Wydział Matematyki i Nauk Informacyjnych

Politechniki Warszawskiej

Metody Sztucznej Inteligencji II

Agent grający w grę Connect4

Dokumentacja

Bartłomiej Chechliński

Mateusz Chiliński

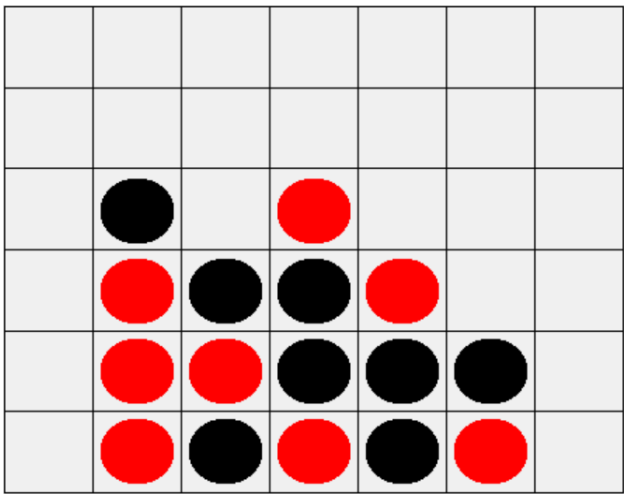
12 kwietnia 2019

Spis treści

1. Konspekt

1.1. Opis gry Connect4

Gra Connect4 polega na ułożeniu 4 żetonów w linii na planszy, która posiada 7 kolumn oraz 6 rzędów. Żetony możemy dodawać do dowolnej niepustej kolumny, z tym, że układamy je od dołu - czyli żeton jest ustawiany w najniższym możliwym rzędzie. Grę ilustruje rysunek 1.1.



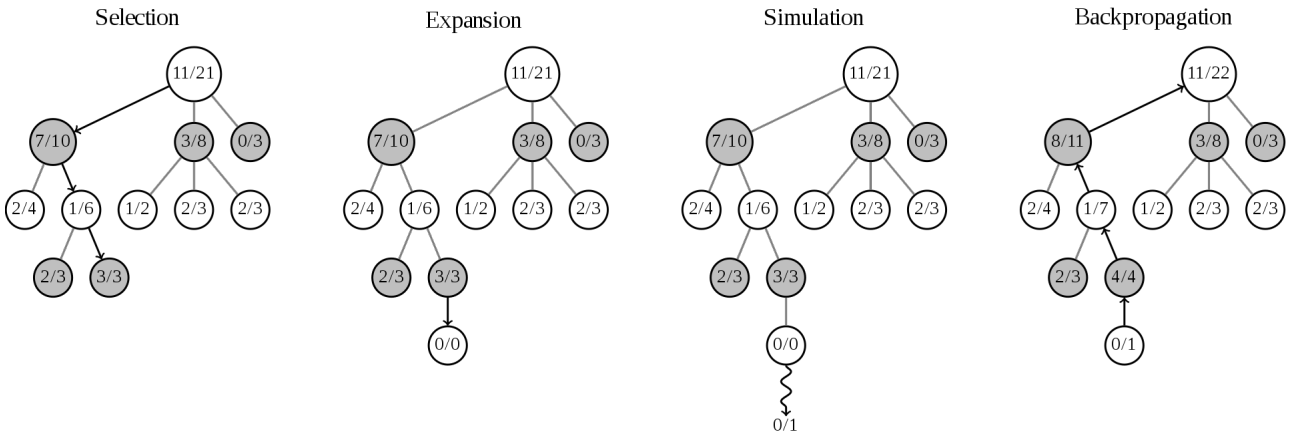
Rysunek 1.1: Wygląd rozgrywki w grę Connect4 - wygrał gracz czarny.

1.2. Opis algorytmu MCTS

Monte-Carlo Tree Search (w skrócie MCTS) jest heurestyką, która pozwala na podejmowanie decyzji w pewnych szczególnych zastosowaniach sztucznej inteligencji. Metoda ta jest używana do gier, w których przebadanie wszystkich możliwych ruchów i znalezienie tego najlepszego jest niemożliwe - najlepszymi przykładami są tutaj gry takie jak szachy, GO i gra Connect4.

W metodzie MCTS analizujemy najbardziej obiecujące ruchy poprzez rozszerzanie naszego drzewa decyzyjnego na podstawie skończonej liczby gier testowych. Drzewo tworzymy za pomocą 4 etapów, które są reprezentowane przez rysunek 1.2

Ilustracja ta przedstawia kolejne etapy, które pomagają rozwinąć drzewo decyzyjne. Na sa-



Rysunek 1.2: Przykład tworzenia drzewa MCTS.[2]

mym początku mamy etap wyboru (z ang. *Selection*), idziemy od korzenia do liścia, który jest najbardziej obiecującym z naszej perspektywy. Następnie przechodzimy do etapu rozrostu (z ang. *Expansion*) - tworzymy potomka naszego wybranego liścia (lub też kilka) i wybieramy jeden z utworzonych przez nas potomków. Naturalnym etapem następującym po rozroście jest symulacja - nasz wybrany, jeden z utworzonych potomków jest wykorzystywany do stworzenia losowej symulacji (wybieramy losowe ruchy, aż gra jest rozstrzygnięta). Mając wynik naszych symulacji, przechodzimy do ostatniej fazy - propagacji wstecz (z ang. *Backpropagation*). Polega ona na tym, że rezultat gry jest przekazywany do ścieżki od naszego wybranego potomka, aż do korzenia. Tak wygląda pojedyncza iteracja rozrostu - iteracji wykonujemy tyle, żeby zmieścić się w limicie czasowym na daną turę (czas na wykonanie pojedynczego ruchu). Ostateczną decyzję (tj. wybranie ruchu z następników korzenia) podejmujemy nie na podstawie wygranych, ale łącznej liczby wykonanych symulacji. W przypadku uwzględnionym na obrazku wybralibyśmy ten z lewej, bo ma 11 wykonanych symulacji - nawet jeśli byłby inny następnik, który miałby zapisane w sobie 9/10. Wtedy też przechodzimy do następnika uczyniając go korzeniem i od nowa symulujemy według tej metody - dzięki temu nie tracimy wyników, a każdy następny ruch jest oparty o większą ilość symulacji, więc jest lepszy. Należy jeszcze doprecyzować w jaki sposób będziemy oceniać które kolejne wierzchołki powinniśmy wybierać w etapie wyboru idąc ścieżką od korzenia w dół, by ostatecznie zatrzymać się na liściu. Będąc w pewnym wierzchołku drzewa pójdziemy do jego potomka, którego zdefiniowana poniżej wartość będzie największa.

$$v = \frac{w}{n} + \sqrt{\frac{\log N}{n}}$$

gdzie:

w — liczba wygranych symulacji przeprowadzonych dla rozpatrywanego potomka

n — liczba symulacji przeprowadzonych dla rozpatrywanego potomka

N — liczba wszystkich przeprowadzonych symulacji

Taki sposób wyboru kolejnych wierzchołków ścieżki pozwala na zbalansowanie częstotliwości wyboru tych wierzchołków, których symulacje często kończą się zwycięstwem, oraz tych, które rzadko były rozwijane.

1.3. Rozwiązanie problemu Connect4.

W naszym rozwiązaniu zastosowaliśmy wiele optymalizacji, które pozwalają zaimplementowanej sztucznej inteligencji na granie na bardzo wysokim poziomie. Po pierwsze, trenujemy model przed grą. Polega to na tworzeniu drzewa MCTS zaczynając od początku. W naszym kodzie zastosowaliśmy 100000000 iteracji uczących. Jednak tworząc drzewo MCTS, korzystamy z kilku optymalizacji - jeśli gracz może wygrać grę, to ją wygrywa oraz jeśli może powstrzymać drugiego gracza przed wygraną, to to robi. Dzięki temu możemy nakierować MCTS na lepsze rozwiązania. Otrzymane w ten sposób drzewo (model) serializujemy za pomocą rozwiązania Google - protocol buffers[4] - zaimplementowanego w C w postaci pakietu protobuf-net[5]. Dzięki temu możemy bardzo szybko i wydajnie serializować i deserializować olbrzymie drzewa MCTS.

Podczas gry, AI wybiera najlepsze rozwiązanie z modelu (to, które prowadziło do największej liczby zwycięstw), przy czym w każdym ruchu rozszerza drzewo przez pewien okres czasu (maksymalny dozwolony).

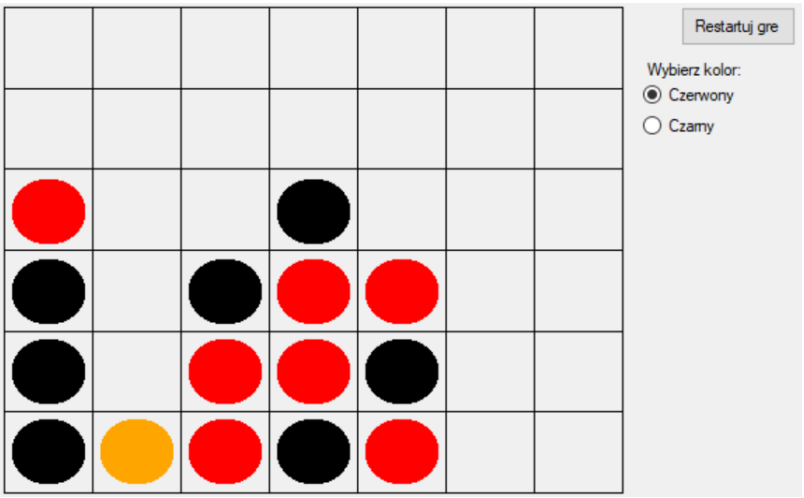
2. Instrukcja obsługi.

2.1. Generowanie nowego modelu MCTS.

Aby wygenerować nowy model (drzewo MCTS) uruchamiamy MCTS.exe, czekamy, aż obliczenia się zakończą, a następnie dostajemy plik wynikowy - *model.bin*. Jeśli chcemy go użyć w aplikacji, musimy go przenieść do folderu, w którym się znajduje (*Connect4Game*).

2.2. Gra Connect4.

Włączamy grę, wybieramy kolor (jeśli gra jest rozpoczęta, to spowoduje to jej zrestartowanie - o czym użytkownik zostnaie poinformowany). Możemy także zrestartować rozgrywkę poprzez przycisk Żestartuj grę".



Rysunek 2.1: Przykład tworzenia drzewa MCTS.[2]

Bibliografia

[1] Algorytmy uczące w grach na przykładzie gry w GO - Patryk Cisło, Mateusz Chiliński, Albert Dziugieł, Michał Dzidziewicz

[2] Monte Carlo tree search - https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

[3] MCTS.ai - http://mcts.ai/about/

[4] Protocol Buffers - https://developers.google.com/protocol-buffers/

[5] protobuf-net git repository - https://github.com/mgravell/protobuf-net