Schaak engine in SWI-Prolog

Inleiding

In dit project heb ik geprobeerd om een schaak engine te implementeren dat gebruik maakt van de eigenschappen van Swi-prolog, deze schaak engine gebruikt een min-max boom om de beste zet te zoeken.

In dit verslag zal ik proberen mijn manier van werken uit te leggen.

Dit project mist wel nog enkele gevraagde elementen voor de volledige opgave, maar is er zo goed als.

Bord voorstelling

Mijn bord wordt voorgesteld als een relatie van relaties. Namelijk als volgt:

```
b(R1, R2, R3, R4, R5, R6, R7, R8)
```

Waarbij `Rx` iedere keer ook een relatie is die de aparte rijen voorstelt. Van structuur lijkt dit veel op ons bord relatie.

```
r(P1, P2, P3, P4, P5, P6, P7, P8)
```

P1..P8 stellen de schaak stukken voor die op die positie staan. een stuk is op zijn eigen ook een relatie.

```
p(Color, Type)
```

Op deze manier van het bord voorstellen is het simpel om op te halen waar een stuk staat. Of gebruik makende van prolog, kunnen we ook simpel opvragen welk stuk op een bepaalde positie staat.

Als een veld op ons schaakbord geen schaakstuk bevat dan als je dit veld zou opvragen krijg je de volgende relatie.

```
p(empty)
```

Algoritme

High Level overzicht

De volgende stappen gebeuren vanuit een high level perspectief:

1. Parsen.
2. Initialisatie van globale waarden. (geen verdere uitleg)
3. Het bord opzetten volgens de voorgaande zetten.
4. De volgende mogelijke zet(ten) genereren.
5. Beste zet kiezen door middel van min-max.
6. Alles uitprinten.

Parsen

Hier ga ik niet al te diep in gaan. Voor de lezer die meer wil weten lees gerust de broncode. Het parsen kan je in een paar stappen uitleggen.

1. Het beurt nummer parsen
2. De daarop volgende 2 halve zetten parsen.
    1. Ale gegeven data bijhouden.
3. Parse als het spel nog niet afgelopen is.

Het bord opzetten

Om het bord op te zetten genereert het algoritme steeds alle mogelijke volgende zetten, uit deze nemen we dan de overeenkomende zet en voeren deze uit op het bord. Dit herhalen we tot we alle voorgaande zetten hebben gedaan. En we dus uiteindelijk het bord hebben zoals het er moet uitzien.

Volgende mogelijke zet(ten) genereren

Zet generatie kan opgesplitst worden in 3 delen alle stuk-plaats combinaties ophalen en voor dit alle mogelijke destinaties (zetten) te genereren samen met alle nodige data (denk promotion, en_passent, aanval). Dit wordt opgesplitst onder directionele(koningin, toren, loper) zetten en niet directionele (de rest) zetten. Dit omdat deze leden van deze groepen onderling eigenschappen delen. Voor meer concretere informatie bekijk je best eens de bron code.

Min-Max

Ik heb mijn min-max algoritme gebaseerd op hetgeen dat gegeven is geweest in de les. Als huidige snoei methode, heb ik momenteel geopteerd voor een strategie die goed zou moeten werken tegen random. Als er zetten zijn die een ander stuk slaan (aanvallen), dan kijken we alleen tussen deze. Want De score van het bord verandert toch alleen maar als er een stuk verdwijnt of in uitzonderlijke gevallen als het schaakmat is. Ik heb nog geen alfa-bèta snoeien kunnen implementeren omdat ik hiervoor niet meer genoeg tijd had. Momenteel zoeken we 2 diep om binnen de tijd limieten te blijven. Ik vermoed dat ik mogelijks ook 3 zou kunnen gebruiken als diepte maar dat mijn gemiddelde snelheid dan te traag zal zijn.

Zet uitprinten

Eenmaal we een zet hebben moeten we deze nog altijd terug geven aan de gebruiker in het pgn formaat en niet in de interne structuur van onze engine die in theorie morgen kan veranderen. Dit doen we door te proberen matchen op de gegeven zet met zo weinig mogelijk data en als we zien dat er maar 1 zet uit alle zetten hiermee patroon-matchen dan geven we dit terug. Als er meerdere zetten met onze zet matchen dan proberen we steeds meer data toe te voegen zoals pgn het vraagt tot we maar 1 matchende zet vinden.

Conclusie

Ik denk dat ik een behoorlijk resultaat heb bereikt ook al heb ik niet alle gevraagde features geïmplementeerd. In het begin kon ik het pgn formaat niet afstaan en om eerlijk te zijn de taal Swi-prolog ook niet. Maar door de manier waarop men kan parsen in prolog bleek dit al bij al nog mee te vallen om pgn te parsen. De taal zelf is ook een beetje op mij gegroeid in de mate dat ik mij er niet meer constant aan stoor, en al vlotter kan denken in de concepten van de taal.

Er zijn duidelijk nog verbeteringen mogelijk aan mijn schaak engine. Ik zou kunnen beginnen met de rest van de gevraagde functionaliteit te implementeren.