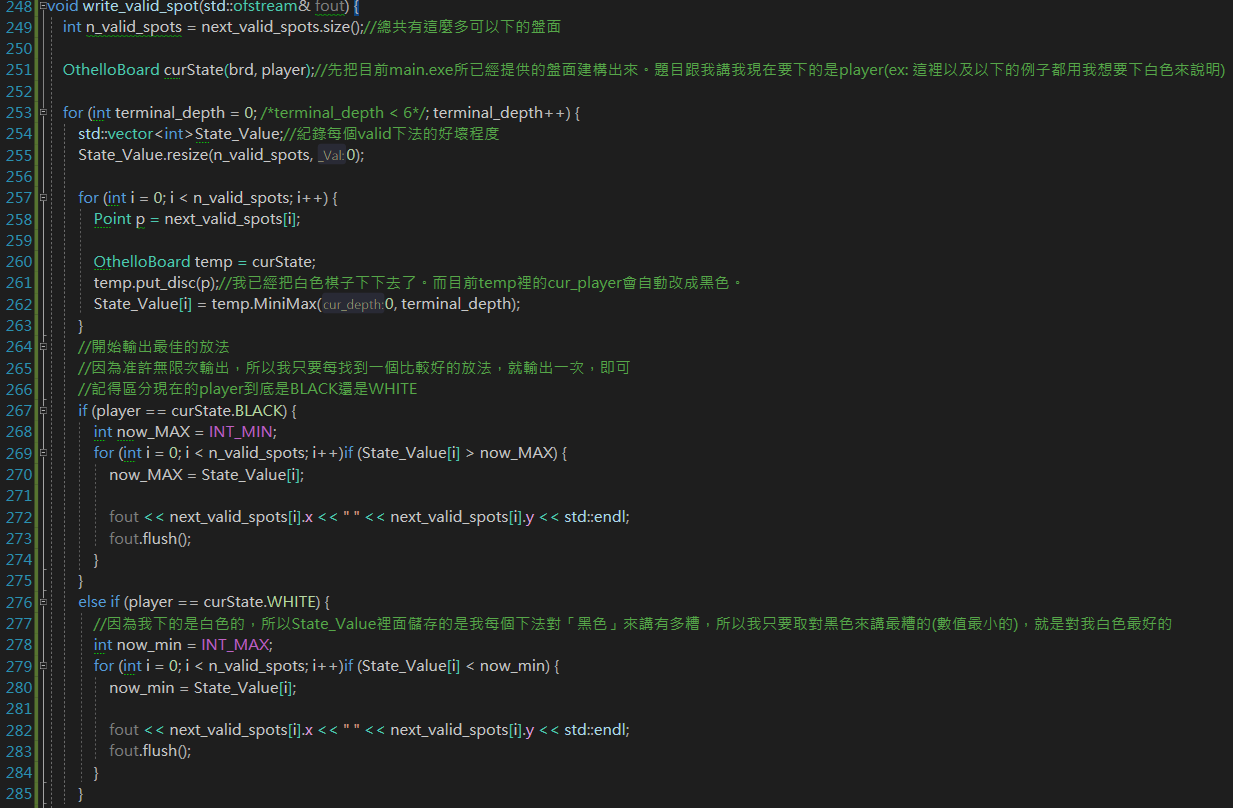
* Tree Search +2 point
  + 

int OthelloBoard::MiniMax(int cur\_depth, int terminal\_depth) {

//先計算目前的valid的格子有哪些

//不管是if(cur\_depth == terminal\_depth)還是後面兩個，都會用到

next\_valid\_spots = get\_valid\_spots();

if (cur\_depth == terminal\_depth) {

int score = 0;

if (cur\_player == BLACK) {

for (int i = 0; i < SIZE; i++)for (int j = 0; j < SIZE; j++) {

Point p(i, j);

if (p == Corners[0] || p == Corners[1] || p == Corners[2] || p == Corners[3]) {

if (board[p.x][p.y] == BLACK)score += Good\_Corner;

else if (board[p.x][p.y] == WHITE)score += Bad\_Corner;

}

else if (p.x == 0 || p.x == SIZE - 1) {

if (board[p.x][0] == BLACK && board[p.x][SIZE - 2] == BLACK)score += Four\_Sides;

else if (board[p.x][0] == WHITE && board[p.x][SIZE - 2] == WHITE)score += Bad\_Four\_Sides;

}

else if (p.y == 0 || p.y == SIZE - 1) {

if (board[0][p.y] == BLACK && board[SIZE - 2][p.y] == BLACK)score += Four\_Sides;

else if (board[0][p.y] == WHITE && board[SIZE - 2][p.y] == WHITE)score += Bad\_Four\_Sides;

}

if (disc\_count[WHITE] + disc\_count[BLACK] >= Latter\_Quarter\_Threshold) {

if (board[p.x][p.y] == BLACK)score += Ordinary\_Disc;

else if (board[p.x][p.y] == WHITE)score += Enemy\_Ordinary\_Disc;

}

}

}

else if (cur\_player == WHITE) {

for (int i = 0; i < SIZE; i++)for (int j = 0; j < SIZE; j++) {

Point p(i, j);

if (p == Corners[0] || p == Corners[1] || p == Corners[2] || p == Corners[3]) {

if (board[p.x][p.y] == WHITE)score += Good\_Corner;

else if (board[p.x][p.y] == BLACK)score += Bad\_Corner;

}

else if (p.x == 0 || p.x == SIZE - 1) {

if (board[p.x][0] == WHITE && board[p.x][SIZE - 2] == WHITE)score += Four\_Sides;

else if (board[p.x][0] == BLACK && board[p.x][SIZE - 2] == BLACK)score += Bad\_Four\_Sides;

}

else if (p.y == 0 || p.y == SIZE - 1) {

if (board[0][p.y] == WHITE && board[SIZE - 2][p.y] == WHITE)score += Four\_Sides;

else if (board[0][p.y] == BLACK && board[SIZE - 2][p.y] == BLACK)score += Bad\_Four\_Sides;

}

if (disc\_count[WHITE] + disc\_count[BLACK] >= Latter\_Quarter\_Threshold) {

if (board[p.x][p.y] == WHITE)score += Ordinary\_Disc;

else if (board[p.x][p.y] == BLACK)score += Enemy\_Ordinary\_Disc;

}

}

}

if (disc\_count[WHITE] + disc\_count[BLACK] < Latter\_Quarter\_Threshold) score += next\_valid\_spots.size()\*Mobility;

//注意return的時候，到底是誰在下棋?黑色?白色?

return (cur\_player == BLACK) ? score : -score;

}

if (this->cur\_player == BLACK/\*MAX\*/) {

//max of(a belongs to Action(cur state), MINIMAX(result of (cur state, a)))

//針對現有的盤面，去依序放上有所valid的格子，得到所有新的盤面。把所有新的盤面取MINIMAX，找最大的MINIMAX值回傳

int cur\_MAX = INT\_MIN;

for (int i = 0; i < next\_valid\_spots.size(); i++) {

OthelloBoard temp = \*this;

temp.put\_disc(next\_valid\_spots[i]);

int tmp = temp.MiniMax(cur\_depth + 1, terminal\_depth);

if (tmp > cur\_MAX)cur\_MAX = tmp;

}

return cur\_MAX;

}

else if (this->cur\_player == WHITE) {

//min of(a belongs to Action(cur state), MINIMAX(result of (cur state, a)))

int cur\_min = INT\_MAX;

for (int i = 0; i < next\_valid\_spots.size(); i++) {

OthelloBoard temp = \*this;

temp.put\_disc(next\_valid\_spots[i]);

int tmp = temp.MiniMax(cur\_depth + 1, terminal\_depth);

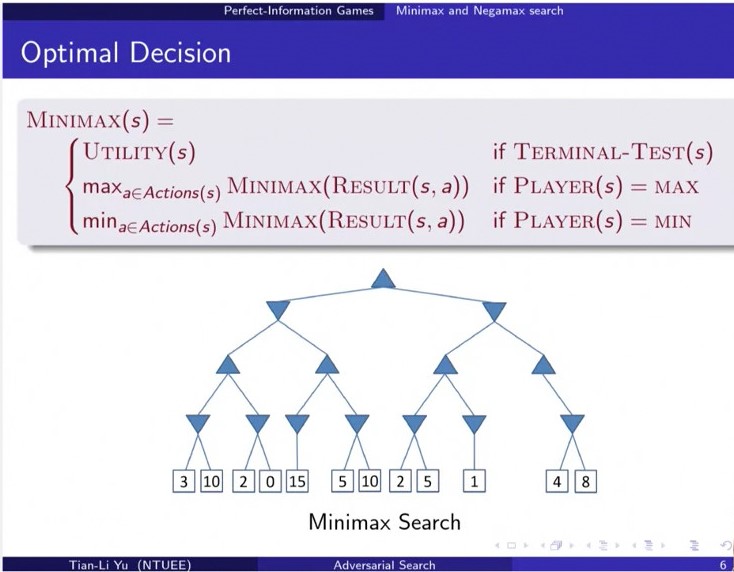
if (tmp < cur\_min)cur\_min = tmp;

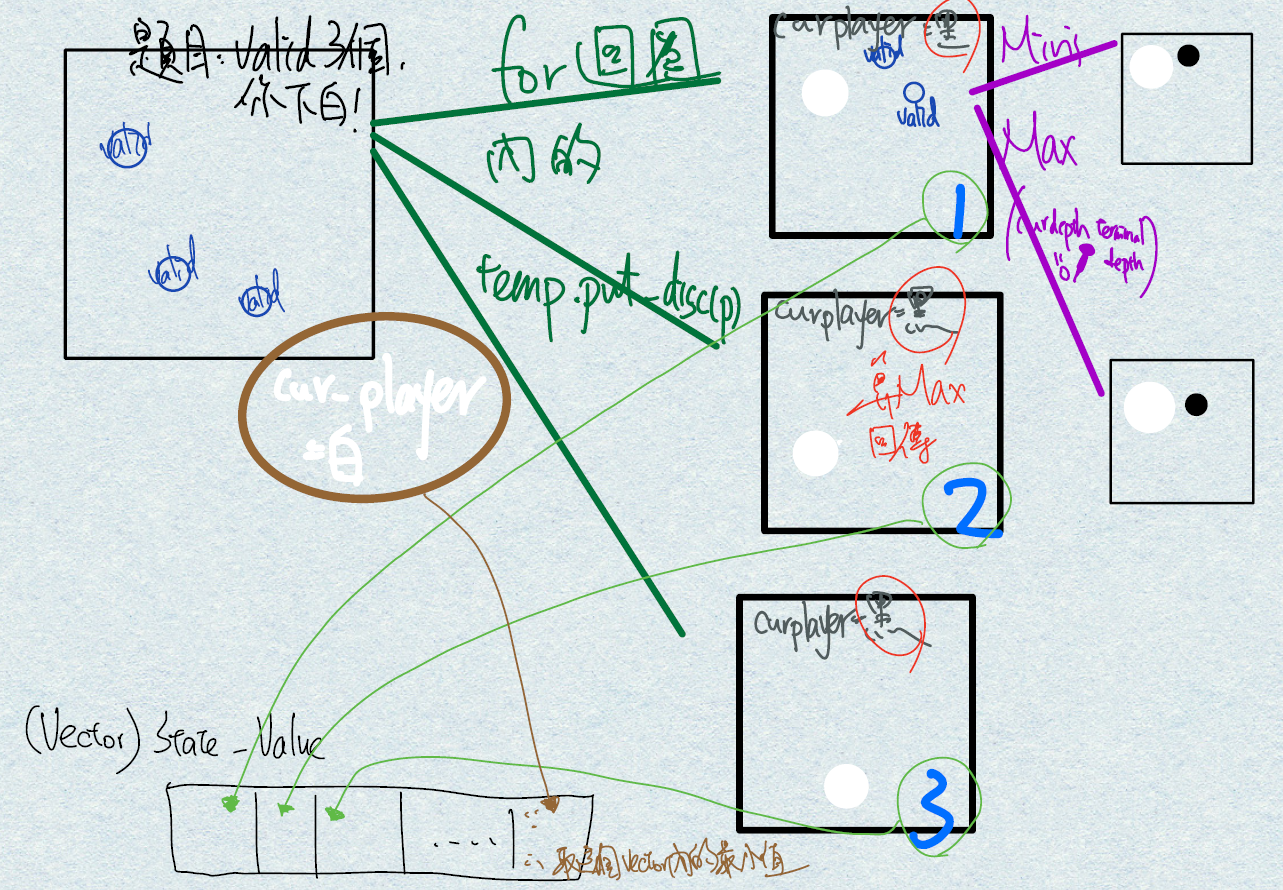
}

return cur\_min;

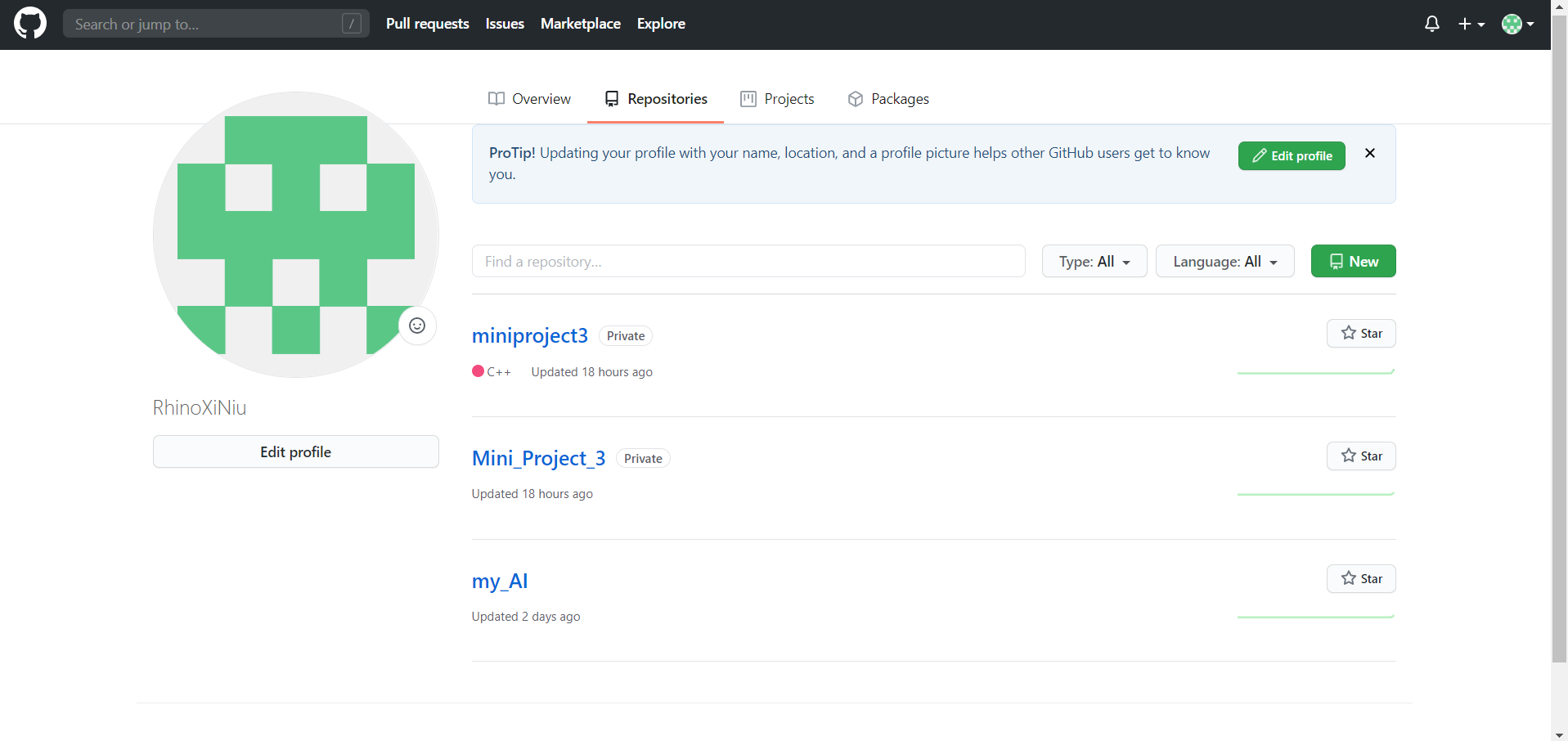
}

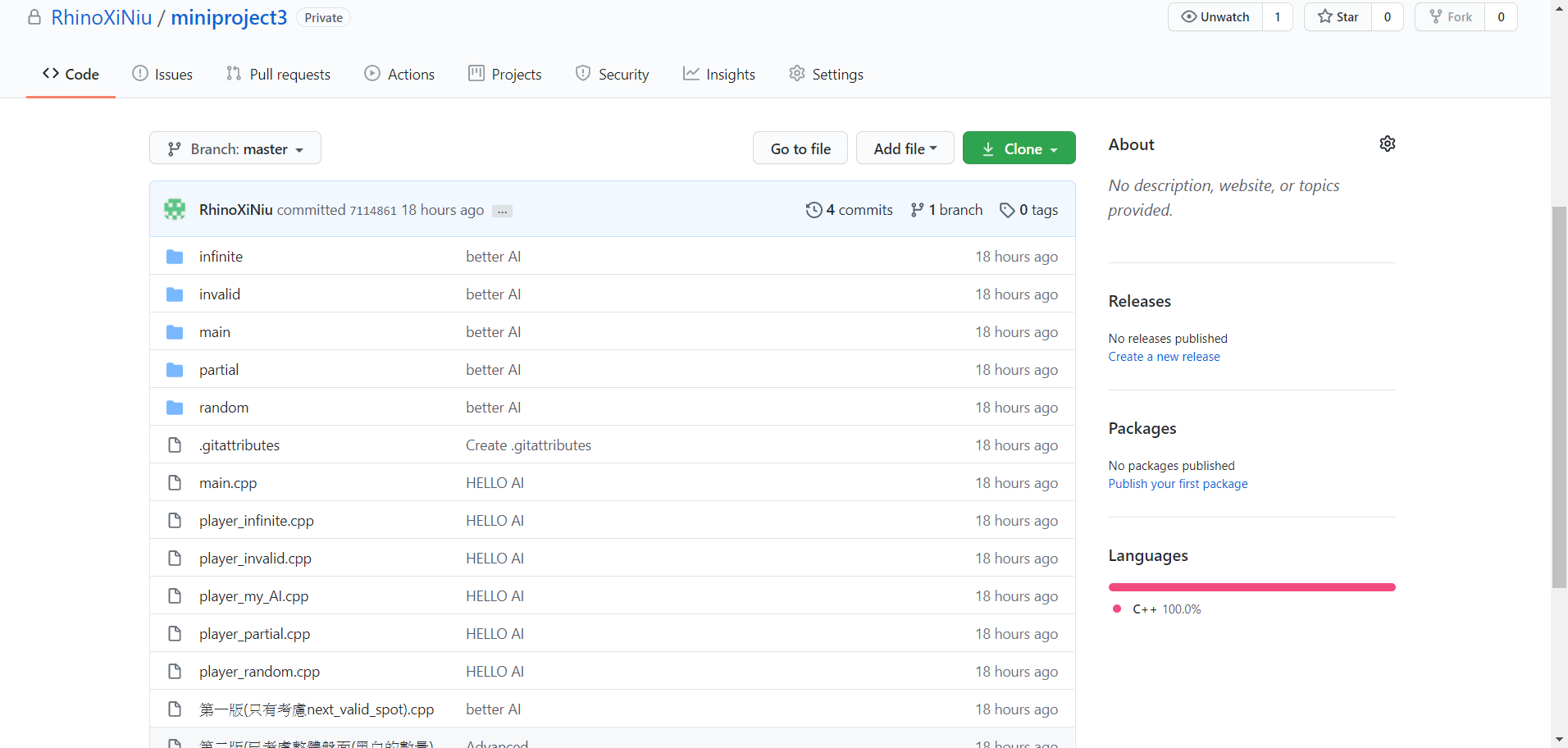
* }



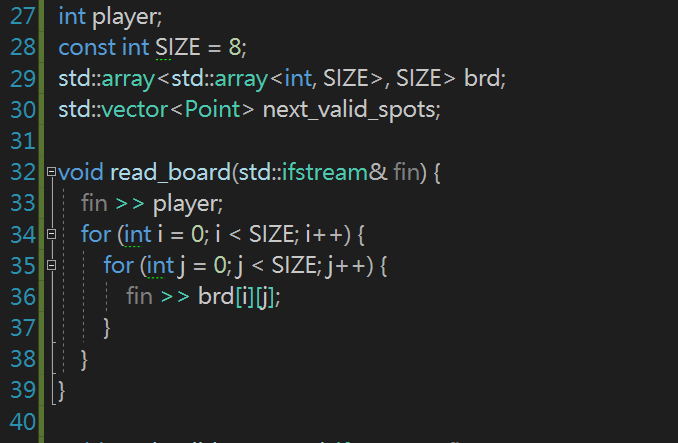
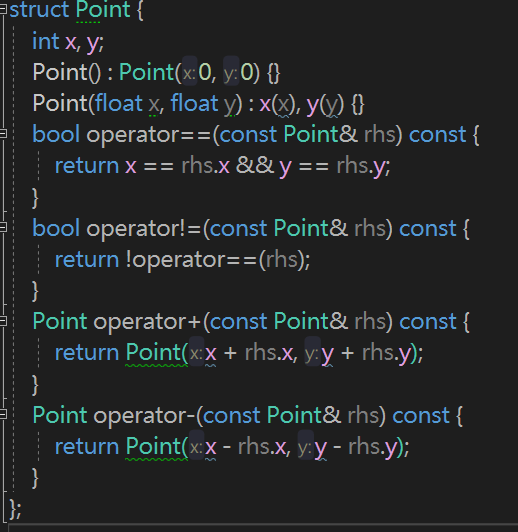


* State Value Function Design +1 point
  + 在上方文字函式的開頭處
* Alpha-beta Pruning +1 point
  + 沒有實作
* Version Control (Bonus) +1 point

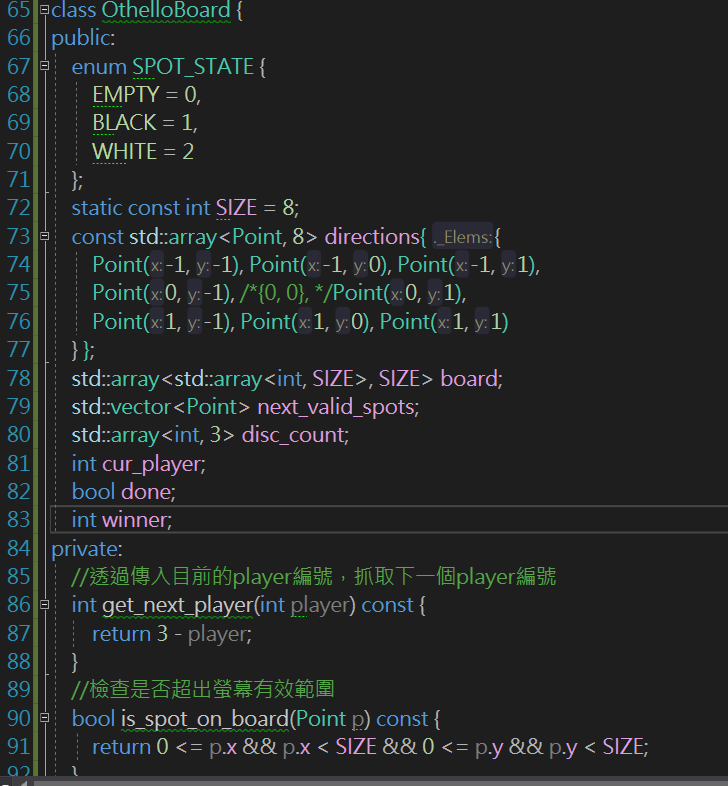
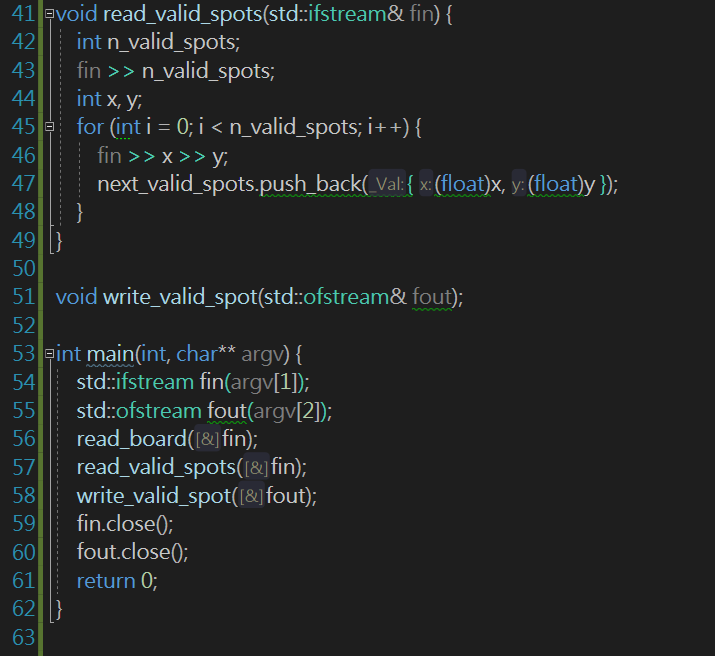




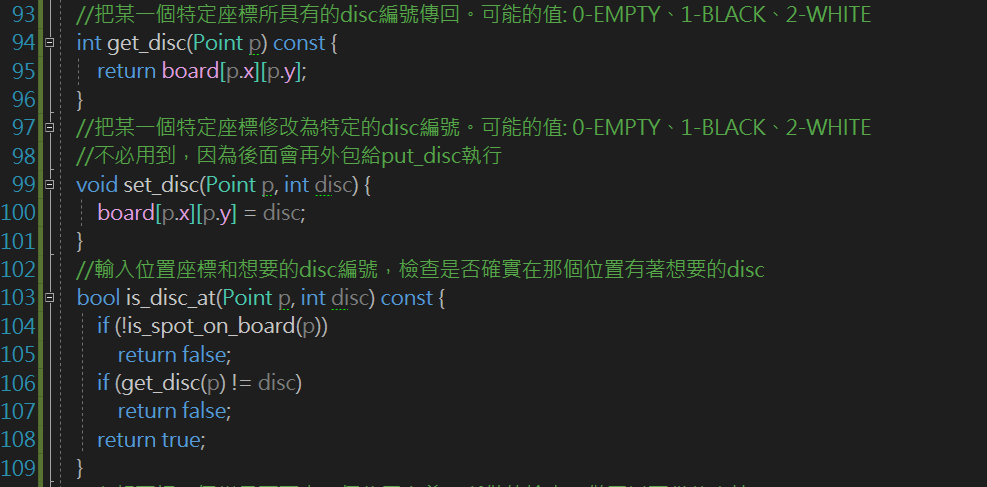
Struct Point、void read\_board、void read\_valid\_spots、int main。



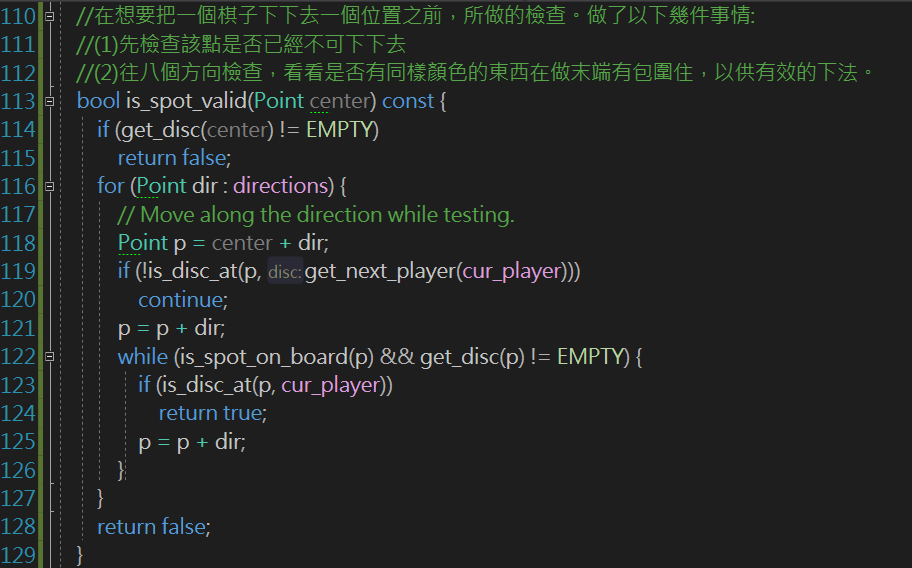
Class OthelloBoard: int get\_next\_player、bool is\_spot\_on\_board。



Int get\_disc、void set\_disc、bool is\_disc\_at。



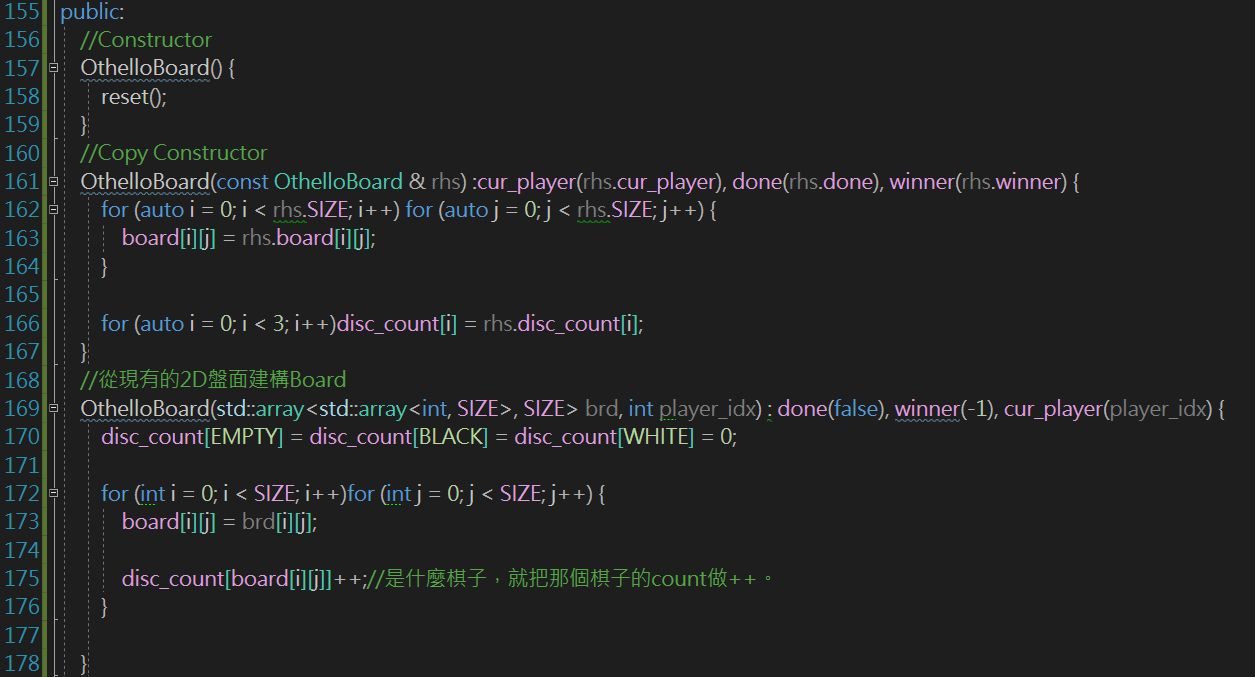
Bool is\_spot\_valid。



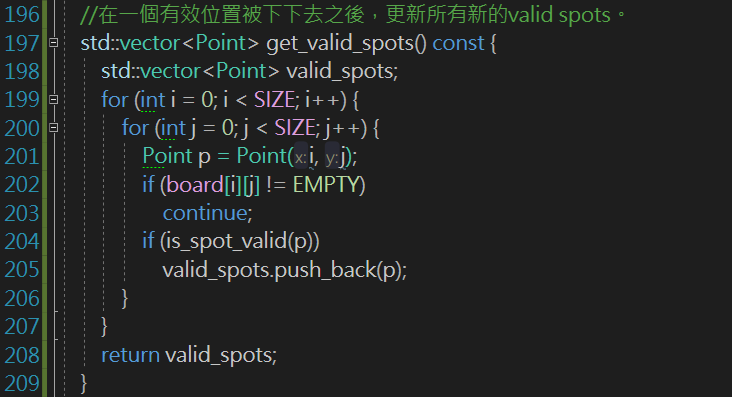
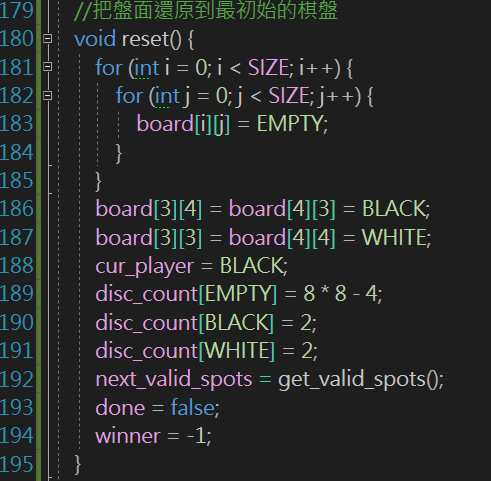
Void flip\_discs。



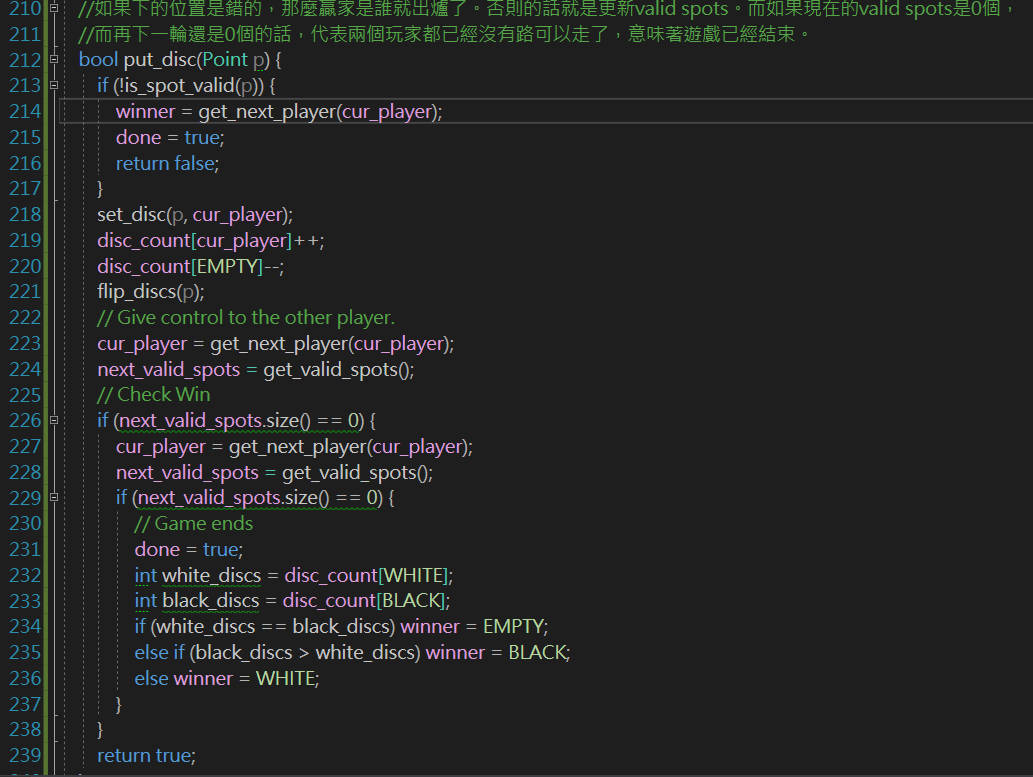
Constructor \* 3。



Void reset、std::vector<Point> get\_valid\_spots。



Bool put\_disc。



參數設定。

