

Implementacja gry wykorzystującej uczenie przez wzmacnianie

Politechnika Warszawska, Wydział Elektryczny

Informatyka Stosowana

Łukasz Czajka

13 września 2025

Spis treści

1	Wstęp	3
1.1	Cel projektu	3
1.2	Wykorzystane biblioteki	3
2	Część ogólna	4
2.1	Struktura projektu	4
2.2	Struktura gier	5
2.3	Skrypt uruchamiający	5
3	Snake	7
3.1	Model AI - DQN	7
3.2	Wyniki	7
3.3	Porównanie wyników uzyskanych przez model AI z wynikami uzyskanymi przez człowieka	7
4	Pac-Man	10
4.1	Utworzenie gry	10
4.2	Zasady gry	11
4.3	Model AI - DQN	11
4.3.1	Cechy	11
4.3.2	Trenowanie	12
4.3.3	Wyniki	13
4.3.4	Wnioski	13
4.3.5	Pomysły na poprawę jakości modelu	13

Todo list

Można dodać UMLa	5
Napisać że zmieniłem cechy	12
Dodać gif pokazujący działanie	12

Rozdział 1

Wstęp

1.1 Cel projektu

Celem projektu indywidualnego było stworzenie gier wykorzystującej głębokie uczenie wraz z uczeniem przez wzmacnianie. Uzyskane przez AI wyniki następnie miały zostać porównane z wynikami uzyskanymi przez człowieka.

1.2 Wykorzystane biblioteki

Projekt został w Pythonie 3.13.7. Wykorzystane biblioteki zostały zamieszczone w pliku `requirements.txt`. Do najważniejszych bibliotek należą:

- TensorFlow - Biblioteka do uczenia maszynowego
- Matplotlib - Biblioteka do tworzenia wykresów
- Pygame - Biblioteka do tworzenia gier

Rozdział 2

Część ogólna

Na początku projektu poświęciłem czas na zaprojektowanie klas bazowych, które są wykorzystywane we wszystkich grach. Dzięki temu możliwe było zapewnienie spójności oraz łatwiejsze rozwijanie i utrzymanie kodu w kolejnych etapach pracy. Znajdują się katalogu `src/general`

2.1 Struktura projektu

Katalog `src` zawiera podkatalog `general`, który zawiera podstawowe klasy i funkcje, z których korzystają zaimplementowane gry. Zapewnia to ujednolicenie zachowań oraz ułatwia rozwijanie i utrzymanie kodu. Przykładowe klasy znajdujące się w `general` to:

- `AGameCore` - Odpowiedzialna za rysowanie, wykonywanie nowych ramek.
- `APlayer` - Abstrakcyjna klasa gracza. Tworzy instancję `GameCore`, i pozwala na interakcję z nią. Implementowana przez Agentów AI oraz gracza.
- `Drawable` - Interfejs dla obiektów, które mogą być rysowane na ekranie.
- `AGameState` - Klasa reprezentująca stan gry. Zawiera informacje o aktualnym stanie gry, takie jak pozycje graczy, stan planszy itp. Pozwala na łatwe kopiowanie ważnych informacji.
- `AGameStatsDisplay` - Klasa odpowiedzialna za wyświetlanie statystyk gry.

Część rzeczy zaimplementowanych w ramach tworzonych gier ostatecznie zostało przeniesione do katalogu `general`. Przykładem może być pakiet `maze`, który zawiera klasy związane z planszami. Nazwa jest pozostałością po Pac-Manie.

Pozostałe katalogi zawarte w katalogu `src` to poszczególne gry, które zostały zaimplementowane w ramach projektu.

2.2 Struktura gier

Pakiety agentów znajdują się w katalogu `src/nazwa_gry/agents/nazwa_agenta`.

Pakiet gry znajduje się w katalogu `src/nazwa_gry`.

Gry korzystające z mojego środowiska muszą zaimplementować klasy `APlayer`, `AGameCore` oraz `AGameState`.

Pakiety gier oraz pakiety agentów muszą udostępnić implementację klasy `APlayer` jako `Player`.

Można dodać
UMLa

2.3 Skrypt uruchamiający

W celu wygodnego uruchamiania i interakcji z grami i modelami stworzyłem skrypt uruchamiający `start.py`. Pozwala on na uruchamianie gry w różnych trybach, wybór agenta, zapisywanie i wczytywanie modeli, a także na modyfikacje konfiguracji.

```
./img/start_main_help.png
```

`./img/start_launch_help.png`

Rozdział 3

Snake

Snake to pierwsza gra, którą zaimplementowałem w ramach projektu. W trakcie tworzenia gry wzorowałem się częściowo na tutorialu przez Patricka Loebera [3]. Jak wspomniałem w części, podszedłem do tworzenia gry w sposób bardziej obiektowy - najpierw zaprojektowałem i utworzyłem klasy bazowe, a następnie dla każdej gry implementuję je.

Podczas implementacji gry Snake utworzyłem funkcje ułatwiające rysowanie obiektów i tekstu na ekranie, które następnie, w celu ułatwienia tworzenia kolejnych gier, przenieśliem do pakietu `general`.

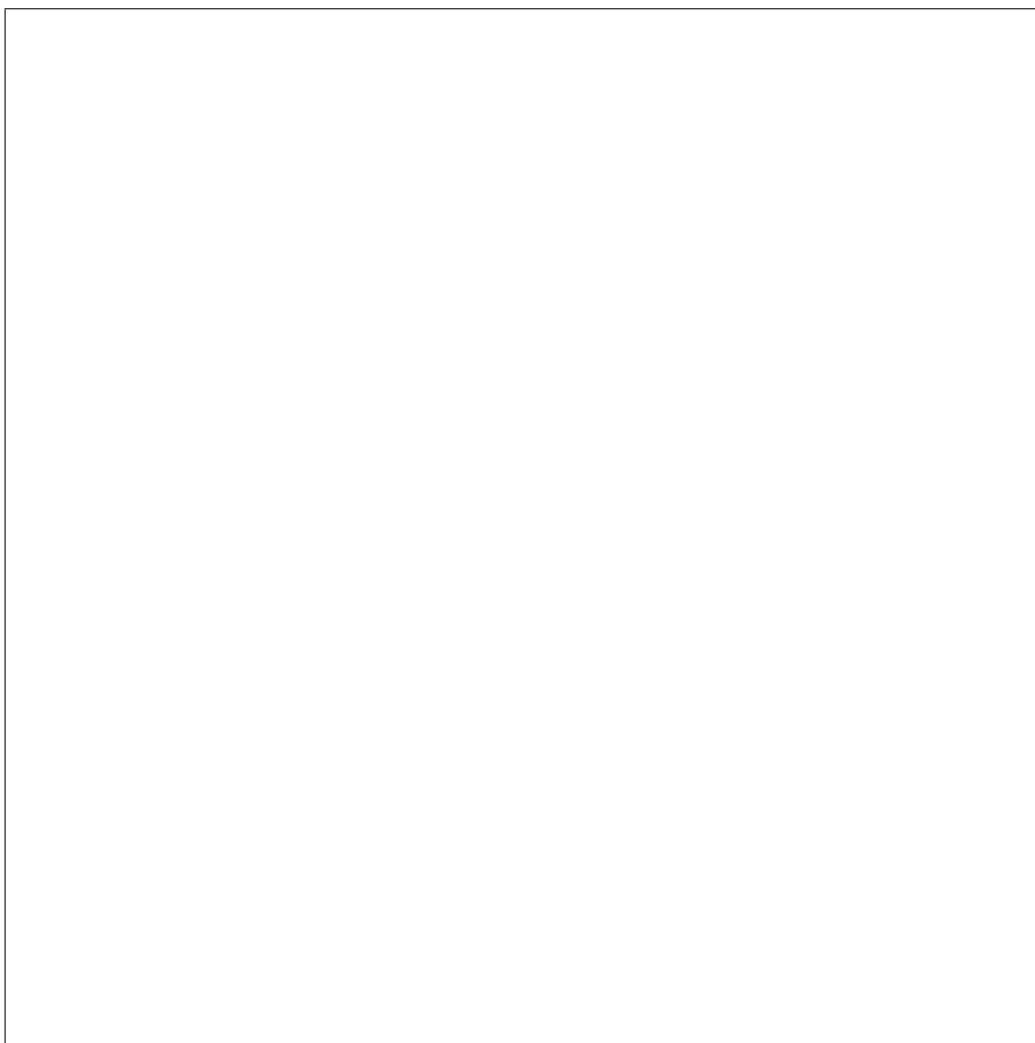
Kolizje z ogonem węża są wykrywane poprzez sprawdzenie, czy nowa pozycja głowy węża znajduje się w zbiorze pozycji zajmowanych przez ogon węża.

Ponieważ w każdej chwili jest co najwyżej jeden owoc, w celu sprawdzenia kolizji z owocem wystarczy sprawdzić, czy nowa pozycja głowy węża jest równa pozycji owocu.


3.1 Model AI - DQN

3.2 Wyniki

3.3 Porównanie wyników uzyskanych przez model AI z wynikami uzyskanymi przez człowieka



Rysunek 3.1: AI grające w Snake'a



`./img/snake_score.png`

Rysunek 3.2: Wyniki uzyskane przez AI podczas treningu na planszy 30x30

Rozdział 4

Pac-Man

4.1 Utworzenie gry

Korzystając z wcześniej utworzonych klas w katalogu `general` stworzyłem implementację Pac-Mana. W tym celu utworzyłem:

- `Maze` odpowiedzialny za przechowywanie i rysowanie obiektów na planszy.
- `MazeObject` będący klasą bazową dla wszystkich obiektów na planszy.
- `Actor` będący klasą bazową dla wszystkich obiektów poruszających się po planszy.
- `MazeUtils` będący klasą pozwalającą na wykonywanie pozostałych operacji na labiryncie.
- Rozwinięto możliwości rysowania na ekranie. Dodano system warstw, wprowadzono interfejs `Drawable`, dodano pozycje ciągłe.
- System podpięć pod każdą ramkę pozwalający na aktualizacje obiektów.
- Funkcję oznaczającą klatki jako te, w których można podjąć decyzję.
- System kolizji.
- Możliwość wczytywania labiryntu z pliku.

- Logikę dla każdego z duchów.
- System transakcji. Pozwala on na korzystanie z tymczasowych atrybutów. Pozwala on na symulowanie następnego ruchu.
- Rozwinięto system statystyk.

4.2 Zasady gry

Podczas projektowania gry wzorowałem się na oryginalnej wersji Pac-Mana, korzystając z zasad opisanych w the Pac-Man Dossier [2].

W celu uproszczenia gry wprowadziłem następujące modyfikacje:

- Pac-Man ma tylko jedno życie.
- Pac-Man nie może wykonywać tzw. Corneringu.
- Brak bonusowych punktów.
- Zmodyfikowany system punktacji.
- Jest tylko jeden poziom.

Wcześniejsze modyfikacje, jakie wprowadziłem, ale ostatecznie z nich zrezygnowałem to:

- Skokowy ruch - Pac-Man i duchy poruszały się o jedną kratkę na turę.
- Decyzje Pac-Mana podejmowane jedynie na skrzyżowaniu.
- Brak możliwości zawracania Pac-Mana na skrzyżowaniu.

4.3 Model AI - DQN

4.3.1 Cechy

Na początku próbowałem przekazać informację o odwiedzaniu poszczególnych tuneli w grze wraz z pozycją absolutną obiektów. Ponieważ Pac-Man nie miał możliwości zawracania oraz miał tylko jedno życie, wszystkie odwiedzone krawędzie musiały być puste. Miałem nadzieję, że model będzie w

stanie się nauczyć układu ścieżek w grze. Takie rozwiązanie okazało się nieefektywne - Spadek wartości ϵ wiązał się z spadkiem uzyskiwanego wyniku.

Bardziej efektywnym rozwiązaniem okazało się przekazywanie do modelu odwrotności najkrótszej odległości dla każdego możliwego kierunku, czyli wartości w postaci $\frac{1}{\text{odległość}}$. Dzięki temu wszystkie dane mieściły się w przedziale $[0,1]$. Taką metodę nawigacji zastosowałem również w przypadku pozycji duchów, skrzyżowań, oraz energizerów. Zdecydowałem również, że nie warto kodować odległości od każdego ducha indywidualnie, lecz wystarczy odległość od najbliższego ducha w każdym kierunku. Biorąc pod uwagę fakt, że duch może zawracać jedynie w wyjątkowych przypadkach wykorzystuje 2 nawigacje do ducha - Bez uwzględnienia możliwości zawrotu ducha oraz z możliwością zawrotu ducha. Pozostałe cechy to:

- Czas do zmiany stanu
- Pozostały czas działania energizera
- Ilość dostępnych energizerów
- Stan globalny duchów - Czy w trybie pościgu? Czy w trybie rozproszenia?

Napisać że
zmieniłem ce-
chy

4.3.2 Trenowanie

Ponieważ Pac-Man to skomplikowana gra, wzorując się na przykładzie CodeBullet[1], zdecydowałem się podzielić proces trenowania na kilka etapów.

Nawigacja labiryntu

Celem tego etapu było nauczenie Pac-Mana poruszania się po labiryncie i zbierania punktów.

Trenowałem Pac-Mana na klasycznym labiryncie, z wyłączonym spawnem duchów. Za zbliżenie się do punktów/jedzenia dodawany był mały bonus, a w przypadku oddalenia się - kara. Dodatkowo wprowadzona była kara za każdą wykonaną turę, aby zachęcić Pac-Mana do szybszego poruszania się po planszy.

Aby zabezpieczyć się przed utknięciem w pętli, dodałem głód - Jeżeli Pac-Man nie będzie się zbliżał do jedzenia przez określoną ilość tur, następuje śmierć.

Dodać gif po-
kazujący dzia-
łanie

Unikanie i zjadanie duchów

Celem tego etapu było nauczenie Pac-Mana unikania i jedzenia duchów. Aby to zrobić utworzyłem mnieszy labirynt, w którym duch byłby zmuszony do jedzenia punktów oraz jednocześnie unikania blinky’ego. Pozycje gracza i ducha były losowe. W środku labiryntu znajdował się energizer pozwalający na zjedzenie ducha.

W celu ułatwienia uczenia zmniejszyłem prędkość duchów oraz zwiększyłem prędkość Pac-Mana. Zachowałem dodane wymagania z poprzedniego etapu. Model był w stanie poprawiać swój wynik przez pewien czas.

Pełna gra

Ostatnim etapem było trenowanie Pac-Mana na pełnej planszy z większością zasad z oryginalnej gry. Prędkości aktorów zostały ustawione na domyślne dla poziomu 1. Podobnie jak wyżej zachowałem wymagania z poprzednich etapów. Model był w stanie poprawiać swój wynik przez pewien czas.

4.3.3 Wyniki

4.3.4 Wnioski

Dynamiczność Pac-Mana sprawiła, że jest to trudna gra do nauczenia się przez model. Wymaga ona od modelu umiejętności długoterminowego planowania, a także umiejętności przewidywania ruchów duchów. Przygotowanie dobrego modelu wymagałoby dużej ilości eksperymentów z cechami, architekturą sieci oraz parametrami trenowania.

4.3.5 Pomysły na poprawę jakości modelu

- Wykorzystanie Hybrid Reward Architecture [4]. Architektura ta została zaprezentowana na Pac-Manie, gdzie uzyskała ona naddludzkie wyniki. Mimo efektywności nie jest ona popularna, ponieważ wymaga zaimplementowania modeli dla każdego obiektu (punkty, duchy, gracze).
- Wykorzystanie CNN - Konwolucyjne sieci neuronowe są często wykorzystywane w grach, gdzie stan gry może być reprezentowany jako obraz. W przypadku Pac-Mana można by wykorzystać mapę gry jako

wejście do CNN, co pozwoliłoby modelowi na lepsze zrozumienie przestrzennej struktury gry. Zrezygnowałem z powodu zbyt dużego czasu trenowania.

- Wykorzystanie DQN wraz z siecią polityki oraz siecią celem. Jest ona stabilniejszą wersją DQN.

Bibliografia

- [1] CodeBullet. *AI learns to play PACMAN using NEAT*. <https://www.youtube.com/watch?v=QpyHYRBKy8U>. 2018.
- [2] Jamey Pittman. *The Pac-Man Dossier*. <https://pacman.holenet.info/>. 2015.
- [3] Patrick Loeber. *Teach AI To Play Snake - Reinforcement Learning Tutorial With PyTorch And Pygame*. <https://youtube.com/playlist?list=PLqns1RFeH2UrDh7vUmJ60YrmWd64mTTKV&feature=shared>. 2020. ■
- [4] Harm van Seijen i in. *Hybrid Reward Architecture for Reinforcement Learning*. 2017. arXiv: 1706.04208 [cs.LG]. URL: <https://arxiv.org/abs/1706.04208>.