# Project AI

Brent Matthys - Mats Van Belle

23 december 2022

**Samenvatting**

In dit project wordt een AI ontwikkeld die zichzelf aanleert om het kaartspel mao te spelen. Hierbij zullen we verschillende manieren bekijken en bespreken om dit probleem aan te pakken. Alle code is terug te vinden in onze github repository.

## Inhoudsopgave

# 1 Mao spelregels

Aangezien het kaartspel mao een niet zo bekend kaartspel is zullen we in deze sectie het spel uitleggen.

## 1.1 Algemene regels

Mao is een spel waarbij elke speler initieel 3 kaarten krijgt. Iedere speler heeft als doel om als eerste al zijn kaarten uit te spelen. Het speciale aan het spel is dat er niet over de regels gesproken mag worden. Er wordt dus steeds met een vaste set van regels gespeeld. Sommige mensen kennen de regels, sommige niet. Indien je (sommige/alle) regels niet kent kan je die ontdekken door andere mensen tijdens het spelen te volgen. Je kan dus observeren wanneer welke kaarten door wie gelegd worden en zo zelf de regels ontdekken. Ook mag je als speler zelf proberen leggen. Indien je een fout maakt zal iemand die de regels wel kent je verbeteren, en zul je een strafkaart krijgen. Zo kan je zelf ook de regels ontdekken.

De persoon die deelt bepaalt de regels waarmee men speelt. Vaak blijft men een lange tijd spelen met dezelfde regels, maar het kan dat er met nieuwe regels gespeeld wordt wanneer iemand anders deelt. In dit project zullen wij onze AI trainen op de regels die wij gewoon zijn.

## 1.2 Een beurt

Tijdens een beurt moet een speler 3 zaken uitzoeken om een kaart correct te mogen leggen.

- Wanneer is de speler aan beurt?
- Welke kaarten mogen er gelegd worden op de huidige stapel?
- Welke acties moeten er gebeuren bij het leggen van de kaart?

Hieronder bespreken we deze 3 zaken.

**Wanneer aan beurt** Als speler mag men uiteraard niet altijd leggen, maar moet men uitzoeken wanneer hij aan beurt is. Als een speler legt buiten zijn beurt of niet legt/bij pakt wanneer hij aan beurt is zal de speler bestraft worden en een extra kaart ontvangen. Bij de regels die wij gebruiken mogen de spelers elk om beurt spelen, in de wijzers van de klok. Wanneer er een 10 gespeeld wordt, draait men de speelrichting om.

**Welke kaarten leggen** Als de speler denkt aan beurt te zijn heeft hij 2 opties. Enerzijds kan hij ervoor kiezen om een kaart bij te nemen, dan is zijn beurt voorbij. Dit is uiteraard enkel slim wanneer de speler denkt niet te kunnen leggen. Anderzijds kan de speler een kaart leggen. Indien de gelegde kaart fout is krijgt de speler de kaart terug met een extra kaart.
Centraal ligt er steeds een kaart. De speler mag elke kaart leggen die hetzelfde symbool, of hetzelfde cijfer heeft als de centrale kaart. De beurt van de speler is pas voorbij wanneer hij een correcte kaart heeft gelegd of wanneer hij zelfstandig heeft bijgepakt.

**Actie tijdens de beurt** Als een speler aan beurt is, is het mogelijk dat hij een actie moet doen. Meestal komt dit neer op het zeggen van een speciaal woord, of soms iets kleins doen. We zullen hier niet alle mogelijke acties bespreken. Een voorbeeld van een actie is als volgt. Wanneer een speler een acht legt, moet de speler "BONG"zeggen. Het kan zijn dat een speler in een beurt meerdere acties, of helemaal geen acties moet doen. Normaal gezien wordt elke foutieve, overbodige of ontbrekende actie afgestraft met 1 strafkaart. Wij versimpelen het probleem echter door maar 1 strafkaart te geven voor alle foutieve acties. Dit voorkomt dat alle kaarten bij de spelers zitten en de bijpakstapel dus leeg is. Indien dit het geval is en iemand moet een kaart ontvangen eindigt het spel zonder een winnaar.

# 2 Onderzoeksvraag

Onze onderzoeksvraag is zeer eenvoudig: "Kan een AI mao zoals een mens leren spelen". Deze eenvoudige vraag heeft echter een diepe betekenis, mao is namelijk een spel vol nuances en hindernissen. Wanneer een mens het kaartspel voor het eerst tegenkomt, is er bijna altijd eerst een fase van verwarring/proberen vol met fouten. Door deze foute keuzes te maken (en door voorkennis van andere spellen) kan hij/zij wel bijna altijd na verloop van tijd de spelregels ontcijferen en het spel vlekkeloos spelen. Een AI heeft echter helemaal geen voorkennis, maar heeft wel het voordeel van een groot geheugen en grote rekenkracht. Ons doel is om een AI te ontwikkelen die een waardige tegenstander is van een mens die het spel ook voor het eerst ontdekt. We focussen ons in dit verslag niet op optimaliteit, in het spel zelf heeft men namelijk redelijk veel geluk nodig. Meestal hebben spelers niet zoveel keuze in het leggen van hun kaart. De moeilijkheid ligt in het snel en correct ontcijferen van de regels om zo sneller dan tegenstanders te weten hoe het spel in elkaar zit.

# 3 Ideeën

## 3.1 Literatuur

Voor we beginnen programmeren is het belangrijk dat we eerst wat onderzoek doen. In deze sectie formuleren we even kort welke literatuur/artikels we hebben bekeken om uiteindelijk bij onze 2 algoritmes te komen.

**Het spel** We zijn begonnen met extra informatie over het kaartspel zelf op te zoeken. Maar aangezien er in de regels vermeld wordt dat men er niet over mag praten is het moeilijk om er veel informatie over terug te vinden ([nab]). Ook hebben we opgemerkt dat, aangezien het spel al spelend wordt overgedragen, er soms veel verschil zit op de regels van persoon tot persoon. Daarom hebben we beslist onze AI's zo algemeen mogelijk te maken, hierdoor zouden ze voor elke set van regels moeten werken.

**Een algoritme** Online zijn er veel AI-algoritmes te vinden, maar meestal zijn deze ver van ideaal. Na de lessen bij te wonen en enkele artikels te lezen, zijn we bij 2 algoritmes terechtgekomen. Deze algoritmes bespreken we kort in de volgende secties.

## 3.2 Neuraal netwerk

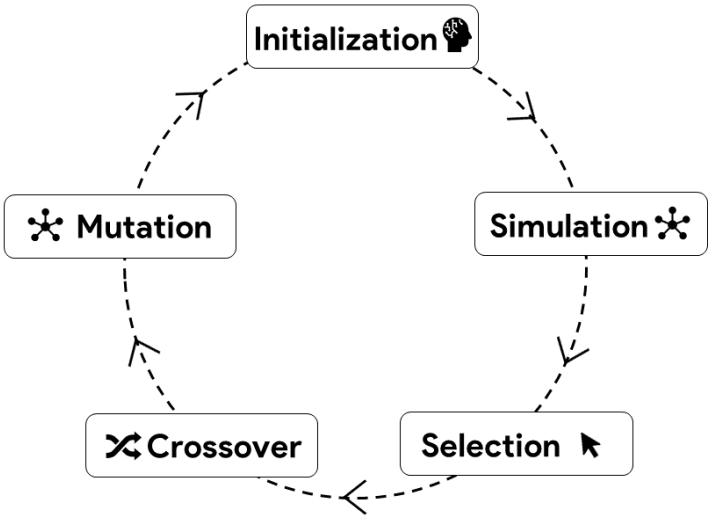
**De voorstelling** Een eerste oplossing om onze AI te ontwikkelen kwam in de vorm van een neuraal netwerk. Na wat onderzoek bleek al snel dat we de structuur van een netwerk kunnen gebruiken om beslissingen te nemen. In het geval van ons kaartspel doen we dat door bijvoorbeeld kaarten als input en output van het netwerk mee te geven. Voor de exacte implementatie van ons neuraal netwerk hebben we gekozen voor:

- Een inputlaag van booleans met een bepaalde lengte.
- Een aantal hidden layers van doubles met bepaalde lengte.
- Een outputlaag van booleans met een bepaalde lengte

Door onze input en output booleans te maken is het makkelijk beslissingen te nemen welke acties er zullen moeten worden uitgevoerd (namelijk diegene waar de boolean van de output true is). Voor de interne stuctuur kozen we ervoor Sigmoid [Woo] op de hidden layers (meer flexibiliteit) en Perceptron [nac] op de output toe te passen (true/false).

**Hoe trainen?** Nu vragen we ons nog af hoe we dit neurale netwerk exact moesten trainen. De oplossing kwam uit een online artikel over genetische modellen [Sim]. Dit artikel gaf een voorbeeld van een genetisch model toegepast op een neuraal netwerk. Dit leek ons een simpel maar effectief voorbeeld om op verder te bouwen. Aangezien we beiden voor dit project nog niet zeer vertrouwd met AI's waren bleek dit het perfecte startpunt: eenvoudig, leerrijk en bovenal effectief voor ons probleem.

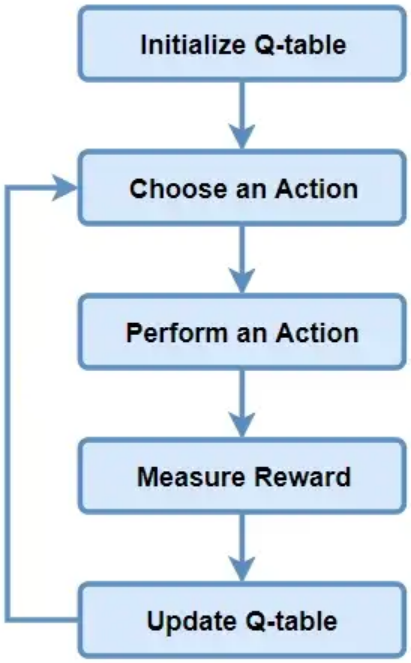
**Genetisch algoritme** Het exacte algoritme zit redelijk eenvoudig in elkaar. Eerst nemen we een pool van een aantal AI's die elk de gewichten van ons netwerk bijhouden. Vervolgens voeren we voor elke AI een simulatie uit die bepaalt hoe goed deze AI presteert. In onze implementatie hebben we ervoor gekozen om het totaal aantal fouten (op een game) te proberen minimaliseren. Nadat we weten hoe elke AI presteert, gooien we de slechtste helft van onze pool weg. Op onze beste helft passen we crossover toe om nieuwe AI's te krijgen die de pool opnieuw opvullen. Op deze nieuwe AI's passen we ook een kleine mutatie toe om variatie te creëren en evolutie te stimuleren. Uiteindelijk herhalen we deze stappen totdat we een AI bekomen die een game speelt zonder fouten te maken.




Figuur 1: Een visuele voorstelling van het genetisch algoritme.

## 3.3 Q-Learning

Als 2de mogelijke aanpak zagen we Q-learning. Het idee om het spel als een sequentie van state-action paren voor te stellen leek ons een goed idee. Op deze manier zouden we veel vlugger een resultaat bekomen dan bij een genetisch algoritme.




Figuur 2: Een visuele voorstelling van het Q-learning algoritme. [Shy]

**Q-tabel updaten** De Q-tabel updaten we volgens de Bellman vergelijking [nad]. We zijn geïnteresseerd in altijd correct spelen en niet in optimaal spelen (2). Het maakt ons niet uit of we winnen of verliezen, zolang we maar correct de regels ontcijferen. Dit heeft als gevolg dat de discount factor $\gamma$ irrelevant is voor ons probleem. We gebruiken dus een aangepaste Bellman vergelijking:

$$Q^{new}(s,a) \leftarrow Q(s,a) + \alpha(r - Q(s,a)) \tag{1}$$

Wij kiezen ervoor om een action uit te voeren wanneer $Q(s,a) \geq 0$. Onze tabel wordt steeds op 0 geïnitialiseerd. Hierdoor is onze start policy elke actie zeker 1 keer te nemen. Op deze manier hopen we het meeste te leren uit onze fouten.

# 4 Implementatie

Alle code is terug te vinden in onze github repository. [MV]

**Programeertaal** Voor de implementatie van de AI hebben we de object georiënteerde programeertaal C++ gekozen om de volgende redenen:

- **Efficiëntie** Berekeningen bij een AI kunnen computationeel zwaar worden, een efficiënte taal is noodzakelijk. Zeker wanneer we kleinere AI's lokaal willen trainen.
- **Eigen implementatie** Dit is onze eerste aanraking met AI, door alles zelf te implementeren krijgen we meer voeling met het onderwerp dan als we een library zouden gebruiken. We hopen dan ook veel bij te leren over AI door dit project te maken.
- **Object georiënteerd** Het OOP paradigma laat ons toe om alle ideeën op een overzichtelijke manier te implementeren.

**Mao-AI ontwerp** Ons doel is om een AI te ontwikkelen die correct Mao speelt. Hiervoor zal de AI correct een beurt in het spel moeten uitvoeren. Zoals in sectie 1.2 reeds vermeld bestaat een beurt uit 3 onderdelen. Aangezien deze delen bijna onafhankelijk van elkaar zijn hebben we ervoor gekozen om voor elk van de 3 een AI te ontwikkelen. Onze finale AI zal dus een combinatie zijn van 3 sub AI's. In onderstaande 3 secties worden deze AI's uitvoerig besproken.

**Genetic model** Zoals in sectie 3.2 reeds besproken zullen we gebruik maken van een genetisch algoritme op een neuraal netwerk. Het genetisch algortime is gedefineerd in de klasse GeneticAlgorithm [1]. We hebben ook een NeuralNetworkAI [2] voorzien die de gewichten van het Netwerk bijhoudt. Alle verdere Genetische AI's zullen van deze klasse gebruik maken in hun Algoritme. Verder gebruikt ons Genetisch algoritme een Simulator om 1 game van een Ai te simuleren, hiervoor gebruiken we de NeuralNetworkSimulator [3] aangezien we ons genetisch model altijd op een neuraal netwerk uitvoeren.

**Q model** Zoals in sectie 3.3 reeds besproken zullen we gebruik maken van Q-learning. Hiervoor hebben we een generieke klasse [4] geschreven die een Q tabel bijhoudt. De klasse ondersteunt functionaliteit om de tabel te updaten aan de hand van onze aangepaste Bellman vergelijking (1). Voor elke Q-AI hebben we een trainer [5]. De trainers zullen de AI n games laten spelen tegen bots die altijd juist zijn, maar af en toe een fout maken zodat het spel pas eindigt wanneer de AI wint. Op deze manier duren de spellen langer en zal de AI meer leren.

## 4.1 Move

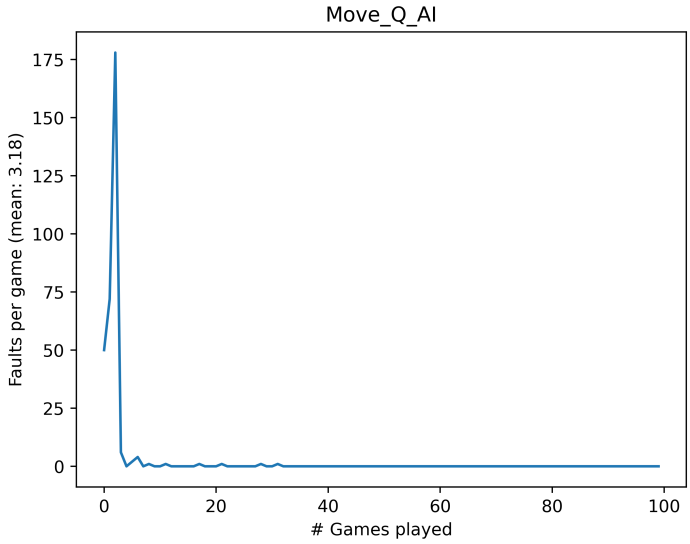
Een eerste AI is de moveAI. Dit is de eerste AI die wordt aangesproken wanneer er een beslissing moet gemaakt worden. De moveAI moet weten welke speler er aan beurt is. De focus van deze AI is dus het weten of we al dan niet aan beurt zijn.

Meestal is het voor een gewoon persoon al snel heel duidelijk wie er wanneer aan de beurt is aangezien we zoals elk kaartspel een richting hebben waarmee we de cirkel rond gaan. Natuurlijk kunnen er ook extra regels zoals overslaan en wisselen van richting worden ingevoerd. Het probleem is dat een AI deze voorkennis niet heeft, maar toch hebben we ontdekt dat we met Q-learning dit probleem efficiënt kunnen oplossen.

### 4.1.1 Q-Move

De Q-Move AI[6] is dus de AI die Q-learning (sectie 3.3) gebruikt om dit probleem op te lossen. We hebben beslist om voor de states van onze Q-tabel een combinatie van de vorige 2 spelers met de bovenste kaart van de stapel te kiezen. Met de regels die wij hebben uitgeprobeerd gaf dit zo goed als altijd een mooi resultaat, maar een kleine uitbreiding is natuurlijk altijd mogelijk. Zoals we zien in onderstaande figuur (Figuur 3) zien we dat deze AI zeer snel een optimale policy bereikt. In de eerste games maakt de AI soms foute beslissingen maar al snel verbetert hij zichzelf. Het is duidelijk om te zien dat deze AI optimaal presteert en niet veel ruimte heeft voor verbetering heeft.

[1] Zie /ai/util/GeneticAlgorithm.h
[2] Zie /ai/util/NeuralNetworkAi.h
[3] Zie /ai/util/NeuralNetworkSimulator.h
[4] Zie /ai/q_learning/Qmodel.h
[5] Zie /ai/util/Algorithm.h
[6] Zie /ai/MoveAi/QMoveAI.h



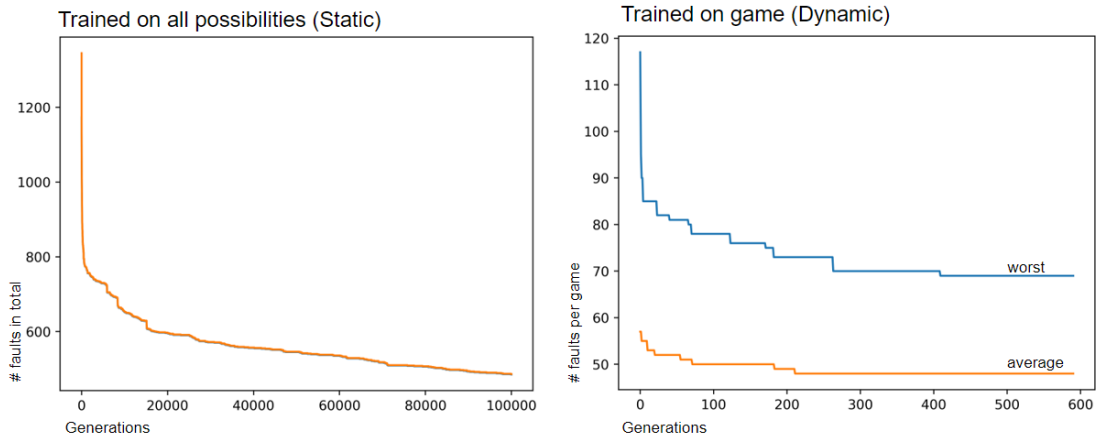

Figuur 3: Het aantal fouten van de Q-Move AI voor 100 spelletjes.

## 4.2 Play

De playAI is een AI die weet welke kaarten er mogen gelegd worden op de huidige stapel. Deze AI weet dus ook of de speler al dan niet een kaart zal moeten bijnemen. De playAI wordt enkel bevraagd wanneer de moveAI denkt dat de speler aan beurt is. De playAI moet dus concreet weten welke kaart de speler kan leggen en wanneer we een kaart moeten bijnemen.

### 4.2.1 Genetisch

Ons genetisch algoritme (sectie 3.2) voor het leggen van kaarten heeft geen groot neuraal netwerk nodig om acties te kunnen kiezen. We leggen namelijk steeds een kaart op een andere kaart, dit zijn 52 input nodes en 52 outputs. Echter merken we snel op dat de resultaten (Figuur 4) van deze AI[7] helemaal niet zo efficiënt zijn. Het probleem is iets te simpel om met een complex neuraal netwerk te berekenen.




Figuur 4: Een statisch (alle mogelijkheden) en een dynamisch (op spelletjes) getrainde genetische play-AI.

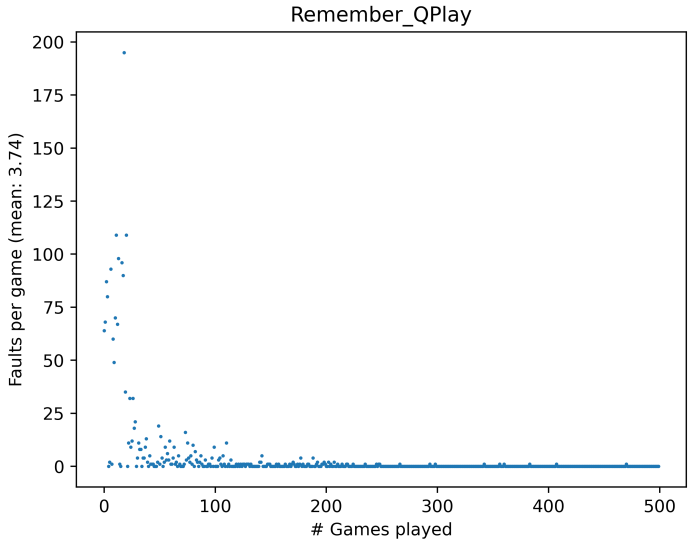
[7] Zie /ai/PlayAi/NeuralNetworkPlayAi.h

We kunnen de dynamisch getrainde AI iets verbeteren door meer games te spelen na elkaar per simulatie, maar dit zorgt er alleen maar voor dat ons algoritme nog trager wordt. We zien dat het algoritme veel te traag is om in de praktijk bruikbaar te zijn, daarom bespreken we verder ook enkele Q-learning AI's.

### 4.2.2 Q-learning

We leggen steeds 1 kaar op 1 andere kaart, dit kan op 2652 $(52 * 51)$ verschillende manieren aangezien elke kaart maar 1 keer voor komt in een kaartspel. Het lijkt ons dan ook geen groot probleem om snel een optimale oplossing te vinden. Bij onderstaande Ai's gebruiken we dan steeds ook een Q-model (sectie 3.3) met alle mogelijke kaarten als state, en alle mogelijke kaarten als actie.

**Remember** Een eerste idee is Q-learning toe passen met de learning factor $\alpha = 1$. Dit zal als gevolg hebben dat de AI[8] simpelweg voor elk state-action paar onthoudt of het al dan niet een geldige zet is.



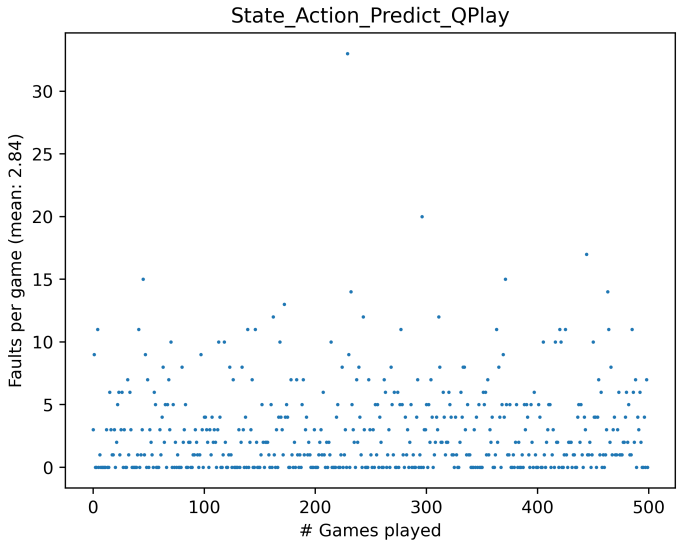

Figuur 5: Remember Q-AI voor 500 spelletjes

Zoals verwacht levert dit een optimaale policy op. We zijn echter niet tevreden met dit resultaat om verschillende redenen. Het enige dat de AI doet is voorgaande zetten onthouden. Hiernaast merken we ook op dat de AI relatief veel tijd nodig heeft om aan de perfecte policy te komen. Als mens zouden we zonder enige voorkennis na enkele spelletjes al een quasi perfecte policy kennen. We wensen dus beter te doen.

**Verband tussen 2 kaarten** Als mens is het zeer intiütief dat er een verband kan bestaan tussen 2 kaarten. Voor ons lijkt het zeer logisch dat we 2 kaarten van hetzelfde type op elkaar kunnen spelen. Voor de AI zijn dit echter 2 totaal verschillende kaarten.

[8] Zie /ai/PlayAI/QPlayAI.h

**State-action-predict AI** Het idee dat 2 kaarten een verband kunnen hebben zullen we nu gebruiken in onze AI[9]. Bij een beurt zullen we voor een gegeven state (de kaart in het midden) alle actions (mogelijke kaart die wij leggen) die hetzelfde type of nummer hebben updaten. We doen dit niet enkel voor de huidige state, maar voor alle states met hetzelfde type of nummer. Dit is de meest algemene manier om de tabel te updaten. We geven een positieve reward wanneer de zet correct was en een negatieve reward wanneer de zet fout was.
Door heel veel velden in de tabel te updaten updaten we soms ook velden foutief. Hierdoor zal de AI nooit de optimale policy bereiken, maar hij zal wel snel goede verbanden kunnen leggen.

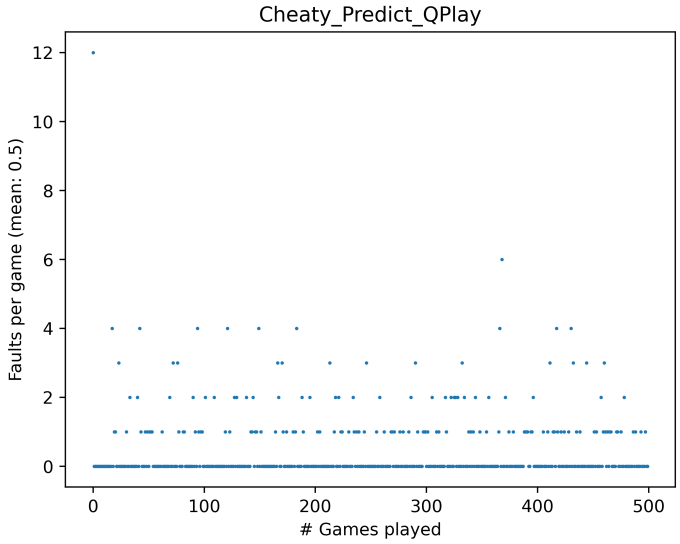



Figuur 6: State action voorspelling Q-AI voor 500 spelletjes

We zien dat de AI direct zijn best mogelijke policy bereikt. Dit is wat we wensen. Hij maakt hierbij gemiddeld een kleine 3 fouten per game. Dit is een mooi resultaat, maar we hopen nog steeds beter te doen.

**Cheaty-predict AI** Zoals in de voorgaande paragraaf vermeld updaten we te veel velden in de Q-tabel. De velden die we fout updaten zijn bijna allemaal degene waarbij we het type van de state linken met het nummer van de actie, of andersom. Het is echter niet logisch dat het correct zou zijn om een 4 op gelijk welke harte te spelen. We lossen dit probleem op door simpelweg niet meer te speculeren over de link tussen type en nummer.

[9]Zie /ai/PlayAi/StateActPredictQPlayAI.h

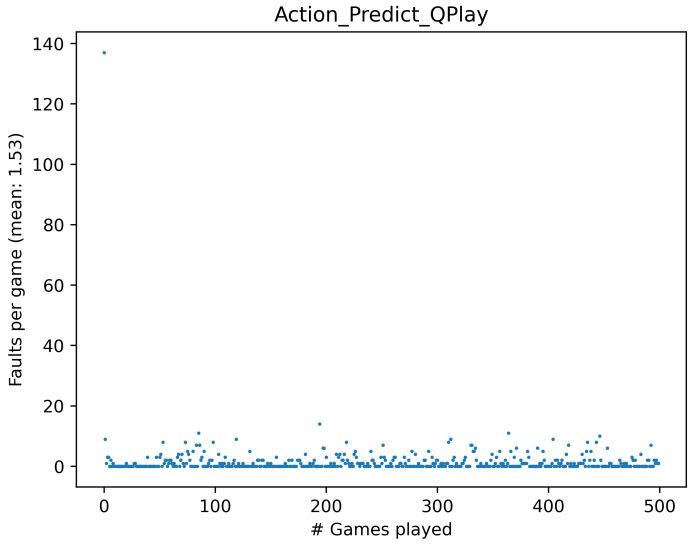



Figuur 7: Cheaty Q-AI voor 500 spelletjes.

Dit levert een zeer goed resultaat op, maar helaas hebben we nu voorkennis gegeven aan de AI. De updates die nu gebeuren zijn bijna exact gelijk aan de effectieve regel die wij gebruiken. Waar we voorheen alle mogelijke combinaties updaten, hebben we nu op basis van voorkennis van de regels de AI[10] gemaakt. Dit voelt als valsspelen aan en zal bovendien niet werken wanneer er andere regels gebruikt worden.

**Action-predict AI** We zijn dus op zoek naar een AI die op een algemene manier update, maar die slechts een minimaal aantal fouten maakt bij het updaten. We zullen nog steeds alle acties updaten met eenzelfde nummer of type als de state, maar we zullen dit enkel doen voor de gegeven state en niet meer speculeren over andere states. Waar we voorheen in 2 dimensies fouten maakten zullen we nu dus maar in 1 dimensie fouten maken. Hierdoor kan het misschien wel ietwat langer duren voor we aan de optimale policy van deze AI komen.

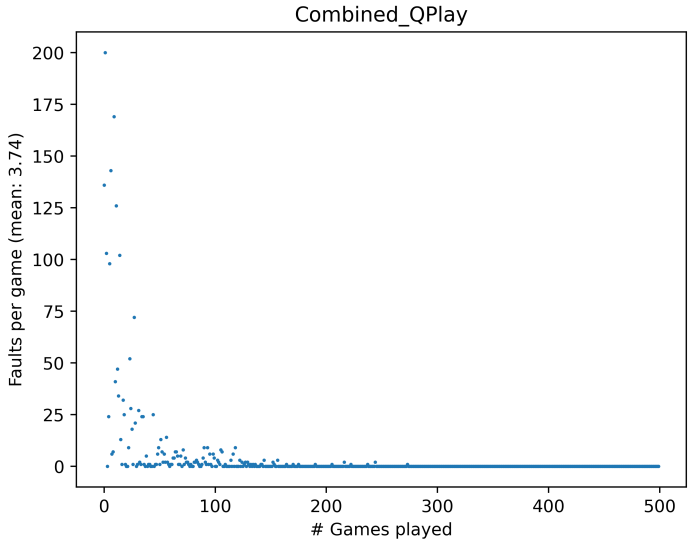
[10]Zie /ai/PlayAi/StateActPredictQPlayAI.h




Figuur 8: Action voorspelling Q-AI voor 500 spelletjes.

De AI[11] maakt gedurende 1 game heel veel fouten, maar leert daar veel uit, waardoor hij vervolgens direct in zijn optimale policy zit. Indien we deze ene game met veel fouten niet mee tellen is zijn gemiddelde 1.25 fouten per game. Deze policy is in tegenstelling tot de remember AI (sectie 4.2.2) niet perfect, maar dit is een zeer goede policy en we hoeven niet te wachten op het resultaat.

**Een combinatie van 2 werelden** De action voorspelling geeft een zeer mooi resultaat, maar heeft op termijn niet de optimale oplossing. We zouden deze AI dus kunnen combineren met de remember AI. Op deze manier zouden we snel een goed resultaat moeten verkrijgen en op termijn een optimaal[12].




Figuur 9: Combined Q-AI voor 500 spelletjes.

[11]Zie /ai/PlayAi/ActPredictQPlayAI.h
[12]Zie /ai/PlayAi/CombinedQPlayAI.h

De figuur gaat echter tegen onze verwachtingen in. Hij gedraagt zich bijna exact zoals de remember AI. Aangezien we de remember AI gebruiken zodra die iets zeker weet zal onze predict AI veel minder fouten maken. Dit heeft als gevolg dat de begin policy (leg altijd) veel minder overschreven zal worden en de predict AI dus veel minder invloed zal hebben. Als resultaat bekomen we dus opnieuw een AI die relatief veel tijd nodig heeft om het spel te begrijpen.

## 4.3 Act

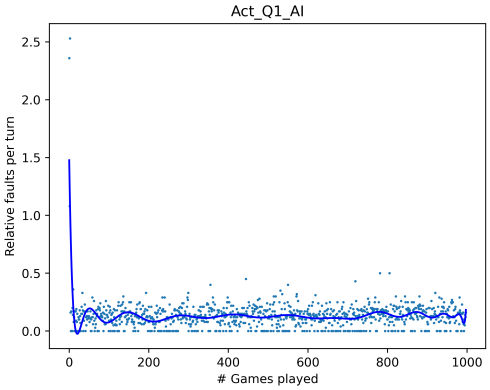
De actAI is de derde en laatste AI. Wanneer we weten dat het onze beurt is en we een kaart gelegd/genomen hebben moeten we meestal ook een actie doen. Deze AI is verantwoordelijk om te weten welke acties wanneer zullen moeten worden uitgevoerd. Dit is veruit de meest complexe AI, maar daarom misschien ook de meest interessante.

De act regels kunnen vaak eenvoudig beschreven worden in 1 of 2 zinnen, maar dit kan al snel een zeer ingewikkeld patroon opleveren. Meestal zijn de act regels simpel en is enkel de kaart die pas gelegd werd op de stapel nodig om de act correct uit te voeren. Bijvoorbeeld: "Wanneer de waarde van de gespeelde kaart een veelvoud van 3 is, zeg het woord *tri*". Wanneer men slechts 1 of 2 kaarten diep in de stapel van gelegde kaarten moet kijken om te weten welke act er moet gebeuren kan Q-learing dit probleem makkelijk oplossen.
Er kunnen echter ook regels bedacht worden waarbij men dieper in de stapel zal moeten kijken. Bijvoorbeeld: "Zeg het woord *pi* wanneer het volgende cijfer van $\pi$ gelegd wordt, begin opnieuw wanneer er een nieuwe 3 gespeeld wordt". Hierbij kan het dus zijn dat er een 3 gespeeld wordt, het vervolgens 6 beurten duurt tegen dat er een 1 gespeeld wordt. We zouden dan al 7 kaarten diep in de stapel moeten kijken. De Q-tabel zou dus veel te groot worden, hiervoor kijken we dus naar een genetisch algoritme.

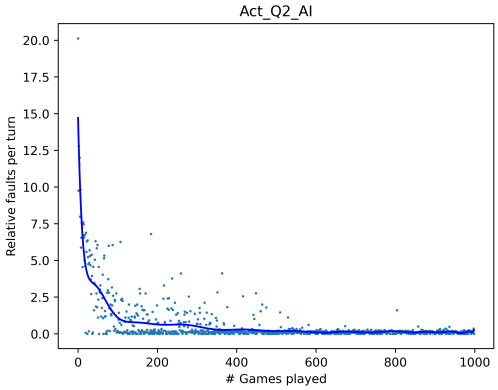
### 4.3.1 Q-Act

Zoals eerder besproken lijkt Q-learning een goede aanpak wanneer we niet te diep in de stapel moeten kijken. We schrijven dus een Q-learning AI[13]. Als state voor het Q-model (sectie 3.3) kiezen we de n (diepte) bovenste kaarten van de stapel. Als action kiezen we alle mogelijke acts in het spel. De AI zal ons dus voor een gegeven state proberen te vertellen welke acts we allemaal moeten uitvoeren. Dit kunnen er nul of meer zijn. Indien we de diepte groter maken zullen we betere resultaten verkrijgen, maar zal het veel langer duren tegen dat de Q-tabel goede waarden bevat.

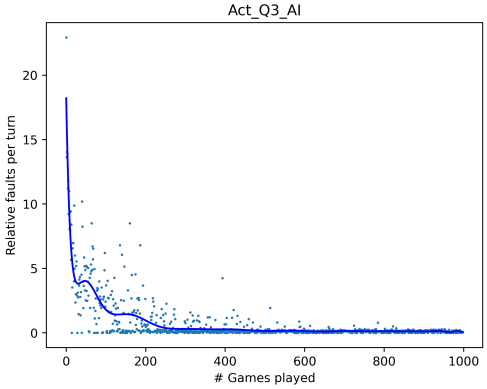
[13]Zie /ai/ActAi/QActAI.h




(a) Diepte 1




(b) Diepte 2




(c) Diepte 3

Figuur 10: Q-Act met verschillende diepte

In onze set van regels zijn er maar enkele regels die grotere diepte nodig hebben. Hierdoor is het verschil op het einde van de grafieken niet groot.
In onderstaande tabel kan men zien hoeveel mogelijke verschillende states er zijn wanneer we dieper in de stapel kijken. Er zijn 53 kaarten: de 52 klassieke kaarten en een None kaart die aangeeft dat er is bijgenomen.

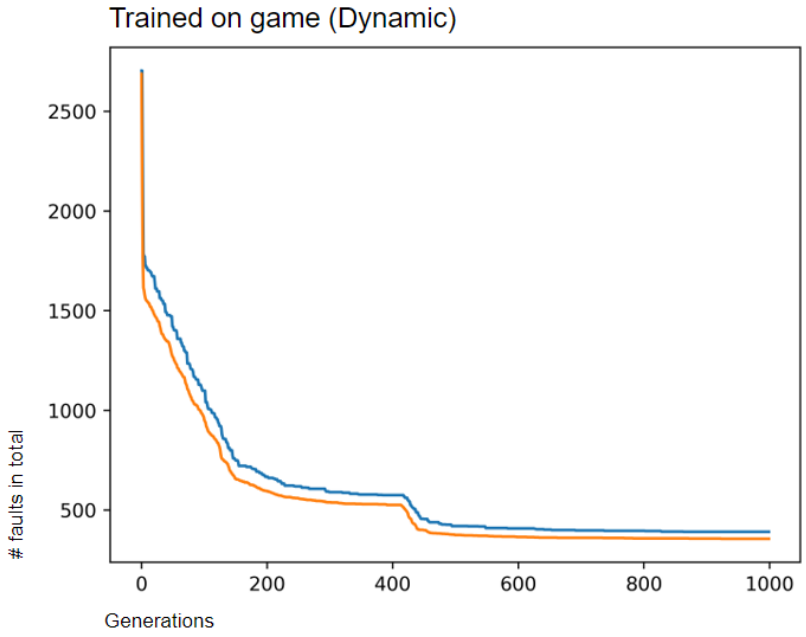
| Diepte | #States |
|---|---|
| 1 | 53 |
| 2 | $53^2 = 2809$ |
| 3 | $53^3 = 148877$ |
| 4 | $53^4 = 7890481$ |

Tabel 1: Een tabel met het aantal states voor een kaartdiepte.

Het aantal states stijgt dus exponentiëel en is niet meer schaalbaar voor Q-learning. Daarom bekijken we in de volgende sectie een genetisch algoritme.

### 4.3.2 Genetisch

Het genetisch algoritme[14] voor de Acts heeft een veel ingewikkelder neuraal netwerk. We kunnen zoals hierboven besproken namelijk zeer veel kaarten als input hebben wanneer we veel in het verleden willen kijken. Zoals we in sectie 4.2.1 hebben gezien is een genetisch algoritme zeer traag. Toch heeft dit algoritme ook enkele voordelen. Wanneer we namelijk een zeer complex probleem hebben kan het neuraal netwerk hier een structuur in terug vinden. Zoals we in figuur 11 kunnen zien presteert deze AI al veel beter dan de AI in sectie 4.2.1. Het aantal fouten is nog helemaal niet minimaal maar we merken wel duidelijk een mooie leercurve. Ook opmerkzaam aan een genetisch algoritme zien we dat we na 400 games plots een sterke daling in fouten hebben. Dit komt doordat het algoritme hier een zeer goede mutatie heeft doorgevoerd. Het voordeel van een neuraal netwerk is dat, aangezien regels meestal een structuur hebben, deze mutatie voor meerdere fouten geldig is. Met een beetje geluk krijgen we dus goede mutaties waardoor het probleem sneller wordt opgelost.




Figuur 11: Het aantal fouten van de genetic Act AI (diepte 4) voor 1000 spelletjes.

Alhoewel het probleem hier dus trager wordt opgelost, zijn we ervan overtuigd dat het probleem uiteindelijk wel compleet correct zal worden opgelost. Dit kan over Q-learning met dezelfde resources niet gezegd worden.

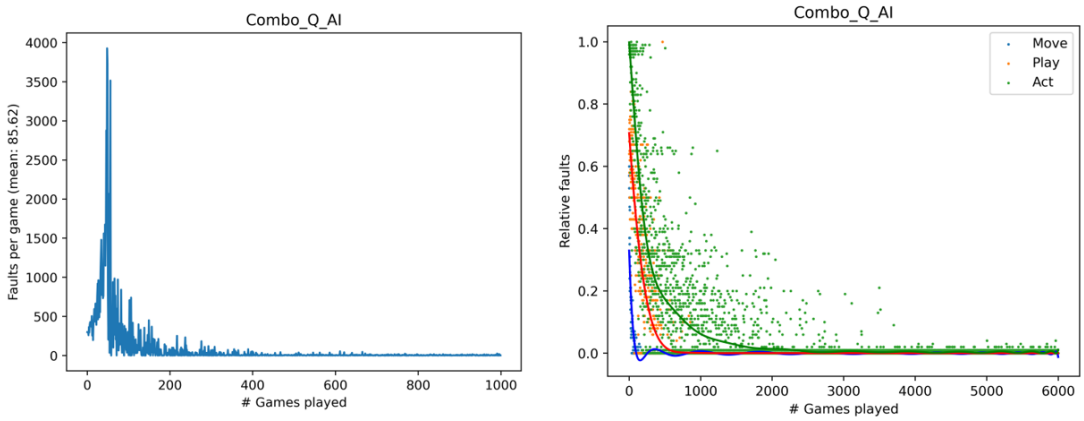
[14] Zie /ai/ActAi/NeuralNetworkActAi.h

# 5 Resultaat

In deze sectie bespreken we kort de resultaten en bevindingen van ons project. Hierin bespreken we onze finale AI (de combined AI) en de prestatie van deze AI. We bekijken ook eens kort welk verder onderzoek er misschien mogelijk is.

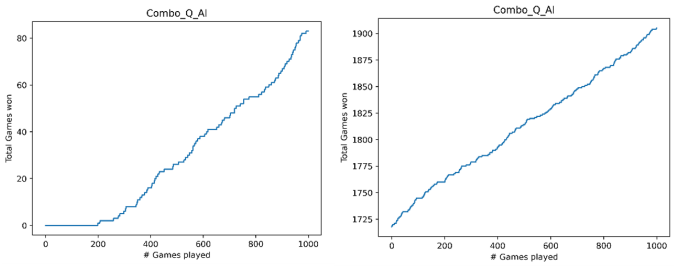
## 5.1 Combined AI

De combined AI[15] is een AI die onze 3 AI's combineert tot 1 grote AI die de game volledig zelfstandig leert te spelen. We hebben ervoor gekozen om de 3 Q-learning AI's te gebruiken aangezien de genetische AI's gewoon te traag zijn om praktisch bruikbaar te zijn. De resultaten van onze combined AI kunnen we bekijken in onderstaande figuur (Figuur 12). Merk op dat de de rechterfiguur de fouten relatief afbeeldt (herschaald tussen 0 en 1) en dat de absolute Act-fouten ongeveer 35 keer zo groot zijn. We zien ook dat de Move en Play AI slechter presteren dan toen we ze aan het trainen waren. Dit komt omdat we toen tegen bots speelden die het spel nooit lieten eindigen zolang de AI niet won. Hier proberen de bots echter zelf ook te winnen. Dit heeft als gevolg dat de games korter zijn en de AI dus minder kans heeft om te leren.




Figuur 12: Het absoluut aantal fouten van de combined AI over 1000 spelletjes (links). Het relatief aantal fouten van de combined AI voor de 3 sub-AI's over 6000 spelletjes (rechts).

In deze grafieken merken we op hoe de AI start met enkele foute beslissingen maar zich zeer snel herpakt en naar een quasi correcte state convergeert. De AI is niet helemaal correct want sommige Acts zijn moeilijk te ontcijferen. In de 2de figuur (Figuur 13) zien we hoeveel keer de AI wint, deze figuur is minder relevant maar nog steeds interessant om te zien dat de AI wel leert.

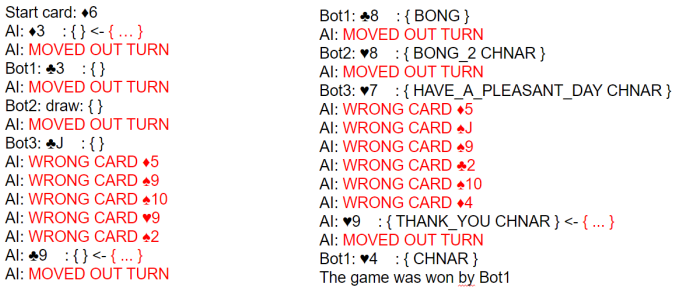



Figuur 13: Het absoluut aantal gewonnen spelletjes van de combined AI over 1000 spelletjes voor en na het trainen van 10.000 spelletjes.

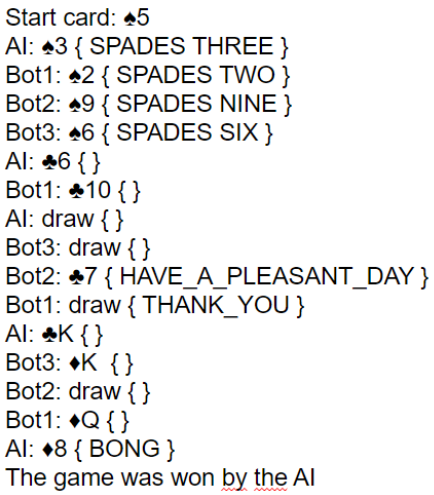
[15] Zie /ai/ComboAI

## 5.2 Prestatie

We kunnen ook de games van de AI uitprinten voor en nadat de AI heeft getrained. Hier merken we ook een groot verschil op in de prestatie. Wanneer we figuur 14 en figuur 15 bekijken is het duidelijk te zien hoe snel de AI het concept van het spel doorheeft.




Figuur 14: Een spelletje gespeeld door een ongetrainde AI (fouten in het rood).




Figuur 15: Een spelletje gespeeld door een getrainde AI na 10.000 spelletjes (zonder fouten).

We kunnen zien dat de combined AI goed presteert, maar zoals verwacht is de AI nooit optimaal. Er zijn namelijk enkele Acts die deze snelle AI nooit kan ontcijferen. We zijn echter wel zeer tevreden over deze AI omdat hij zeer snel is. Met de code op onze github [MV] kan je ook zelf op een paar minuutjes de AI trainen om te zien hoe die na 10.000 games presteert.

## 5.3 Toekomst

Voor toekomstig onderzoek over dit onderwerp lijkt het ons interessant een 3de algoritme erbij te halen. Hiervoor denken we zelf aan decision trees [naa], deze zouden ongeveer functioneren zoals het neuraal netwerk. Maar met een ander trainingsalgoritme (geen genetisch) zouden we dit misschien ook snel kunnen laten presteren. We denken dat Q-learning heel goed presteert dus enkele variaties zoals onze Q-Play AI modellen zouden ook interessant zijn om te bekijken.
Voor de Q-Act AI zou men bijvoorbeeld kunnen opteren om een Q-model voor diepte 1 bij te houden en ook bijhouden hoe vaak een state aangepast werd. Indien de state te vaak aangepast werd ligt het misschien aan de diepte. Op dat moment kan men aan de gegeven state een extra kaart toevoegen. Op deze manier kijkt men enkel diep waar nodig.

# 6 Conclusie

De conclusie van onze onderzoeksvraag *"Kan een AI zoals een mens Mao leren?"* is genuanceerd. We bekijken eerst enkele aparte conclusies.

**Genetisch Algoritme** Het genetisch algoritme presteert in enkele gevallen redelijk goed. Het algoritme is echter veel te traag om gebruikt te kunnen worden in de praktijk. Zeker omdat wij snelheid voor dit project een zeer belangrijk punt vinden.

**Q-learning** Ons 2de algoritme werkt zeer snel en correct maar kan zeer moeilijk linken leggen tussen de beschikbare data. We zien dit probleem vooral opduiken bij complexere problemen (zoals de Act AI).

**Combined AI** Onze finale AI heeft een zeer duidelijke leercurve en kan zeer snel de regels ontcijferen. Deze AI presteert echter nog altijd iets slechter als een optimale Bot. Onze AI kan het op lange periode niet opnemen tegen een normaal persoon die de regels op termijn wel volledig zal begrijpen.

Uit al deze resultaten zien we dat we duidelijk moeten afwegen tussen snelheid en correctheid. Als we ons toespitsen op snelheid kan onze AI op korte periode misschien sneller dan een mens correcte conclusies nemen, maar dan zal die onvermijdelijk over langere periode niet correct presteren. Wanneer we wel correct willen presteren is de AI echter veel te traag. We zien dat we hier duidelijk moeten afwegen tussen deze 2 factoren. Met onze combined AI hebben wij al een afweging gemaakt, maar naar gelang de speler's wensen kan een andere AI samengesteld worden. We denken dat het alvast leuker is om tegen onze AI te spelen dan tegen een perfecte bot, aangezien onze AI samen met de speler leert. Om dus terug te komen op onze onderzoeksvraag: Een AI kan duidelijk een waardige tegenstander van een mens zijn.

# Referenties


[MV] Brent Matthys en Mats Van Belle. *Mao AI*. URL: https://github.ugent.be/brmatthy/mao_ai.

[naa] n.a. *Decision tree*. Wikipedia. URL: https://en.wikipedia.org/wiki/Decision_tree. (accessed: 23/12/2022).

[nab] n.a. *Mao (kaartspel)*. Wikipedia. URL: https://nl.wikipedia.org/wiki/Mao_(kaartspel). (accessed: 23/12/2022).

[nac] n.a. *Perceptron*. Wikipedia. URL: https://en.wikipedia.org/wiki/Perceptron. (accessed: 23/12/2022).

[nad] n.a. *Q-learning*. Wikipedia. URL: https://en.wikipedia.org/wiki/Q-learning.

[Shy] Chathurangi Shyalika. *A beginners Guide to Q-Learning*. URL: https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c. (accessed: 23/12/2022).

[Sim] Victor Sim. *Using Genetic Algorithms to Train Neural Networks*. URL: https://towardsdatascience.com/using-genetic-algorithms-to-train-neural-networks-b5ffe0d51321. (accessed: 23/12/2022).

[Woo] Thomas Wood. *What is the Sigmoid Function?* URL: https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function. (accessed: 23/12/2022).