Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 2 з дисципліни
«ПІІС»

Виконав(ла) _____ *ІТ-02 Макаров И.С.*
(шифр, прізвище, ім'я, по батькові)

Перевірив _____
(прізвище, ім'я, по батькові)

Київ 2021

ЗМІСТ

МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи - вивчити основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Ознайомитися з підходами до програмування алгоритмів штучного інтелекту в іграх з повною інформацією.

ВИКОНАННЯ

Для прикладу, на якому я буду реалізовувати алгоритм мінімаксу, я взяв гру «Гомоку». Чому саме цю гру? Колись давно, ще на початку першого курсу я було написав якийсь набросок цієї гри, однак для двох гравців.

В рамках лабораторної роботи, я переписав клієнт (бо два роки тому я навіть уявлення про чистий код не знав, та код додатку був нечитабельний) та реалізував примітивний ШІ опонент для гри. Написаний ШІ базується на алгоритмі Minimax, та з ціллю оптимізації алгоритму я написав модифікацію до мінімаксу під назвою Alpha Beta відсікання. **Надалі мій ШІ також може називатись агентом, супротивником, опонентом.**

Зупинятись на тому як функціонує сам клієнт для гри я не буду, в нас всеж пара алгоритмів, тому перейдемо одразу до реалізації супротивника.

Метод, що відповідає за хід агента виглядає так, як зображено знизу, приймаючи на вхід стан дошки, що є зараз та колір того хто ходить (завжди Black, бо за White граємо ми). Повертає метод нову дошку, що замінить поточну.

kinfi4
```
def make_move(self, current_board, color_making_move):
    depth = self.difficulties_to_depth[self.difficulty]

    _, new_board = self.minimax(current_board, depth, color_making_move, float('-inf'), float('inf'))
    return new_board
```

Не знаю, що тут можна описати, єдине, що як видно глубина нашого мінімаксу залежить від складності, чим більше складність тим глибше буде дивитись наш опонент.

Давайте подивимось на сам мінімакс:

```python
def minimax(self, board: Board, depth, color_making_move, i_alpha, i_beta):
    previous_color = CheckerType.BLACK if color_making_move == CheckerType.WHITE else CheckerType.WHITE
    if depth == 0 or board.is_game_winner(previous_color):
        return board.evaluate_board(whos_move=previous_color), board

    best_board = None

    if color_making_move == CheckerType.WHITE:
        max_eval = float('-inf')
        for new_board in self.get_all_possible_children_boards(board, color_making_move):
            new_evaluation, _ = self.minimax(new_board, depth - 1, CheckerType.BLACK, i_alpha, i_beta)

            if new_evaluation > max_eval:
                max_eval = new_evaluation
                best_board = new_board

            i_alpha = max(new_evaluation, i_alpha)
            if i_beta <= i_alpha:
                break

        return max_eval, best_board

    else:
        min_eval = float('inf')
        for new_board in self.get_all_possible_children_boards(board, color_making_move):
            new_evaluation, _ = self.minimax(new_board, depth - 1, CheckerType.WHITE, i_alpha, i_beta)

            if new_evaluation < min_eval:
                min_eval = new_evaluation
                best_board = new_board

            i_beta = min(i_beta, new_evaluation)
            if i_beta <= i_alpha:
                break

        return min_eval, best_board
```

нічого складного, та незвичайного, просто мінімакс з альфа-бета відсіканнями, не бачу сенсу особливо зупинятись на алгоритмі, давайте подивимось як виглядає функція оцінки стану дошки.

Забігаючи на перед, також слід сказати, що в моєму алгоритмі задача White максимізувати оцінку, задача Black мінімізувати.

Виклик функції `board.evaluate_board(whos_move=previous_color)` відбувається коли алгоритм досяг максимальної глибини, або хтось вже виграв гру. Ось код цієї євристики.

```python
def evaluate_board(self, whos_move: CheckerType) -> int:
    white_points, black_points = 0, 0

    # checking columns
    for column in self.board.transpose():
        str_col = ''.join(column)
        w_p, b_p = self._get_heuristic_points_of_sequence(str_col, whos_move)
        white_points += w_p
        black_points += b_p

    # checking rows
    for row in self.board:
        str_row = ''.join(row)
        w_p, b_p = self._get_heuristic_points_of_sequence(str_row, whos_move)
        white_points += w_p
        black_points += b_p

    # checking right diagonal
    for diagonal in [self.board.diagonal(offset=-x) for x in np.arange(-BOARD_SIZE + 1, BOARD_SIZE)]:
        str_diag = ''.join(diagonal)
        w_p, b_p = self._get_heuristic_points_of_sequence(str_diag, whos_move)
        white_points += w_p
        black_points += b_p

    # checking left diagonal
    for diagonal in [self.board[::-1].diagonal(offset=-x) for x in np.arange(-BOARD_SIZE + 1, BOARD_SIZE)]:
        str_diag = ''.join(diagonal)
        w_p, b_p = self._get_heuristic_points_of_sequence(str_diag, whos_move)
        white_points += w_p
        black_points += b_p

    # print(f'Board, {self.board} evaluation = {white_points + black_points}')
    return white_points + black_points
```

Одразу може не до кіньця зрозуміло, що тут відбувається, тож давайте поясню. Ми пробігаємо по кожній колонці, кожному рядку, кожній діагоналі нашої зводячи задачу до оцінки на скільки “добре” для кожного гравця виглядає кожна послідовність. І сумуючи всі оцінки отримаємо два результати white_pointa (додатній) та black_points (від’ємний), повернувши їх суму отримаємо оцінку дошки.

```python
@staticmethod
def _get_heuristic_points_of_sequence(string, whos_move: CheckerType):
    white_points, black_points = 0, 0
    white_enemy_coef = 2 if whos_move == CheckerType.WHITE else 1
    black_enemy_coef = 2 if whos_move == CheckerType.BLACK else 1

    w, b, e = CheckerType.WHITE, CheckerType.BLACK, CheckerType.EMPTY
    if ''.join([w, w, w, w, w]) in string:
        white_points = 30_000 * white_enemy_coef
    elif ''.join([e, w, w, w, w, e]) in string:
        white_points = 9_000
    elif ''.join([e, w, w, w, e]) in string:
        white_points = 5_000
    elif ''.join([e, w, w, w]) in string:
        white_points = 400
    elif ''.join([w, w, w, e]) in string:
        white_points = 400
    elif ''.join([e, w, w, e]) in string:
        white_points = 50
    elif ''.join([w, w, e]) in string:
        white_points = 20
    elif ''.join([e, w, w]) in string:
        white_points = 20

    if ''.join([b, b, b, b, b]) in string:
        black_points = -25_000 * black_enemy_coef
    elif ''.join([e, b, b, b, b, e]) in string:
        black_points = -8_000
    elif ''.join([e, b, b, b, e]) in string:
        black_points = -3_000
    elif ''.join([e, b, b, b]) in string:
        black_points = -200
    elif ''.join([b, b, b, e]) in string:
        black_points = -200
    elif ''.join([e, b, b, e]) in string:
        black_points = -100
    elif ''.join([b, b, e]) in string:
        black_points = -50
    elif ''.join([e, b, b]) in string:
        black_points = -50

    return white_points, black_points
```

Залишилась лише одна функція, а саме оцінка рядку.

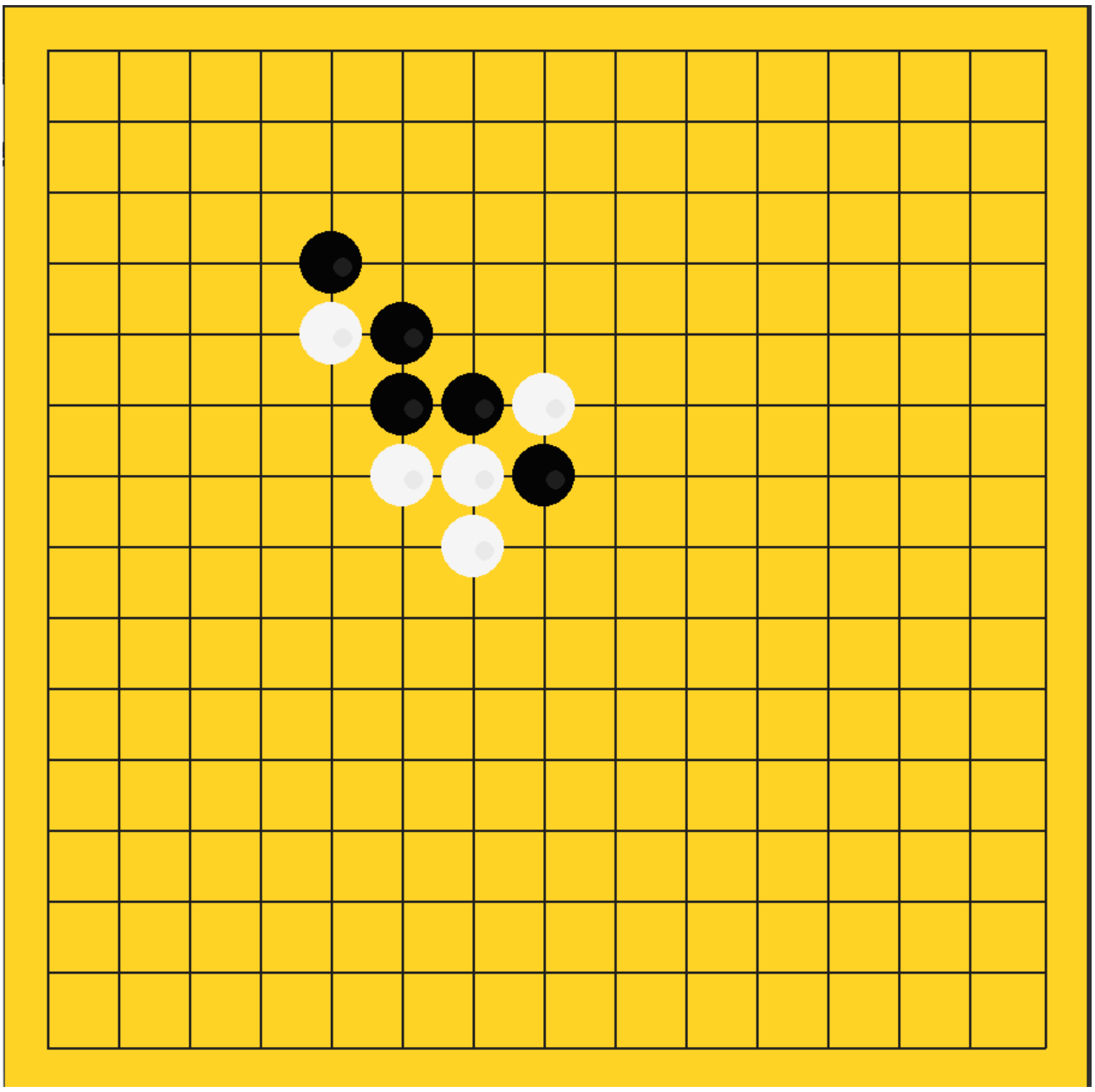
Вона не влазила в екран, тож довелось зменшити шрифт.

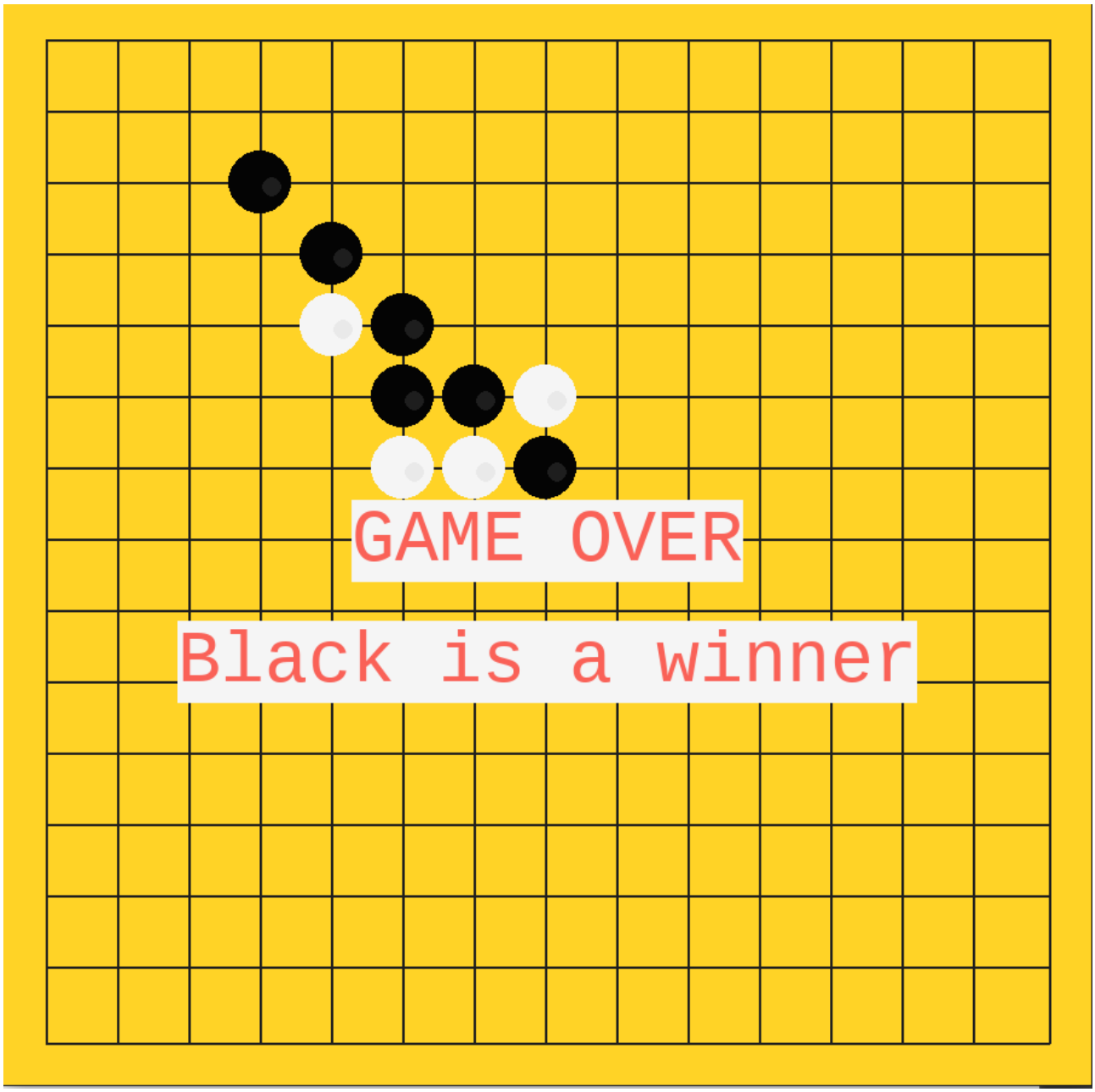
Тут довго описувати, що відбувається, я краще проговорю це на захисті більш детально, скажу лише, що ми можемо змінювати тактику агента балансуючи ці цифри. В той час, як не правильна комбіанція цих оцінок може призвести до не оптимальних рішень агента.

Такс, я тут зрозумів, що мені ще треба було зробити просто minimax, без оптимізації альфа-бета, тому ось.

```
new *
def plain_minimax(self, board: Board, depth, color_making_move):
    previous_color = CheckerType.BLACK if color_making_move == CheckerType.WHITE else CheckerType.WHITE
    if depth == 0 or board.is_game_winner(previous_color):
        return board.evaluate_board(whos_move=previous_color), board

    best_board = None

    if color_making_move == CheckerType.WHITE:
        max_eval = float('-inf')
        for new_board in self.get_all_possible_children_boards(board, color_making_move):
            new_evaluation, _ = self.plain_minimax(new_board, depth - 1, CheckerType.BLACK)

            if new_evaluation > max_eval:
                max_eval = new_evaluation
                best_board = new_board

        return max_eval, best_board

    else:
        min_eval = float('inf')
        for new_board in self.get_all_possible_children_boards(board, color_making_move):
            new_evaluation, _ = self.plain_minimax(new_board, depth - 1, CheckerType.WHITE)

            if new_evaluation < min_eval:
                min_eval = new_evaluation
                best_board = new_board

        return min_eval, best_board
```

ПРИКЛАД РОБОТИ





Висновок: багато писати тут не бачу сенсу, осьновун частину висновку, як мені здається я написав на початку у вступі. Скажу лише, що робота виявилась цікавою, алгоритм мінімаксу хоча і працює не швидко, однак в більшості випадків повертає досить таки оптимальні рішення. Не знаю, чи то я такий поганий гравець, чи що, але виграти я в нього майже не можу.