Raport

Jan Gurgul, Gilbert Guszcza

28.05.2025

Wstęp

Celem projektu było stworzenie środowiska uczenia ze wzmocnieniem (Reinforcement Learning, RL), w którym agent porusza się po planszy ruchem skoczka szachowego (knight) i ma za zadanie dotrzeć do losowo wybranego celu. Problem ten jest przykładem zadania nawigacyjnego z ograniczonymi ruchami, gdzie agent musi nauczyć się optymalnej sekwencji działań, aby jak najszybciej osiągnąć cel.

Środowisko

Środowisko zostało zaimplementowane w pliku 'knight_jump_word.py' jako klasa KnightWorldEnv, zgodna z API biblioteki Gymnasium. Najważniejsze cechy środowiska:

- Plansza: kwadratowa o rozmiarze konfigurowalnym (np. 5x5, 12x12)
- Agent: reprezentowany przez współrzędne na planszy
- Cel: losowo wybrane pole, inne niż pozycja agenta
- Akcje: 8 możliwych ruchów skoczka szachowego (zgodnie z zasadami szachów)
- Nagrody:
  - +1 za osiągnięcie celu
  - -0.01 za każdy poprawny ruch
  - -0.1 za próbę wyjścia poza planszę
- Obserwacja: pozycja agenta i celu, przekazywana jako słownik lub spłaszczona tablica (przy użyciu FlattenObservation)
- Wizualizacja: środowisko korzysta z biblioteki pygame do graficznego przedstawienia planszy, agenta i celu

Dodatkowo, w folderze 'wrappers' znajdują się klasy umożliwiające modyfikację nagród, przestrzeni akcji i obserwacji, np. ClipReward, DiscreteActions, RelativePosition.

W pliku 'grid_world.py' znajduje się uproszczona wersja środowiska, w której agent porusza się w czterech kierunkach (prawo, lewo, góra, dół).

Algorytm

Do nauki optymalnej polityki ruchów wykorzystano klasyczny algorytm Q-learning. Q-learning polega na budowie tablicy Q, która dla każdego możliwego stanu i akcji przechowuje oczekiwaną sumę nagród. W trakcie treningu agent eksploruje środowisko, aktualizując wartości Q na podstawie otrzymywanych nagród.

Reprezentacja stanu

Stan środowiska kodowany jest jako pozycja agenta oraz pozycja celu. W pliku learning.py zastosowano funkcję state_to_index, która zamienia cztery liczby (y, x agenta i celu) na unikalny indeks w tablicy Q. Dzięki temu możliwe jest efektywne indeksowanie tablicy Q nawet dla dużych plansz (np. 12x12).

Eksperymenty z AI

Trening agenta odbywa się w pliku 'learning.py':

1. Środowisko jest inicjalizowane dla planszy 12x12
2. Tablica Q jest inicjalizowana zerami
3. Agent uczy się przez 10 000 epizodów, korzystając z polityki epsilon-greedy
4. Po każdym epizodzie zapisywana jest suma nagród, a na końcu rysowana jest krzywa uczenia
5. Wytrenowana tablica Q oraz rozmiar planszy są zapisywane do plików ('Q_table.npy', 'grid_size.npy')

Testowanie agenta odbywa się w pliku 'main.py':

1. Ładowana jest wytrenowana tablica Q oraz rozmiar planszy
2. Środowisko uruchamiane jest w trybie graficznym ("human")
3. Agent w każdej turze wybiera najlepszą możliwą akcję (np.argmax(Q[state_idx]))
4. Wynik działania agenta (suma nagród) jest wyświetlany po zakończeniu epizodu
5. Maksymalna liczba kroków została ograniczona do 100, aby uniknąć zapętlenia

Wnioski

- Środowisko poprawnie modeluje ruchy skoczka i umożliwia wizualizację procesu uczenia oraz działania agenta, choć czasami sekwencja ruchów agenta wpada w zapętlenie
- Q-learning pozwala na skuteczne nauczenie agenta optymalnych ruchów nawet na dużych planszach, choć wymaga to odpowiedniej reprezentacji stanu i pamięci na tablicę Q
- Implementacja środowiska w stylu Gymnasium umożliwia łatwą integrację z innymi algorytmami RL oraz narzędziami do eksperymentów
- Dzięki wrapperom możliwe jest szybkie modyfikowanie sposobu nagradzania, przestrzeni akcji i obserwacji, co ułatwia eksperymentowanie z różnymi wariantami środowiska