# Report

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

```
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;
    }
}
```

### 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);
}
```