Project 1: Tic Tac Toe

By Haiting Zhu

Content

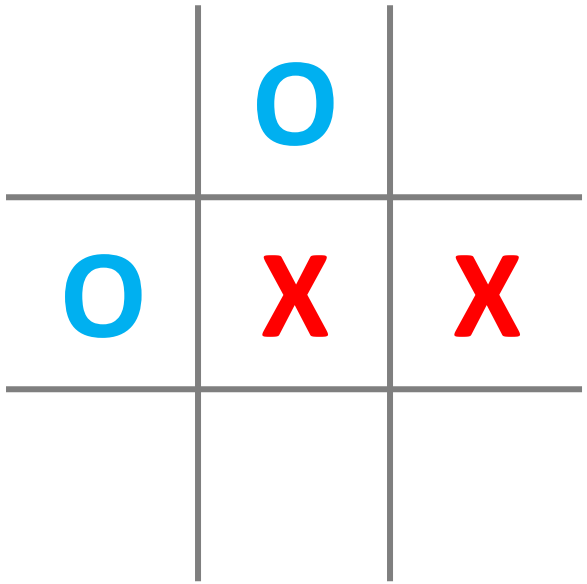
1) Project Summary
2) Part I: Basic Tic-Tac-Toe
   a) Algorithm: MINIMAX
   b) Competition between Human and AI
3) Part II: Advanced Tic-Tac-Toe
   a) Program structure
   b) Algorithm explanation: H-MINIMAX, alpha-beta pruning
   c) Competition between Human and AI
4) Self-examination

Project Summary

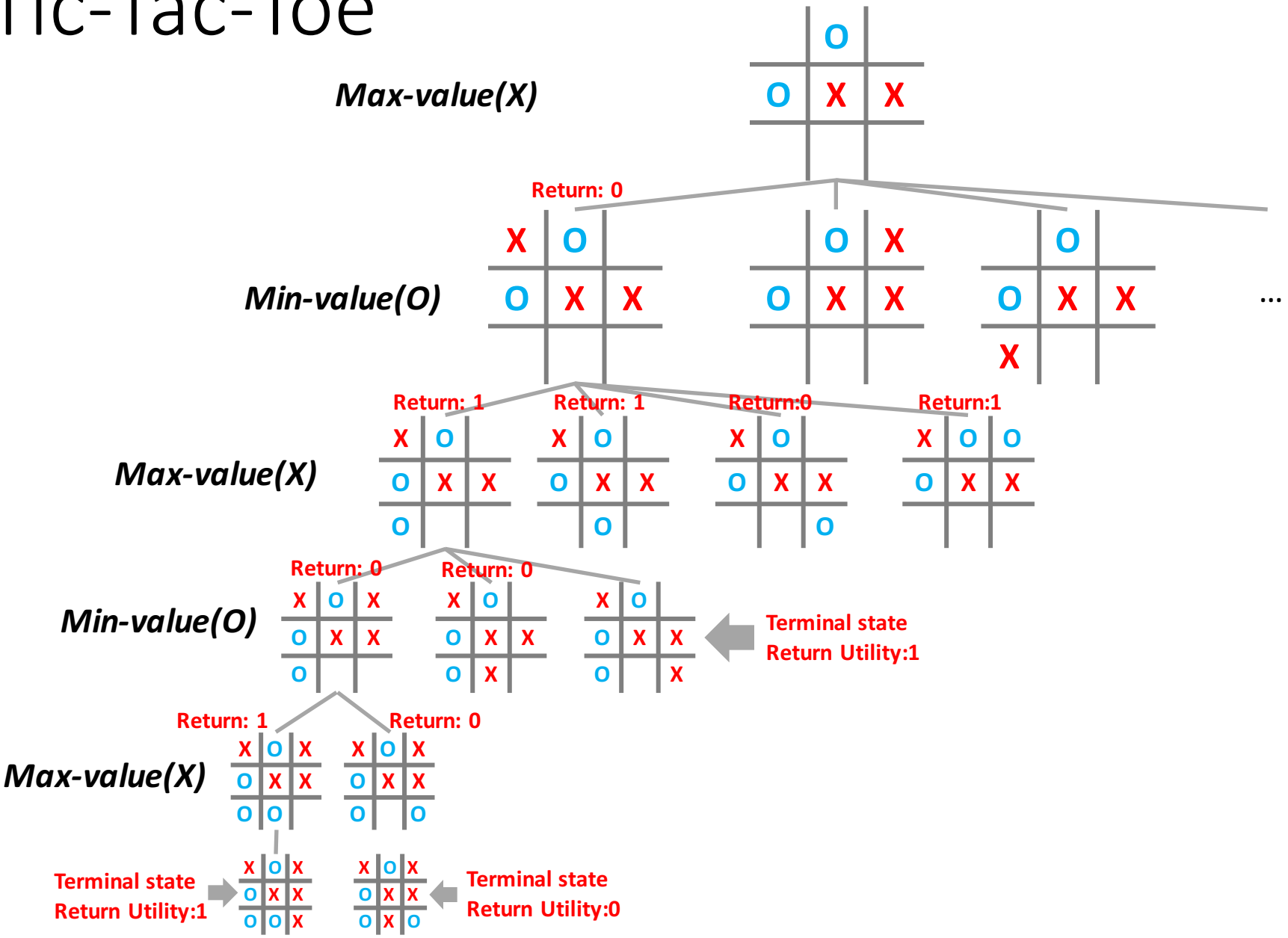
- Aim: Construct the program to play Tic-Tac-Toe with human
- Tool(Language): Python 3
- Individual work
- Result: Works most optimally in basic Tic-Tac-Toe game while makes several mistakes in advanced Tic-Tac-Toe game(but works pretty good under most conditions)

Part I: Basic Tic-Tac-Toe

Algorithm

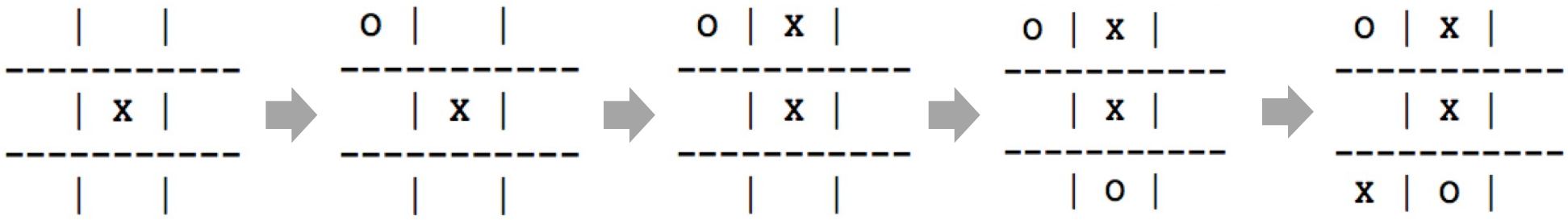


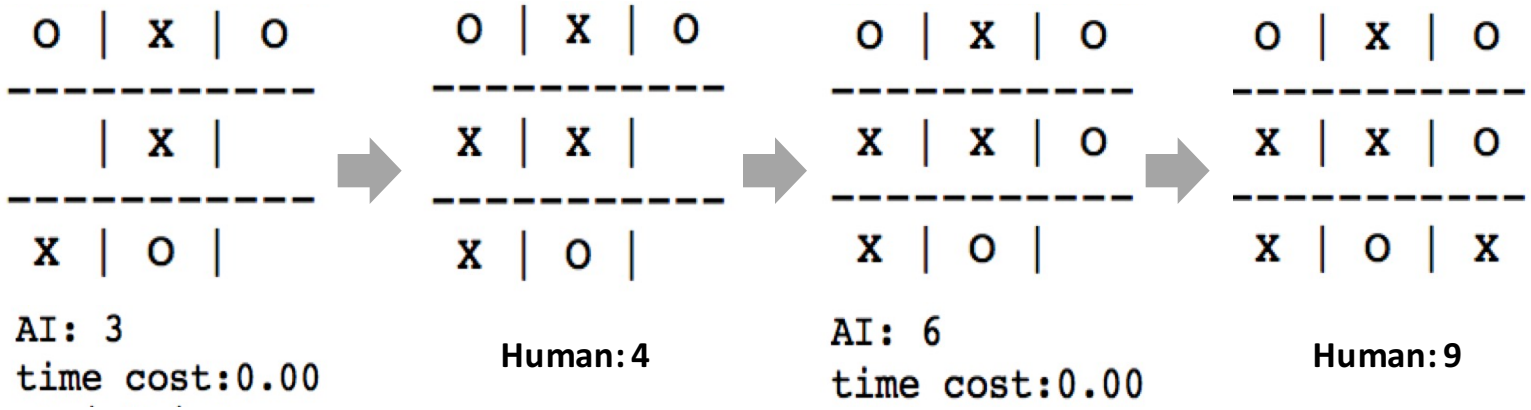
Algorithm in basic TTT program: Explore every possible state and use MINIMAX function to simulate the game as every player perform most optimally. Under such a condition, the program return the wisest move for AI.



Part I: Basic Tic-Tac-Toe

Competition between human and AI





Result: Draw

In another case where AI plays first, it cost about 4 seconds to find the first move while other moves will only cost less than 0.1 second. This is because it will construct the biggest(deepest) tree to find the first move which needs about 9!=362880 Operations.

Part II: Advanced Tic-Tac-Toe

Rule changes:

Grid: 3*3 → 9*9
Move constraint: play in the corresponding board decided by the previous player, but if the small board is full, the move can be anywhere
How to win: win a single small board like in basic TTT

Difficulty comes…

Only MINIMAX can not solve the problem in acceptable time, the depth of the search can be as big as 81 (in basic TTT, it is only 9 and will cost 4 seconds to get the result)

New algorithm needed:

H-MINIMAX + Alpha-beta pruning + Small tactics for move under certain condition

Part II: Advanced Tic-Tac-Toe

program structure

Initial State	Use the list to construct a two-dimension board
Actions	**1. If AI plays first, the first move is the center of the board(5,5), this grid has the biggest possibility to win (similar to the basic TTT)** 2. Get previous player's move and find possible moves in the corresponding small board(if that board is not full) **3. If that small board is full, choose 9 random moves**
Result	Use the list to represent the result state
Terminal test	1. Test whether game is over and return result of the game **2. Cutoff test: as mentioned above, to ask program to explore every move is impractical(the search tree is too deep), the program should stop searching under certain condition and make decision. In this program, I limit the search depth(depth <= 6)**
Utility	1. For terminal state, return the final numeric value of the game for AI **2. For non-terminal state, return the evaluation result of the game for AI, namely, the chance to win**
Strategy	**1. H-MINIMAX(like basic TTT): suppose every player will make the wisest decision, simulate the game (not the whole game) and find the move with most possibility to win** **2. alpha-beta pruning: record the best choice for AI and human along the path and thus eliminate some search if next move definitely won't be considered as the best choice for AI or human**

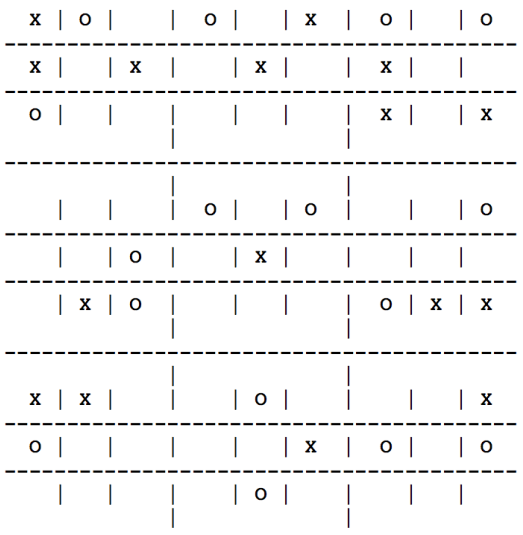
Part II: Advanced Tic-Tac-Toe

Algorithm explanation(1/2)

H-MINIMAX

UTILITY(s)	If TERMINAL-TEST(s)
EVAL(s)	If CUTOFF-TEST(s,d)
max$_{a \epsilon Actions(s)}$ H-MINIMAX(RESULT(s,a),d+1)	If PLAYER(s)=MAX
min$_{a \epsilon Actions(s)}$ H−MINIMAX(RESULT(s,a),d+1)	If PLAYER(s)=MIN

* Except the function with red color, other functions are same as that in basic TTT



For this non-terminal state:
AI(X): 8 points Human(O): 12 points
AI-Human: -4 points

Cutoff-test

Assign a specific depth as the indicator that the program should stop running. According to the performance of different depth, the program is limited to depth 6(see next page).

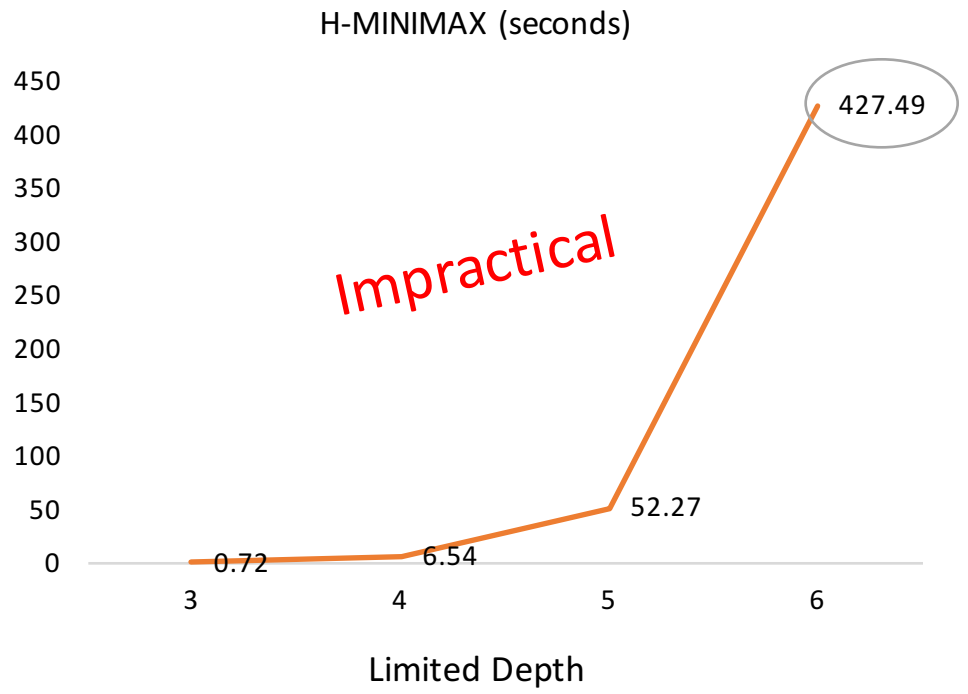
Evaluation Function

Return an estimated chance for AI to win:
Scan every small board, check each row, column and diagonal line. If there are two moves from same player and the other box is blank, then that player gets two points;
Calculate the scores of two players respectively and the estimated chance for AI to win is (AI's score – human's score);
The result is not accurate, but it is close to the real result. As AI has more points than human, it has more opportunity to make up the final blank box with its flag and win the game.

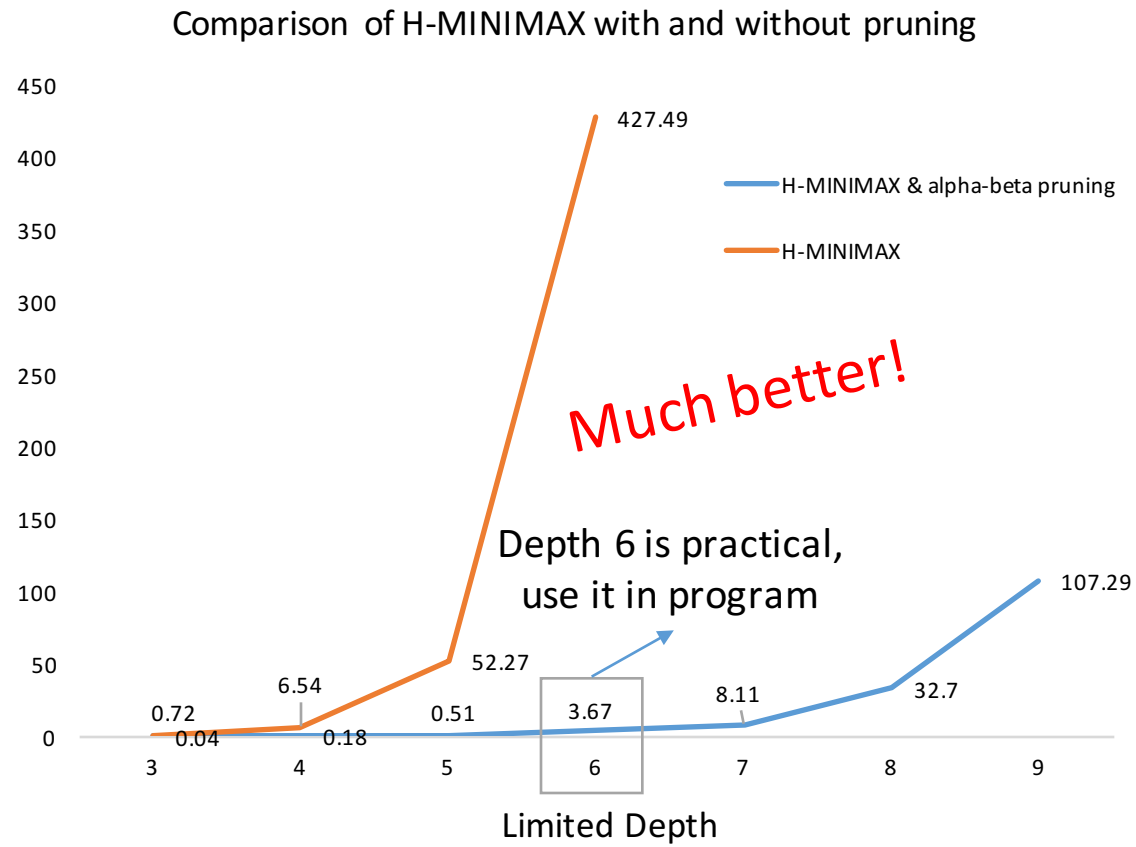
Part II: Advanced Tic-Tac-Toe

Algorithm explanation(2/2)

Only H-MINIMAX can not solve the problem:

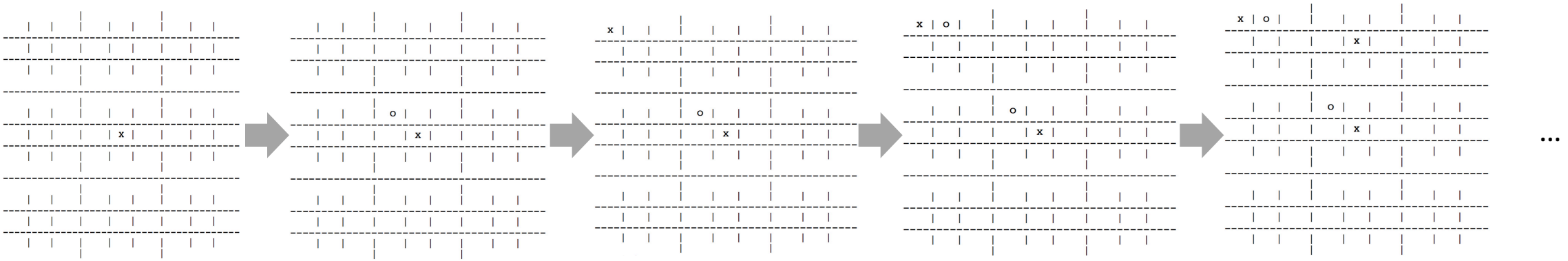


Need pruning (use *alpha-beta pruning* from textbook):

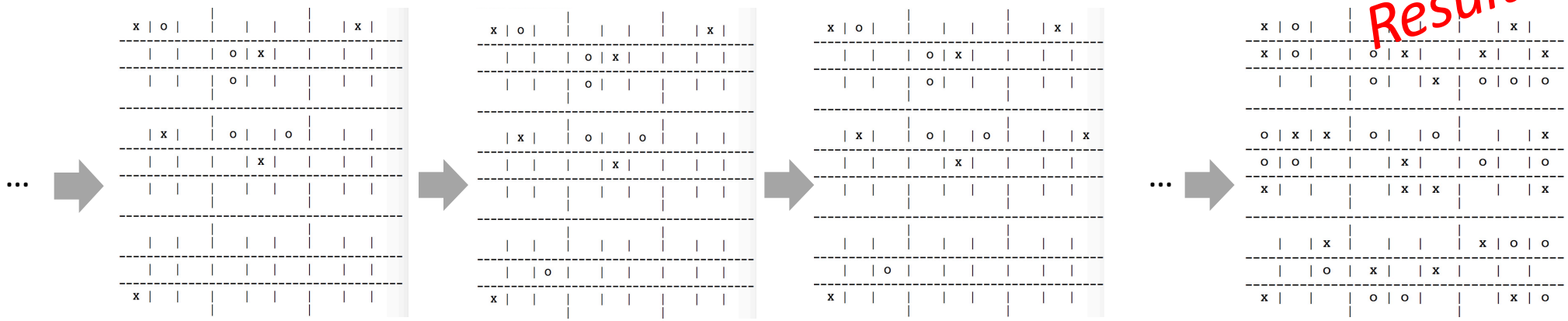


Part II: Advanced Tic-Tac-Toe

Competition between human and AI*



Human: 5,5 | AI: 5,1 | Human: 1,1 | AI: 1,2 | Human: 2,5 | ...



... | Human: 7,7 | AI: 7,6 | Human: 6,3 | ... | AI: 3,7

Result: AI wins

* Full game is in file "advanced TTT between human and AI.txt"

Self-examination

- 1. Should try more evaluation function. In this case, my AI beats human and another AI(from Xuexun Xiao), which indicates that this function is acceptable and also simple. But actually, this simple function should be polished and have better performance;
- 2. Should try to improve alpha-beta pruning algorithm by sorting the value of the subtree, which will improve the program's performance;
- 3. The coding style is not that good, which needs more practice…