Ghent University

Logisch Programmeren

Verslag Con-tac-tix project

Jan-Pieter Baert

Academiejaar 2019-2020

Inhoudsopgave

Jan-Pieter Baert
Bachelor of Sciences in Informatics
Stamnummer: 01703178

1 Inleiding

Dit verslag is een bespreking van de AI geschreven voor het spel Con-tac-tix.

1.1 Korte speluitleg

Het spel con-tac-tix, soms ook wel hex genoemd is een spel gespeeld op een hexagonale 'rechthoek' bord, met arbitraire grootte. De bedoeling van het spel is een pad te creëren van boven naar onder voor speler 1 (en links naar rechts voor speler 2), de spelers mogen beurt om beurt een tegel leggen en het spel stop wanneer een speler zijn twee zijkanten verbonden heeft.

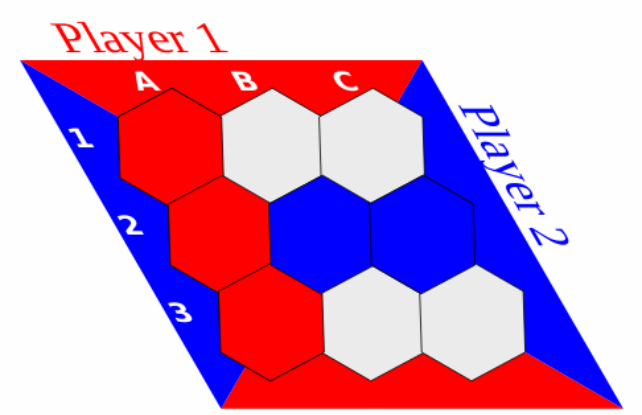
2 Interne bordvoorstelling

We stellen ons bord voor als een feit dat de volgende elementen bevat:

1. De grootte van het veld, opgeslagen als het paar `X/Y`
2. Het kleur van de speler die aan zet is
3. De kleuren van beide spelers, opgeslagen als het paar `Player1/Player2`
4. De status van het spel (bezig of gewonnen door één van de spelers), dit wordt voorgesteld als een getal, -100 als speler 1 gewonnen heeft en 100 als speler 2 gewonnen heeft, alle waarden ertussen zijn spelen die nog bezig zijn.
5. Een lijst van tegels die het bord voorstellen.
   Een tegel stellen we voor als een feit dat de volgende elementen bevat:
   - De coordinaten, opgeslagen als het paar `X/Y`
   - Het kleur van de tegel

2.1 Voorbeeld

`board(3/3,blue,red/blue,-100,[tile(0/0,red),tile(0/1,red),tile(0/2,red),tile(1/1,blue),tile(2/1,blue)])`
stelt het bord in figuur 1 voor.



Figuur 1: Een 3x3 bord van con-tac-tix

3 Algoritme

Het gebruikte algoritme is als volgt:

- Genereer de eerste laag opvolgers van het huidige bord, als hier een winnend bord inzit wordt deze genomen.
- Anders, gebruik een minimax boom om te bepalen wat de beste volgende bord is. De minimax boom wordt gegenereerd gebruikmakende van de heursitiek uitgelegd in 3.3.

3.1 Spelstatus bepalen

De manier om te bepalen of we een winnend spel hebben is gebasseerd op het *floodfill* algoritme. We nemen het coordinaat `-1/-1` als beginpunt en we zoeken een weg naar het coordinaat `X/Y` waarbij `X` de hoogte van het speelveld is en `Y` de breedte van het speelveld. We doen het floodfill algoritme met enkel de coordinaten van de tegels in het kleur van de speler waarvoor we nagaan of deze gewonnen is, dit is inclusief de coordinaten op de randen van het speelveld die die speler probeert te verbinden.

3.2 Minimax

Voor minimax gebruiken we de volgende interface `minimax(Pos, BestNextPos, Val, Depth)`, hier is `Pos` de huidige bord, `BestNextPos` het meest optimale opvolgende bord, `Val` de waarde van de meest optimale direct opvolgende bord (namelijk `BestNextPos`) en `Depth` de maximale diepte tot waar de minimax boom gegenereerd wordt. [1]

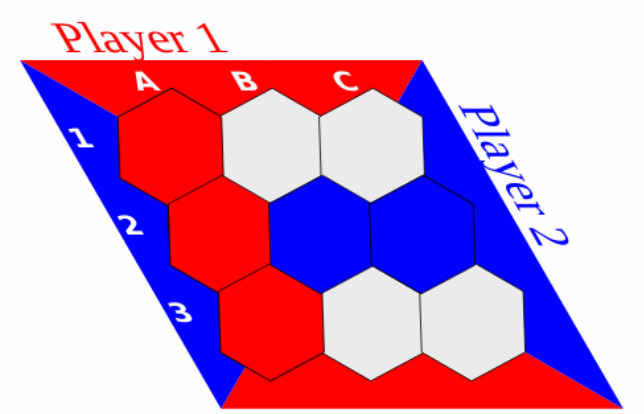
Het minimax algoritme zal dus van het beginnende bord alle opvolgers genereren tot een maximale diepte van `Depth` en dan evalueren, gebruikmakende van waarden van de heuristiek, welke directe opvolger de meest optimale is en welke waarde deze heeft.

[1] Deze minimax interface is gebaseerd op de minimax vanuit de les logisch programmeren (de code van hoorcollege 7)

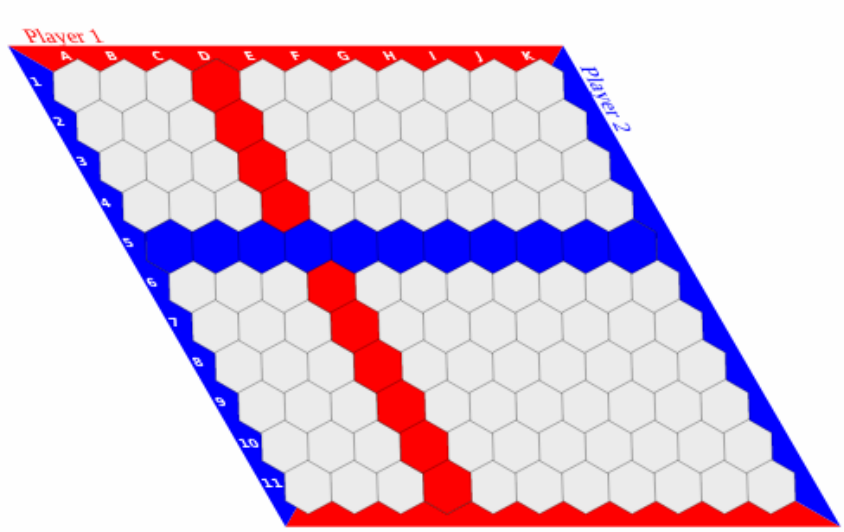
3.3 Gebruikte heuristiek

De heuristiek die we gebruiken is niet echt complex, het is namelijk het verschil tussen `P2` en `P1` waar `P1` het aantal rijen is dat speler 1 bezet heeft en `P2` het aantal kolommen dat speler 2 bezet heeft. Hieruit concluderen we dat speler 1 voor een zo hoog mogelijke heuristiek waarde gaat en speler 2 voor een zo laag mogelijke heuristiek waarde gaat, dit is ideaal om in een minimax boom te gebruiken. Hieronder zullen we een aantal voorbeelden geven om aan te tonen hoe de heuristiek werkt.

3.3.1 Voorbeelden



Figuur 2: Bord met heuristiek -1 (2-3)



Figuur 3: Bord met heuristiek 1 (11-10)

3.4 Versnelling door $\alpha - \beta$ snoeien

Als versnelling hebben we $\alpha - \beta$ snoeien toegevoegd aan de minimax boom, hiervoor hebben we gebruik gemaakt van dynamische feiten in prolog om de waarden voor α en β aan te passen.

Tabel 1 geeft een vergelijking van de uitvoeringstijd van de AI, op een leeg bord waar de hoogte en breedte dezelfde is, we zien duidelijk dat $\alpha - \beta$ snoeien de AI enorm versnelt, hierdoor kunnen we ook de maximale diepte op -1 zetten om de AI tot ieder blad te laten gaan zonder dat dit snelheidsproblemen oplevert.

Grootte	Uitvoeringstijd zonder	Uitvoeringstijd met
2	0.198s	0.201s
3	23.116s	0.213s
4	2500s	0.200s

Tabel 1: Vergelijking van uitvoeringstijd zonder en met $\alpha - \beta$ snoeien

4 Conclusie

Het algoritme samen met het $\alpha - \beta$ snoeien zijn goed geïmplementeerd, maar een meer uitgebreide heuristiek zou helpen om deze AI beter te maken. Verder zijn er zeker nog verbeteringen mogelijk bij het genereren van opvolgers van een bord en bij het nagaan of een spel gewonnen is, want deze nemen nu veel tijd beslag.