Projektowanie algorytmów i metody sztucznej inteligencji
Projekt 4 - Gra

Termin zajęć:
Wtorek 15:15

Autorstwo:
Rafał Rzewucki

Prowadzący: mgr inż. Marcin Ochman

18 maja 2020

1 Wstęp

Celem projektu było stworzenie gry dla dwóch lub więcej graczy warz z możliwością gry z wirtualnym graczem. Z podanych propozycji wybrano grę Kółko i Krzyżyk, gdzie gracz ma możliwość definiowania rozmiaru planszy, oraz ilości rund, które trzeba wygrać, aby wygrać całą grę. Całość gry z wirtualnym przeciwnikiem miała opierać się o algorytm *MinMax* oraz algorytm cięć *Alfa-Beta*.

2 Budowa programu

Program składa się z jednej głównej klasy, w której możemy wyróżnić kilka bardzo ważnych metod, które kierują działaniami programu:

- **showMainMenu** - wyświetla główne menu gry
- **playerVSplayer** oraz **playerVSCPU** - rozpoczynają grę odpowiednio gracz przeciwko graczowi i gracz przeciwko maszynie.
- **startGame** - rozpoczyna grę w odpowiednio wybranym trybie
- **getNextMove** - szuka następnego ruchu dla wirtualnego gracza z wykorzystaniem algorytmu *MinMax* oraz cięć *Alfa-Beta*

3 Opis algorytmów

3.1 Algorytm MinMax

Algorytm *MinMax* jest rekurencyjnym algorytmem, którego kolejne wywołania budują strukturę drzewa. Skupia się on na minimalizowaniu strat dla danego gracza i maksymalizowaniu szansy wygranej tego gracza. Rozważa on aktualny stan planszy i symulacyjnie przewiduje ruchy obydwóch graczy dla każdego wolnego miejsca na planszy. Na podstawie tego, czy dany gracz zyska na wybranym ruchu i jego następstwach wyciąga on wniosek, czy dany ruch się opłaca, czy nie.

3.2 Algorytm cięć Alfa-Beta

Jest to rozszerzenie algorytmu *MinMax* gdzie warunkiem zatrzymania budowania drzewa poprzez rekurencję jest znalezienie przynajmniej jednego ruchu, który czyni całą badaną opcję gorszą od poprzednich, przez co nie trzeba kończyć sprawdzania jej do samego końca, gdyż już na tym etapie jest ona nieopłacalna. Pozwala to znacząco zaoszczędzić czas szukania najlepszej opcji ruchu dla gracza.

4 Zapis algorytmu w pseudokodzie

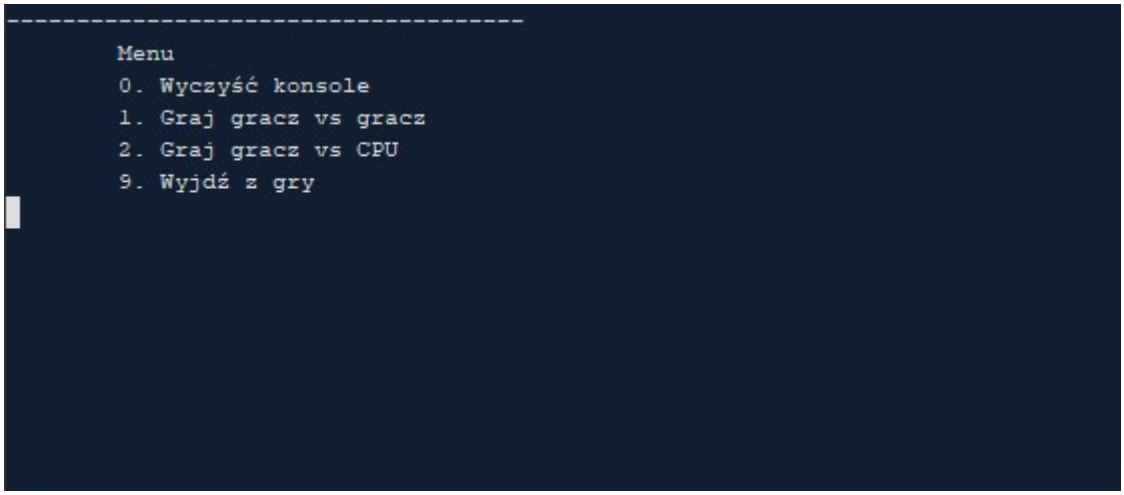
```
minmax(obecny_poziom_rekurencji, gracz, alfa, beta)
jesli rekurencja zbyt gleboka,
skonczyla sie mozliwosc ruchow,
lub ktorys z graczy wygral
-> zakoncz algorytm

jesli gracz jesz przeciwnikiem obecnego gracza
dla wszystkich ruchow
wykonaj ruch
znajdz minimalna wartosc minmax
cofnij ruch
```

```
w innym wypadku
dla wszystkich ruchow
wykonaj ruch
znajdz maksymalna wartosc minmax
cofnij ruch
zwroc wartosc minmax
```

5 Przebieg gry

5.1 Menu główne gry

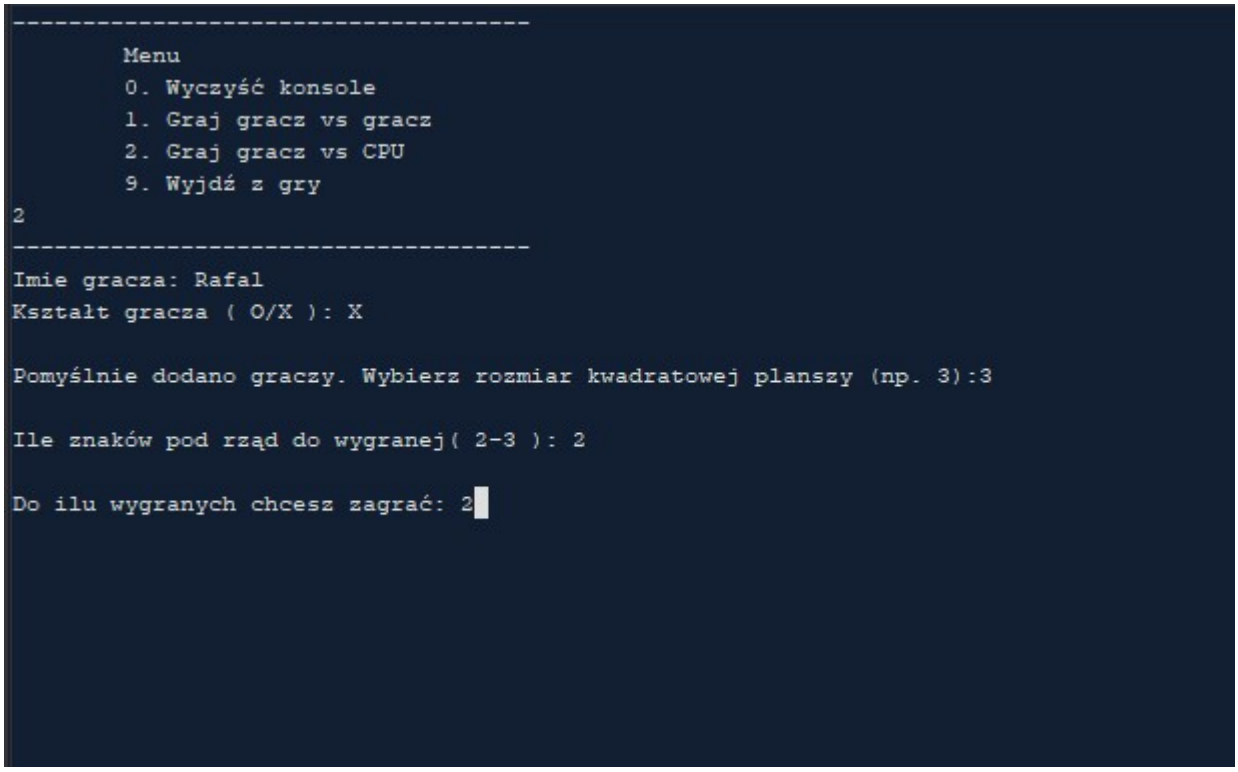
Program udostępnia nam prosty interfejs do poruszania się w grze.



Rysunek 1: Menu główne gry

5.2 Ustawienia początkowe gry

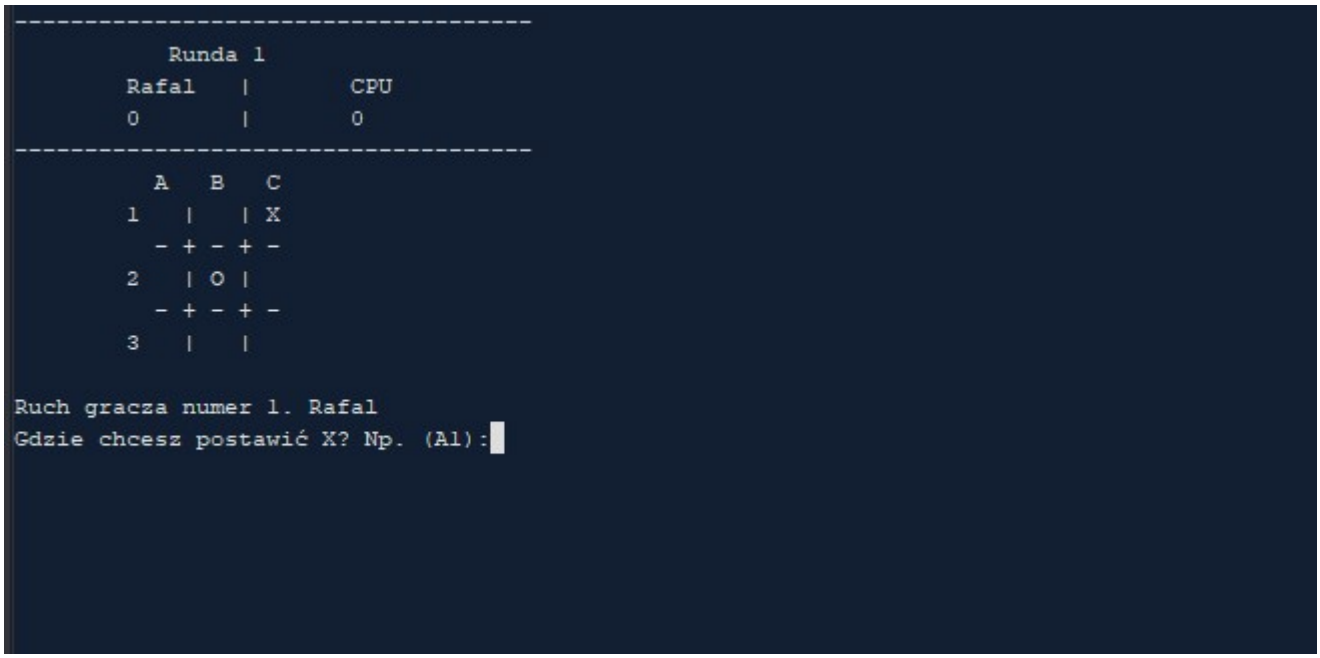
Po wybraniu opcji numer dwa, czyli gry z wirtualnym przeciwnikiem jesteśmy proszeni o kilka informacji konfiguracyjnych w tym o wybranie wielkości planszy, na której będzie toczyć się rozgrywka. Po zatwierdzeniu wprowadzonych informacji rozpoczyna się rozgrywka.



Rysunek 2: Wprowadzanie podstawowych informacji

5.3 Prowadzenie gry

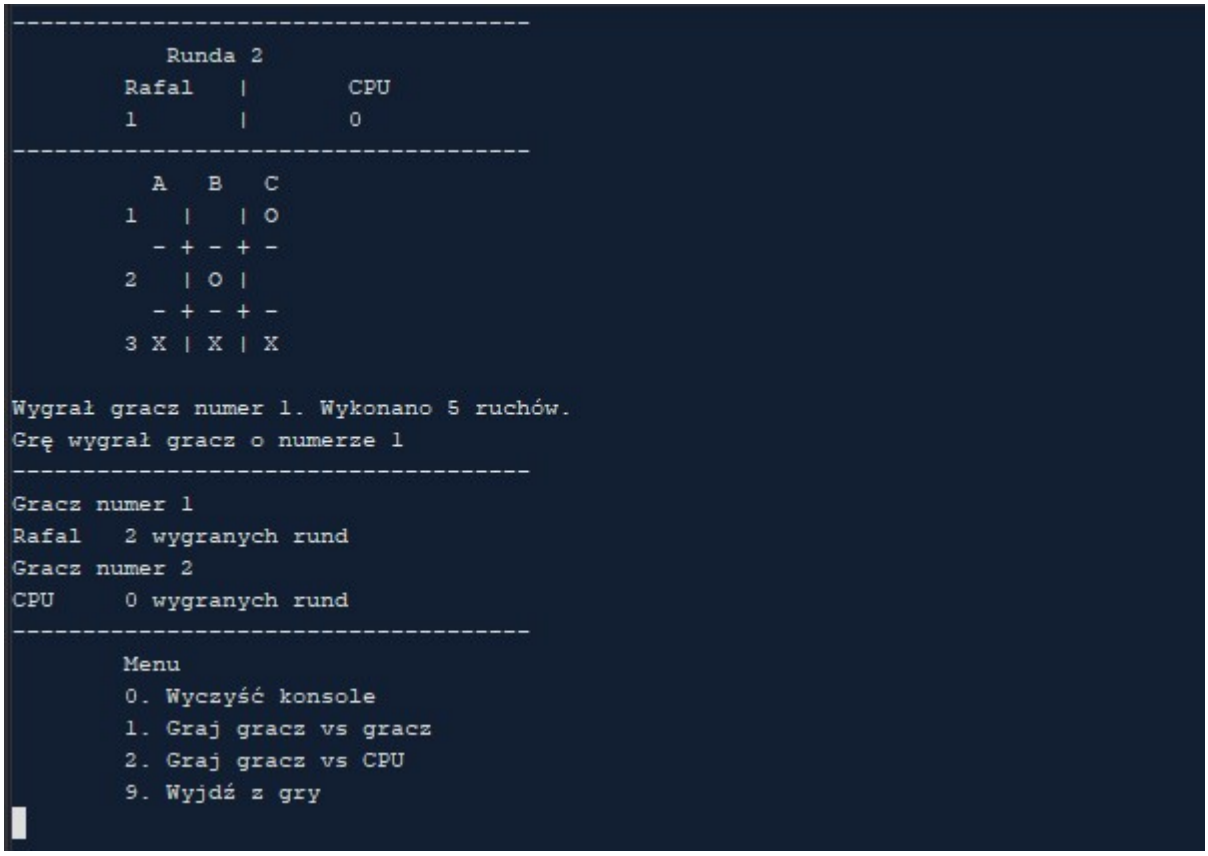
Gracze naprzemiennie proszeni są o wprowadzenie współrzędnych w formacie np *a3* aż do wygranej jednego z graczy lub wyczerpania możliwości ruchów.



Rysunek 3: Przebieg gry

5.4 Kończenie gry

Gra kończy się, gdy któryś z graczy wygra zadaną liczbę rund. Po zakończeniu gry wyświetlają się statystyki graczy oraz menu główne, aby można było rozpocząć grę ponownie, bądź ją zakończyć.



Rysunek 4: Koniec gry

6 Wnioski

Podczas tworzenia algorytmu napotkano następujące trudności:

- Zanim wprowadzono usprawnienia metoda wykonująca algorytm *MinMax* przyjmowała jako kolejny argument kopię tablicy znaków, aby wykonać kolejne ruchy. Zajmowało to ogromne ilości czasu i można było w bardzo prosty sposób temu zaradzić. Przed wykonaniem algorytmu wykonywano wymagany ruch, a po jego zakończeniu cofano go, więc końcowy stan planszy był taki jak początkowy. Ominięto tym samym konieczność kopiowania planszy do zmiennej tymczasowej.
- Początkowo nie planowano wprowadzania algorytmu Alfa-Beta, jednak podczas testów okazało się, że dla większych planszy (np. 5x5) algorytm *MinMax* wykonuje się stosunkowo długo. Po wprowadzeniu owego rozszerzenia czas ten skrócił się do granic akceptowalnych podczas gry.

Zastosowany algorytm *MinMax* wraz z algorytmem *Alfa-Beta*, który umożliwia ograniczenie rekurencji, a co za tym idzie ogranicza czas przeszukiwania drzewa jest bardzo prosty w budowie i dość wydajny dla tak prostych gier jak Kółko i Krzyżyk. Mimo to potrzebuje naprawdę dużo czasu na podjęcie decyzji o najlepszym ruchu dla większych rozmiarów plansz (6x6 i więcej). W przyszłości program ten można by rozszerzyć o bazę danych z predefiniowanymi ustawieniami kształtów na planszy, co pozwoliłoby maszynie dążyć do danych ustawień z obliczonymi wcześniej wagami przydatności. Czasami mogłoby to w ogóle wyeliminować potrzebę przeszukiwania drzewa.

Literatura

[1] Algorytm min-max - wikipedia
`https://pl.wikipedia.org/wiki/Algorytm_min-max`

[2] Algorytm alfa-beta - wikipedia
https://pl.wikipedia.org/wiki/Algorytm_alfa-beta

[3] Minimax Algorithm in Game Theory
https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

[4] Sztuczna inteligencja/SI Moduł 8 - Gry dwuosobowe
http://wazniak.mimuw.edu.pl/index.php?title=Sztuczna_inteligencja/SI_Modu%C5%82_8_-_Gry_dwuosobowe

[5] Coding Challenge 154: Tic Tac Toe AI with Minimax Algorithm
https://www.youtube.com/watch?v=trKjYdBASyQ&t=1367s