Projektowanie algorytmów i metody sztucznej inteligencji, bud C-3,s.103

Temat projektu: Gry i sztuczna inteligencja

Projekt wykonał:

Jakub Kędzierski 248915

Data: 28.05.2020r

Termin zajęć: czwartek 9:15-11:00

Prowadzący: Dr inż. Łukasz Jeleń

1 Wstęp

Celem projektu było stworzenie gry, w której wykorzystane zostaną algorytmy sztucznej inteligencji. Wybrano grę warcaby i zastosowano w niej algorytm min-max. Dodatkowo posłużono się funkcją heurystyczną do oceny wartości stanu gry w dowolnym momencie.

Kod źródłowy gry został napisany w języku C++, a do oprawy graficznej skorzystano z biblioteki graficznej ułatwiającej tworzenie gier - SFML (Simple and Fast Multimedia Library). W kodzie programu można wyróżnić część odpowiedzialną za oprawę graficzną oraz część odpowiedzialną za matematyczny opis gry. Do tworzenia szaty graficznej gry skorzystano z tekstur dostępnych na stronach internetowych zapewniających darmową grafikę na użytek własny (linki w bibliografii), które następnie zmodyfikowano w programie GIMP. Jakoś oprawy graficznej jest na średnim poziomie, jednak głównym priorytetem było wykorzystanie metod sztucznej inteligencji.

Po zaimplementowaniu algorytmów wykonano kilka testów zachowania sztucznej inteligencji w grze z człowiekiem.

2 Opis gry - warcaby

Warcaby to dwuosobowa gra planszowa, w którą gra się na warcabnicy/szachownicy o 64 polach. Gra ma wiele odmian, spośród których wybrano i zaimplementowano najpopularniejsze i najczęściej spotykane warcaby klasyczne. W grze na start wykorzystuje się 24 pionki po 12 dla gracza. Pionki ustawione są na ciemniejszych polach i tylko po nich mogą się poruszać. Dotarcie do linii końcowej oznacza promocję, czyli zamianę zwykłego pionka na damkę. Zbijanie pionków polega na "przeskakiwaniu" po ukosie, gdy dostępne jest wolne miejsce za pionkiem przeciwnika.

Ponadto zastosowano poniższe zasady:

- wszystkie dozwolone bicia są obowiązkowe (wielokrotne, następujące po sobie również)
- możliwe są bicia wielokrotne
- niedozwolone są bicia do tyłu dla zwykłego pionka, czyli w przeciwnym kierunku niż ten którym porusza się pionek
- zwykły pionek może się poruszać jedno pole po ukosie w stronę przeciwnika lub zbijać po ukosie w stronę przeciwnika
- damka może poruszać się jedno pole po ukosie w dowolną stronę lub zbijać w dowolną stronę
- gra kończy się gdy jeden z graczy straci wszystkie pionki lub nie ma żadnych możliwych ruchów do wykonania

3 Zastosowane techniki SI

3.1 Algorytm min-max

Jest to metoda minimalizacji maksymalnych możliwych strat. W algorytmie zakładamy że przeciwnik za każdym razem wybierze najlepszy dla siebie ruch i dla tego ruchu musimy przygotować najlepszą dla nas odpowiedź (najlepszy nasz ruch). Algorytm jest rekurencyjny i do analizy kolejnych ruchów wywołuje sam siebie. Na początku tworzymy drzewo wszystkich możliwych ruchów (stanów w grze) do pewnej głębokości ograniczonej przez moc obliczeniową komputera. Każdemu stanowi przypisujemy pewną ocenę (do tego celu używamy funkcji heurystycznej opisanej w następnym podpunkcie). Mamy dwóch graczy gracz min i gracz max. Celem graczy jest zdobycie mniejszej (min) lub większej (max) liczby punktów po ostatnim ruchu. Dla gracza min wybieramy ruchy które w ostatecznym rozrachunku doprowadzą do mniejszej wartości, a dla gracza max do większej wartości końcowej.
Złożoność obliczeniowa algorytmu to: b^m, gdzie b to liczba dostępnych ruchów w każdym wierzchołku drzewa, a m maksymalna głębokość drzewa. Na potrzeby obliczenia złożoności obliczeniowej zakładamy że liczba dostępnych ruchów się nie zmienia (zakładamy średnią liczbę dostępnych ruchów).
Złożoność pamięciowa algorytmu to: $b \cdot m$

Zapis algorytmu w pseudokodzie:

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value :=
        for each child of node do
            value := max(value, minimax(child, depth  1, FALSE))
        return value
    else (* minimizing player *)
        value := +
        for each child of node do
            value := min(value, minimax(child, depth  1, TRUE))
        return value
```

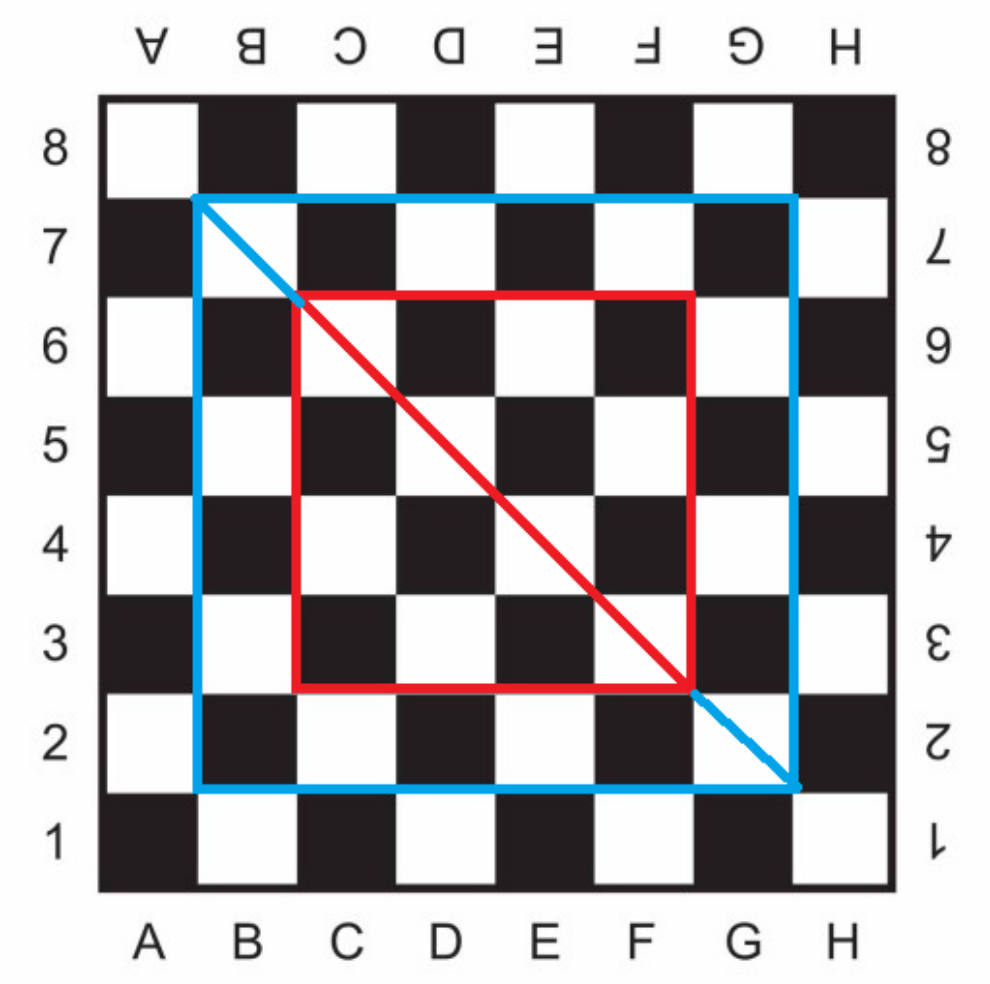
3.2 Funkcja heurystyczna

Funkcja heurystyczna jest funkcją oceniającą wartość stanu gry w danym momencie. Funkcja tłumaczy stan gry na formę zrozumiałą dla komputera, czyli np. liczbę pionków na określone punkty. Jest to metoda szukania rozwiązania optymalnego. Poprawnie ułożona funkcja heurystyczna jest konieczna do właściwego działania algorytmu min-max. Funkcja heurystyczna zaimplementowana w projekcie przyznawała punkty:

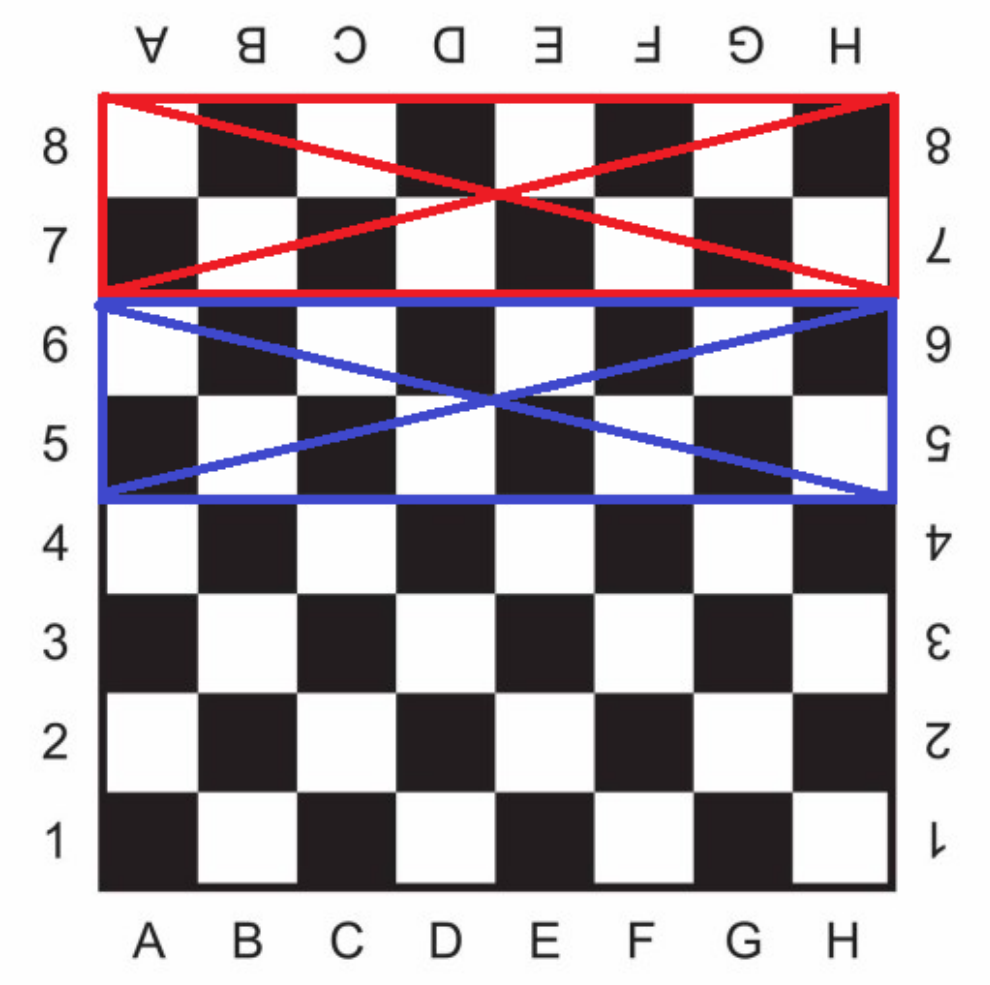
- 40pkt zwykły pionek
- 100pkt damka
- 5pkt położenie pionka w ścisłym centrum (na rysunku 1. zaznaczone czerwonym kwadratem)

- 3pkt położenie pionka w słabszym centrum (na rysunku 1. zaznaczone niebieskim kwadratem)
- 0pkt położenie pionka na obrzeżach szachownicy/warcabnicy
- 10 pkt położenie zwykłego pionka na I poziomie (na rysunku 2. zaznaczone czerwonym prostokątem)
- 5 pkt położenie zwykłego pionka na II poziomie (na rysunku 2. zaznaczone niebieskim prostokątem)

W perspektywie analizy kilka ruchów do przodu funkcja zapewnia również analizę bicia (strata pionka i jego punktów) oraz promocje (+60pkt przy zamianie pionka na damkę). Ilość zdobywanych punktów dobierana była eksperymentalnie oraz na podstawie strategi używanych przez graczy. Wyróżniono poziomy, aby pionki dążyły do promocji oraz centrum, aby bierki danego gracza poruszały się w zwartym szyku, co zmniejsza prawdopodobieństwo wielokrotnych stratnych bić oraz pozwala na "odbijanie" straconych pionków.



Rysunek 1: Warcabnica/szachownica z zaznaczonym ścisłym (czerwony kolor) oraz słabszym (niebieski kolor) centrum



Rysunek 2: Warcabnica/szachownica z zaznaczonym poziomem I (czerwony kolor) oraz poziomem II (niebieski kolor). Dla przeciwnika poziomy analogicznie, po drugiej stronie warcabnicy/szachownicy.

4 Testy i wnioski

Wraz ze zwiększaniem rozpatrywanych ruchów do przodu analizowanych przez komputer znacznie zwiększała się złożoność obliczeniowa algorytmu a tym samym czas oczekiwania na ruch komputera (sztucznej inteligencji). Optymalna liczba kroków liczonych do przodu, dla których gra jest swobodna to 5-6. Podczas testów szybkiej rozgrywki z komputerem (do 15 sekund namysłu na ruch) większość razy wygrywał człowiek (znający różne strategie gry). Wydajność algorytmu przetestowałem również na domownikach, którzy nie znali żadnych strategi gry w warcaby, a jedynie podstawowe zasady poruszania. W takim wypadku przy dowolnej szybkości rozgrywki częściej wygrywał komputer (4 razy na 5 domowników). Stworzone algorytmy sztucznej inteligencji nie są doskonałe jednak dla początkującego gracza znającego jedynie podstawowe zasady mogą stwarzać wrażenie mocnego przeciwnika. W celu poprawy efektywności algorytmu należałoby dodać dodatkowe warunki oceny w funkcji heurystycznej oraz zmniejszyć złożoność algorytmu np. poprzez modyfikację min-max (cięcia alfa-beta). Podczas pracy nad projektem wykonałem szybki internetowy kurs pracy z biblioteką SFML, co pozwoliło na zaprezentowanie gry w formie graficznej (jest to moja pierwsza styczność z bibliotekami graficznymi).

5 Bibliografia

Tekstury pobrane ze stron udostępniających tekstury na użytek własny, które następnie modyfikowano w programie GIMP:

- szachownica - https://www.vectorstock.com/royalty-free-vector/empty-chess-board-vector-3515645, dostęp 05.2020
- biały pionek i czarny pionek- https://emojiterra.com, dostęp 05.2020
- czarna oraz biała damka - https://ya-webdesign.com/imgdownload.html, dostęp 05.2020

Literatura z której korzystałem przy pracy nad projektem:

- strona z opisem algorytmu min - max :https://cis.temple.edu/ vasilis/Courses /CIS603/Lectures/l7.html, dostęp 05.2020
- strona z opisem strategi stosowanych przy grze w warcaby: https://kcir.pwr.edu.pl/ witold/aiarr/2009_projekty/warcaby/, dostęp 05.2020
- strona z opisem algorytmu min - max: https://en.wikipedia.org/wiki/Minimax, dostęp 05.2020
- strona z opisem algorytmu min - max: https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/, dostęp 05.2020
- strona z internetowym kursem SFML: http://cpp0x.pl/kursy/Kurs-SFML-2-x-C++/460, dostęp 05.2020