Reinforcement Learning en Computerspellen

Hoe beïnvloeden de specifieke kenmerken van computerspellen de effectiviteit van specifieke reinforcement learning-algoritmes?

Matthijs Gorter
Thom Brinkhorst
Pepijn van Iperen

Profielwerkstuk
onder begeleiding van
S. Rook
Christelijk Lyceum Zeist
Natuur en Techniek
Februari 2025

Voorwoord

Toen we begonnen na te denken over een onderwerp voor ons profielwerkstuk, wilden we graag een thema kiezen dat zowel uitdagend als actueel was. Kunstmatige intelligentie fascineert ons al enige tijd, vooral vanwege de invloed die het heeft op onze toekomst en de vele toepassingen die het nu al kent. Het idee om ons te verdiepen in reinforcement learning ontstond omdat deze tak van KI niet alleen theoretisch interessant is, maar ook praktisch ontzettend krachtig is.

Reinforcement learning staat aan de basis van indrukwekkende prestaties, zoals zelflerende spelprogramma's, geavanceerde robotsystemen en zelfrijdende auto's. De manier waarop een computer 'leert' door beloningen en straffen sprak ons aan, omdat het lijkt op hoe wij als mensen leren. Het leek ons daarom een perfecte uitdaging om dit complexe onderwerp te onderzoeken en te begrijpen hoe het precies werkt.

Matthijs Gorter, Thom Brinkhorst, Pepijn van Iperen
Christelijk Lyceum Zeist
Februari 2025

Notatie

Variabele	**Definitie**
t	Tijdstap
T	Laatste tijdstap van een episode (horizon)
x	Toestand (state)
x_t	Toestand op tijdstap t
x'	De volgende toestand
$\mathcal{X}$	Toestandsruimte
a	Actie
$\mathcal{A}$	Actieruimte
a_t	Actie op tijdstip t
r	Beloning (reward)
$\mathcal{R}$	Beloningsruimte
r_t	Beloning op tijdstip t
$r(x, a)$	Beloningsfunctie
μ	Deterministisch beleid
π	Stochastisch beleid
π^*	Optimale stochastisch beleid
α	Leersnelheid tussen 0 en 1
γ	Kortingsfactor tussen 0 en 1
ϵ	Exploratieparameter tussen 0 en 1
$p(x'\|x, a)$	Overgangswaarschijnlijkheidsfunctie
$V(x)$	Waardefunctie
$Q(x, a)$	Q-functie
$\mathbb{E}[X]$	Verwachtingswaarde van variabele X
θ	De gewichten van het hoofd neuraal netwerk
θ^-	De gewichten van het target netwerk
Φ	De voorbewerkte toestand
$\mathcal{D}$	Replay-geheugen voor opslag van transities.
$\mathcal{N}$	Capaciteit van het replay-geheugen
y_j	Doelwaarde voor training

Tabel 1: Notatie

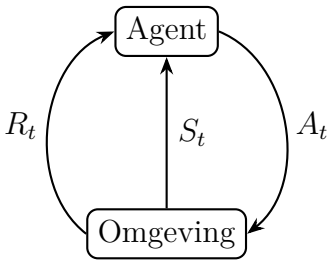
Inhoudsopgave

Hoofdstuk 1

Inleiding

Reinforcement Learning (RL) is een subdiscipline binnen de kunstmatige intelligentie (KI) die zich richt op het trainen van een agent om optimale acties te ondernemen binnen een specifieke omgeving. Een agent is een entiteit die leert en acties onderneemt. Bij een zelfrijdende auto is het besturingssysteem de agent, en bij een schaakspel is de schaker de agent.

De omgeving is alles waarmee de agent interageert en die reageert op de acties van de agent. Bij een zelfrijdende auto is dit de weg waar de auto op rijdt en de voertuigen om de auto heen. Bij een schaakspel is dit het schaakbord. De agent leert door interactie met zijn omgeving. De agent ontvangt beloningen of straffen (negatieve beloningen) als gevolg van zijn acties. Het doel van de agent is om een strategie te ontwikkelen die de cumulatieve beloning maximaliseert over tijd.



Figuur 1.1: RL model tussen agent en omgeving.

Dit proces vindt plaats door middel van een vallen en opstaan aanpak, waarbij de agent beloningen ontvangt voor correcte acties en straffen voor incorrecte acties(negatieve beloningen). Het uiteindelijke doel is het maximaliseren van de cumulatieve beloning over tijd.

Computerspellen vormen een ideaal testplatform voor RL vanwege de veelzijdige uitdagingen die ze bieden, zoals dynamische omgevingen, complexe regels en onvoorspelbare scenario's. RL heeft bewezen effectief te zijn in spellen met zowel discrete als continue actie- en toestandruimtes, variërend van actiespellen zoals Snake tot strategische spellen zoals Schaken. Het toepassen van RL in gaming vereist een diep begrip van zowel de kenmerken van de spellen als de algoritmes die worden ingezet.

1.1 Doel van het onderzoek

Het doel van dit onderzoek is om te begrijpen hoe de kenmerken van verschillende computerspellen de effectiviteit van verschillende reinforcement learning (RL) algoritmes beïnvloeden bij het verbeteren van spelprestaties. Dit onderzoek richt zich op het identificeren van de eigenschappen van verschillende soorten spellen en de kenmerken van RL-algoritmes.

Door verschillende RL-algoritmes toe te passen op een reeks spellen met verschillende kenmerken, willen we ontdekken welke algoritmes het beste presteren in welke soorten spellen. Dit kan variëren van strategische spellen die planning vereisen tot actiespellen die snelle beslissingen vragen.

1.2 Onderzoeksvragen

Hoofdvraag

Hoe beïnvloeden de specifieke kenmerken van computerspellen de effectiviteit van verschillende reinforcement learning-algoritmes in het optimaliseren van spelprestaties?

Deelvragen

Om beter te begrijpen hoe de kenmerken van computerspellen de prestaties van verschillende reinforcement learning (RL) algoritmes beïnvloeden, hebben we drie belangrijke deelvragen opgesteld

1. **Wat zijn de specifieke kenmerken van verschillende soorten computerspellen?**

   Deze vraag richt zich de eigenschappen van verschillende soorten computerspellen. Spellen kunnen sterk verschillen in hoe ze zijn opgebouwd, hoe snel spelers beslissingen moeten nemen en hoe complex de spelregels zijn. Door deze kenmerken te onderzoeken, kunnen we inzicht krijgen in welke aspecten van een spel een uitdaging vormen voor RL-algoritmes.

2. **Welke reinforcement learning-algoritmes zijn beschikbaar en wat zijn hun kenmerken?**

   Hier willen we kijken naar de verschillende soorten RL-algoritmes die beschikbaar zijn en wat hen uniek maakt. Sommige algoritmes zijn beter in het leren van een-

voudige taken, terwijl andere juist goed zijn in het omgaan met complexe situaties.

3. **Hoe beïnvloeden de spelkenmerken de prestatie van reinforcement learning-algoritmes?**

   Deze vraag gaat in op het belangrijkste deel van het onderzoek: het verband tussen de kenmerken van een spel en hoe goed een RL-algoritme presteert. We willen weten hoe bepaalde eigenschappen van een spel, zoals de noodzaak voor snelle beslissingen of lange-termijnplanning, invloed hebben op de effectiviteit van een algoritme. Door de prestaties van verschillende algoritmes in verschillende spellen te vergelijken, kunnen we ontdekken welke het beste werken voor bepaalde soorten spellen en waarom dat zo is.

1.3 Hypothese

We verwachten dat:

1. Deep Q-Network het beste zal presteren in Snake omdat het algoritme snel kan leren in omgevingen met beperkte ruimte en snel veranderende situaties, waar directe beloningen een grote rol spelen.
2. Proximal Policy Optimization zal beter presteren in Mario Super Bros, omdat dit algoritme geschikt is voor dynamische omgevingen en situaties waar zowel snelheid en planning belangrijk zijn.
3. AlphaZero zal beter zijn in Schaken, vanwege het planning en lange-termijnstrategie die nodig zijn.

1.4 Relevantie van het Onderzoek

Dit onderzoek laat effectiviteit van reinforcement learning algoritmes in verschillende omgevingen laat zien, wat bijdraagt aan het beter gebruik van KI-systemen. Deze kennis kan niet alleen worden toegepast binnen de game-industrie, maar ook in andere sectoren zoals de gezondheidszorg, zelfrijdende auto's en robotica.

Hoofdstuk 2

Theoretisch Kader

Reinforcement Learning (RL) opereert binnen het kader van Markov Decision Processes (MDP's), die een wiskundige basis bieden voor het modelleren van sequentiële beslissingsproblemen. Dit hoofdstuk bespreekt de fundamentele concepten en wiskundige formuleringen die ten grondslag liggen aan RL, gestructureerd rond vier kernonderdelen: de basiselementen van MDP's, de Markov-eigenschap en overgangsdynamiek, beleid en waarde-functies, en de Bellman-vergelijkingen.

2.1 Fundamentele Elementen van MDP's

2.1.1 Toestandsruimte

Laat ($\mathcal{X}$) de toestandsruimte zijn, waarbij elke toestand ($x \in \mathcal{X}$) de huidige situatie of staat is van de omgeving waarin de agent opereert. Op de aanvangsstap ($t = 0$) begint de agent in een initiële toestand (x_0). Naarmate het proces vordert, bevindt de agent zich in nieuwe toestanden gebaseerd op zijn acties.

2.1.2 Actieruimte

Laat ($\mathcal{A}$) de actieruimte zijn, waarbij elke actie ($a \in \mathcal{A}$) een mogelijke beslissing van de agent vertegenwoordigt. De interactie tussen de agent en de omgeving verloopt in discrete tijdstappen ($t = 0, 1, 2, \ldots, T$), waarbij de horizon (T) eindig of oneindig kan zijn.

2.1.3 Beloningsfunctie

De beloningsfunctie ($r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$) koppelt toestand-actieparen aan beloningen, waarbij ($r(x, a)$) de directe beloning vertegenwoordigt die wordt ontvangen na het uitvoeren van actie (a) in toestand (x).

2.2 Markov-eigenschap en Overgangsdynamiek

2.2.1 De Markov-eigenschap

Het onderscheidende kenmerk van MDP's is de Markov-eigenschap, die stelt dat de toekomstige toestand alleen afhankelijk is van de huidige toestand en actie, onafhankelijk van de geschiedenis:

$$p(x_{t+1}|x_t, a_t, x_{t-1}, a_{t-1}, \ldots, x_0, a_0) = p(x_{t+1}|x_t, a_t) \tag{2.1}$$

Voorbeeld van het Markov-eigenschap:

- **Snake**: De toekomstige toestand (positie van de slang en voedsel) is volledig bepaald door de huidige toestand (huidige positie en locatie van het voedsel) en de actie (richting van beweging) zonder afhankelijk te zijn van de geschiedenis van eerdere bewegingen.

Voorbeeld van geen Markov-eigenschap:

- **Poker**: De beslissingen in poker zijn afhankelijk van niet alleen de huidige hand, maar ook van de geschiedenis van inzetten en het gedrag van andere spelers in vorige rondes.

2.2.2 Overgangswaarschijnlijkheidsfunctie

Voor eindige toestands- en actieruimten ($|\mathcal{X}|, |\mathcal{A}| < \infty$) worden de overgangsdynamieken beschreven door een waarschijnlijkheidsfunctie ($p : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0, 1]$), waarbij ($p(x'|x, a)$) de waarschijnlijkheid vertegenwoordigt om over te gaan naar toestand (x') gegeven de huidige toestand (x) en actie (a).

2.3 Beleid en Verwachte Waarden

2.3.1 Beleid

In RL is een beleid de strategie die een agent volgt om beslissingen te nemen. Het bepaalt welke actie een agent moet uitvoeren, gegeven de huidige toestand van de omgeving. Een beleid in reinforcement learning kan op twee manieren worden gedefinieerd:

- **Deterministisch Beleid:**
  $\pi : \mathcal{X} \to \mathcal{A}$, waarbij $a_t = \pi(x_t)$
  Voor elke toestand x_t schrijft het beleid exact één actie a_t voor.
- **Stochastisch Beleid:**
  $\pi : \mathcal{X} \times \mathcal{A} \to [0, 1]$, waarbij $\pi(a|x)$ de waarschijnlijkheid geeft van het kiezen van actie a in toestand x
  Voor een gegeven toestand x definieert het beleid een waarschijnlijkheidsverdeling over mogelijke acties.

2.3.2 Verwachte Waarden

De verwachtingswaarde $\mathbb{E}[X]$ (Expected value), of het gemiddelde, van een willekeurige variabele X is een manier om het gemiddelde resultaat te berekenen dat je zou verwachten als je een groot aantal experimenten uitvoert. Bijvoorbeeld, als X een dobbelsteenworp vertegenwoordigt, dan is $\mathbb{E}[X]$ het gemiddelde van de uitkomsten 1, 2, 3, 4, 5, en 6, wat gelijk is aan 3,5. De conditionele verwachting ($\mathbb{E}[X|Y]$) geeft de verwachte waarde van (X) gegeven (Y).

2.4 Leerparameters in Reinforcement Learning

Bij reinforcement learning spelen verschillende hyperparameters een cruciale rol in het leerproces van de agent. Drie van de belangrijkste parameters zijn het leerpercentage (α), de kortingsfactor (γ), en de exploratieparameter (ϵ). Deze worden hieronder uitgelegd.

2.4.1 Leerpercentage

Het leerpercentage (α) bepaalt hoe sterk nieuwe informatie wordt gewogen ten opzichte van bestaande kennis.

De waarde van α ligt tussen 0 en 1:

- Als $\alpha = 0$: De agent leert niets; bestaande kennis blijft onveranderd.
- Als $\alpha = 1$: Alleen nieuwe informatie wordt gebruikt; bestaande kennis wordt genegeerd.
- Voor $0 < \alpha < 1$: Oude en nieuwe informatie worden gecombineerd, wat doorgaans de voorkeur heeft.

2.4.2 Kortingsfactor

De kortingsfactor (γ) bepaalt hoe belangrijk toekomstige beloningen zijn in vergelijking met onmiddellijke beloningen. Het beïnvloedt de totale beloning die de agent probeert te maximaliseren. De totale beloning wordt gedefinieerd als:

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.2}$$

Waarbij r_t de beloning is die ontvangen wordt op tijdstip t. De waarde van γ varieert meestal tussen 0 en 1:

- Als $\gamma = 0$: Alleen directe beloningen worden overwogen.
- Als $\gamma = 1$: Toekomstige beloningen zijn even belangrijk als directe beloningen.
- Voor $0 < \gamma < 1$: Toekomstige beloningen worden gedisconteerd, met een lagere waarde naarmate ze verder in de toekomst liggen.

2.4.3 Exploratieparameter

De exploratieparameter (ϵ) wordt gebruikt in de epsilon-greedy strategie om een balans te vinden tussen exploratie (het verkennen van nieuwe acties) en exploitatie (het uitvoeren van de momenteel beste actie). De strategie werkt als volgt. Met kans ϵ: Kies een willekeurige actie (exploratie). Anders kiest de agent de actie die momenteel de hoogste geschatte Q-waarde heeft (exploitatie).

De waarde van ϵ bepaalt het gedrag van de agent:

- Als $\epsilon = 0$: De agent exploiteert alleen, wat kan leiden tot suboptimale oplossingen.
- Als $\epsilon = 1$: De agent verkent alleen, zonder gericht gebruik van kennis.

2.5 Waarde-functies

De toestandswaarde-functie geeft aan hoe goed een bepaalde toestand is, terwijl de Q-functie aangeeft hoe goed een actie in een bepaalde toestand is.

2.5.1 Toestandswaarde-functie

De toestandswaarde-functie ($V^\pi : \mathcal{X} \to \mathbb{R}$) onder beleid ($\pi$) wordt gedefinieerd als:

$$V^\pi(x) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid x_0 = x \right] \tag{2.3}$$

Deze functie geeft de verwachte waarde van de totale beloning die een agent zal ontvangen vanaf de toestand x.

2.5.2 Q-functie

De Q-functie ($Q^\pi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$) onder beleid ($\pi$) wordt gedefinieerd als:

$$Q^\pi(x, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid x_0 = x, a_0 = a \right] \tag{2.4}$$

Deze functie geeft de verwachte waarde van de totale beloning die een agent zal ontvangen vanaf de toestand x en na het nemen van actie a.

2.6 Leerstrategieën

2.6.1 Model-Based vs Model-Free Learning

Model-Based Learning

Bij model-based learning construeert de agent expliciet een model ($\hat{p}$) van de overgangsdynamiek en beloningsfunctie van de omgeving ($\hat{r}$). Het geleerde model benadert:

- Overgangsfunctie: $\hat{p}(x'|x, a) \approx p(x'|x, a)$
- Beloningsfunctie: $\hat{r}(x, a) \approx r(x, a)$

De agent kan dit model gebruiken voor:

- *Planning*: Het simuleren van trajecten zonder interactie met de omgeving.
- *Counterfactual reasoning*: Het evalueren van acties die niet daadwerkelijk zijn uitgevoerd.

Model-Free Learning

Model-free methoden leren waardefuncties of beleidsregels direct uit ervaring, zonder een expliciet model te construeren. Deze methoden werken door:

- Directe updates van waarde-schattingen.
- Beleidsverbetering op basis van gesamplede trajecten.
- Geen expliciete representatie van overgangsdynamiek.

2.6.2 On-Policy vs Off-Policy Learning

On-Policy Learning

On-policy methoden leren over het beleid dat momenteel wordt uitgevoerd. De update van de waardefunctie volgt:

$$\text{Target} = r + \gamma \mathbb{E}_{a' \sim \pi(\cdot|x')}[Q(x', a')]$$

waarbij π zowel:

- Het gedragsbeleid (dat ervaringen genereert) als
- Het doelbeleid (dat wordt geleerd) is.

Off-Policy Learning

Off-policy methoden leren over één beleid (π) terwijl een ander beleid (μ) wordt gevolgd. Dit omvat:

- *Gedragsbeleid* (μ): Gebruikt voor exploratie.
- *Doelbeleid* (π): Het beleid dat wordt geleerd.

Hoofdstuk 3

Kenmerken van specifieke Algoritmes

3.1 Q-Learning

Q-learning, geïntroduceerd door Chris Watkins in 1989, is een van de belangrijkste vooruitgangen binnen reinforcement learning. Dit model-vrije, off-policy algoritme is een van de meest gebruikte algoritmen binnen reinforcement learning vanwege zijn eenvoud en effectiviteit.

3.1.1 Proces

Het proces van het Q-learning-algoritme, zoals weergegeven in **Algoritme 1** en de flowchart in **Figuur 3.1**, begint met het opstellen van een Q-tabel. Deze tabel bevat de Q-waarden voor alle combinaties van toestanden en acties. Aan het begin zijn alle waarden ingesteld op nul.

Vervolgens start het spel, waarbij de agent de volgende actie bepaalt. Hierbij heeft de agent twee mogelijkheden:

- Exploratie: De agent voert een willekeurige actie uit om nieuwe informatie te verkennen.
- Exploitatie: De agent selecteert een actie op basis van de bestaande Q-tabel, waarbij de actie met de hoogste Q-waarde in de huidige toestand wordt gekozen.

De keuze tussen exploratie en exploitatie wordt bepaald door de parameter ϵ. De kans dat de agent een willekeurige actie uitvoert (exploratie) is gelijk aan ϵ. Aan het begin van

de training is ϵ gelijk aan 1, en deze waarde neemt exponentieel af naarmate de training vordert. Tegen het einde van de training is ϵ vrijwel 0.

Nadat een actie is uitgevoerd, ontvangt de agent een beloning. Op basis van deze beloning wordt de Q-waarde voor de combinatie van de uitgevoerde actie en de huidige toestand bijgewerkt in de Q-tabel zoals te zien is in formule (3.1). Dit proces wordt herhaald totdat de training is voltooid.

Algorithm 1 Q-Learning Algoritme

Initialisatie: Stel $Q(s, a)$ willekeurig in voor alle toestanden s en acties a
for elke episode **do**
  Initialiseer begin-toestand s
  while s is niet een terminale toestand **do**
    Kies actie a in toestand s op basis van een beleid π
    Voer actie a uit, observeer beloning r en de volgende toestand s'
    Update de Q-waarde:

$$Q(x, a) \leftarrow Q(x, a) + \alpha \left[r + \gamma \max_{a'} Q(x', a') - Q(x, a) \right] \tag{3.1}$$

    $s \leftarrow s'$
  end while
end for

3.1.2 Beperkingen

Een van de grootste beperkingen van Q-learning is schaalbaarheid. De toestandsruimte neem exponentieeel toe wanneer het aantaal dimensies toeneemt:

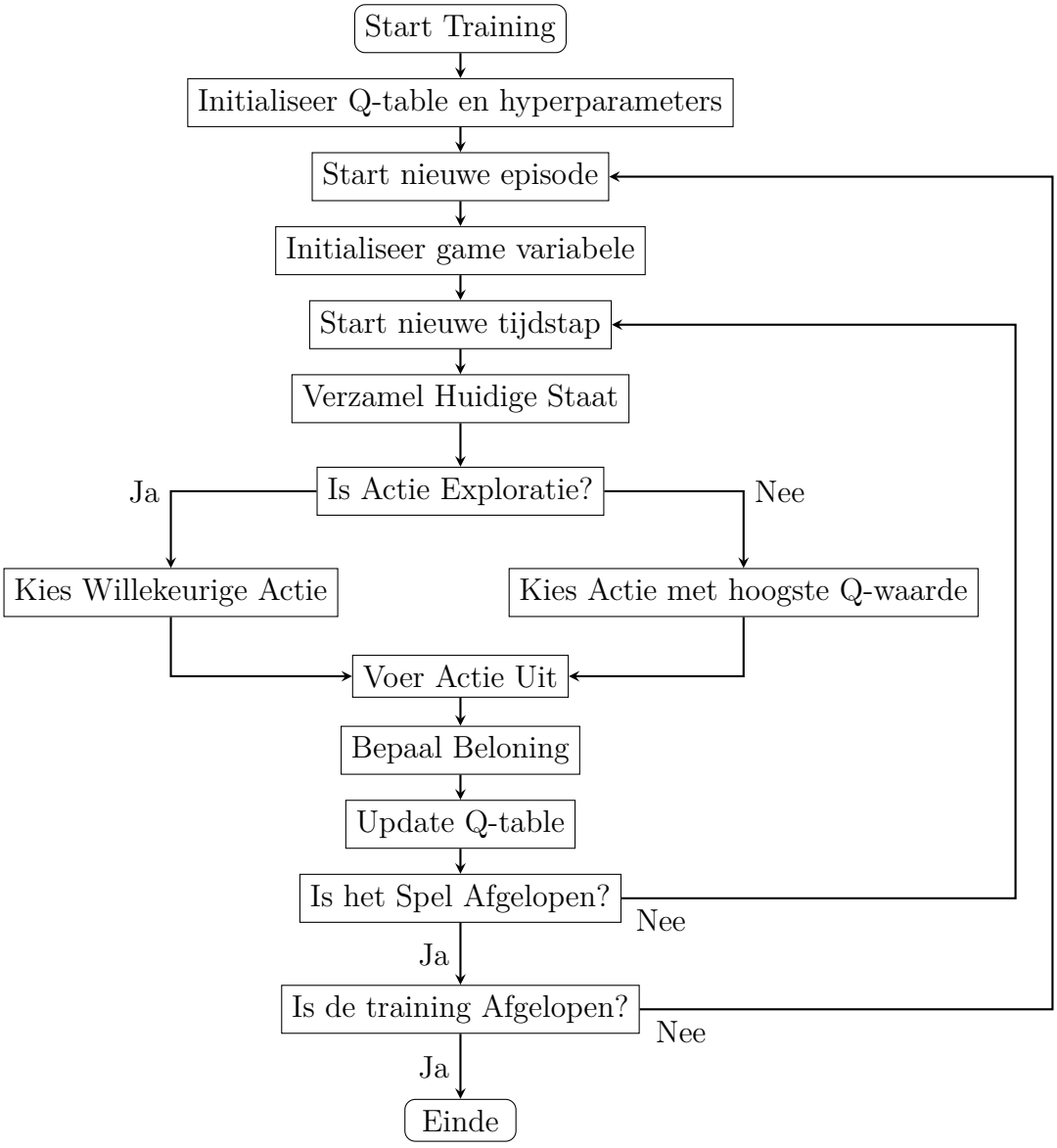
$$|\mathcal{S}| = d^n \tag{3.2}$$

waarbij d het aantal mogelijke waarden per dimensie is en n het aantal dimensies.

Dit leidt tot hoge eisen aan geheugen en rekenkracht:

$$\text{Complexiteit} = O(|\mathcal{S}| \times |\mathcal{A}|) \tag{3.3}$$

3.1.3 Toepassingen



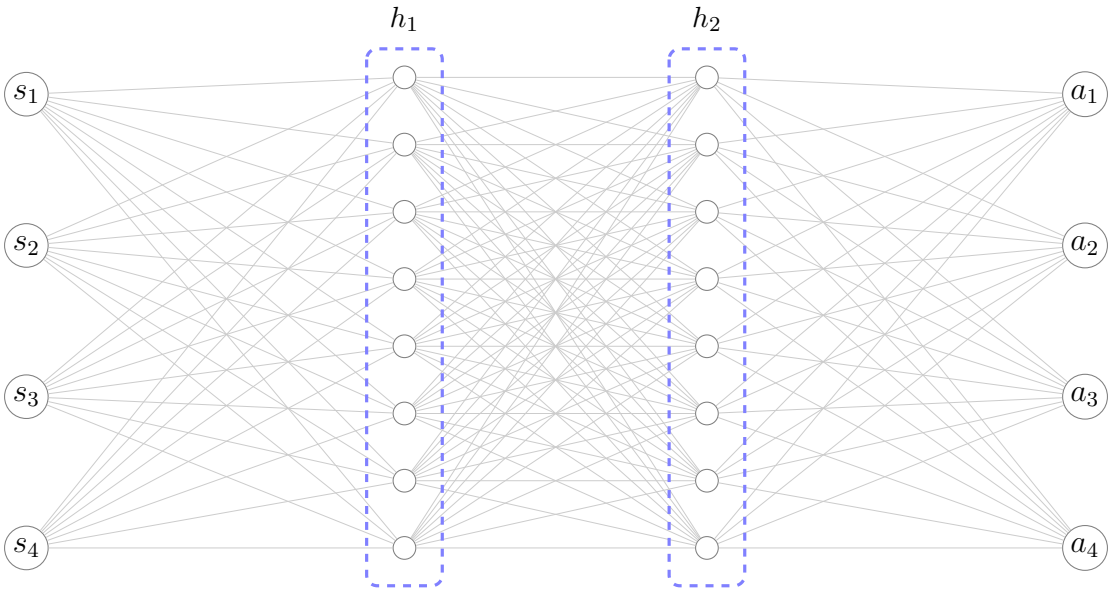
Figuur 3.1: Flowchart van het Q-Learning Algoritme

3.2 Deep Q-Network

Deep Q-Network (DQN), geïntroduceerd door DeepMind in 2013 combineert Q-learning met diepe neurale netwerken. Deze innovatie maakte het mogelijk om reinforcement learning toe te passen op problemen met grootte toestandruimtes, zoals pixels van video-spellen.

3.2.1 Neuraal Netwerk

Het DQN maakt gebruikt van neurale-netwerkarchitectuur die is ontworpen om de optimale actie-waarde functie te benaderen. De netwerkarchitectuur, geïllustreerd in Figuur 3.2, is een typisch neuraal netwerk in DQN.



Figuur 3.2: Diagram van een typisch neuraal netwerk in een DQN met een invoerlaag, twee verborgen lagen en een uitvoerlaag.

Het neurale netwerk wordt gekenmerkt door de volgende architecturale elementen:

De **invoerlaag** bestaat uit een verzameling toestanden $s_1, s_2, \ldots, s_n$, waarbij elke toestand een neuron is en een specifieke toestandseigenschap vastlegt. De **uitvoerlaag** bestaat verzameling acties $a_1, a_2, \ldots, a_n$, waarbij elke actie. De **verborgen lagen** in een neuraal netwerk, genoteerd als $h_1, h_2, \ldots, h_n$, zijn de lagen die zich bevinden tussen de invoerlaag en de uitvoerlaag. Deze verborgen lagen zijn belangrijk voor het leren en modelleren van complexe patronen in de data. Elk neuron in een verborgen laag voert een bewerking uit volgens de formule:

$$h_j = f\left(\sum_i w_{ij} x_i + b_j\right), \tag{3.4}$$

waarbij:

- h_j de output van neuron j is,
- x_i de invoerwaarden zijn,
- w_{ij} de gewichten die de sterkte van de verbindingen tussen de invoer i en het neuron j vertegenwoordigen,
- b_j de bias-term,
- $f(\cdot)$ de activatiefunctie.

In dit geval wordt de Rectified Linear Unit (ReLU) activatiefunctie gebruikt, gedefinieerd als:

$$f(x) = \max(0, x). \tag{3.5}$$

De ReLU introduceert niet-lineariteit door negatieve waarden naar nul te transformeren, terwijl positieve waarden onveranderd blijven. Deze niet-lineariteit stelt het netwerk in staat om complexe functies te benaderen die niet mogelijk zouden zijn met een louter lineaire transformatiestap.

Door de toepassing van n verborgen lagen, wordt een complexe niet-lineaire transformatie uitgevoerd van de oorspronkelijke invoer $\mathbf{x}$ naar de uiteindelijke uitvoer $\mathbf{y}$, als volgt:

$$\mathbf{y} = f_n\left(W_n f_{n-1}\left(W_{n-1} \cdots f_1\left(W_1 \mathbf{x} + \mathbf{b}_1\right) \cdots + \mathbf{b}_{n-1}\right) + \mathbf{b}_n\right), \tag{3.6}$$

waarbij W_i de gewichtsmatrix van laag i en $\mathbf{b}_i$ de bias-term van laag i vertegenwoordigen. Het gebruik van meerdere verborgen lagen met ReLU maakt het netwerk krachtig genoeg om complexe patronen te leren en invoeromstandigheden nauwkeurig te vertalen naar acties of beslissingen.

3.2.2 Proces

Het Q-learning-algoritme, zoals weergegeven in **Algoritme 2** en de flowchart in **Figuur 3.3** breidt het traditionele Q-learning uit door de Q-tabel te vervangen door een neuraal netwerk. Dit netwerk leert de mapping tussen toestanden en Q-waarden voor alle

mogelijke acties. Het proces bestaat uit verschillende componenten die voor de stabiliteit en effectiviteit van het algoritme zorgen. DQN maakt gebruik van experience replay, waarbij ervaringen (toestand, actie, beloning, volgende toestand) worden opgeslagen in een replay buffer en willekeurig worden gebruikt voor training.

Daarnaast wordt een tweede neuraal netwerk, het target network, ingezet om de doelwaarden te berekenen. Dit netwerk wordt regelmatig geüpdatet met de gewichten van het hoofdnetwerk, wat de training verder stabiliseert. Voor visuele inputs implementeert DQN convolutionele lagen die relevante informatie halen uit de pixels, dit maakt het algorithme goed in het verwerken van complexe visuele informatie.

Algorithm 2 Deep Q-learning met Experience Replay (Mnih e.a., 2013)

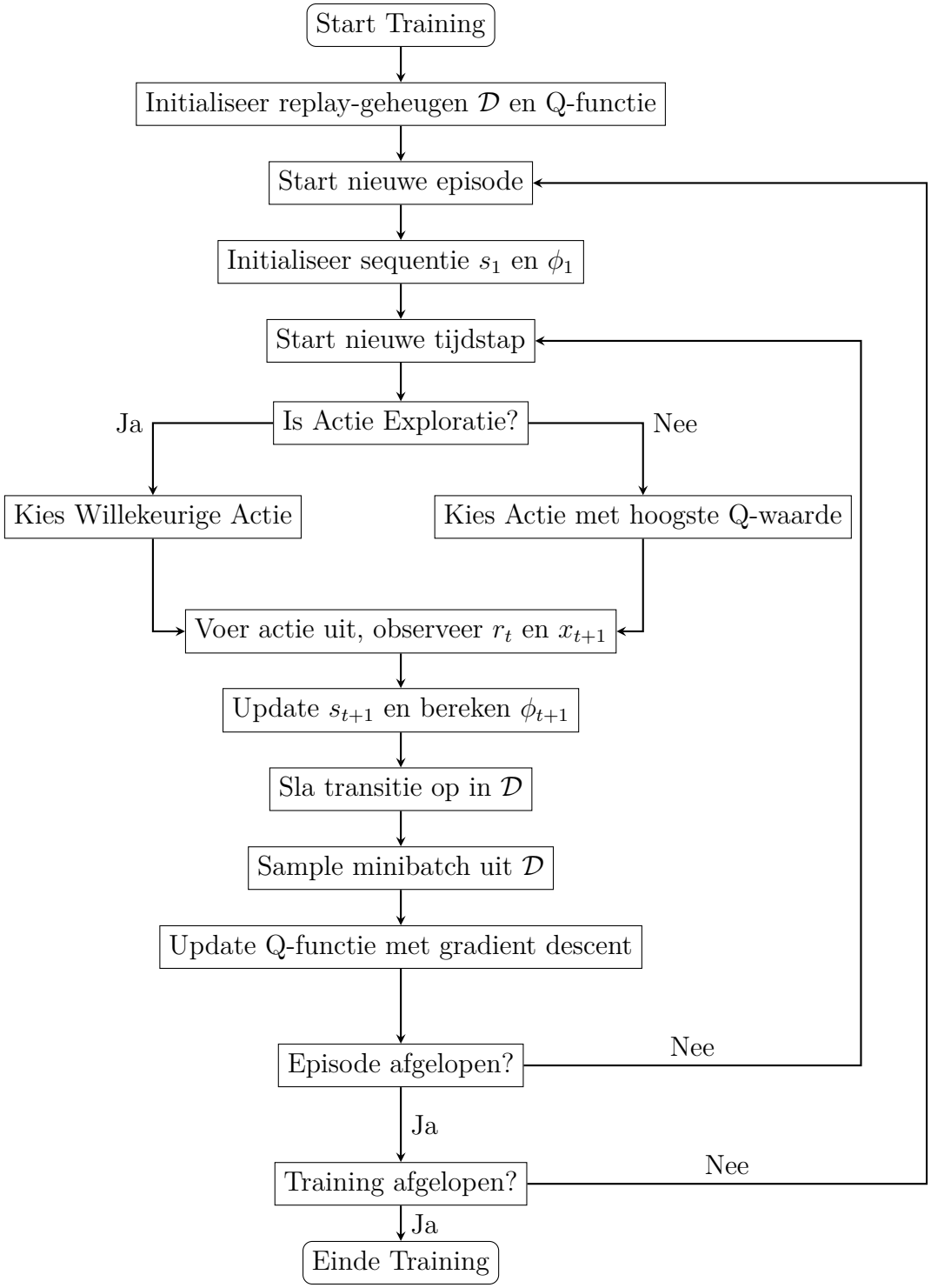
Initialiseer replay-geheugen $\mathcal{D}$ met capaciteit N
Initialiseer de actie-waarde functie Q met willekeurige gewichten
for elke episode $1, \ldots, M$ **do**
  Initialiseer sequentie $s_1 = \{x_1\}$ en preprocess $\phi_1 = \phi(s_1)$
  for $t = 1, \ldots, T$ **do**
    Met kans ϵ: kies een willekeurige actie a_t
    Anders: kies $a_t = \arg\max_a Q^*(\phi(s_t), a; \theta)$
    Voer actie a_t uit in de omgeving en observeer beloning r_t en beeld x_{t+1}
    Stel $s_{t+1} = s_t, a_t, x_{t+1}$ en preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Sla transitie $(\phi_t, a_t, r_t, \phi_{t+1})$ op in $\mathcal{D}$
    Neem een willekeurige minibatch van transities $(\phi_j, a_j, r_j, \phi_{j+1})$ uit $\mathcal{D}$
    Bereken:

$$y_j = \begin{cases} r_j & \text{als } \phi_{j+1} \text{ terminaal is} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{als } \phi_{j+1} \text{ niet terminaal is} \end{cases}$$

    Voer een gradient-descent stap uit op $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

3.2.3 Verbeteringen op Klassiek Q-learning

DQN lost verschillende problemen van Q-learning op. Het neurale netwerken zorgt dat het algoritme in staat om effectief om te gaan met continue en hoog-dimensionale toestandsruimtes. Het netwerk bezit ook sterke generalisatie-eigenschappen, waardoor het patronen kan herkennen en toepassen op ongeziene toestanden. De stabiliteit van het leerproces wordt significant verbeterd door de introductie van experience replay en target networks, wat een belangrijke vooruitgang is ten opzichte van Q-learning methoden.



Figuur 3.3: Flowchart van het Deep Q-learning met Experience Replay Algoritme

3.2.4 Voordelen en Beperkingen

DQN is goed in complexe taken met visuele inputs effectief aan pakken zonder dat handmatige feature-extractie nodig is. Het algoritme heeft goede generalisatie-eigenschappen, waardoor het zich kan aanpassen aan nieuwe situaties binnen het geleerde domein. Deze voordelen worden echter vergezeld door enkele significante beperkingen. DQN is computationeel intensief en vereist grote hoeveelheden trainingsdata om tot goede resultaten te komen. Daarnaast is het algoritme gevoelig voor de keuze van hyperparameters, wat het optimalisatieproces complex maakt. Een andere uitdaging ligt in het verwerken van zeer lange-termijn afhankelijkheden, wat in bepaalde toepassingen een limiterende factor kan zijn.

3.2.5 Toepassingen

DQN heeft zijn effectiviteit bewezen in veel toepassingsgebieden. In de context van videogames heeft het algoritme indrukwekkende resultaten behaald, met name bij het leren spelen van Atari 2600 spellen op menselijk en bovenmenselijk niveau. Bij robotica en motion control biedt DQN nieuwe mogelijkheden voor het ontwikkelen van verfijnde besturingsstrategieën. Ook binnen autonome systemen wordt DQN gebruikt door complexe beslissingsprocessen te optimaliseren. Deze succesvolle toepassingen demonstreren de veelzijdigheid en het potentieel van het algoritme in verschillende praktische contexten.

3.3 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG), ontwikkeld door onderzoekers van DeepMind in 2015, is een algoritme dat zich richt op continue actieruimtes. Als een uitbreiding op de policy gradient methoden en Actor-Critic architecturen, combineert DDPG de voordelen van deterministische beleidsoptimalisatie en neurale netwerken.

3.3.1 Actor-Critic model

Het Actor-Critic model is een model dat kenmerken van policy

De Actor is verantwoordelijk voor het genereren van acties. Hij geeft de optimale actie bepaalt gegeven een specifieke toestand. De Actor word gedefinieerd door een functie:

$$a = \pi_\theta(s) \tag{3.7}$$

De Critic evalueert de kwaliteit van de door de Actor gegenereerde acties. Hij schat

de waarde-functie $Q(s, a)$ of de staat-waarde functie $V(s)$. De Critic levert kritische informatie terug aan de Actor over de effectiviteit van zijn gegenereerde acties:

$$Q(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a] \tag{3.8}$$

Leerproces

Het unieke van het Actor-Critic model is de wederzijdse terugkoppeling tussen beide netwerken:

1. De Actor genereert een actie op basis van de huidige toestand 2. De Critic beoordeelt de kwaliteit van deze actie 3. De Actor wordt bijgestuurd op basis van de feedback van de Critic

De Actor wordt geüpdatet met een gradiënt die de verwachte beloning maximaliseert:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q(s, a)] \tag{3.9}$$

Voordelen

- Geschikt voor zowel discrete als continue actieruimtes - Lagere variantie in gradiënt-schatting - Mogelijkheid tot gelijktijdige optimalisatie van beleid en waarde-inschatting

3.3.2 Proces

3.3.3 Verbeteringen op DQN

3.3.4 Voordelen en Beperkingen

3.3.5 Toepassingen

Algorithm 3 DDPG-algoritme Lillicrap e.a., 2015

Initialiseer willekeurig het critic-netwerk $Q(s, a|\theta^Q)$ en het actor-netwerk $\mu(s|\theta^\mu)$ met gewichten θ^Q en θ^μ.
Initialiseer het target netwerk Q' en μ' met gewichten $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialiseer de replay buffer R
for episode = 1, M **do**
  Initialiseer een willekeurig proces $\mathcal{N}$ voor actie-exploratie
  Ontvang de initiële waarnemingsstatus s_1
  for t = 1, T **do**
    Selecteer actie $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ volgens het huidige beleid en exploratieruis
    Voer actie a_t uit en observeer beloning r_t en nieuwe status s_{t+1}
    Sla de transitie (s_t, a_t, r_t, s_{t+1}) op in R
    Neem een willekeurige minibatch van N transities (s_i, a_i, r_i, s_{i+1}) uit R
    Bereken:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{3.10}$$

    Update de critic door de volgende verliesfunctie te minimaliseren:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{3.11}$$

    Update de actor-policy met behulp van de volgende policy-gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \tag{3.12}$$

    Update de targetnetwerken:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{3.13}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{3.14}$$

  end for
end for

Hoofdstuk 4

Kenmerken van specifieke Computerspellen

Er zijn talloze computerspellen met diverse en uitdagende omgevingen voor de toepassing van reinforcement learning (RL)-algoritmes. Spellen onderling kunnen erg van elkaar variëren onder andere in structuur, dynamiek, complexiteit, en tijdsduur. Al deze aspecten kunnen dergelijke invloed hebben op de effectiviteit van RL-algoritmes. Elk type spel stelt specifieke eisen aan een RL-algoritme, afhankelijk van aspecten zoals de omvang van de toestandsruimte, de regels en beperkingen, en de vereiste strategische vaardigheden.

In dit hoofdstuk worden de specifieke kenmerken van vier computerspellen met elkaar vergeleken: een *auto-racespel* waar de agent het autobesturingssysteem is, *Snake* waar de agent de slang is, *Schaken* waar de agent de schaker is en *Super Mario Bros* waar de agent Mario is. Een overzicht van alle speleigenschappen is te zien in Tabel ??.

4.1 Indeling en Strategische Diepgang van Spellen

Spellen kunnen worden ingedeeld op basis van hun genre en de mate van strategische diepgang die nodig is om succesvol te zijn. Actie- en platformspellen, zoals *Super Mario Bros*, hebben een gemiddelde strategische diepgang. Het doel is obstakels te overwinnen, vijanden te ontwijken of verslaan en tegelijkertijd gouden munten te verzamelen. Dit soort spellen vereist doorgaans korte termijn optimalisatie en directe reacties.

Strategiespellen, zoals *Schaken*, vragen daarentegen om diepgaande planning en vooruitdenken. Hier moet een speler of agent een reeks mogelijke toekomstige toestanden analyseren en anticiperen op de acties van een tegenstander. De strategische diepgang maakt leren complex, omdat beloningen vaak cumulatief en pas aan het einde van het spel duidelijk worden. Dergelijke spellen vereisen geavanceerde reinforcement learning (RL)-algoritmes die langetermijnplanning ondersteunen.

Puzzel-/actiespellen, zoals *Snake*, zijn minder afhankelijk van strategie. Hier draait het om patroonherkenning en korte termijn optimalisatie, waarbij eenvoudige RL-algoritmes voldoende zijn om succesvol te leren. Het doel is bijvoorbeeld appels te verzamelen zonder jezelf te raken, waarbij de beloningsstructuur rechtlijnig is.

Simulatie- en racegames, zoals een racespel met een zelfrijdende auto, richten zich op efficiënt en veilig navigeren over een parcours. Hier ligt de nadruk op het optimaliseren van gedrag in een gesimuleerde omgeving, vaak zonder de noodzaak van complexe planningsstrategieën.

Deze variatie in genres en strategische eisen bepaalt welk type RL-algoritme het meest geschikt is voor een spel. Complexere spellen met hogere strategische diepgang vereisen geavanceerdere algoritmes, terwijl eenvoudigere spellen vaak volstaan met directe responsmechanismen.

4.2 Indeling van Spellen

Super Mario Bros valt binnen het genre van actie- en platformspellen. Het doel van het spel is over hindernissen springen en vijanden te ontwijken en verslaan en tegelijkertijd gouden munten te verzamelen. *Schaken* daarentegen is een strategiespel, dat volledig turn-based is en gericht op denkvermogen, vooruitdenken en strategische planning. *Snake* wordt vaak als een puzzel-/actiespel beschouwd, waarbij het doel is een appel te eten terwijl je niet jezelf raakt; hier is patroonherkenning belangrijk. Een zelfrijdende auto in een racespel valt binnen het genre van simulatie en racegames. Het draait om het efficiënt en veilig navigeren over een parcours. Dit wordt vaak gebruikt bij offline racespellen waar je tegen de computer speelt.

4.3 Strategische Diepgang

De mate van strategische diepgang in een spel is een van de belangrijkste factoren die bepalen welk type RL-algoritme geschikt is. Strategie verwijst naar het vermogen om vooruit te denken en acties te plannen die op lange termijn voordelig zijn. Dit varieert sterk tussen spellen.

Strategische spellen, zoals *Schaken*, vereisen vereisen dat een agent ver vooruit denkt en een reeks mogelijke toekomstige toestanden analyseert. Hier is langetermijnplanning essentieel. De agent moet niet alleen rekening houden met de huidige toestand, maar ook anticiperen op de mogelijke acties van een tegenstander en de daaropvolgende uitkomsten. Bij strategische spellen is het leren complex, omdat beloningen vaak cumulatief en pas aan het einde van het spel duidelijk worden.

Aan de andere kant zijn er spellen zoals *Snake*, waarin strategie een veel minder be-

langrijke rol speelt. In deze spellen zijn acties vaak gebaseerd op eenvoudige regels en is de beste keuze meestal direct duidelijk. Het succes van een speler hangt hier voornamelijk af van korte termijn optimalisatie. Dergelijke spellen vereisen relatief eenvoudige RL-algoritmes, die zijn ontworpen om direct te reageren op beloningen of straffen zonder complexe planningsstrategieën. De eenvoudige structuur en beloningsmechanismen maken het leerproces rechtlijnig en efficiënt.

4.4 Beslissingsdynamiek en Tijdgevoeligheid

De snelheid waarmee de omgeving van een spel verandert, bepaalt in grote mate hoe moeilijk het is voor een RL-agent om effectief te leren en te reageren.

De beslissingsdynamiek verschilt sterk tussen de spellen. *Super Mario Bros* vereist snelle real-time beslissingen. De agent moet op het juiste moment springen of een vijand ontwijken, en timing is hierbij cruciaal. Bij *Schaken* is er juist geen tijdsdruk; de agent kan lang "nadenken" over elke zet. *Snake* zit er tussenin: hoewel het spel niet zo snel is als *Mario*, zit er wel een kleine tijdsdruk achter, maar dit is meestal verwaarloosbaar. Timing en patroonherkenning worden belangrijker naarmate het spel vordert. Bij een zelfrijdende auto in een racespel ligt de beslissingsdynamiek real-time. Hier zijn snelheid en precisie van beslissingen belangrijk, omdat milliseconden het verschil kunnen maken tussen een succesvolle race en een botsing.

4.5 Complexiteit

De complexiteit van de regels en doelen varieert sterk. *Super Mario Bros* heeft relatief eenvoudige regels: de agent moet vijanden vermijden en verslaan, munten verzamelen, en het einde van het level bereiken. De complexiteit van de vijanden en het terrein neemt echter toe naarmate de levels moeilijker worden. *Schaken* heeft relatief eenvoudige basisregels: zes verschillende stukken met elk unieke bewegingsmogelijkheden. Het doel, het schaakmat zetten van de tegenstander, vereist echter strategisch inzicht en planning. Dit maakt *Schaken* bijzonder uitdagend voor een RL-agent vanwege de enorme toestandsruimte en de langetermijnplanning die nodig is. *Snake* heeft zeer eenvoudige regels: de agent moet voedsel verzamelen en mag niet botsen met de muur of zichzelf. De uitdaging ligt in de toenemende snelheid en lengte van de slang. Een zelfrijdende auto in een racespel heeft daarentegen te maken met complexe regels die gebaseerd zijn op realistische fysica. Het doel is simpel: zo snel mogelijk de finish bereiken.

4.5.1 Regels en Beperkingen

De complexiteit en het aantal regels in een spel spelen een cruciale rol in de uitdaging die een RL-agent tegenkomt.

Spellen met veel regels en vaste patronen

Spellen zoals 4-op-een-rij of boter-kaas-en-eieren hebben een voorspelbare structuur en strikte regels. De mogelijke zetten en uitkomsten zijn beperkt, wat het spel eenvoudiger maakt om te modelleren. RL-algoritmes kunnen hier profiteren van waarschijnlijkheidsmodellen en gestructureerde planning. De voorspelbaarheid van deze spellen vermindert de onzekerheid in het leerproces. Een RL-agent kan relatief eenvoudig een optimale strategie leren door alle mogelijke acties te analyseren en te kiezen voor de meest belonende uitkomst.

Spellen met weinig regels en veel vrijheid

Spellen zoals GTA of Call of Duty bieden een grote mate van keuzevrijheid. De speler kan vrij bewegen in een open wereld, interacties aangaan en talloze acties uitvoeren. Deze spellen hebben een enorme toestandsruimte, die driedimensionaal en dynamisch is. Dergelijke spellen vereisen een flexibel en adaptief RL-algoritme. Het is onrealistisch voor een RL-agent om alle mogelijke acties en toestanden volledig te doorzoeken. Algoritmes zoals Proximal Policy Optimization (PPO) zijn hier geschikt. PPO gebruikt stochastische beleidsmodellen en leert door te experimenteren met acties, waarbij het snel aanpassingen kan maken op basis van feedback.

4.6 Dynamiek en Tijdgevoeligheid

De snelheid waarmee de omgeving van een spel verandert, bepaalt in grote mate hoe moeilijk het is voor een RL-agent om effectief te leren en te reageren.

4.6.1 Turn-based spellen

Spellen zoals *Schaken* of Monopoly bieden de speler voldoende tijd om de optimale actie te berekenen. Omdat de omgeving niet continu verandert, kan een RL-algoritme worden ingezet om een uitgebreide analyse te maken van alle mogelijke uitkomsten van een actie. Dit type algoritme is bijzonder effectief in spellen waar de agent kan profiteren van gestructureerde planning en voorspelbare omgevingen.

4.6.2 Realtime spellen

In spellen zoals *Mario Bros* of Tetris veranderen de omstandigheden continu. Obstakels bewegen, vijanden verschijnen en de tijdsdruk vereist snelle besluitvorming. Voor deze spellen zijn algoritmes nodig die snel leren en direct reageren, met neurale netwerken, waardoor het algoritme in real-time beslissingen kan nemen op basis van eerdere ervaringen.

4.7 Beloningsstructuur

Beloningen vormen de kern van RL en bepalen hoe een agent leert. De manier waarop beloningen worden toegekend, varieert sterk tussen spellen.

4.7.1 Directe beloningen

Spellen zoals *Snake* bieden onmiddellijke feedback. Elke actie resulteert direct in een beloning (zoals punten voor het eten van voedsel) of een straf (zoals botsingen). RL-algoritmes die afhankelijk zijn van directe beloningen werken goed in deze context, omdat ze snel leren welke acties voordelig zijn.

4.7.2 Cumulatieve beloningen

In strategische spellen zoals *Schaken* worden beloningen vaak pas aan het einde van het spel toegekend. Dit vereist dat de agent leert om acties te nemen die op lange termijn voordelig zijn. Het leren wordt complexer omdat de agent beloningen moet toeschrijven aan acties die mogelijk vele stappen eerder werden ondernomen.

Hoofdstuk 5

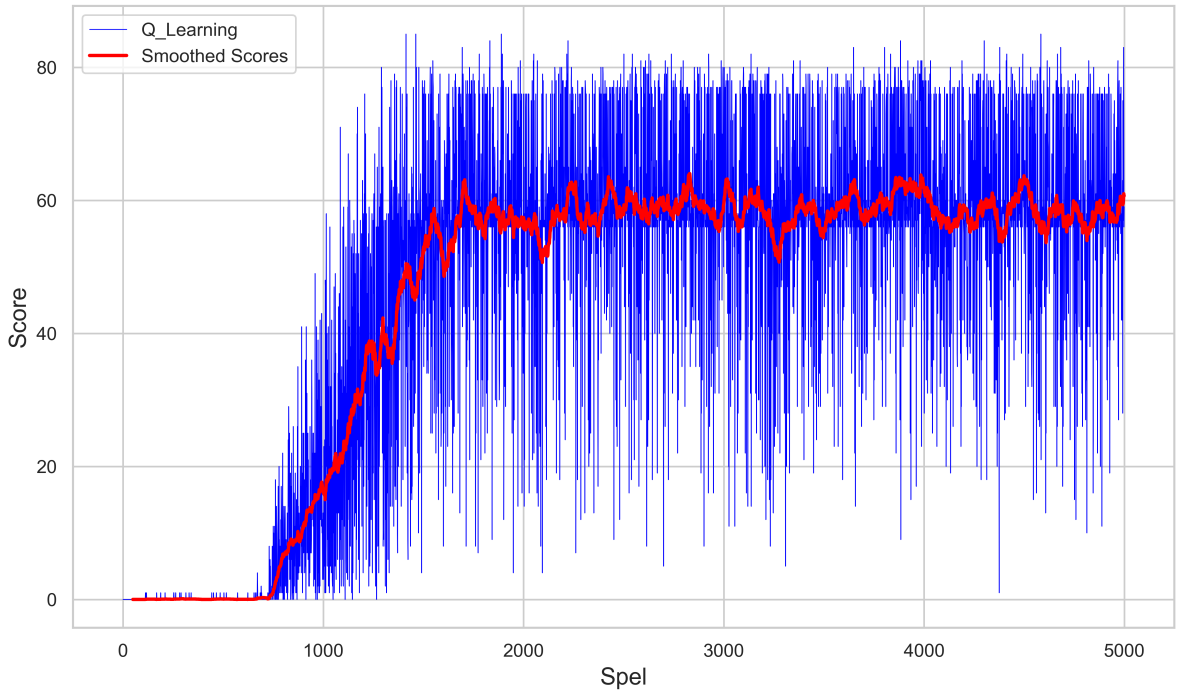
Onderzoeksmethoden

5.1 Q-Learning

Hoofdstuk 6

Analyse en Resultaten

6.1 Snake



Hoofdstuk 7

Conclusie

Hoofdstuk 8

Discussie

Appendix

Bibliografie

Introduction to RL. (2018). https://spinningup.openai.com/en/latest/user/introduction.html

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. https://arxiv.org/abs/1509.02971

Millington, I., & Funge, J. (2009). *Artificial Intelligence for Games, Second Edition* (2nd). Morgan Kaufmann Publishers Inc.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*. https://arxiv.org/abs/1312.5602

Puterman, M. L. (1994, april). *Markov Decision processes*. https://doi.org/10.1002/9780470316887

Sanz, M. (2024). Introduction to Reinforcement Learning (Part 3): Q-Learning with Neural Networks Algorithm (DQN) [Accessed: 2024-12-03].

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic Policy Gradient Algorithms. *Proceedings of the 31st International Conference on Machine Learning (ICML)*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book.

Tang, Y. (2021). *Reinforcement Learning: New Algorithms and An Application for Integer Programming* (tech. rap.).