Metody Sztucznej Inteligencji 2

—

Zastosowanie Upper Confidence Bound Applied To Trees do stworzenia sztucznej inteligencji grającej w Taifho dla dwóch graczy

—

Raport końcowy

Paulina Przybyłek oraz Przemysław Chojecki

Numery albumu: 298837 oraz 298814

03.07.2022

Streszczenie

Algorytm Monte-Carlo Tree Search (MCTS) jest powszechnie używany w zadaniu szukania najlepszego ruchu w grach planszowych. Postanowiono zbadać możliwości wykorzystania i usprawnienia podejścia tego algorytmu do implementacji sztucznej inteligencji w grze Taifho, która przypomina warcabo-szachy. W tym celu przygotowano kilka wersji MCTS: z wykorzystaniem algorytmu Upper Confidence Bound Applied To Trees (UCT) lub z jego modyfikacją PUCT („Predictor" + UCT), gdzie sprawdzano podstawową implementację MCTS bądź rozszerzoną o zastosowanie własnych podejść heurystycznych, które miały za zadanie przyśpieszyć jego działanie. Spróbowano znaleźć najlepsze parametry algorytmów oraz sprawdzić działanie MCTS do zasad gry Taifho. Badanie skuteczności jakości przygotowanych sztucznych inteligencji oparto na porównaniu ich między sobą w rozgrywkach turniejowych, a także przy grze z graczem poruszającym się losowo bądź w grze przeciwko człowiekowi. Stwierdzono nieadekwatność podstawowej implementacji MCTS do wybranej gry, ponieważ etap symulacji oparty na losowych ruchach nie był w stanie doprowadzić gry do pozycji terminalnej. Stąd zastosowanie heurystyk okazało się kluczowe. Algorytmy sztucznej inteligencji wykorzystujące podejścia heurystyczne sprawdziły się w grze Taifho wykazując w miarę inteligentne zachowania. Wykazywały się również zachowaniem "fair-play", gdyż nie próbowały uniemożliwić przeciwnikowi wygranej. Jednak efektywność i sposób działania zaimplementowanych algorytmów były wielce zaskakujące. Żaden MCTS nie wygrał rozgrywki przeciwko człowiekowi. W początkowych fazach gry, bierki algorytmu przesuwały się optymalnie do przodu, lecz w końcowym etapie ich ruchy stawały się raczej losowe. Gra, którą można byłoby skończyć w kilka ruchów toczyła się przez kolejne kilkanaście czy kilkadziesiąt. Z tego powodu wybór parametrów czy ustalenie kolejności, który algorytm jest najlepszy nie było możliwe, gdyż przy losowych ruchach szansę na wygraną miały obaj gracze i ten który skończył szybciej miał po prostu „szczęście".

Spis treści

1. Wstęp

Niniejszy dokument zawiera raport końcowy projektu zrealizowanego w ramach przedmiotu *Metody Sztucznej Inteligencji 2*.

Istotą projektu był świadomy i celowy proces badawczy. W raporcie zawarto opis rozwiązania wraz z przeprowadzonymi eksperymentami, które miały na celu zweryfikowanie hipotez postawionych na początku projektu. Na dokument składa się również opis problemu oraz przegląd literatury dotyczący użytych metod i algorytmów, który został uprzednio przedstawiony w konspekcie w nieco zmienionej formie.

1.1. Cel projektu

Celem projektu było stworzenie odpowiedniego algorytmu Sztucznej Inteligencji (ang. Artificial Inteligence, AI) do grania w grę Taifho oraz porównanie ze sobą wykorzystywanych algorytmów, które są różnymi wersjami podejścia Upper Confidence Bound Applied To Trees. Projekt jest o podłożu badawczym, dlatego testom i analizie algorytmów należało poświęcić najwięcej czasu. Stąd też, celem ostatecznym jest sam rozwój autorów, aby mogli oni zapoznać się w szczegółach z działaniem algorytmów AI grających w gry dla dwóch grach z pełną informacją.

1.2. Opis problemu

Sekcja stanowi przypomnienie opisu problemu, tj. krótkie przedstawienie gry Taifho, z konspektu projektu.

1.2.1. Dzieje gry Taifho

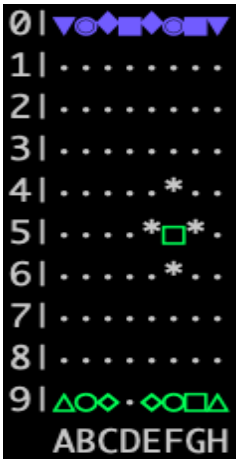
Taifho, zwana czasem szacho-warcaby, jest grą planszową dla dwóch bądź czterech graczy. Wywodzi się z Japonii, a w USA znana jest szerzej pod nazwą Traverse. W 1997 zawitała

do europejskich sklepów z grami planszowymi właśnie pod nazwą Taifho będąc wydana przez szwedzką firmę *Alga*. Wygląd pudełka tego wydania wraz z opisem można zobaczyć pod linkiem *Strona internetowa z opisem zasad gry w Taifho z wydania planszowego z 1999 roku* b.d.

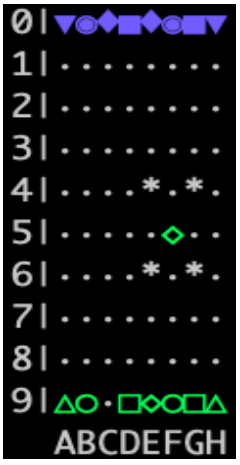
1.2.2. Opis gry Taifho

Wersja dla dwóch graczy rozgrywa się na planszy 8×10. Obaj zawodnicy mają po 8 pionków, po 2 z każdego rodzaju: koło, kwadrat, romb i trójkąt. Na początku rozgrywki zawodnicy ustawiają swoje pionki na odpowiadającej graczowi krawędzi planszy długości 8 (z tego powodu na początku rozgrywki środkowa część planszy 8x8 jest pusta). Wszystkie pionki pozostają na planszy do końca rozgrywki. Celem gracza jest ustawienie wszystkich swoich pionków na pozycji startowej gracza przeciwnego.

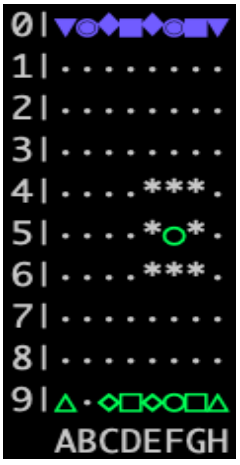
Gracze ruszają się na zmianę, a każdy pionek może poruszać się o jeden w kierunku w którym wskazuje bok figury, co pokazano za pomocą symboli * (gwiazdka) na Rysunkach od 1.1 do 1.5. Koło może poruszać się w dowolnym kierunku, kwadrat wzdłuż i wszerz, romb po ukosie, a trójkąt albo po ukosie do przodu, albo po prostej do tyłu. Ruchy te są dozwolone pod warunkiem, że na polu docelowym nie znajduje się inna bierka.



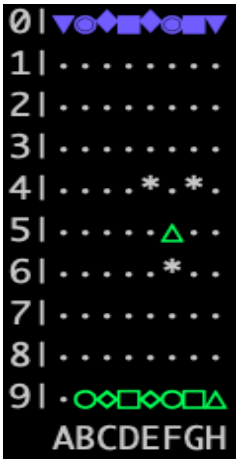
Rysunek 1.1: Legalne ruchy figury kwadrat; takie same dla obu graczy



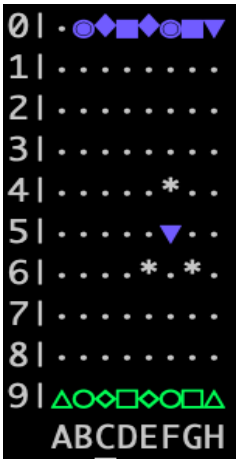
Rysunek 1.2: Legalne ruchy figury romb; takie same dla obu graczy



Rysunek 1.3: Legalne ruchy figury koło; takie same dla obu graczy



Rysunek 1.4: Legalne ruchy figury trójkąt ukazane dla gracza zielonego



Rysunek 1.5: Legalne ruchy figury trójkąt ukazane dla gracza niebieskiego

Istnieje specjalny ruch zwany *skokiem* i polega on na tym, że bierka może „przeskoczyć" inną bierkę pod warunkiem, że skok będzie wykonany o nieparzystą liczbę pól w kierunku poruszania się bierki oraz że „przeskakiwana" bierka będzie znajdować się dokładnie na środkowym polu

skoku. Bierką „przeskakiwaną” może być zarówno bierka zawodnika wykonującego skok, jak i bierka przeciwna. Możliwym jest wykonanie kilku skoków w ramach jednego ruchu, ale muszą być one wykonane tą samą bierką. Nie można jednak kończyć ruchu w pozycji, z której bierka zaczynała swoje skoki.

Ostatnią, wyjątkową zasadą jest zasada, którą nazywamy „fair-play”, a która mówi: „nie można uniemożliwić przeciwnikowi zwycięstwa”. W *Strona internetowa z opisem zasad gry w Taifho z wydania planszowego z 1999 roku* (b.d.) piszą po prostu „Players can not force a draw”. Przyjęto następującą interpretację: jeśli gracz będzie blokował swoje pola startowe przed dostępem dla przeciwnika - natychmiast przegrywa. Dla przykładu sytuacja na Rysunku 1.6 jest pozycją terminalną, zwycięską dla gracza niebieskiego, mimo, że nie ustawił on wszystkich swoich bierek na polach startowych gracza zielonego. Stąd Taifho nie przewiduje zakończenia gry remisem. Gra zawsze kończy się zwycięstwem jednego z graczy.

Rysunek 1.6: Przykład zastosowania zasady „fair-play”. Gracz zielony uniemożliwia zwycięstwo graczowi niebieskiemu, dlatego gra się kończy i wygrywa gracz niebieski.

2. Opis algorytmów

W niniejszym rozdziale znajduje się skrócony opis algorytmu Monte-Carlo Tree Search (MCTS), który będzie wykorzystany jako główna część AI grającego w Taifho. Dodatkowo opisano algorytm Upper Confidence Bound Applied To Trees (UCT), który często jest z powodzeniem wykorzystywany jako jeden z etapów algorytmu MCTS. Na koniec rozdziału przedstawiono dodatkowe modyfikacje, które zostaną również zaimplementowane jako alternatywna wersja AI.

2.1. Motywacja

Oczywiste jest, że najlepszym sposobem na znalezienie optymalnego ruchu w danym stanie gry jest przeszukanie wszystkich możliwych rozwinięć aż do pozycji terminalnych, a potem zastosowanie zasady min-max. Niestety, dla wielu zadań tego typu podejście jest niemożliwe do zrealizowania, ze względu na wielkość drzewa gry.

Od algorytmu sztucznej inteligencji grającego w grę oczekuje się, że będzie wykonywał dobre ruchy, czyli takie, które najpewniej będą go doprowadzać do zwycięstwa. Wymaga się jednak, aby nie zajmowało mu to zbyt dużo czasu, gdyż chce się doczekać końca rozgrywki. Z tego powodu pożądanym zachowaniem byłoby, gdyby algorytm AI potrafił już po krótkim czasie działania mieć jakąś estymację jakości możliwych do wykonania ruchów. Natomiast z czasem działania by ową estymację poprawiał. Dzięki temu można by w dowolnym momencie wykonać ruch.

Algorytm MCTS jest przykładem algorytmu spełniającego powyższe wymagania. Jest on powszechnie używany w zadaniu szukania najlepszego ruchu w grach planszowych.

2.2. MCTS

MCTS był wprowadzony przez Browne i in. (2012) i odpowiada na potrzeby opisane w poprzedniej sekcji. Polega on na sekwencyjnym rozszerzaniu drzewa gry od korzenia (pozycji startowej), w dół drzewa. Algorytm ten zdobył ogromną popularność i jest powszechnie używanym podejściem do tego celu. Jest m.in. używany w znanym na cały świat algorytmie „AlphaGo" opracowanym przez Silver i in. (2016), który pokonał mistrza świata w grę Go w słynnym meczu z 2016 roku. O owym meczu można przeczytać w artykule Mullen (2016).

Algorytm składa się z czterech etapów: **selekcja**, **ekspansja**, **symulacja** i **propagacja wsteczna wyniku**. Zaczyna on swoje działanie z drzewem odwiedzonych pozycji, w którym znajduje się jedynie węzeł korzenia, czyli pozycja gry, na którym MCTS został wywołany. W każdym momencie działania algorytm trzyma swoje estymacje na temat jakości poszczególnych ruchów, a dokładniej, jakości pozycji, do których ruchy te prowadzą.

W **etapie selekcji** algorytm podróżuje w dół drzewa rozważonych pozycji w następujący sposób: jeżeli jest pozycja, do której reguły gry pozwalają dotrzeć w jednym kroku, ale MCTS jeszcze tam nie dotarł, to idzie tam i przechodzi do następnego etapu. W przeciwnym przypadku w jakiś sposób wybiera jaki ruch tym razem rozważyć i kontynuuje etap selekcji rekurencyjnie. UCT jest przykładową strategią jaką można wykorzystać w tym etapie. Jest on opisany w następnym podrozdziale.

W poprzednim etapie algorytm MCTS dotarł do nowej pozycji gry, w której nigdy nie był. **Etap ekspansji** polega na dodaniu tego węzła do drzewa rozważonych pozycji.

Etap symulacji dotyczy symulowania losowych ruchów obu graczy. Pozycje, do których MCTS dotrze w tym etapie, NIE są uznawane za „odwiedzone". Po dotarciu do pozycji końcowej znany jest wynik gry.

Etap propagacji wstecznej wyniku polega na zapamiętaniu przez algorytm wyniku gry otrzymanym w poprzednim kroku i wpisaniu go do wszystkich węzłów na ścieżce od korzenia, do nowo dodanego węzła w etapie ekspansji.

W każdej takiej czteroetapowej iteracji algorytmu MCTS rozważane drzewo gry rozszerza się o jeden węzeł, a estymacja jakości ruchów się poprawia. Działanie algorytmu MCTS można przerwać po dowolnej liczbie iteracji.

2.3. Podstawowy UCT

Algorytm UCT jest modyfikacją strategii UCB (Peter Auer, Cesa-Bianchi i Fischer 2002) zaaplikowaną do podążania w dół drzewa. Strategia UCB opiera się na twierdzeniu dającemu optymalną strategię grania przy problemie wielorękiego bandyty (opisanym przez P. Auer i in. (1995)). Gra polega na tym, że mamy skończoną liczbę maszyn do grania, gdzie każda zwraca jakąś wypłatę z nieznanego graczowi rozkładu prawdopodobieństwa, być może innym dla każdej z maszyn. Zadaniem gracza jest zmaksymalizowanie osiąganej wypłaty.

Optymalna strategia UCT zaproponowana przez Kocsis i Szepesvári (2006) wygląda następująco: jeśli jakieś dziecko rozważanego węzła nie było jeszcze nigdy odwiedzone, to odwiedź je. Jeśli wszystkie dzieci były już odwiedzone, to wybierz to dziecko a, które maksymalizuje wartość:

$$g(a) = \hat{a} + C \cdot \sqrt{\frac{ln(N)}{N(a)}}$$

gdzie $\hat{a}$ to estymacja wartości dziecka a, N jest liczbą odwiedzin rozważanego węzła, $N(a)$ jest liczbą odwiedzin dziecka a, natomiast $C > 0$ jest parametrem metody.

Algorytm ten można w naturalny sposób zastosować w etapie selekcji algorytmu MCTS przyjmując $\hat{a}$ jako średnią z otrzymanych do tej pory wypłat, oraz przechodząc do etapu symulacji w chwili trafienia w nowo odwiedzone dziecko.

2.4. PUCT, czyli „Predictor" + UCT

Większość współczesnych implementacji opiera się na jakimś wariancie UCT, będącym modyfikacją podstawowego podejścia. Niektóre modyfikacje UCT skupiają się na specjalnych ruchach typowych dla danych gier. Uważa się, że preferowanie takiego ruchu może polepszyć skuteczność algorytmu AI. Dla przykładu Wang i in. (2018) w swojej pracy o warcabach skupili się na ruchu tworzącym króla i na tej podstawie wypracowali odpowiednią modyfikację UCT.

Jedną ze znanych modyfikacji jest algorytm PUCT, będący zastosowaniem podejścia PUCB (Rosin 2011) do drzew. PUCB oznacza „Predictor" + UCB, gdyż modyfikuje pierwotną politykę wielorękich bandytów UCB, poprzez w przybliżeniu przewidywanie dobrego wyboru na początku sekwencji prób jednorękich bandytów. W projekcie zastosowano algorytm PUCT do preferowania ruchu zwanego *skokiem*, który jest jedynym specjalnym ruchem w Taifho. Naj-

bardziej oczekiwany byłby skok bierki do przodu o jak największą liczbę pól. Tego właśnie dotyczyć będzie heurystyka wyboru predyktorów.

Modyfikacja PUCT polega na przypisaniu pewnej wagi M_a do każdego dziecka a. Wszystkie wagi dzieci rozważanego węzła mają sumować się do wartości $\sum_{a\in\{dzieci\}} M_a = 1$. We wspomnianej już pracy Rosin (tamże), wprowadzającej metodę PUBC, jedną z opisanych metod wyboru był „Uogólniony predyktor Bradleya-Terry'ego" (sekcja 3.1.1 w tamtej pracy). Zmodyfikowana wersja tego wzoru dla rozważanej gry wygląda następująco:

$$M_a = \frac{exp(\frac{1}{K}x_a)}{\sum_{i=1}^{K} exp(\frac{1}{K}x_i)}$$

gdzie wartości x_j dla $1 \leqslant j \leqslant K$ związane są z ruchem bierki na podstawie liniowego przeskalowania wartości z wektora D do przedziału $[\frac{1}{K^3}, 1 - \frac{1}{K}]$, gdzie K to liczba dzieci. Rozważana jest sytuacja, gdy $K > 1$. Przy czym $D = (d(1), \ldots, d(K))$ oznacza wektor różnic odległości od linii startowej przed i po ruchu danej bierki dla wszystkich dzieci, co określa następujący wzór:

$$d(a) = \mu_2 - \mu_1$$

gdzie a dotyczy dziecka, dla którego liczona jest wartość $d(a)$, a wartości μ_1, μ_2 oznaczają odpowiednio odległość ruszającej się bierki od linii startowej przed ruchem i po ruchu w daną stronę. Jeśli ruch nastąpił w poprzek to wartość wynosi 0, jeśli jest do przodu to jest dodatni, a jak bierka się cofnęła to ujemny.

Następnie jest to wykorzystywane do obliczania kary:

$$m(N,a) = \begin{cases} \frac{2}{M_a} \cdot \sqrt{\frac{ln(N)}{N}} & gdy\ N > 1, \\ \frac{2}{M_a} & wpp \end{cases}$$

gdzie N jest liczbą odwiedzin rozważanego węzła, $N(a)$ jest liczbą odwiedzin dziecka a, M_a jest wagą przypisaną do dziecka a. Wówczas wzór wykorzystywany do wyboru dziecka wygląda następująco:

$$g(a) = \hat{a} + C \cdot \sqrt{\frac{ln(N)}{N(a)}} - m(N,a)$$

Wykorzystanie wag w UCT jest swego rodzaju predyktorem. Zastosowanie modyfikacji PUCT w niektórych rozwiązaniach AI odniosło znaczne korzyści. Pewne wariacje PUCT zostały użyte m.in. we wspomnianym już „AlphaGo" a także w „AlphaZero" czy „Leela Chess Zero".

2.5. Ocenianie pozycji nieterminalnych

W niektórych grach istnieje naturalny mechanizm oceny pozycji nieterminalnej. Tak jest np. w Monopoly (liczba pieniędzy posiadanych przez graczy) lub Scrabble (liczba punktów posiadanych przez graczy, na stronie *scrabblemania.pl, oficjalne zasady gry* b.d. opisano, że „Celem gry jest uzyskanie jak najwyższego wyniku"). Jednakże wiele gier nie ma wbudowanych takich zasad.

Możliwość wstępnego ocenienia pozycji jest przydatna w praktycznym zastosowaniu algorytmu MCTS. Dla niektórych gier bowiem etap symulacji jest bardzo długi, co podaje w wątpliwość zależność znalezionego wyniku gry od pozycji dodanej w etapie ekspansji.

Można w takiej sytuacji skrócić etap symulacji i estymować jakość pozycji zgodnie z jakąś heurystyczną oceną. Takie podejście pozwoli dodatkowo na wykonanie większej liczby iteracji algorytmu MCTS w tym samym czasie.

W projekcie rozważono dwie heurystyki oceniające – heurystykę h oraz heurystykę h_G, które opisano poniżej.

2.5.1. Wybrane podejścia heurystyczne

W celu przeciwdziałaniu ogromnej liczbie ruchów losowych na drodze do pozycji terminalnej, wprowadzony zostanie nowy parametr *steps* sterujący, ile losowych kroków będzie wykonanych. Po ich wykonaniu pozycja zostanie oceniona za pomocą heurystycznej funkcji oceny.

Ocena pozycji p będzie następować według następującego wzoru:

$$h(p) = \Big[\sum_{i \in \{\text{bierki gracza}\}} b_i \Big] - \Big[\sum_{j \in \{\text{bierki przeciwnika}\}} b_j \Big]$$

gdzie b_i oznacza odległość bierki i od jej linii startowej. Oznacza to, że jeśli bierki zawodnika będą na swoim docelowym miejscu, to pierwszy nawias będzie wynosił $8 \cdot 9 = 72$. Inny przykład, uwzględniający rozmieszczenie wszystkich bierek, można zobaczyć na rysunku 2.1, gdzie według heurystyki h gracz niebieski ma przewagę.

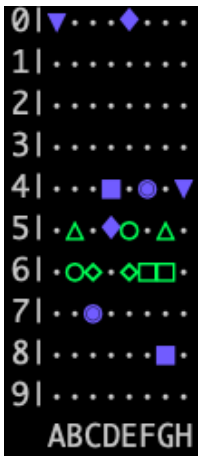
Jednakże, jeśli przyjrzymy się pozycji na rysunku 2.1, to wcale nie jest to takie oczywiste, że gracz niebieski ma przewagę. Gracz niebieski bowiem zostawił swoje dwie bierki na początku planszy i prawdopodobnie później będzie miał trudności w efektywnym przyprowadzeniu ich do celu. Gracz zielony natomiast trzyma swoje bierki blisko, dzięki czemu będą one mogły sobie nawzajem pomagać.

Wypracowano więc drugą heurystykę, która nieliniowo karze za odległość do celu. Ocena ta będzie następować według następującego wzoru:

$$h_G(p) = \Big[\sum_{i \in \{\text{bierki gracza}\}} \frac{b_i \cdot log(9+G)}{log(b_i+G)} \Big] - \Big[\sum_{j \in \{\text{bierki przeciwnika}\}} \frac{b_j \cdot log(9+G)}{log(b_j+G)} \Big]$$

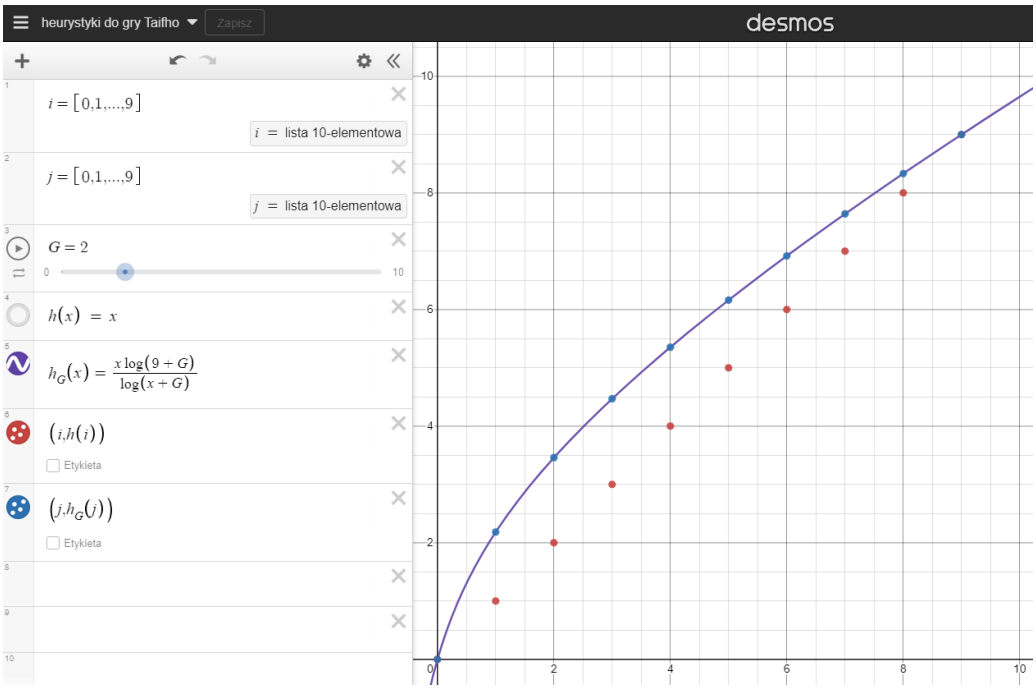
gdzie b_i oznacza odległość bierki i od jej linii startowej, a $G \in (1, \infty)$ jest parametrem, który im mniejszy, tym bardziej karane jest zostawianie bierek „z tyłu".

Wpływ parametru G można prześledzić na stronie *Strona Desmos na której można zobaczyć jaki wpływ ma parametr G na heurystykę h_G* b.d., z której to również pochodzi rysunek 2.5.1 wizualizujący różnicę między h, a h_2. Warto zwrócić uwagę, że wcześniejsza heurystyka h jest granicą wartości heurystyk h_G przy parametrze G zbiegającym do ∞. Heurystyka h_2 ocenia pozycję na rysunku 2.1 jako lekko przeważającą dla gracza zielonego.



Rysunek 2.1: Przykładowy stan gry. Ocena planszy z punktu widzenia gracza zielonego według heurystyki h wynosi $h(p) = [5 \cdot 3 + 3 \cdot 4] - [2 \cdot 0 + 3 \cdot 4 + 5 + 7 + 8] = 27 - 32 = -5$, czyli niebieski ma przewagę. Natomiast według heurystyki h_2 ocena wynosi $h_2(p) \approx +0.21$, czyli gra jest wyrównana, a zielony ma lekką przewagę.

Oczekiwanym jest więc, że heurystyka h_G będzie bardziej preferować wspólne poruszanie się do przodu wieloma bierkami, zamiast zostawiać kilka z nich z tyłu. Jest tak dlatego, że dla $G = 2$, $h_2(0) = 0, h_2(1) \approx 2.18, h_2(2) \approx 3.46$. Oznacza to, że przejście jednej bierki z pozycji startowej na następną linię jest według h_2 warte około 2.18, natomiast przejście z pierwszej na drugą około $3.46 - 2.18 = 1.28$. Heurystyka h_G dla każdego $G > 1$ w ten sposób stopniowo coraz gorzej ocenia kolejne przejścia na następną linię.



Rysunek 2.2: Zdjęcie ze *Strona Desmos na której można zobaczyć jaki wpływ ma parametr G na heurystykę h_G* b.d. Czerwonymi kropkami zaznaczono heurystykę h, a niebieskimi heurystykę h_G dla $G = 2$. Na stronie tej można się przekonać, że $h_1(0) > 2$, a pożądaną wartością jest 0, co jest spełnione dla każdego $G > 1$.

3. Opis rozwiązania

Rozdział krótko przedstawia implementację gry w postaci aplikacji oraz wykorzystane technologie w projekcie.

3.1. Technologia

Gra Taifho została zaimplementowana jako pakiet do języka Python. Sama aplikacja pozwalająca zagrać w Taifho oraz wszystkie wersje algorytmów AI zaimplementowano jako skrypty do języka Python importujące pakiet z grą. Pliki z projektu są dostępne na repozytorium GitHub pod adresem Przemysław b.d. Dzięki tej postaci, użytkownik będzie mógł w prosty sposób zagrać z zaimplementowanym AI. Szczegóły jak to zrobić znajdują się w pliku *README.md* na wspomnianym repozytorium. Dodatkowo repozytorium zawiera wszystkie skrypty i pliki wykorzystane w tym projekcie do weryfikacji hipotez, analiz działania algorytmów, jak i wszystkie wykonane wizualizacje.

Głównym zadaniem była własna implementacja algorytmów i metod użytych w projekcie, dlatego też pakiety z których korzystano należą do bazowych bibliotek pythonowych, jak na przykład:

– numpy – optymalna praca z wektorami liczb,

– copy – kopia obiektu,

– math – funkcje pierwiastka czy logarytmu,

– random – losowanie liczby z danego przedziału, ziarno losowości,

– time – mierzenie czasu działania,

– collections – możliwość używania struktury wbudowanego w Python słownika.

3.2. Gra Taifho

Zaimplementowana gra jest typu człowiek versus komputer, gdzie gracz ma możliwość wyboru AI z jakim chce zagrać. Działa ona w trybie tekstowym (aplikacja konsola) w wersji pojedynczej rozgrywki. Zaimplementowane zostały następujące wersje AI do grania w Taifho:

1. MCTS z podstawowym UCT,
2. MCTS z podstawowym PUCT,
3. MCTS + UCT wykorzystujący heurystykę h,
4. MCTS + UCT wykorzystujący heurystykę h_G,
5. MCTS + PUCT wykorzystujący heurystykę h,
6. MCTS + PUCT wykorzystujący heurystykę h_G,

Szczegóły poszczególnego działania algorytmów czy heurystyk opisano w rozdziale 2. Dodatkowo istnieje możliwość zagrania z silnikiem losowym, który wybiera po prostu losowy ruch ze wszystkich możliwych w danej turze. Przy wyborze odpowiedniego AI użytkownik dostaje możliwość ustawienia parametrów wybranego AI, mianowicie:

– C – parametr występujący w UCT,

– maksymalny czas (w sekundach) dla każdego ruchu silnika,

– *steps* – parametr, będący liczbą losowych kroków w drodze do pozycji terminalnej, po których liczona jest heurystyka,

– G – parametr heurystyki h_G.

Wybór dotyczy tylko tych parametrów, które są potrzebne do danego algorytmu.

W projekcie założono, że gracze mają z góry ustalone startowe położenie bierek. Mianowicie: trójkąt, koło, romb, kwadrat, romb, koło, kwadrat, trójkąt. Gracz ma jedynie wybór koloru bierek, którymi chce rozgrywać, tj. co oznacza tyle czy on będzie zaczynać grę czy komputer.

Aplikacja będzie wyświetlać ankietę z wyborem ustawień gry dla użytkownika i informować co należy wpisać do konsoli oraz w przypadku braku działania użytkownika będzie wyświetlać co teraz się dzieje w środku aplikacji. Pytania w ankiecie dotyczą wprowadzenia własnych wartości bądź wyboru jednej z proponowanych opcji. Aplikacja po przyjęciu parametrów dotyczących

silnika będącego przeciwnikiem gracza, jego własnych parametrów, koloru bierek gracza definiuje i rozpoczyna odpowiednią rozgrywkę. Wówczas gra wyświetla odpowiednie informacje, które dzieją się podczas rozgrywki jak ruchy przeciwnika oraz co i kiedy powinien wykonać sam gracz, aby się ruszyć. Po zakończeniu rozgrywki gracz ma możliwość rozpocząć od nowa bądź zakończyć działanie programu.

4. Eksperymenty

Zaimplementowano sześć wersji algorytmów AI opartych na MCTS do grania w Taifho. Wymieniono je w sekcji 3.2, natomiast opis teoretyczny wykorzystanych metod znajduje się w rozdziale 2. Jak wspomniano we wstępie projekt ma podłoże badawcze, dlatego aby wybrać odpowiedni algorytm AI skupiono się na analizie przygotowanych wersji algorytmów i na porównaniu ich ze sobą. W tym celu przygotowano hipotezy badawcze, które za pośrednictwem testów poddano weryfikacji. Niniejszy rozdział skupia się na przedstawieniu etapu przeprowadzania eksperymentów i analizy stworzonych algorytmów AI.

4.1. Hipotezy badawcze

Sekcja stanowi przypomnienie hipotez z dokumentacji wstępnej pracy.
Postawiono następujące hipotezy badawcze:

1. Etap symulacji MCTS w podstawowym algorytmie UCT będzie działała na tyle długo, że podstawowy UCT będzie niepraktycznym algorytmem AI do gry Taifho.

2. Algorytm UCT będzie grał podobnie niezależnie, czy w implementacji będzie istnieć zasada „fair-play", czy też nie.

3. AI będzie grało najlepiej z wartością parametru C inną niż teoretycznie najlepsze $C = \sqrt{2}$.

4. Najlepsza wartość parametru G dla heurystyki h_G będzie w przedziale $G \in [2, 5]$.

5. Skuteczność zaimplementowanych AI będzie następująca: Najgorzej będzie grał podstawowy UCT, lepiej będzie grał UCT z heurystyką h, jeszcze lepiej będzie grał UCT z heurystyką h_G, a najlepiej będzie grał UCT z modyfikacją PUCT.

6. Algorytmy AI będą w stanie pokonać ludzkiego gracza amatora.

Hipotezy te zostały zweryfikowane poprzez przeprowadzone eksperymenty.

4.2. Metodologia

W tej sekcji opisano szczegółowo jak wykonano zaplanowane eksperymenty. Numer przypisany do eksperymentu odpowiada hipotezie, którą za jego pomocą sprawdzano. Wyniki i wnioski z eksperymentów zawarto w sekcji 4.3.

4.2.1. Ekperyment 1

Ekperyment 1 dotyczył etapu symulacji w algorytmie MCTS. Pierwotnie zakładano, że algorytm losowego poruszania się włączony będzie na kilku wybranych pozycjach. Później zauważono, że wystarczy włączyć go na jednym, gdyż i tak po np. 500 ruchach pozycja jest zupełnie inna i można ją traktować jak „ponowne uruchomienie". Wykonano błądzenie losowe, w którym każdy legalny ruch miał równe prawdopodobieństwo bycia zagranym. Rysunek 4.1 pokazuje efekt uruchomienia tego skryptu po około 1 minucie. Widać na nim, że różnica między planszami po 500 losowych krokach jest ogromna i można to traktować jako ponowne uruchomienie błądzenia na innej pozycji startowej.

Kod wykonanego błądzenia znajduje się na repozytorium Przemysław b.d. w skrypcie 'project2/src/Game_py/basic_MCTS_experiment.py'.

4.2.2. Ekperyment 2

Ekperyment 2 dotyczył sprawdzenia zasady „fair-play". Uruchomiono algorytm MCTS (wykorzystujący UCT z heurystyką h i z parametrami $C = \sqrt{2}$ oraz $steps = 6$) jako gracz zielony na pozycji z Rysunku 1.6. Dano mu „do namysłu" 60 sekund na podjęcie decyzji. Sprawdzono, czy podejmie decyzję o kontynuowaniu blokady, czy też o umożliwieniu przeciwnikowi zwycięstwa. Eksperyment powtórzono kilkakrotnie.

Logikę przeprowadzonego eksperymentu zapisano na repozytorium tamże w skrypcie 'project2/src/Game_py/fair_play_experiment.py'.

4.2.3. Ekperyment 3

W eksperymencie 3 sprawdzano jak wartość parametru C wpływa na jakość AI. Mowa o parametrze C do UCT ze wzoru z sekcji 2.3. Analizowano wartości parametru C z dyskretnego zbioru: $C \in \{1, \sqrt{2}, 2, 3.5, 5\}$. Eksperyment przeprowadzono w dwóch częściach. W pierwszej algorytmy MCTS z heurystyką h z odpowiednią wartością C grały 10 razy przeciwko losowo ruszającemu się przeciwnikowi. Algorytm MCTS zawsze zaczynał jako gracz niebieski, tj. jako

gracz drugi. Następnie oceniono, która z wartości parametru C najszybciej wygrywała. Ocena ta oparta była na średniej liczbie wykonanych ruchów podczas rozgrywki. Najlepsza wartość parametru przechodziła do kolejnej części eksperymentu, w której algorytm z tą wartością grał 10 meczów przeciw wartości $C = \sqrt{2}$, podejrzewanej przed eksperymentem o bycie najlepszą na podstawie artykułów naukowych. Kolejność zaczynania gry była losowana za każdym razem przy innym ziarnie losowości. Wykorzystano do tego funkcję *seed* z pakietu *random*, a wartości liczbowe należały do dyskretnego zbioru: $\{123, 245, 456, 786, 999, 567, 582, 11, 765, 66\}$. Za każdym razem dano algorytmom MCTS czas na wykonanie ruchu równy 1.5 sekundy, natomiast parametr *steps* dla heurystyki h wynosił 6.

Kod wykonanej pierwszej części eksperymentu znajduje się na repozytorium Przemysław b.d. w skrypcie 'project2/src/Game_py/C_parameter_experiment.py'. Natomiast do części drugiej, czyli gry dwóch algorytmów MCTS z różnymi parametrami C, wykorzystano napisany skrypt rozgrywki pomiędzy dwoma AI: 'project2/src/Game_py/tournament_experiment.py'. Skrypt ten był wykorzystywany również do eksperymentu 5, dlatego przed jego uruchomieniem należy sprawdzić jakie algorytmy są aktualnie wpisane do rozgrywki i jaki jest docelowy plik zapisu wyników.

4.2.4. Ekperyment 4

W ekperymencie 4 wybierano najlepszą wartość parametru G do heurystyki h_G. Analizowano wartości parametru G z dyskretnego zbioru: $G \in \{1.1, 2, 3.5, 5, 7, 10, 20\}$. Eksperyment przeprowadzono podobnie do eksperymentu 3, opisanego w podsekcji 4.2.3, jedynie z tą różnicą, że tym razem sprawdzano algorytm MCTS z heurystyką h_G, a wartość C była stała i ustawiona na $C = 3.5$. Dodatkowo tym razem dano algorytmom MCTS czas "do namysłu" 2 sekundy. Wartość parametru *steps* się nie zmieniła i nadal wynosiła 6. W przypadku drugiej części eksperymentu sprawdzano między sobą dwie najlepsze wartości G z części pierwszej.

Kod wykonanej pierwszej części eksperymentu znajduje się na repozytorium tamże w skrypcie 'project2/src/Game_py/G_parameter_experiment.py'. Natomiast do części drugiej analogicznie jak w przypadku eksperymentu 2 wykorzystano skrypt rozgrywki pomiędzy dwoma AI: 'project2/src/Game_py/tournament_experiment.py'.

4.2.5. Ekperyment 5

Eksperyment 5 był wykonany w celu porównania algorytmów AI. Porównanie to odbyło się na zasadzie turnieju między wybranymi najlepszymi wersjami algorytmów. W tym celu

przeprowadzono trzy rozgrywki po 10 meczy każda:

1. pomiędzy algorytmami MCTS z UCT i różnymi heurystykami (h oraz h_G),

2. pomiędzy algorytmami MCTS z PUCT i różnymi heurystykami (h oraz h_G),

3. pomiędzy zwycięzcami z pierwszego i drugiego meczu, tzw. finał turnieju.

Kolejność zaczynania gry była losowana za każdym razem przy innym ziarnie losowości. Wykorzystano do tego funkcję *seed* z pakietu *random*, a wartości liczbowe należały do dyskretnego zbioru: $\{123, 245, 456, 786, 999, 567, 582, 11, 765, 66\}$. Parametr *steps* ustawiono na 6, a czas do podjęcia decyzji o kolejnym ruchu wynosił 2. Wartości parametrów C i G wybrano na podstawie wyników eksperymentów 3 i 4. Na podstawie analizy wyników rozgrywek w turnieju zdecydowano ułożyć kolejnością AI pod względem jakości działania dla wybranej gry.

Logikę turnieju zapisano w postaci skryptu pojedynczej rozgrywki (10 meczy) między dwoma algorytmami AI. Nosi on nazwę ‘project2/src/Game_py/tournament_experiment.py‘. Podobnie jak poprzednie skrypty znajduje się na repozytorium projektu, przed jego uruchomieniem należy sprawdzić jakie algorytmy są aktualnie wpisane do rozgrywki i jaki jest docelowy plik zapisu wyników.

4.2.6. Ekperyment 6

Eksperyment 6 sprawdzał kto jest lepszy w Taifho – człowiek czy algorytm AI. W tym celu skorzystano z aplikacji konsolowej pozwalającej na rozgrywkę człowiek vs komputer (AI), opisanej w sekcji 3.2. Zdecydowano się na rozgrywkę z najlepszym algorytmem, który został wyłoniony w turnieju z eksperymentu 5. Rozgrywkę powtórzono 3 razy dla różnych wartości parametru *steps* – dla $steps = 6$ oraz $steps = 10$. Dodatkowo dano algorytmowi AI więcej czasu na podjęcie decyzji o ruchu, mianowicie wybrano 10 sekund zamiast 2 jak w poprzednich testach. Na koniec odbyto jedną rozgrywkę, gdzie algorytmowi dano aż 60 sekund na każdy ruch.

Skrypt zawierający aplikację konsolową pozwalającą zagrać w Taifho przeciwko wybranemu algorytmowi AI znajduje się na repozytorium pod nazwą ‘project2/src/Game_py/main.py‘. Szczegółowa instrukcja jak zagrać znajduje się na repozytorium projektu w pliku README.md.

4.3. Wyniki

Sekcja przedstawia wyniki przeprowadzonych eksperymentów, ich analizę oraz wyciągnięte wnioski z każdego z nich z osobna. Wykresy przedstawiające wyniki w przystępnej formie wykonano za pośrednictwem języka R (R Core Team 2020) i pakietu *ggplot2*.

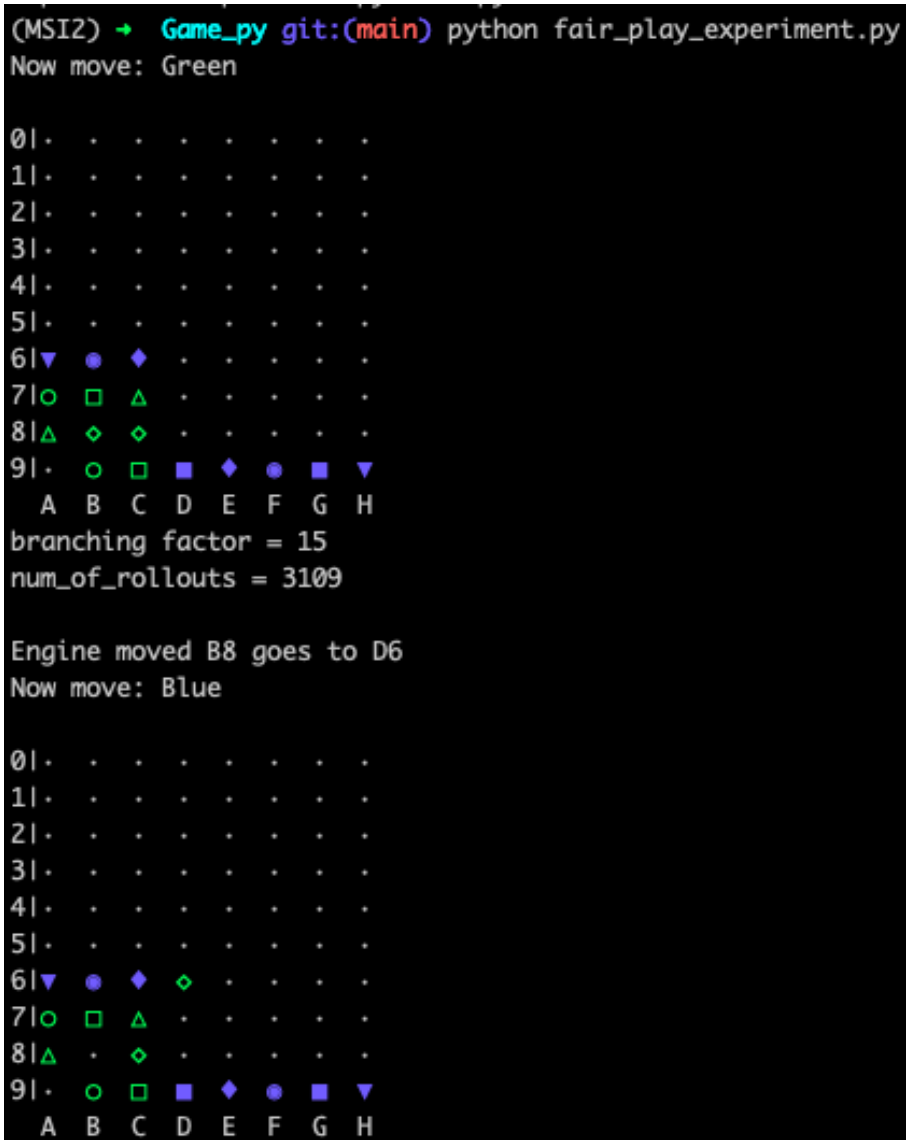
4.3.1. Ekperyment 1

Na obrazku 4.1 pokazano, że losowe ruchy nie są w stanie doprowadzić gry do pozycji terminalnej. Potwierdza to więc wcześniejsze podejrzewania. Algorytm zachowuje się zgodnie z oczekiwaniami. Etap symulacji oryginalnego algorytmu MCTS nie nadaje się do zastosowanie przy grze Taifho. Postawiona hipoteza jest więc prawdziwa.

Rysunek 4.1: Efekt 14000 losowych ruchów od pozycji startowej. Eksperyment powtórzono kilkakrotnie i nigdy gra nawet nie zbliżyła się do pozycji terminalnej.

4.3.2. Ekperyment 2

Na obrazku 4.2 pokazano przykładowy wynik działania algorytmu. Widać, że algorytm MCTS nie próbuje utrzymać blokady. Wręcz przeciwnie, już po jednym ruchu odblokował przeciwnikowi możliwość wygranej. Podobnie zachowywały się pozostałe uruchomienia. Potwierdza to więc wcześniejsze podejrzewania. Algorytm zachowuje się zgodnie z oczekiwaniami i potwierdza założoną hipotezę.



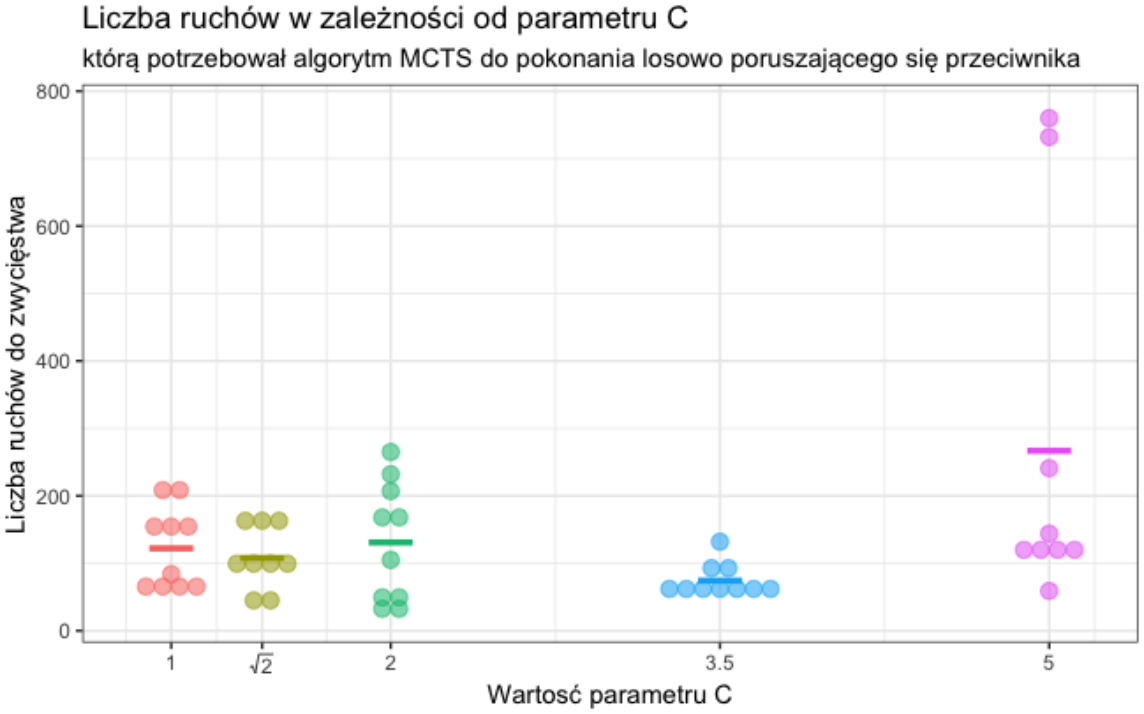
Rysunek 4.2: Efekt działania algorytmu MCTS na pozycji z rysunku 1.6. Algorytm miał czas 1 minutę i podjął decyzję o odblokowaniu przejścia. Umożliwił przeciwnikowi zwycięstwo.

4.3.3. Ekperyment 3 i 4

Eksperymenty 3 i 4 podzielone były na dwie części: grę MCTS z graczem wybierającym ruchy losowo oraz na grę między algorytmami MCST, z ustawionymi parametrami podejrzewanymi

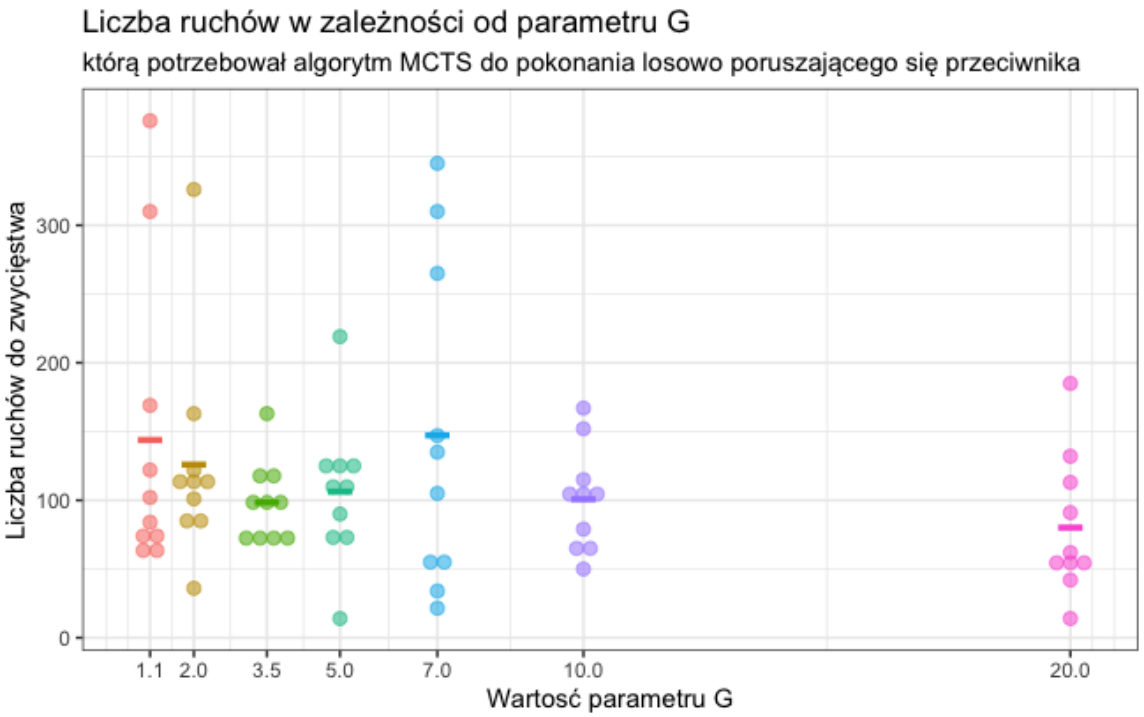
o bycie najlepszymi.

W części pierwszej tych eksperymentów wszystkie gry wygrał algorytm AI. Jakość oceny algorytmu dla danej wartości parametru C dla eksperymentu 3 (i analogicznie G dla eksperymentu 4) oceniano na podstawie liczby ruchów, które zostały wykonane do zakończenia rozgrywki z graczem, który poruszał się losowo. Wyniki części pierwszej przedstawiono dla eksperymentu 3 na Rysunku 4.3 a dla eksperymentu 4 na Rysunku 4.4. Pozioma kreska umieszczona na wykresach skrzypcowych oznacza średnią liczbę ruchów spośród rozgrywek dla różnych ziaren losowości. Wykresy sporządzono na podstawie 10 meczy.



Rysunek 4.3: Wykres skrzypcowy pokazujący rozkład liczby ruchów w grze, jaką potrzebował algorytm MCTS z heurystyką h na pokonanie losowo ruszającego się przeciwnika. Różne wartości parametru C, przy stałych: parametr *steps* = 6, czas na ruch = 1.5 sekundy. Kreską poziomą oznaczono średnią. Wykres powstał na podstawie 10 gier dla każdej z wartości parametru C.

Rysunek 4.3 pokazuje, że najlepszą średnią wyników miała wartość $C = 3.5$, a wartość $C = \sqrt{2}$, sugerowana powszechnie w literaturze, była na drugim miejscu. Kolejne wyniki są bardziej „rozjechane" w liczbie ruchów w zależności od tury oraz ich średnia jest już większa. W przypadku szukania najlepszej wartości parametru G, najlepszą średnią wyników miała wartość $G = 20$. Drugie miejsce zajmuje $G = 3.5$ (Rysunek 4.4). W przypadku tego eksperymentu liczba ruchów była bardziej różnorodna dla pojedynczej wartości parametru G niż w przypadku eksperymentu 3.



Rysunek 4.4: Wykres skrzypcowy pokazujący rozkład liczby ruchów w grze, jaką potrzebował algorytm MCTS z heurystyką h_G na pokonanie losowo ruszającego się przeciwnika. Różne wartości parametru G, przy stałych: parametr $C = 3.5$, parametr $steps = 6$, czas na ruch $= 2$ sekundy. Kreską poziomą oznaczono średnią. Wykres powstał na podstawie 10 gier dla każdej z wartości parametru G.

Nie widać na tych wykresach jednak żadnej wyraźnej zależności. Aby wyciągać na ich podstawie dalej idące wnioski, przydałoby się raczej wpierw zdobyć więcej danych.

W drugiej części porównano algorytmy MCTS między sobą – dla parametru C wartości 3.5 oraz $\sqrt{2}$, a dla parametru G wartości 20 i 3.5. W Tabelach 4.1 i 4.2 przedstawiono wyniki dla 10 rozgrywek między tymi algorytmami. Przewagę wygranych gier $6:4$ miała wartość $\sqrt{2}$ dla parametru C i taką samą przewagę wygranych miała wartość 20 dla parametru G. Uznano więc je za lepsze. Warto jednak zauważyć, że przewaga wygranych względem przegranych nie jest znacząca. Dodatkowo zauważono, że algorytm MCTS zbliżając się do pozycji terminalnej nie potrafi poruszać bierkami tak optymalnie jak we wcześniejszych częściach rozgrywki. Jego ruchy przypominają raczej błądzenie zamiast drogi ku zwycięstwu. Stąd MCTS umożliwia przeciwnikowi zbliżenie się do swojej pozycji terminalnej i wówczas to kto wygra mecz jest dość losowe. Z powodu tego problemu algorytmu AI nie można jednoznacznie stwierdzić, że znalezione parametry są najlepsze.

Dlatego potwierdzenie bądź zaprzeczenie postawionych hipotez badawczych jest niemożliwe.

Można bowiem zauważyć, że wygrywające parametry zaprzeczają hipotezom, natomiast te, z którymi walczyły w grze potwierdzałyby je. Jakby natomiast patrzeć tylko na wyniki pierwszej części eksperymentów to hipoteza 3 byłaby potwierdzona a 4 obalona.

Wartość parametru	Liczba rozpoczętych rozgrywek	Liczba wygranych gier
$C = 3.5$	3	4
$C = \sqrt{2}$	7	**6**

Tablica 4.1: Liczba rozpoczętych oraz wygranych rozgrywek dla gry między algorytmami MCTS z UCT z heurytyką h i różnymi wartościami parametru C, przy stałych: parametr $steps = 6$, czas na ruch = 1.5 sekundy.

Wartość parametru	Liczba rozpoczętych rozgrywek	Liczba wygranych gier
$G = 20$	3	**6**
$G = 3.5$	7	4

Tablica 4.2: Liczba rozpoczętych oraz wygranych rozgrywek dla gry między algorytmami MCTS z UCT z heurytyką h_G i różnymi wartościami parametru G, przy stałych: parametr $steps = 6$, czas na ruch = 2 sekundy.

4.3.4. Ekperyment 5

Turniej będący eksperymentem 5 dotyczył trzech meczy pomiędzy:

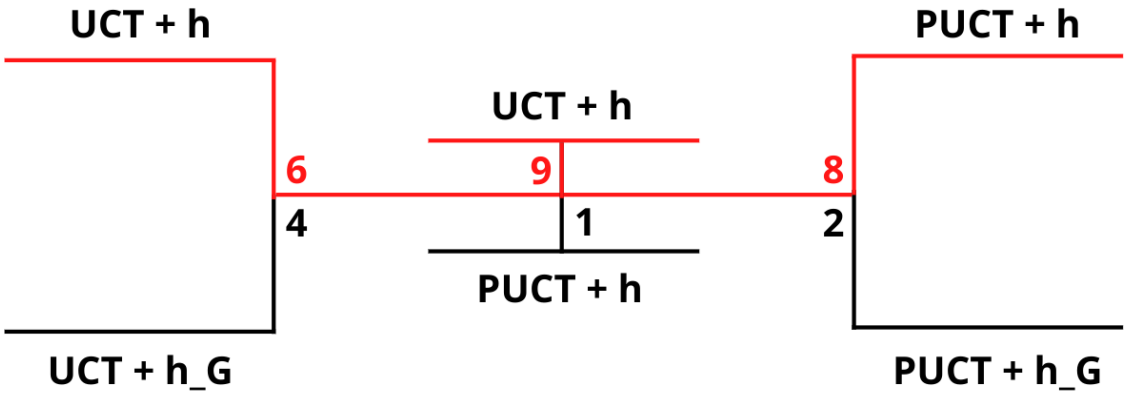
1. algorytmem MCTS z UCT z heurystyką h przy stałych parametrach: $C = \sqrt{2}$, $steps = 6$ i czasie na wykonanie ruchu równym 2 sekundy i algorytmem MCTS z UCT z heurystyką h_G przy stałych parametrach: $C = 3.5$, $G = 20$, $steps = 6$ i czasie na wykonanie ruchu równym 2 sekundy,

2. algorytmem MCTS z PUCT z heurystyką h przy stałych parametrach: $C = \sqrt{2}$, $steps = 6$ i czasie na wykonanie ruchu równym 2 sekundy i algorytmem MCTS z PUCT z heurystyką h_G przy stałych parametrach: $C = 3.5$, $G = 20$, $steps = 6$ i czasie na wykonanie ruchu równym 2 sekundy,

3. zwycięzcami z pierwszego i drugiego meczu.

Wyniki rozgrywek turnieju przedstawiono na Rysunku 4.5. Okazało się, że algorytmy z heurystyką h wygrały w turnieju z tymi z heurystyką h_G: w przypadku UCT stosunek wygranych do

przegranych wyniósł 6 : 4, a przy PUCT 8 : 2. Finał odbył się na algorytmach, których różniło jedynie zastosowanie PUCT zamiast UCT. Turniej zakończył się wygraną 9/10 meczy przez algorytm MCTS z UCT.



Rysunek 4.5: Przebieg turnieju, który miał na celu porównanie wybranych wersji algorytmów. Czerwonym kolorem zaznaczono wygrane algorytmy i ich liczbę wygranych meczy z 10 przeprowadzonych rozgrywek.

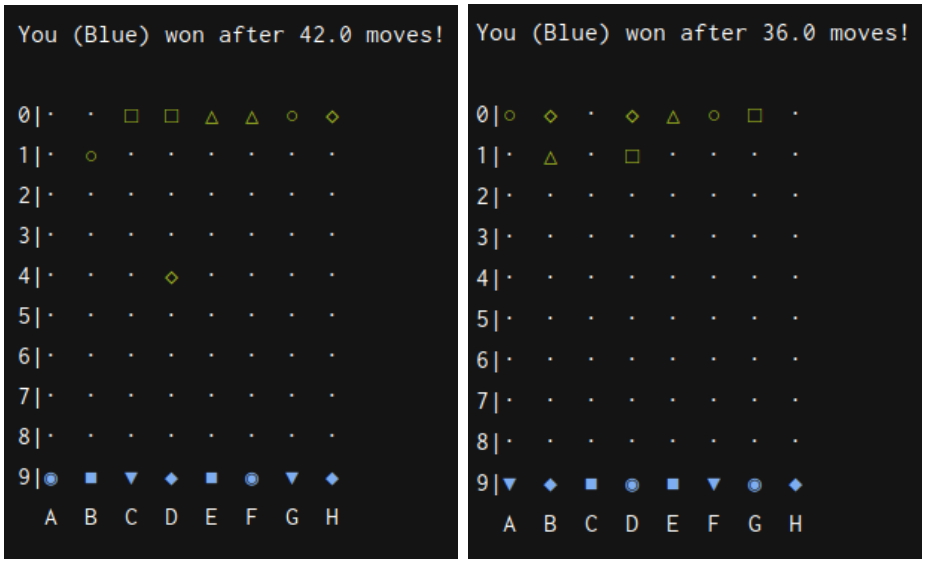
Postawiona hipoteza zakładała skuteczność algorytmów w kolejności: podstawowy UCT, UCT z heurystyką h, UCT z heurystyką h_G, PUCT. Przeprowadzony turniej zaprzeczył tym założeniom. Z zaimplementowanych algorytmów najlepiej grał UCT z heurystyką h, następnie UCT z heurystyką h_G, a potem odpowiednio PUCT z h i h_G. Natomiast podstawowy UCT nie był możliwy do zastosowania do tej gry co wynika z eksperymentu 1.

4.3.5. Ekperyment 6

Na podstawie eksperymentu 5 zdecydowano, że najlepszym jest algorytm MCTS z UCT i heurystyką h, gdzie parametr $C = \sqrt{2}$. Podczas wszystkich rozegranych gier z algorytmem AI wygrał człowiek. Zmiana parametru $steps = 6$ na wartość $steps = 10$ nie wpłynęła na zmianę zwycięzcy. Stąd postawiona hipoteza okazała się nieprawdziwa.

Zauważono, że mimo przegranej w każdym meczu, algorytm nie grał dużo gorzej od człowieka. Większość rozgrywek była wyrównana do pewnego momentu. Okazywało się, że im bliżej pozycji terminalnej znajdował się algorytm tym jego ruchy nie miały za bardzo sensu i nie posuwały go do wygranej. W ten sposób przeciwnik (człowiek) zyskiwał czas, aby zakończyć grę swoim zwycięstwem. Gdy algorytmowi MCTS zostawała do ułożenia jedna bierka, to wykazywał duże problemy z ostatecznym ustawieniem jej na swoim miejscu. Na Rysunku 4.6 przedstawiono dwa przykłady planszy po zakończeniu rozgrywki, gdzie wygrał człowiek amator.

W ostatniej rozgrywce, gdzie MCTS miał aż 60 sekund na każdy ruch, również zwyciężył człowiek. Pod koniec wyrównanej rozgrywki algorytmowi brakowało 2 ruchów do zwycięstwa. Słusznym jest więc podejrzewać, że gdyby dać algorytmowi odrobinę większy czas na obliczenia, to być może udałoby mu się pokonać przeciwnika, człowieka, amatora.



Rysunek 4.6: Dwie pozycje terminalne, zwycięskie dla gracza niebieskiego, którym kierował człowiek z małym doświadczeniem w grze Taifho. Można zauważyć, że gracz zielony będący wybranym algorytmem AI, był bliski wygranej.

5. Podsumowanie i wnioski

Projekt zakończył się sukcesem. Wszystkie cele techniczne zostały zrealizowane. Algorytmy AI potrafią w miarę inteligentnie poruszać się po planszy gry Taifho.

Efektywność i sposób działania zaimplementowanych algorytmów były dla autorów tego projektu wielce zaskakujące. Udało się potwierdzić jedynie dwie spośród sześciu postawionych przed projektem hipotez badawczych.

Zdaniem graczy, ludzi grających w eksperymencie 6, algorytmy MCTS radziły sobie w początkowej i w środkowej fazie gry. Potrafiły one zarówno znacznie przesuwać swoje bierki do przodu, wykorzystując zasadę wielokrotnego skoku, jak i również blokować swojego przeciwnika przed tym samym. Okazało się jednak, że algorytmy MCTS w formie zaimplementowanej w tym projekcie nie potrafią poradzić sobie w końcowej fazie rozgrywki. Jest taki moment w grze Taifho, gdy gracze "schodzą sobie z drogi", gdyż obaj są wszystkimi swoimi bierkami blisko linii końcowej. W tym końcowym etapie gry dla zawodnika nie ma znaczenia, jakie ruchy podejmie przeciwnik. Zadaniem gracza w tym etapie jest ustawienie swoich bierek na linii końcowej w jak najkrótszym czasie. Algorytm MCTS zawsze rozważa swój drugi ruch oddzielnie dla każdej z możliwych ruchów przeciwnika. Chce się dostosować do ruchów swojego oponenta. Ogólnie to jest pożądane zachowanie, jednakże na tym etapie rozgrywki niepotrzebne jest rozważanie ruchów przeciwnika. Sugeruje to więc użycie dla końcówek gry innego algorytmu, który nie zważa na bierki przeciwnika. Pozwoliłoby to na lepsze przeszukanie drzewa końcowej fazy gry.

Przez to, że algorytm MCTS słabo radzi sobie w końcówkach, trudno jest ocenić jego jakość w bezpośredniej rozgrywce między algorytmami. Zdarzało się, że jeden z algorytmów AI dochodził do etapu końcowego, który człowiek potrafiłby rozwiązać np. w 8 ruchów, jednak algorytm błąkał się przez kolejne 20. To powodowało, że jego adwersarz również dochodził do podobnego etapu u siebie. W tym momencie obaj błąkali się i zwycięstwo jednego trudno jest określić jako objaw lepszości jednego algorytmu nad drugim. Z tego względu być może rozsądniejszym byłoby ocenianie oddzielnie trzech etapów gry zamiast gry jako całości.

Ze względu na te trudności ciężko jest jednoznacznie ocenić, czy znalezione wartości parametrów C oraz G są rzeczywiście najlepiej grające. W eksperymentach 3, 4 oraz 5 bardzo

dużo zależało od ziarna losowości, a co za tym idzie od "szczęścia" jednego, czy drugiego z zawodników. Widać to na wykresach 4.3 oraz 4.4, gdzie rozrzut wyników jest bardzo duży. Podobnie mecze MCTS kontra MCTS w eksperymentach 3 i 4 nie pozwalały z czystym sumieniem wskazać lepszego algorytmu. Wynik turnieju 6 do 4 sugeruje raczej niewielką przewagę jednego algorytmu nad drugim.

Wynik turnieju w rundzie finałowej był jednoznaczny. Algorytm UCT spośród 10 meczów zwyciężył 9 razy nad algorytmem bliźniakiem z heurystyką PUCT. Zdaniem autorów najciekawsze spośród obserwacji jest, że heurystyka PUCT zastosowana zamiast UCT pogarsza działanie algorytmu. PUCT jest powszechnie chwaloną w literaturze metodą poprawiającą działanie. Być może zastosowana w tym raporcie do PUCT heurystyka była zbyt słaba, aby sensownie pomóc w przeszukiwaniu przestrzeni gry.

Bibliografia

Auer, P. i in. (1995). „Gambling in a rigged casino: The adversarial multi-armed bandit problem". W: *Proceedings of IEEE 36th Annual Foundations of Computer Science*, s. 322–331. DOI: 10.1109/SFCS.1995.492488.

Auer, Peter, Nicolò Cesa-Bianchi i Paul Fischer (maj 2002). „Finite-time Analysis of the Multiarmed Bandit Problem". W: *Machine Learning* 47.2, s. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352. URL: https://doi.org/10.1023/A:1013689704352.

Browne, Cameron B. i in. (2012). „A Survey of Monte Carlo Tree Search Methods". W: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1, s. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.

Kocsis, Levente i Csaba Szepesvári (2006). „Bandit Based Monte-Carlo Planning". W: *Machine Learning: ECML 2006*. Red. Johannes Fürnkranz, Tobias Scheffer i Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 282–293. ISBN: 978-3-540-46056-5.

Mullen, Jethro (sty. 2016). *Computer scores big win against humans in ancient game of Go*. https://money.cnn.com/2016/01/28/technology/google-computer-program-beats-human-at-go/index.html.

Przemysław, Chojecki (b.d.). *GitHub page with the code for this project*. https://github.com/PrzeChoj/MSI2/. [Online; accessed 03-05-2022].

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Rosin, Christopher D. (mar. 2011). „Multi-armed bandits with episode context". W: *Annals of Mathematics and Artificial Intelligence* 61.3, s. 203–230. ISSN: 1573-7470. DOI: 10.1007/s10472-011-9258-6. URL: https://doi.org/10.1007/s10472-011-9258-6.

scrabblemania.pl, oficjalne zasady gry (b.d.). https://scrabblemania.pl/oficjalne-zasady-gry-w-scrabble. [Online; accessed 03-05-2022].

Silver, David i in. (sty. 2016). „Mastering the game of Go with deep neural networks and tree search". W: *Nature* 529.7587, s. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

Strona Desmos na której można zobaczyć jaki wpływ ma parametr G na heurystykę h_G (b.d.). https://www.desmos.com/calculator/uhbknmltyg. [Online; accessed 03-05-2022].

Strona internetowa z opisem zasad gry w Taifho z wydania planszowego z 1999 roku (b.d.). https://bastardcafe.dk/games/taifho/. [Online; accessed 03-05-2022].

Wang, Yajie i in. (2018). „Application and Improvement of UCT in Computer Checkers". W: *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, s. 274–278. DOI: 10.1109/CCIS.2018.8691199.

Wykaz skrótów

AI	Artificial Inteligence
MCTS	Monte-Carlo Tree Search
UCB	Upper Confidence Bound
UCT	Upper Confidence Bound Applied To Trees
PUCB	Predictor UCB
PUCT	Predictor UCT