清華學院學士班 陳冠宇

Mini3 demo報告

我的evaluate分為兩段

```
14  int State::evaluate(){
15      // [TODO] design your own evaluation function
16      // std::cout << "evaluate" << std::endl;
17      int white = countMaterial(0);
18      int black = countMaterial(1);
19      int evaluation = white - black;
20      return evaluation * ((player == 0) ? -1 : 1);
21  }
```

所謂下個階段的分數其實是敵方視角的
所以對先手來說後面要乘上負號

至於各玩家的分數則是套用同個計算func

```
23  int State::countMaterial(int color){
24      int material = 0;
25      for(int i=0; i<BOARD_H; i+=1){
26          for(int j=0; j<BOARD_W; j+=1){
27              switch (board.board[color][i][j]){
28              case 1:
29                  material += 100;
30                  break;
31              case 2:
32                  material += 500;
33                  break;
34              case 3:
35                  material += 300;
36                  break;
37              case 4:
38                  material += 300;
39                  break;
40              case 5:
41                  material += 900;
42                  break;
43              case 6: // king
44                  material += 10000;
45                  break;
46              default:
47                  break;
48              };
49          }
50      }
51      return material;
```

```
15  Move Player::get_move(State *state, int depth){
16      if(!state->legal_actions.size())
17      state->get_legal_actions();
18      int num = 0;
19      auto actions = state->legal_actions;
20      int maxValue = -100000;
21      int minValue = 100000;
22      if((state->player == 0)){
23        for(unsigned long i = 0; i < actions.size(); i++){
24          int nextValue = choose_move(state->next_state(actions[i]), depth, -10000, 10000, false, 0);
25          if(nextValue > maxValue){  // tmp > maxValue
26            num = i;
27            maxValue = nextValue;
28          }
29        }
30      }
31      else if(state->player == 1){
32        for(unsigned long i = 0; i < actions.size(); i++){
33          int nextValue = choose_move(state->next_state(actions[i]), depth, -10000, 10000, true, 1);
34          if(nextValue < minValue){  // tmp > maxValue
35            num = i;
36            minValue = nextValue;
37          }
38        }
39      }
40      return actions[num];
41  }
```

這是第一個func，回傳究竟要走哪一步，他會讀他的子代的evaluation，至於詳細數值則是由下面的遞迴進行

```
15  Move Player::get_move(State *state, int depth){
16      if(!state->legal_actions.size())
17      state->get_legal_actions();
18      int num = 0;
19      auto actions = state->legal_actions;
20      int maxValue = -100000;
21      int minValue = 100000;
22      if((state->player == 0)){
23        for(unsigned long i = 0; i < actions.size(); i++){
24          int nextValue = choose_move(state->next_state(actions[i]), depth, -10000, 10000, false, 0);
25          if(nextValue > maxValue){  // tmp > maxValue
26            num = i;
27            maxValue = nextValue;
28          }
29        }
30      }
31      else if(state->player == 1){
32        for(unsigned long i = 0; i < actions.size(); i++){
33          int nextValue = choose_move(state->next_state(actions[i]), depth, -10000, 10000, true, 1);
34          if(nextValue < minValue){  // tmp > maxValue
35            num = i;
36            minValue = nextValue;
37          }
38        }
39      }
40      return actions[num];
41  }
```

這是第一個func，回傳究竟要走哪一步，他會讀他的子代的evaluation，至於詳細數值則是由下面的遞迴進行。因為前面evaluate計算的問題，都是以先手為主，所以這裡將先後手分開討論

```cpp
int Player::choose_move(State *state, int depth, int alpha, int beta, int isMinimaxPlayer, int f){
  if(!state->legal_actions.size())
    state->get_legal_actions();

  if(depth == 0)
    return state->evaluation * (f ? -1 : 1);
  auto actions = state->legal_actions;
  // player
  if(isMinimaxPlayer){
    int maxValue = -100000;
    for(unsigned long i = 0; i < actions.size(); i++){
      int nextValue = choose_move(state->next_state(actions[i]), depth - 1, alpha, beta, false, f);
      maxValue = std::max(maxValue, nextValue);
      // parameter(參數) can be modify!!!
      alpha = std::max(alpha, maxValue);
      // opponent will choose thier best way, that is, our worst way.
      // (When move was too good, opponent will avoid it)
      // We choose the best way only if it is root. And that is get_move's job.
      // Hence we can, and we should cut the better way.
      if(alpha >= beta)
        break;
    }
    return maxValue;
  }
```

這邊則是同時包含了minimax和prune的部分，maxValue和for迴圈去找最大值，去達到minimax的效果，else會放在下一頁。而關於prune的部分因為我太笨了花很多時間理解，筆記已經寫在註解裡面

```
67      else{
68          int minValue = 100000;
69          for(unsigned long i = 0; i < actions.size(); i++){
70              int nextValue = choose_move(state->next_state(actions[i]), depth - 1, alpha, beta, true, f);
71              minValue = std::min(minValue, nextValue);
72              beta = std::min(beta, minValue);
73              if(beta <= alpha)
74                  break;
75          }
76          return minValue;
77      }
78  }
```

後面用個f則是儲存，這個player的先手後手，遞迴進去只能檢測當前state，所以我就這樣存起來了。在深度為零的時候，就可以透過F去決定導出來的數值正負。

結果來說，還是沒有用就是了，一直到最後都沒找到解決辦法，debug de不出，助教太瞧得起我了，說12小時就做得完，阿巴阿巴。