Monte Carlo Tree Search op Dou Dizhu

Project Artificiële Intelligentie

Sheng Tao Tian en Jahid Chetti

Academiejaar 2022-2023

Bachelor of Science in de informatica
Universiteit Gent

Inhoudsopgave

1 Inleiding

Turn-based games zijn een erg populair medium om AI-gebaseerde methoden op toe te passen. Enkele voorbeelden van zulke turn-based games zijn schaken en Go, waarvoor er ook zeer goede algoritmes bestaan zoals Stockfish [1] voor schaken en AlphaGo [2] voor Go. Een subcategorie van zulke games zijn kaartspellen, meer specifiek zullen wij het hebben over het Chinese kaartspel Dou Dizhu [3]. Er zijn al verschillende onderzoeken gedaan naar de beste manier om dit spel met behulp van AI te spelen, waaronder met Deep Monte-Carlo, Deep Q-Learning [1] en Combinational Q-Learning [2]. Het is echter opmerkzaam dat al deze methoden gebruik maken van deep learning. Dit valt te verklaren doordat Dou Dizhu een relatief complex spel is op vlak van kaartspellen implementeren met AI; er is een heel grote action space ($\sim$ 27 000 mogelijke zetten) en er zijn meerdere spelers bij betrokken waarbij het de bedoeling is dat je als speler zowel kan samenwerken als tegenspelen met de andere spelers.

Wat wij echter willen proberen is een manier vinden om toch aanvaardbare resultaten te krijgen voor een AI van Dou Dizhu zonder gebruik te maken van deep learning. Een mogelijke reden waarom dit wenselijk zou kunnen zijn, is bijvoorbeeld wanneer je geen toegang hebt tot de nodige resources om zo een neuraal netwerk voldoende te trainen. Een belangrijkere reden is echter dat er tot op de dag van vandaag nog steeds geen deep learning algoritmen bestaan dat Dou Dizhu op (hoog) menselijk niveau kunnen spelen. Ook nieuw bedachte (Deep Monte-Carlo, Combinational Q-Learning) algoritmen ondervinden hetzelfde probleem, dus kan het interessant zijn om naar een manier te zoeken waarbij er geen deep learning is vereist zodat we zelf kunnen bijsturen waar mogelijk, in tegenstelling tot deep learning algoritmen waarbij het lastiger kan zijn om bepaald gedrag te forceren. Daarnaast zijn deep learning algoritmen vaak ook lastiger om te implementeren, zeker voor Dou Dizhu. Hiertoe hebben wij besloten om met Monte Carlo Tree Search (MCTS) te werken als basis voor onze agent om het spel Dou Dizhu mee te spelen.

1.1 Dou Dizhu

Dou Dizhu (vertaald naar vechten tegen de landheer/ Fight the Landlord) is een van de populairste kaartspellen in China en wordt veel gespeeld op zowel online platforms als in het echt, al dan niet voor geld. Het wordt gespeeld met een standaard deck kaarten inclusief beide jokers (54 kaarten in totaal), waarbij de kleuren van de kaarten niet uitmaken. Het wordt altijd gespeeld met exact drie spelers waarvan één landheer en twee boeren. Als landheer is het de bedoeling dat je wint van beide boeren, maar als boer is het voldoende dat een van de twee boeren wint. Indien er met echt geld gespeeld wordt is de winst van de landheer natuurlijk groter dan die van de boeren. Winnen doe je door al je kaarten kwijt te spelen, de eerste persoon die dit doet wint en eindigt hiermee het spel. Iedere speler krijgt in het begin van het spel 17 kaarten. Hierna volgt een korte inzetronde waar er geboden wordt (dit wordt buiten beschouwing gehouden voor ons project) om landheer te zijn. De landheer krijgt vervolgens drie extra kaarten die zichtbaar zijn voor heel de tafel en mag het spel starten.

Het spel wordt gespeeld in rondes waarbij er combinaties aan kaarten worden gespeeld die voor eenzelfde ronde altijd hetzelfde moeten blijven. De persoon die een ronde begint bepaalt de combinatie voor deze ronde en dan moeten er om de beurt strikt hogere kaarten gespeeld worden van dezelfde combinatie totdat iedereen past. De persoon die dan de hoogste kaarten heeft gespeeld mag een nieuwe ronde beginnen. Het is verder niet verplicht om kaarten te spelen als je dat wel kan maar niet zou willen, en indien je past maar het terug aan jou is in dezelfde ronde mag je nog steeds kaarten uitkomen en ben je niet verplicht om opnieuw te passen.

De volgende tabel geeft een basisoverzicht van de mogelijke combinaties die gespeeld kunnen worden:

Type	Kicker[4]	Beschrijving	Voorbeeld
Solo	Geen	Een kaart die op zichzelf wordt gespeeld.	3
Paar	Geen	Een paar aan kaarten van dezelfde waarde.	3-3
Triplet	Geen	Een triplet aan kaarten van dezelfde waarde.	3-3-3
Triplet	Solo	Een triplet met 1 kaart als kicker.	3-3-3 + 4
Triplet	Paar	Een triplet met 1 paar als kicker.	3-3-3 + 4-4
Quad	2 solo's	Een quad met 2 solo's als kicker.	3-3-3-3 + 4-5
Quad	2 paren	Een quad met 2 paren als kicker.	3-3-3-3 + 4-4-5-5
Bom	Geen	Een speciaal type quad dat op elke combinatie gespeeld mag worden, ongeacht wat er het laatst gespeeld is.	3-3-3-3
Raket	Geen	De hoogste bom in het spel bestaande uit de 2 jokers.	Beide jokers

Tabel 1: Basiscombinaties in Dou Dizhu

[1] https://www.chess.com/terms/stockfish-chess-engine
[2] https://en.wikipedia.org/wiki/AlphaGo
[3] https://en.wikipedia.org/wiki/Dou_dizhu

Enkele opmerkingen hierbij zijn dat:

- De nummering van de kaarten gaat van 3 tot A zoals bij de meeste kaartspellen, maar de 2 is groter dan de A, gevolgd door de zwarte en rode jokers respectievelijk.
- In de meeste varianten van het spel mogen de 2's en jokers niet gebruikt worden als kicker, en mogen deze ook niet gebruikt worden in straten of dergelijke combinaties waarbij je opeenvolgende kaarten nodig hebt. Wij zullen dit echter wel toestaan.
- Zoals hierboven vermeld is een bom een soort catch-all dat op alles gespeeld mag worden, maar het is belangrijk om een distinctie te maken tussen een bom en een quad met kickers. Indien je een quad speelt met kickers bij is het immers geen bom meer, dus als iemand bijvoorbeeld A-A-A-A + 3-4 speelt kan je er de bom van 5-5-5-5 op spelen als hogere combinatie. Verder is het niet toegestaan om een raket met kickers te spelen.
- Hoewel de waarde van de kicker niet uitmaakt (zo lang deze verschilt in waarde van de hoofdcombinatie), is iedereen wel verplicht om te volgen met hetzelfde type kicker. Als er bijvoorbeeld 3-3-3 + 4-4 wordt uitgekomen, mag je enkel triplets spelen met een paar als kickers.

Buiten deze basiscombinaties bestaan er nog combinaties die gespeeld kunnen worden, maar dit zijn allemaal uitbreidingen op de basiscombinaties waarbij er kaarten met opeenvolgende waardes worden gecombineerd. De condities hiervoor zijn als volgt:

Type	Minimum hoeveelheid	Opmerkingen
Solo	5 solo kaarten	Geen.
Paar	3 paren	Geen.
Triplet	2 triplets	De kickers die je gebruikt moeten van hetzelfde type zijn, dus in het geval dat je twee triplets speelt ben je verplicht om ofwel twee verschillende kaarten als kickers te gebruiken of twee verschillen paren. Het is daarnaast enkel toegestaan om ofwel geen kickers te gebruiken, ofwel alle kickers die normaal horen bij deze triplets. Deze zet wordt ook wel een vliegtuig genoemd.
Quad	2 quads	Idem als bij een vliegtuig. Deze zet wordt een ruimteschip genoemd.

Tabel 2: Uitbreidingen op de basiscombinaties in Dou Dizhu

Hoewel het op eerste zicht misschien raar kon lijken dat de landheer alleen speelt en daarbij zelfs extra kaarten krijgt, zou het nu duidelijk moeten zijn waarom die extra kaarten in veel gevallen een voordeel is. Soms mis je immers een of twee kaarten om een zeer sterke of lange combinatie te hebben, dus kan het het waard zijn om de gok te nemen met de drie extra kaarten die je krijgt, zeker als de andere kaarten in je hand al sterk zijn.

[4]Een extra kaart die geen invloed heeft op de waarde van een combinatie

2 Methodiek

Wij hebben ervoor gekozen om onze agent te implementeren met behulp van het Monte Carlo Tree Search (MCTS) algoritme. Dit algoritme vereist echter dat we beschikken over alle informatie van de huidige staat van een spel, wat hier niet het geval is omdat we de kaarten van de andere spelers niet weten. Om toch met MCTS te kunnen werken hebben we dus enkele zaken aan het basis algoritme moeten aanpassen zodat het overweg kan gaan met imperfecte informatie. Voordat we hiermee beginnen zullen we echter eerst een kort overzicht geven over hoe MCTS normaal werkt in het geval dat we wel beschikken over perfecte informatie.

2.1 Gewone MCTS

Voor elke zet wordt er vanuit de huidige staat van het spel een boom opgesteld. Elke top in deze boom stelt een zet voor die gedaan wordt vanuit de staat van de ouder top. De wortel is dus de staat waar men zich bevindt als de agent aan zet is. Elke top houdt naast de zet ook twee waarden bij: hoeveel keer hij bezocht geweest is en een bepaalde score. Deze boom wordt gemaakt door voor een aantal iteraties bepaalde stappen uit te voeren:

1. Selectie: In deze stap wordt er gebruikt gemaakt van de upper confidence bound[5] (UCT) functie om een blad te selecteren.

$$UCT_j = X_k + C * \sqrt{\frac{ln(n)}{n_j}}$$

   Waarbij X_j de gemiddelde reward is van van de top, n het aantal keer is dat de ouder van de top werd bezocht, n_j het aantal keer dat de ouder van de top werd bezocht, en C een constante is die in principe vrij gekozen mag worden maar meestal wordt ingesteld op $\sqrt{2}$.

   In de eerste iteratie zal de geselecteerde top uiteraard gewoon de wortel zijn. De UCT functie zal ervoor zorgen dat er een goede afweging gemaakt wordt tussen exploratie en exploitatie. Zo worden alle mogelijke zetten wel degelijk overwogen maar gaan zetten die goed lijken te zijn meer gesimuleerd worden.

2. Expansie: Nadat een blad geselecteerd wordt krijgt dat blad alle mogelijke zetten vanuit deze staat als kinderen. Een van deze kinderen wordt dan willekeurig geselecteerd.

3. Rollout: Nu wordt het spel vanuit de bekomen staat gesimuleerd voor een aantal zetten of tot het spel gedaan is. Met de staat na deze simulatie berekenen we een bepaalde score.

4. Back-propagation: Nu we een score hebben gaan we terugkeren naar de wortel. De score van elke top op het pad krijgt nu de bekomen score erbij opgeteld. Het aantal bezoeken van elke top op het pad wordt met 1 verhoogd.

Nadat deze stappen voor een aantal iteraties zijn uitgevoerd kiezen we als volgende zet het kind met de hoogste UCT waarde van de (huidige) wortel.

2.2 Onze MCTS

Zoals eerder al vermeld moeten we om MCTS toe te passen eerst alle informatie van het spel hebben. We kunnen namelijk geen simulaties doen als we niet weten welke kaarten de tegenstanders hebben. Om dit op te lossen gaan we de tegenstanders willekeurige kaarten geven die nog in spel zijn. Echter zullen we met deze manier van werken nog steeds geen volledig representatieve voorstelling hebben van de kaarten van de tegenstanders. Daarom gaan we niet één, maar meerdere bomen opstellen, elk met een nieuwe random verdeling van de kaarten. Op deze manier kunnen we de willekeurigheid in zekere mate onder controle houden.

Eens de kaarten verdeeld zijn verloopt MCTS verder normaal. Bij de selectie stap wordt er in toppen waar de zet aan een tegenstander is een kind gekozen met een zo laag mogelijke UCT waarde, omdat de tegenstanders uiteraard zelf ook proberen winnen. Voor de simulatie in de rollout stap gebruiken we de minimum agent, die altijd de laagst mogelijk zet kiest, als default agent.

Nadat de verschillende bomen opgesteld zijn worden de waarden van de overeenkomstige kinderen van de wortels opgeteld. We hebben echter ondervonden dat voor het selecteren van de uiteindelijke zet UCT niet altijd goed werkt. In plaats daarvan wordt deze gekozen door de waarde van elk kind te delen door zijn aantal bezoeken en daaruit de hoogste te nemen. Dit zorgt ervoor dat de agent meestal een goede zet kiest in tegenstelling tot UCT waar de agent vaker een (heel) slechte zet kiest.

[5] `https://www.chessprogramming.org/UCT`

2.3 Slimme herverdeling van de kaarten

Hoewel het random verdelen van de resterende kaarten aan de andere spelers een mogelijke optie is die perfect zal werken, is het mogelijk om hier een verbetering op te maken. Het is immers zo dat we informatie kunnen halen over de kaarten van de andere spelers door te kijken naar welke combinaties ze hebben uitgekomen. Om te illustreren wat we precies hebben gedaan hiervoor zullen we onszelf speler A noemen, en de andere spelers geven we de naam B en C. Concreet bekijken we de volgende gevallen:

- Wanneer B een paar uitkomt (zowel als hoofdcombinatie als als kickers) kunnen we ervan uitgaan dat hij geen triplet of quad heeft van dat type, aangezien het slimmer geweest zou zijn om dat paar te sparen en in plaats daarvan de triplet of bom uit te komen. De enige uitzondering die we hierop maken is wanneer het gaat over een paar van K of hoger. In dat geval sluiten we het niet uit dat B ook een triplet kan hebben omdat het mogelijk is dat hij door dit paar te spelen de ronde wou winnen. Echter wordt de mogelijkheid op een bom nog steeds uitgesloten omdat dit in bijna alle gevallen nog steeds beter is om te hebben dan het op te splitsen in twee paren.
- Dezelfde redenering kunnen we toepassen op een triplet. Hier wordt dus uitgesloten dat speler B een bom heeft.
- Wanneer B een enkele kaart als kicker gebruikt kunnen we ervan uitgaan dat dit de enige kaart is van die specifieke waarde dat B heeft, aangezien het meestal geen goed idee is om de combinatie die je kan maken met die kicker te verbreken.
- Wanneer B een joker speelt, de andere joker nog niet is uitgekomen en A hem niet heeft, gaan we er van uit dat C deze heeft. Het is immers zo dat het meestal een beter idee is om de twee jokers te sparen als raket dan om ze apart uit te komen.

Hier kunnen de rollen van B en C ook omgewisseld worden. Nu hebben we dus een manier om een beredeneerde gok maken over welke kaarten de andere spelers nog hebben.

2.4 Score functie

Op het einde van de rollout stap van MCTS wordt de score functie gebruikt om een waarde te geven van hoe goed de bekomen staat is. Het randgeval van deze functie is als het spel voorbij is. Indien de agent dan gewonnen heeft wordt er een heel hoge score gegeven, indien de agent verloren heeft wordt er een heel lage score gegeven. Als het spel nog niet gedaan is wordt de score berekend door de volgende formule:

$$Score = \sum_{i}^{N} \frac{k_i}{N}$$

Waarbij N het aantal kaarten is in een hand en k_i de waarde van de i-de kaart. Op deze manier berekenen we een soort van gemiddelde waarde van de hand die het leggen van veel en lage kaarten aanmoedigt. Als je wilt volgen op een zet is het ten slotte vaak een goed idee om zo laag mogelijk te volgen zodat je je hoge kaarten bespaart, en wanneer je een kicker gebruikt is het vaak ook een goed idee om deze zo laag mogelijk te houden.

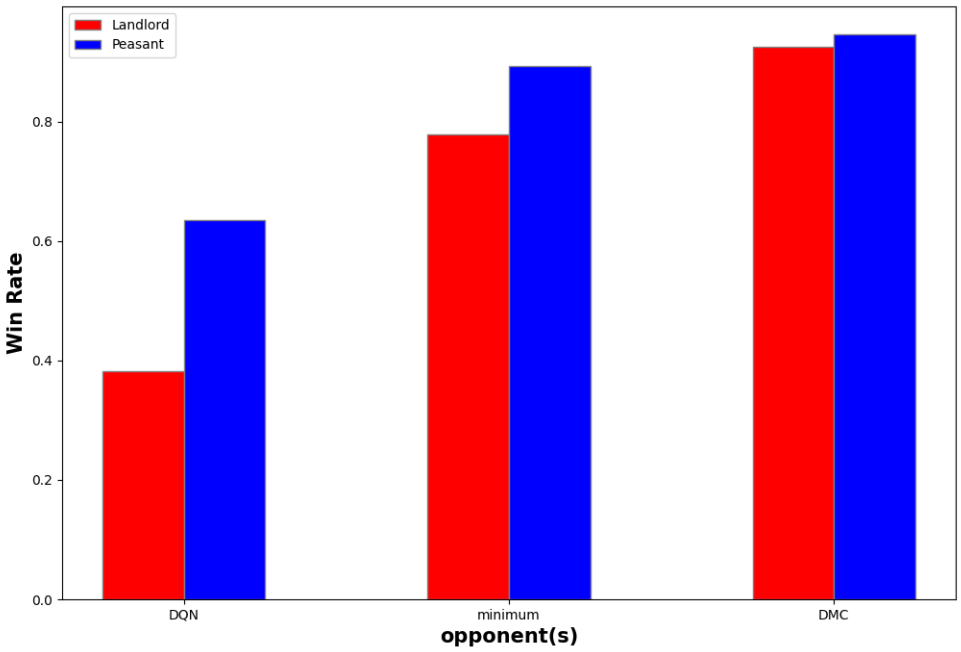
Een probleem dat we echter hiermee ondervonden hebben is dat passen volgens de score functie nu vaak een goede zet zal lijken. Immers zal de score van je hand onveranderd blijven, waardoor er geen incentive is om hoge kaarten te spelen wanneer dat in veel situaties wel nodig zou zijn om voortgang te boeken in het spel. Om dit probleem op te lossen hebben we besloten om passen een score van 0 te geven. Hoewel deze oplossing werkt om het pasprobleem mee te elimineren, zullen we later nog bespreken waarom dit ook een aantal problemen met zich meebrengt.

3 Resultaten

Om te kijken hoe goed onze Monte Carlo agent (MCA) presteert, hebben we het geïmplementeerd met behulp van RLCard[6]. Dit is een toolkit om reinforcement learning mee toe te passen in kaartspellen. Het heeft onder andere Dou Dizhu als een built in environment en is volledig open source waardoor we deze makkelijk konden aanpassen en uitbreiden waar nodig. RLCard heeft ook reeds enkele built in agents die we kunnen gebruiken om de prestatie van MCA mee te meten. Hiertoe beschouwen we de volgende agents:

- Random Agent: speelt een random combinatie
- Minimum Agent: speelt altijd de laagste mogelijk combinatie
- Deep Q-Learning Agent (DQN): maakt gebruik van deep q-learning om Dou Dizhu te leren spelen
- Deep Monte-Carlo Agent (DMC): maakt gebruik van MCTS in combinatie met deep q-learning om de scores van een top te bepalen

De random agent was reeds aanwezig in RLCard en de minimum agent hebben we zelf toegevoegd. Er was ook reeds code aanwezig om modellen te trainen aan de hand van DQN en DMC. Deze hebben wij dan ook met de standaard parameters getraind om onze modellen hiervoor te bekomen. Onderstaande grafiek toont de prestatie van de agents waarmee we onze MCA zullen vergelijken als ze spelen tegen de random agent.



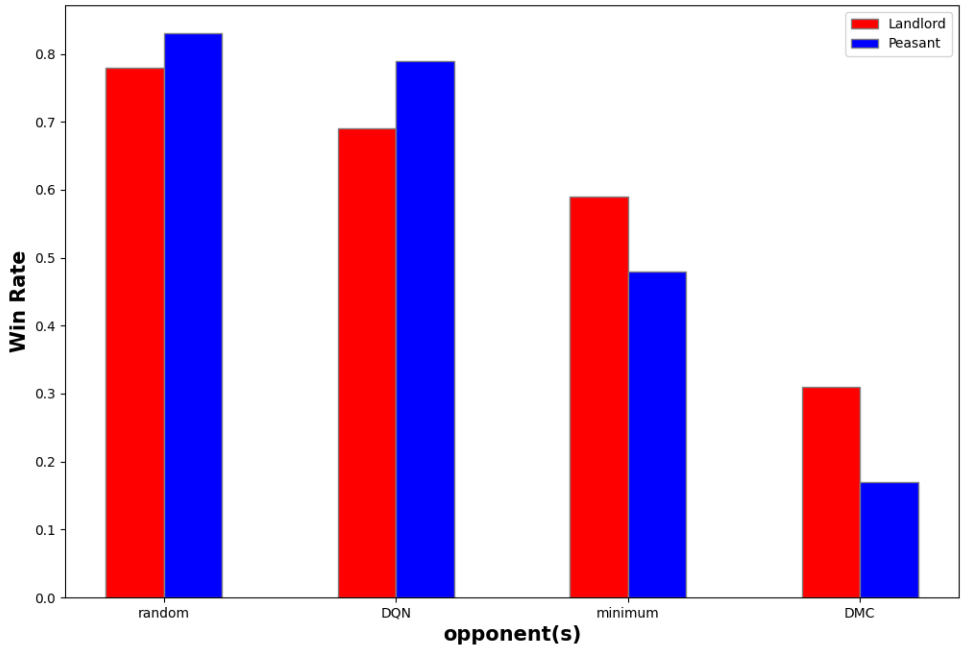
Figuur 1: Winrates tegen de random agent

We zien dat DQN bijzonder slecht presteert. Dit komt vermoedelijk door de heel grote hoeveelheid aan mogelijke acties in Dou Dizhu in combinatie met het feit dat er in DQN pas een score gegeven kan worden als het spel gedaan is wat computationeel erg zwaar is. De minimum agent presteert daarentegen bijzonder goed voor zo een simpel algoritme, terwijl DMC duidelijk de beste agent is die bijna nooit tegen random verliest.

Onderstaande grafiek toont het percentage aan gewonnen spellen van MCA tegen de andere agents, zowel als landheer als als boer. Aangezien de HPC niet beschikbaar was en onze MCA nogal traag speelt hebben we minder spellen kunnen simuleren[7] dan we wilden waardoor de resultaten kunnen afwijken van de eigenlijke winrates. Desalniettemin geeft het ons wel nog steeds een redelijk goed inzicht van de prestatie van MCA.

[6] `https://rlcard.org/`

[7] MCA vs Random: 400, Random vs MCA: 207, MCA vs Min: 400, Min vs MCA: 180, MCA vs DQN: 304, DQN vs MCA: 165, MCA vs DMC: 283, DMC vs MCA: 227



Figuur 2: Winrates van MCA tegen alle andere agents

We zien dat MCA heel goed presteert tegen tegen random en DQN. Tegen de minimum agent heeft MCA meer moeite maar wint nog steeds als landheer meer dan de helft van de spellen. Als boer daalt de winrate van MCA naar ongeveer 50%, maar het is hierbij belangrijk om er rekening mee te houden dat wanneer je twee boeren hebt, ongeacht welke agent ze gebruiken, ze nooit goed kunnen samenwerken dus is het logisch dat de winrate van een landheer vaak hoger is. Hieruit kunnen we dus concluderen dat MCA wel effectief beter kan spelen dan de minimum agent, ook al scheelt het niet veel.

DMC daarentegen is duidelijk beter dan MCA, maar het is wel opmerkelijk dat MCA als landheer een veel hogere winrate heeft dan als boer, namelijk meer dan 30%. We vermoeden dat dit onder andere te maken heeft met een combinatie van twee factoren:

1. De DMC agents zullen elkaar als boer tegenwerken omdat ze elk individueel proberen te winnen.

2. De slimme herverdeling van de kaarten zal bij het spelen tegen DMC beter werken, omdat DMC minder snel goede combinaties zal breken in vergelijking met andere agents. Een DMC agent zal minder geneigd zijn om bijvoorbeeld 6-6 te spelen wanneer hij een bom heeft van 6-6-6-6. Hierdoor zal de slimme verdeling van de kaarten accurater zijn dan tegen bijvoorbeeld DQN en de minimum agent. Dit wil ook zeggen dat wanneer MCA tegen echte mensen speelt, het vaak een beter inzicht kan krijgen over wie welke kaarten heeft dan wanneer het tegen de besproken agents speelt.

4 Mogelijke verbeteringen

Hoewel onze agent nu al redelijk goed kan spelen zijn er nog een aantal verbeteringen die zijn performantie zouden kunnen verhogen. We bespreken hier de belangrijkste.

4.1 Parameters aanpassen

De meest voor de hand liggende manier die we kunnen gebruiken is de waarden van de parameters voor MCTS te verhogen. Voor onze benchmarks hebben we de maximale diepte van rollouts op 3 gehouden, het aantal rollouts op 200 en het aantal bomen op 3. Deze waarden zijn gekozen door een overweging te maken tussen de uitvoertijd van onze testen en nog steeds goede resultaten te krijgen. Doch wanneer er tegen echte mensen gespeeld wordt, kan het in bepaalde situaties aanvaardbaar zijn om wat uitvoertijd op te offeren in ruil voor betere performantie.

4.2 Een betere score functie

Onze score functie zou ook verbeterd kunnen worden, aangezien we enkel kijken naar de waarden van de kaarten in de hand van de agent. Er zijn namelijk veel gevallen waar aantal hoge kaarten hebben niet wenselijk is. Wanneer je bijvoorbeeld enkel hoge kaarten hebt die samen geen goede combinaties vormen, zal je niet kunnen volgen op lage kaarten die goede combinaties vormen.

Daarnaast houdt onze score functie ook geen rekening met het verbreken van (goede) combinaties. Wanneer je bijvoorbeeld een vliegtuig hebt is het geen goed idee om de triplets daarvan als solo uit te komen, terwijl de score functie dit wel als beter zal beschouwen als de waarde van de kaarten in de triplets laag zijn.

Ten slotte is er ook nog een probleem met hoe we passen behandelen in onze score functie, zoals hiervoor al vermeld. Soms willen we bijvoorbeeld passen omdat we anders een hoge combinatie zouden verbreken door te volgen, en het is moeilijk om te bepalen voor welke combinaties te behouden passen het waard is. Daarnaast zal dit ook afhangen van hoe ver je in het spel zit: naar het einde toe zal je sneller (hoge) kaarten willen uitkomen om te voorkomen dat de tegenpartij aan zet kan komen. Verder wordt er in echte spellen ook vaak gepast om als boer de andere boer te laten winnen, en natuurlijk heb je hiervoor ook de juiste timings nodig. Dit alles zorgt ervoor dat passen op een eerlijke manier behandelen in de score functie een moeilijke taak is waarvoor wij geen goede oplossing hebben gevonden.

4.3 Minimum agent gebruiken in het begin

Voor de eerste paar zetten van de agent zijn er meestal heel veel mogelijke opties, zeker als er als landheer begonnen wordt. MCTS kan vaak niet de optimale zet vinden als er veel mogelijke zetten (>60) zijn en we maar 200 rollouts gebruiken. Het is ons opgevallen dat het dan ook niet ongebruikelijk is dat er in de eerste paar zetten suboptimale combinaties worden uitgekomen. Aangezien in het begin van het spel het vaak een goed idee om een zo laag mogelijke combinatie uit te komen om je betere kaarten te sparen voor later, hebben we geprobeerd om als eerste paar zetten de minimum agent te gebruiken om het spel mee te spelen. Hiermee zouden we (zeer) slechte zetten in het begin van het spel kunnen beperken.

In tegenstelling tot onze verwachting maakte dit echter geen merkbaar verschil. We vermoeden dat dit komt omdat de zetten die MCTS gaat kiezen ondanks de vele mogelijkheden vaak toch wel goed genoeg zijn op lange termijn.

4.4 Rollouts en simulatie diepte laten variëren

We weten dat er in het begin van het spel veel meer mogelijke zetten zijn dan naar het einde toe. Het is ook zo dat je in het begin van het spel meer moet focussen op gewoon zoveel mogelijk lage kaarten kwijt te spelen zodat je een goede hand kan overhouden voor de endgame. Dit wil zeggen dat we eigenlijk in dezelfde rekentijd betere resultaten kunnen krijgen door in het begin van het spel het aantal rollouts hoog te houden en de simulatie diepte laag, zodat er vooral wordt gefocust op het kwijtspelen van lage kaarten. Naarmate het spel vordert kunnen we dan het aantal rollouts verlagen en de simulatie diepte groter maken, zodat we een beter idee krijgen over welke zetten op langere termijn goed zijn. We hebben dit echter niet zelf geïmplementeerd omdat het moeilijk is om de juiste verhouding tussen het aantal rollouts en de simulatie diepte te vinden, en het testen hiervan ook zeer tijdsintensief zou zijn als we genoeg mogelijkheden willen uitproberen.

4.5 Parallellisatie

Het is mogelijk om de performantie zowel op vlak van uitvoeringstijd als op vlak van hoe goed onze agent kan spelen te verbeteren door het uitwerken van verschillende Monte Carlo bomen te parallelliseren. Deze bomen zijn onafhankelijk van elkaar en kunnen dus perfect tegelijkertijd uitgewerkt worden, en dan terug samengevoegd

worden. Dit ligt echter buiten de scope van wat we wilden bereiken met onze agent dus hebben we dit niet geïmplementeerd.

5 Als mens spelen tegen AI

We weten nu hoe de agents presteren in vergelijking met elkaar, maar de bedoeling van AI's voor spellen is om er als mens tegen te kunnen spelen. We bespreken hier dus kort wat de ervaring is om als mens tegen MCA en DMC te spelen. We laten random, DQN en de minimum agent buiten beschouwing. De random en DQN agents zijn namelijk te onvoorspelbaar en zijn ook niet in staat om op een aanvaardbaar niveau te spelen, en de minimum agent is juist te voorspelbaar waardoor je het makkelijk kan uitbuiten. Hiertoe hebben we zelf een simpele grafische interface gemaakt om tegen DMC en MCA te kunnen spelen.

Als landheer tegen AI boeren winnen is erg makkelijk. Geen enkele van de agents is namelijk in staat om samen te werken met zijn medeboer. Zelfs tegen DMC is winnen als landheer niet moeilijk, ook voor beginners. Wanneer een boer bijvoorbeeld bijna al zijn kaarten heeft uitgespeeld, kan het vaak een goed idee zijn voor de andere boer om ofwel te passen op zijn kaarten, ofwel de huidige ronde zelf te winnen zodat hij de ronde erna een lage combinatie kan uitkomen waarop de medeboer zijn eigen lage kaarten kan kwijtspelen. Alle agents die we beschouwd hebben weten echter niet wat ze exact moeten doen in zo een situatie en gaan vooral focussen op zelf proberen winnen. Dit kan resulteren in veel situaties waar een anders gegarandeerde win verandert in een verlies.

Als boer winnen tegen een MCA of DMC landheer kan wat moeilijker zijn. De reden hiervoor is dezelfde als wat we hierboven al hebben uitgelegd; je medeboer zal je namelijk vaak proberen tegenwerken. Het is nog steeds goed doenbaar om als boer te winnen tegen een MCA of DMC landheer, maar je bent eigenlijk tegelijk ook tegen je medeboer aan het spelen in plaats van samen tegen de landheer, wat het spel nogal frustrerend kan maken. Dit probleem zal je natuurlijk niet ondervinden als je samenspeelt met iemand anders als boer, maar langs de andere kant zal het ook wel extra moeilijk zijn om te winnen als je met een AI boer speelt tegen een echt persoon als landheer.

6 Conclusie

Zoals uiteindelijk gebleken is uit onze benchmarks kan onze MCA niet op hetzelfde niveau spelen als een deep learning algoritme zoals DMC. Echter is het wel al een positief resultaat dat MCA beter kan spelen dan DQN, alsook een simpele, maar sterke heuristiek zoals de minimum agent. Wanneer er in de praktijk wordt gespeeld zal het verschil tussen MCA en DMC ook minder genuanceerd zijn, omdat ze allebei nog tekortkomingen hebben ten op zichte van menselijke spelers. Verder weten we ook dat MCA tegen echte mensen accurate simulaties kan uitvoeren doordat hij een beter zicht heeft over wie welke kaarten heeft. Dit wil zeggen dat mits enkele aanpassingen aan MCA door de voorheen vermelde verbeteringen al dan niet deels toe te passen, we ervan overtuigd zijn dat de prestatie van MCA op een niet insignificante manier verhoogd kan worden. Het blijft dan wel nog een vraag of het uiteindelijk beter zou kunnen spelen dan DMC, maar we sluiten de mogelijkheid niet uit.

Referenties

[1] Zha Daochen et al. “DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning”. 2021. DOI: 10.48550/arXiv.2106.06135.

[2] You Yang et al. “Combinational Q-Learning for Dou Di Zhu”. 2019. DOI: 10.48550/arXiv.1901.08925.