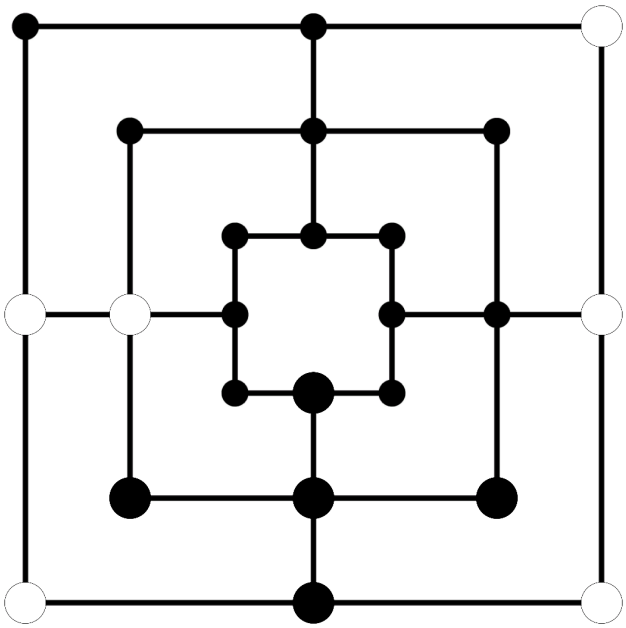
Algorytmy rozwiązywania gier o sumie zerowej

Szymon Woźniak, 235040

16.05.2019

1 Wstęp teoretyczny

1.1 Gra planszowa Młynek



Rysunek 1: Plansza do gry w młynek z kilkoma rozstawionymi pionkami

1.1.1 Skrót zasad

Młynek jest dwuosobową, turową, logiczną grą planszową. W rozpatrywanej wersji, na planszy znajdują się 24 rozmieszczone na 3 koncentrycznych kwadratach pola. Na każdym z tych pól gracze mogą umieszczać swoje pionki. Obaj gracze posiadają po 9 pionków do rozmieszczenia.
Gracze poprzez odpowiednie rozstawianie swoich pionków, mogą blokować lub zbijać pionki przeciwników. Bicie następuje gdy jeden z graczy ustawi 3 swoje pionki w linię. Może wtedy wybrać jeden z pionków przeciwnika, który zostanie usunięty z planszy.

1.1.2 Fazy rozgrywki

Pojedyncza partia młynka składa się z trzech faz.

- rozstawianie pionków,
- przesuwanie pionków,

- ”latanie”.

W pierwszej fazie rozgrywki gracze na zmianę umieszczają po jednym z dostępnych 9 pionków na wolnych polach planszy. Jeżeli któremuś z nich uda się ustawić młynek, może usunąć z planszy wybrany pionek przeciwnika. W rozpatrywanej wersji gry, jest to jedyny moment kiedy gracz może ustawić podwójny młynek.
W drugiej, podstawowej fazie rozgrywki gracze na zmianę swoje pionki. Mogą wybrać dowolne puste pole połączone linią z polem na którym znajduje się aktualnie pionek.
Trzecia faza następuje dla każdego gracza osobno, kiedy pozostaną mu tylko 3 pionki. Może on wtedy w swojej turze przemieszczać pionki na dowolne puste miejsca na planszy (stąd angielska nazwa *flying*).

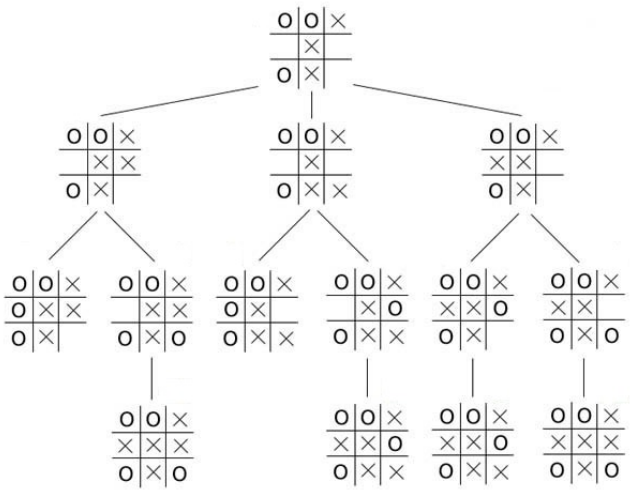
1.1.3 Cel rozgrywki

Celem rozgrywki jest doprowadzenie do sytuacji, w której przeciwnikowi pozostaną tylko 2 pionki, lub nie posiada on żadnego możliwego ruchu.

1.1.4 Dodatkowe modyfikacje

W rozpatrywanej wersji gry stosuje się zasadę, że pionka nie można przesunąć na pole, z którego został przesunięty wcześniej. Tym samym gracze zmuszeni są budować młynki, zamiast korzystać z już istniejących.
W niektórych wersjach gry stosuje się też zasadę, że gracze mogą zbijać pionki przeciwników tylko pod warunkiem, że nie stoją w młynku. W tej pracy zasada ta nie została zastosowana.

1.2 Drzewo gry



Rysunek 2: Przykładowy fragment drzewa dla gry kółko i krzyżyk

Drzewo gry jest grafem skierowanym, w którym każdy z węzłów reprezentuje stan rozgrywki w danym momencie. Z każdego stanu, w którym gra jeszcze się nie skończyła, można wygenerować zbiór następnych stanów gry reprezentujących różne możliwe decyzje aktualnie ruszającego się gracza. Następnie z każdego z tych stanów można wygenerować ruchy przeciwnika itd.
Jak widać już na przykładzie kółka i krzyżyk, drzewo to rozrasta się bardzo szybko i w większości gier nie jest ono możliwe do zbudowanie i przejrzenia w całości.

1.3 Badane algorytmy

Rozpatrywane algorytmy przeglądają fragmenty operują na wspomnianym w sekcji 1.2 drzewie gry. Przeglądając jego fragment, estymują jakość możliwych do podjęcia decyzji, oceniając stan rozgrywki kilka ruchów dalej. Oba korzystają w tym celu z pewnej heurystycznej funkcji, oznaczanej dalej jako *heuristic*, do statycznej ewaluacji stanu rozgrywki.

1.3.1 Algorytm min-max

Algorytm min-max przegląda drzewo gry do pewnej zadanej głębokości *depth*, na zmianę wybierając odpowiednio stan oceniany jako najlepszy i jako najgorszy przez funkcję *heuristic*. Reprezentuje to podejmowanie możliwie najlepszych decyzji zarówno przez siebie jak i przez przeciwnika. Jego działanie przedstawia poniższy pseudokod.

Algorithm 1 Algorytm Min-Max

```
1: function MinMax(state, depth, maximizing)
2:     if game finished in state or depth = 0 then
3:         return heuristic(state)
4:     end if
5:     if maximizing then
6:         maxEval ← −∞
7:         childStates ← getAllNextStates(state)
8:         for child in childStates do
9:             eval ← MinMax(child, depth − 1, false)
10:            maxEval ← max(maxEval, eval)
11:        end for
12:        return maxEval
13:    else
14:        minEval ← ∞
15:        childStates ← getAllNextStates(state)
16:        for child in childStates do
17:            eval ← MinMax(child, depth − 1, true)
18:            minEval ← min(minEval, eval)
19:        end for
20:        return minEval
21:    end if
22: end function
```

1.3.2 Algorytm alfa-beta cięć

Algorytm alfa-beta cięć jest usprawnieniem algorytmu min-max. Przeglądając kolejne stany wgłąb drzewa podejmuje on decyzje czy rozpatrywana gałąź jest warta rozwijania. Jeżeli w dowolnym momencie nie istnieje możliwość znalezienia lepszego stanu na pewnym poziomie drzewa, algorytm nie przegląda kolejnych stanów. Pozwala to zaoszczędzić czas pracy procesora i potencjalnie przeglądać drzewo na większą głębokość w takim samym czasie jak algorytm min-max dla mniejszych głębokości. Jego działanie zostało przedstawione na poniższym pseudokodzie.

Algorithm 2 Algorytm Alfa-Beta

```
1: function AlfaBeta(state, depth, maximizing, α, β)
2:     if game finished in state or depth = 0 then
3:         return heuristic(state)
4:     end if
5:     if maximizing then
6:         maxEval ← −∞
7:         childStates ← getAllNextStates(state)
8:         for child in childStates do
9:             eval ← AlfaBeta(child, depth − 1, false, α, β)
10:            maxEval ← max(maxEval, eval)
11:            α ← max(α, eval)
12:            if α ⩾ β then
13:                break
14:            end if
15:        end for
16:        return maxEval
17:    else
18:        minEval ← ∞
19:        childStates ← getAllNextStates(state)
20:        for child in childStates do
21:            eval ← AlfaBeta(child, depth − 1, true, α, β)
22:            minEval ← min(minEval, eval)
23:            β ← min(β, eval)
24:            if α ⩾ β then
25:                break
26:            end if
27:        end for
28:        return minEval
29:    end if
30: end function
```

2 Plan pracy

W pierwszej kolejności gra młynek zostanie zaimplementowana w wybranym języku programowania i środowisku programistycznym. Implementacja będzie zawierać silnik gry pozwalający na grę zarówno graczy ludzkich jak i kierowanych przez algorytmy sztucznej inteligencji. Będzie również posiadać interfejs graficzny ułatwiający rozgrywkę i umożliwiający obserwowanie rozgrywek SI graczom ludzkim. W silniku gry zostaną również zaimplementowane algorytmy min-max i alfa-beta oraz różne heurystyki oceny stanu planszy.
Następnie przeprowadzone zostaną badania zaimplementownych rozwiązań.