Gomoku Game in Haskell

Qinhan Zhou (qz2380) Zheng Yao (zy2388)

Contents

1	Introduction	1
2	Implementation 2.1 Board.hs 2.2 AI.hs 2.3 Main.hs	1 1 1 2
3	Performance	2
4	Codes 4.1 Main.hs 4.2 Board.hs 4.3 AI.hs	4 4 4 8
5	Reference	10

1 Introduction

We implement an AI v.s. AI Gomoku game, also called Five in a Row. Here it's played on a board with size 10*10. Black and white players alternate turns to place a stone of their color on an empty intersection.

We use minimax search algorithm to depth of three employing alpha-beta cut-off strategy to address the two players playing against each other.

2 Implementation

There are three files: Board.hs, AI.hs and Main.hs.

2.1 Board.hs

Define a 10*10 Board and each player's move can denote by a Point. Each point has its color(Black or White) and its corresponding position (Int, Int).

Key methods are addPoint and checkWin. After each move, we check four directions whether there is already five same color points in a row. There is no need to check all the boards, just four lines those includes the latest point.

2.2 Al.hs

Key method here is where to put the current point to realize a game of competition. We use Minimax algorithm to alternate between the two AI players, the player desires to pick the move with the maximum score. In turn, the scores for each of available moves are determined by the opposing player which of its available moves has the minimum score. Scores are calculated as: score 100000 when 5 in a row; 10000 with 4 in a row; 1000 with 3 in a row and 100 with 2 in a row. Build a tree of depth 3 to compare all possible next three moves and pick the most favorable one for current move.

Then we improve Minimax by alpha-beta cut-off. Each node has a boundary [alpha, beta]. alpha means lower boundary and beta means the upper boundary. Each time when $beta \leq alpha$, we no longer search more sub-trees, This is a process of pruning. We refer to other's codes when we implement the minmaxAlpha and minmaxBeta methods but we modified it to fit our data structures.

We used parallel strategies in two places. First, we run the minmax algorithm on each child board of the current board in the board tree in parallel and choose the one with highest score as the next move. Second, to rate each board, we use parallel to get the score of current board in each possible directions. In both cases, we use parMap rdeepseq as our parallel strategy.

Some steps of alternatively running two AIs on a 10*10 board is shown in the table below. O and X denotes the two users and _ denote a vacant place in board.

1	1	1	L
[_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,0,_,_,_,_]
[_,_,_,_,_,_,_,_,_]	[_,_,_,0,_,_,_,_,_]	[_,_,_,0,X,_,_,_,_,_]	[_,_,_,0,X,_,_,_,_]
[_,_,_,X,_,_,_,_]	[_,_,_,X,_,_,_,_]	[_,_,_,X,_,_,_,_]	[_,_,_,_,X,_,_,_,_]
[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_,_]		[_,_,_,_,_,_,_,_,_,_]	[[_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,X,_,_,_]	[_,_,_,_,X,_,_,_]
[_,_,_,0,_,_,_]	[_,_,_,0,_,_,_,_]	[_,_,_,_,0,_,_,_,_,_]	[_,_,_,0,_,_,_,_]
[,,,0,X,,,,,]	[_,_,_,0,X,_,_,_,_,_]	[,,,0,X,,,,,]	[_,_,_,0,X,_,_,_,_]
	[_,_,_,X,_,_,_]	[]	[,,0,,X,,,,,]
	[_,_,_,X,_,_,_,]	[[,,,,X,,,,,]
	[_,_,_,0,_,_,_,]		[_,_,_,0,_,_,_,]
	[_,_,_,_,_,_,_,_,_]		
	[_,_,_,_,_,_,_,_,_]		
	[_,_,_,_,_,_,_,_,_]		
		/_/_/_/_/_/_/_/_/	
[_,_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]	[[[]
[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]	[_,_,_,_,_,_,_,_,_]
[_,_,_,_,_,_,_,_,_] [_,_,_,_,X,_,_,_] [, , , ,0, , , , ,]	[_,_,_,_,_,_,_,_,_] [_,_,_,_,X,_,_,_]	[_,_,_,_,_,_,_,_,_] [_,_,_,_,X,_,_,_]	[_,_,_,0,_,X,_,_,_,]
[_,_,_,_,_,_,] [_,_,_,_,X,_,_,_] [_,_,_,0,_,_,_,]	[_,_,_,_,_,X,_,_,_] [_,_,_,_,X,_,_,_] [_,_,_,0,_,_,_,]	[_,_,_,_,X,_,_,_] [_,_,_,0,_,_,X,_,_,_]	[_,_,_,0,_,X,_,_,_] [_,_,_,0,_,X,_,_,_]
[_,_,_,X,_,_,_,] [_,_,_,X,_,_,_] [_,_,_,0,_,_,_,] [_,_,0,X,_,_,_,]	[_,_,_,X,_,_,_] [_,_,_,X,_,_,_] [_,_,_,0,X,0,_,_,] [_,_,_0,X,0,_,_,]	[_,_,_,_,X,_,_,_] [_,_,_,0,,,,,_,] [_,_,0,X,0,_,_,]	[_,_,_,0,_,X,_,_,_] [_,_,_,0,_,X,_,_,_] [_,_,_,0,X,0,_,_,_]
[_,_,_,X,_,_,_,] [_,_,_,X,_,_,_] [_,_,_,0,X,_,_,_,] [_,_,0,X,_,_,_,] [_,_,0,X,_,_,_,]	[_,_,_,,,,,_,_,] [_,,_,,,,,,,,,,,] [_,,_,0,x,0,,,,,,,] [_,,0,,x,0,,,,,,]	[_,_,_,,x,,,_,_] [_,_,_,0,,,,,,,] [_,_,0,x,0,_,_,] [_,0,x,0,_,,_,]	[_,_,_0,_X,_,_,_] [_,_,_0,_X,_,_,_] [_,_,_0,X,0,_,_,] [_,_,0,X,0,,_,_] [_,0,X,0,X,0,_,]
[_,_,_,,X,_,_,_] [_,_,_,X,_,,_,] [_,_,_,0,X,_,,_,] [_,0,X,_,,_,,] [_,X,_,X,,_,,_,]	[_,_,_,,X,_,_,_] [_,_,_,0,X,0,_,_,_] [_,,0,X,0,_,_,_] [_,X,_,X,_,,,_,]	[_,_,_,,X,_,_,_] [_,_,_,0,,,,,,] [_,,_,0,X,0,,_,,] [_,0,X,0,,,,,] [_,X,,X,,X,,,,]	[_,_,_0,_X,_,_,_] [_,_,_0,_X,_,_,_] [_,_,_0,X,0,_,_,_] [_,_,0,X,0,,_,_] [_,0,_X,_X,_,1,_] [_,X,_,X,_,1,_]
[_,_,_,,,,,_,_] [_,,_,,,,,,,,,,,] [_,,,,,0,,,,,,,,,] [_,,,0,X,,,,,,,,] [_,,0,,X,,,,,,,,] [_,X,,,X,,,,,,,,] [_,2,,,0,,,,,,,,]	[_,_,_,,_,,_,_,_] [_,_,_,0,,,,,,] [_,,_,0,x,0,,,,,] [_,,0,x,0,,,,,] [_,0,,x,,,,,] [_,x,,,x,,,,,,]	[_,_,_,,,X,_,,_,] [_,_,_,0,,,,,,,] [_,,_,0,X,0,,,,,] [_,0,,X,,X,,,,] [_,X,,,X,,,,,]	[,0,.X,.,.,] [.,.,0,.,.,.] [.,.,0,.,.,.] [.,.,0,X,0,.,.,] [.,0,X,0,.,.,] [.,0,.X,.,1,.,] [.,1,0,.,1,1,1,1,1]
[_,_,_,,,,,_,_,] [_,,_,,,,,,,,,,,,,] [_,,,,,0,,,,,,,,,,] [_,,,0,,,,,,,,,,,] [_,,0,,,,,,,,,,,,,] [_,,0,,,,,,,,,,,,,,] [_,,1,,,,,,,,,,,,,,,] [_,,1,,,,,,,,,,,,,,,] [_,,1,,,,,,,,,,,,,,,]	[_,_,_,,_,,_,_,_] [_,_,_,0,,,,,,] [_,,0,X,0,,,,,] [_,0,X,0,,,,,] [_,0,X,1,,,,] [_,0,1,X,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1	[_,_,_,,,,,,_,,,,,,] [_,,_,,0,,,,,,,,] [_,,0,X,0,,,,,,] [_,0,X,0,,,,,,] [_,0,X,X,X,,,,] [_,X,,,X,,,,,] [_,,,,0,,,,,,]	[,0,.X,.,.,] [.,.,0,.,.,.] [.,.,0,.,.,.] [.,.,0,X,0,.,.,] [.,0,X,0,.,.,] [.,0,.X,.X,.,.] [.,0,.X,.,1,0,.,.] [.,0,.,0,.,0,.,.]
[_,_,_,,,,,_,_,] [_,,_,,0,,,,,,,] [_,,,0,X,,,,,,,] [_,,0,X,,,,,,,] [_,0,X,,,,,,,] [_,1,0,X,,,,,,,,] [_,2,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1	[_,_,_,,_,_,_,_,_] [_,,_,,0,,,,,,,] [_,,0,X,0,,,,,] [_,0,X,0,,,,,] [_,0,X,0,,,,,] [_,0,X,0,,,,,] [_,0,1,0,1,0,1,0,1,0]\\[,0,1,0,1,0,1,0,1,0,1,0]\\[,0,1,0,1,0,1,0,1,0]\\[,0,1,0,1,0,1,0,1,0]\\[,0,1,0,1,0,1,0,1,0]\\[,0,1,0,1,0,1,0,1,0]\\[,0,1,0,1,0,1,0]\\[,0,1,0,1,0,1,0]\\[,0,1,0,1,0,1,0]\\[,0,1,0]\\[,0,1,0	[_,_,_,,,,,,_,,,,,,] [_,,_,,0,,,,,,,,,] [_,,0,X,0,,,,,,] [_,0,X,0,,,,,,] [,x,,,X,,X,,,,] [,,,,,0,,,,,,,] [,,,,,0,,,,,,] [,,,,,0,,,,,,]	[,,,,0,,X,,,,,] [,,,,0,,,,,,,] [,,,,0,,,,,,,] [,,,0,X,0,,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
[_,_,_,,X,_,_,_] [_,_,_,0,X,_,_,_] [_,,_,0,X,_,_,_] [_,0,X,_,,_,] [_,X,_,,2,] [_,X,_,2,,2,] [_,2,2,2,2,2,2,2] [_,2,2,2,2,2,2,2,2] [_,2,2,2,2,2,2,2,2,2] [_,2,2,2,2,2,2,2,2,2] [_,2,2,2,2,2,2,2,2,2,2]	[_,_,_,,,,,,,,,,,] [_,,_,,0,X,0,_,,,,,] [_,,,0,X,0,_,,,,] [_,X,,,,,,,,,] [,,X,,,,,,,,,] [,,2,,2,,2,,,,,] [,,2,,2,,2,,,,,] [,,2,,2,,2,,,,,,] [,,2,,2,,2,,2,,,,] [,,2,,2,,2,,2,,2,,,] [,,2,,2,,2,,2,,2,,2,,2,]	[_,_,_,,,,,,,,,,,,] [_,,,,,0,,,,,,,,] [_,,,0,,X,0,,,,,,] [_,X,,,X,,,,,,] [,,2,,,0,,,,,,,,,,] [,,2,,,0,,,,,,,,] [,,2,,1,0,,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,	[,,,,,,,,,,,,] [,,,,,,,,,,,,,] [,,,,,,,,
[_,_,_,,,,,,,,,,,,] [_,,,,,0,,,,,,,,,,] [_,,,,0,X,,,,,,,,,] [_,,0,,X,,,,,,,,] [_,X,,,,X,,,,,,,,] [_,,2,,2,,,,,,,,,] [_,,2,,2,,2,,,,,,,] [_,,2,,2,,2,,,,,,,,,,,] [_,,2,,2,,2,,,,,,,,,,,,] [_,,2,,2,,2,,,,,,,,,,,,,,] [_,,2,,2,,2,,,,,,,,,,,,,] [_,,2,,2,,2,,2,,,,,,,,,,,]	[_,_,_,,,,,,,,,,,] [_,_,_,0,X,0,_,,,,,] [_,,0,X,0,_,,,,] [_,X,,,X,,,,,,] [_,X,,,,,,,,,] [_,2,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [_,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,	[_,_,_,,,,,,,,,,,,] [_,,,,,,,,,,,,,,,,,,	[,,,,,,,,,,,,] [,,,,,0,,X,,,,,,] [,,,,0,X,0,,,,,,] [,,,,0,X,0,,,,,,] [,,0,,X,,X,,,,,] [,,0,,X,,X,,,,,] [,,0,,,X,,,,,,,] [,,0,,,0,,,,,,,,] [,,,,,,0,,,,,,,] [,,,,,,,,,,
[_,_,_,,,,,,,,,,,] [_,_,,,0,,,,,,,,,,] [_,,0,,X,,,,,,,,] [_,X,,,,X,,,,,,,] [,,2,,2,,,,,,,,,,,] [,,2,,2,,2,,,,,,,,,,	[_,_,_,,,,,,,,,,] [_,,,,,0,,,,,,,,] [_,,,0,,X,0,,,,,,] [_,,0,,X,,,,,,] [_,X,,,X,,,,,,] [_,2,,0,,2,,,,,] [_,2,,2,0,,2,,,] [_,2,2,0,0,2,2,1,2,1] [_,2,2,2,2,2,1] [_,2,2,2,2,2,2,2,2] [_,2,2,2,2,2,2,2,2] [_,2,2,2,2,2,2,2,2,2] [_,2,2,2,2,2,2,2,2,2] [_,2,2,2,2,2,2,2,2,2]	[_,_,_,,,,,,,,,,,,] [_,_,,,,,,,,,,,,,,,,	[,,,,0,,X,,,,,,] [,,,,0,,X,,,,,] [,,,,0,,X,,,,,] [,,,,0,,X,,,,,] [,,0,,X,,X,,,,] [,,0,,X,,1,,,,] [,,0,,X,,1,,,,] [,,0,,1,,1,,1,] [,,0,,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
[_,_,_,,,,,,,,,,] [_,_,,,0,,,,,,,,] [_,,0,,X,,,,,,,] [_,X,,,X,,,,,,,] [,X,,,X,,,,,,,] [,Y,,Y,,Y,,,,,,] [,Y,,Y,,Y,,,,,,] [,Y,,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,	[_,_,_,,_,,_,_,_,] [_,_,_,0,,0,_,,_,,] [_,,0,,X,0,_,,_,] [_,X,,,X,,,,,,] [_,Y,,,0,,X,,,,,,] [_,Y,,,0,,X,,,,,,] [_,Y,,1,0,,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1	[_,_,_,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	[,,,,0,,X,,,,,,] [,,,,0,,X,,,,,] [,,,,0,X,0,,,,,] [,,,0,X,0,,,,,] [,X,,,X,,,,,,] [,X,,,0,,,,,,,,] [,,,,,0,,,,,,,] [,,,,,0,,,,,,,] [,,,,,0,,,,,,,] [,,,,,0,,,,,,,]
[_,_,_,,,,,,,,,,] [_,_,,,0,,,,,,,,] [_,,0,,X,,,,,,,] [_,0,,X,,,,,,,] [_,X,,,X,,,,,,,] [_,X,,,X,,,,,,,] [_,,,,0,,X,,,,,,] [_,,,,0,,,,,,,] [_,,,,0,,X,,,,,,] [_,,,,0,,X,,,,,,] [_,,,,X,0,,,,,,] [_,,,,X,0,,,,,,] [_,,,,X,0,,,,,,]	[_,_,_,,_,,_,_,_,] [_,_,_,0,,,,_,,] [_,_,0,,X,0,,,,,] [_,,0,,X,,,,,,] [_,,0,,X,,,,,,] [_,X,,,X,,,,,,] [_,,,0,,X,,,,,] [_,,,,0,,,,,,] [_,,,,0,,,,,,,] [_,,,,0,0,,X,,,,,,] [_,,,,X,0,,,,,,] [_,,,,X,0,,,,,,] [_,,,,X,0,,,,,,] [_,,,,X,0,,,,,,]	[_,_,_,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	[_,_,_0,_X,_,_,_] [_,_,_0,_X,_,_,_] [_,_,_0,X,0,,_,_] [_,_,0,X,0,,_,_] [,,0,,X,,X,_,] [,,0,,X,,X,,,] [,,1,0,,1,,1,] [,,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
[_,_,_,,,,,,,,,,] [_,_,,,0,,,,,,,] [_,,0,,X,,,,,,,] [_,,0,,X,,,,,,,] [_,2,0,,X,,,,,,,] [_,2,0,,X,,,,,,,] [_,2,0,,X,,,,,,,] [_,2,0,,X,,,,,,,] [_,2,0,,X,0,,,,,,] [_,2,0,,X,0,,,,,,] [_,2,0,,X,0,,,,,,]	[_,_,_,,_,,_,_,_,] [_,,_,_,,,,,_,,,,] [_,,_,0,X,0,,,,,,] [_,,0,,X,,,,,,] [_,X,,,X,,,,,,,] [_,X,,,X,,,,,,,] [_,,,,0,,X,,,,,] [_,,,,0,,,,,,,] [_,,,,0,,,X,,,,,] [_,,,,0,X,0,,,,,,] [_,,,,0,X,0,,,,,,] [_,,,,0,X,0,,,,,]	[_,_,_,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	[,,,,0,,X,,,,,,] [,,,,0,,X,,,,,] [,,,,0,X,0,,,,,] [,,,0,X,0,,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
[_,_,_,,,,,,,,,,,,,] [_,,,,,0,,,,,,,,,,] [_,,,0,,X,,,,,,,,] [_,,0,,X,,,,,,,,] [,,0,,X,,,,,,,,] [,,0,,X,,,,,,,,] [,,0,,X,,,,,,,] [,,0,,X,,,,,,,] [,,0,,X,,,,,,,] [,,0,,1,,0,,1,,,,] [,,1,0,,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,	[_,_,_,,,,,,,,,,,,] [_,,,,,0,X,0,,,,,,,] [_,,,,0,X,0,,,,,,,] [_,,0,,X,,,,,,,] [_,X,,,,,,,,,] [,,X,,,,,,,,,] [,,2,,2,,,,,,,,,,] [,,2,,2,,2,,,,,,,,] [,,2,,2,,2,,,,,,,,] [,,2,,2,,2,,,,,,,,,,	[_,_,_,,,,,,,,,,,,] [_,,,,,0,,,,,,,,] [_,,0,,X,0,,,,,,] [_,X,,,X,,,,,,] [,,X,,,X,,,,,] [,,X,,,,,,,,,] [,,,,,0,,,,,,,] [,,,,,,,,,,,,	[,,,,,,,,,,,,] [,,,,,0,,X,,,,,,] [,,,,,0,X,0,,,,,,] [,,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,] [,,0,,X,,X,,X,,]]
[_,_,_,,,,,,,,,,,,] [_,,,,,0,,,,,,,,,,] [_,,,,0,,,,,,,,,,] [_,,0,,,,,,,,,,,] [_,0,,,,,,,,,,,] [,,0,,,,,,,,,,,] [,,0,,,,,,,,,,	[,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	[_,_,_,_,,_,,_,_,_] [_,_,_,0,,,,,,,] [_,,0,,X,0,,,,,] [,,0,,X,,X,,,] [,,0,,X,,X,,,] [,,0,,X,,X,,,] [,,0,,X,,X,,] [,,0,0,,,1,,] [,,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,	[,,,,,,,,,,,,,] [,,,,,,,,,,,,,,,] [,,,,,,,,
[_,_,_,,,,,,,,,,,,] [_,,,,,0,,,,,,,,,,,,] [_,,,0,,X,,,,,,,,,] [_,,0,,X,,,,,,,,,] [_,,0,,X,,,,,,,,] [,,0,,X,,,,,,,,] [,,0,,X,,,,,,,] [,,0,,X,,,,,,,] [,,0,,X,,,,,,,] [,,0,,X,0,,,,,,] [,,1,1,0,,X,0,,,,,,] [,,1,1,0,,X,0,,,,,,] [,,1,1,0,,X,0,,,,,,] [,,1,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,] [,,1,0,,X,0,,,,,,]	[_,_,_,,_,,_,_,_,_] [_,_,_,0,X,0,_,,,,] [_,0,,X,,,,,] [_,0,,X,,,,,] [_,X,,,X,,,,,] [_,X,,,X,,,,,] [,,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,	[_,_,_,_,,_,_,_,_] [_,_,_,0,,,,,,,] [_,,0,,X,0,,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,] [,,1,0,0,,1,1,] [,,1,1,0,0,1,1,1,1] [,,1,1,0,0,1,1,1,1] [,,0,0,1,1,1,1] [,,0,0,1,1,1,1] [,,0,0,1,1,1,1] [,,0,0,1,1,1,1] [,,0,0,1,1,1,1] [,,0,0,1,1,1,1] [,,0,0,1,1,1,1] [,,0,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1,1] [,,0,1,1,1] [,,0,1,1,1] [,,0,1,1,1] [,,0,1,1,1] [,,0,1,1,1] [,,0,1,1,1] [,,0,1] [,,0,1]\\[,0	[,,,,,,,,,,,,,] [,,,,,0,,,,,,,,,] [,,,,,0,,,0,,,,,,] [,,,,0,,X,0,,,,,,] [,,0,,X,,,,,,,,] [,,0,,X,,,,,,,,,] [,,0,,X,,,,,,,,,] [,,0,,,,,,,,,,,,,] [,,0,,,,,,,,,,
[_,_,_,,,,,,,,,,,] [_,_,,,0,,,,,,,,,] [_,,0,,X,,,,,,,] [_,0,,X,,,,,,,] [_,0,,X,,,,,,,] [,,0,,X,,,,,,] [,,0,,X,,,,,,] [,,0,,X,,,,,,] [,,0,,X,,,,,,] [,,1,0,,X,,,,,] [,,0,,X,,,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,1,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,1,0,,X,,X,,,,] [,,0,,X,,X,,,,] [,,1,0,,X,,X,,,,] [,,1,0,,X,,X,,,,] [,,1,0,,X,,X,,,,] [,,1,0,,X,,X,,,,]	[_,_,_,,_,,_,_,_,] [_,_,_,0,,0,_,,,,] [_,,0,,X,,,,,,] [_,X,,,X,,,,,,] [_,X,,,X,,,,,,] [_,Y,,,X,,,,,,] [_,Y,,,Y,,,,,] [_,Y,,Y,,Y,,,,] [_,Y,,Y,,Y,,,,] [_,Y,,Y,,Y,,,,] [_,Y,,Y,Y,,Y,,,] [_,Y,Y,Y,Y,Y,Y,Y,Y] [_,Y,Y,Y,Y,Y,Y,Y,Y] [_,Y,Y,Y,Y,Y,Y,Y,Y] [_,Y,Y,Y,Y,Y,Y,Y,Y] [_,Y,Y,Y,Y,Y,Y,Y,Y] [_,Y,Y,Y,Y,Y,Y,Y,Y] [_,Y,Y,Y,Y,Y,Y,Y,Y] [_,Y,Y,Y,Y,Y,Y,Y,Y] [_,Y,Y,Y,Y,Y,Y,Y,Y,Y]	[_,_,_,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	[,,,,0,,X,,,,,,] [,,,,0,,X,,,,,,] [,,,,0,X,0,,,,,] [,,,0,X,0,,,,,] [,,0,,X,,X,,,,] [,X,,,X,,,,,,,] [,X,,,0,,,,,,,,] [,,,,,0,,,,,,,,] [,,,,,,0,,,,,,,] [,,,,,,,,,,
[_,_,_,,_,,_,_,_,] [_,_,_,0,,,,_,,,] [_,0,,X,,,,,,] [_,0,,X,,,,,,] [_,0,,X,,,,,,] [,1,1,0,,X,,,,,,] [,1,1,0,,1,1,0,] [,1,1,0,,1,1,0,1,0] [,1,1,0,1,1,0,0,1,0,0] [,1,1,0,1,1,0,0,0,0,0,0] [,1,1,0,0,1,0,0,0,0,0] [,1,1,0,0,1,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0,0,0] [,1,1,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]\\[,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,	[_,_,_,,_,,_,_,_,] [_,_,_,0,,0,_,,_,,] [_,,0,,X,,,,,,] [_,X,,,X,,,,,,] [_,Y,,,X,,,,,,] [_,Y,,,0,,Y,,,,] [_,Y,,0,,Y,,,] [_,Y,,0,,Y,,,] [_,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1	[_,_,_,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	[,,,,0,,X,,,,,,] [,,,,0,,X,,,,,,] [,,,,0,X,0,,,,,] [,,,0,X,0,,,,,] [,X,,,X,,,,,,] [,X,,,0,,,,,,,,,] [,,,,,0,,,,,,,,] [,,,,,0,,,,,,,,] [,,,,,0,,,,,,,,] [,,,,,0,,,,,,,,] [,,,,,0,,,,,,,,] [,,,,,0,,,,,,,,]
[_,_,_,,,,,,,,,,,] [_,_,,,0,,,,,,,,,] [_,,0,,X,,,,,,,] [_,0,,X,,,,,,,] [_,2,,0,,X,,,,,,] [,,2,,0,,X,,,,,,] [,,2,,0,,,,,,,] [,,2,,0,,,,,,,] [,,2,,0,,2,,,,,] [,,2,,0,,2,,,,,] [,,2,,0,,2,,,,,] [,,2,,0,,2,,,,] [,,2,,0,,2,,,,] [,,2,,0,,2,,,,] [,,2,,0,,2,,,,] [,,2,,0,,2,,,,] [,,2,,0,,2,,2,,] [,,2,,2,,2,,2,,] [,,2,,2,,2,,2,,] [,,2,,2,,2,,2,,] [,,2,,2,,2,,2,,] [,,2,,2,,2,,2,,] [,,2,,2,,2,,2,,] [,,2,,2,,2,,2,,2,] [,,2,,2,,2,,2,,2,] [,,2,,2,,2,,2,,2,] [,,2,,2,,2,,2,,2,] [,,2,,2,,2,,2,,2,] [,,2,,2,,2,,2,,2,,2] [,,2,,2,,2,,2,,2,] [,,2,,2,,2,,2,,2,,2] [,,2,,2,,2,,2,,2,,2] [,,2,,2,,2,,2,,2,,2] [,,2,,2,,2,,2,,2,,2] [,,2,,2,,2,,2,,2,,2] [,,2,,2,,2,,2,,2] [,,2,,2,,2,,2,,2] [,,2,,2,,2,,2] [,,2,,2,,2] [,,2,,2,,2] [,,2,,2,,2] [,,2,,2,,2] [,,2,,2,,2] [,,2,,2,,2] [,,2] [,,2	[_,_,_,,,,,,,,,,,] [_,,,,,,,,,,,,,,,,,,,	[_,_,_,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	[.,.,.,.,.,.,.] [.,.,.,0,.,X,.,.,.,] [.,.,.,0,X,0,.,.,.] [.,.,0,X,0,.,.,.] [.,.,0,,X,.,X,.,.] [.,X,.,X,.,.,.,.] [.,.,.,0,.,.,.,.] [.,.,.,0,.,.,.,.] [.,.,.,0,.,.,.,.] [.,.,.,0,.,.,.,.] [.,.,.,0,.,.,.,.]

2.3 Main.hs

Method gameLoop turns on the game by repeatedly calling method moveAI until method checkWin no longer returns Empty, i.e. one player wins.

3 Performance

We compare performance on one core and that on four cores. We find that running time on one core is 1.44s and 0.94 on four cores.



Figure 1: Running on one core



Figure 2: Running on four cores

4 Codes

4.1 Main.hs

```
1 module Main where
2
3 import
                    ΑI
4 import
                    Board
5 import
                    Data.<mark>Char</mark>
6
7 gameLoop :: Board -> Color -> [Board] -> IO ()
8 gameLoop board color list
    | null curPoint = putStrLn "Tie"
9
    | checkWin (head curPoint) curBoard == Empty = do
10
      putStrLn (show curBoard ++ "\n")
11
      gameLoop curBoard (oppositeColor color) (list ++ [curBoard])
12
    | otherwise = do
13
    putStrLn (show curBoard ++ "\n")
14
      putStrLn (show color ++ "wins")
15
16
   where
      curPoint = getCurPoint board curBoard
17
      curBoard = moveAI board color
18
19
20 main :: IO ()
21 main = gameLoop initBoard Black []
```

Listing 1: Main.hs

4.2 Board.hs

```
1 module Board
   ( Color(..)
2
   , Point(..)
3
4
    , Board(..)
5
    , initBoard
    , oppositeColor
6
    , filterBoard
7
    , isEmptyBoard
8
    , addPoint
9
    , isValidPoint
10
    , isVacant
11
    , checkWin
12
    , getCurPoint
13
14
    ) where
15
16 import Data.List
17
18 data Color
  = Black
19
    | White
20
   | Empty
21
22 deriving (Eq)
```

```
24 instance Show Color where
   show Black = "X"
25
    show White = "0"
26
   show Empty = "_"
27
28
29 data Point =
30
    Point
      { color :: Color
31
      , position :: (Int, Int)
32
33
34
35 instance Show Point where
   show (Point color _) = show color
36
37
38 instance Eq Point where
    (Point color1 (x1, y1)) == (Point color2 (x2, y2)) = x1 == x2 && y1 ==
39
     y2 && color1 == color2
40
41 instance Ord Point where
   compare (Point _ (x1,y1)) (Point _ (x2,y2)) = compare (x1*10+y1) (x2
42
     *10+y2)
43
44 newtype Board = Board [[Point]]
45
46 instance Show Board where
   show (Board points) = intercalate "\n" $ map show points
47
48
49 instance Eq Board where
   (Board points1) == (Board points2) = points1 == points2
50
51
52 initBoard :: Board
53 initBoard = Board points
54
    where
      points = [initRow x 10 | x <- [1 .. 10]]
      initRow _ 0 = []
56
      initRow row col = initRow row (col - 1) ++ [Point Empty (row, col)]
57
58
59 getPoint :: Board -> (Int, Int) -> Point
60 getPoint (Board points) (x, y) = (points !! (x - 1)) !! (y - 1)
61
62 isValidPoint :: Point -> Bool
63 isValidPoint (Point _ (x, y))
   | x > 0 && x <= 10 && y > 0 && y <= 10 = True
64
    | otherwise = False
65
66
67 isVacant :: Point -> Board -> Bool
68 isVacant (Point color (x, y)) (Board points) = curColor == Empty
69
    where
      (Point curColor (_, _)) = getPoint (Board points) (x, y)
70
71
72 addPoint :: Board -> Color -> Int -> Int -> Board
73 addPoint (Board points) color x y
```

23

```
| isValidPoint (Point color (x, y)) && isVacant (Point color (x, y)) (
74
      Board points) =
       add (Point color (x, y)) (Board points)
75
76
     | otherwise = Board points
77
78 add :: Point -> Board -> Board
79 add (Point color (newx, newy)) (Board points) = Board newPoints
80
     where
       newPoints = upperRows ++ (leftCells ++ (Point color (newx, newy) :
81
      rightCells)) : lowerRows
       (upperRows, thisRow:lowerRows) = splitAt (newx - 1) points
82
       (leftCells, _:rightCells) = splitAt (newy - 1) thisRow
83
84
85 checkWin :: Point -> Board -> Color
86 checkWin (Point color (x, y)) (Board points)
     | winRow (Point color (x, y)) (Board points) /= 0 ||
87
         (winCol (Point color (x, y)) (Board points) /= 0) ||
88
         (winDiag (Point color (x, y)) (Board points) /= 0) || (winAntiDiag
89
       (Point color (x, y)) (Board points) /= 0) =
       color
90
     | otherwise = Empty
91
92
93 checkRow :: [Point] -> Color -> Int -> Int
94 checkRow [] preColor cnt =
    if cnt == 5
95
       then if preColor == Black
96
97
              then 1
              else 2
98
       else O
99
100 checkRow (head:xs) preColor cnt
     | preColor == Empty = checkRow xs color 1
101
     | preColor == color && cnt < 4 = checkRow xs color (cnt + 1)
     | preColor == color && cnt == 4 =
103
104
       if color == Black
         then 1
106
         else 2
     | otherwise = 0
107
     where
108
       (Point color _) = head
109
110
111 getDiag :: Board -> Board
112 getDiag (Board points) = Board $ diagonals points
113
114 getAntiDiag :: Board -> Board
115 getAntiDiag (Board points) = Board $ diagonals ((transpose . reverse)
      points)
116
117 diagonals :: [[a]] -> [[a]]
118 diagonals = tail . go []
     where
119
       go b es_ =
120
         [h | h:_ <- b] :
121
122
         case es_ of
```

```
-> transpose ts
           []
           e:es -> go (e : ts) es
124
         where
           ts = [t | _:t <- b]
126
127
128 winRow :: Point -> Board -> Int
129 winRow (Point color (x, y)) (Board points) = checkRow (newPoints !! (x -
      1)) Empty 1
     where
130
       Board newPoints = addPoint (Board points) color x y
131
133 winCol :: Point -> Board -> Int
134 winCol (Point color (x, y)) (Board points) = checkRow ((transpose .
       reverse) newPoints !! (y - 1)) Empty 1
135
     where
       Board newPoints = addPoint (Board points) color x y
136
138 winDiag :: Point -> Board -> Int
139 winDiag (Point color (x, y)) (Board points) = checkRow (diagonals
      newPoints !! (x + y - 2)) Empty 1
140
     where
       Board newPoints = addPoint (Board points) color x y
141
142
143 winAntiDiag :: Point -> Board -> Int
144 winAntiDiag (Point color (x, y)) (Board points) =
     checkRow (diagonals ((transpose . reverse) newPoints) !! (9 - x + y))
145
      Empty 1
     where
146
       Board newPoints = addPoint (Board points) color x y
147
148
149 isEmptyBoard :: Board -> Bool
150 isEmptyBoard (Board points) = Board points == initBoard
151
152 oppositeColor :: Color -> Color
153 oppositeColor color
     | color == White = Black
154
     | color == Black = White
     | otherwise = error "Invalid opposite color"
156
158 filterBoard :: Board -> Color -> [Point]
159 filterBoard (Board points) color =
     [p | rows <- points, p <- rows, isSameColor p]</pre>
160
     where
161
       isSameColor (Point c (_,_)) = c == color
162
163
164 flatten :: [[a]] -> [a]
165 flatten xs = (z n \rightarrow foldr (flip (foldr z)) n xs) (:) []
166
167 getCurPoint :: Board -> Board -> [Point]
168 getCurPoint (Board points1) (Board points2) = flatten points2 \\ flatten
      points1
```

Listing 2: Board.hs

4.3 Al.hs

```
1 module AI
    ( moveAI
2
    ) where
3
4
5 import
                    Board
6 import
                    Control.Parallel.Strategies
7 import
                    Data.List
8 import
                    Data.Maybe
9 import qualified Data.Set
                                                  as Set
10 import
                    Data.Tree
11
12 minInt :: Int
13 minInt = -(2^{2})
14
15 maxInt :: Int
16 \text{ maxInt} = 2 29 - 1
17
18 moveAI :: Board -> Color -> Board
19 moveAI board color
20
    | isEmptyBoard board = addPoint board color 1 1
21
    otherwise = bestMove
22
    where
       neighbors = possibleMoves board
23
       (Node node children) = buildTree color board neighbors
24
      minmax = parMap rdeepseq (minmaxBeta color 3 minInt maxInt) children
25
      index = fromJust $ elemIndex (maximum minmax) minmax
26
       (Node bestMove _) = children !! index
27
28
29 buildTree :: Color -> Board -> [Point] -> Tree Board
30 buildTree color board neighbors = Node board $ children neighbors
31
    where
      newNeighbors point =
32
        Set.toList $
33
         Set.union (Set.fromList (Data.List.delete point neighbors)) (Set.
34
      fromList (stepFromPoint board point))
      oppoColor = oppositeColor color
35
       children [] = []
36
       children (Point c (x, y):ns) =
37
        buildTree oppoColor (addPoint board color x y) (newNeighbors (Point
38
       c (x, y))) : children ns
39
40 minmaxAlpha :: Color -> Int -> Int -> Int -> Tree Board -> Int
41 minmaxAlpha _ _ alpha _ (Node _ []) = alpha
42 minmaxAlpha color level alpha beta (Node b (x:xs))
    | level == 0 = curScore
43
    | canFinish curScore = curScore
44
    | newAlpha >= beta = beta
45
    | otherwise = minmaxAlpha color level newAlpha beta (Node b xs)
46
47
    where
      curScore = scoreBoard b color
48
   canFinish score = score > 100000 || score < (-100000)
49
```

```
newAlpha = maximum [alpha, minmaxBeta color (level - 1) alpha beta x]
50
51
52 minmaxBeta :: Color -> Int -> Int -> Int -> Tree Board -> Int
53 minmaxBeta _ _ beta (Node _ []) = beta
54 minmaxBeta color level alpha beta (Node b (x:xs))
    | level == 0 = curScore
55
    | canFinish curScore = curScore
56
    | alpha >= newBeta = alpha
57
    | otherwise = minmaxBeta color level alpha newBeta (Node b xs)
58
    where
59
      curScore = scoreBoard b color
60
      canFinish score = score > 100000 || score < (-100000)</pre>
61
      newBeta = minimum [beta, minmaxAlpha color (level - 1) alpha beta x]
62
63
64 scoreBoard :: Board -> Color -> Int
65 scoreBoard board color = score (pointsOfColor color) - score (
      pointsOfColor $ oppositeColor color)
    where
66
      score points = sum $ map sumScores $ scoreDirections points
67
      pointsOfColor = filterBoard board
68
69
70 sumScores :: [Int] -> Int
71 \text{ sumScores } [] = 0
72 sumScores (x:xs)
    | x == 5 = 100000 + sumScores xs
73
    | x == 4 = 10000 + sumScores xs
74
    | x == 3 = 1000 + sumScores xs
75
    | x == 2 = 100 + sumScores xs
76
    otherwise = sumScores xs
77
78
79 scoreDirections :: [Point] -> [[Int]]
80 scoreDirections [] = [[0]]
81 scoreDirections ps@(point:rest) =
82
    parMap
83
      rdeepseq
84
      (scoreDirection point ps 0)
      [(xDir, yDir) | xDir <- [0 .. 1], yDir <- [-1 .. 1], not (xDir == 0
85
      && yDir == (-1)), not (xDir == 0 && yDir == 0)]
86
87 scoreDirection :: Point -> [Point] -> Int -> (Int, Int) -> [Int]
88 scoreDirection _ [] cont (_, _) = [cont]
89 scoreDirection (Point c (x, y)) ps@(Point c1 (x1, y1):rest) cont (xDir,
      yDir)
     | Point c (x, y) 'elem' ps =
90
      scoreDirection (Point c (x + xDir, y + yDir)) (Data.List.delete (
91
      Point c (x, y) ps) (cont + 1) (xDir, yDir)
     otherwise = cont : scoreDirection (Point c1 (x1, y1)) rest 1 (xDir,
92
      yDir)
93
94 possibleMoves :: Board -> [Point]
95 possibleMoves board = Set.toList $ stepBoard board $ filterBoard board
      White ++ filterBoard board Black
96
```

```
97 stepBoard :: Board -> [Point] -> Set.Set Point
98 stepBoard _ [] = Set.empty
99 stepBoard board (point:rest) = Set.union (Set.fromList (stepFromPoint
       board point)) $ stepBoard board rest
100
101 stepFromPoint :: Board -> Point -> [Point]
102 stepFromPoint board (Point _ (x, y)) =
     [ Point Empty (x + xDir, y + yDir)
103
     | xDir <- [-1 .. 1]
104
     , yDir <- [-1 .. 1]
, not (xDir == 0 && yDir == 0)
105
106
     , isValidPoint (Point Empty (x + xDir, y + yDir))
107
     , isVacant (Point Empty (x + xDir, y + yDir)) board
108
     ]
109
```

Listing 3: AI.hs

5 Reference

- 1. https://github.com/sowakarol/gomoku-haskell
- 2. https://github.com/lihongxun945/myblog/issues/14