for international draughts 100. URL: http://www.fmjd.org/?p=v-100.

[Noda] *Node*. Node. URL: https://nodejs.org.

[Nodb] *NodeC++Performance*. Node C++ Performance. URL: https://medium.com/the-node-js-collection/speed-up-your-node-js-app-with-native-addons-5e76a06f4a40.

[Ras] *Raspberry*. Raspberry Pi. URL: https://www.raspberrypi.org/.

[Rea] *Reactjs*. React.js. URL: https://reactjs.org/.

Anhang

Inhalt des beigefügten Datenträgers:

- ...
- ...

A Domändenmodell

Ein toller Anhang, der nicht nur als „*Müllhalde*“ genutzt wird, sondern in dem Bilder und Inhalte auch mit eigenen Worten erklärt werden und den man auch für sich alleine lesen kann. Es sollten auch Referenzen auf die zugehörige ausführliche Behandlung im Hauptteil inklusive Seitenangabe mit `\pageref` gegeben werden.

Screenshot

Unterkategorie, die nicht im Inhaltsverzeichnis auftaucht.