IP2 Mini project3 report

姓名:蕭翔駿
學號:111062228

要完成這次的mini project，首先要先將state.cpp中的int State::evaluate()完成。基本概念就是，在盤面上的每個棋子都可以被拿來評估盤面的優劣與否，因此我們透過將每個棋子定一個分數，再將我方的棋子加分敵方的棋子減分，來將盤面的優劣數值化，方便此AI思考要怎麼走下一步。後來精進evaluate() function，由於每種棋子在棋盤上每個位置的效益不同，於是又增加了能依據棋子位置額外增減分的功能。

```
int State_Value = 0;
const int Chess_Point[7] = {0,4,20,12,12,36,100000};
for (int i = 0;i < BOARD_H;i++){
  for (int j= 0;j < BOARD_W;j++){
    int WhiteBlock = board.board[0][i][j];
    int BlackBlock = board.board[1][i][j];
    int WhiteBouns = bouns[WhiteBlock][i][j];
    int BlackBouns = bouns[BlackBlock][5-i][4-j];
    //for (auto action = legal_actions.begin();action != legal_actions.end();action++){
      //State_Value += 1;       You, 1 小時前 • depth 5 to 6 no table …
    //}
    if (player == 0){
      //if (WhiteBlock == 1){
        //if (i - 1 >= 0 && board.board[0][i-1][j] == '1' ) State_Value -= 2;
      //}
      State_Value += (Chess_Point[WhiteBlock] + WhiteBouns + 12);
      State_Value -= (Chess_Point[BlackBlock] + BlackBouns + 4);
    }else if (player == 1){
      //if (BlackBlock == 1){
        //if (i - 1 >= 0 && board.board[1][i-1][j] == '1' ) State_Value -= 2;
      //}
      State_Value += (Chess_Point[BlackBlock] + BlackBouns + 4);
      State_Value -= (Chess_Point[WhiteBlock] + WhiteBouns + 12);
    }
  }
}
return State_Value;
```

再來要完成MiniMax的部分，利用遞迴搜索每個深度的所有棋子移動的可能。code的思路是假設我下的每步都會是最好的，而對手也會選擇對他自己最好的一手。最後從中選擇能創造最好棋路的一手來作為下一手。
首先是get_move的部分，此函式是為了為了能傳回最佳的move。為了能利用遞迴透過數值挑選最優的盤面值，又另外寫了get_evaluate的函式，傳回值是int方便利用遞迴比較每個深度，一直往下搜索每層，直到沒有步數可以走或深度==0時會開始傳回evaluate()的值。

```cpp
Move MiniMax::get_move(State *state, int depth){
  if(!state->legal_actions.size())
    state->get_legal_actions();

  Move BestStep;
  int BestValue = -100000000;
  for (auto it : state->legal_actions){
    State* next_S = (*state).next_state(it);
    int NowCal = get_evaluate(next_S,depth -1,false);
    if (NowCal > BestValue){
      BestValue = NowCal;
      BestStep = it;
    }
    delete next_S;
  }
  return BestStep;
}
```

```cpp
int MiniMax::get_evaluate(State *state2, int depth2,bool me){
  if(!state2->legal_actions.size())
    state2->get_legal_actions();

  if (depth2 == 0){      You, 前天 • FirstWinBaseline1 …
    if (me){
      return (*state2).evaluate();
    }else {
      return (*state2).evaluate() * (-1);
    }
  }else if (me){
    int CalBest = -100000000;
    for (auto it2 : state2->legal_actions){
      int Next_Val = get_evaluate((*state2).next_state(it2),depth2 -1,false);
      CalBest = MAX(CalBest,Next_Val);
    }
    return CalBest;
  }else {
    int CalBest = 100000000;
    for (auto it3 : state2->legal_actions){
      int Next_Val = get_evaluate((*state2).next_state(it3),depth2 -1,true);
      CalBest = MIN(CalBest,Next_Val);
    }
    return CalBest;
  }
}
```

最後是Alpha-Beta Pruning的部分，簡單來說是優化MiniMax。α會記錄我們目前搜索完的的最優解，而β則記錄對方選擇的最優解盤面值，若α>=β則代表我們選的最優解已經不可能被選了，就可以break，停止搜索這個節點，來減少不必要的搜索時間。
與MiniMax一樣，get_move的部分，是為了為了能傳回最佳的move。再另外透過get_alphabeta來遞迴計算每個深度的最佳α，β。函式中添加了更多細節，不只沒有步數可以走或深度==0時會開始傳回evaluate()的值，當找到有人贏的情況時也會回傳值，當我方贏時傳回一個極大值，代表著最優解，敵方贏時，則傳回極小值，代表著對我方最差的情況。

```
Move AlphaBeta::get_move(State *state, int depth){

  if(state->legal_actions.empty())
    state->get_legal_actions();

  //stored_state state_table
  Move BestStep;
  int val = -100000;
  for (const auto &it : state->legal_actions){
    State* next_S = state->next_state(it);
    int AB_culculate = get_alphabeta(next_S,depth -1,false,-100000,100000);
    if (AB_culculate > val){
      val = AB_culculate;
      BestStep = it;
    }
  }
  return BestStep;
}
```

```cpp
int AlphaBeta::get_alphabeta(State *state, int depth,bool me,int alpha,int beta){
  //std::string table_key = state->encode_state();
  //if (state_table.count(table_key)){
  //  return state_table[table_key];
  //}
  if(state->legal_actions.empty()) state->get_legal_actions();
  if (depth == 0 || state->legal_actions.empty()){
    if (me){
      int state_val = state->evaluate();
      //state_table[table_key] = state_val;
      return state_val;
    }else {
      int state_val = -state->evaluate();
      //state_table[table_key] = state_val;
      return state_val;
    }
  }
  //return me ? state->evaluate() : -state->evaluate();
  if(state->game_state == WIN) {
    if (me) {
      //state_table[table_key] =  INT32_MAX;
      return INT32_MAX;
    }else {
      //state_table[table_key] = INT32_MIN;
      return INT32_MIN;
    }
  }
```

```cpp
if (me){
  int value = -100000;
  for (const auto& it : state->legal_actions){
    State* NewState = state->next_state(it);
    value = std::max(value,get_alphabeta(NewState, depth - 1, false, alpha, beta));
    alpha = std::max(value,alpha);
    if (alpha >= beta) break;
  }
  //state_table[table_key] = value;
  return value;
}else {
  int value = 100000;
  for (const auto& it : state->legal_actions){
    State* NewState = state->next_state(it);
    value = std::min(value, get_alphabeta(NewState, depth - 1, true, alpha, beta));
    beta = std::min(value,beta);
    if (alpha >=beta) break;
  }
  //state_table[table_key] = value;
  return value;
}
}
```

(bool)true