Wydział Informatyki i Telekomunikacji

Sztuczna Inteligencja i Inżynieria Wiedzy

Lista nr 2

Jakub Krupiński
255356

1. Wprowadzenie

- Reversi – gra o sumie stałej, główny problem tego ćwiczenia, strategiczna gra rozgrywana na planszy o wymiarach 8x8
- Algorytm Min-Max – algorytm optymalizacji zastosowany w rozwiązaniu
- Algorytm Alfa-Beta-Cięć – algorytm rozszerzający Min-Max, pozwalający przerywać przeszukiwanie drzewa i co za tym idzie skracać czas działania algorytmu za pomocą ewaluacji dodatkowych parametrów wprowadzonych do algorytmu Min-Max (alfa i beta).

2. Implementacja

- Plik main.py – plik służący do uruchamiania mojego programu i zestawu symulacji, a także do zapisywania wyników symulacji do odpowiednich plików .csv. Symulacje uruchamiane są dla różnych strategii oraz różnych głębokości przeszukiwania drzewa – zestawy głębokości które przyjąłem to:

  depths = [(1, 1), (1, 2), (1, 3),

  (2, 1), (2, 2), (2, 3),

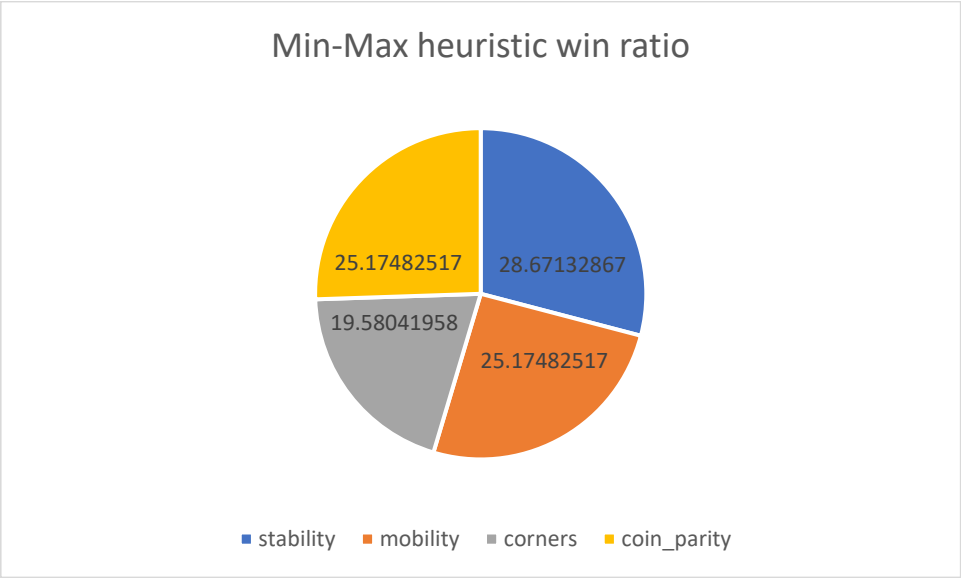
  (3, 1), (3, 2), (3, 3)]

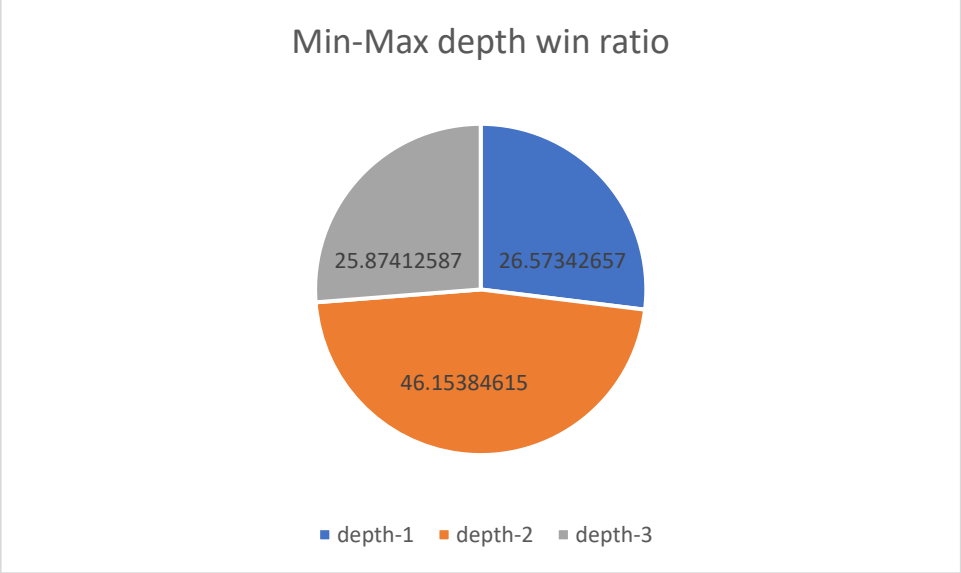
- Plik heuristics.py – plik zawierający implementacje funkcji heurystycznych, które determinują na podstawie jakiej strategii gracz rozegra swoją partię. Zaimplementowane przeze mnie strategie to:
  - stability – funkcja ewaluująca stabilność pionów na planszy: stabilność pionów to reprezentacja tego jak podatne są one na bycie przewróconym. Piony stabilne to takie, które nie mogą zostać przewrócone w żadnym momencie w grze w przyszłości.
  - corners – rogi planszy pełnią szczególnie ważną funkcję w przypakdu gry Reversi – strategia ta przypisuje wagi każdemu polu na planszy do gry, kładąc nacisk przede wszystkim na pola narożników (bardzo pozytywne wartości) oraz tzw pola ‘C’ oraz ‘X’ (bardzo negatywne wartości)
  - coin_parity – strategia ewaluująca różnice w ilości pionów, które zostały przejęte przez gracza. Jest to najprostsza ze strategii i nie jest ona optymalna w przypadku gry Reversi.
  - mobility – strategia priorytezująca ilość ruchów, które są możliwe do wykonania – kładzie ona nacisk na zmniejszanie ilości dozwolonych ruchów u przeciwnika i zwiększenie możliwych ruchów dla siebie.

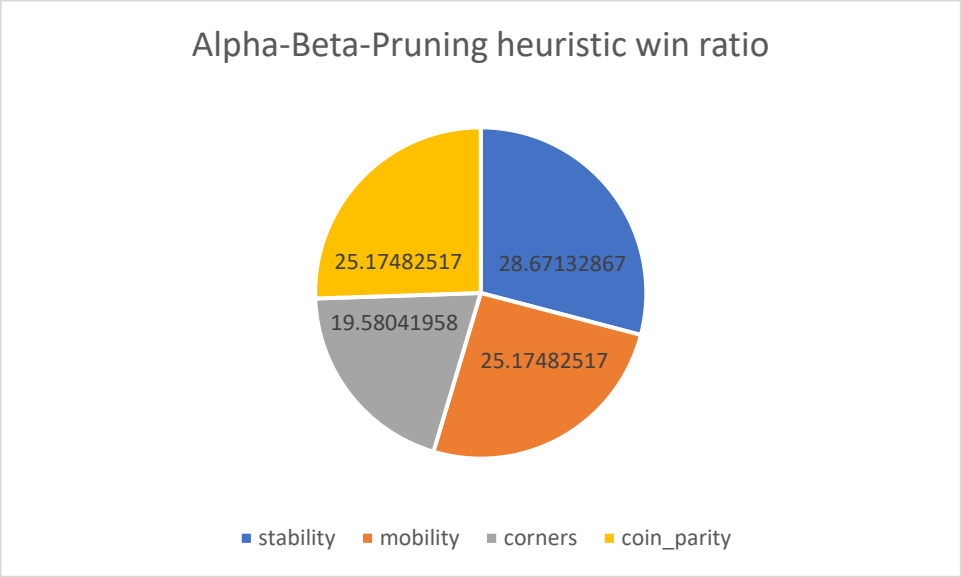
  Wartości ewaluujących wartości funkcji heurystycznych są w moim programie przeskalowane, by reprezentowały stan gry na skali wartości od -100 do 100.

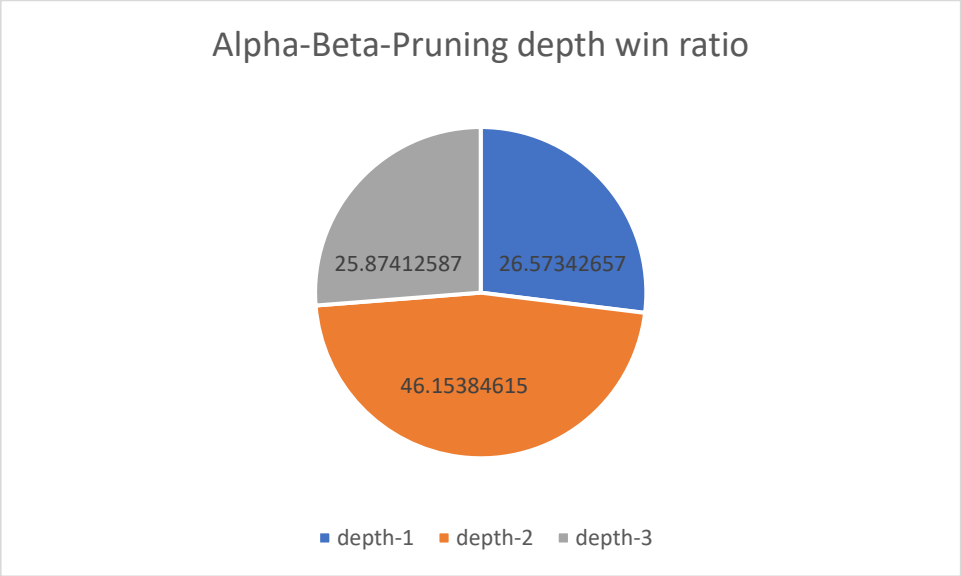
- Plik errors.py – plik zawierający kilka autorskich klas błędów, które pojawiają się w różnych miejscach kodu, które były pomocne w procesie debugowania programu.
- Klasa Board – klasa reprezentująca planszę do gry, zachowująca informacje na temat stanu pól na planszy oraz obecnego gracza (czyj ruch będzie wykonywany) i zawierająca znaczną część funkcjonalności i operacji, które są wykonywane na planszy do gry
- Klasa Reversi – klasa reprezentująca partię gry Reversi, zawierająca w sobie planszę do gry, informację o tym ile rund partii minęło oraz jaki jest status gry (w trakcie, zwycięstwo białych, czarnych, remis)
- Klasa Algorithms – klasa zawierająca implementacje algorytmów Min-Max oraz Alfa-Beta-Cięć, a także informacje o tym jaka funkcja heurystyczna będzie użyta do ewaluacji stanu gry oraz jaka jest maksymalna głębokość przeszukiwania drzewa.
- Klasa Player – klasa będąca pozostałością po poprzednich koncepcjach tego, jak uważałem że program będzie wyglądać – w obecnej formie służy jedynie do wywoływania algorytmów Min-Max i Alfa-Beta-Cięć
- Biblioteki: csv, time, copy, csv, itertools, numpy

3. Analiza wyników









4. Podsumowanie

Jak widać po sporządzonych przeze mnie wykresach kołowych, nie ma żadnych różnic w wynikach rozgrywanych meczy między algorytmami min-max i alfa-beta-cięć – to dobrze, jako że rozwiązania powinny zawsze być takie same. Współczynniki zwycięstw dla różnych heurystyk pokazują które zaimplementowane heurystyki zostały wykonane w sposób najbardziej optymalny – jak widać, najgorzej radzi sobie heurystyka kładąca nacisk na zdobywanie kluczowych pól na planszy, głównie narożników, co wynika zapewne z nieodpowiednich wartości w macierzy planszy, bądź z nieodpowiedniego przeskalowana otrzymywanych wyników względem pozostałych strategii. Nieoczekiwanym wynikiem jest również to, że głębokość przeszukiwania drzewa 2 okazała się bardziej skuteczna niż 3, co też ma na pewno swoje źródło w nieoptymalnej implementacji algorytmów. Średni czas symulacji partii (po odcięciu skrajnych 5% wartości brzegowych z obu stron zakresu) to około 61.86s dla algorytmu min-max, oraz około 18.94s dla algorytmu alfa-beta-cięć. Tak wysokie czasy spowodowane są przede wszystkim przez symulacje przeprowadzane na głębszych głębokościach przeszukiwania drzewa, pokazują jednak wyraźnie przewagę algorytmu alfa-beta nad min-max, jako że różnica średnich czasów wykonania jest bardzo wyraźna.

5. Materiały źródłowe

- Reversi - Wikipedia

- Reversi Rules - Reversi Documentation
- Inquiry-Based Introduction to Game Theory (nordstromjf.github.io)
- https://kartikkukreja.wordpress.com/2013/03/30/heuristic-function-for-reversiothello/
- Wiatr, Robert. *Implementacja Zaawansowanych Strategii Gry Reversi W C# Dla Platformy .NET. Projektowanie Modułów AI I Gra Z Użyciem Połączenia Sieciowego.* 2007.
- Ganter Mathias, and Jonas Klink. *A New Experience: O-Thell-Us – an AI Project*. p. 6, courses.cs.washington.edu/courses/cse573/04au/Project/mini1/O-Thell-Us/Othellus.pdf.