AI Design For Reversi

Project 1 Report of
CS303A Artificial Intelligence

11812804
董正

Department of Computer Science and Engineering

Oct. 2020

Content

1 Preliminaries

1.1 Problem Description

Reversi is a relatively simple board game. Players take turns placing pieces on the board with their assigned color facing up. During a play, any piece of the opponent's color that is in a straight line and bounded by the piece just placed and another piece of the current player's color are turned over to the current player's color. The object of the game is to have the majority of pieces turned to display your color when the last playable empty square is filled.[1]

Our goal is to design an AI to play Reversi and compete with other AIs through kinds of **Game Algorithms**.

This project is written in *Python 3.8.5* and the testing platforms are my PC and the online server of the project.

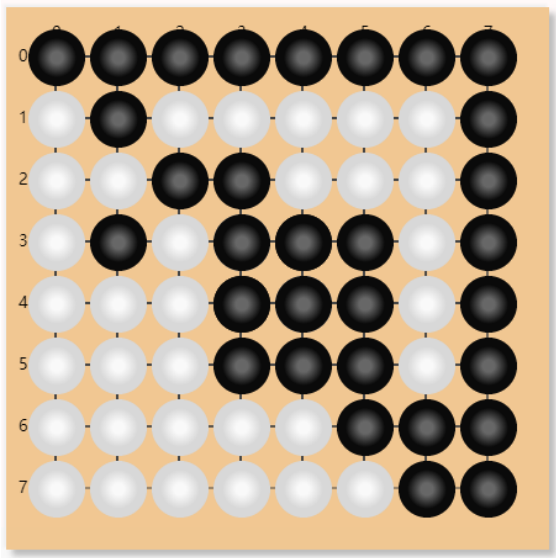

Figure 1: Reversi example

1.2 Problem Applications

This project will help us to learn basic methods to design an AI and practice our skill of programming.

2 Methodology

2.1 Notations

$E(p)$	The evaluation function
α_i	Parameter of the evaluation function

2.2 Data Structure

`chessboard`	A 2D array representing the chessboard 1: white chess, -1: black chess, 0: empty position
`tempboard`	A 2D array representing the chessboard after some steps
`value_table`	A 2D array representing the value of each position
`candidate_list`	A list of all possible positions
`color`	1: white, -1: black, 0: empty
`mycolor`	The color I am playing with
`mychess`	A list of all chess of given ‘color
`emptypos`	A list of all empty positions
`MAX_DEPTH`	The max depth of search
`MAX_INT`	The upper bound of an `int` variable
`MIN_INT`	The lower bound of an `int` variable

2.3 Model Design

Chess game is a typical application of **adversarial search** that we need to try to plan ahead of the world and other agents are planning against us. Therefore, we need to design a heuristic evaluation function and a search algorithm.

We use **MiniMax Search Algorithm**[2] to design the AI.

For the evaluation fuction, we consider these four parts as listed:[3]

- Value table
- Stable discs
- Frontier discs
- Number of my chess

2.4 Detail of Algorithms

2.4.1 Evaluation Function

There are two kinds of stable discs:

1. All 8 directions are filled with chess.
2. For every collinear direction, there is at least one direction which is filled with chess of the same color.

Algorithm 1 Count Stable Discs

```
function CountStable(chessboard, color)
    count ← 0
    mychess ← positions of all chess of given color
    for position in mychess do
        if IsStable(chessboard, color, position) then
            count ++
        end if
    end for
    return count
end function

function IsStable(chessboard, color, position)
    flag ← True
    for all direction do
        for i = 1 to 8 do
            currentpos ← move i steps from position
            if currentpos is beyond border then
                break
            else if currentpos is empty then
                flag ← False
                break
            end if
        end for
    end for
    if flag then
        return True
    end if
```

```
    for all collinear direction do
        flag1 ← True, flag2 ← True
        for i = 1 to 8 do
            currentpos1 ← move i steps in the direction
            currentpos2 ← move i steps in the opposite direction
            check if currentpos1 or currentpos2 is beyond border
            if chessboard[currentpos1] ≠ color then
                flag1 ← False
            end if
            if chessboard[currentpos2] ≠ color then
                flag2 ← False
            end if
            if not flag1 and not flag2 then
                return False
            end if
        end for
    end for
    return True
end function
```

Frontier discs refer to the chess which are adjacent to at least one empty position.

Algorithm 2 Count Frontier Discs

```
function CountFrontier(chessboard, color)
    count ← 0
    mychess ← positions of all chess of given color
    for position in mychess do
        if IsFrontier(chessboard, color, position) then
            count++
        end if
    end for
    return count
end function

function IsFrontier(chessboard, position)
    if position is on the edge of chessboard then
        return False
    end if
```

```
    for all adjacent positions do
        if it is empty then
            return True
        end if
    end for
end function
```

The number of my chess is used near the end of the game.

Algorithm 3 Count My Chess

```
function CountMyChess(chessboard, color)
    mychess ← positions of all chess of given color
    return length of mychess
end function
```

Adding in the `value_table`, together these are the compositions of the evaluation function E.

```
value_table=np.array([
[9999, 0, 11, 8, 8, 11, 0, 9999],
[0, -10, -4, 1, 1, -4, -10, 0],
[11, -4, 5, 5, 5, 5, -4, 11],
[8, 1, 5, -3, -3, 5, 1, 8],
[8, 1, 5, -3, -3, 5, 1, 8],
[11, -4, 5, 5, 5, 5, -4, 11],
[0, -10, -4, 1, 1, -4, -10, 0],
[9999, 0, 11, 8, 8, 11, 0, 9999]
])
```

$$E(p) = \alpha_1 \textsc{ValueTable} + \alpha_2 \textsc{CountStable} + \alpha_3 \textsc{CountFrontier} + \alpha_4 \textsc{CountMyChess}$$

2.4.2 Search Algorithm

Algorithm 4 MiniMax Search with $\alpha - \beta$ Pruning

```
function MiniMax(chessboard, color, current_level_value, pos, depth)
    if depth > MAX_DEPTH then
        return 0
    end if

    tempboard ← Filp(chessboard, color, pos)
    templist ← new empty list
```

$\textsc{FindAllPos}(tempboard,\ -color,\ templist)$

$step \leftarrow \textsc{CountStep}(tempboard)$
if $step < 16$ **then**
  $MAX\_DEPTH \leftarrow 4$
  $\alpha_1 \leftarrow 2,\ \alpha_2 \leftarrow 0,\ \alpha_3 \leftarrow -3,\ \alpha_4 \leftarrow 0$
else if $16 \leq step < 56$ **then**
  $MAX\_DEPTH \leftarrow 3$
  $\alpha_1 \leftarrow 2,\ \alpha_2 \leftarrow 20,\ \alpha_3 \leftarrow -5,\ \alpha_4 \leftarrow 0$
else
  $MAX\_DEPTH \leftarrow 4$
  $\alpha_1 \leftarrow 1,\ \alpha_2 \leftarrow 15,\ \alpha_3 \leftarrow 0,\ \alpha_4 \leftarrow 5$
end if

$value \leftarrow \alpha_1 value\_table[pos] + \alpha_2\textsc{CountStable}(tempboard,\ color)$
$+\alpha_3\textsc{CountFrontier}(tempboard,\ color) + \alpha_4\textsc{CountMyChess}(tempboard,\ color)$

if $color \neq mycolor$ **then**
  $value \leftarrow -value$
end if

$maxvalue \leftarrow MIN\_INT$
$minvalue \leftarrow MAX\_INT$

if $color = mycolor$ **then**
  for p **in** $templist$ **do**
    $tempvalue \leftarrow \textsc{MiniMax}(tempboard,\ -color,\ minvalue,\ p,\ depth+1)$
    if $value + tempvalue \leq current\_level\_value$ **then**
      return MIN_INT
    end if
    if $tempvalue < minvalue$ **then**
      $minvalue \leftarrow tempvalue$
    end if
  end for
else
  for p **in** $templist$ **do**

```
            tempvalue ← MiniMax(tempboard, −color, maxvalue, p, depth + 1)
            if value + tempvalue ≥ current_level_value then
                return MAX_INT
            end if
            if tempvalue > maxvalue then
                maxvalue ← tempvalue
            end if
        end for
    end if

    if color = mycolor then
        delta ← minvalue
    else
        delta ← maxvalue
    end if
    return value + delta
end function
```

2.4.3 Auxiliary Functions

Algorithm 5 Find All Possible Positions

```
function FindAllPos(chessboard, color, list)
    emptypos ← list of all empty positions
    for pos in emptypos do
        Check(chessboard, color, list, pos)
    end for
end function

function Check(chessboard, color, list, pos)
    for all direction do
        flag ← False
        for i = 1 to 8 do
            currentpos ← move i steps from pos
            if currentpos is beyond border then
                break
            else if currentpos is empty then
                break
```

```
            else if chessboard[currentpos] = −color then
                flag ← True
            else
                if flag then
                    append pos to list
                    return
                end if
                break
            end if
        end for
    end for
end function
```

Algorithm 6 Get the Chessboard After a Move

```
function FILP(chessboard, color, pos)
    tempboard ← a copy of chessboard
    tempboard[pos] ← color
    for all direction do
        flag ← False
        for i = 1 to 8 do
            currentpos ← move i steps from pos
            if currentpos is beyond border then
                break
            else if currentpos is empty then
                break
            else if chessboard[currentpos] = −color then
                flag ← True
            else
                if flag then
                    flip all the chess from pos to currentpos
                end if
                break
            end if
        end for
    end for
    return tempboard
end function
```

3 Empirical Verification

3.1 Dataset

As we know, there is no standard test dataset for Reversi. Therefore, during the whole project, I basically designed some data to test the correctness of my algorithms. And in usability test, I also used the `local_code_check` program to test my code. Here I will list some of my own test cases:

1. Test stable discs

```
[[1, 0, 0, 0, 1, 0, 0, 0],
[0, 1, 0, 0, 1, 0, 0, 1],
[0, 0, 1, 0, 1, 0, 1, 0],
[0, 0, 0, 1, 1, 1, 0, 0],
[0, 0, 0, -1, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0]]

[[-1, 0, 0, 0, -1, 0, 0, 0],
[0, -1, 0, 0, 1, 0, 0, -1],
[0, 0, -1, 0, -1, 0, -1, 0],
[0, 0, 0, -1, 1, 1, 0, 0],
[1, -1, 1, -1, 1, -1, -1, 1],
[0, 0, 0, 1, 1, 0, 0, 0],
[0, 0, -1, 0, -1, 0, 0, 0],
[0, 1, 0, 0, 1, 0, 0, 0]]
```

In these two cases, the chess at `(4, 4)` is a stable disc.

2. Test frontier discs

```
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
```

```
[0, 0, 0, 1, 1, 1, 0, 0],
[0, 0, 0, 1, 1, 1, 0, 0],
[0, 0, 0, 1, 1, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0]]
```

Only the chess at `(4, 4)` is not a frontier disc.

3. Others are from the real-time game, I used them with debug mode to check the inner logic of my code. Here are some examples:

```
[[0, 0, 1, 1, 1, 1, 1, 0],
[0, 0, 1, -1, -1, 1, 0, 0],
[0, 0, 1, -1, 1, 1, -1, 1],
[0, 0, -1, -1, 1, 1, 1, 1],
[1, -1, 1, -1, 1, 1, 0, 1],
[1, 0, 1, 1, 1, 1, 0, 0],
[1, 0, 1, -1, 0, 0, 0, 0],
[0, 0, 1, 0, -1, 0, 0, 0]]

[[0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 1, -1, 0, 0, 0],
[0, 1, -1, -1, -1, -1, 0, 0],
[0, -1, 1, 1, -1, 0, 0, 0],
[0, 0, 0, -1, 1, 0, 0, 0],
[0, 0, 0, 1, 1, -1, 0, 0],
[0, 0, 1, 0, 1, 0, 0, 0],
[0, 1, -1, -1, 0, 1, 0, 0]]

[[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 1, -1, 0, 0, 0, 0],
[0, 0, -1, 0, -1, -1, 0, 0],
[-1, -1, 1, 1, 1, -1, 0, 0],
[-1, -1, 1, 1, -1, -1, 0, 0],
[-1, -1, -1, -1, -1, -1, -1, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
```

```
[0, 0, 0, 0, 0, 0, 0, 0]]
```

In the point race and round robin, I basically relied on the game platform to get the test data. If my AI has chosen a weird position, I would copy the chessboard and debug. And I have not used any other platform. After the “Playto” function was disabled, I adjusted the search depth dynamically according to the progress of the game because the game server was overloaded.

3.2 Performance Measure

Honestly, I measured the performance according to the real game and the log file that our server provided. From my observation, there were approximately 3%~%5 steps which were overtime. In the preparation phase, the highest rank of my AI was #4. But in the points race, I only got #59, and also #63 in round robin. I am sad.

3.3 Hyperparameters

The parameters were decided by some chess strategies. For example, the four corners are the most important positions on the chessboard. Therefore, I should take the corners as soon as possible, and in the mean time, prevent my opponent from taking the corners. So I set the value of the corners as 9999(which means infinity). Secondly, avoid frontier discs as possible, so that I can have more choices and let my opponent run out of moves. These two are important on the beginning. While in the middle stage, it is more significant to get stable discs and also avoid frontier discs, so I set a high coefficient to stable discs. At the end of game, the main task is to flip chess as many as possible, also, stable discs are vital. As a result, I adjusted the parameters according to the progress of the game.

3.4 Experimental Results

Well, the best way to measure performance is the rank.

- Usability test: passed all test cases
- Points race: #59
- Round robin: #63

3.5 Conclusion

I think the advantage of my AI is that it is considerate and stable. This is because I used many game strategies to design my AI. Every move was logical and based on some chess strategies. Therefore, my AI would not make any over-offensive or too defensive move.

The disadvantage is mainly speed, I think. Because in the end of points race, our server was overloaded so my AI got lots of timeout. I should continue to do some speed optimizing. Also, there were many aspects that I have not considered such as opening and endgame strategies, even number theory and so on.

Well, through this project, I learnt some basic techniques to design an AI. Although my AI was not so powerful, I am sort of satisfied with my design. What's more, I learned how to write python code.

The possible improvement directions are adding in more game strategies to my evaluation function and use hash technology to make my algorithms faster.

References

[1] Project 1 -AI Algorithm for Reversi.

[2] Aijun Bo. (2005). 几种智能算法在黑白棋程序中的应用. *USTC.*

[3] Rose, B. (1978). Othello: A Minute to Learn... A Lifetime to Master.