Wprowadzenie do sztucznej inteligencji

Laboratorium nr. 3

Bartosz Latosek

310 790

1. Opis ćwiczenia

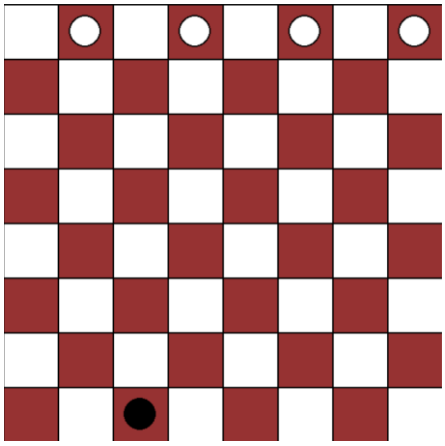
Ćwiczenie polega na implementacji algorytmu minmax dla deterministycznej gry dwuosobowej – „Wilk i owce”.

Użyte technologie:

Python 3.9.0 wraz z bibliotekami:

- Matplotlib(3.4.3)
- Pygame
- Argparse

1.1 Opis gry „Wilk i owce”



Rysunek 1

W grze rywalizuje ze sobą dwoje graczy, z których jeden kontroluje białe, a drugi czarne pionki. Celem gracza owiec jest zablokowanie drogi wilkowi, aby ten nie przedostał się na drugą stronę planszy. Pionki mogą poruszać się jedynie po ukosie, analogicznie do warcab, jednakże podczas gdy wilk może poruszać się w każdą stronę, owce nie mogą się cofać.

1.2 Algorytm MinMax

```
Dane: aktualna_pozycja, głębokość, tura_max

if głębokość == 0 lub KoniecGry(aktualna_pozycja):
return ewaluacja(aktualna_pozycja)

if tura_max:
   maxEval = -infinity
   for each child of aktualna_pozycja:
      eval = minmax(child, głębokość – 1, tura_min)
      maxEval = max(maxEval, eval)
   return maxEval

if tura_min:
   minEval = infinity
   for each child of aktualna_pozycja:
      eval = minmax(child, głębokość – 1, tura_max)
      minEval = min(minEval, eval)
   return minEval
```

Gdzie:

aktualna_pozycja - W moim przypadku tabela reprezentująca aktualne rozmieszczenie pionków na planszy

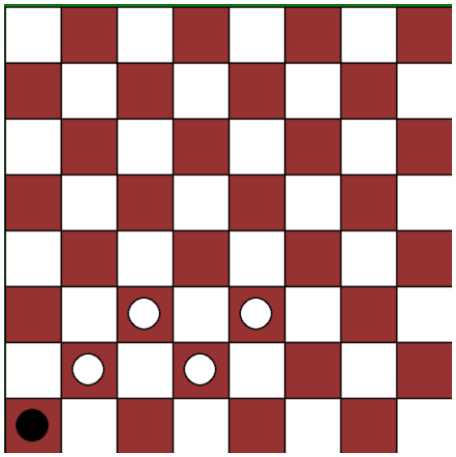
głębokość – Aktualne zagłębienie algorytmu **minmax()**

tura_max, tura_min – Wartości boolowskie, reprezentują dla kogo aktualnie przeprowadzamy ewaluacje

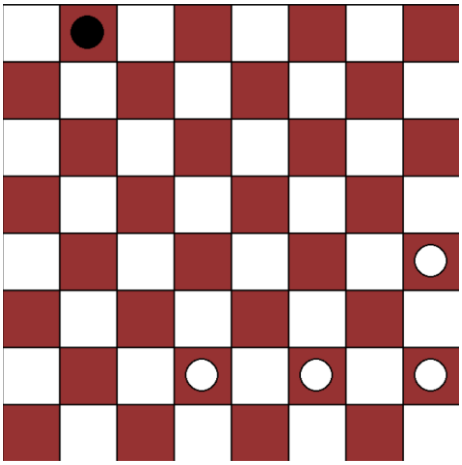
ewaluacja(*aktualna_pozycja****)*** **–** Funkcja ewaluująca, szczegółowiej opisana w dalszej części sprawozdania.

1.3 Przykład działania

W związku z formą ćwiczenia ciężko będzie zamieścić przykład działania dla całej rozgrywki więc ograniczę się jedynie dla końcowych rozmieszczeń pionków na planszy.



Zwycięstwo owiec



Zwycięstwo wilka

Powyższe obrazki obrazują niejednostronność gry, jednakże różnią się one funkcją ewaluującą aktualne położenie pionków. Ruchy wilka i owiec były determinowane poprzez działanie algorytmu minmax przy początkowym zagłębieniu *depth = 5*.

2. Analiza algorytmu

Jedyną możliwością zmiany działania algorytmu jest zmiana funkcji ewaluującej. Wilk zawsze dąży do jak najwyższego wyniku, z kolei owce do najniższego. Z pracy przy niej dowiedziałem się, że niemożliwe jest (albo bardzo ciężkie) odnalezienie funkcji, która nie będzie faworyzować żadnej ze stron. Ostateczna wersja tej funkcji prezentuje się następująco:

```
Dane: aktualna_pozycja

wynik  = 0
wilcze_ruchy = WyznaczMożliweRuchyWilka(aktualna_pozycja)
if wilk znajduje się w pierwszym rzędzie :                    // 1
   wynik += 10000

wynik += n * (rozmiar planszy – aktualna_pozycja_wilka.rząd() )    // 2

for each ruch in wilcze_ruchy:                                // 3
   wynik += m
```

Gdzie **m, n** – parametry całkowite.

Ad. 1

Jeżeli wilk doszedł do końca planszy, wygrał grę – dodaj do wyniku dużą sumę.

Ad. 2

Im wyżej na planszy znajduje się wilk, tym lepiej dla niego

Ad. 3

Im więcej wolnych ruchów ma wilk, tym lepiej dla niego

Jak widać, ogólny wynik *aktualnej_pozycji* zależy głównie od wilka. Jest to w zupełności wystarczające, i bardziej optymalne gdyż algorytm minmax ma jedną funkcję ewaluującą dla obydwu graczy.

PRZYKŁADOWO:

Dla **n** = 10 i **m** = 10

Rozgrywka wydaje się zbalansowania. Na początku żaden z graczy nie wykazuje przewagi nad drugim. Dopiero w końcowej fazie gry, owce popełniają **błąd** przez co wilkowi udaje się wygrać.

Błąd ten jest spowodowany tym, że dla „niskich" położeń wilka większy wynik uzyskuje on przez posiadanie dużej ilości wolnych ruchów aniżeli przez wysokość, na której się znajduje, przez co owce priorytetyzują pozbawienie wilka możliwości ruchów nad zablokowaniem mu przemieszczania się bliżej celu.

W celi poprawy działania algorytmu, w wyniku metody prób i błędów udało mi się ustalić, że parametr **m** powinien przyjąć wartość 5.

Przy tak dobranych parametrach nie udało mi się w licznych próbach zaobserwować zwycięstwa gracza wilka. Owce są w stanie zablokować każdy jego ruch, przez co doszedłem do poniższego wniosku.

WNIOSEK 1

Dla najoptymalniejszych ruchów gracza owiec i gracza wilka, na podstawie przeprowadzonych rozgrywek stwierdzam, że gra faworyzuje gracza owiec. Taktyka którą udało mi się zaobserwować jest przemieszczanie pionków – owiec równolegle, tak żeby żadna z nich nie odstawała od grupy. W ten sposób wilkowi nie pozostaje żadna ścieżka do pierwszego rzędu. Ponadto, nie ma on możliwości przejść za owce w żadnym momencie rozgrywki i kończy zablokowany w dolnym rogu planszy.

Przeanalizujmy teraz , jak wygląda przebieg rozgrywki w przypadku, gdy:

Wilk wykonuje ruchy losowe, podczas gdy owce korzystają z minmax.

Wynik tej rozgrywki był przewidywalny, owce w łatwy sposób pokonały wilka w każdej przeprowadzonej rozgrywce.

Owce wykonują ruchy losowe, podczas gdy wilk korzysta z minmax.

Losowe ruchy owiec nie stanowiły żadnej przeszkody dla wilka, za każdym razem to on wygrywał rozgrywkę.

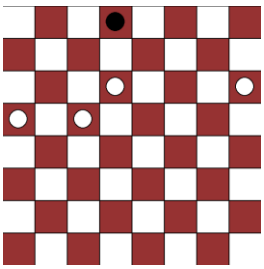
Owce i wilk wykonują losowe ruchy.

Nie specjalnie dużo wynika z takiego sposobu przeprowadzania rozgrywki, ale większość razy wygrywał wilk ponieważ jako ostatni miał jeszcze wolne ruchy.

Wpływ głębokości algorytmu minmax na jego działanie.

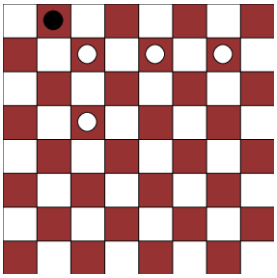
Przeanalizujmy teraz w jaki sposób zachowa się algorytm minmax przy różnych wartościach zagłębienia. Działanie najlepiej będzie zaobserwować na rozgrywce, w której obydwie strony korzystają z tego algorytmu.

N = 1



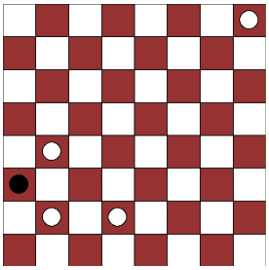
Wilk pokonał owce całkiem sprawnie, ponieważ dobierał swoje ruchy tak, żeby jak najszybciej dojść do celu. Owce z kolei nie mogły przewidzieć gdzie poruszy się wilk przez co ich szyk szybko został przełamany.

N = 2



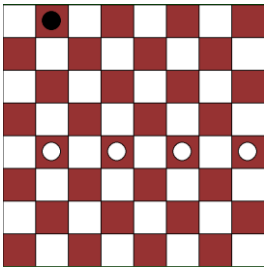
Analogicznie dla n = 1.

N = 3



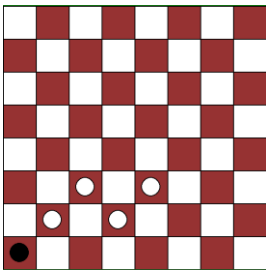
Zaskakująco, n = 3 wystarcza, żeby owce skutecznie przewidywały i blokowały ruchy wilka. Nie jest to wystarczające dla wilka, który nie przewiduje wystarczająco szybko, że za kilka ruchów zostanie zablokowany.

N = 4



Dla n = 4 wilk jest w stanie skutecznie przewidzieć kiedy zostanie zablokowany i po kilkunastu ruchach udaje mu się wyminąć owce.

N = 5



Dla n = 5 owce są w stanie przewidzieć gdzie wilk skieruje swoje następne ruchy przez co nie rozbijają swojego szyku i nie pozostawiają wilkowi drogi do celu.

Dla n > 5 czas znalezienia najoptymalniejszego ruchu jest odczuwalnie długi, przez co bezsensowne jest analizowanie działania algorytmu dla dalszych wartości zagłębienia.

3. Porównanie wyników rozgrywki.

Przy losowej pozycji startowej wilka, głębokości minmax = 5, obydwie strony korzystające z algorytmu minmax.

DLA 100 rozgrywek:

Wilk wygrał 11 razy, a **Owce** 89.

Potwierdza to moją wcześniejszą tezę, że owce mają znaczną przewagę w tej grze. Wygrywały większość rozgrywek dla rzędu startowego wilka < 3 (licząc od góry) i wszystkie dla których rząd startowy wilka >= 3.

(Kod na którego podstawie doświadczenie było przeprowadzone)

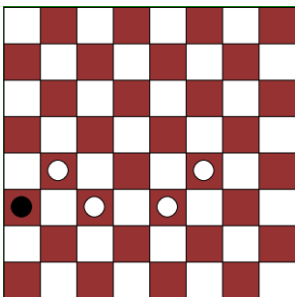
```python
def main(w_x: int, w_y: int, depth: int, w_r: bool, s_r: bool):

    seed = 0
    if not seed:
        seed = random.randrange(sys.maxsize)
    random.seed(seed)
    print(f"The seed is : {seed}")
    sheep_score = 0
    wolf_score = 0

    for n in range(0, 100):
        run = True
        w_x = random.randint(1, 7)
        w_y = random.choice((2, 4, 6, 0))
        if w_x % 2 == 0:
            w_y = random.choice((1, 3, 5, 7))
        board = Board(WIN, WIDTH, HEIGHT, BOARD_SIZE, w_x, w_y, depth, w_r, s_r)

        while run:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    run = False
            if board.update():
                run = False

        if board._turn == 0:
            sheep_score += 1
        else:
            wolf_score += 1

    print(f"Sheep: {sheep_score}, Wolf: {wolf_score}")
```

4. Wnioski Ogólne

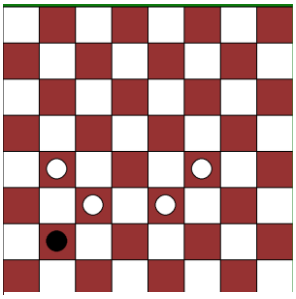
Tak jak wcześniej pisałem, gra faworyzuje raczej gracza owiec niż wilka przez co większość rozgrywek przy obustronnym wybieraniu najoptymalniejszych ruchów za pomocą algorytmu minmax kończy się zwycięstwem owiec. W przypadku, gdy jedna ze stron wykonuje losowe ruchy, znaczną przewagę ma strona korzystająca z algorytmu minmax.

5. Ostatnie kroki rozgrywki

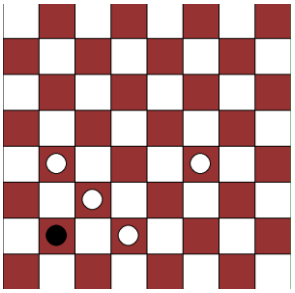
Przy losowej pozycji startowej wilka (7, 2), głębokości minmax = 5, obydwie strony korzystające z algorytmu minmax.



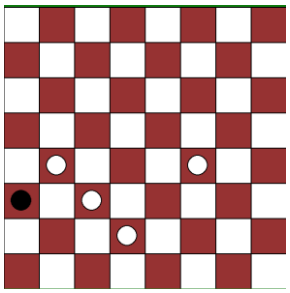
1



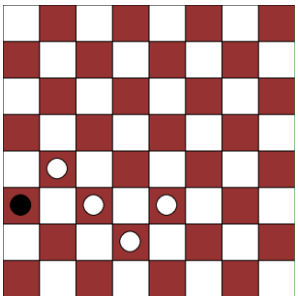
2



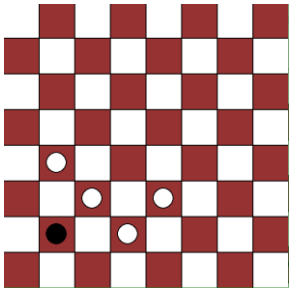
3



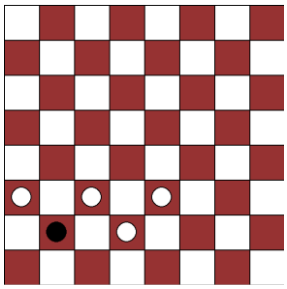
4



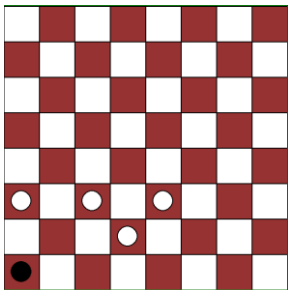
5



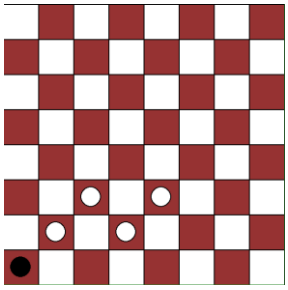
6



7



8



9