Algorytm min-max w zastosowaniu do gry w kółko i krzyżyk

1 Opis algorytmu min-max

Algorytm min-max to inaczej metoda minimalizowania maksymalnych strat. Służy ona do wybrania optymalnego ruchu w danym momencie. Danych jest dwóch graczy. Jeden z nich chce zmaksymalizować wartość stanu gry, podczas gdy drugi będzie dążył do jej zminimalizowania. Posiadając funkcję oceniającą wartość stanu gry w danym momencie, gracz konstruuje drzewo wszytskich możliwych stanów, a następnie dokonuje optymalnego dla niego wyboru. Gracz maksymalizujący zawsze wybiera ruch prowadzący do większej wartości końcowej, a gracz minimalizujący - przeciwnie.

2 Implementacja

Program składa się z pliku *main.py* oraz modułu *minmax*, w którym zaimplementowany jest opisany algorytm.

2.1 minmax

Moduł minmax składa się z następujących funkcji:

Funkcja *is_available* służy do sprawdzenia, czy dana pozycja jest już zajęta

```
def is_available(grid, row, column):
    return grid[row][column] == 0
```

Funkcja *is_winner* sprawdza, czy gra w danym momencie nie zakończyła się zwycięstwem jednego z graczy

```
def is_winner(grid, player):
    for i in range(3):
        if grid[i][0] == player and grid[i][1] == player and grid[i
    ][2] == player:
            return True
        if grid[0][i] == player and grid[1][i] == player and grid
    [2][i] == player:
            return True
```

```
    if grid[0][0] == player and grid[1][1] == player and grid[2][2]
     == player:
        return True
    if grid[2][0] == player and grid[1][1] == player and grid[0][2]
     == player:
        return True
    return False
```

Funkcja *is_full* sprawdza czy plansza jest zapełniona, a co za tym idzie (w braku zwycięstwa jednego z graczy), gra zakończyła się remisem

```
def is_full(grid):
    for i in range(3):
        for j in range(3):
            if grid[i][j] == 0:
                return False
    return True
```

Funkcja *minmax*, to opisany algorytm wyszukiwania optymalnego wyboru

```
def minmax(grid, is_maximizing):
    if is_winner(grid, 1):
        return -1
    if is_winner(grid, 2):
        return 1
    if is_full(grid):
        return 0

    if is_maximizing:
        best = -inf
        for row in range(3):
            for col in range(3):
                if is_available(grid, row, col):
                    grid[row][col] = 2
                    best = max(best, minmax(grid, False))
                    grid[row][col] = 0
        return best

    if not is_maximizing:
        best = inf
        for row in range(3):
            for col in range(3):
                if is_available(grid, row, col):
                    grid[row][col] = 1
                    best = min(best, minmax(grid, True))
                    grid[row][col] = 0
        return best
```

Pierwsze instrukcje warunkowe służą do oceny stanu gry w danym momencie. W przypadku zwycięstwa gracza 1 (minimalizującego), funkcja zwraca wartość -1. Z racji iż jest to najmniejsza możliwa wartość, gracz minimalizujący będzie dążył do takiego stanu gry, podczas gdy gracz maksymalizujący będzie starał się temu zapobiec. Sytuacja analogicznie wygląda, kiedy to gracz 2 (maksymalizujący) wygrywa. Gdy potyczka kończy się remisem, funkcja zwraca 0, co w

wypadku braku możliwości zwycięstwa staje się optymalnym wyborem.
W następnym kroku, zależnie od argumentu *is_maximizing*, symulowany jest ruch gracza (maksymalizującego w przypadku gdy $is\_maximizing = True$, lub minimalizującego, gdy $is\_maximizing = False$). W każdej wolnej pozycji stawiana jest figura gracza, a następnie wybierana maksymalna lub minimalna wartość między aktualnie najlepszym ruchem a rekurencyjnie wywoływaną funkcją *minmax*, tym razem z przeciwnym parametrem *is_maximizing* (symulowany będze ruch drugiego z graczy).

Ostatnią funkcją w module *minmax* jest *get_move*. Zapoczątkowywuje ona symulowanie możliwych scenariuszy przebiegu gry.

```
def get_move(grid):
    best = -inf
    move = ()
    for row in range(3):
        for col in range(3):
            if is_available(grid, row, col):
                grid[row][col] = 2
                score = minmax(grid, False)
                grid[row][col] = 0
                if score > best:
                    best = score
                    move = (row, col)
    return move
```

W przeciwieństwie do funkcji *minmax*, zwraca ona krotkę, reprezentującą wybrany, optymalny ruch komputera.

2.2 main

W pliku głównym inicjowany jest cały interfejs graficzny. Do jego stworzenia skorzytsałem z biblioteki *pygame*.

```
import pygame as pg
from sys import exit
from numpy import zeros
from minmax import get_move, is_available

# Constant
WIDTH = 900
HEIGHT = 900
BACKGROUND_COLOR = (98, 114, 164)
GRID_COLOR = (68, 71, 90)
CIRCLE_COLOR = (182, 185, 200)
CROSS_COLOR = (55, 57, 73)
LINE_WIDTH = 15
RADIUS = 100
SPACE_SIZE = 300

pg.init()
screen = pg.display.set_mode((WIDTH, HEIGHT))
pg.display.set_caption('Tic Tac Toe')
screen.fill(BACKGROUND_COLOR)
```

```
icon = pg.image.load("icon.png")
pg.display.set_icon(icon)

# Drawing grid
pg.draw.line(screen, GRID_COLOR, (0, SPACE_SIZE), (3 * SPACE_SIZE,
    SPACE_SIZE), LINE_WIDTH)
pg.draw.line(screen, GRID_COLOR, (0, 2 * SPACE_SIZE), (3 *
    SPACE_SIZE, 2 * SPACE_SIZE), LINE_WIDTH)
pg.draw.line(screen, GRID_COLOR, (SPACE_SIZE, 0), (SPACE_SIZE, 3 *
    SPACE_SIZE), LINE_WIDTH)
pg.draw.line(screen, GRID_COLOR, (2 * SPACE_SIZE, 0), (2 *
    SPACE_SIZE, 3 * SPACE_SIZE), LINE_WIDTH)

grid = zeros((3, 3))
```

Plansza gry w programie jest reprezentowana jako tablica dwuwymiarowa 3×3. Jeżeli w danym polu występuje 0, to dana pozycja jest wolna. Gracz (człowiek) reprezentowany jest przez liczbę 1, a komputer przez 2.

Plik *main.py* zawiera również funkcje, które później wywoływane są w pętli głównej. Procedura *draw_figure* jako argumenty przyjmuje numer wiersza, kolumny oraz id gracza. Następnie na ich podstawie wylicza miejsce na planszy i wpisuje w nie odpowiednią figurę.

```
def draw_figure(row, column, player):
    grid[row][column] = player
    if player == 1:     # Circle
        center = (int(column * SPACE_SIZE + SPACE_SIZE / 2), int(
    row * SPACE_SIZE + SPACE_SIZE / 2))
        pg.draw.circle(screen, CIRCLE_COLOR, center, RADIUS,
    LINE_WIDTH)
    else:               # Cross
        top_left = (column * SPACE_SIZE + 50, row * SPACE_SIZE +
    50)
        bottom_right = (column * SPACE_SIZE + SPACE_SIZE - 50, int(
    row * SPACE_SIZE + SPACE_SIZE - 50))
        pg.draw.line(screen, CROSS_COLOR, top_left, bottom_right,
    20)
        bottom_left = (column * SPACE_SIZE + 50, row * SPACE_SIZE +
     SPACE_SIZE - 50)
        top_right = (column * SPACE_SIZE + SPACE_SIZE - 50, int(row
     * SPACE_SIZE + 50))
        pg.draw.line(screen, CROSS_COLOR, bottom_left, top_right,
    20)
```

W podobny sposób działa procedura *draw_winning_line*. Przyjmuje ona jako argumenty numer wiersz oraz kolumny. Następnie w wierszu/kolumnie o danym numerze rysowana jest linia. Jeśli wyrysowana ma być linia pionowa lub pozioma, jeden z argumentów jest równy *None*. Przykładowo dla argumentów $row = 1$ i $col = None$, zostanie wyrysowana linia w drugim (*row* liczone od 0) wierszu. W wypadku gdy wyrysowana ma zostać przekątna, tworzona jest krotka, zawierająca współrzędne y początku i końca lini.

```
def draw_winning_line(row, col):
    if col is None:     # vertical line
```

```
        pg.draw.line(screen, GRID_COLOR, (20, row * SPACE_SIZE +
    SPACE_SIZE / 2), (3 * SPACE_SIZE - 20, row * SPACE_SIZE +
    SPACE_SIZE / 2), LINE_WIDTH)
    elif row is None:   # horizontal line
        pg.draw.line(screen, GRID_COLOR, (col * SPACE_SIZE +
    SPACE_SIZE / 2, 20), (col * SPACE_SIZE + SPACE_SIZE / 2, 3 *
    SPACE_SIZE - 20), LINE_WIDTH)
    else:
        if row == 0:
            start_end = (50, 3 * SPACE_SIZE - 50)
        else:
            start_end = (3 * SPACE_SIZE - 50, 50)
        pg.draw.line(screen, GRID_COLOR, (50, start_end[0]), (3 *
    SPACE_SIZE - 50, start_end[1]), 20)
```

Funkcja *check_win* sprawdza zwycięstwo gracza, którego id zostało podane jako argument funkcji. Jeśli tak, wywoływana jest funkcja *draw_winning_line*, a następnie zwrócone *True*, co kończy rozgrywkę.

```
def check_win(player):
    for i in range(3):
        if grid[i][0] == player and grid[i][1] == player and grid[i
    ][2] == player:
            draw_winning_line(i, None)
            return True
        if grid[0][i] == player and grid[1][i] == player and grid
    [2][i] == player:
            draw_winning_line(None, i)
            return True
    if grid[0][0] == player and grid[1][1] == player and grid[2][2]
     == player:
        draw_winning_line(0, 0)
        return True
    if grid[2][0] == player and grid[1][1] == player and grid[0][2]
     == player:
        draw_winning_line(2, 0)
        return True
    return False
```

Funkcja *game_over* sprawdza zwycięstwo któregoś z graczy (wywołanie funkcji *check_win*) lub ewentualny remis.

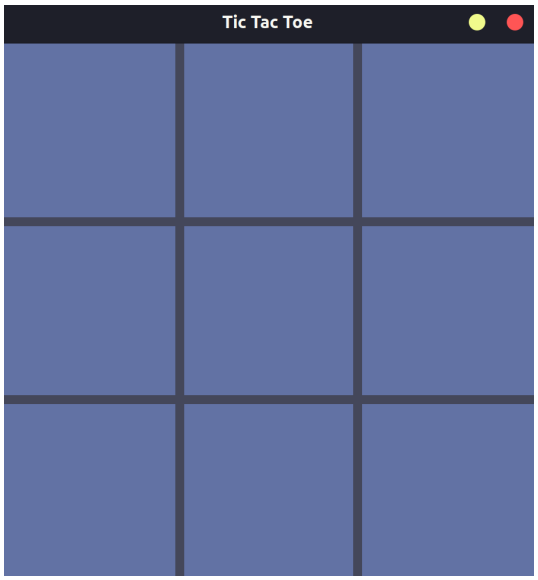
```
def game_over():
    if check_win(1) or check_win(2):
        return True
    for i in range(3):
        for j in range(3):
            if grid[i][j] == 0:
                return False
    return True
```

Pętla główna wygląda następująco:

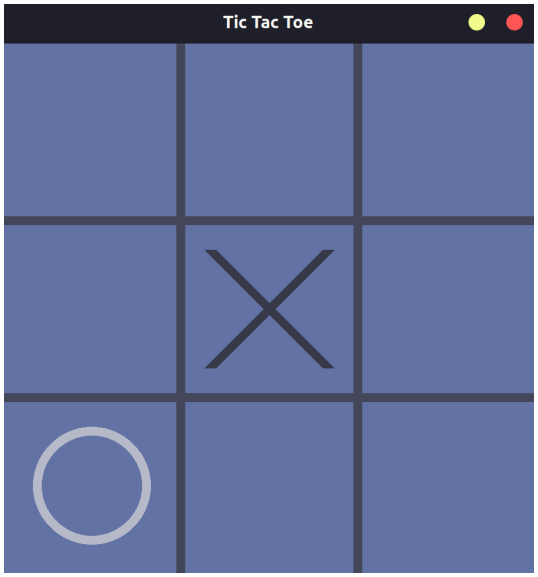
```
your_turn = True
# Game loop
while True:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            exit()
        if not your_turn and not game_over():
            choice = get_move(grid)
            draw_figure(choice[0], choice[1], 2)
            game_over()
            your_turn = True
        if event.type == pg.MOUSEBUTTONDOWN and your_turn and not
    game_over():
            clicked_column = int(event.pos[0] / SPACE_SIZE)
            clicked_row = int(event.pos[1] / SPACE_SIZE)
            if is_available(grid, clicked_row, clicked_column):
                draw_figure(clicked_row, clicked_column, 1)
                game_over()
                your_turn = False

    pg.display.update()
```

3 Uruchomienie i rozgrywka

Aby uruchomić program należy wykonać komende **make run**
Aby utworzyć plik exe należy wykonać komende **make exe**
Po uruchomieniu wyświetlona zostanie plansza



Gra niezmiennie zaczyna się od ruchu gracza (człowieka). Po dokonaniu wyboru komputer automatycznie znajdzie optymalny ruch i zakreśli odpowiednie pole



Po zakończenia rozgrywki okno zostanie widoczne, do czasu jego zamknięcia