26.05.2020r.

Paweł Troszczyński 248925

Prowadzący zajęcia: mgr inż. Marcin Ochman

Termin zajęć: wtorek 15:15-16:55

Projektowanie Algorytmów i Metody Sztucznej Inteligencji
Gry i AI

1. Wstęp

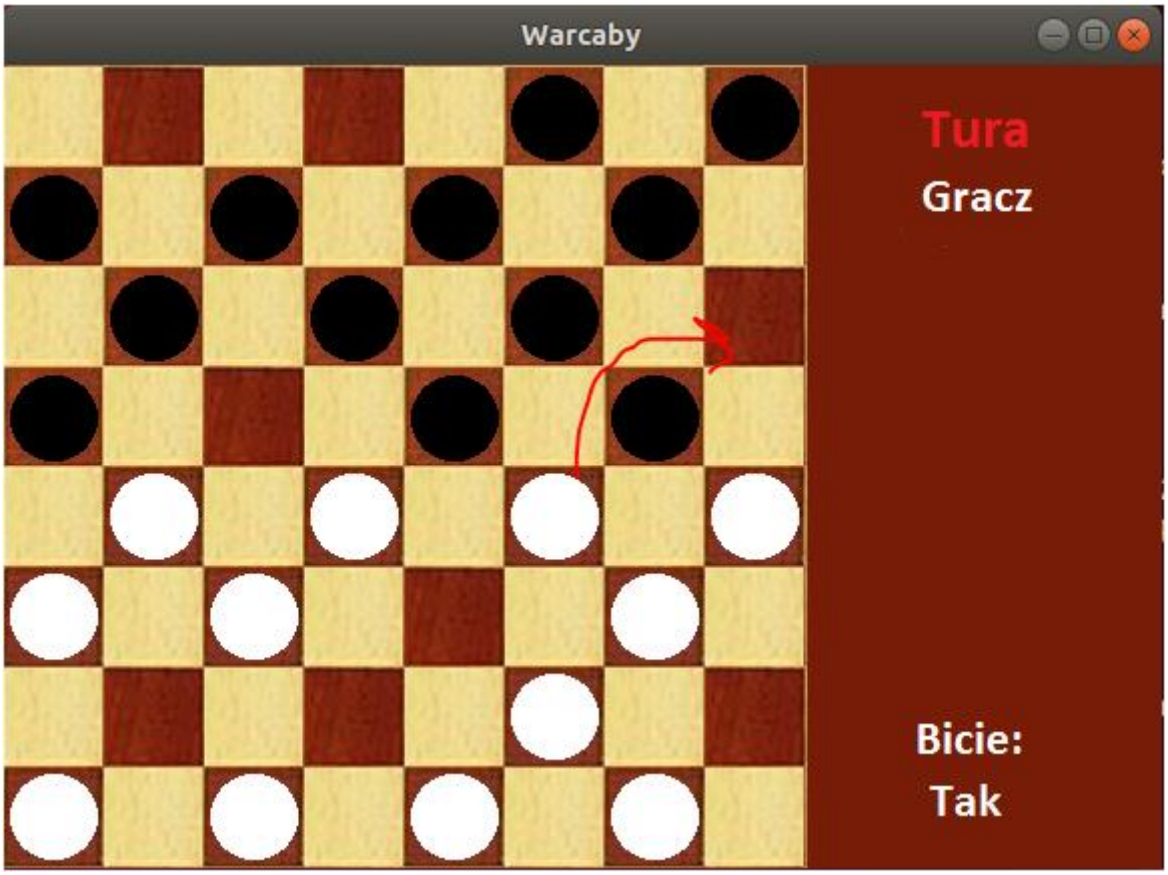
Celem projektu było napisanie gry zawierającej zaimplementowany algorytm, który stanowić będzie sztuczną inteligencję.

2. Gra i zasady

Wybraną grą są warcaby. Grafika została zrealizowana za pomocą SFML w wersji 2.5. Na początku każdy z graczy posiada 12 pionków znajdujących się na planszy 8 na 8. Pionki mogą znajdować się tylko na ciemniejszych polach. Białe pionki zaczynają na 3 dolnych rzędach (po 4 pionki na rząd) zaś czarne pionki na 3 górnych rzędach. Wybór pionka następuje po naciśnięciu go lewym przyciskiem myszy zaś ponowne naciśnięcie wskazuje miejsce gdzie ma się udać pionek. Wybór niepoprawnej operacji związany jest z brakiem reakcji i oczekiwaniem, aż gracz wskaże poprawne działanie.

Gracz kontroluje białe pionki zaś ruch czarnych realizowany jest za pomocą algorytmu min-max, co stanowi metodę sztucznej inteligencji. Grę zawsze zaczynają biali, dlatego w tym przypadku jest tura gracza. Pionki mogą poruszać się po ukosie na wolne pola: białe w górę, czarne w dół. Jeżeli pole, na które może wejść pionek, zajęte jest przez przeciwny pionek oraz następne pole jest wolne dostępne staje się bicie. Jest to obowiązkowe działanie, co oznacza, że gdy jest ono dostępne trzeba je wykonać. W przypadku wystąpienia kilku możliwości należy zrealizować dowolną z nich. Jeżeli po wykonaniu bicia pionek dalej może bić to także jest do tego zobligowany. Bicie może następować zarówno w górę jak i w dół niezależnie od koloru pionka.

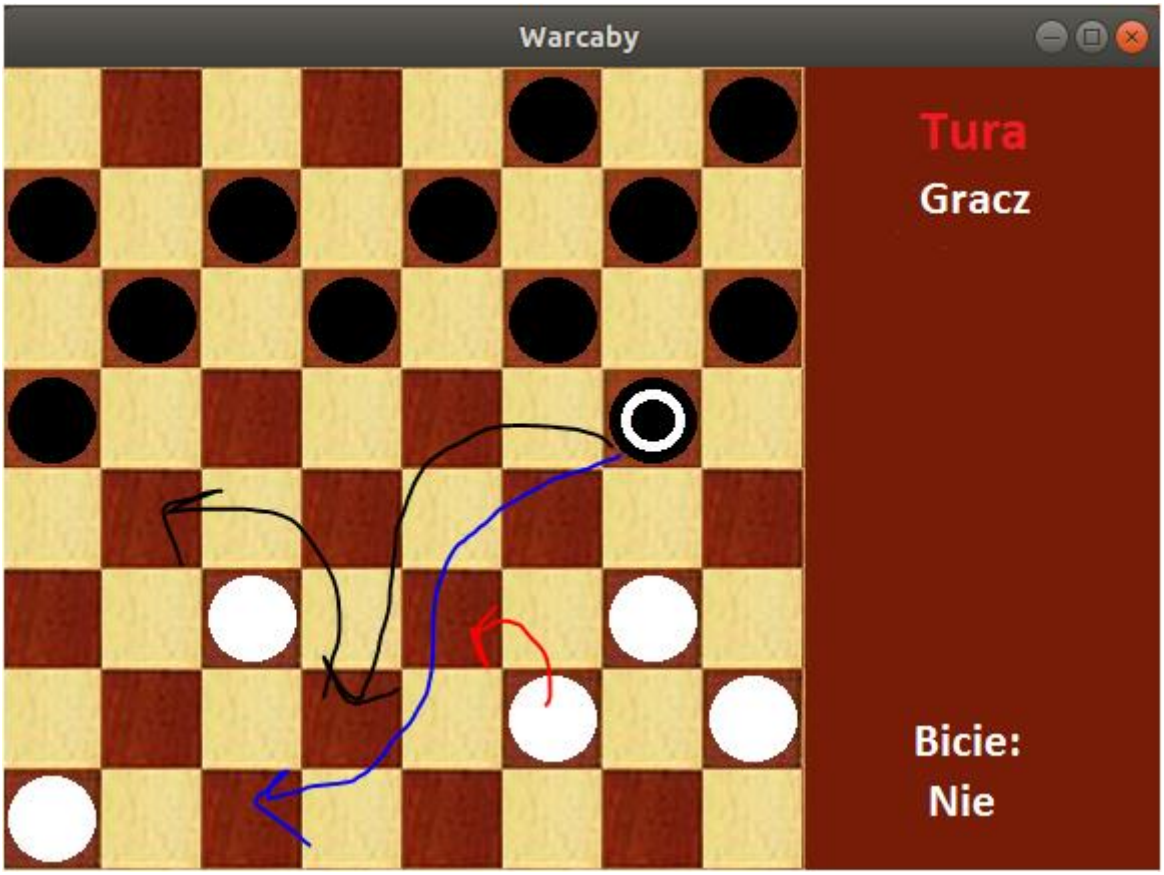
Przykład obowiązkowego bicia dla białych:



Jeżeli pionek dojdzie do najdalszego możliwego rzędu dla siebie tj. górnego dla białych i dolnego dla czarnych – staje się damką. Damka może poruszać się nieograniczoną ilość pól po skosach pod warunkiem, że wszystkie pola po drodze są wolne. Jeżeli na drodze takiej damki znajduje się dokładnie jeden z przeciwnych pionków, damka może go zbić pod warunkiem, że za tym pionkiem znajduje się przynajmniej jedno wolne pole. Jeżeli pól za zbijanym pionkiem jest więcej damka może wybrać dowolne z tych dwóch pół. Bicie w przypadku damki także jest obowiązkowe, więc po wybraniu tego pola, jeżeli wciąż jest możliwe jakieś bicie tą damką, należy wykonać to bicie. Jeżeli pionek w wyniku bicia znajdzie się na polu, które zamienia go w damkę, ale wciąż ma możliwe bicie to musi je wykonać i nie staje się damką. Jeżeli pionek w wyniku bicia staje się damką jego ruch kończy się.

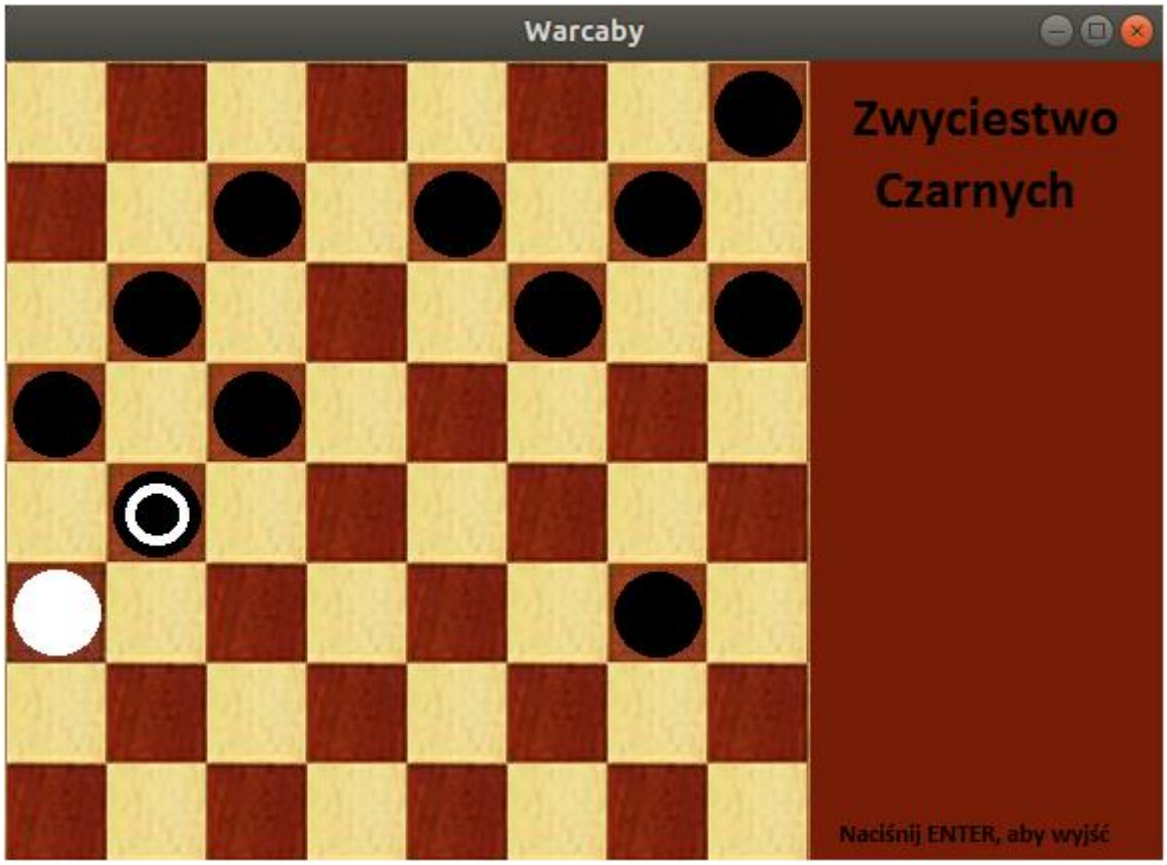
Jeżeli pionek biały dokona ruch jak pokazano czerwoną strzałką to czarny będzie mógł wykonać np. pokazane czarnymi strzałkami bicie. Przejdzie on przez najniższy rząd, ale przez to, że nadal będzie mógł bić nie stanie się damką. Wynik po wykonaniu ruchu białym pionkiem zgodnie z czerwoną strzałką:

Jeżeli biały pionek wykona ruch zgodnie z czerwoną strzałką to czarny pionek będzie musiał wykonać podwójne bicie, które zakończy na ostatnim dla jego koloru rzędzie, a co za tym idzie zamieni się w damkę. Jednak w tym momencie jego ruch się zakończy i nie będzie mógł kontynuować bicia następnego pionka jako damka. Wynik opisanej sytuacji:



Jeżeli biały pionek wykona ruch zgodnie z czerwoną strzałką, czarna damka będzie mogła wykonać ruch w miejsce wskazane niebieską lub czarną strzałką. Jeżeli wykona ruch zgodnie z czarną strzałką będzie musiała wykonać kolejne bicie. Wynik opisanej sytuacji przy wyborze przez damkę drogi czarnych strzałek:

Gra kończy się, gdy zabraknie możliwych ruchów(z powodu utraty wszystkich pionków lub zablokowaniu wszystkich dostępnych) dla koloru, który właśnie powinien wykonywać turę. Oznacza to że, gdy np. następuje tura białych i nie może on wykonać żadnego ruchu zwycięzcą zostają czarni.



Powyżej następują tura białych, który posiada jeden pionek, jednak jedyne pole, na które mógłby się ruszyć, jest zajęte przez czarną damkę, a pole za nim przez czarny pionek, co oznacza zwycięstwo czarnych.

Plansza gry to macierz 8 na 8, która w każdej ze swoich komórek przechowuje pewną wartość określającą co znajduje się na danym polu:

a) 0 – oznacza puste pole
b) 1 – oznacza pionek czarny
c) -1 – oznacza pionek biały
d) 2 – oznacza damkę czarną
e) -2 – oznacza damkę białą

3. Algorytm min-max

Aby zastosować algorytm, należy na początku wygenerować tzw. drzewo gry. Polega to na wygenerowaniu możliwych ruchów dla gracza, który obecnie musi wykonać turę. Następnie dla każdego z tych ruchów generowane są możliwe ruchy przeciwnego gracza jakie będzie mógł wykonać po wykonaniu wybranego z ruchów. Głębokość dokonywania tej operacji zależy od implementacji, jednak powinno przyjąć się głębokość zależną od złożoności gry oraz dostępnej mocy obliczeniowej. Im większa głębokość tworzonego drzewa, tym zostaje przewidziane więcej ruchów. Oznacza to, że większa głębokość będzie powodowała trudniejszą do pokonania sztuczną inteligencję. Po wygenerowaniu takiego drzewa należy ocenić wartość liczbową każdej gałęzi na najniższym poziomie

zależną od przyjętej funkcji oceniającej. Jeden z graczy dąży do uzyskania jak największej tej wartości(oznaczającej największą przewagę dla niego nad rywalem),zaś drugi do jak najmniejszej wartości(co określa największą przewagę drugiego gracza). Następnie zaczynając od przedostatniego poziomu drzewa dokonujemy jednej z dwóch operacji:

a) Jeżeli aktualny poziom odpowiada graczowi, który aktualnie musi wykonać ruch to należy wybrać maksymalną z wartości ze wszystkich węzłów potomnych poziomu niżej.
b) Jeżeli aktualny poziom nie odpowiada graczowi, który aktualnie musi wykonać ruch to wybieramy najmniejszą z wartości ze wszystkich węzłów potomnych poziomu niżej

Operację tę powtarzamy dla każdego wyższego poziomu. Ostatecznie wybieramy ruch, który prowadzi do węzła o maksymalnej ocenie. Algorytm min-max przy przewidywaniu ruchów zakłada, że przeciwnik wykona najlepszy dla siebie ruch.

Do realizacji algorytmu wykorzystałem dwie funkcje void Computer(bool white) oraz int minmax(int depth, bool white). Pierwsza z funkcji generuje listę możliwych ruchów dla czarnych (jeżeli argument white jest nieprawną) lub dla białych(jeżeli argument white jest prawdą). W programie gracz steruje białymi, zaś algorytm oblicza najlepszy ruch dla czarnych. Funkcja ta dla każdego z możliwych ruchów wywołuje funkcje minmax(int, bool), podając jako argumenty głębokość zmniejszoną o jeden oraz przeciwną wartość argumentu white, co oznacza zmianę gracza. Przed wykonaniem ruchu tworzona jest kopia wykonanego ruchu, aby po zakończeniu przywrócić planszę do pierwotnego wyglądu. Po uzyskaniu wyniku funkcji określane jest czy jest ona korzystniejsza(większa dla czarnych, mniejsza dla białych) od poprzednio wybranej ścieżki oraz ustawiany jest iterator na tę ścieżkę, jeżeli jest ona korzystniejsza. Plansza zostaje cofnięta do pierwotnego stanu, po czym operacja powtarzana jest dla kolejnych możliwych ruchów. Po przeanalizowaniu wszystkich możliwych ruchów zostaje wykonany ruch na który wskazuje iterator kontrolujący najkorzystniejszą z dróg. Funkcja minmax(int, bool), działa podobnie, jednak posiada kilka różnic. Jeżeli głębokość wynosi 0 dla tej funkcji to obliczana jest wartość planszy zgodnie z przyjętą funkcją, która opisana jest poniżej. Jeżeli głębokość jest jednak dodatnia, funkcja ta różni się tym, że nie przechowuje iteratora na daną ścieżkę, a zaledwie jej wartość oraz na końcu działania zwraca tę wartość, a plansza jest identyczna do tej z przed wywołania tej funkcji. Jeżeli funkcja ta wygeneruje pusty zbiór możliwych ruchów dla danego gracza zwróci wartość maksymalną (gdy rozpatruje ruch białych) lub minimalną (gdy rozpatruje ruch czarnych) dla typu danych całkowitych (int). Zapewnia to odrzucenie danej drogi, jeżeli nie jest ona jedyną z możliwych dróg, ponieważ oznacza ona porażkę rozpatrywanego gracza. Dzięki wykorzystaniu rekurencji, funkcje te tworzą drzewo gry oraz dokonują operacje zgodne z algorytmem min-max.

Funkcja oceniająca zwraca wartość dodatnią, gdy przewagę mają czarni, ujemną, gdy przewagę mają biali oraz zero, gdy szanse są wyrównane. Wartość początkowa tej funkcji wynosi 0. Funkcja ta przeszukuje całą planszę. Za każdą damkę czarną zwiększa wartość o 4, za każdą białą damkę obniża wartość o 4. Za każdy czarny pionek zwiększa wartość o 1 oraz jeżeli pionek znajduje się jeden lub dwa rzędy od stania się damką jej wartość ponownie zostaje zwiększona o jeden. Dla białych pionków sytuacja jest analogiczna z tym, że wartość jest obniżana. Powoduje to, że strona kontrolowana przez sztuczną inteligencję dąży do damek oraz pozycjonowania pionków jak najbliżej ostatniego ze swoich rzędów.

Maksymalną głębokością zapewniającą racjonalny czas okazała się głębokość równa 8. Wartość określono w sposób doświadczalny. Przeprowadzono analizę czasu wykonywania ruchu przez sztuczną inteligencję (czyli czas od rozpoczęcia generowania możliwych ruchów do wykonania najkorzystniejszego z ruchów). Dokonano jej dla pięciu głębokości: 4, 5, 6, 7, 8. Dla każdej z nich określono maksymalny, minimalny oraz średni czas potrzebny sztucznej inteligencji na wykonanie ruchu. Wyniki badań przedstawia tabela:

głębokość		4	5	6	7	8
czas[ms]	największy	12,82	57,87	289,33	2185,83	12692,30
	najmniejszy	0,03	0,04	0,04	0,04	25,63
	średni	5,19	18,09	93,33	626,67	4082,04

3. Wnioski

- Algorytm min-max można stosować dla gier, w których uczestniczą dwaj gracze wykonujący ruchu naprzemiennie albo jednocześnie.
- Strategia minimaksowa dąży do minimalizacji szans przeciwnika na zwycięstwo przy jednoczesnym maksymalizowaniu własnych szans na zwycięstwo.
- Algorytm min-max zakłada, że przeciwnik będzie wykonywał najlepsze z dostępnych dla siebie ruchów, co powoduje, że często decyzja nie pada na najlepszy ruch w aktualnym momencie, a na ruch, który w przyszłości spowoduje największą przewagę.
- Głębokość określa liczbę przewidywanych ruchów do przodu, co oznacza, że definiuje ona poziom trudności sztucznej inteligencji. Im większa głębokość tym trudniej będzie wygrać przeciwko sztucznej inteligencji.
- Tworzenie drzewa gry jest czasochłonną operacją i czas potrzebny na jego wygenerowanie znacząco zwiększa się wraz z głębokością. Przyjmując, że dla każdej sytuacji liczba możliwych ruchów wynosi średnio x oraz przyjęta głębokość wynosi d, otrzymujemy x^d wygenerowanych ścieżek dla pełnego drzewa. Jest to funkcja wykładnicza, w której w wykładniku znajduje się głębokość. Potwierdza to znaczący wpływ głębokości na czas wykonywania algorytmu.
- W praktyce jednak liczba możliwych ruchów zależy od sytuacji w grze. Pokazują to różnice między największymi i najmniejszymi czasami potrzebnymi do wykonania ruchu przez sztuczną inteligencję, które różnią się nawet o 4 (dla głębokości 8) czy 6 (dla głębokości 7) rzędów. Niezależnie od tego średni czas na wykonanie tych algorytmów wciąż ulega nawet ponad sześciokrotnemu zwiększeniu przy zwiększeniu głębokości o jeden. A wielokrotność ta rośnie z każdym kolejnym wzrostem głębokości.

Bibliografia:
http://wazniak.mimuw.edu.pl/index.php?title=Sztuczna_inteligencja/SI_Modu%C5%82_8_-_Gry_dwuosobowe
https://en.wikipedia.org/wiki/Minimax
https://www.sfml-dev.org/tutorials/2.5/