

**Projectsamenvatting: vier op een rij AI**

**Probleembeschrijving**

Casper de Vries vind het leuk om op ze tijd vier op een rij te spelen. Nu met de corona is het lastig om iemand te vinden om mee te spelen. Dus wilt hij graag dat er een applicatie gemaakt word om vier op een rij tegen te spelen, die intelligente zetten kan maken. De ai in de applicatie moet meerdere stappen vooruit kunnen denken om het zo concurrerend mogelijk te maken om tegen te spelen en, mag de AI niet te lang doen over een zet om de gebruiker een betere speel ervaring te geven. Mogelijk kan er ook vier op een rij gespeelt kunnen worden in de applicatie zelf. zodat Casper in ze eentje vier op een rij kan spelen voor als er niemand anders is.

**Eisen**

* De AI moet minstens 4 beurten diep zoeken naar de beste zet.
* Gemiddeld mens moer het een uitdaging vinden om van te winnen.
* De Ai mag niet langer dan 5 seconden doen over het zoeken inclusief het plaatsen van de steen.
* Optie om de moeilijkheid te aanpassen.
* Applicatie moet simpel te gebruiken zijn.

**Algoritme**

De AI werkt met een minmax-algoritme met Alpha-Beta pruning. In een minmax-algoritme word voor een bepaalt aantal beurten doorgespeeld. Alle mogelijke posities waar het bord in kan zijn na die beurten, worden beoordeeld met een score. Voor de AI maximaliseert de score, en voor de speler zijn beurt minimaliseert de score. Zo probeert de AI te voorspellen wat er zal gebeuren in de komende beurten en zijn kans vergroten om te winnen. Alpha-Beta pruning verkleind het aantal zoekpaden die volledig doorzocht worden door, bijvoorbeeld als je een score weet die beter is dan een andere scores, is er geen rede om zetten te ontdekken met die lagere scores.

Verder is de volgorde waarin de kolommen doorzocht worden van middelste naar buiten. Dit is omdat de middelste kolommen meestal de beste scores halen, en kan je optimaal gebruik maken van Alpha-Beta pruning dus bespaard dat veel zoektijd.

Posities worden opgeslagen in een numpy array. 0 = een lege plek, 1 = de speler, 2 = de AI. Om het plaatsen van stenen snel te laten verlopen worden ze bovenin het bord geplaatst. Voor het visualiseren word die ondersteboven gedraaid.

Zoals in bron:

Medad, N. (2020). Roboticsproject.com. connect four robot. Connect 4 algorithm, setup functions.

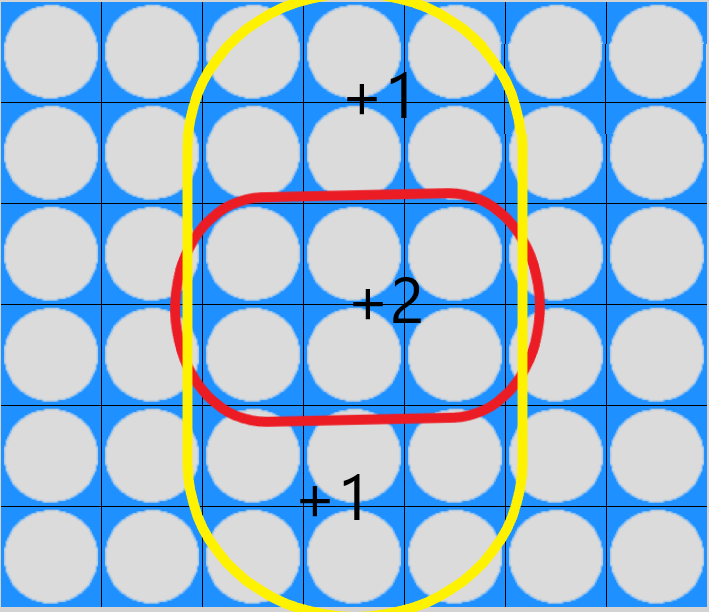
https://roboticsproject.readthedocs.io/en/latest/ConnectFourAlgorithm.html

Bij het zoeken worden de posities gescored door alle (69) vier op een rij mogelijkheden te doorzoeken. Iedere mogelijkheid word gescored volgens:

* 3 stenen van eigen speler en 1 lege = score += 5
* 2 stenen van eigen speler en 2 lege = score += 2
* 3 stenen van de tegenstander en 1 lege = score -= 4

Dit stimuleert de AI om vier stenen op een rij te leggen en het te voorkomen dat de tegenstander mogelijk kan winnen.

Ook krijgen de positie punten voor de hoeveelheid stenen in het middel van het bord(score += 2) en middelste kolommen(score += 1). Voor bijvoorbeeld een (6, 7) bord:



Dit is gedaan omdat stenen in het midden van het bord meer mogelijkheden creëren tegenover kolommen aan de zijkant.

Als er in een positie winst is krijgt afhankelijk van de winnende partij (bot 99999) of (mens-99999) als score. Op die manier worden positie waarin gewonnen word door de bot altijd gekozen boven andere, en als de mens wint worden die positie nooit gekozen.

Er is ook overwogen om bij posities met een gelijken score een random te kiezen, maar dat resulteerde in dat de AI aan het begin zijn stenen aan de zijkant ging leggen.

**Algoritme testen**

Het algoritme is getest op dit bestand: (/project/test\_data/Test\_L1\_R3.csv). in dit bestand staan de posities die het moeilijkst zijn voor de AI om te verwerken. De data komt van:

Pascal, P. (2019). Gamesolver.org. Solving connect 4: how to build a perfect AI. Part 2. Connect 4 solver benchmarking.

<http://blog.gamesolver.org/solving-connect-four/02-test-protocol/>

de zoek dieptes die gebruikt zijn: 4, 5, 6, 7.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| diepte | 4 | 5 | 6 | 7 |
| Tijd in sec | 0.175 | 0.765 | 2.167 | 8.938 |

Het algoritmen had sneller kunnen werken door in plaats van een numpy array een Bit board te gebruiken. Dit bespaard de data die gebruikt word enorm en had er in dezelfde tijd 2 of 3 beurten dieper kunnen zoeken.

Maar over het algemeen presteert het algoritme goed. Bij een diepte van 4 is het al een redelijke uitdaging, en met hogere diepte word het algoritmen niet enorm beter dus bereikt dit systeem bijna zijn limit. Uit het testen blijkt de AI vaker te winnen dan niet.

Een minpunt aan de AI is dat aan het eind van een spel kan soms het minmax-algoritme een kolom terug geven die vol zit. Ondanks dat die volle kolom niet in de lijst met speelbare kolommen.