Zastosowanie algorytmu UCT do stworzenia sztucznej inteligencji grającej w *Connect4*

Wstępna dokumentacja projektu

Patryk Fijałkowski

Mateusz Burczaniuk

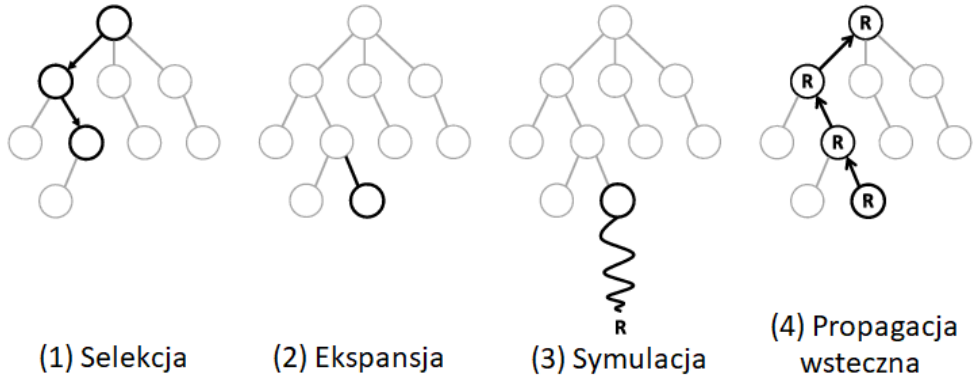
29 kwietnia 2020

1 Opis *Connect4*

TODO

2 Algorytmy MCTS

Monte-Carlo Tree Search to heurystyka, której celem jest podejmowanie decyzji w pewnych zadaniach sztucznej inteligencji, na przykład wybieranie ruchów w grach. Metoda jest oparta na przeszukiwaniu możliwych stanów gry zapisanych w wierzchołkach drzewa i losowym symulowaniu rozgrywek. Algorytmy MCTS opierają się na rozbudowywaniu drzewa ze stanami gry przez iteracyjne wykonywanie czterech faz. Jednym z najpowszechniejszych wariantów MCTS jest algorytm UCT. Pseudokod opisany w Listingu 1 oraz implementacja MCTS w projekcie bazują na [1]. Przykład działania algorytmu ze szczególnym uwzględnieniem kolejnych faz znajduje się na Rysunku 1.



Rys. 1: Fazy MCTS, źródło: [2]

1. **Faza selekcji** (wiersz 6 w listingu) – wybór pewnego liścia drzewa. Rozdział 2.1 opisuje jeden ze sposobów na wybranie wierzchołka w tej fazie.
2. **Faza ekspansji** (wiersz 7 w listingu) – utworzenie wierzchołków potomnych dla wierzchołka wybranego w fazie selekcji. Tworzone wierzchołki odpowiadają stanom możliwym do uzyskania przez wykonanie jednego ruchu ze stanu rodzica.
3. **Faza symulacji** (wiersz 10 w listingu) – rozegranie partii składającej się z losowych ruchów ze stanu jednego z wierzchołków utworzonych w poprzedniej fazie. Rozgrywana jest ona do końca, czyli do wyłonienia zwycięzcy lub spowodowania remisu, lub jest ucinana po pewnej liczbie ustalonych ruchów i wynik gry jest ewaluowany przez pewną funkcję.
4. **Faza propagacji wstecznej** (wiersz 11 w listingu) – aktualizacja informacji na temat wierzchołków na ścieżce od liścia, z którego rozpoczęto symulację, do korzenia drzewa. Główną przekazywaną wartością jest wynik symulacji.

```
def find_next_move(curr_state):
    iterations_counter = 0
    tree = initialize_tree(curr_state)

    while iterations_counter < max_iterations_counter:
        curr_node = select a leaf from tree
        create child nodes from curr_node
        if curr_node has children:
            curr_node = random child of curr_node
        playout_result = simulate random playout from curr_node
        update tree according to playout_result
        iterations_counter++

    best_state = select best child(tree.root)
    return best_state

```

Listing 1: Pseudokod algorytmu Monte Carlo Tree Search

2.1 Algorytm UCT

UCT jest wariantem metody MCTS, który stara się zachować równowagę między eksploatacją bardziej obiecujących ruchów a eksploracją tych rzadko odwiedzonych. Formuła, która odpowiada za wyznaczenie najbardziej obiecującego wierzchołka w fazie wyboru MCTS jest przedstawiona jako wyrażenie (1).

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}} \tag{1}$$

W wyrażeniu (1), indeks i odnosi się do liczby wykonanych przez algorytm iteracji, czyli czterech faz MCTS. W pierwszym składniku sumy wyrażenia (1), licznik w_i oznacza sumę wszystkich wypłat w danym węźle, a mianownik n_i oznacza liczbę rozegranych symulacji. Zatem ułamek ten przyjmuje wartości większe dla ruchów o większej średniej wygranej, co odpowiada ze eksploatację drzewa. Drugi składnik sumy wyrażenia (1) przyjmuje wartości większe dla wierzchołków, dla których wykonano mniej symulacji i odpowiada eksploracji drzewa. $N_i = \sum_i n_i$, a c jest parametrem eksploracji, który może być dostosowany do badanego problemu.

2.1.1 Usprawnienie UCB-Minimal

Przedstawione w [3] usprawnienie UCB-Minimal polega na zastosowaniu wzoru (2) w celu wyboru najbardziej obiecującego wierzchołka w fazie selekcji.

$$\frac{w_i}{n_i} + \frac{C_1}{n_i^{C_2}} \tag{2}$$

W wyrażeniu (2), C_1 jest parametrem balansującym eksploatację i eksplorację drzewa. Z kolei C_2 to parametr, którego zadaniem jest skorygować zmniejszenie wpływu eksploracji, które następuje z kolejnymi iteracjami algorytmu. Dobór parametrów powinien zostać

wykonany empirycznie – autorzy [3] sugerują wartości $C_1 = 2.5, C_2 = 1$ jako ogólnie optymalne. Przykładowo, badania autorów [4] wykazały, że najlepiej sprawdzają się wartości $C_1 = 8.4, C_2 = 1.8$.

2.1.2 Usprawnienie UCB-V

Przedstawione w [5] usprawnienie UCB-V opiera swoją skuteczność na wykorzystaniu wariancji wypłat powodując, że algorytm w fazie selekcji wybiera jak najlepsze i jak najmniej zróżnicowane ruchy w kontekście oczekiwanej wypłaty. UCB-V polega na zastosowaniu wzoru (3) w celu wyboru najbardziej obiecującego wierzchołka w fazie selekcji.

$$\frac{w_i}{n_i} + \sqrt{2\frac{\sigma_i^2 \cdot \varepsilon}{n_i}} + c\frac{3 \cdot \varepsilon}{n_i} \tag{3}$$

W wyrażeniu (3), σ_i^2 oznacza wariancję wypłat w danym węźle, a ε to tak zwana *funkcja eksploracji*. Zgodnie z sugestią autorów [5], za *funkcję eksploracji* przyjęta zostanie postać zaprezentowana w (4), gdzie ζ jest parametrem, który powinien zostać dopasowany empirycznie.

$$\varepsilon = \zeta \cdot \ln N_i \tag{4}$$

Autorzy [5] sugerują wartości parametrów $c = 1, \zeta = 1$ jako optymalne dla prezentowanego usprawnienia. Z drugiej strony, w zastosowaniu go do gry *Tron*, autorzy pracy [4] najlepsze efekty zaobserwowali dla wartości parametrów $c = 1.68, \zeta = 0.54$.

3 Podejście heurystyczne

TODO

4 Rozwiązanie

5 Hipotezy badawcze

6 Harmonogram działań

W tabeli 1 przedstawiono planowany harmonogram działań podczas pracy nad projektem.

Tab. 1: Harmonogram pracy

Deadline	**Przygotowane zadania**
06.05.2020	Stworzenie dokumentacji projektu Zaimplementowanie struktur potrzebnych do operowania na drzewach
13.05.2020	Zaimplementowanie logiki gry Przygotowanie aplikacji okienkowej
20.05.2020	Zaimplementowanie algorytmu UCT
07.05.2020	Zaimplementowanie algorytmu heurystycznego Zaimplementowanie dwóch wariantów UCT
03.06.2020	Przeprowadzenie eksperymentów w celu weryfikacji hipotez Wstępna weryfikacja postawionych hipotez
10.06.2020	Pełna weryfikacja postawionych hipotez Stworzenie raportu

7 Projekt techniczny

Projekt zostanie sporządzony przy użyciu języka C#. Nie jest planowane użycie żadnych specjalistycznych bibliotek tego języka.

Literatura

[1] Levente Kocsis, Csaba Szepesvári, *Bandit based Monte-Carlo Planning*, European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006.

[2] Steven James, George Konidaris, Benjamin Rosman, *An Analysis of Monte Carlo Tree Search*, University of the Witwatersrand, Johannesburg, South Africa.

[3] Francis Maes, Louis Wehenkel, Damien Ernst, *Automatic Discovery of Ranking Formulas for Playing with Multi-armed Bandits*, European Workshop on Reinforcement Learning, Athens, Greece, September 9–11, 2011.

[4] Pierre Perick, David L. St-Pierre, Francis Maes, Damien Ernst, *Comparison of Different Selection Strategies in Monte-Carlo Tree Search for the Game of Tron*, IEEE Conference on Computational Intelligence and Games, Granada, Spain, September 12–15, 2012.

[5] Jean-Yves Audibert, Remi Munos, Csaba Szepesvári, *Tuning Bandit Algorithms in Stochastic Environments*, Algorithmic Learning Theory 18th International Conference, Sendai, Japan, October 1–4, 2007.