COMP 472 [or COMP 6721] Template for the Reports (essentially, this is the LNCS template)

Zhongxu Huang[1] and Yixuan Li[2]

[1] realdonald9@gmail.com

[2] liyixuan4030614@gmail.com

1 Introduction to technical details (1/2 to 1 page)

1.1 Minimax (1/2 page)

Minimax is a state space search approach for game playing. Normally, it is used in zero-sum games, meaning gains and losses of all players sum up to 0 at each step[1]. There are 2 (or 3) attributes in Minimax: state space search tree T, heuristic function e(n), (and maximum search depth in n-ply minimax).

In minimax, each node only contains one value, that is heuristic value. At Max layer, each node will choose the maximum heuristic value of its children; At Min layer, each node will choose the minimum heuristic value of its children. At the end, the root will choose the node based on the values.

However, minimax algorithm is impractical for complex problem in real life, because it completely analyses each node in the game tree. Without affecting the result, the performance can be optimized by alpha-beta pruning.[1]

1.2 Alpha-beta pruning (1/2 page)

Alpha-beta pruning is a search algorithm that decreases the number of nodes that are evaluated by the minimax algorithm in state space search tree[2]. Because number of nodes in state space for zero-sum games will increase exponentially as depth increases, this reason restricts minimax algorithm to look ahead and choose a better move for the root.

In alpha-beta pruning, each node contains two values, alpha and beta. alpha means the lower-bound value, and beta means the upper-bound value. For Max player, it will prune the branch if alpha value $\geq$ beta value of its children; For Min player, it will prune the branch if beta value $\leq$ alpha value of its children. This allows alpha-beta pruning is not going to construct the complete game tree by ignoring branches that cannot be beneficial to evaluate current node.

2 Description and justification of heuristics (1 to 1 ½ page)

2.1 Heuristic function

The heuristic evaluation function is an important part of the AI in the game of the game, and it determines the decision-making steps of the AI. The main task is to evaluate the importance of each node in the state space tree.

In general, the heuristic function evaluates the current form of the game for the AI. It simulates the form of the game after each step of decision for AI and makes the most favorable step.

The most important step in the design of the heuristic function is also a very complicated part, which is like a process of building a mathematical model. Designing the evaluation function for the game Double Card requires good understanding of the game and some knowledge about specific procedures to derive heuristic.[3]

2.2 Heuristic 1 (1 page)

Before starting describing our first heuristic function, we will make some assumption signs for clarification.

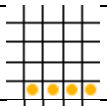
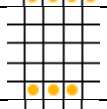
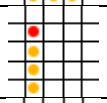
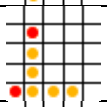
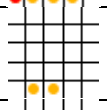
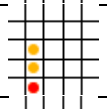
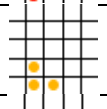
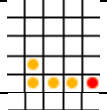
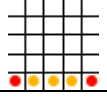
Assuming:

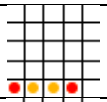
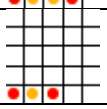
1. O means occupied point by one player
2. @ means the other side or boundary
3. X means empty point.

- When there is a "4-connected" on the board, meaning "OOOO", this situation indicates one of players has won the game. So, we rate the heuristic value of this situation as 100,000.
- When there is an "Alive-3", meaning "XOOOX", this situation indicates one of players has a three-segment connection which is not blocked by another player or the boundary. So, we consider this state most likely leads to win, and we rate the heuristic value of this situation as 10,000.
- When there is a "Dead-3", such as "@OOOX", "@OOXO", "@OXOO" and so on, this situation indicates one of players have a three-segment connection which is blocked on one side but is alive on another side. So, we consider this state is advantageous for the current player, and we rate the heuristic value of this situation as 1,000.
- When there is a "Double-Dead-3", meaning that there is more than 1 "Dead-3" starting from one position, then we consider this situation most likely leads to win, so we rate the heuristic value of this situation as 10,000
- When there is an "Alive-2", such as "XOOXX", "XOXOX", "XXOOX" etc., this situation indicates one of players has a two-segment connection which is not

blocked on both sides. Because an “Alive-2” can form an “Alive-3” easily, we rate the heuristic value of this situation as same as “Dead-3” which is 1,000.

- When there is a “Dead-2”, such as “@OXOX” etc., this situation indicates one of players has a two-segment connection which is blocked on one side but is alive on another side. We rate the heuristic value of this situation as 500.
- When there is a “Double-Alive-2”, meaning that there is more than 1 “Alive-2” starting from one position, we rate the heuristic value of this situation as 10,000.
- When there is a “Dead-3-and-Alive-2”, meaning that one side has a “Dead-3”, and one side has an “Alive-2”, we rate the heuristic value of this situation as 10,000.
- When there is an “Unavailable-3”, meaning “@OOO@”, this situation indicates it is impossible to form 4-connected segment, so we rate the heuristic value of this situation as -100.
- When there is an “Unavailable-2”, meaning “@OO@”, we rate the heuristic value of this situation as -100.
- When there is an “Unavailable-1”, meaning “@O@”, we rate the heuristic value of this situation as -100.

Situation	Diagram (take ‘color’ as an example)	Heuristic value
“4-connected”		100,000
“Alive-3”		10,000
“Dead-3”		1,000
“Double-Dead-3”		10,000
“Alive-2”		1,000
“Dead-2”		500
“Double-Alive-2”		10,000
“Dead-3-and-Alive-2”		10,000
“Unavailable-3”		-100

“Unavailable-2”		-100
“Unavailable-1”		-100

These are the weight ratios of the evaluation factors we used in heuristic 1. In our program, we have a function called *evaluation* which is a method that computes scores for all the pieces on the board, and gives a final score according to the current situation.

Processes in evaluate function:

1) We named ScoreAI as the sum of unilateral score for AI player, and *ScoreHuman* as the sum of score for human player
2) Then turn the 2-dimensional board into N 1-dimensional arrays in four directions (row, column, positive diagonal, negative diagonal)
3) Perform score calculation on all 1-dimensional arrays
4) 可以再加

2.3 Heuristic 2[4] (1 page)

Heuristic 2 is implemented in the function called *evaluation_2*, and this heuristic is a more naïve heuristic than heuristic 1, because instead of evaluating a node by checking “dead” or “alive” same consecutive segments, heuristic 2 only evaluates a node by the occurrence times in each 4 consecutive segment. The detailed process describes below:

1) We named danger_color, good_color, danger_dot, good_dot as the sum of dangerous factor and goodness factor for color and dot.
2) Then, as what we did in heuristic 1, we turn the 2-dimensional board into N 1-dimensional arrays in four directions (row, column, positive diagonal, negative diagonal)
3) Perform score calculation on all 1-dimensional arrays
    a. Dangerous factor calculation
    In a four consecutive segments, if there are more red color or white color segments in the particular line, then the line is potentially dangerous for dot player, and verse visa.
    While iterating four consecutive segments in each row, each column, each positive and negative diagonal, we record the occurrence times of “red”, “white”, “solid”, “hollow”. And dangerous calculation follows this:

$$maxAllowPerc = \frac{Number\ of\ connected\ segments - 1}{Number\ of\ connected\ segments}$$

(Number of connected segments means how many consecutive segments we check each time, in our program, it is equal to 4.)

$$PercWon = \frac{Occurrence\ times}{Number\ of\ connected\ segments}$$

(e.g. for "red", Occurrence time means how many times does "red" appear in every check.)

$$dangerous = PercWon * \frac{DANGER\ FACTOR}{maxAllowPerc}$$

(we set DANGER FACTOR to 100)

The value of dangerous is the evaluation value for one non-empty position on the board, and we sum dangerous value of all non-empty position to danger_dot and danger_color variables.

b. Goodness factor calculation
Similar with Dangerous evaluation, in a four consecutive segments, if there are more red color or white color segments in the particular line, then the line is considered good for color player. And goodness evaluation follows this:

$$goodFac = \frac{GOOD\ FACTOR}{(Number\ of\ connected\ segment - 1)^2}$$

(we set GOOD FACTOR to 50, since we consider heuristic 2 is a more defensive heuristic even though offensive evaluation may lead to better results.)

$$goodness = Occurrence\ times^2 * goodFac$$

The value of goodness is the evaluation value for one non-empty position on the board, and we sum goodness value of all non-empty value to good_dot and good_color variables.

4) At the end, we calculate ScoreAI and ScoreHuman based on the four values, danger_color, danger_dot, good_color, good_dot, and we give a weight to each value while doing calculation, then return ScoreAI – ScoreHuman.

3 Analysis of heuristics and results (3 pages)

3.1 Analysis of heuristic No.1 (1/2 page)

3.2 Analysis of heuristic No.2 (1/2 page)

Pl

3.3 Comparison of two heuristics (1 page)

Ple

3.4 Alpha-beta pruning effectiveness analysis (1 page)

Plea

4 Description of encountered difficulties and solutions (1/2 to 1 page)

4.1 Difficulty 1: Time complexity of heuristic function

In the heuristic function, the more winning and failing conditions are considered, the more computing time and space it consumes and the lower the CPU utilization is.
In minimax approach, the algorithm searches can be approximated to m^n where m means number of branches on second level to the last level, n means number of branches on the last level. Although performance is improved if we switch to alpha-beta pruning, but the number of searches still cannot be greatly reduced.

Solution 1: Discard isolated segment.

While searching on the board, we discard isolated points and only search for the position which has other segments in the surrounding area. This avoids searching for obviously useless nodes and greatly improve the overall search speed.

Solution 2: Stop evaluating when 100% win or lose

When there is a losing or a winning game in the search process, it will return the evaluation value directly and no longer keep searching.

Solution 3: Use history table (not implemented)

In each iteration of alpha-beta pruning function, the pieces on the board only increase by 1, and the scores in most positions of the board are not changed, so it is not required to scan the entire board for each iteration during evaluation.
Instead, we can save the score of the board in the previous iteration. Before each iteration, we scan the history table at the very beginning, if one of the positions in history table is same as the new position in current iteration, we won't re-evaluate the position. This eliminates the need to rescan the entire board every time, and increases efficiency.

4.2 Difficulty 2

Solution:

5 Team work (1/2 page)

Tasks	Team members	
	ZhongXu Huang	Yixuan Li
Coding in minimax and alpha-beta	√	
Design heuristic functions	√	√
Optimize heuristic functions	√	
Testing minimax and alpha-beta		√
Participation at the Tournament	√	√
Report	√	√

References

1. Wikipedia, zero-sum game, https://en.wikipedia.org/wiki/Zero-sum_game
2. Wikipedia, alpha-beta pruning, https://en.wikipedia.org/wiki/Alpha–beta_pruning
3. Creating admissible heuristics from functions, https://cs.stackexchange.com/questions/19976/creating-admissible-heuristics-from-functions
4. Connect 4 heuristic function, http://git.mrman.de/ai-c4/blob/master/s3505919.pdf
5.