Tiny Chess Bots

Aufgabe

Tiny Chess Bots ist eine Challenge von einem YouTuber namens „Sebastian Lague“. Er stellte ein Framework zur Verfügung und gab der Community die Challenge den bestmöglichen Chess Bot zu erstellen. In dem Folgenden Repo stehen alle Regeln, die befolgt werden müssen: <https://github.com/SebLague/Chess-Challenge> .

Außerdem ist die Rede von einer „Brain Bot Capacity“. Diese besagt, dass die sogenannten Tokens nicht mehr als 1024 werden können. 1 Token ist zum Beispiel die Deklaration eine Variable.

Implementation

Zunächst habe ich mir die Dokumentation angeschaut:

<https://seblague.github.io/chess-coding-challenge/documentation/> .

Anschließend habe ich mir überlegt, wie ich vorgehen möchte. Ich habe mich dazu entschlossen, je nachdem wie es auf dem Schachbrett aussieht zu agieren. Zuerst muss ich dafür sorgen, dass der Bot eine Bewegungsmöglichkeit hat. In der Funktion Think soll der Bot also alle legalen Moves finden und in einem Array speichern. Zum Glück gibt es in der Dokumentation eine Funktion, die uns genau das erlaubt. Danach setze ich das Intelligenzlevel vom dem Bot. Dieses gibt an wie lange er zum evaluieren Zeit hat. Nun habe ich eine Funktion geschrieben, die auf dem Schachbrett nach mehreren Dingen Sucht. Die Funktion EvaluateBoardState überprüft, wie das Schachbrett gerade aussieht und wer mehr Materialpunkte (Schachfiguren) hat. Es gibt drei Kriterien, nach denen wir prüfen:

1. gameState = 0

Es wird geschaut, ob die Königin schon bewegt wurde und an einem schlechten Ort steht.

if (gameState == 0)

{

Piece whiteQueenSquare = board.GetPiece(new Square("d1"));

if (!whiteQueenSquare.IsQueen && whiteQueenSquare.IsWhite)

sum -= 200 \* color;

Piece blackQueenSquare = board.GetPiece(new Square("d8"));

if (!blackQueenSquare.IsQueen && !blackQueenSquare.IsWhite)

sum -= 200 \* color;

}

1. gameState = 0 || 1

Es wird geschaut, ob der König schon bewegt wurde und an einem schlechten Ort steht.

if (gameState == 0 || gameState == 1)

{

Piece whiteKingSquare = board.GetPiece(new Square("e1"));

if (!whiteKingSquare.IsKing)

sum -= 200 \* color;

Piece blackKingSquare = board.GetPiece(new Square("e8"));

if (!blackKingSquare.IsKing)

sum -= 200 \* color;

}

1. gameState = 2

Es wird geschaut, wer wie viele Bauern hat und wo diese stehen.

if (gameState == 2)

{

var playerPawns = board.GetPieceList(PieceType.Pawn, board.IsWhiteToMove);

var opponentPawns = board.GetPieceList(PieceType.Pawn, !board.IsWhiteToMove);

if(board.IsWhiteToMove)

sum += (2 \* playerPawns.Sum(p => p.Square.Rank) \* color) + (2 \* opponentPawns.Sum(p => 7 - p.Square.Rank) \* (color \* -1));

if(!board.IsWhiteToMove)

sum += (2 \* playerPawns.Sum(p => 7 - p.Square.Rank) \* color) + (2 \* opponentPawns.Sum(p => p.Square.Rank) \* (color \* -1));

var kingSquare = board.GetKingSquare(board.IsWhiteToMove);

var distance = Math.Min(Math.Abs(kingSquare.File), Math.Abs(kingSquare.File - 7));

sum -= (10 - distance) \* 20 \* color;

}

Anschließend wird nun der bestmögliche Zug anhand der soeben erfahrenen Methoden erzeugt. Dies geschieht in dem ich in einer foreach Schleife jede mögliche legale Bewegung prüfe und erst einen Wert zurückschreibe, der den gewünschten Kriterien entspricht.

Schlusswort

Nachdem ich dieses Projekt nun abgeschlossen habe, kann ich sagen, dass es auf jeden Fall nicht einfach war. Ich habe mich hier ran getraut ohne großes Wissen über C# oder Visual Studio zu haben. Zum Glück konnten meine Mitschüler mir hier helfen, da diese sich gut auskennen. Es war auf jeden Fall eine interessante und herausfordernde Aufgabe, da es nicht nur Schachstrategie und -taktik erfordert, sondern auch verschiedene Algorithmen und Techniken. Der Schachbot ist noch sehr Simple und kann einige Bewegungen wie die Rochade oder en Passant noch nicht. Außerdem schaut er nur ca. 2 Züge in die Zukunft, was auch noch ausbaufähig wäre. Er ist noch lange nicht perfekt, aber ich bin dennoch zufrieden mit meiner Leistung und konnte einiges neues lernen. Insgesamt ist die Entwicklung eines Schachbots in C# eine lohnende Erfahrung, die sowohl das schachspezifische Wissen als auch die Programmierfähigkeiten des Entwicklers verbessern kann.