CS303 Reverse-Reversi Project Report

A Chess AI Project for Reverse-Reversi

Xiong Muxing
Student Number: 12011818

I. Introduction

In recent years, with the popularity of AlphaGo, the chess AI that beats the world champion in GO chess, people are paying more and more attention to chess AI and the whole field of artificial intelligence.

The AlphaGo we mentioned is a zero-sum chess game. For a zero-sum chess game, the result is either win or lose, so in a fully observable and known environment (like GO game or Reversi game), we can easily make a intelligent agent for us to make better decisions in the zero-sum game and win the game. With proper transformation, the intelligent agent may help us in more zero-sum conditions.

In this project, we intend to make a chess AI that can automatically play Reversed-Reversi. We intend to make this chess AI reliable and powerful that it can beat more human and algorithms and winning the game.

II. Prelimilary

A. Introduction to Reverse-Reversi

- Reversi is a zero-sum game. It consists of a 8x8 square board, and pieces with one black and one white side. Each player has a color, and the aim of the game is to get more of your pieces on the board than the opponent's pieces.
- At the start of the game there are four pieces on the board, two white and two black. You must try to capture opponent pieces and flip them over so they turn into your color. You do this by making a horizontal, vertical or diagonal line of pieces, where your pieces surround the other player's pieces. The surrounded opponent pieces are then captured and will be flipped over to your color, increasing the number of your pieces on the board. Every move you make must capture some opponent pieces. If there is no available move on the board that captures pieces then you must say Pass and your opponent gets to play again. If both players say pass in a row then there are no more moves on the board and the game ends.
- The game ends when the board is full, or both players say pass. At that time the pieces on the board are counted and the player with more pieces wins.

B. Formulation for Reverse-Reversi

- Location: A 2-tuple that marks the position, e.g. (1,2)
- Chessboard: The chessboard of the game. It is represented by a numpy of array. The items in this array are integers with values -1, 0, 1, which respectively represents black chess, empty, white chess.
- Player: The Player that has the move in the current state. Represented by a integer -1 or 1. -1 means black chess will have the move next, 1 means white chess will have the move next.
- State: a class that contains both chessboard and state.

C. Formulation for Min-Max algorithm

- Initial State s_0 : The initial State when the algorithm starts.
- Player(s) : Defines which player has the move in state s.
- Actions(s) : Returns the set of legal moves in s.
- Transition Function $S \times A \rightarrow S$: Defines the result of a move
- Terminal test Terminal(s) : True when the game is over or current search depth exceeds limit, False otherwise. States where game ends are called terminal states.
- Eval(s, Player(s)) : Evaluation function for a game of state s for player Player(s). The function judge that if the Player has the superiority at that state s. The bigger the the return utility value is, the better the situation is for the player.
- Result(s,a) : Returns the state that the input state s (contains the current Player and Chessboard) conducts a step at position a.

III. Methodology

A. General Workflow

In this part we will discuss the general workflow of the algorithm. Basically we use Mini-max algorithm and α-β pruning to implement our intelligent agent.

Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.The minimax algorithm performs a depth-limited depth first search algorithm for the exploration of the game tree.

In this algorithm, we have two basic functions, Minimum Value(State) and Maximum Value(State). Minimum Value(State) tries to get the minimum value among all the choices while Maximum Value(State) tries to get the maximum. Since both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit, we can suppose there is a Player Min who tries to get the minimum of the opponent's value and a Player Max tries to

get the maximum of his value.So our Max perform Maximum Value(State) among the minimum values that Min choose for him; and Min perform Minimum Value(State) among the maximum value that Max chooses.

However, the game can lasts many round so the search tree must be deep, and there are often multiple choices for the Player so the none-leaf nodes of the search tree may have many children. So the game tree is often too big to search. Thus, we add a search depth limit to the tree and add an evaluation function to evaluate the value of the none-leaf state. Besides, we also add an alpha-beta pruning to cut the useless branches of the tree.

The evaluation function basically evaluates the position value of the state and how many chesses on the board are stable. And there is a parameter to determine how much they contribute to the evaluate.

$$Eval(s, Player(s)) = \gamma \times EvalPosition(s, Player(s)) + (1-\gamma) \times EvalStability(s, (Player(s)) \tag{1}$$

1) The matrix for position evaluation:

$$\begin{matrix} -100 & 25 & -1 & -5 & -5 & -1 & 25 & -100 \\ 25 & 45 & -1 & -1 & -1 & -1 & 45 & 25 \\ -1 & -1 & -1 & -2 & -2 & -1 & -1 & -1 \\ -5 & -1 & -2 & -1 & -1 & -2 & -1 & -5 \\ -5 & -1 & -2 & -1 & -1 & -2 & -1 & -5 \\ -1 & -1 & -1 & -2 & -2 & -1 & -1 & -1 \\ 25 & 45 & -1 & -1 & -1 & -1 & 45 & 25 \\ -100 & 25 & -1 & -5 & -5 & -1 & 25 & -100 \end{matrix}$$

B. Detailed Algorithm Design

The detailed algorithm design includes MinValue and MaxValue with and without the Alpha-beta pruning and the evaluation algorithm mentioned above. (Pseudocode Algorithm 1-4 and function (1))

Algorithm 1 Minimum Value without Alpha-beta pruning

Input: State
Output: Utility Value

```
if Terminal(State) then
    return Eval(State,Player(State)
end if
v → ∞
for a in Actions(State) do
    v → MIN(v, MaximumValue(Result(State, a)))
end for
return v
```

C. Analysis

- Time Complexity:
  Suppose the branching factor of the search tree is m, the search begins at depth h, and the searching depth limit is l. The we have to evaluate nodes of number m^{h+l-1} and search nodes of number $m^h + m^{h+1} + m^{h+2} + ... + m^{h+l-1}$. For both search and evaluate, the time complexity is O(1). So the total time complexity is $O(m^{h+l})$
- Space Complexity:
  The space Complexity of the algorithm is $O(m \times (h+l))$

Algorithm 2 Maximum Value without Alpha-beta pruning

Input: State
Output: Utility Value

```
if Terminal(State) then
    return Eval(State,Player(State)
end if
v → −∞
for a in Actions(State) do
    v → MAX(v, MinimumValue(Result(State, a)))
end for
return v
```

Algorithm 3 MaxValue with Alpha-beta pruning

Input: State,Alpha,Beta
Output: Utility Value

```
if Terminal(State) then
    return Eval(State,Player(State)
end if
v → −∞
for a in Actions(State) do
    v → MAX(v, MinValue(Result(State, a), Alpha, Beta))
    if v >= Beta then
        return v
    end if
    Alpha = MAX(Alpha, v)
end for
return v
```

IV. Experiments

In the experiments part, I will introduce the experiments I have done to my intelligent agent of Reversed-Reversi.

A. SetUp

I do the experiment based on the win/lose condition on the Reversed-Reversi testing platform. I adjusted some parameters in the position matrix and the parameter γ in the evaluation function.

B. result

1) Adjusting parameter γ: In this experiment, I adjust the evaluation parameter γ and compete with the opponents ranked 41-50. The result is listed below.

γ	Win	Lose
0.5	7	13
0.7	9	11
0.9	7	13

Algorithm 4 MinValue with Alpha-beta pruning

Input: State,Alpha,Beta
Output: Utility Value

if Terminal(State) **then**
  return Eval(State,Player(State)
end if
$v \to \infty$
for a **in** Actions(State) **do**
  $v \to MIN(v, MaxValue(Result(State, a), Alpha, Beta))$
  if $v <= Alpha$ **then**
    return v
  end if
  $Beta = MIN(Beta, v)$
end for
return v

2) Varying the position matrix: In this experiment. I vary the position matrix and use the original/adjusted matrix to compete with the opponents ranked 41-60. The original/adjusted matrix and competition result are listed below.

original matrix

$$\begin{matrix} -500 & 50 & -1 & -5 & -5 & -1 & 50 & -500 \\ 50 & 45 & -1 & -1 & -1 & -1 & 45 & 50 \\ -1 & -1 & -1 & -2 & -2 & -1 & -1 & -1 \\ -5 & -1 & -2 & -1 & -1 & -2 & -1 & -5 \\ -5 & -1 & -2 & -1 & -1 & -2 & -1 & -5 \\ -1 & -1 & -1 & -2 & -2 & -1 & -1 & -1 \\ 50 & 45 & -1 & -1 & -1 & -1 & 45 & 50 \\ -500 & 50 & -1 & -5 & -5 & -1 & 50 & -500 \end{matrix}$$

adjusted matrix

$$\begin{matrix} -100 & 25 & -1 & -5 & -5 & -1 & 25 & -100 \\ 25 & 45 & -1 & -1 & -1 & -1 & 45 & 25 \\ -1 & -1 & -1 & -2 & -2 & -1 & -1 & -1 \\ -5 & -1 & -2 & -1 & -1 & -2 & -1 & -5 \\ -5 & -1 & -2 & -1 & -1 & -2 & -1 & -5 \\ -1 & -1 & -1 & -2 & -2 & -1 & -1 & -1 \\ 25 & 45 & -1 & -1 & -1 & -1 & 45 & 25 \\ -100 & 25 & -1 & -5 & -5 & -1 & 25 & -100 \end{matrix}$$

Win/Lose Condition

Matrix	Win	Lose
Original	9	11
Adjusted	11	9

C. Analysis

For evaluation parameter γ, in the range of 0.5-0.9 the wining samples increase first and then decrease, so the best evaluation parameter γ is likely to be in the range of 0.5-0.9, so we choose 0.7 as our final result.

For the position matrix, the adjusted matrix performs better than the original one, so we should accept the adjusted one.

V. Conclusion

A. Advantages and disadvantages of the algorithm

Mini-max algorithm is an relatively simple algorithm which supposes the opponent will choose the best decision. The advantage of it is that it can use plenty of prior knowledge to help decision in the evaluation function. And the disadvantage of it is that the algorithm is determined and is restricted by your knowledge of the game. Besides, usually your opponents will not act as what you evaluate as his best choice, which decreases the correctness of the algorithm. And because of time limit, the searching depth is also limited.

B. Lessons learned in the project

In this project, I learned that we have to be careful when we try to add or change something in the original project. You need to do experiments carefully about the changes it will bring. Adding too many unnecessary things to your project is complicated and may cause performance decrease.

C. Further thoughts on improving the project

1) Using Numba to increase the program running speed. In that case, the searching depth can be increased.:

2) Add more components(like how many positions you can put your chess on) to the evaluation function to increase the accuracy of evaluation function:

References

[1] Einar Egilsson, "Online Reversi gameplay and rules". https://cardgames.io/reversi/

[2] JavaTPoint, "Mini-Max Algorithm in Artificial Intelligence". https://www.javatpoint.com/mini-max-algorithm-in-ai

[3] Orion Nebula, "AI+AlphaBeta". https://zhuanlan.zhihu.com/p/35121997