

Ontwerpdocument Pacman demonstrator

REINFORCEMENT LEARNING DEMONSTRATOR - PACMAN

Finn Alberts, Laurent Dassen en Noud Wijngaards
ZUYD HOGESCHOOL | HBO ICT



Inhoud

1 Inleiding.....	3
2 Gebruikte tools en Pacman-versie	3
2.1 Python	3
2.2 Gym	3
2.3 Baselines3	3
2.4 Gebruikte Pacman-versie.....	3
2.5 Matplotlib	4
3 Globaal overzicht van werking	4
4 <i>Main.py</i> -bestand	5
5 Wijzigingen in Pacman-game	5
6 Gym-environment	6
6.1 <i>__init__(self)</i> -functie.....	6
6.2 <i>step(self, action: int)</i> -functie	6
6.3 <i>_get_reward(gamestate, action)</i> -functie	7
6.4 <i>reset(self)</i> -functie	7
6.5 <i>render(self, mode='human', close=False)</i> -functie	8
7 <i>Graph.py</i> -bestand	8
8 Toelichting op het doel van de AI	8
9 Verwijzingen.....	9

1 Inleiding

Het programma geeft de mogelijkheid tot het spelen van Pacman middels reinforcement learning. Tijdens het spel van Pacman zijn de ghosts uitgeschakeld, omdat deze een te grote complexiteit geven voor het succesvol spelen van het spel. Het doel van de AI is het behalen van het eerste level door alle pellets te verzamelen.

Het programma kan worden gestart middels het commando: *python main.py*. Dit commando start tevens ook twee grafieken die progressie van de behaalde in-game score en de behaalde reward tonen.

2 Gebruikte tools en Pacman-versie

2.1 Python

Binnen het project wordt er gebruikgemaakt van de Python-programmeertaal voor het opzetten van de Pacman-game en het realiseren van de AI-agent. Een van de redenen dat voor Python is gekozen is dat voor Python veel verschillende gerealiseerde versies van Pacman te vinden zijn op het internet. Het gebruik van Python biedt daarnaast ook de keuze uit een groot aantal libraries. Deze libraries kunnen worden gebruikt voor het realiseren en ondersteunen van de AI-agent. Er wordt vanwege de comptabiliteit van verschillende libraries gebruik gemaakt van Python versie 3.9.

2.2 Gym

Om het gebruik van reinforcement learning te realiseren wordt gebruikgemaakt van een toolkit 'Gym' (OpenAI, sd). Deze is gemaakt door OpenAI. Gym biedt de mogelijkheid voor het realiseren van environments waarin agents kunnen worden getraind. Er is voor Gym gekozen omdat dit de standaard is binnen Python voor het creëren van environments voor reinforcement learning.

2.3 Baselines3

Voor het implementeren van de reinforcement learning algoritmes wordt gebruik gemaakt van Baselines3 (Raffin, et al., 2021). Er is voor Baselines3 gekozen vanwege de grote eenvoud van de implementatie, ondanks dat dit ten koste gaat van functionaliteit. De library ondersteunt een groot aantal algoritmen, waaronder PPO en DQN. Daarnaast werkt deze library samen met OpenAI Gym.

2.4 Gebruikte Pacman-versie

Er is gekozen om niet te werken met de originele Pacman-game van het Nintendo Entertainment System (NES), omdat dit een aantal onnodige complexiteiten met zich meebrengt. Een van deze complexiteiten is het uitlezen van de juiste variabelen uit de environment (zoals de positie van Pacman, de huidige score, enzovoorts). Deze moeten vanuit de memory (werkgeheugen) worden uitgelezen aan de hand van memory-adressen. Ditzelfde geldt ook voor het geven van input en de daar bijbehorende memory-adressen.

In plaats daarvan is gebruikgemaakt van een Pythonversie van Pacman door Pacmancode (Pacmancode, 2022). Het voordeel van het gebruik van deze versie van Pacmancode is dat de volledige sourcecode beschikbaar is, inclusief documentatie. Dit zorgt voor overzichtelijkheid gedurende het aanpassen van de code en tevens gedurende het realiseren van de AI-agent. De code is daarnaast ook van goede kwaliteit waardoor nauwelijks tot geen aanpassingen gemaakt hoeven te worden aan de structuur van de code.

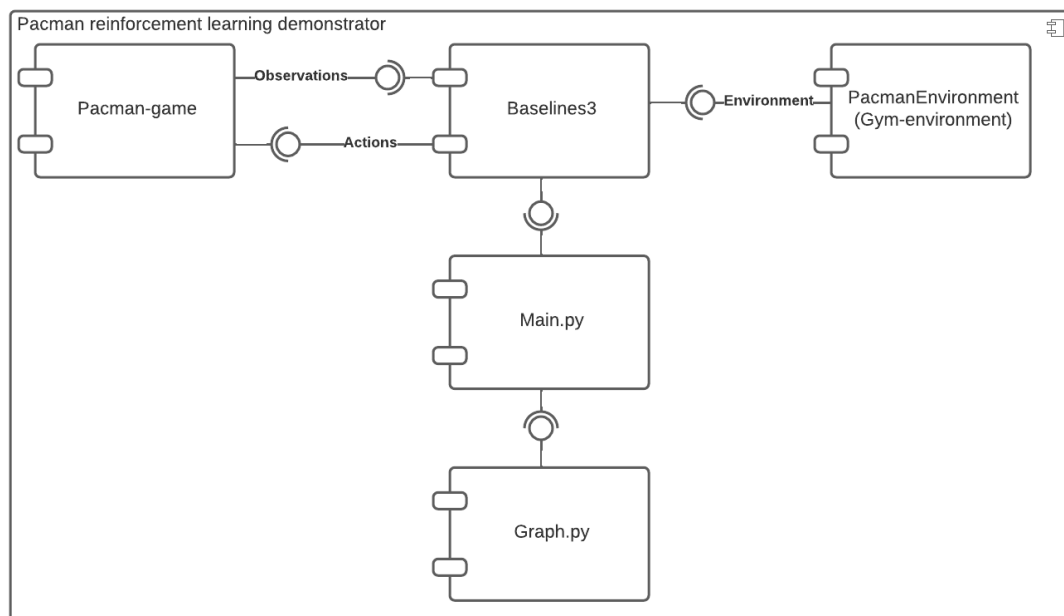
Verder is deze versie met betrekking tot zowel het functioneren als het visuele aspect vrijwel geheel gelijk aan de originele Pacman. Er hoeven dus ook geen aanpassingen gemaakt te worden betreffende de spelregels en het functioneren van de game.

Ook kunnen, doordat er vanuit de sourcecode kan worden gewerkt, variabelen (zoals de positie van Pacman, de huidige score, enzovoorts) direct worden uitgelezen en in andere Python-scripts kunnen worden gebruikt. Hiervoor hoeft niet te worden gewerkt met memory-adressen. Hetzelfde geldt voor het geven van input.

2.5 Matplotlib

Matplotlib (Matplotlib, 2021) is een library die kan worden gebruikt voor het tekenen van grafieken. Binnen de demonstrator wordt deze gebruikt voor het tekenen van de grafieken voor de behaalde rewards en de behaalde in-game scores.

3 Globaal overzicht van werking



Figuur 1 Componentendiagram

De applicatie bestaat grofweg uit twee onderdelen. De verbinding tussen deze twee zit in *main.py*. Vanuit *main.py* wordt *graph.py* gestart en wordt het reinforcement learning algoritme gestart (PPO). Deze lopen parallel aan elkaar.

Graph.py is verantwoordelijk voor het visualiseren van de behaalde rewards en in-game scores gedurende het trainen van de agent.

Voor het reinforcement learning algoritme wordt gebruikgemaakt van *baselines3*. Deze library maakt het erg eenvoudig om deze algoritmes te gebruiken. Om deze library te gebruiken, moet ook gebruik worden gemaakt van een OpenAI Gym-environment. In een Gym-environment worden de action space (uitvoerbare acties) en observation space (observaties van de AI) gedefinieerd, evenals de reward-functie en hoe het spel opnieuw kan worden gestart (het spel wordt opnieuw gestart, zodra Pacman gepakt wordt door een spookje). Het Gym-environment voor Pacman staat in *environment.py*.

Baselines3 voert met behulp van de PacmanEnvironment acties uit in de Pacman-game. Hiervoor worden de pijltjes op het toetsenbord ingedrukt. Wanneer de applicatie dus wordt gestart is het niet

mogelijk om deze te minimaliseren. De agent zal namelijk de pijltoetsen blijven gebruiken op hoog tempo, waardoor het navigeren op de computer zo goed als onmogelijk is.

Mocht het voorkomen dat de applicatie geminimaliseerd start, dan kan het zijn dat het spel zich nog in de ‘pauze’ modus bevindt. Met de spatietoets kan het spel dan worden gestart.

4 *Main.py*-bestand

Het *main.py*-bestand is het bestand waarin de verschillende stappen worden gestart. Wanneer dit script wordt gerund, worden *main()* en *graph()* gelijktijdig gerund. De *graph()*-functie staat in het *graph.py*-bestand.

De *main()*-functie bestaat uit een aantal stappen. Allereerst wordt in regel 21 de Gym-environment geregistreerd en wordt aan de hand hiervan een model gemaakt in regel 22. Vervolgens wordt gecheckt of er al een model staat opgeslagen. Indien dit zo is, wordt deze ingeladen. Vervolgens komt het programma in een ‘oneindige’ lus terecht, waarbinnen voor 20 000 stappen wordt geleerd en vervolgens het model wordt opgeslagen. Daarna wordt het model weer geladen voor een volgende run van 20 000 stappen.

5 Wijzigingen in Pacman-game

In de Pacman-game van Pacmancode zijn enkele wijzigingen aangebracht en enkele toevoegingen gedaan voor het uitlezen van de variabelen. Al deze wijzigingen zijn in de code aangegeven met een comment startende met de prefix “*RLD* – “.

Allereerst is de snelheid waarop het spel gespeeld wordt verhoogd. Dit gebeurt op regel 102 van *run.py*. Hierdoor kan de AI sneller trainen.

Daarnaast is op regel 196 van *run.py* het loggen van fruits die worden spawned uitgeschakeld.

In *run.py* zijn vier functies toegevoegd voor het uitlezen van de *gamestate*. De *gamestate* is een dictionary waarin de huidige staat van het spel wordt opgeslagen. Deze functies zijn als volgt:

- *receive_gamestate(self)*: algemene functie voor het ophalen van de *gamestate*. Deze wordt gereturned in de vorm van een dictionary. De keys inclusief betekenis zijn te zien in Tabel 1.

Tabel 1 Definitie *gamestate*-keys

Key	Beschrijving
lives	Integer. Aantal levens.
is_alive	Boolean. True als Pacman nog leeft, anders False.
score	Integer. De behaalde in-game score.
level	Integer. Het huidige level. Er wordt geteld vanaf 0 (het eerste level is dus level 0)
map	2D-list. Bevat de observatie voor de AI, zie de <i>get_map(self)</i> -functie hieronder.
min_ghost_distance	Float. De afstand tot de ghost die het dichtste bij Pacman staat. De afstand wordt uitgedrukt in aantal hokjes (de map is 31 bij 28 hokjes)

- *find_closest_ghost(self)*: functie voor het verkrijgen van de afstand tot de ghost welke het dichtste bij Pacman staat. Deze functie wordt door *receive_gamestate(self)* aangeroepen.
- *get_map(self)*: functie voor het ophalen van de map in een 31 bij 28 grid. Hierbij staat 0 voor een lege cell, 1 voor een muur, 2 voor Pacman, 3 voor een (power)pellet of fruit en 4 voor een ghost. Deze map wordt gebruikt als observatie voor de AI.
- *get_map_walls(self)*: leest het *maze1.txt*-bestand uit. Dit bestand bevat de layout van het speelveld.

Ook zijn in *run.py* de ghosts uitgeschakeld. Dit gebeurt op regels 64 en 155-159 van *run.py* en regels 136-140 van *ghosts.py*.

Tot slot is bij alle imports, waarbij scripts worden geïmporteerd van de game zelf, "*Pacman_Game*." De reden hiervoor is dat het spel in een map is geplaatst (*Pacman_Game*) en de verwijzing naar deze map moet worden gemaakt. Deze wijzigingen zijn niet gecomet met de "RLD –"-prefix.

6 Gym-environment

In *environment.py* is de Gym-environment gecreëerd. Binnen iedere Gym-environment zijn een *__init__(self)*, *step(self, action: int)*, *reset(self)*, *render(self)* vereist.

6.1 *__init__(self)*-functie

In de *__init__(self)*-functie worden een aantal dingen ingesteld en gedefinieerd. Allereerst worden in regels 12 en 15 de action space (uitvoerbare acties) en observation space (observatie van de AI) gedefinieerd. De action space wordt gedefinieerd door een integer tussen 0 en 4 voor de vijf verschillende acties. Hierbij komt 0 overeen met niks doen, 1 met omhoog, 2 met rechts, 3 met omlaag en 4 met links. De observation space bestaat uit een grid (ookwel een twee dimensionale lijst) van 31 hoog bij 28 breed waarin waardes tussen 0 en 4 staan opgeslagen. Ieder cijfer komt hierin overeen met een ander element. 0 staat voor een lege space, 1 voor een muur, 2 voor Pacman, 3 voor een (power)pellet of fruit en 4 voor een ghost. De positie in de grid komt overeen met de positie op het speelveld.

In regel 18 wordt een *keyboard*-variabelen geïnitieerd, welke wordt gebruikt voor het geven van input.

In regel 20 en 21 wordt het spel gestart.

6.2 *step(self, action: int)*-functie

In de *step(self, action: int)*-functie wordt één stap gezet in het spel. Dat wil zeggen: het spel wordt één frame verder gezet, inclusief de actie voor dat frame door de AI.

De eerste stap die binnen *step(self, action: int)* wordt gezet is het geven van een input. Dit wordt gedaan middels de *_give_input(action)*-functie. Deze functie verwacht een integer (0-4) en voert de corresponderende actie uit (dat wil zeggen, hij duwt deze toets op het toetsenbord in). Vervolgens wordt via de *receive_gamestate()*-functie in de game-controller de *gamestate* uitgelezen.

Vervolgens wordt de *map* opgeslagen in de *observation*-variable. De *map* dient als observatie voor de AI. Daarna wordt gecontroleerd of een run voor de AI afgerond is. Dit is het geval wanneer de AI een leven verliest of wanneer de AI het eerste level heeft gehaald. In het geval de run is afgerond worden de totale rewards geprint (inclusief reward per categorie).

Vervolgens wordt de reward berekend. Dit gebeurt middels de *_get_reward(gamestate, action)*-functie. De *gamestate* bevat vanzelfsprekend de *gamestate* en *action* bevat de actie die voor dat

frame wordt uitgevoerd (wederom uitgedrukt met een integer tussen 0 en 4). De `_get_reward(gamestate, action)`-functie zal worden toegelicht in hoofdstuk 6.3 `_get_reward(gamestate, action)`-functie.

Er wordt nu een *info*-variabele aangemaakt. Deze is nodig binnen Gym en kan worden gebruikt voor debugging, maar wordt hier niet gebruikt.

Tot slot wordt de game afgespeeld voor één frame, de *step_counter* met één opgehoogd en de *observatie*, *reward*, *done* en *info* gereturned.

6.3 `_get_reward(gamestate, action)`-functie

De `_get_reward(gamestate, action)`-functie berekent aan de hand van een *gamestate* en een uitgevoerde actie de behaalde *reward*. De *gamestate* bevat vanzelfsprekend de *gamestate* en *action* bevat de actie die voor dat frame wordt uitgevoerd (wederom uitgedrukt met een integer tussen 0 en 4).

Deze functie kan vele verschillende invullingen hebben, door de rewards en penalties op verschillende manieren te definiëren (bijvoorbeeld: wordt er wel of geen reward gegeven voor het verkrijgen van een hogere score).

De opbouw is wel altijd hetzelfde. Er wordt eerst een *reward*-variabele aangemaakt met als waarde 0. Deze wordt later opgehoogd of verlaagd afhankelijk van de behaalde rewards/penalties. Vervolgens worden de verschillende rewards en penalties toegekend. Voor iedere reward/penalty dient de *reward*-variabele te worden bijgewerkt (middels een `+=` of `-=` operatie). Daarnaast wordt aangeraden om dit zelfde te doen voor de `self.total_reward["REWARD NAAM HIER"]` variabele, zodat deze later kan worden gedebugged. De namen voor de rewards zijn in de `reset(self)`-functie gedefinieerd (en kunnen hier vrij worden aangepast). Zie een voorbeeld van het toekennen van een reward, wanneer het eerste level wordt behaald hieronder.

```
# Reaching Level 2 gives a (big) reward
if gamestate["level"] > 0:
    reward += 10000
    self.total_reward["level_complete"] += 10000
```

Tot slot wordt *reward* gereturned.

6.4 `reset(self)`-functie

De `reset(self)`-functie wordt aangeroepen om het spel te resetten voor de volgende run. Deze functie wordt dus iedere keer aangeroepen wanneer Pacman een leven verliest of het eerste level behaald.

Eerst worden een aantal variabele geïnitieerd op hun beginwaarde 0. De `self.score` bevat de score van het vorige frame (deze wordt gebruikt om te vergelijken met de nieuwe score); `self.step_counter` bevat het aantal stappen (= timeframes) wat is gezet sinds de laatste keer dat de AI is 'doodgegaan' (of in het geval van de eerste run, sinds de AI is gestart) en `self.total_reward` bevat de behaalde rewards voor de verschillende categorieën.

Vervolgens wordt het spel zelf opnieuw gestart en wordt de spatietoets kort ingedrukt (spatie moet op het begin van het spel worden ingedrukt om te starten). Vervolgens wordt de input gereset op 0 (dus geen input geven).

Tot slot wordt een eerste observatie uitgelezen en gereturned.

6.5 `render(self, mode='human', close=False)`-functie

De `render(self, mode='human', close=False)`-functie kan binnen Gym worden gebruikt voor het renderen van het spel. Aangezien in dit geval het spel al wordt gerenderd binnen de scripts in de `Pacman_Game`-map wordt deze functie hier niet gebruikt en wordt `None` gereturned.

7 `Graph.py`-bestand

Om de resultaten van de simulaties gedurende het trainen van de AI-agent te visualiseren wordt er gebruikgemaakt van de `matplotlib`-library om twee grafieken te tonen tijdens het trainingsproces. Ten eerste wordt een grafiek getoond van de “reward progression”. Hierbij wordt getoond hoeveel de agent aan “reward” (des te beter de agent het spel uitvoert, des te hoger de reward) heeft verzameld tijdens iedere run. Daarnaast wordt een grafiek getoond van de “score progression”, welke simpelweg aantoont welke score de agent heeft behaald bij iedere run. Door het gebruik van deze grafieken kunnen op twee manieren de vooruitgang van de agent geanalyseerd worden.

De reward en iedere score wordt gedocumenteerd door het overschrijven van een tekstbestand binnen de repository (genaamd `rewards.txt`).

```
# Log total reward and score
with open('rewards.txt', 'a', encoding='utf-8') as file:
    file.write(str(self.total_reward["total"]) + " " + str(self.score) +
"\n")
```

Het bestand wordt binnen `graph.py` uitgelezen en omgezet naar de twee grafieken. De eerste kolom van cijfers wordt gebruikt voor het tonen van de rewardprogressie. De tweede kolom van cijfers wordt gebruikt voor het tonen van de scoreprogressie.

Om ervoor te zorgen dat de grafieken tijdens het trainen van de AI-agent continu geüpdatet worden wordt gebruikgemaakt van een functie genaamd “animation” binnen de `matplotlib`-library. Zo krijgt de gebruiker in real-time de statistieken van de simulaties weergegeven binnen de grafieken.

```
# Update the graphs every second
animated = animation.FuncAnimation(figure, animate, interval=1000)
```

8 Toelichting op het doel van de AI

Het doel van de AI-agent is om uiteindelijk alle pellets te kunnen verzamelen bij het spelen van Pacman. Het verzamelen van alle pellets binnen Pacman zorgt er voor dat de speler het level voltooit. Voor het realiseren van de AI-agent wordt gebruikgemaakt van reinforcement learning. Bij reinforcement learning wordt een rewardfunctie gebruikt waarmee de AI-agent naar verloop van tijd aanleert om bepaalde acties wél uit te voeren, en andere acties te vermijden.

Bij het uittesten van de agent middels traditionele reinforcement learning algoritmen onder de normale omstandigheden van Pacman (het toelaten van alle ghosts binnen het spel) is gebleken dat de agent na verloop van tijd qua vooruitgang tegen stagnatie aanloopt. Zo kwam de agent nooit verder dan enkele seconden binnen een simulatie. Er is hieruit geconcludeerd dat het spel met alle aanwezige ghosts (nog) te complex is voor de agent om te leren, aangezien stagnatie niet als onderdeel wordt gezien van het leerproces van een agent.

Ook uit onderzoeken van Stanford University is gebleken dat het gebruik van reinforcement learning matig bleek te werken op het aanleren van Pacman (Gnanasekaran, Faba, & An) (Amelia

Christensen, 2016). Verder geeft een artikel van Medium vergelijkbare resultaten weer (Contreras, 2021).

De reden dat reinforcement learning niet geschikt is voor Pacman is niet uitgebreid onderzocht. Alhoewel er in de literatuur geen concrete redenen worden genoemd, is het vermoeden dat het dynamische van de omgeving (ghosts die bewegen afhankelijk van door de AI gegeven acties), gecombineerd met de grootte van de omgeving (observatie van een grid van 31 bij 28) een te complexe omgeving creëert voor reinforcement learning.

Uit onderzoek uitgevoerd door medewerkers van Microsoft Maluuba blijkt dat het mogelijk is Hybrid Reward Architecture (HRA) te gebruiken voor het leren van Pacman (het gebruik van meerdere agents met ieder een eigen doel) (van Seijen, 2017). Dit laat zien dat een reinforcement learning implementatie voor Pacman wel mogelijk is middels deze techniek.

Wegens de tegenvallende resultaten is ook getest zonder ghosts, om te zien of dit betere resultaten oplevert. Dit bleek niet het geval. De reden dat ook hier reinforcement learning niet geschikt is, is onduidelijk.

9 Verwijzingen

Amelia Christensen, B. R. (2016). *Recurrent Deep Q-Learning for PAC-MAN*. Stanford University.

Contreras, J. A. (2021, mei 27). *How to Train Ms-Pacman with Reinforcement Learning*. Opgehaald van Medium: <https://medium.com/analytics-vidhya/how-to-train-ms-pacman-with-reinforcement-learning-dea714a2365e>

Gnanasekaran, A., Faba, J. F., & An, J. (sd). *Reinforcement Learning in Pacman*. Stanford University.

Matplotlib. (2021). *Matplotlib: Visualization with Python*. Opgehaald van Matplotlib: <https://matplotlib.org/>

OpenAI. (sd). *Gym*. Opgehaald van OpenAI: <https://gym.openai.com/>

Pacmancode. (2022). *Pacmancode*. Opgehaald van Pacmancode: <https://pacmancode.com/>

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021, maart 7). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 1-8. Opgehaald van GitHub: <https://github.com/DLR-RM/stable-baselines3>

van Seijen, H. a. (2017, Juni 13). *Hybrid Reward Architecture for Reinforcement Learning*. Opgehaald van <https://www.microsoft.com/en-us/research/publication/hybrid-reward-architecture-reinforcement-learning/>