Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **INA**
Specjalność: **-**

Praca Dyplomowa Inżynierska

Algorytm Minimax do gry w Warcaby

Janusz Witkowski

Opiekun pracy
dr Maciej Gębala

Sztuczna inteligencja, Algorytmika, Algorytmy metaheurystyczne, Teoria gier

Wrocław 2022

Streszczenie

Tutaj tekst streszczenia po polsku.

Abstract

Tutaj treść streszczenia po angielsku.

Spis treści

Spis rysunków

Spis tabel

Wstęp

We wstępie zostanie zawarte tak zwane “lanie wody”. Najprawdopodobniej będzie to część dokumentu która swój ostateczny obraz obierze pod sam koniec pisania. Aby dobrze wprowadzić Czytelników w temat pracy, autor musi mieć przed oczyma całość swojego dzieła.

Pierwsza wersja tejże pisemnej pracy dyplomowej posiada wyszczególnione rozdziały i podrozdziały, wraz ze szczątkowym opisem ich przyszłej zawartości.

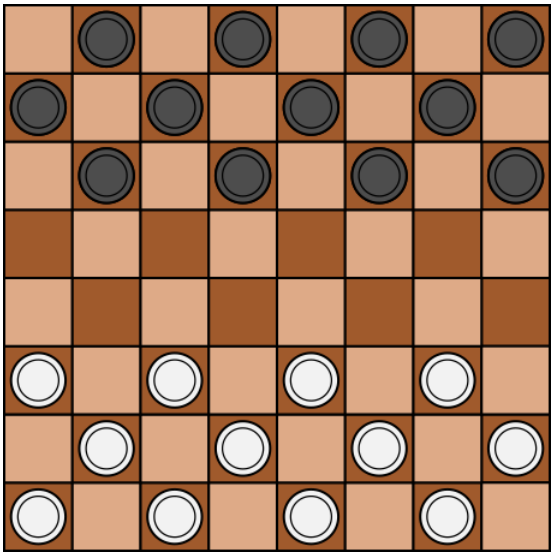
Rozdział 1

Warcaby

Warcaby to jedna z najpopularniejszych klasycznych gier dwuosobowych, zaliczanych do gier z doskonałą informacją i gier o sumie zerowej. Przyjmuje się, że gra ta zrodziła się w XII wieku, najprawdopodobniej na południu Francji lub w Hiszpanii, oraz że wywodzi się ona z dawnej arabskiej gry Alquerque [5]. Istnieją coroczne turnieje i mistrzostwa światowe w różnych odmianach Warcabów, choć dzięki nieskomplikowanym zasadom są one popularne również w mniejszych kręgach.

1.1 Reguły warcabów standardowych

Pojedynczą partię Warcabów rozgrywa się na szachownicy 8x8 o polach na zmianę pomalowanych na jasno lub ciemno. W grze wykorzystywane są dwa rodzaje figur - piony i damki. Obydwaj gracze rozstawiają po 12 pionów na ciemnych polach w swoich pierwszych trzech rzędach. Dla rozróżnienia, piony pierwszego gracza są koloru białego, natomiast piony drugiego gracza - czarnego. Celem gry jest wyeliminowanie wszystkich figur przeciwnika lub zablokowanie go, poprzez serię naprzemiennych ruchów swoimi figurami. (Zablokowanie gracza oznacza doprowadzenie do takiej sytuacji, w której gracz ten nie jest w stanie wykonać żadnego legalnego ruchu w momencie gdy następuje jego kolej.)



Rysunek 1.1: Stan początkowy planszy w Warcabach.

Wszystkie figury w grze mogą poruszać się tylko i wyłącznie na ukos (przez co żadne jasne pole na planszy nie zostanie zajęte przez żadną figurę w trakcie rozgrywki). Piony z którymi zaczynają gracze poruszają się tylko o jedno pole w przód względem ich właściciela. Tzn. pion może skoczyć na pole ukośnie

sąsiadujące w kierunku oponenta, o ile pole to nie jest zajęte przez inną figurę. W grze istnieje również druga figura - jeżeli pion gracza dojdzie do końca planszy znajdującego się po stronie jego oponenta (do tzw. rzędu awansu), pion zamieniany jest na damkę. Damka jest najpotężniejszą figurą w grze, jako że potrafi poruszyć się we wszystkich czterech kierunkach na ukos, a na dodatek przebyć dowolną liczbę pól w linii w jednym ruchu. Pod tym względem damkę najłatwiej porównać z figurą gońca w szachach.

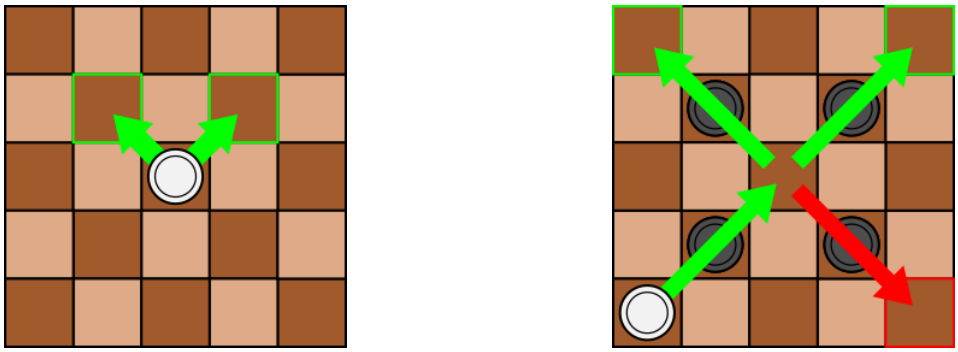
Każdą figurę w Warcabach można zbijać, tj. usuwać z obecnej rozgrywki. Piony mogą zbijać sąsiednie figury przeciwnika, wykonując skok nad tą figurą na następne pole w linii prostej, o ile takie pole jest wolne. Piony mogą bić zarówno do przodu, jak i do tyłu. Damki w standardowych Warcabach zbijają ze znacznie większego dystansu (można powiedzieć, że w momencie zbijania ich "sąsiadowanie"z przeciwnymi figurami nie jest ograniczone do jednego pola różnicy). Zbita figura zostaje zdjęta z planszy i nie bierze udziału w rozgrywce do momentu jej zakończenia. Nie można bić swoich figur.

Bicia w Warcabach mają specjalne reguły wyróżniające je spośród innych gier planszowych. Po pierwsze, w jednym ruchu jedna figura może wykonać wiele bić. Jeśli po jednym biciu figura wskoczyła w miejsce z którego jest w stanie przeprowadzić kolejne bicie, można takie bicie wykonać w tej samej turze. W jednym ruchu nie można dwa razy zbić tej samej figury. Po drugie, bicia są obowiązkowe. Jeżeli gracz w swojej turze jest postawiony w sytuacji w której co najmniej jedna z jego figur ma możliwość bicia, gracz ten musi wykonać taki ruch. Dodatkowo, o ile ruch bicia pionem można wybrać w przypadku większej liczby możliwych bić, damki mają obowiązek maksymalnego bicia, tj. należy przeprowadzić bicie o największej możliwej liczbie zbijanych figur w jednym ruchu.

1.2 Wariant angielski

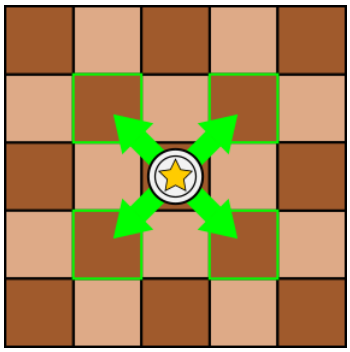
Praca skupiona jest na szczególnej wersji gry w Warcaby, nazywanej na ogół wariantem angielskim, lub w niektórych kręgach wariantem amerykańskim. Został on wybrany głównie ze względu na ograniczenie przestrzeni stanów w jakich może znaleźć się rozgrywka - reguły gry dostosowane do tego wariantu znacznie zmniejszają liczbę możliwości które rozgrywający algorytm musi rozpatrzeć.

Wariant ten wprowadza dwie zmiany do zasad gry względem wariantu standardowego (opierając się o [5]). Po pierwsze, zwykłe piony nie mogą bić do tyłu. Po drugie, damkom ogranicza się możliwość ruchu o dowolną liczbę pól do jednego sąsiedniego pola oraz do bicia wyłącznie sąsiadujących przeciwnych figur, lecz wciąż mogą poruszać się we wszystkich kierunkach na ukos. Jedyną przewagą damek nad pionkami w tym wariancie jest możliwość ruchu i bicia do tyłu.

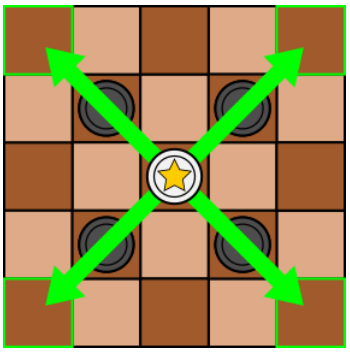


(a) Legalne ruchy (b) Legalne bicia

Rysunek 1.2: Zestaw możliwych ruchów dla piona w wariancie angielskim



(a) Legalne ruchy



(b) Legalne bicia

Rysunek 1.3: Zestaw możliwych ruchów dla damki w wariancie angielskim

Przydatna w implementacji rzecz o której warto wspomnieć jest rozpatrywanie remisów. W grach towarzyskich remis zazwyczaj następuje za obopólną zgodą graczy. Na arenie turniejowej istnieje parę kryteriów determinujących remis. W rozpatrywanej wersji wariantu angielskiego wykorzystywana będzie zasada 40 ruchów, która mówi że rozgrywka kończy się remisem w momencie gdy w 40 naprzemiennych ruchach obu graczy nie została zbita ani jedna figura.

1.2.1 Badania wariantu

Do rozwoju badań nad Warcabami w wariancie angielskim najmocniej przyczynił się Jonathan Schaeffer, profesor Uniwersytetu Alberty w Kanadzie. Jego zespół opracował Chinook'a, przeszukujący w głąb program do grania w Warcaby angielskie, który w roku 1992 oraz 1994 stanął naprzeciw ówczesnego mistrza świata Marion'a Tinsley'a i ogłoszony został pierwszym komputerowym zwycięzcą mistrzostw [1]. Następnie, z pomocą programu, udowodnili poprzez słabe rozwiązanie (*weakly solved*, pojęcie omówione w [7]) że każda gra w Warcabach angielskich kończy się remisem, pod warunkiem że gracze wykonują ruchy doskonale [6]. Pomimo uproszczonych zasad względem klasycznej wersji, wariant angielski posiada przestrzeń stanów wielkości rzędu 10^{20}, dlatego też rozwiązanie zajęło zespołowi Schaeffer'a około 18 lat i 200 równolegle liczących maszyn.

Rozdział 2

Idea rozwiązania i algorytmy

Głównym celem pracy jest stworzenie prostego modelu sztucznej inteligencji do grania w Warcaby w wariancie angielskim z pewną strategią. Istnieją różne podejścia do tego zagadnienia, spośród których najpopularniejszymi są te stosujące sieci neuronowe (zestawem danych byłaby np. baza meczów rozegranych na mistrzostwach na przestrzeni kilkunastu lat). Praca skupia się na bardziej nadzorowanej metodzie. Zastosowany został algorytm Minimax połączony z funkcją oceny heurystycznej. Do częściowego wyznaczenia funkcji oceny wykorzystano odmianę algorytmu genetycznego.

2.1 Minimax

Minimax jest szczególną wersją algorytmu przeszukującego w grafie. Jego idea jest bardzo prosta i zbliżona do ludzkiego rozumowania. Mając dany stan planszy oraz głębokość przeszukiwania, algorytm rekurencyjnie rozpatruje kolejne stany planszy symulując wykonanie jednego możliwego ruchu. Kiedy już osiągnie maksymalną głębokość przeszukiwań, funkcją oceny heurystycznej przypisuje wartości stanom, po czym zwraca tę wartość do swojego stanu-rodzica. Mając wartości oceny od każdego swojego dziecka, stan wybiera jedną z nich i przekazuje ją do swojego rodzica. Gdy wybór dojdzie do stanu będącego korzeniem drzewa przeszukiwań, algorytm wybierze jedną z dostępnych mu ocen i zwróci ruch do stanu któremu ta ocena odpowiada.

Pseudokod 2.1: Prosty algorytm Minimax

```
   Input: Stan gry state, flaga gracza maximizing, głębokość przeszukiwań h, rozpatrujący gracz player
   Output: Wartość funkcji oceny eval
 1 if h = 0 then
 2 |   eval ← evaluateState(state, player);
 3 else
 4 |   if maximizing = True then
 5 |   |   maxEval ← −∞;
 6 |   |   foreach child ∈ getChildren(state) do
 7 |   |   |   childEval ← minimax(child, False, h - 1, player);
 8 |   |   |   if childEval ⩾ maxEval then
 9 |   |   |   |   maxEval ← childEval
10 |   |   eval ← maxEval
11 |   else
12 |   |   minEval ← +∞;
13 |   |   foreach child ∈ getChildren(state) do
14 |   |   |   childEval ← minimax(child, True, h - 1, player);
15 |   |   |   if childEval ⩽ minEval then
16 |   |   |   |   minEval ← childEval
17 |   |   eval ← minEval
```

Swoją nazwę algorytm zawdzięcza naprzemiennemu rozpatrywaniu ocen heurystycznych w drzewie

przeszukiwań. W momencie gdy dany gracz wywołuje procedurę Minimaxa rozpatrując możliwe do wykonania przez niego ruchy, oznaczany jest jako gracz MAX. W powstałych stanach gry gracz symuluje tok rozumowania jego przeciwnika, rozpatrując ruchy za niego i oznaczając go jako gracza MIN. Stany na kolejnym poziomie w drzewie przeszukiwań rozpatruje gracz MAX, i tak dalej. Gdy gracz MAX dokonuje wyboru oceny do przekazania do stanu-rodzica, wybiera maksymalną wartość. Analogicznie, gracz MIN wybiera ocenę o minimalnej wartości.

. . . PRZYKŁAD DZIAŁANIA MINIMAXA . . .

2.1.1 Alpha-Beta-prunning

. . . Cięcia alfa-beta to rezygnacja z rozpatrywania innych podgałęzi ze względu na brak możliwości poprawienia wyniku. . . .

. . . PRZYKŁAD ALFA-BETA-CIĘĆ . . .

. . .

2.2 Funkcja oceny heurystycznej

Ze względu na ogromny rozmiar przestrzeni stanów w Warcabach, obecne komputery nie potrafią przeszukać jej całości w „rozsądnym” czasie. Jeśli jednym z celów budowania sztucznej inteligencji są w miarę szybkie decyzje prowadzące do zwycięstwa w grze, należy ukrócić przeszukiwanie przestrzeni stanów i w jakiś sposób obejść się z niedoskonałą informacją posiadaną przez komputer. Ocena danego stanu na planszy ma wspomóc taki komputer w wyrobieniu „intuicji” poprzez analizę sytuacji na planszy.

Podejście w pracy do funkcji oceny heurystycznej polega na rozpatrzeniu wielu parametrów na planszy (np. liczba pionów, liczba damek przeciwnika, liczba ruchów), przemnożeniu wartości tych parametrów przez ustalone z góry wagi, a na koniec zsumowaniem powstałych iloczynów. Suma tych iloczynów to wartość funkcji oceny, którą przypisuje się pod dany stan gry.

Nr	Parametr	Nr	Parametr
1	Liczba sojuszniczych pionów	31	Liczba przeciwnych damek w środkowych wierszach
2	Liczba sojuszniczych damek	32	.
3	Liczba przeciwnych pionów	33	.
4	Liczba przeciwnych damek	34	.
5	Liczba sojuszniczych pionów przy ścianie	35	.
6	Liczba sojuszniczych damek przy ścianie	36	.
7	Liczba przeciwnych pionów przy ścianie	37	.
8	Liczba przeciwnych damek przy ścianie	38	.
9	Liczba ruchomych pionów gracza	39	.
10	Liczba ruchomych damek gracza	40	.
11	Liczba ruchomych pionów przeciwnika	41	.
12	Liczba ruchomych damek przeciwnika	42	.
13	Liczba możliwych ruchów gracza	43	.
14	Liczba możliwych ruchów przeciwnika	44	.
15	Istnienie bijącego ruchu gracza	45	.
16	Liczba bijących ruchów gracza	46	.
17	Rozmiar najdłuższego bijącego ruchu gracza	47	.
18	Istnienie bijącego ruchu przeciwnika	48	.
19	Liczba bijących ruchów przeciwnika	49	.
20	Rozmiar najdłuższego bijącego ruchu przeciwnika	50	.
21	Suma dystansów pionów gracza do rzędu awansu	51	.
22	Suma dystansów pionów przeciwnika do rzędu awansu	52	.
23	Liczba niezajętych pól w rzędzie awansu gracza	53	.
24	Liczba niezajętych pól w rzędzie awansu przeciwnika	54	.
25	.	55	.
26	.	56	.
27	.	57	.
28	.	58	.
29	.	59	.
30	.	60	.

Tabela 2.1: Wszystkie parametry rozpatrywane w pracy. Część parametrów została zaczerpnięta z [4].

Wartość funkcji oceny heurystycznej można określić następującym wzorem: $\sum_{i=1}^{n}(param_i * weight_i)$

Funkcja oceny heurystycznej zależy oczywiście od podjętej strategii oraz od podejścia do problemu. Opisana wyżej funkcja ma parę zalet na których opiera się praca. Po pierwsze, samo wyznaczenie oceny z gotowymi wartościami parametrów odbywa się szybko, bo w czasie liniowym (nie uwzględnia to czasu potrzebnego do rozpatrzenia tychże parametrów). Po drugie, listowanie w ten sposób parametrów pozwoli na przeprowadzenie eksperymentów i wyciągnięcie wniosków o teorii gry w Warcaby, o czym w poniższym podrozdziale.

2.3 Algorytm genetyczny

Jednym z najważniejszych celów pracy jest znalezienie odpowiedniej strategii gry w Warcaby w postaci dobrej funkcji oceny heurystycznej, co w tym przypadku sprowadza się do wyznaczenia jak najlepszego zestawu wartości wag (dalej zwanego ciągiem wag). Naiwnym podejściem do tego problemu byłoby przypisanie priorytetów każdego parametru stanu planszy przez człowieka bądź zespół ludzi. Może się jednak okazać, że pewne parametry nie będą tak wartościowe dla zwycięstwa jak podpowiedziałaby ludzka intuicja. Ponadto, zakładając że nie istnieje obiektywne spojrzenie na wartość strategii grającej, należałoby przeprowadzić ogromną liczbę testów gry z człowiekiem w celu sprawdzenia poprawności wyznaczonych wag (tj. czy podane wartości ciągu wag są wystarczające by algorytm uznać za kompetytywnego gracza). Z pomocą tutaj przychodzi zastosowanie algorytmu genetycznego.

Algorytm genetyczny jest przedstawicielem klasy algorytmów metaheurystycznych. Konkretniej jest to odmiana algorytmu ewolucyjnego, który z kolei jest pochodną algorytmu populacyjnego. Metaheurystyka genetyczna symuluje zjawisko doboru naturalnego w przyrodzie, operując kolejnymi pokoleniami populacji osobników reprezentującymi potencjalne rozwiązania danego problemu, starając się odnaleźć rozwiązanie suboptymalne.

Najprostszy zarys algorytmu genetycznego można zobrazować w następujący sposób. W początkowej populacji losowo utworzonych osobników (rozwiązań) dochodzi do selekcji, mającej na celu wyznaczenie populacji rodziców. W populacji rodziców dochodzi do wzajemnego krzyżowania i tworzenia populacji dzieci, które dziedziczą po swoich potomkach informacje genetyczne (tj. elementy rozwiązania). Wśród dzieci może z pewnym prawdopodobieństwem dojść do mutacji, która losowo zmienia jeden z elementów genotypu u osobnika. W świeżo powstałej populacji powtarza się proces selekcji, krzyżowania i mutacji, aż do osiągnięcia z góry ustalonego warunku stopu (np. limit czasowy).

. . . PRZYKŁAD PRZEBIEGU ALGORYTMU GENETYCZNEGO . . .

Biorąc pod uwagę fakt, że to od rodziców zależy jakość każdej populacji, szczególny nacisk należy położyć na fazę selekcji. Niezbyt intuicyjną taktyką jest wprowadzanie w pewne miejsca przebiegu algorytmu elementów losowości, aby w populacji nie dochodziło do tak zwanego zjawiska stagnacji (sytuacja w której zbyt wiele osobników w populacji jest do siebie bardzo podobnych) oraz aby algorytm był w stanie znajdować inne, być może lepsze rozwiązania.

2.3.1 Populacja i osobniki

Osobnikiem populacji będzie ciąg wag funkcji oceny heurystycznej, reprezentowany jako tablica liczb całkowitych. Wagi będą mogły przyjmować zarówno dodatnie, jak i ujemne wartości.

. . . RYSUNEK OBRAZUJĄCY OSOBNIKI W POPULACJI . . .

2.3.2 Selekcja i ewaluacja

W wielu problemach do których stosuje się algorytmy genetyczne, funkcja ewaluacji osobnika jest podana w treści problemu bądź łatwa do wyznaczenia. Tak jednak nie jest z problemem znalezienia najlepiej grającej sztucznej inteligencji. Dlatego też zastosowane zostało podejście lokalne, czyli zwracające uwagę na umiejętności osobników w danej populacji. W procesie selekcji każdy ciąg wag gra z każdym innym ciągiem wag w podwójnym pojedynku (białe/czarne i czarne/białe). Im więcej gier wygra ciąg wag, tym wyższe prawdopodobieństwo że zostanie on wylosowany do populacji rodziców.

2.3.3 Krzyżowanie i mutacja

W pracy zaimplementowano dobieranie rodziców w pary w których się wzajemnie krzyżują, produkując dwójkę dzieci. Aby zachować potencjalnie dobre informacje genetyczne, dzieli się ciągi wag rodziców w tym samym losowo wyznaczonym miejscu i zamienia się ze sobą części. Dzięki temu każde z dwójki dzieci posiada cechy po każdym swoim rodzicu.

... RYSUNEK KRZYŻOWANIA ...

Mutacja u nowo narodzonych osobników następuje z pewnym prawdopodobieństwem i wpływa na maksymalnie jedną wartość w ciągu wag. Mutowana wartość zostaje zastąpiona losową wartością z odpowiedniego przedziału.

Rozdział 3

Implementacja

Kody źródłowe dołączone do niniejszej pracy (patrz Dodatek A) są wynikiem implementacji algorytmów opisanych w rozdziale 2. Przed ich omówieniem warto wspomnieć o dwóch niuansach które miał wpływ na pisanie tej części pracy.

Po pierwsze, priorytetem w implementacji nie była szczegółowa optymalizacja. W założeniu czas przydzielony na przeprowadzenie sesji algorytmu genetycznego wynosił kilka miesięcy, natomiast ograniczenia maszynowe były zaniedbywane. Z tego powodu praca nie skupia się na wielu optymalizacjach, a projekt nie został napisany w języku bardzo wysokiej wydajności (jak C/C++, Rust).

Po drugie, interfejs użytkownika ograniczony jest do niezbędnego minimum. Interakcja z programami odbywa się z poziomu konsoli, nie ma też wielu elementów graficznych w projekcie. Mimo to, system posiada wszystkie funkcjonalności potrzebne do przeprowadzania gier i eksperymentów.

3.1 Język i środowisko

Projekt został napisany w języku Java. Jest to wygodne narzędzie pozwalające na naturalną strukturyzację projektu. Dzięki wirtualnej maszynie Javy (JVM) powstałe programy można uruchamiać na wszystkich wspieranych systemach operacyjnych, co ułatwia przeprowadzanie testów na większą skalę. Jest to też język dobrze zoptymalizowany - nie jest koniecznym przejmowanie się wydajnością „rozsądnie" napisanego kodu.

Struktura projektu tworzona była w duchu programowania obiektowego. Posiada ona klasy z podzielonymi funkcjonalnościami i odpowiedzialnościami. Zadbano również o przechwytywanie podstawowych wyjątków i błędów.

Projekt pisany był w środowisku Javy *OpenJDK 17.0.4 2022-07-19*, aczkolwiek wszystkie wykorzystywane funkcjonalności zawarte są w *OpenJDK 14*.

3.2 Struktura projektu

. . . TBA . . .

3.3 Instrukcja obsługi programów

Jak już wspominano we wcześniejszym podrozdziale 3.2, projekt obejmuje dwa dostępne dla użytkownika programy:

- **Play** - kierownik rozgrywek, prowadzi gry dla graczy zarówno ludzkich jak i komputerowych;
- **Find** - interfejs do uruchamiania sesji algorytmu genetycznego.

Oba programy uruchamia się przekazując argumenty z poziomu konsoli. Sygnatury wywołań każdego z programów można wydrukować na standardowe wyjście, podając *„–help"* jako pierwszy argument.

3.3.1 Play

Dostępne sygnatury wywołania programu:

- Rozgrywka (standardowy model gry w Warcaby, dostępny dla graczy ludzkich i/lub komputerowych; gracz komputerowy musi otrzymać ścieżkę do pliku z którego wczyta ciąg wag do swojej funkcji oceny heurystycznej)
    1. Typ pierwszego gracza [0 → gracz ludzki; liczba naturalna większa od zera → gracz komputerowy z głębokością przeszukiwań równą podanej liczbie]
    2. Ścieżka do pliku z ciągiem wag dla gracza pierwszego [ciąg znaków; argument omijany jeśli pierwszym graczem jest człowiek]
    3. Typ drugiego gracza [0 → gracz ludzki; liczba naturalna większa od zera → gracz komputerowy z głębokością przeszukiwań równą podanej liczbie]
    4. Ścieżka do pliku z ciągiem wag dla gracza drugiego [ciąg znaków; argument omijany jeśli drugim graczem jest człowiek]
- Szybka gra z „głupim" komputerem (gra między graczem ludzkim a komputerowym; ciąg wag dla gracza komputerowego zostanie wylosowany)
    1. Gracz zaczynający [1 → zaczyna człowiek; -1 → zaczyna komputer]
    2. Głębokość przeszukiwań gracza komputerowego [liczba naturalna większa od zera]

Poprawnie wywołany program rozpocznie rozgrywkę w Warcaby angielskie. Będzie na zmianę prosił graczy o wykonanie ruchu, lub, jeśli gracz jest komputerowy, obsłuży jego ruch. Po każdym ruchu drukowany będzie obecny stan planszy. Pokazywane są tez oistatnio wykonane ruchy.

W trakcie wykonywania ruchu gracz ludzki musi podać pole z którego chce wykonać ruch, a następnie pole (pola) na które chce przeskoczyć wybraną figurą. Można się odnosić do pól na planszy na dwa sposoby: albo poprzez współrzędne (kolumna i wiersz), albo poprzez numerację pól.

. . . OBRAZEK Z NUMERACJĄ PÓL NA PLANSZY . . .

W przypadku podania błędnego wejścia, program poprosi gracza o ponowne wprowadzenie ruchu od zera. W momencie gdy rozgrywka miałaby się skończyć, program informuje o zwycięzcy i kończy działanie.

3.3.2 Find

Dostępne sygnatury wywołania programu:

- Pierwsze uruchomienie algorytmu genetycznego (zalecane jeśli użytkownik chce utworzyć nową sesję algorytmu genetycznego od zera)
    1. Liczebność populacji osobników [liczba naturalna podzielna przez 4]
    2. Liczba pojedynków osobnika w procesie selekcji [liczba naturalna, przy czym 0 oznacza pojedynek z każdym innym osobnikiem]
    3. Głębokość przeszukiwania w Minimaxie [liczba naturalna większa od zera]
    4. Współczynnik losowej selekcji osobników [liczba całkowita]
    5. Szansa na mutację [liczba wymierna między 0 a 1]
    6. Rodzaj kryterium stopu [0 → czas (w sekundach); 1 → liczba iteracji]
    7. Limit dla kryterium stopu [liczba naturalna]
- Reaktywacja algorytmu genetycznego z wybranego pliku populacji
    1. Nazwa pliku [ciąg znaków]
- Reaktywacja algorytmu genetycznego z ostatniego pliku populacji [BRAK ARGUMENTÓW]
- Kontynuacja algorytmu genetycznego z nowymi parametrami

1. Nazwa pliku z którego należy wczytać populację [ciąg znaków]
2. Liczba pojedynków osobnika w procesie selekcji [liczba naturalna, przy czym 0 oznacza pojedynek z każdym innym osobnikiem]
3. Głębokość przeszukiwania w Minimaxie [liczba naturalna większa od zera]
4. Współczynnik losowej selekcji osobników [liczba całkowita]
5. Szansa na mutację [liczba wymierna między 0 a 1]
6. Rodzaj kryterium stopu [0 → czas (w sekundach); 1 → liczba iteracji]
7. Limit dla kryterium stopu [liczba naturalna]

Bezbłędne uruchomienie rozpocznie sesję algorytmu genetycznego z podanymi argumentami. W trakcie działania program operuje na plikach w katalogu *heuristics* i jego podkatalogach: *population* (pliki zapisanych populacji, z których można reaktywować sesje; program na bieżąco aktualizuje plik populacji) oraz *output* (najlepiej przystosowany osobnik, tworzony w katalogu po osiągnięciu kryterium stopu). Oprócz tych podkatalogów istnieją również *single* (katalog na samodzielne tworzenie plików ciągów wag) i *archive* (archiwum w którym można bezpiecznie zapisywać pliki ciągów wag), lecz program na tych dwóch podkatalogach nie wykonuje żadnych operacji oprócz ich stworzenia.

Sesja algorytmu genetycznego na bieżąco informuje o postępie - w każdej iteracji drukuje numer porządkowy obecnie ewaluowanej generacji oraz stosunek postępu do limitu we wcześniej podanym kryterium stopu (np. 5/100 generacji lub 100/2000 sekund). W razie niespodziewanego wcześniejszego zakończenia programu, możliwym jest odczyt argumentów sesji i postępu z pliku ostatniej populacji.

Rozdział 4

Wyniki i dalsze pomysły

Można wyróżnić 2 główne cele eksperymentów w pracy:

- **Sprawdzenie parametrów** - uruchomienie sesji algorytmu genetycznego w celu przypisania względnych wartości pod parametry w funkcji oceny heurystycznej;
- **Porównanie głębokości Minimaxa** - przeprowadzenie dwóch sesji algorytmu genetycznego z głębokościami różniącymi się od siebie o 1, a następnie porównanie wynikowych ciągów wag.

Do przeprowadzenia eksperymentów uruchomiono kilka sesji algorytmu genetycznego z różnymi argumentami. Poniżej znajdują się ich omówione wyniki.

4.1 Sprawdzenie parametrów

... TBA ...

4.2 Porównanie głębokości Minimaxa

... TBA ...

4.3 Możliwości rozwoju projektu

Mimo osiągnięcia zamierzonych celów, wciąż pozostaje kilka aspektów pracy które da się rozwinąć lub ulepszyć. Otwarcie kodów źródłowych na rozszerzenia może ułatwić dodanie nowych funkcjonalności, bądź modyfikację kopii niektórych klas. Poniżej znajduje się kilka propozycji pociągnięcia projektu dalej.

4.3.1 Optymalizacje

Przeszukiwanie przestrzeni stanów w Minimaxie można usprawnić o bazę rozpocząć i zakończeń - algorytm mógłby zaczynać gry lub odpowiadać jednym ze standardowych rozpocząć turniejowych i, gdy nadarzy się sposobność, dążyć do jednego z zakończeń. Na podobnej zasadzie działał np. Deep Blue, komputer firmy IBM który jako pierwsza maszyna na świecie zwyciężyła w partii szachów z ówczesnym mistrzem świata Garri Kasparowem w 1996 roku [3].

Innym pomysłem na potencjalne skrócenie obliczeń jest zastosowanie Hashmapy przeszukanych stanów z przypisanymi im ocenami. Pomysł ten bazuje na obserwacji, że do niektórych stanów na planszy można dojść z kilku innych stanów. Rozpatrując stan, algorytm sięgałby do takiej Hashmapy i jeżeli znalazłby hash tego stanu, automatycznie przyznawałby ocenę bez konieczności schodzenia głębiej w drzewie. Warto jednak zaznaczyć, że operacje dodania i przeszukania w Hashmapie nie są stałe (zajmują czas logarytmiczny zależny od liczby stanów) i wykonywane są dla każdego stanu, a samych stanów może być w pewnym momencie bardzo dużo.

Jeszcze innym miejscem w którym optymalizacja mogłaby znacznie poprawić wydajność, szczególnie dla sesji algorytmu genetycznego, są funkcje analizujące planszę i wyliczające parametry do oceny heurystycznej. W momencie pisania pracy istnieje wiele funkcji wyliczających wartość jednego parametru w czterech wariantach (sojusznicze piony, sojusznicze damki, przeciwne piony, przeciwne damki). Warto jednak zwrócić uwagę na fakt, że prawie każda z tych funkcji musi przeanalizować całą planszę po każdym polu. Stąd pomysł na optymalizację: utworzyć specjalną klasę *StateAnalyzer* przechowującą informacje o wszystkich parametrach oraz booleanową flagę mówiącą czy informacje te są aktualne. Przy każdym ruchu flagę ustawiałoby się na *false*, a w koniecznej chwili (gdy zewnętrzny obiekt prosi o wartość parametru) ustawia flagę na *true* i oblicza wartości każdego parametru od razu. Innym sposobem na optymalizację w tym zakresie jest obliczanie na nowo wartości tylko tych parametrów, które rzeczywiście zmieniają się w danym ruchu.

Na samym końcu można by było spróbować polepszyć wydajność samego algorytmu genetycznego. Łatwo zauważyć, że do wybrania lepiej przystosowanej połowy populacji osobników (z czynnikiem losowym) nie potrzeba sortowania gorzej przystosowanej połowy populacji, ponieważ ta zostaje odrzucona. Sortowanie tylko części populacji miałoby szansę wywrzeć zauważalny wpływ na czas wykonania obliczeń. Można też eksperymentować z wprowadzaniem innych algorytmów sortujących, chociażby QuickSort.

4.3.2 Walka z efektem horyzontu

Efektem horyzontu nazywamy problem w którym ograniczenie na wielkość przeszukiwanego kawałka przestrzeni stanów uniemożliwia dojście do potencjalnie lepszego rozwiązania znajdującego się krok dalej. Obecna implementacja Minimaxa w pracy jest podatna na problem horyzontu. Można częściowo temu zapobiec, zmuszając algorytm do rozpatrzenia dzieci stanu na maksymalnej głębokości, jeżeli jest on jedynym stanem pochodnym swojego rodzica (możliwość wykonania tylko jednego ruchu najczęściej oznacza wymuszone bicie, które jest poza kontrolą gracza). Można oszacować koszt takiego sprawdzenia jako większy zaledwie o jeden od średniego kosztu przejścia drzewa przeszukiwań - to tak jakby przenieść jedno poddrzewo o rząd niżej i umieścić w jego miejsce jeden stan. Ideę tę rozwija „przeszukiwanie uspokajające” (*Quiescence Search* [2]), które zatrzymuje przeszukiwanie poddrzew tylko na stanach spokojnych, czyli na takich które w najbliższych paru krokach nie zmienią się drastycznie (np. w Warcabach stany tuż przed biciem lub awansem piona do damki nie są spokojne).

4.3.3 Interfejs

W obecnej chwili program prowadzący rozgrywki w Warcaby uruchamiany jest z poziomu konsoli. Miłym rozszerzeniem byłoby stworzenie przyjaznego ludzkiemu użytkownikowi interfejsu do uruchamiania rozgrywek oraz intuicyjnego wprowadzania ruchów, chociażby poprzez kliknięcia. Nie zaszkodzą również dodatkowe opcje, np. możliwość cofnięcia ruchu, zapis i odczyt rozgrywki, przechowywanie interaktywnej historii rozgrywek. To wszystko można by było opakować w stronę webową i ją udostępnić.

Ciekawym pomysłem jest również napisanie adaptera który pozwalałby sztucznej inteligencji stworzonej w pracy prowadzić rozgrywki z innymi dostępnymi graczami komputerowymi w internecie. Pozwoliłoby to na poznanie siły wyznaczonej sztucznej inteligencji.

Podsumowanie

W podsumowaniu należy określić stan zakończonych prac projektowych i implementacyjnych. Zaznaczyć, które z zakładanych funkcjonalności systemu udało się zrealizować. Omówić aspekty pielęgnacji systemu w środowisku wdrożeniowym. Wskazać dalsze możliwe kierunki rozwoju systemu, np. dodawanie nowych komponentów realizujących nowe funkcje.

W podsumowaniu należy podkreślić nowatorskie rozwiązania zastosowane w projekcie i implementacji (niebanalne algorytmy, nowe technologie, itp.).

Bibliografia

[1] How checkers was solved. Web pages: https://www.theatlantic.com/technology/archive/2017/07/marion-tinsley-checkers/534111/.

[2] Quiescence search. Web pages: https://en.wikipedia.org/wiki/Quiescence˙search.

[3] I. Belda. *Umysł, maszyny i matematyka*. BUKA Books Sławomir Chojnacki, 2012.

[4] J. Mańdziuk, M. Kusiak, K. Walędzik. Evolutionary-based heuristic generators for checkers and give-away checkers. 2007.

[5] L. Pijanowski. *Przewodnik gier*. Iskry, 1978.

[6] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, S. Sutphen. Checkers is solved. 2007.

[7] J. Schaeffer, R. Lake. Solving the game of checkers. 1996.

Załącznik A

Zawartość płyty CD

W tym rozdziale należy krótko omówić zawartość dołączonej płyty CD.