Schaakprogramma in prolog

Dries Huybens

14 augustus 2024

0.1 Inleiding

Dit project richt zich op het ontwikkelen van een werkend schaakprogramma en een schaakbot die gebruik maakt van alpha-beta snoeien tot een diepte van drie, geïmplementeerd in Prolog. De algemene structuur van mijn programma is als volgt: het hoofdprogramma bepaalt op basis van de ingevoerde argumenten welk type spel gestart wordt. De module `menu` stelt de spelinstellingen in en vraagt de benodigde opties aan de speler wanneer dat nodig is.

Het programma bevat twee vertalingsmodules: de module `converter` zet SAN-notatie om naar een intern coördinatensysteem, terwijl `converter_to_san` deze interne coördinaten weer omzet naar SAN-notatie. De module `game_handler` beheert de algemene staat van het spel, identificeert correct wie aan zet is en handelt de huidige spelstatus (schaak, schaakmat, normaal...) correct af.

De module `rules` overziet de basisregels van het schaakspel, waarbij de belangrijkste functie `legal_move` bepaalt of een zet legaal is op basis van het huidige bord, de speler, en de spelgeschiedenis. Deze functie laat het stuk op de vertrekcoördinaat controleren of de bestemmingscoördinaat correct is afhankelijk van de regels van dat specifieke stuk. Bijvoorbeeld, voor een loper betekent dit dat de beweging diagonaal moet zijn, het pad vrij moet zijn, en het bestemmingsvakje leeg of bezet door een vijandelijk stuk moet zijn.

`Legal_move` simuleert vervolgens de zet om te controleren of de speler zijn eigen koning schaak zou zetten; als dat het geval is, is de zet ongeldig. Hierbij speelt de module `checks` een belangrijke rol, omdat deze verantwoordelijk is voor het uitvoeren van alle controles, zoals het bepalen van de bordstatus en het vaststellen van schaak, schaakmat, of stalemate. Bijvoorbeeld, om schaakmat te controleren, wordt gekeken of de koning momenteel in gevaar is (schaak). Als dat zo is, worden alle mogelijke zetten gegenereerd en wordt gecontroleerd of de koning nog steeds schaak staat na elke zet. Als er geen enkele zet is die de koning uit schaak haalt, is het schaakmat en is het spel afgelopen.

0.2 Bordvoorstelling

In mijn schaakprogramma wordt het schaakbord intern gerepresenteerd met behulp van een eenvoudig coördinatensysteem, waarbij zowel de rijen als de kolommen worden geïndexeerd van 0 tot 7. Zetten binnen dit systeem worden voorgesteld door het paar coördinaten van het vertrekpunt naar het bestemmingspunt te noteren in de vorm (van RijNummer-KolomNummer, naar RijNummer-KolomNummer). Deze representatie is dus wel wat afwijkend van de SAN notatie dus dit vereist een vertaling.

De conversie van SAN naar het interne coördinatensysteem wordt gerealiseerd door de `converter` module en werkt als volgt: we halen de bestemmingscoördinaat en het stuktype uit de SAN en bepalen vervolgens alle legale zetten die tot die coordinaat kunnen leiden. Door de eventuele extra rij- of kolomaanduiding in de SAN te gebruiken kan het juiste stuk geïsoleerd worden. Nadat het juiste stuk is bepaald, blijft er slechts één geldige vertrek coördinaat over. Deze methode vergemakkelijkt de interne werking van het programma omdat mijn schaakbot niet voortdurend SAN hoeft te vertalen, maar rechtstreeks met precieze coördinaten werkt, wat de efficiëntie verhoogt bij het bepalen van stukken en hun zetten op het bord. De vertaling van en naar SAN wordt slechts eenmaal uitgevoerd, hoewel deze niet bijzonder efficiënt is.

0.3 Algoritme

0.3.1 Introductie

De code van mijn schaakbot en dit algoritme bevind zich voornamelijk in de `chess_bot` module.

0.3.2 Het minimax algoritme

Ik heb gebruik gemaakt van het minimax algoritme [SMB22] met behulp van alpha beta snoeien voor de werking van mijn schaakbot. Dit algoritme simuleert de beurten van beide spelers om de optimale zetten te bepalen. Veronderstel dat er een theoretisch perfecte evaluatiefunctie bestaat die een score toekent aan een bepaalde spelpositie. Een positieve score betekent dat speler 1 zich in een voordelige positie bevind, terwijl een negatieve score aangeeft dat speler 2 in het voordeel is. Hoe groter de absolute waarde van deze score, hoe groter het voordeel voor de betreffende speler.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figuur 1: Pseudocode van het minimax algoritme.

Het minimax-algoritme werkt als volgt: bij elke beurt probeert speler 1 de score te maximaliseren (max-speler), terwijl speler 2 bij zijn beurt de score probeert te minimaliseren (min-speler). Het algoritme doorzoekt de beslissingsboom tot een bepaalde diepte, waarbij het beurtelings de zetten van beide spelers simuleert. In mijn code wordt een diepte van drie gebruikt. Het algoritme analyseert alle mogelijke zetten tot deze diepte en kiest uiteindelijk de zet die resulteert in de maximale score (absolute waarde) voor de bot.

Alpha-beta snoeien is een optimalisatie van het minimax-algoritme, die de efficiëntie verhoogt door onnodige berekeningen te vermijden. Dit wordt bereikt door het gebruik van twee waarden, alpha en beta, die respectievelijk de hoogste score vertegenwoordigen die de max-speler kan behalen en de laagste score die de min-speler kan behalen. Tijdens dit verloop door de beslissingsboom worden takken die geen invloed kunnen hebben op de uiteindelijke beslissing overgeslagen. Dit gebeurt wanneer een deel van de boom een slechter resultaat geeft dan een eerder onderzochte optie waardoor deze score dus definitief al niet meer de beste zal zijn en hierop verder gaan irrelevant wordt.

Deze optimalisatie zorgt ervoor dat het algoritme minder toppen hoeft te bekijken, wat de efficientie van het algoritme aanzienlijk verbeterd.

0.3.3 spel evaluatie

De spel evaluatie functie vormt een cruciaal onderdeel van het minimax-algoritme aangezien de kwaliteit van de evaluaties direct de kwaliteit van de beslissingen beïnvloedt. Als de evaluaties niet consistent waardevol zijn zal het algoritme niet in staat zijn de optimale zetten te selecteren. In deze context worden punten toegewezen aan verschillende stuktypes, zoals weergegeven in tabel 1. Witte stukken krijgen positieve scores toegewezen, terwijl zwarte stukken negatieve scores ontvangen. In deze implementatie fungeert wit altijd als de max-speler en zwart als de min-speler.

De basis evaluatie van het spelbord wordt bepaald door de scores van alle stukken op het bord op te tellen [Che23] [Wik21]. In een neutrale positie resulteert dit in een totale score van 0. Daarnaast wordt een positionele bonus toegekend aan stukken die zich op de vier centrale velden bevinden. Deze bonus is vooral relevant tijdens de opening van het spel, omdat controle over het centrum een cruciale

strategische rol speelt. Hierdoor zal het algoritme eerder geneigd zijn om openingszetten zoals d4 of e4 te spelen in plaats van willekeurige zetten aangezien deze bijdragen aan de controle over het centrum. In het schaken is controle over het centrum vaak van groot belang, en veel openingsstrategieën beginnen met dergelijke zetten om een sterke positie te garanderen. Hierdoor kan voorkomen worden dat de speler vroeg in het spel in een nadelige positie terechtkomt. Verder word de score ook oneindig hoog als de koning schaakmat staat aangezien dit logischerwijze altijd de best mogelijke zet is.

In de variant "King of the Hill"krijgt de koning een zeer hoge score als hij zich in het midden van het bord bevindt, aangezien dit de belangrijkste factor is zolang de koning niet schaak staat.

Stuk	**Symbool**	**Waarde**
Pion	P, p	1
Paard	N, n	3
Loper	B, b	3
Toren	R, r	5
Koningin	Q, q	9
Koning	K, k	0

Tabel 1: Waarden van schaakstukken

0.4 Interactieve modus

Mijn interactieve modus werkt als volgt; Als enkel het argument 'GAME' meegegeven word en geen bestand dan creer ik een nieuw spel vanaf nul waarbij ik de speler vraag het volgende te selecteren, spelvariant (classic of king of the hill), kleur van de speler (wit of zwart) en naam van de speler. Dit doet hij door middel van 1 of 2 in te typen gevolgd door enter of de gewenste naam. Als er enkel op enter geduwd word zonder iets mee te geven worden de standaarwaarden ingevuld (classic, wit, 'Player').

Als er wel een pgn bestand meegegeven word samen met het GAME argument zal ik dit inladen, de bestaande tags in het bestand extraheren en deze correct instellen, voor ontbrekende waarden worden de standaardwaarden ingevuld.

Als de beurt aan de speler is en inplaats van een zet in te typen hij save, quit of resign typt zal het spel respectievelijk opgeslagen en afgesloten of afgesloten zonder op te slaan worden. Dit opslaan gebeurt door de huidige instellingen in de vorm van tags naar het pgn bestand te schrijven, de computer speler zal de naam 'ai' krijgen zodat wanneer dit bestand terug ingeladen word we absoluut zeker zijn welke kleur de computer hoort te zijn. Verder wordt in klassieke SAN notatie elke uitgevoerde zet weggeschreven naar het pgn bestand en is dit ook compatibel met chess.com en andere schaakprogrammas die gebruik maken van pgn bestanden. Als de speler een spel gestart heeft met een pgn bestand als argument zal dit bestand overschreven worden, anders word er gevraagd waar dit bestand moet worden opgeslaan.

0.5 Testen

Mijn plunit testen zijn te vinden onder het testmapje/mytests. Hier zal U vier bestanden vinden, testen voor bewegingen, regels, mijn SAN vertaler en checks die de status van het spel zal nagaan (bijvoorbeeld of het schaakmat is).

Bibliografie

[Che23] Chessify. Behind the numbers: Understanding chess engine evaluations, 2023. Retrieved from Chessify.

[SMB22] Sumit Shevtekar, Mugdha Malpe, and Mohammed Bhaila. Analysis of game tree search algorithms using minimax algorithm and alpha-beta pruning. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pages 328–333, 11 2022.

[Wik21] Chessprogramming Wiki. Simplified evaluation function, 2021. Retrieved from Chessprogramming Wiki.