# AI五子棋实验报告

报告人：杨清晨

## 基本原理

(1)概要

在博弈树上利用极大极小值算法进行搜索，搜索到一定深度，利用局面评估函数进行估值，并向上返回结果，同时根据Alpha----Beta剪枝算法进行优化。

(2)博弈树

根据落子数目进行分层，每下一子相当于从父节点向下选择一个子节点，这种结构

就是博弈树。

(3)极大极小值算法

在零和博弈中，假定对手和我方实力强劲，双方每次都能做出最优决策，以此为基

础进行推演，这种算法就是极大极小值算法。

(4)局面评估函数

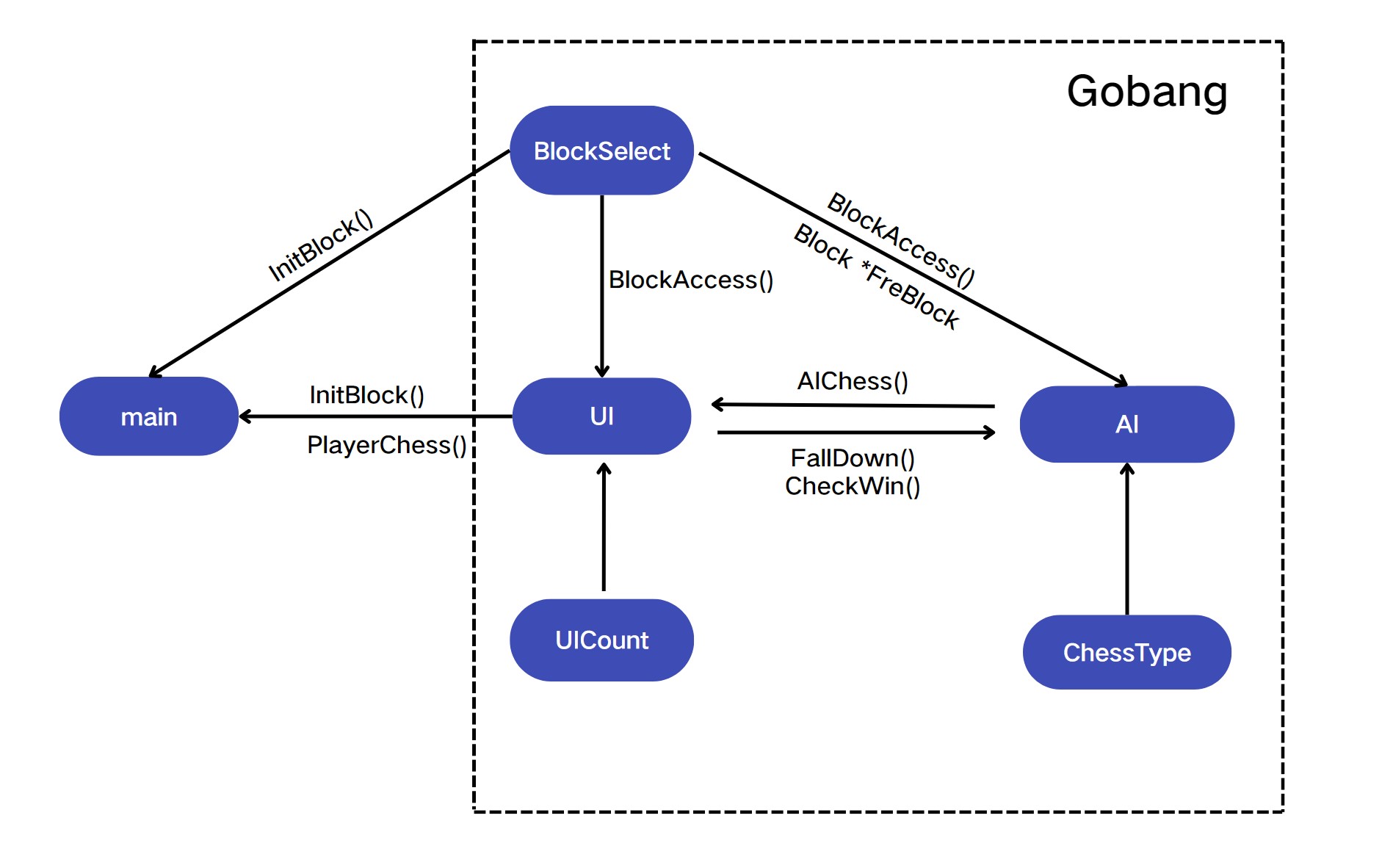
根据双方的棋型，对局面进行估分，对我方越有利，则分值越高，反之亦然。

(5)Alpha-Beta剪枝算法

基于极大极小值算法和局面评估函数，可得出某一结点的值所在的区间，若该节点的某一字节点当前已经评估出其值超出该区间，则该节点的值一定不会被选取，无需被计算，故对该节点进行剪枝。

## 算法设计

(1)基本模块展示



(2)模块简介

UI负责图形界面设计和玩家落子响应。UICount主要负责将鼠标点击的控制台坐标和棋盘坐标进行转化，以及一些细小的UI相关的计算。AI负责根据当前落子情况与玩家进行对弈。ChessType负责对各种棋型进行判断。BlockSelect通过BlockAccess()函数更新各个区域的落子情况,并将最适合的落子区域选出,置于全局变量FreBlock中。Gobang负责常用变量和常量的声明和定义和结构体的声明，其他各个模块都需要对其进行引用。main负责整体汇总到主函数。

## 代码展示

##### main.cpp

1. #include <raylib.h>
2. #include "UI.h"
3. #include "BlockSelect.h"
4. using namespace std;
5. int main()
6. {
7. InitUI();
8. InitBlock();
9. while(!WindowShouldClose()&&win==NOTWIN)
10. {
11. PlayerChess();
12. flush();
13. }
14. CloseWindow();
15. return 0;
16. }

##### Gobang.h

1. #ifndef GOBANG\_H
2. #define GOBANG\_H
3. #include <raylib.h>
4. #define PLAYER  1
5. #define AI     -1
6. #define NOTWIN  0
7. struct point
8. {
9. int x;
10. int y;
11. };
12. struct Block
13. {
14. int x;
15. int y;
16. int c;
17. int w;
18. Block \*last;
19. Block \*next;
20. };
21. const int Sqr=50;
22. const int MapSize=15;
23. const int ObWid=7;
24. const Color PlayerColor=BLACK;
25. const Color AIColor=WHITE;
26. extern int win;
27. extern Color board[MapSize+2][MapSize+2];
28. extern int time;
29. bool operator !=(Color a,Color b);
30. bool operator ==(Color a,Color b);
31. bool operator ==(point a,point b);
32. bool operator !=(point a,point b);
33. int Max(int a,int b);
34. int Min(int a,int b);
35. #endif

##### Gobang.cpp

1. #include <raylib.h>
2. #include "Gobang.h"
3. Color board[MapSize+2][MapSize+2]={};
4. int   win=NOTWIN;
5. int   time=0;
6. bool operator !=(Color a,Color b)
7. {
8. return !(a==b);
9. }
10. bool operator ==(Color a,Color b)
11. {
12. return a.a==b.a&&a.b==b.b&&a.g==b.g&&a.r==b.r;
13. }
14. bool operator ==(point a,point b)
15. {
16. return a.x==b.x&&a.y==b.y;
17. }
18. bool operator !=(point a,point b)
19. {
20. return !(a==b);
21. }
22. int Max(int a,int b)
23. {
24. if(a>b) return a;
25. else return b;
26. }
27. int Min(int a,int b)
28. {
29. if(a<b) return a;
30. else return b;
31. }

##### UI.h

1. #ifndef UI\_H
2. #define UI\_H
3. #include <raylib.h>
4. #include "Gobang.h"
5. void InitUI();
6. void PlayerChess();
7. void FallDown(point pos,Color colour);
8. void DrawChess(point pos);
9. void flush();
10. bool CheckWin(point pos);
11. #endif

##### UI.cpp

1. #include "UI.h"
2. #include <raylib.h>
3. #include "Gobang.h"
4. #include "BlockSelect.h"
5. #include "UICount.h"
6. #include "AI.h"
7. void InitUI()
8. {
9. InitWindow((Sqr+1)\*(MapSize-1)+1+Sqr\*2,(Sqr+1)\*(MapSize-1)+1+Sqr\*2,"Gobang");
10. SetTargetFPS(60);
11. for(int x=0;x<=MapSize+1;++x)
12. {
13. board[x][0]=YELLOW;
14. board[x][MapSize+1]=YELLOW;
15. board[0][x]=YELLOW;
16. board[MapSize+1][x]=YELLOW;
17. }
18. FallDown((point){MapSize/2+1,MapSize/2+1},AIColor);
19. return;
20. }
21. void PlayerChess()
22. {
23. point p;
24. if(IsMouseButtonPressed(MOUSE\_LEFT\_BUTTON))
25. {
26. p = GetClickPos();
27. if (board[p.x][p.y] == BLANK && p.x != 0)
28. {
29. FallDown(p,PlayerColor);
30. if (CheckWin(p)) {
31. win = PLAYER;
32. return;
33. }
34. AIChess();
35. }
36. }
37. }
38. void FallDown(point pos,Color colour)
39. {
40. ++time;
41. board[pos.x][pos.y] = colour;
42. BlockAccess(pos);
43. flush();
44. return;
45. }
46. void DrawChess(point pos)
47. {
48. Color colour=board[pos.x][pos.y];
49. pos.x=(pos.x-1)\*(Sqr+1)+Sqr;
50. pos.y=(pos.y-1)\*(Sqr+1)+Sqr;
51. DrawRing((Vector2){(float)pos.x,(float)pos.y},0,Sqr/4,0,360,500,colour);
52. DrawRing((Vector2){(float)pos.x,(float)pos.y},Sqr/4-1,Sqr/4,0,360,500,BLACK);
53. return;
54. }
55. void flush()
56. {
57. BeginDrawing();
58. ClearBackground(YELLOW);
59. for(int i=1,x=Sqr;i<=MapSize;++i,x+=Sqr+1)
60. {
61. DrawLine(Sqr,x,(Sqr+1)\*(MapSize-1)+Sqr,x,BLACK);
62. DrawLine(x,Sqr,x,(Sqr+1)\*(MapSize-1)+Sqr,BLACK);
63. }
64. for(int x=1;x<=MapSize;++x)
65. {
66. for(int y=1;y<=MapSize;++y)
67. {
68. if(board[x][y]!=BLANK)
69. {
70. DrawChess((point){x, y});
71. }
72. }
73. }
74. EndDrawing();
75. }
76. bool CheckWin(point pos)
77. {
78. Color colour=board[pos.x][pos.y];
79. int i=pos.x,j=pos.x;
80. for(i=pos.x;board[i-1][pos.y]==colour&&i>=2;--i);
81. for(j=pos.x;board[j+1][pos.y]==colour&&j+1<=MapSize;++j);
82. if(j-i>=4) return true;
83. for(i=pos.y;board[pos.x][i-1]==colour&&i>=2;--i);
84. for(j=pos.y;board[pos.x][j+1]==colour&&j+1<=MapSize;++j);
85. if(j-i>=4) return true;
86. for(i=0;board[pos.x-i-1][pos.y-i-1]==colour&&pos.x-i>=2&&pos.y-i>=2;++i);
87. for(j=0;board[pos.x+j+1][pos.y+j+1]==colour&&pos.x+j+1<=MapSize&&pos.y+j+1<=MapSize;++j);
88. if(i+j>=4) return true;
89. for(i=0;board[pos.x+i+1][pos.y-i-1]==colour&&pos.x+i+1<=MapSize&&pos.y-i>=2;++i);
90. for(j=0;board[pos.x-j-1][pos.y+j+1]==colour&&pos.x-j>=2&&pos.y+j+1<=MapSize;++j);
91. if(i+j>=4) return true;
92. return false;
93. }

##### UICount.h

1. #ifndef UICOUNT\_H
2. #define UICOUNT\_H
3. #include <raylib.h>
4. #include "Gobang.h"
5. int dist(point p1,point p2);
6. int NearLine(int x);
7. point BoardPos(point pos);
8. point NearPos(point mPos);
9. point GetClickPos();
10. #endif

##### UICount.cpp

1. #include <raylib.h>
2. #include "UICount.h"
3. #include "Gobang.h"
4. #include <math.h>
5. int dist(point p1,point p2)
6. {
7. return sqrt((long double)(p1.x-p2.x)\*(p1.x-p2.x)+(long double)(p1.y-p2.y)\*(p1.y-p2.y));
8. }
9. int NearLine(int x)
10. {
11. if(x<=Sqr/2||x>((Sqr+1)\*(MapSize-1)+Sqr\*3/2-1)) return 0;
12. else
13. {
14. int tem=(x-Sqr/2)/(Sqr+1);
15. x=tem\*(Sqr+1)+Sqr;
16. return x;
17. }
18. }
19. point BoardPos(point pos)
20. {
21. pos.x=(pos.x-Sqr)/(Sqr+1)+1;
22. pos.y=(pos.y-Sqr)/(Sqr+1)+1;
23. return pos;
24. }
25. point NearPos(point mPos)
26. {
27. point res = {NearLine(mPos.x), NearLine(mPos.y)};
28. return res;
29. }
30. point GetClickPos() {
31. Vector2 mPos = GetMousePosition();
32. point   nPos;
33. nPos = NearPos((point){(int)mPos.x,(int)mPos.y});
34. if (dist((point){(int)mPos.x,(int)mPos.y}, nPos) <= (Sqr / 4))
35. {
36. nPos=BoardPos(nPos);
37. } else
38. {
39. nPos.x = 0;
40. nPos.y = 0;
41. }
42. return nPos;
43. }

##### AI.h

1. #ifndef AI\_H
2. #define AI\_H
3. #define MaxLayer(a) ((a)%2==1)
4. #include <raylib.h>
5. #include "Gobang.h"
6. const int Deepest=2\*2+1;
7. const int unlim=10000;
8. const int RushThreeScore=10;
9. const int LiveTwoScore=1;
10. void AIChess();
11. int MiniMax(int depth,int alpha,int beta);
12. int value();
13. bool PlayerMustWin();
14. bool AIMustWin();
15. bool sparse(point pos);
16. bool dying(int who);
17. bool TendWin(Color colour);
18. bool OneStepWin(Color colour);
19. #endif

##### AI.cpp

1. #include "AI.h"
2. #include "Gobang.h"
3. #include "UI.h"
4. #include "ChessType.h"
5. #include <raylib.h>
6. static point p;
7. static point vital;
8. static point osw;
9. extern Block\* FreBlock;
10. void AIChess()
11. {
12. if(!sparse((point){FreBlock->x,FreBlock->y})) {*//目标区域不稀疏*
13. if(OneStepWin(AIColor))
14. {
15. p=osw;
16. }
17. else if(OneStepWin(PlayerColor))
18. {
19. p=osw;
20. }
21. else if(dying(AI))
22. {
23. p=vital;
24. }
25. else if(dying(PLAYER))
26. {
27. p=vital;
28. }
29. else{
30. MiniMax(1, -2 \* unlim, 2 \* unlim);
31. }
32. }
33. FallDown(p,AIColor);
34. if (CheckWin(p)) {
35. win = AI;
36. }
37. return;
38. }
39. int MiniMax(int depth,int alpha,int beta)
40. {
41. if(depth!=Deepest){
42. if(MaxLayer(depth)){
43. for(int x=FreBlock->x;x<=(FreBlock->x)+ObWid-1;++x) {
44. for (int y = FreBlock->y; y <= (FreBlock->y)+ObWid-1; ++y) {
45. if (board[x][y] == BLANK) {
46. int tem;
47. board[x][y] = AIColor;
48. if (!CheckWin((point) {x, y})) tem = MiniMax(depth + 1, alpha, beta);
49. else tem = unlim;
50. board[x][y] = BLANK;
51. if (tem > alpha) {
52. if (tem >= beta) return beta;
53. alpha = tem;
54. if (depth == 1) {
55. p = (point) {x, y};
56. }
57. }
58. }
59. }
60. }
61. return alpha;
62. }
63. else{
64. for(int x=FreBlock->x;x<=(FreBlock->x)+ObWid-1;++x) {
65. for (int y = FreBlock->y; y <= (FreBlock->y)+ObWid-1; ++y) {
66. if (board[x][y] == BLANK) {
67. int tem;
68. board[x][y] = PlayerColor;
69. if (!CheckWin((point) {x, y})) tem = MiniMax(depth + 1, alpha, beta);
70. else tem = -unlim;
71. board[x][y] = BLANK;
72. beta = Min(beta, tem);
73. if (beta <= alpha) return beta;
74. }
75. }
76. }
77. return beta;
78. }
79. }
80. else{
81. return value();
82. }
83. }
84. int value()
85. {
86. int val=0;
87. int a,b,c,d;
88. if(AIMustWin()) return unlim;
89. if(PlayerMustWin()) return -unlim;
90. if(MultiWinPos(PlayerColor)) return -unlim;
91. if(MultiWinPos(AIColor)) return unlim;
92. for(int x=1;x<=MapSize;++x)
93. {
94. for(int y=1;y<=MapSize;++y)
95. {
96. a=RushThree((point){x,y},AIColor);
97. b=RushThree((point){x,y},AIColor);
98. c=LiveTwo((point){x,y},AIColor);
99. d=LiveTwo((point){x,y},PlayerColor);
100. val+=(a-b)\*RushThreeScore;
101. val+=(c-d)\*LiveTwoScore;
102. if(a!=0&&c!=0) val+=RushThreeScore\*10;
103. if(b!=0&&d!=0) val-=RushThreeScore\*10;
104. if(a>1) val+=RushThreeScore\*5;
105. if(c>1) val+=LiveTwoScore\*5;
106. if(b>1) val-=RushThreeScore\*5;
107. if(d>1) val-=LiveTwoScore\*5;
108. }
109. }
110. return val;
111. }
112. bool AIMustWin()
113. {
114. for(int x=1;x<=MapSize;++x)
115. {
116. for(int y=1;y<=MapSize;++y)
117. {
118. if(MultiThree((point){x,y},AIColor)) return true;
119. if(UsefulFour((point){x,y},AIColor)) return true;
120. if(LiveThree((point){x,y},AIColor)) return true;
121. }
122. }
123. return false;
124. }
125. bool PlayerMustWin()
126. {
127. for(int x=1;x<=MapSize;++x)
128. {
129. for(int y=1;y<=MapSize;++y)
130. {
131. if(MultiThree((point){x,y},PlayerColor)) return true;
132. if(LiveFour((point){x,y},PlayerColor)) return true;
133. }
134. }
135. return false;
136. }
137. bool sparse(point pos)
138. {
139. int plNum=0,AINum=0,total=ObWid+ObWid+1;
140. bool flag=true;
141. total\*=total;
142. p=(point){0,0};
143. for(int x=FreBlock->x;x<=(FreBlock->x)+ObWid-1;++x) {
144. for (int y = FreBlock->y; y <= (FreBlock->y)+ObWid-1; ++y) {
145. if(board[x][y]==PlayerColor) plNum++;
146. if(board[x][y]==AIColor)
147. {
148. AINum++;
149. if(board[x+1][y]==BLANK) p=(point){x+1,y};
150. if(board[x-1][y]==BLANK) p=(point){x-1,y};
151. if(board[x][y+1]==BLANK) p=(point){x,y+1};
152. if(board[x][y-1]==BLANK) p=(point){x,y-1};
153. }
154. if(board[x][y]==BLANK&&p==(point){0,0}) p=(point){x,y};
155. if(LiveThree((point){x,y},AIColor)) flag=false;
156. if(LiveThree((point){x,y},PlayerColor)) flag=false;
157. if(UsefulFour((point){x,y},AIColor)) flag=true;
158. if(UsefulFour((point){x,y},PlayerColor)) flag=true;
159. if(!flag) break;
160. }
161. if(!flag) break;
162. }
163. if(AINum<=1&&plNum<=1&&(AINum+plNum)<=total/5&&flag) return true;
164. else return false;
165. }
166. bool dying(int who)
167. {
168. Color me,opp;
169. vital=(point){0,0};
170. bool flag;
171. if(who==AI)
172. {
173. me=AIColor;
174. opp=PlayerColor;
175. }
176. else
177. {
178. me=PlayerColor;
179. opp=AIColor;
180. }
181. for(int x=1;x<=MapSize;++x)
182. {
183. for(int y=1;y<=MapSize;++y)
184. {
185. if(board[x][y]==BLANK) {
186. board[x][y] = me;
187. if (OneStepWin(opp)) flag = true;
188. else if(TendWin(opp)) flag=true;
189. else if(MultiWinPos(opp)) flag=true;
190. else {
191. vital = (point) {x, y};
192. }
193. board[x][y] = BLANK;
194. if (vital != (point) {0, 0} && flag) return true;
195. }
196. }
197. }
198. return false;
199. }
200. bool TendWin(Color colour)
201. {
202. for(int x=1;x<=MapSize;++x)
203. {
204. for(int y=1;y<=MapSize;++y)
205. {
206. int cnt=0;
207. if(board[x][y]==BLANK) {
208. board[x][y] = colour;
209. for (int a = 1; a <= MapSize; ++a) {
210. for (int b = 1; b <= MapSize; ++b) {
211. if (LiveFour((point) {a, b}, colour)) {
212. board[x][y]=BLANK;
213. return true;
214. }
215. else if (LiveThree((point) {a, b}, colour)) {
216. cnt++;
217. if (cnt == 2) break;
218. }
219. else if(CheckWin((point){x,y}))
220. {
221. cnt++;
222. if (cnt == 2) break;
223. }
224. }
225. if (cnt == 2) break;
226. }
227. board[x][y] = BLANK;
228. if (cnt == 2) {
229. return true;
230. }
231. }
232. }
233. }
234. return false;
235. }
236. bool OneStepWin(Color colour)
237. {
238. for(int x=1;x<=MapSize;++x)
239. {
240. for(int y=1;y<=MapSize;++y)
241. {
242. if(UsefulFour((point){x,y},colour))
243. {
244. osw=(point){x,y};
245. return true;
246. }
247. }
248. }
249. return false;
250. }

##### ChessType.h

1. #ifndef CHESSTYPE\_H
2. #define CHESSTYPE\_H
3. #include "Gobang.h"
4. #include <raylib.h>
5. bool UsefulFour(point pos,Color colour);
6. bool LiveFour(point pos,Color colour);
7. bool LiveThree(point pos,Color colour);
8. bool MultiThree(point pos,Color colour);
9. bool MultiWinPos(Color colour);
10. bool RushFour(point pos,Color colour);
11. int  RushThree(point pos,Color colour);
12. int  LiveTwo(point pos,Color colour);
13. #endif

##### ChessType.cpp

1. #include "Gobang.h"
2. #include "ChessType.h"
3. #include <raylib.h>
4. bool UsefulFour(point pos,Color colour)
5. {
6. int i,j,cnt;
7. if(board[pos.x][pos.y]!=BLANK) return false;
8. for(i=pos.x;board[i-1][pos.y]==colour&&i>=2;--i);
9. for(j=pos.x;board[j+1][pos.y]==colour&&j+1<=MapSize;++j);
10. if(j-i>=4) return true;
11. for(i=pos.y;board[pos.x][i-1]==colour&&i>=2;--i);
12. for(j=pos.y;board[pos.x][j+1]==colour&&j+1<=MapSize;++j);
13. if(j-i>=4) return true;
14. for(i=0;board[pos.x-i-1][pos.y-i-1]==colour&&pos.x-i>=2&&pos.y-i>=2;++i);
15. for(j=0;board[pos.x+j+1][pos.y+j+1]==colour&&pos.x+j+1<=MapSize&&pos.y+j+1<=MapSize;++j);
16. if(i+j>=4) return true;
17. for(i=0;board[pos.x+i+1][pos.y-i-1]==colour&&pos.x+i+1<=MapSize&&pos.y-i>=2;++i);
18. for(j=0;board[pos.x-j-1][pos.y+j+1]==colour&&pos.x-j>=2&&pos.y+j+1<=MapSize;++j);
19. if(i+j>=4) return true;
20. return false;
21. }
22. bool LiveFour(point pos,Color colour)
23. {
24. if(pos.x<=MapSize-5)*//水平型*
25. {
26. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y]==colour&&board[pos.x+2][pos.y]==colour&&board[pos.x+3][pos.y]==colour&&board[pos.x+4][pos.y]==colour&&board[pos.x+5][pos.y]==BLANK)
27. {
28. return true;
29. }
30. }
31. if(pos.x<=MapSize-5&&pos.y<=MapSize-5)*//单调递减型*
32. {
33. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y+1]==colour&&board[pos.x+2][pos.y+2]==colour&&board[pos.x+3][pos.y+3]==colour&&board[pos.x+4][pos.y+4]==colour&&board[pos.x+5][pos.y+5]==BLANK)
34. {
35. return true;
36. }
37. }
38. if(pos.y<=MapSize-5)*//竖直型*
39. {
40. if(board[pos.x][pos.y]==BLANK&&board[pos.x][pos.y+1]==colour&&board[pos.x][pos.y+2]==colour&&board[pos.x][pos.y+3]==colour&&board[pos.x][pos.y+4]==colour&&board[pos.x][pos.y+5]==BLANK)
41. {
42. return true;
43. }
44. }
45. if(pos.y<=MapSize-5&&pos.x>=6)
46. {
47. if(board[pos.x][pos.y]==BLANK&&board[pos.x-1][pos.y+1]==colour&&board[pos.x-2][pos.y+2]==colour&&board[pos.x-3][pos.y+3]==colour&&board[pos.x-4][pos.y+4]==colour&&board[pos.x-5][pos.y+5]==BLANK)
48. {
49. return true;
50. }
51. }
52. return false;
53. }
54. bool LiveThree(point pos,Color colour)
55. {
56. if(pos.x<=MapSize-4)*//水平型*
57. {
58. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y]==colour&&board[pos.x+2][pos.y]==colour&&board[pos.x+3][pos.y]==colour&&board[pos.x+4][pos.y]==BLANK)
59. {
60. return true;
61. }
62. }
63. if(pos.x<=MapSize-4&&pos.y<=MapSize-4)*//单调递减型*
64. {
65. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y+1]==colour&&board[pos.x+2][pos.y+2]==colour&&board[pos.x+3][pos.y+3]==colour&&board[pos.x+4][pos.y+4]==BLANK)
66. {
67. return true;
68. }
69. }
70. if(pos.y<=MapSize-4)*//竖直型*
71. {
72. if(board[pos.x][pos.y]==BLANK&&board[pos.x][pos.y+1]==colour&&board[pos.x][pos.y+2]==colour&&board[pos.x][pos.y+3]==colour&&board[pos.x][pos.y+4]==BLANK)
73. {
74. return true;
75. }
76. }
77. if(pos.y<=MapSize-4&&pos.x>=5)
78. {
79. if(board[pos.x][pos.y]==BLANK&&board[pos.x-1][pos.y+1]==colour&&board[pos.x-2][pos.y+2]==colour&&board[pos.x-3][pos.y+3]==colour&&board[pos.x-4][pos.y+4]==BLANK)
80. {
81. return true;
82. }
83. }
84. return false;
85. }
86. bool MultiThree(point pos,Color colour)
87. {
88. int i=pos.x,j=pos.x;
89. int cnt=0;
90. for(i=pos.x;board[i-1][pos.y]==colour&&i>=2;--i);
91. for(j=pos.x;board[j+1][pos.y]==colour&&j+1<=MapSize;++j);
92. if(j-i>=3&&(board[i-1][pos.y]==BLANK||board[j+1][pos.y]==BLANK)) cnt++;
93. for(i=pos.y;board[pos.x][i-1]==colour&&i>=2;--i);
94. for(j=pos.y;board[pos.x][j+1]==colour&&j+1<=MapSize;++j);
95. if(j-i>=3&&(board[pos.x][i-1]==BLANK||board[pos.x][j+1]==BLANK)) cnt++;
96. if(cnt>=2) return true;
97. for(i=0;board[pos.x-i-1][pos.y-i-1]==colour&&pos.x-i>=2&&pos.y-i>=2;++i);
98. for(j=0;board[pos.x+j+1][pos.y+j+1]==colour&&pos.x+j+1<=MapSize&&pos.y+j+1<=MapSize;++j);
99. if(i+j>=3&&(board[pos.x-i-1][pos.y-i-1]==BLANK||board[pos.x+j+1][pos.y+j+1]==BLANK)) cnt++;
100. if(cnt>=2) return true;
101. for(i=0;board[pos.x+i+1][pos.y-i-1]==colour&&pos.x+i+1<=MapSize&&pos.y-i>=2;++i);
102. for(j=0;board[pos.x-j-1][pos.y+j+1]==colour&&pos.x-j>=2&&pos.y+j+1<=MapSize;++j);
103. if(i+j>=3&&(board[pos.x+i+1][pos.y-i-1]==BLANK||board[pos.x-j-1][pos.y+j+1]==BLANK)) cnt++;
104. if(cnt>=2) return true;
105. else       return false;
106. }
107. bool MultiWinPos(Color colour)
108. {
109. int cnt=0;
110. for(int x=1;x<=MapSize;++x)
111. {
112. for(int y=1;y<=MapSize;++y)
113. {
114. if(LiveThree((point){x,y},colour)) ++cnt;
115. if(RushFour((point){x,y},colour)) ++cnt;
116. if(cnt>=2) return true;
117. }
118. }
119. return false;
120. }
121. bool RushFour(point pos,Color colour)
122. {
123. if(pos.x<=MapSize-5)*//水平型*
124. {
125. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y]==colour&&board[pos.x+2][pos.y]==colour&&board[pos.x+3][pos.y]==colour&&board[pos.x+4][pos.y]==colour&&board[pos.x+5][pos.y]!=colour&&board[pos.x+5][pos.y]!=BLANK)
126. {
127. return true;
128. }
129. }
130. if(pos.x>=6)*//anti水平型*
131. {
132. if(board[pos.x][pos.y]==BLANK&&board[pos.x-1][pos.y]==colour&&board[pos.x-2][pos.y]==colour&&board[pos.x-3][pos.y]==colour&&board[pos.x-4][pos.y]==colour&&board[pos.x-5][pos.y]!=colour&&board[pos.x-5][pos.y]!=BLANK)
133. {
134. return true;
135. }
136. }
137. if(pos.x<=MapSize-5&&pos.y<=MapSize-5)*//单调递减型*
138. {
139. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y+1]==colour&&board[pos.x+2][pos.y+2]==colour&&board[pos.x+3][pos.y+3]==colour&&board[pos.x+4][pos.y+4]==colour&&board[pos.x+5][pos.y+5]!=colour&&board[pos.x+5][pos.y+5]!=BLANK)
140. {
141. return true;
142. }
143. }
144. if(pos.x>=6&&pos.y>=6)*//anti单调递减型*
145. {
146. if(board[pos.x][pos.y]==BLANK&&board[pos.x-1][pos.y-1]==colour&&board[pos.x-2][pos.y-2]==colour&&board[pos.x-3][pos.y-3]==colour&&board[pos.x-4][pos.y-4]==colour&&board[pos.x-5][pos.y-5]!=colour&&board[pos.x-5][pos.y-5]!=BLANK)
147. {
148. return true;
149. }
150. }
151. if(pos.y<=MapSize-5)*//竖直型*
152. {
153. if(board[pos.x][pos.y]==BLANK&&board[pos.x][pos.y+1]==colour&&board[pos.x][pos.y+2]==colour&&board[pos.x][pos.y+3]==colour&&board[pos.x][pos.y+4]==colour&&board[pos.x][pos.y+5]!=colour&&board[pos.x][pos.y+5]!=BLANK)
154. {
155. return true;
156. }
157. }
158. if(pos.y>=6)*//anti竖直型*
159. {
160. if(board[pos.x][pos.y]==BLANK&&board[pos.x][pos.y-1]==colour&&board[pos.x][pos.y-2]==colour&&board[pos.x][pos.y-3]==colour&&board[pos.x][pos.y-4]==colour&&board[pos.x][pos.y-5]!=colour&&board[pos.x][pos.y-5]!=BLANK)
161. {
162. return true;
163. }
164. }
165. if(pos.y<=MapSize-5&&pos.x>=6)
166. {
167. if(board[pos.x][pos.y]==BLANK&&board[pos.x-1][pos.y+1]==colour&&board[pos.x-2][pos.y+2]==colour&&board[pos.x-3][pos.y+3]==colour&&board[pos.x-4][pos.y+4]==colour&&board[pos.x-5][pos.y+5]!=colour&&board[pos.x-5][pos.y+5]!=BLANK)
168. {
169. return true;
170. }
171. }
172. if(pos.y>=6&&pos.x<=MapSize-5)
173. {
174. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y-1]==colour&&board[pos.x+2][pos.y-2]==colour&&board[pos.x+3][pos.y-3]==colour&&board[pos.x+4][pos.y-4]==colour&&board[pos.x+5][pos.y-5]!=colour&&board[pos.x+5][pos.y-5]!=BLANK)
175. {
176. return true;
177. }
178. }
179. return false;
180. }
181. int RushThree(point pos,Color colour)
182. {
183. int cnt=0;
184. int i=0;
185. Color anti\_colour=(colour==AIColor)?PlayerColor:AIColor;
186. if(board[pos.x][pos.y]!=BLANK) return 0;
187. for(i=0;board[pos.x-i-1][pos.y]==colour;++i);*//左*
188. if(i==3&&board[pos.x-i-1][pos.y]!=BLANK) ++cnt;
189. for(i=0;board[pos.x-i-1][pos.y-i-1]==colour;++i);*//左上*
190. if(i==3&&board[pos.x-i-1][pos.y-i-1]!=BLANK) ++cnt;
191. for(i=0;board[pos.x][pos.y-i-1]==colour;++i);*//上*
192. if(i==3&&board[pos.x][pos.y-i-1]!=BLANK) ++cnt;
193. for(i=0;board[pos.x+i+1][pos.y-i-1]==colour;++i);*//右上*
194. if(i==3&&board[pos.x+i+1][pos.y-i-1]!=BLANK) ++cnt;
195. for(i=0;board[pos.x+i+1][pos.y]==colour;++i);*//右*
196. if(i==3&&board[pos.x+i+1][pos.y]!=BLANK) ++cnt;
197. for(i=0;board[pos.x+i+1][pos.y+i+1]==colour;++i);*//右下*
198. if(i==3&&board[pos.x+i+1][pos.y+i+1]!=BLANK) ++cnt;
199. for(i=0;board[pos.x][pos.y+i+1]==colour;++i);*//下*
200. if(i==3&&board[pos.x][pos.y+i+1]!=BLANK) ++cnt;
201. for(i=0;board[pos.x-i-1][pos.y+i+1]==colour;++i);*//左下*
202. if(i==3&&board[pos.x-i-1][pos.y+i+1]!=BLANK) ++cnt;
203. return cnt;
204. }
205. int LiveTwo(point pos,Color colour)
206. {
207. int cnt=0;
208. if(pos.x<=MapSize-4)*//水平型*
209. {
210. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y]==colour&&board[pos.x+2][pos.y]==colour&&board[pos.x+3][pos.y]==BLANK)
211. {
212. ++cnt;
213. }
214. }
215. if(pos.x<=MapSize-4&&pos.y<=MapSize-4)*//单调递减型*
216. {
217. if(board[pos.x][pos.y]==BLANK&&board[pos.x+1][pos.y+1]==colour&&board[pos.x+2][pos.y+2]==colour&&board[pos.x+3][pos.y+3]==BLANK)
218. {
219. ++cnt;
220. }
221. }
222. if(pos.y<=MapSize-4)*//竖直型*
223. {
224. if(board[pos.x][pos.y]==BLANK&&board[pos.x][pos.y+1]==colour&&board[pos.x][pos.y+2]==colour&&board[pos.x][pos.y+3]==BLANK)
225. {
226. ++cnt;
227. }
228. }
229. if(pos.y<=MapSize-4&&pos.x>=5)
230. {
231. if(board[pos.x][pos.y]==BLANK&&board[pos.x-1][pos.y+1]==colour&&board[pos.x-2][pos.y+2]==colour&&board[pos.x-3][pos.y+3]==BLANK)
232. {
233. ++cnt;
234. }
235. }
236. return cnt;
237. }

##### BlockSelect.h

1. #ifndef BLOCKSELECT\_H
2. #define BLOCKSELECT\_H
3. #include "Gobang.h"
4. #include <raylib.h>
5. const int AccessScore=20;
6. void   InitBlock();
7. void   BlockAccess(point pos);
8. void   BlockAdjust(Block \*blk);
9. void   TakeOut(Block \*blk);
10. void   insert(Block \*pre,Block \*blk);
11. #endif

##### BlockSelect.cpp

1. *#include <cstdlib>*
2. *#include "BlockSelect.h"*
3. *#include "Gobang.h"*
4. static Block\*  Handle[MapSize-ObWid+2][MapSize-ObWid+2];
5. Block\*  FreBlock=0;
6. void InitBlock()
7. {
8. Block \*tem=0;
9. for(int x=1;x<=MapSize-ObWid+1;++x)
10. {
11. for(int y=1;y<=MapSize-ObWid+1;++y)
12. {
13. Handle[x][y]=(Block\*)malloc(sizeof(Block));
14. if(FreBlock==0) FreBlock=Handle[x][y];
15. Handle[x][y]->x=x;
16. Handle[x][y]->y=y;
17. Handle[x][y]->c=0;
18. Handle[x][y]->w=0;
19. Handle[x][y]->last=tem;
20. Handle[x][y]->next=0;
21. if(tem) tem->next=Handle[x][y];
22. tem=Handle[x][y];
23. }
24. }
25. return;
26. }
27. void BlockAccess(point pos)
28. {
29. for(int x=Max(1,pos.x-ObWid+1);x<=Min(MapSize-ObWid+1,pos.x);++x)
30. {
31. for(int y=Max(1,pos.y-ObWid+1);y<=Min(MapSize-ObWid+1,pos.y);++y)
32. {
33. int mx=x+ObWid/2,my=y+ObWid/2;
34. if(Handle[x][y]==0) continue;
35. Handle[x][y]->w+=AccessScore-abs(mx-x)-abs(my-y)+time\*2;
36. Handle[x][y]->c++;
37. if((Handle[x][y]->c)==ObWid\*ObWid)
38. {
39. TakeOut(Handle[x][y]);
40. free(Handle[x][y]);
41. Handle[x][y]=0;
42. }
43. BlockAdjust(Handle[x][y]);
44. }
45. }
46. return;
47. }
48. void BlockAdjust(Block \*blk)
49. {
50. Block \*cur=blk->last;
51. while(cur!=NULL)
52. {
53. if((cur->w)>(blk->w)) break;
54. cur=cur->last;
55. }
56. TakeOut(blk);
57. insert(cur,blk);
58. }
59. void TakeOut(Block \*blk)
60. {
61. if(blk->last==0)
62. {
63. FreBlock=blk->next;
64. if(blk->next) blk->next->last=0;
65. blk->next=0;
66. }
67. else
68. {
69. blk->last->next=blk->next;
70. if(blk->next) blk->next->last=blk->last;
71. blk->last=0;
72. blk->next=0;
73. }
74. return;
75. }
76. void insert(Block \*pre,Block \*blk)
77. {
78. if(pre==0)
79. {
80. blk->next=FreBlock;
81. FreBlock->last=blk;
82. FreBlock=blk;
83. }
84. else
85. {
86. blk->next=pre->next;
87. pre->next->last=blk;
88. pre->next=blk;
89. blk->last=pre;
90. }
91. return;
92. }