COMP3270 Artificial Intelligence

MiniProject

Wu Tien Hsuan 2013500516

1 Introduction

- Development environment:
  - OS: Windows 10 pro 64-bit
  - IDE: Dev-C++ 5.63
  - Compiler: TDM-GCC 4.8.1 64-bit Release, g++.exe
  - CPU: Intel Core i5-4200U 2.30GHz
  - RAM: 4GB

The source code submitted (`miniproject.cpp`) is a C++ **Chinese chess** program. The minimax algorithm and α-β pruning are implemented. In the beginning, user shall specify number of steps (5 or 6) to look ahead by computer (stored in `LOOKAHEAD`) Usually, the response time for 5 steps is *1 second*, and the response time for 6 steps is *10 to 30 seconds*. The *database* stored in the program includes some typical openings of Chinese chess.

Each chess piece is represented by a character, where UPPER case is of human side and lower case is of computer side. The names and the abbreviations are displayed in table 1, and a dot '.' is used to represent an empty space.

Table 1: Abbreviations of pieces

Abbreviation	Name
G	General
A	Advisor
E	Elephant
H	Horse
R	Chariot
C	Cannon
S	Soldier

If the player wants to moves the piece at c1 to c4, enter "`c1 c4`".

2 Methodology

After user enters a valid move, the program prints out the configuration of the board and starts searching the best move with minimax algorithm and alpha-beta pruning from the root. Before returning the maximum/minimum value, we first generate all possible

moves(states), and then apply the same algorithm (generate all moves and search with minimax) to its children until the depth of search (`LOOKAHEAD`) is reached. When a node's minimum/maximum value of its children is determined, all the children nodes are deleted for memory efficiency, except for the case that the node to be deleted is one of the immediate move of current configuration.

2.1 Minimax Algorithm

Assume the opponent (i.e., human) will choose the move that maximizes the utility value. The program will therefore select the move that will lead to the largest minmum value.

2.2 Alpha-Beta Pruning

At each of the computer's move (max level), keep track of the minimum value (β) of children found so far. If a grandchild is found to have value greater than β, no need to further expand the subtree rooted at such child.

At each of the human's move (min level), keep track of the maximum value (α) of children found so far. If a grandchild is found to have value smaller than α, no need to further expand the subtree rooted at such child.

2.3 Evaluation Function and Heuristics

The program evaluates the total points of remaining pieces. The points assigned to each piece[1] are displayed in table 2.

Table 2: Values of the Pieces

Piece [p]	Points [$Points(p)$]
Soldier before crossing the river	2
Soldier after crossing the river	4
Advisor	4
Elephant	4
Horse	8
Cannon	9
Chariot	18
General	200

`int Board.val` stores the utility value calculated by `void Board::calVal()`. The utility functions is given by:

$$\sum_{p \in Computer'sRemainingPiece} Points(p) - \sum_{p \in Human'sRemainingPiece} Points(p)$$

[1] https://en.wikipedia.org/wiki/Xiangqi

2.4 Database

The program includes a built-in typical opening database[2]. The openings with the following ECCO[3] (Encyclopedia of Chinese Chess Openings) is used:

A01, A02, A05, A06, A07, A08, A40, A51, A62, D00, E10

3 Classes and Functions

3.1 Class: Piece

3.1.1 Member Variables

- `char symbol`: The abbreviation of the piece.
- `bool player`: 0 indicates that the piece belongs to computer; 1 otherwise.
- `int value`: Points assigned to the piece.
- `int locx`: X coordinate of the piece.
- `int locy`: Y coordinate of the piece.
- `alive`: 1 shows that the piece has not been captured.

3.1.2 Member Functions

- `void pos(char s, bool p, int v, int x, int y)`: To set the status of the piece. `alive` is set to 1 by default.
  - `char s`: To set `symbol` to `s`.
  - `bool p`: To set `player` to `p`.
  - `bool v`: To set `value` to `v`.
  - `int x`: To set `locx` to `x`.
  - `int y`: To set `locy` to `y`.

3.2 Class: Board

3.2.1 Member Variables

- `int** config`: The board configuration. The index number in `pieces` is stored.
- `Piece* pieces`: All 32 pieces in the board.
- `Board* lc`: Leftmost child of possible moves.
- `Board* rs`: Right sibling of possible moves.
- `int val`: Utility value as described in Section 2.3. For internal nodes, `val` will be used to store the backed-up value of a state.

[2] `https://zh.wikibooks.org/zh-tw/%E4%B8%AD%E5%9C%8B%E8%B1%A1%E6%A3%8B/%E9%96%8B%E5%B1%80`
[3] http://www.xqbase.com/ecco.htm

3.2.2 Member Functions

- `bool move(int ox, int oy, int dx, int dy, bool p)`: Set the board to the status after player `p` moves the piece at (`ox`,`oy`) to (`dx`,`dy`).
  - `return`: 1 if moved successfully; 0 otherwise.
- `bool moveHint(int ox, int oy, int dx, int dy, bool p, bool h)`:Set the board to the status after player `p` moves the piece at (`ox`,`oy`) to (`dx`,`dy`). If `h`=1, error message will be shown to user.
- `bool move(int ox, int oy, int dx, int dy, bool p)`: Create a new board and set to the status after player `p` moves the piece at (`ox`,`oy`) to (`dx`,`dy`).
  - `return`: The new board.
- `bool validMove(Piece p, int dx, int dy)`: Check whether `p` is legal to move to (`dx`,`dy`).
  - `return`: 1 if `p` is legal to move to (`dx`,`dy`); 0 otherwise.
- `void init()`: Initialize the board to starting setup.
- `void printBoard()`: Show the current board to user.
- `generateAll(int p)`: Generate all possible moves(nodes) and store the nodes as children.
  - `int p`: The player to make the next move. 0: computer; 1: user.
- `void calVal()`: Calculate utility value and update `val`.
- `Board()`: Constructor of `Board`. Both `lc` and `rs` are set to `NULL`.

3.3 Functions

- `ostream& operator<<(ostream& cout, const Piece& p)`: Overload the operator to print out `Piece`.
- `void userMove(Board *b)`: Prompt the user to enter the next move and update the board.
  - `Board* b`: The current board.
- `Board* alphaBetaSearch(Board* b)`: Use alpha-beta pruning and minimax to select the next move given the current board `b`.
- `int maxValue(Board* b, int x, int y)`: Generate all possible moves and return the maximum value of the children given the status `b`.
  - `int x`: Alpha value.
  - `int y`: Beta value.
- `int minValue(Board* b, int x, int y)`: Generate all possible moves and return the minimum value of the children given the status `b`.

- `Board* databaseMove(Board *b)`: Compare the moves with the database as described in Section 2.4 in the beginning ot the game.
  - `Board* b`: The current board.
  - `return`: A new board with corresponding move if the user move(s) matches the patterns in the database; `NULL` otherwise.
- `void printComputerMove(Board *b, Board *result)`: Print the move done by computer.
  - `Board* b`: The previous board.
  - `Board* result`: The new board.
- `Board* computerMove(Board *b)`: Decide the move made by the computer and update the board.
  - `Board* b`: The current board.
  - `return`: A new board with the move done by the computer.