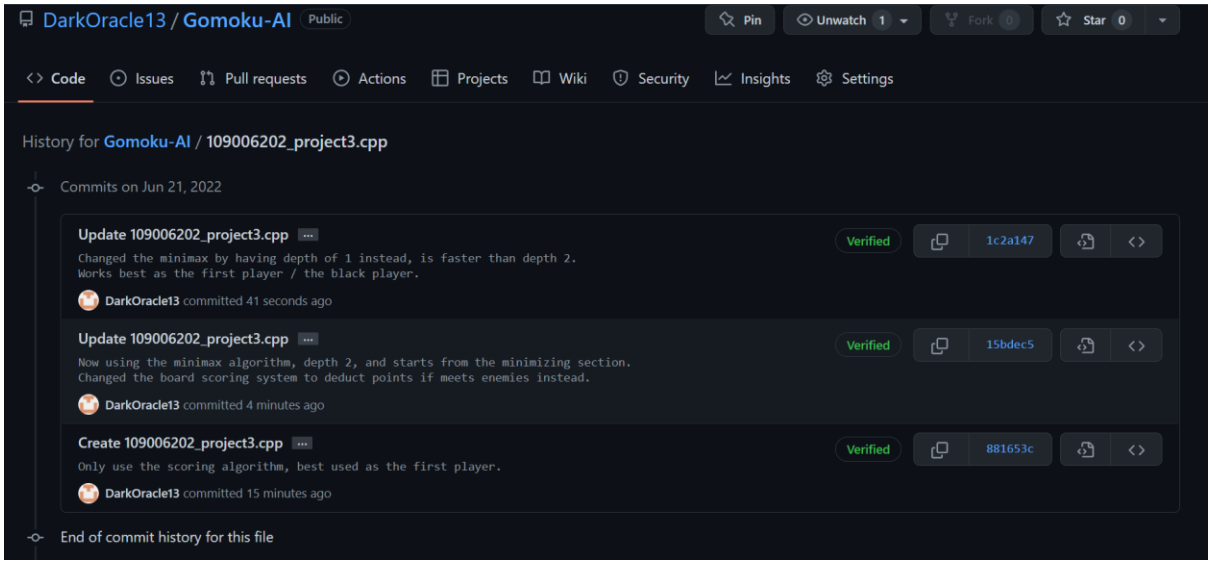


```cpp
void write_valid_spot(std::ofstream& fout) {

    int x;
    int y;

    int enemy;
    if(player == BLACK)
        enemy = WHITE;
    else
        enemy = BLACK;

    std::pair<int, std::pair<int,int>> best_move;
    // if(player == BLACK)
    //     best_move = setBoardScore(player, enemy, true);
    // else if(player == WHITE)
    //     best_move = setBoardScore(enemy, player, true);
    // x = best_move.second.first;
    // y = best_move.second.second;
    best_move = minimax(1, player, enemy, false);
    x = best_move.second.first;
    y = best_move.second.second;
    // int best_score = INT_MIN;
    // for(int i = 0; i < SIZE; i++){
    //     for(int j = 0; j < SIZE; j++){
    //         if(board[i][j] == EMPTY){
    //             board[i][j] = player;
    //             best_move = minimax(1, player, enemy, true);
    //             board[i][j] = EMPTY;
    //             if(best_move.first > best_score){
    //                 best_score = best_move.first;
    //                 x = i;
    //                 y = j;
    //             }
    //         }
```

Driver code to choose action, can choose if we just want to use the setBoardScore function, or we can use the minimax. If we want to use setBoardScore, then change all the value counter to add the total value, but if we want to use minimax, deduct the scores if meet enemy pieces.

```
33  std::pair<int,std::pair<int, int>> minimax(int depth, int player, int enemy, bool maximize){
34      if(depth == 0){
35          std::pair<int,std::pair<int, int>> score = setBoardScore(player, enemy, !maximize);
36          return score;
37      }
38      if(maximize == true){
39          int maxval = INT_MIN;
40          std::pair<int, std::pair<int,int>> maxpos;
41          for(int i=0; i<SIZE; i++){
42              for(int j=0; j<SIZE; j++){
43                  if(board[i][j] == EMPTY){
44                      board[i][j] = player;
45                      std::pair<int, std::pair<int,int>> temp;
46                      temp = minimax(depth-1, player, enemy, false);
47                      if(temp.first > maxval){
48                          maxval = temp.first;
49                          maxpos = std::make_pair(maxval, std::make_pair(i,j));
50                      }
51                      board[i][j] = EMPTY;
52                  }
53              }
54          }
55          return maxpos;
56      }else{
57          int minval = INT_MAX;
58          std::pair<int, std::pair<int,int>> minpos;
59          //std::cout <<"Start minimizing\n";
60          for(int i=0; i<SIZE; i++){
61              for(int j=0; j<SIZE; j++){
62                  if(board[i][j] == EMPTY){
63                      board[i][j] = enemy;
64                      std::pair<int, std::pair<int,int>> temp;
65                      temp = minimax(depth-1, player, enemy, true);
66                      if(temp.first < minval){
67                          minval = temp.first;
68                          minpos = std::make_pair(minval, std::make_pair(i,j));
69                      }
```

Minimax algorithm.

During maximization, find the minimum of the enemy.

```
std::pair<int, std::pair<int,int>> setBoardScore(int player, int enemy, bool maximize){
    // we want to check the score for every spot on the board
    // for each position, check for pieces which are connected in a row,
    // with the directions of going up, down, left, right, and diagonally
    // std::ios_base::sync_with_stdio(false);
    // std::cin.tie(NULL);
    int boardscore[SIZE][SIZE];
    for(int i=0; i<SIZE; i++){
        for(int j=0; j<SIZE; j++){
            boardscore[i][j] = 0;
        }
    }

    for(int i = 0; i < SIZE; i++){
        for(int j = 0; j < SIZE; j++){
            if(board[i][j] == EMPTY){
                std::vector<int> player_data;
                std::vector<int> enemy_data;
                int up = 0;
                for(int k = i-1; k >= 0; k--){
                    if(board[k][j] == player){
                        up++;
                    }
                    else{
                        break;
                    }
                }
                player_data.push_back(up);
                int enemyup = 0;
                for(int k = i-1; k >= 0; k--){
                    if(board[k][j] == enemy){
                        enemyup++;
                    }
                    else{
                        break;
                    }
```

Find all positions which are empty, then count the consecutive pieces that are in adjacent in that spot. Count both enemies and friendlies alike.

```
        }
        enemy_data.push_back(enemydown_right);
        //std::cout <<"Player data: ";
        int size = player_data.size();
        for(int k = 0; k < size; k++){
            //std::cout << player_data[k] << " ";
            if(player_data[k] == 5){
                boardscore[i][j] += 3000;//win
            }
            else if(player_data[k] == 4){
                boardscore[i][j] += 120;
            }
            else if(player_data[k] == 3){
                boardscore[i][j] += 80;
            }
            else if(player_data[k] == 2){
                boardscore[i][j] += 5;
            }
            // else if(player_data[k] == 1){
            //     boardscore[i][j] += 2;
            // }
        }
        //std::cout <<"\nEnemy data: ";
        size = enemy_data.size();
        for(int k = 0; k < size; k++){
            //std::cout << enemy_data[k] << " ";
            if(enemy_data[k] == 5){
                boardscore[i][j] -= 3000; //lose
            }
            else if(enemy_data[k] == 4){
                boardscore[i][j] -= 120;
            }
            else if(enemy_data[k] == 3){
                boardscore[i][j] -= 80;
            }
            else if(enemy_data[k] == 2){
                boardscore[i][j] -= 5;
```

Assign score if there exist some pieces. For consecutive twos, give 5, 3s give 80, 4s give 120, 5s give 3000.

If we are only using the setBoardScore algo, and no minimax, then change the boardscore value to increase when meeting enemy pieces, but if we are using minimax, then let it be.

```
339     if(maximize){
340         int maxval = INT_MIN;
341         std::pair<int, std::pair<int,int>> maxpos;
342         for(int i=0; i<SIZE; i++){
343             for(int j=0; j<SIZE; j++){
344                 if(boardscore[i][j] > maxval && board[i][j] == EMPTY){
345                     maxval = boardscore[i][j];
346                     maxpos = std::make_pair(maxval, std::make_pair(i,j));
347                 }
348             }
349         }
350         return maxpos;
351     }else{
352         int minval = INT_MAX;
353         std::pair<int, std::pair<int,int>> minpos;
354         for(int i=0; i<SIZE; i++){
355             for(int j=0; j<SIZE; j++){
356                 if(boardscore[i][j] < minval && board[i][j] == EMPTY){
357                     minval = boardscore[i][j];
358                     minpos = std::make_pair(minval, std::make_pair(i,j));
359                 }
360             }
361         }
362         return minpos;
363     }
364 }
```

Find the maximum or minimum value of the board, according to the mode that we previously selected. The node data is (value , coordinate i, coordinate j).