Zastosowanie *Monte Carlo Tree Search* oraz *Upper Confidence Bound Applied To Trees* do stworzenia sztucznej inteligencji grającej w grę Othello

Raport

Szymon Stasiak Przemysław Woźniakowski

24 września 2022

Streszczenie

Monte Carlo Tree Search to potężna metoda, która znalazła zastosowanie w wielu dziedzinach sztucznej inteligencji. Szczególnie interesującym tematem, jest zastosowanie jej w obszarze gier dwuosobowych, gdzie popularna jest wersja równoważąca eksploatacje z eksploracją za pomocą metody Upper Confidence Bound 1 applied to trees. W ramach projektu stworzona została implementacja metody Monte Carlo Tree Search wykorzystującej metodę UCT dla gry Othello. Dodatkowo zaimplementowane zostało także podejście bazujące na algorytmie *alpha-beta pruning*, oraz dwie modyfikacje UCT: podejście UCB1-tuned oraz podejście, w którym powtarzające się w drzewie stany, były łączone we wspólny węzeł. W ramach testów przeprowadzona została seria pojedynków wszystkich algorytmów w formacie każdy z każdym. Wbrew wstępnym oczekiwaniom, najlepiej wypadła w nich podstawowa wersja algorytmu UCT, natomiast wersja heurystyczna zgodnie z oczekiwaniami zajęła ostatnie miejsce. Wszystkie wersje UCT wykonywały ruchy z podobnym czasem, choć nieznacznie szybsza była wersja z grupowaniem ruchów. Kolejnym testem było przeciwstawienie algorytmów średnio-zaawansowanemu, prawdziwemu graczowi w serii 3 gier *Best of 3*, czyli do 2 zwycięstw. W tych pojedynkach również najlepsza okazała się podstawowa wersja UCT, choć wersja z grupowaniem ruchów osiągnęła porównywalny wynik, gdyż obie wygrały 2 serie i zremisowały jedną. Z kolei algorytm heurystyczny nie był w stanie wygrać ani jednej rundy. Analizując grę najlepszej sztucznej inteligencji, można było zauważyć, że podejmuje ona ruchy w ten sposób, aby w końcowej fazie gry maksymalnie ograniczyć możliwość podejmowania dobrych ruchów przez oponenta, przy jednoczesnej maksymalizacji własnych dobrych ruchów. Dzięki temu w kilka tur była ona w stanie kompletnie odwrócić bieg rozgrywki.

Spis treści

1 Opis problemu

Gry dwuosobowe są znanym obiektem analizy dla algorytmów sztucznej inteligencji. Swoją popularność zawdzięczają między innymi prostym zasadom takich gier oraz łatwości oceny działania - końcowy rezultat rozgrywki jednoznacznie wyznacza, czy algorytm sprawdził się dobrze czy źle. Mechanika gier jest prosta do przedstawienia za pomocą drzewa decyzyjnego, w którego węzłach znajdują się stany rozgrywki, zaś krawędzie determinowane są poprzez możliwe do wykonania ruchy (na zmianę przez pierwszego oraz drugiego gracza). Stany przechowywane w liściach są stanami odpowiadającymi zakończonej rozgrywce, której rezultat jest jednoznaczny.

Problemy w grach dwuosobowych sprowadzają się do znajdowania kolejnych ruchów jednego z graczy, chcąc doprowadzić do jego zwycięstwa. Zagadnienie to okazuje się jednak często zbyt złożone dla algorytmów klasycznych. Pełne rozwiązanie problemu wymaga bowiem wygenerowania całego drzewa gry i oceniania kolejnych pozycji zgodnie z pewną heurystyką, co jest często zbyt czasochłonne.

Główna modyfikacja stosowana w podejściach opartych na metodach sztucznej inteligencji opiera się na ograniczeniu wielkości generowanego drzewa, pomijając ruchy intuicyjnie bardzo słabe. Przykładem algorytmu o takim działaniu jest **Monte Carlo Tree Search** (MCTS) oraz jego modyfikacje takie jak wykorzystanie algorytmu **Upper Confidence Bound Applied To Trees** (UCT), szerzej opisane w sekcjach 1.2 oraz 1.3.

W zakresie tego projektu, przeanalizowano grę w Othello, wykorzystując do tego wspomniane MCTS z UCT (sekcje 1.2 i 1.3), ich modyfikacje (sekcje 1.4.1 i 1.4.2) oraz podejście heurystyczne z sekcji 1.5.

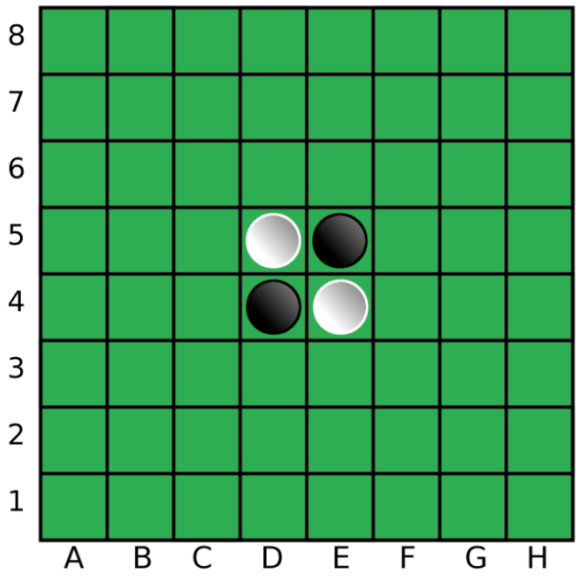
1.1 Zasady gry Othello

Gra w **Othello** odbywa się na na kwadratowej planszy o 64 polach (8×8). Często nazywana jest również mianem **Reversi**, jednak tak naprawdę Othello jest szczególną modyfikacją Reversi, która niemal całkowicie zastąpiła swojego przodka [1]. W grze w Othello wykorzystywane są 64 piony (dyski), które z jednej strony są czarne, a z drugiej białe. Przed rozpoczęciem gry 4 piony są ułożone na czterech centralnych polach planszy, w taki sposób, że po lewej stronie każdego gracza jest pion czarny, zaś po prawej jest biały.

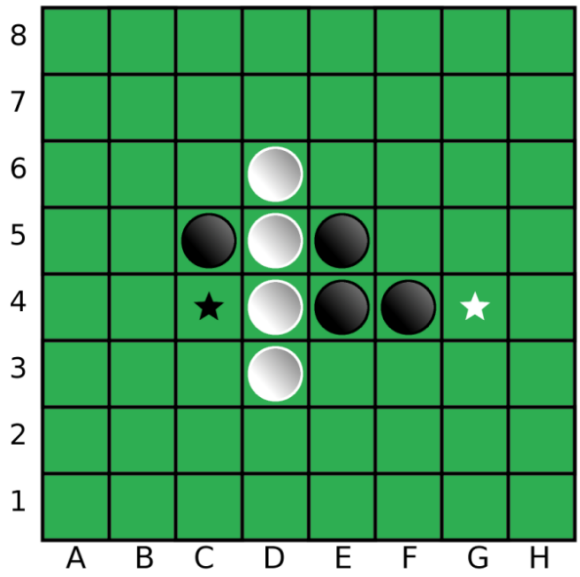
Jeden z graczy gra dyskami czarnymi, drugi białymi. Grę rozpoczyna grający czarnymi. Następnie gracze wykonują ruchy na zmianę. Każdy ruch polega na postawieniu swojego piona na planszy. Po położeniu danego piona, nie może on już zostać przesunięty, jednak może zmieniać swój kolor (czyli zostać odwrócony).

Piony można stawiać wyłącznie w taki sposób, aby w każdym ruchu dokonać bicia przynajmniej jednego piona przeciwnika. Bicie odbywa się przez pojmanie, czyli przez takie postawienie piona, aby pomiędzy nim oraz innym własnym pionem znalazły się piony przeciwnika. Te piony – złapane pomiędzy nowo dostawiony pion i inny własny pion – są bite. Jeżeli dany gracz nie ma możliwości dostawienia piona w legalny sposób, traci on kolejkę a ruch przechodzi do kolejnego gracza.

Gra kończy się w przypadku gdy żaden z graczy nie może wykonać poprawnego ruchu (w szczególności gdy na planszy nie ma już wolnych miejsc). W tym momencie dla każdego koloru zliczane są piony znajdujące się na planszy. Rozgrywkę wygrywa gracz w którego kolorze znajduje więcej dysków.



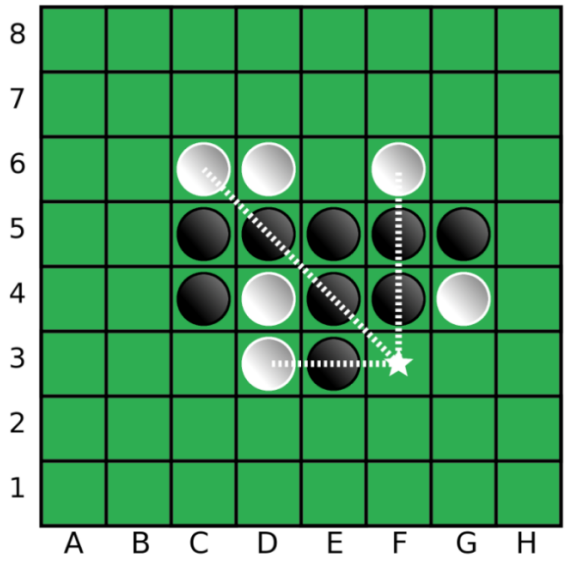
Rysunek 1: Początkowe ustawienie pionów.



Rysunek 2: Diagram pokazujący przykładowe bicia.

Rysunek 2 przedstawia możliwe bicia w danej sytuacji. Gdyby ruch miały czarne, mogłyby zagrać w pole zaznaczone czarną gwiazdką (C4). Wówczas pion stojący na polu D4 zostałby pojmany i zmieniłby kolor na czarny. W pojmaniu uczestniczyłby pion na polu E4.

Przejmowanie pionów jest obowiązkowe w każdym ruchu. Jeśli nie ma możliwości bicia, to nie ma możliwości wykonania ruchu. Bicie jest możliwe w linii pionowej, poziomej i ukośnie. Jeśli w wyniku wykonania ruchu można bić w wielu kierunkach, bicie następuje we wszystkich kierunkach jednocześnie. Taką sytuację przedstawia diagram poniżej.



Rysunek 3: Postawienie białego dysku na polu oznaczonym gwiazdką (F3) spowoduje bicie w pionie (pionów F4 i F5), w poziomie (piona E3) oraz ukośnie (piony E4 i D5). Wszystkie te piony zmienią po biciu kolor na biały.

Celem gry jest osiągnięcie maksymalnej liczby własnych pionów na planszy w momencie zakończenia gry. Gra kończy się, gdy wszystkie pola zostaną zajęte. W rzadkich sytuacjach może dojść do zakończenia gry wcześniej, jeśli żaden z graczy nie może wykonać żadnego ruchu. Najczęściej do takiej sytuacji dochodzi, gdy z planszy znikną wszystkie piony jednego gracza.

Może też dojść do sytuacji, gdy gracz nie może wykonać ruchu, ale jego przeciwnik tak. W takiej sytuacji gracz niemogący wykonać ruchu pasuje (traci kolejkę). Wykonuje następny ruch dopiero wtedy, gdy będzie miał taką możliwość.

1.2 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) jest metodą podejmowania decyzji o wyborze danej akcji w pewnym stanie. Jest ona szeroko używana do tworzenia sztucznych inteligencji zajmujących się graniem w gry dwuosobowe. Znajduje ona swoje zastosowanie przy tworzeniu drzewa gry, opisującego jakość kolejnych ruchów możliwych do wykonania w danych pozycjach.

W metodzie tej, drzewo gry budowane jest asymetrycznie, w kierunku najbardziej obiecujących ruchów. Należy przez to rozumieć ruchy, które przynosiły najlepsze rezultaty dla przeprowadzonych symulacji. Działanie metody można podzielić na etapy, które wykonywane są w każdej iteracji budowy drzewa:

1. Selekcji - rozpoczynając od korzenia drzewa, odpowiadającego stanowi startowemu, wybierane są kolejne wierzchołki potomne (kolejne ruchy) aż do dotarcia do liścia drzewa. Sposób wyboru węzłów potomnych jest zależny od metody, ale ogólnie jego celem jest rozwijanie drzewa w kierunku najbardziej obiecujących ruchów.

2. Ekspansji - jeśli wybrany ruch nie kończy gry to dla odpowiadającego mu węzła tworzone są jeden lub więcej węzłów potomnych i wybierz z nich jeden.

3. Symulacji - rozgrywana jest losowa symulacja rozpoczynana w węźle wybranym w ekspansji.

4. Propagacji wstecz (backpropagation) - na podstawie wyniku rozegranej symulacji uaktualniane są wierzchołki na ścieżce od korzenia do punktu wybranego w etapie ekspansji. To uaktualnienie brane jest pod uwagę przy selekcji w kolejnych iteracjach.

1.3 Upper Confidence Bound Applied To Trees (UCT)

UCT jest najbardziej znanym i najszerzej stosowanym algorytmem mającym na celu ulepszenie procesu selekcji w metodzie MCTS. Jej celem jest znalezienie równowagi między podejściem dążącym do eksploatacji obecnie najlepszego rozwiązania oraz eksploracji w poszukiwaniu innych, potencjalnie lepszych rozwiązań. Algorytm UCT powstał na bazie innego rozwiązania - **Upper Confidence Bound (UCB1)**, przenosząc jego logikę na drzewa. Akcja wybierana w określonym położeniu jest definiowana za pomocą następującego wzoru:

$$a^* = argmax_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln[N(s)]}{N(s,a)}} \right\} \tag{1}$$

Gdzie a oznacza akcję, s to obecny stan, $A(s)$ jest zbiorem akcji możliwych do wykonania w stanie s, $Q(s,a)$ to ocena jakościowa akcji a wykonanej w stanie s, $N(s)$ oznacza liczbę poprzednich wizyt algorytmu w stanie s, $N(s,a)$ to liczba to liczba tych wizyt zakończonych wyborem akcji a, zaś C to współczynnik definiujący jak duży wpływ na wybór wyniku ma eksploracja (może być rozumiany jako „ciekawość" algorytmu). Na potrzeby tego badania, zgodnie z popularną praktyką, przyjęto $C = \sqrt{2}$.

1.4 Usprawnienia UCT

W ramach badań, podstawowa wersja metody MCTS z algorytmem UCT została porównana z dwoma jej ulepszeniami oraz klasycznym podejściem heurystycznym. Obie modyfikacje zaczerpnięte zostały z literatury. Ich opis przedstawiają sekcje 1.4.1 oraz 1.4.2

1.4.1 UCB1-tuned

Bezpośrednim skutkiem powiązań między algorytmem UCT oraz UCB (wspomnianym w sekcji 1.3) jest dokładne przeniesienie modyfikacji UCB na implementowany algorytm UCT. Jednym z takich ulepszeń jest algorytm **UCB1-tuned**. Modyfikacja ta skupia się na „ciekawości" algorytmu określanej wartością parametru C, występującego we wzorze 1. Stała wartość jest w tym przypadku zastąpiona wartością określoną za pomocą wzorów 2 oraz 3:

$$C = \sqrt{min\left\{\frac{1}{4}, V_i(n_i)\right\}} \tag{2}$$

$$V_i(n_i) = \left(\frac{1}{n_i}\sum_{m=1}^{n_i} X_{i,m}^2\right) - \overline{X_{i,n_i}}^2 + \sqrt{\frac{2ln(n)}{n_i}} \tag{3}$$

Intuicyjnie, modyfikacja ta powinna pozytywnie wpływać na jakość rozwiązania, poprzez lepsze dostosowanie wyborów algorytmu do obecnej sytuacji w drzewie gry.

1.4.2 Transpozycje i grupowanie ruchów

Modyfikacja polega na zamianie drzewa na acykliczny graf skierowany, w którym znajdować się będą stany gry. Z racji, że plansza do gry w Othello jest symetryczna, często może dochodzić do sytuacji w których jedna konfiguracja planszy może występować w drzewie gry więcej niż raz, lecz poddana obrotowi lub różniąca się tylko kolorami (tzn. jedna konfiguracja może być otrzymana „odwracając" kolory pionów oraz gracza wykonującego ruch w drugiej). Ponadto wiele konfiguracji może być osiągalnych za pomocą różnych sekwencji ruchu. W takim wypadku, jeżeli dla danej sytuacji w grze, w drzewie znajduje się już węzeł z analogiczną sytuacją, to zamiast dodawać do drzewa nowy węzeł jako potomka, w węźle rodzica umieszczany jest wskaźnik na węzeł już obecny w drzewie. Dzięki temu każdy stan gry jest umieszczany w drzewie tylko raz, a co za tym idzie statystyki dla losowych symulacji będą zbierane wspólnie i nie będą zależne od położenia węzła w drzewie. Powinno to nie tylko zmienić ilość węzłów w strukturze (która w sumie nie będzie już drzewem, a grafem), lecz także poprawić działanie dzięki współdzieleniu informacji o symulacjach.

1.5 Podejście heurystyczne

Zastosowane podejście heurystyczne opiera się na wykorzystaniu algorytmu *alpha–beta pruning*. Algorytm ten używany jest często do przeszukiwania drzew gry, w szczególności w sytuacjach gdy znane jest całe drzewo. Dla takich przypadków, algorytm zawsze znajduje najlepsze rozwiązanie. Ponadto, w takich sytuacjach jest on szybszy od algorytmów takich jak *backtracking*, dzięki wprowadzeniu ograniczeń oznaczanych przez α (*alpha*) oraz β (*beta*), używanych do „obcinania" niektórych ścieżek, niepozwalających na uzyskanie lepszego rezultatu. W przypadku opisywanego zadania, skuteczność *alpha–beta pruning* będzie zdecydowanie niższa, gdyż wygenerowanie i analiza niemal pełnego drzewa gry w Othello jest zadaniem zbyt czasochłonnym dla normalnych komputerów. Z tego też powodu zaimplementowany algorytm bada wyłącznie określoną ilość kolejnych poziomów drzewa gry, a w najniższym z nich pozycja oceniana jest z użyciem pomocniczej funkcji.

Wspomniana funkcja przyporządkowuje każdemu polu na planszy pewną wartość określoną przez oceną ekspertów i badaczy gry w Othello. Następnie, pozycja każdego gracza zostaje oceniona jako suma wartości z pól zajętych przez piony gracza w danej pozycji. Wyższa wartość tej funkcji oznacza lepszą (heurystycznie) pozycję. Wartości pól zdefiniowane są poprzez kwadratową macierz `WEIGHTS`, o wymiarach równych wymiarom planszy. Wartość funkcji dla pola o współrzędnych i, j jest równa `WEIGHTS[i][j]`. Macierz z wagami przedstawiona jest na rysunku 4. Duże wartości w rogach planszy mają promować wybór tych pól, gdyż są one bezpieczne przed przejęciem.

```
WEIGHTS = [
    120, -20,  20,   5,   5,  20, -20, 120,
    -20, -40,  -5,  -5,  -5,  -5, -40, -20,
     20,  -5,  15,   3,   3,  15,  -5,  20,
      5,  -5,   3,   3,   3,   3,  -5,   5,
      5,  -5,   3,   3,   3,   3,  -5,   5,
     20,  -5,  15,   3,   3,  15,  -5,  20,
    -20, -40,  -5,  -5,  -5,  -5, -40, -20,
    120, -20,  20,   5,   5,  20, -20, 120,
]
```

Rysunek 4: Macierz wag pól na planszy.

2 Przegląd literatury

Artykułem, który był pomocny przy poszukiwaniach modyfikacji, był przegląd prac naukowych dotyczących Monte Carlo Tree Search: *A Survey of Monte Carlo Tree Search Methods* [2]. Był on między innymi uzupełnieniem informacji z wykładu dotyczących pierwszej modyfikacji czyli **UCB1**. Ponadto był inspiracją dla sformułowania drugiej modyfikacji.

Dodatkową pracą pomocną przy tworzeniu drugiej modyfikacji był artykuł *Transpositions and move groups in Monte Carlo tree search* [3]. Praca ta dokładniej opisuje zastosowanie skierowanych acyklicznych grafów jako modyfikację UCT. Zaprezentowane w niej eksperymenty pokazują, że metoda ta może dać bardzo dobre rezultaty. Sformułowana modyfikacja łączy opisane w artykule idee transpozycji (możliwości różnych sekwencji ruchów do doprowadzenia do tej samej sytuacji) i grupowania ruchów (łączenia podobnych ruchów w jeden).

Dodatkowo przy opisie zasad gry Othello pomocny przydał się blog autorstwa Marcina Maja *https://bonaludo.com/* [1], na którym opisane zostają różnice pomiędzy grami Othello i Reversi i zasady tej pierwszej.

3 Hipotezy badawcze

- Podstawowa wersja MCTS pokonuj w bezpośredniej rywalizacji algorytm heurystyczny.
- Modyfikacja UCB1-tuned jest w bezpośredniej rywalizacji lepsza niż podstawowa wersja MCTS oraz algorytm heurystyczny.
- Najlepsze efekty daje modyfikacja Transpozycje i grupowanie ruchów, pokonując wszystkie pozostałe wersje w rozgrywkach bezpośrednich.
- Heurystyczny algorytm jest pokonywany przez średnio-zaawansowanego gracza w znacznej większości pojedynków.
- Wszystkie implementacje oparte na bazie MCTS prezentują poziom podobny do średnio-zaawansowanego gracza. Oznacza to, że podobną część pojedynków wygrywają gracze oparci na SI oraz gracz prawdziwy.

4 Eksperymenty

Hipotezy zweryfikowane zostały poprzez przeprowadzenie dwóch rodzajów eksperymentów: bezpośrednich pojedynków między zaimplementowanymi SI oraz pojedynków między każdą z SI a średnio-zaawansowanym graczem.

4.1 Bezpośrednie pojedynki

Pierwszą częścią testów były mecze, w których ruchy obu graczy były wyznaczane przez zaimplementowane algorytmy. W stosunku do początkowych planów zmianie uległa jednak ilość spotkań rozgrywanych przez każdą parę algorytmów. Początkowo miały to być 2 bądź 3 spotkania, z losowym wyborem gracza rozpoczynającego rozgrywkę w 1 i 3 spotkaniu. Po przemyśleniu tego podejścia, uznano je jednak za błędne. W odróżnieniu od pojedynków między rzeczywistymi graczami, algorytmom nie robiło różnicy jakim kolorem pionów rozgrywane było pierwsze spotkanie. Ponadto, przy zadanym ziarnie generatora liczb pseudolosowych, wynik trzeciego spotkania był jednakowy z wynikiem jednego z poprzednich (w zależności od wylosowanych kolorów). Powodowało to wyłącznie niepożądane, losowe zaburzenie wyników. Reasumując, rozegrane zostały wyłącznie po 2 mecze (różnymi kolorami) w każdej parze algorytmów. Tego typu spotkania zostały następnie powtórzone dla 5 różnych ziaren generatorów liczb pseudolosowych. Każde wygrane spotkanie „nagradzane" było 1 punktem dla zwycięskiego algorytmu, zaś remis dawał po $0,5$ punktu obu graczom. Poniższa tabela 1 prezentuje podsumowanie rezultatów przeprowadzonych testów.

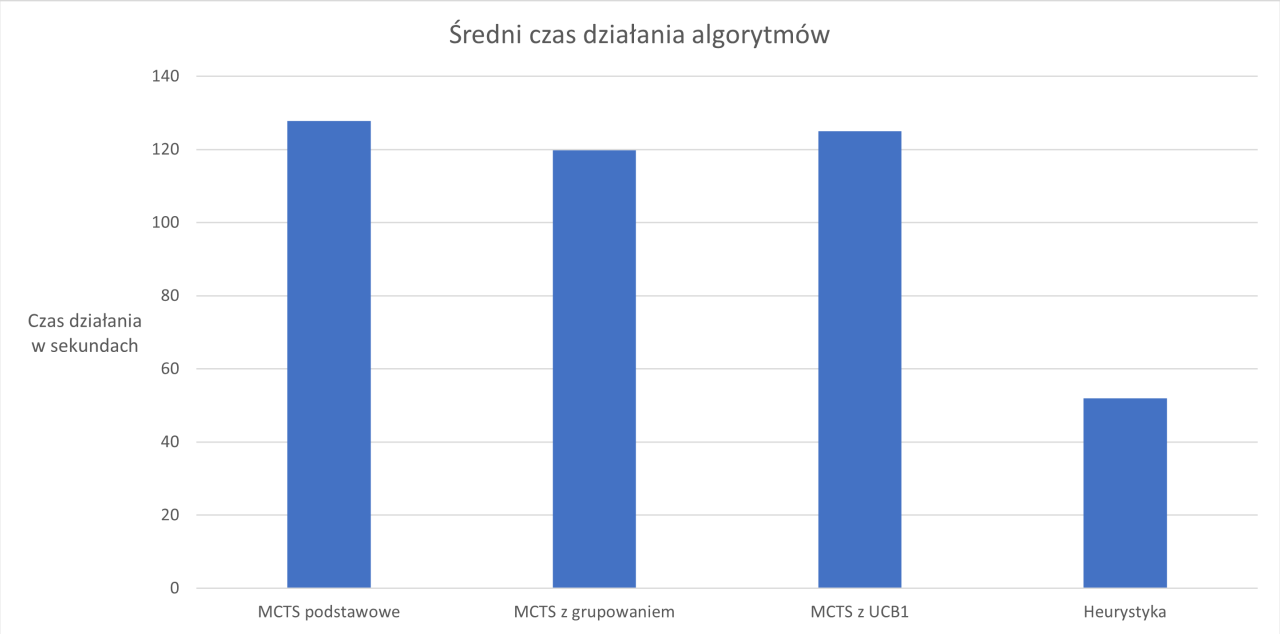
Pozycja	Algorytm	Punkty	Zdobyte	Stracone	Bilans
1	MCTS podstawowe	**20,5**	1062	856	+206
2	MCTS z grupowaniem	**18,0**	1038	882	+156
3	MCTS z UCB1	**12,0**	908	1004	-96
4	Heurystyka	**9,5**	822	1088	-266

Tablica 1: Ranking algorytmów na podstawie przeprowadzonych testów

Wszystkie testy zostały przeprowadzone dla stałych wartości limitu poziomów analizowanych przez *alpha-beta prunning* w algorytmie heurystycznym oraz liczby symulacji uruchamianych w *MCTS*. Były to wartości odpowiednio 6 oraz 300. Wartości te zostały eksperymentalnie wybrane jako kompromis między poziomem gry a czasem działania algorytmów. Czasy działania były zresztą kolejnym badanym parametrem. W każdym spotkaniu, dla obu algorytmów, mierzony był sumaryczny czas wykorzystany przez dany algorytm na wybieranie kolejnych ruchów. Średnia względem wszystkich rozegranych przez dany algorytm spotkań uwzględniona została w tabeli 2 oraz na wykresie na rysunku 5. Maszyna używana w testach wyposażona była w sześciordzeniowy i dwunastowątkowy procesor AMD Ryzen 3600 o taktowaniu $3.6GHz$ oraz $48GB$ pamięci RAM o taktowaniu $3200MHz$.

Algorytm	Średni czas działania [s]
MCTS podstawowe	127,81
MCTS z grupowaniem	119,76
MCTS z UCB1	125,02
Heurystyka	51,99

Tablica 2: Średni czas działania algorytmów w przeprowadzonych testach



Rysunek 5: Średni czas działania algorytmów

Podsumowując wyniki testów, najlepszym algorytmem okazał się podstawowy algorytm *Monte Carlo Tree Search* używający *UCT*. Stan ten wynika ze sposobu implementacji, a w szczególności jednego z dodatkowych usprawnień wprowadzonych w algorytmie z grupowaniem. Aby wzmocnić główny cel wspomnianego ulepszenia, czyli poprawienie wydajności algorytmu, dodatkowo podjęto decyzję o zmniejszeniu liczby przeprowadzanych symulacji w pierwszej fazie gry. W teorii powinno to widocznie przyspieszyć działanie algorytmu, przy nieznacznym spadku jakości. Finalnie, przyspieszenie względem podstawowej wersji MCTS wyniosło średnio ok. 8 sekund (niecałe 6%), zaś rezultaty zmodyfikowanej wersji uplasowały ją na drugim miejscu rankingu.

Trzecie miejsce zajęła SI korzystająca z MCTS ulepszonego poprzez wykorzystanie *UCB1*. Przeprowadzone testy poddały zatem pod wątpliwość sens implementacji tej bardziej skomplikowanej wersji. Przy pomijalnym zysku w kwestii czasu działania, osiągnięte zostały znacznie gorsze rezultaty.

Zgodnie z przewidywaniami, najgorsze rezultaty zanotował algorytm heurystyczny. Z drugiej strony, średni czas jego działania był ponad dwukrotnie krótszy niż w przypadku algorytmów wykorzystujących MCTS. Właściwość ta wynikała jednak z dobranych wartości parametrów. Analizowanie wyłącznie 6 kolejnych kroków (przy ok. 60 ruchach w trakcie rozgrywki), znacznie okroiło możliwości algorytmu. Jednocześnie, zwiększenie wartości parametru do wartości 7 zwiększało czasy testów kilkukrotnie, tak iż heurystyka potrzebowała widocznie więcej czasu niż pozostałe algorytmy. Z tego powodu, uznano użycie wartości parametru równej 6 za bardziej reprezentacyjne.

Potencjalnym czynnikiem, który mógł wpływać na osiągane wyniki były kolory używanych pionów. W większości gier dwuosobowych, gracz rozpoczynający rozgrywkę (w Othello grający czarnymi) jest w pewnym stopniu faworyzowany. Z tego powodu wyniki meczów zostały także pogrupowane względem kolorów, co przedstawia tabela 3. Osiągnięte rezultaty były jednak zbliżone, co przy nieznacznej liczbie testów, zawierających element losowości, uznano za zbyt małą podstawę do wyciągania wniosków dotyczących przewagi danego koloru. Co ciekawe, nieznacznie lepsze wyniki osiągał gracz biały, czyli wykonujący ruch jako drugi.

Pozycja	Kolor	Punkty	Zdobyte	Stracone	Bilans
1	Biały	**32,5**	1945	1885	+60
2	Czarny	**27,5**	1885	1945	-60

Tablica 3: Wyniki przeprowadzonych testów z podziałem na kolory

4.2 Pojedynki z graczem

Drugą częścią testów były pojedynki z prawdziwym graczem. W tym przypadku mecze rozgrywane były na analogicznych zasadach jak w części pierwszej, z tym że ruchy jednego z graczy zawsze były wyznaczane przez prawdziwego, średnio-zaawansowanego gracza. Testy te pozwoliły określić ogólną skuteczność zaimplementowanych algorytmów. Również w tej części rozegranych zostało kilka meczów, jednak z powodu ograniczeń czasowych, ich ilość była znacznie mniejsza niż w poprzednim etapie. Dodatkowo, zbyt duża ilość meczów testowych mogłaby doprowadzić, że znudzony ludzki gracz wykonywałby ruchy szybko, bez odpowiedniego przemyślenia. W związku z tym, gracz rozegrał 3 mecze BO3 (*Best of 3*, do 2 zwycięstw) z każdym botem. Wszystkie parametry testów były takie same jak w przypadku pojedynków botów z botami.

Wyniki przedstawione zostały w tabeli 4. Można zauważyć na niej, że algorytm heurystyczny, podobnie jak w pojedynkach z innymi algorytmami, sprawił się ewidentnie najgorzej i nie był w stanie wygrać żadnej rundy. Nieco lepiej sprawiło się usprawnienie UCB1-tuned, gdyż było ono w stanie wygrać jedno BO3. Podstawowa wersja UCT sprawiła się nieznacznie lepiej od wersji z wykorzystanym grupowaniem, gdyż pomimo że obie wersje wygrały 2 BO3 i zremisowały jedno, to podstawowa wersja odniosła jedno zwycięstwo 2-0. Znaczna przewaga algorytmu UCT jest o tyle imponująca, że gry z nim były rozgrywane pod koniec testów, a poziom gracza stawał coraz lepszy z każdym meczem. Patrząc na podział punktów na kolory przedstawiony w tabeli 5, można zauważyć zdecydowanie większą przewagę dla białego koloru niż dla pojedynków pomiędzy sztucznymi inteligencjami.

Pozycja	Algorytm	Punkty	Zdobyte	Stracone	Bilans
1	MCTS podstawowe	**5,5**	316	196	+120
2	MCTS z grupowaniem	**5,5**	294	282	+12
3	MCTS z UCB1	**3**	226	286	-60
4	Heurystyka	**0**	102	282	-180

Tablica 4: Ranking algorytmów na podstawie przeprowadzonych pojedynków z ludzkim graczem

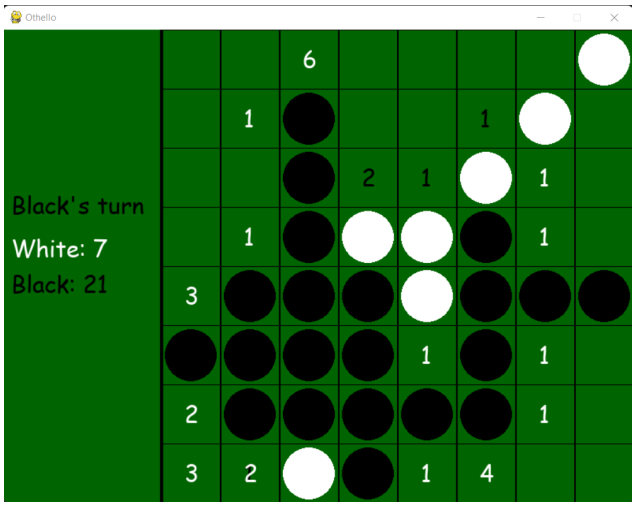
Pozycja	Kolor	Punkty	Zdobyte	Stracone	Bilans
1	Biały	**19**	1056	928	+128
2	Czarny	**12**	928	1056	-128

Tablica 5: Wyniki przeprowadzonych pojedynków botów z graczem z podziałem na kolory

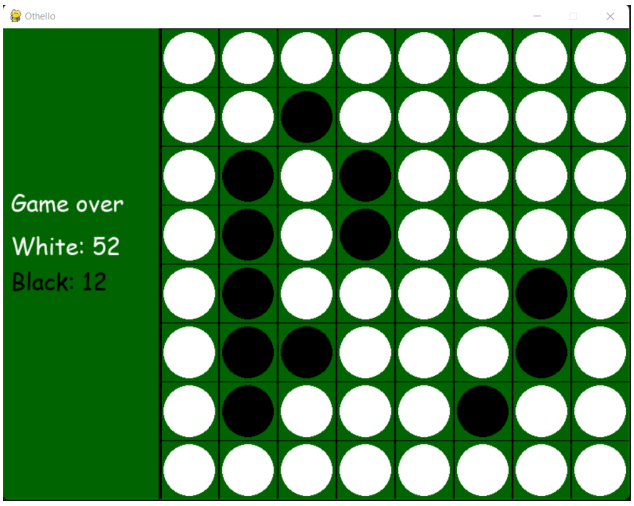
4.3 Sposób gry sztucznej inteligencji

Tak jak wspomniano, algorytm UCT nie przegrał żadnego z 3 BO3 rozegranych z prawdziwym graczem. Warto jednak pochylić się nad tym, w jaki sposób sztuczna inteligencja jest w stanie osiągać tak dobry wynik. Podczas większości pojedynków, ludzki gracz do bardzo późnego etapu gry utrzymywał przewagę posiadanych dysków. Jednakże, w Othello sytuacja może ulec zmianie bardzo szybko i nawet przewaga ponad 20 pionków może zostać stracona w kilka ruchów. W wielu grach, gdy dochodziły one do finalnej fazy, pomimo dużej przewagi punktów graczowi zaczynało brakować ruchów dających mu więcej niż jedno przejęcie. Z kolei sztuczna inteligencja posiadała wiele ruchów przejmujących ok 8-10 dysków. Dzięki temu była ona w stanie szybko przechylić szale zwycięstwa na swoją stronę. Przykład takiej sytuacji w grze został przedstawiony na rysunkach 6 i 7. Liczby widoczne na polach oznaczają ile pól zostanie przejętych przy umieszczeniu piona na tym polu. Na przedstawionych rysunkach gracz grał czarnymi dyskami, a algorytm UCT białymi. Patrząc na dostępne ruchy, można łatwo zauważyć, że gracz miał ich dostępnych zdecydowanie mniej. Inną obserwacją, co prawda niewidoczną na wspomnianych rysunkach, jest fakt, że algorytm UCT nie zawsze skupiał się na przejmowaniu

pól w rogach. Dało się również znacznie zauważyć, że sztuczna inteligencja nawet posiadając możliwość przejęcia bardzo korzystnych pól, odwlekała to w czasie, jeśli wiedziała, że gracz nie jest w stanie ich przejęć. Dzięki temu bardzo skutecznie zawężała ilość możliwych do dokonania przez gracza ruchów (na rysunku 6 można to zauważyć na nieprzejętym polu oznaczonym białym 6).



Rysunek 6: Sytuacja we wczesnej fazie gry.



Rysunek 7: Sytuacja w chwili zakończenia gry.

5 Implementacja

Projekt został zaimplementowany przy użyciu języka *Python*, wykorzystując biblioteki *numpy*, *math*, *random* a także *sys* oraz *timeit*.

Gra udostępnia interfejs graficzny umożliwiający rozgrywkę pomiędzy dwoma graczami, gdzie każdy z nich może być sterowany przez jeden z zaimplementowanych algorytmów bądź przez użytkownika. Konfiguracja rozgrywki odbywa się w oparciu o plik konfiguracyjny w formacie JSON. Do stworzenia interfejsu graficznego wykorzystana została biblioteka *pygame*.

5.1 Konfiguracja

W celu skonfigurowania rozgrywki, należy dokonywać zmian w pliku *config.json*, umieszczonym w jednym katalogu z głównym skryptem. Plik konfiguracyjny powinien zostać stworzony w zgodzie z opisem w formacie *schema*, zawartym w pliku *config-schema.json*. Ponadto, szczegółowy opis parametrów konfiguracyjnych przedstawiono poniżej.

- *game_type* - typ gry: „match” dla pojedynczej gry między dwoma graczami lub „tournament” dla turnieju BO3 rozgrywanego między wszystkimi graczami na zasadzie każdy z każdym.
- *game_repetitions* - opcjonalny (domyślnie: 1) parametr, określający ile razy testy powinny zostać powtórzone (dla różnych ziaren generatora liczb pseudolosowych).
- *seed* - opcjonalny parametr, inicjujący generator liczb pseudolosowych, który następnie generuje kolejne ziarna dla kolejnych powtórzeń gry.
- *heuristic_simulation_depth* - opcjonalny parametr (domyślnie 6), określający ile kolejnych poziomów powinien analizować *alpha-beta pruning*.
- *mcts_simulation_count* - opcjonalny parametr (domyślnie 300), określający ile symulacji powinno być przeprowadzonych w algorytmach bazujących na MCTS.
- *show_visualisation* - opcjonalny parametr (domyślnie *true*), określający czy powinno zostać otworzone okienko z wizualzacją przebiegu gry. Ignorowane jeśli któryś z graczy jest człowiekiem (wtedy wizualizacja jest pokazywana zawsze).
- *output_file* - opcjonalny parametr zawierający ścieżkę do pliku wyjściowego z rezultatami kolejnych gier.
- *players* - tablica zawierająca obiekty opisujące kolejnych graczy, z których każdy ma maksymalnie dwa atrybuty:

– *player_ type* - algorytm używany do symulacji ruchów, lub „user” dla realnego gracza. Dopuszczalne wartości: „user”, „simple_heuristic” (wyłącznie ocena najbliższego ruchu na podstawie funkcji heurystycznej z sekcji 1.5), „heuristic”, „random” (losowe dopuszczalne ruchy), „mcts_uct”, „mcts_ucb1”, „mcts_grouping”.
– *player_ color* - opcjonalny parametr określający kolor pionów gracza („black” lub „white”). Parametr ignorowany dla trybu turniejowego lub gdy nie jest określony dla wszystkich graczy w spotkaniu.

6 Podsumowanie

W wyniku przeprowadzonych testów, udało się, z różnymi wynikiami, zweryfikować prawdziwość postawionych hipotez. Zgodnie z oczekiwaniami, podstawowa wersja MCTS pokonała w bezpośredniej rywalizacji algorytm heurystyczny, który okazał się najgorszy spośród zaimplementowanych algorytmów.

Z kolei wbrew oczekiwaniom zaprezentowały się przedstawione modyfikacje MCTS. Obie osiągnęły rezultaty gorsze niż podstawowa wersja. Jednocześnie, na korzyść ulepszenia z transpozycją i grupowaniem ruchów można zaliczyć nieznaczne przyspieszenie tego algorytmu względem wersji podstawowej.

Porównanie algorytmów z żywym graczem również pozwoliło na potwierdzenie hipotez. Zgodnie z oczekiwaniami, algorytm heurystyczny okazał się gorszy od średnio-zaawansowanego gracza w każdym pojedynku. Z nawiązką została za to potwierdzona hipoteza dotycząca rezultatów MCTS w pojedynkach z graczem. O ile najgorszy wariant (tj. usprawnienie przez wykorzystanie UCB1-tuned) prezentował poziom porównywalny z graczem, to pozostałe modyfikacje znacząco go przewyższały, wygrywając w każdym pojedynku.

Innym aspektem, który nie został początkowo uwzględniony, jednak w wyniku testów okazał się potencjalnie interesującym do dalszej analizy, jest porównanie osiągnięć algorytmów w zależności od kolorów pionów. Przeprowadzone testy wskazują na nieznaczną przewagę osoby grającej pionami białymi. Ograniczony czas nie pozwolił jednak przebadać tego aspektu w sposób wystarczający do wyciągania wniosków.

Bibliografia

[1] Marcin Maj. *Reversi i othello to dwie różne gry. Poznajcie ich różne zasady.* `https://bonaludo.com/2016/01/29/reversi-i-othello-to-dwie-rozne-gry-poznajcie-ich-zasady/`. 2018.

[2] Cameron Browne i in. „A Survey of Monte Carlo Tree Search Methods”. W: *IEEE Transactions on Computational Intelligence and AI in Games* 4:1 (mar. 2012), s. 1–43. DOI: `10.1109/TCIAIG.2012.2186810`.

[3] Benjamin E. Childs, James H. Brodeur i Levente Kocsis. „Transpositions and move groups in Monte Carlo tree search”. W: *2008 IEEE Symposium On Computational Intelligence and Games*. 2008, s. 389–395. DOI: `10.1109/CIG.2008.5035667`.