Minimax algoritme (schaken)

Ik heb voordat ik het derde semester startte, veel interesse gehad in algoritmes. Ik had filmpjes opgezocht van hoe verschillende algoritmes werken. De eerste keer dat ik met dit algoritme in aanraking kwam was in een filmpje over iemand die een schaakapplicatie aan het maken was met een ai (<https://www.youtube.com/watch?v=DZfv0YgLJ2Q>).

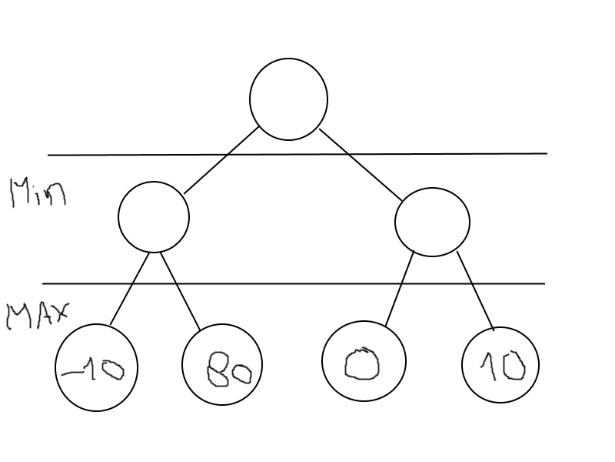
Ik ga voor mijn big idea een schaakapplicatie maken waar mensen online of tegen een bot kunnen spelen. Wanneer ze tegen een bot gaan spelen, gaat de bot zijn zetten berekenen m.b.v. het minimax algoritme.

# Beschrijving

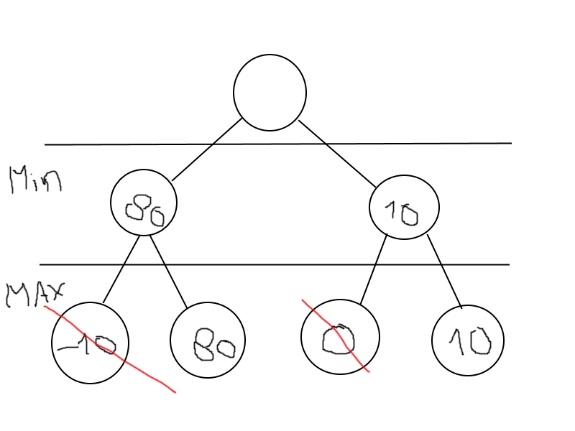
Elk schaakstuk heeft een bepaalde waarde. In dit algoritme ga ik de volgende waardes toepassen:

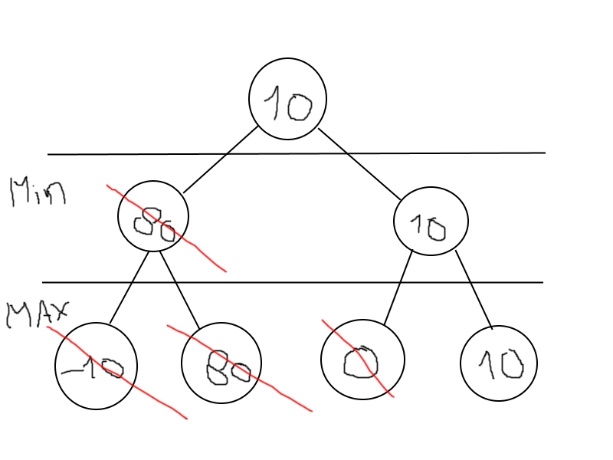
|  |  |  |
| --- | --- | --- |
| Chess piece | White | Black |
| King | 900 | -900 |
| Queen | 90 | -90 |
| Rook | 50 | -50 |
| Horse | 30 | -30 |
| Tower | 30 | -30 |
| Pon | 10 | -10 |

Het minimax algoritme gaat kijken naar wie de maximizer is en wie de minimizer is. Dit houdt dus eigenlijk in wie graag de hoge score wil en wie graag de lage score wil. Op het begin is de totale score van het bord 0. Dit is omdat 8 pionnen, 2 torens, 2 paarden, 2 lopers, 1 koningin en 1 koning aan beide kanten.

Wanneer je, als wit, met de koningin een pion slaat van zwart, verliest zwart 10 punten. Dit ziet er mooi uit voor wit want nu staat de totale score op -10, maar doordat de koningin die pion heeft geslagen kan een andere pion van zwart die koningin slaan wat dus betekent dat de totale score op 80 komt te staan (-10 + 90). Dit is voor wit dus niet het goede want die wil de minimale score bereiken. Dit scenario had wit van te voren al uit kunnen denken. Het minimax algoritme doet dit.

Eerst kijk je als wit welke mogelijke zetten zwart kan zetten wanneer jij een bepaalde zet zet en hoeveel score dit minder wordt voor jou.

Daarna ga je kijken welke zetten zwart daadwerkelijk zou zetten als jij die mogelijke zet zet. Zwart wil natuurlijk dat er zo’n hoog mogelijke waarde van het bord af is.

Hieruit kan wit halen dat hij wel die pion zou kunnen pakken maar dat dat een slechte optie is omdat zwart dan zijn koningin kan slaan dus hij doet een andere zet waardoor zwart zijn pion kan slaan. Dit is een veel betere optie dan de pion zelf slaan.

Dit is natuurlijk een hele kleine versie van het uiteindelijke resultaat. Maar het is wel een makkelijke manier om te begrijpen hoe het algoritme precies werkt. Er zijn veel meer opties in horizontale richting. Dit algoritme denkt 1 beurt vooruit. Uiteindelijk zou ik kunnen implementeren dat hij meerdere beurten vooruit ga denken.

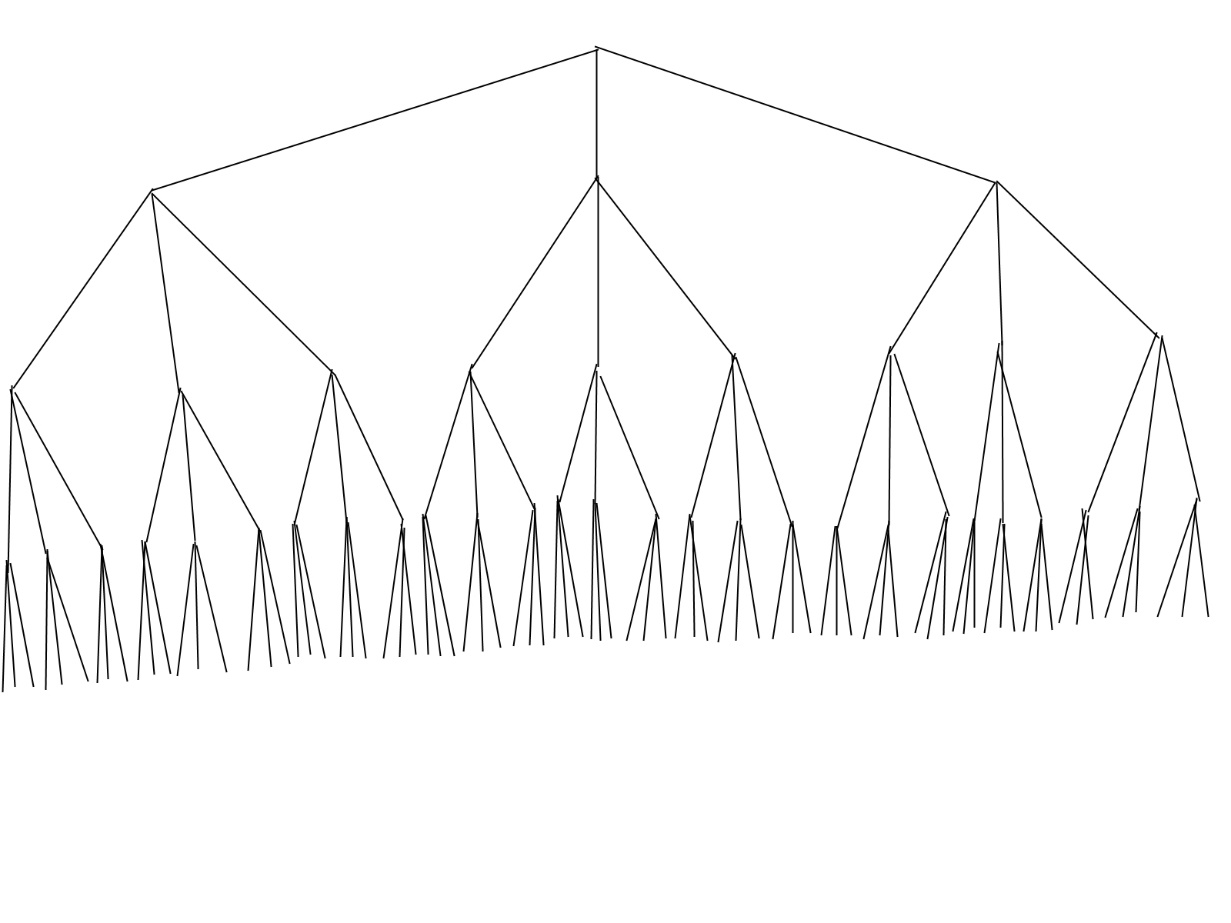
# Complexiteit

Ik heb na zitten denken over de complexiteit van het algoritme. De ruimte complexiteit was niet zo extreem moeilijk. Dit komt omdat je aan het begin (bovenaan) een schaakbord hebt en op dat zelfde schaakbord zetten blijft zetten om te uitproberen. Er worden dus niet meer borden aangemaakt. Dit betekent dat de hoeveelheid geheugen alleen maar verandert om een getalletje op te slaan voor iedere check. Hieruit kan je de big-O halen:

Ruimte: O(n)

Dus voor iedere serie checks moet je 1 getalletje opslaan.

Voor de tijd is het een net iets ander verhaal. Dit ligt er puur aan hoe veel checks je moet maken. Voor iedere stap dat de computer verder denkt wordt exponentieel verhoogd. Dus bij de eerste stap heb je bijvoorbeeld 2, bij de tweede stap heb je 2^2=4, bij de derde stap heb je 2^3=8, bij de vierde stap heb je 2^4=16, etc. Wanneer we het volgende scenario hebt, zal je daar de volgende berekeningen voor hebben:

Dit grid denkt 2x2 (2x wit en 2x zwart) beurten vooruit. Elke beurt kan elke speler 3 verschillende opties doen met 3 verschillende mogelijkheden. Om te berekenen hoeveel stappen er gezet moeten worden moet je voor elke laag het aantal gemiddelde mogelijkheden pakken en daar de macht van op welke laag je zit. Dus dit betekent dat je voor de bovenste laag waar gemiddeld 3 opties per bolletje zijn 3^0=1 hebt. Je hebt dus 1 bolletje op de eerste laag. Voor de tweede doe je 3^1=3. Voor de derde: 3^2=9. Hier zie je hoeveel kruispunten je hebt. Uiteindelijk voor de laatste laag doe je 3^4=81 mogelijkheden hebt. Dus als je het allerbovenste 1ste punt niet meetelt, maar verder alle lagen bij elkaar optelt heb je 4. Daarna het gemiddelde aantal stappen die je per punt kan zetten is in dit geval 3. Dus uiteindelijk is de big O notatie:

O(p^n)

Hierin staat p voor het gemiddeld aantal mogelijkheden (possibilities) en n voor het aantal lagen.