Report

Introduction to programming (II) final project
111000114 Chang Jui-Hsuan

Using MiniMax + Alpha-Beta Pruning implementing AI
State Function Part:

```
int State::evaluate() {
    int value = 0;
    int now_piece, oppn_piece;
    auto self_board = this->board.board[this->player];
    auto oppn_board = this->board.board[1 - this->player];

    if(this->game_state == WIN) {
        return INT_MAX;
    } else if(this->game_state == DRAW) {
        return 0;
    }

    for (int i = 0; i < BOARD_H; i += 1) {
        for (int j = 0; j < BOARD_W; j += 1) {
            // mul a factor to make the value of different place different
            value += (i + j) * 10;

            if ((now_piece = self_board[i][j])) {
                switch (now_piece) {
                    case 1://pawn
                        value += 100;
                        break;
                    case 3://knight
                        value += 320;
                        break;
                    case 4://bishop
                        value += 330;
                        break;
                    case 2://rook
                        value += 500;
                        break;
                    case 5://queen
                        value += 900;
                        break;
                    case 6://king
                        value += 20000;
                        break;
                }
            }
```

```
            if ((oppn_piece = oppn_board[i][j])) {
                switch (oppn_piece) {
                    case 1://pawn
                        value -= 100;
                        break;
                    case 3://knight
                        value -= 320;
                        break;
                    case 4://bishop
                        value -= 330;
                        break;
                    case 2://rook
                        value -= 500;
                        break;
                    case 5://queen
                        value -= 900;
                        break;
                    case 6://king
                        value -= 20000;
                        break;
                }
            }
        }
    }

    return value;
}
```

get_move

player_0 First `max -> min` player_1 Last `min -> max`

```
MiniMax::get_move(root, -INT_MAX, INT_MAX, depth, 1 - root->player);
```

```
if (maximizing_player) {
    // when it's player, pick the largest score
    int value = -INT_MAX;
    Move move;
    for (auto action: state->legal_actions) {
      auto next_state = state->next_state(action);
      auto next_move = get_move(next_state, alpha, beta, depth - 1, false);
      if (next_move.second > value) {
        value = next_move.second;
        move = action;
      }
      alpha = max(alpha, value);
      if (beta <= alpha)
```

```
        break;
    }
    return mp(move, value);
} else {
    // when it's opponent, pick the smallest score
    int value = INT_MAX;
    Move move;
    for (auto action: state->legal_actions) {
      auto next_state = state->next_state(action);
      auto next_move = get_move(next_state, alpha, beta, depth - 1, true);
      if (next_move.second < value) {
        value = next_move.second;
        move = action;
      }
      beta = min(beta, value);
      if (beta <= alpha)
        break;
    }
    return mp(move, value);
}
```

```
std::fstream logout("log.txt", std::ios::in | std::ios::out | std::ios::app);
int depth = 3;
while (true) {
if (depth > 6)
  break;
logout << ++depth << std::endl;
logout.flush();
// Choose a random spot.
//    auto pr = Random::get_move(root, depth++);
//    auto move = pr;
auto pr = MiniMax::get_move(root, -INT_MAX, INT_MAX, depth, 1 - root->player);
auto move = pr.first;
auto value = pr.second;
fout << move.first.first << " " << move.first.second << " "
     << move.second.first << " " << move.second.second << std::endl;

// Remember to flush the output to ensure the last action is written to file.
fout.flush();
//break;
}
```