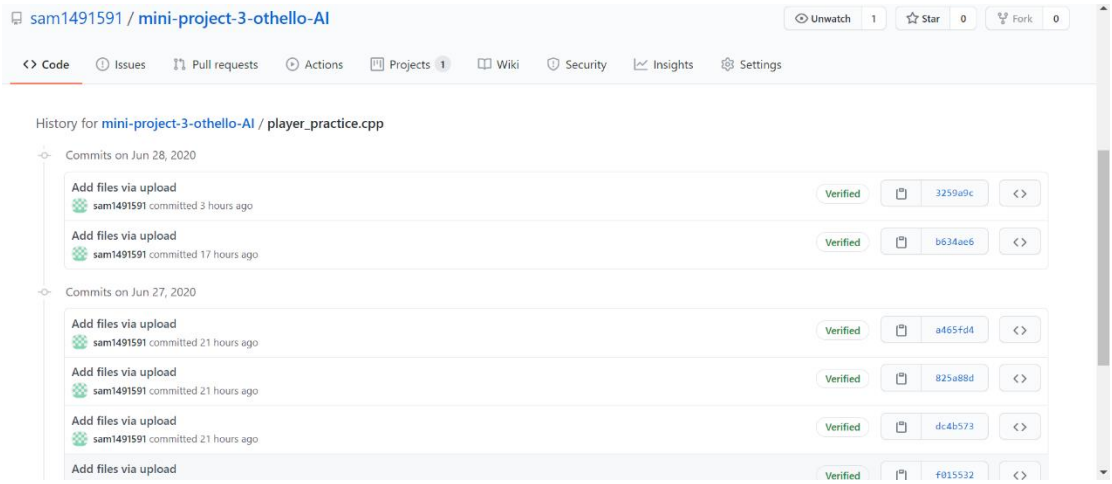
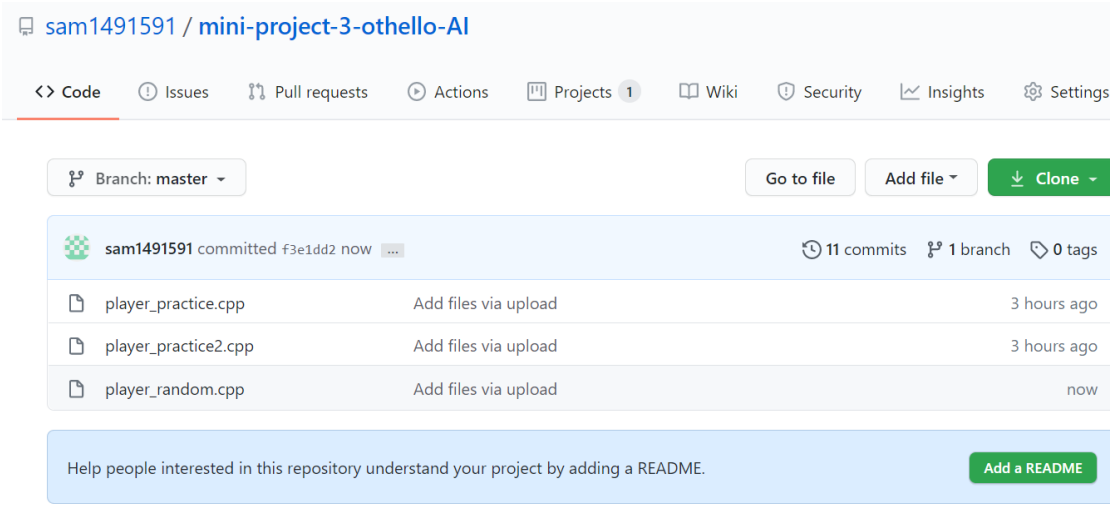

sam1491591 / mini-project-3-othello-AI
Unwatch 1
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects 1
Wiki
Security
Insights
Settings
History for mini-project-3-othello-AI / player_practice.cpp
Commits on Jun 28, 2020
Add files via upload
sam1491591 committed 3 hours ago
Verified
3259a9c
Add files via upload
sam1491591 committed 17 hours ago
Verified
b634ac6
Commits on Jun 27, 2020
Add files via upload
sam1491591 committed 21 hours ago
Verified
a465fd4
Add files via upload
sam1491591 committed 21 hours ago
Verified
825a88d
Add files via upload
sam1491591 committed 21 hours ago
Verified
dc4b573
Add files via upload
Verified
f015532


sam1491591 / mini-project-3-othello-AI
Code
Issues
Pull requests
Actions
Projects 1
Wiki
Security
Insights
Settings
Branch: master
Go to file
Add file
Clone
sam1491591 committed f3e1dd2 now ...
11 commits
1 branch
0 tags
player_practice.cpp
Add files via upload
3 hours ago
player_practice2.cpp
Add files via upload
3 hours ago
player_random.cpp
Add files via upload
now
Help people interested in this repository understand your project by adding a README.
Add a README

```cpp
int estimate[8][8]=
{
    90,-60,10,10,10,10,-60,90,
    -60,-80,5,5,5,5,-80,-60,
    10,5,1,1,1,1,5,10,
    10,5,1,1,1,1,5,10,
    10,5,1,1,1,1,5,10,
    10,5,1,1,1,1,5,10,
    -60,-80,5,5,5,5,-80,-60,
    90,-60,10,10,10,10,-60,90
};

int player;
Point ans;
const int SIZE = 8;
std::vector<Point> next_valid_spots;
std::vector<Point> second_valid_spots;
std::array<int, 3> disc_count;
std::array<std::array<int, SIZE>, SIZE> board;

const std::array<Point, 8> directions{{
        Point(-1, -1), Point(-1, 0), Point(-1, 1),
        Point(0, -1), /*{0, 0}, */Point(0, 1),
        Point(1, -1), Point(1, 0), Point(1, 1)
    }};

struct state
{
        int player;
        int value;
        std::array<std::array<int, SIZE>, SIZE> first_board;
        std::array<std::array<int, SIZE>, SIZE> second_board;
};
```

```cpp
bool is_spot_valid(state next,Point center)
{
    if (next.first_board[center.x][center.y] != 0)
        return false;
    for (Point dir: directions)
    {
        // Move along the direction while testing.
        Point p = center + dir;
        if (next.first_board[p.x][p.y]!=3-next.player)
            continue;
        p = p + dir;
        while ((0 <= p.x && p.x < SIZE && 0 <= p.y && p.y < SIZE) && next.first_board[p.x][p.y]!=0)
        {
            if (next.first_board[p.x][p.y]==next.player)
                return true;
            p = p + dir;
        }
    }
    return false;
}

std::vector<Point> get_valid_spots(state next)
{
    std::vector<Point> valid_spots;
    for (int i = 0; i < SIZE; i++)
    {
        for (int j = 0; j < SIZE; j++)
        {
            Point p = Point(i, j);
            if (next.first_board[i][j] != 0)
                continue;
            if (is_spot_valid(next,p))
                valid_spots.push_back(p);
        }
    }
```

```
int calculate(state cur,int depth,int player1,int player2,std::vector<Point> next_valid_spots,int alpha,int beta)
{
    int p1=player1,p2=player2;
    int max=-INT_MAX;
    if(depth==0)
        return 0;
    for (Point t : next_valid_spots)
    {
        p1=player1,p2=player2;
        for(int i=0;i<SIZE;i++)
            for(int j=0;j<SIZE;j++)
            {
                cur.second_board[i][j]=cur.first_board[i][j];
            }
        for (Point dir: directions)
        {
            // Move along the direction while testing.
            Point p = t + dir;
            if (cur.first_board[p.x][p.y]!=3-cur.player)
                continue;
            std::vector<Point> discs({p});
            p = p + dir;
            while ((0 <= p.x && p.x < SIZE && 0 <= p.y && p.y < SIZE) && cur.first_board[p.x][p.y]!=0)
            {
                if (cur.first_board[p.x][p.y]==cur.player)
                {
                    for (Point s: discs) {
                        cur.second_board[s.x][s.y]=cur.player;
                    }
                    //p1 += discs.size();
                    //p2 -= discs.size();
                    break;
                }
                discs.push_back(p);
                p = p + dir;
```

```
                discs.push_back(p);
                p = p + dir;
            }
        }
        cur.second_board[t.x][t.y]=cur.player;
        state next;
        for(int i=0;i<SIZE;i++)
            for(int j=0;j<SIZE;j++)
                next.first_board[i][j]=cur.second_board[i][j];
        next.player=3-cur.player;
        second_valid_spots=get_valid_spots(next);
        /*
        if(second_valid_spots.size()>10)
            cur.value=1000;
        else if(second_valid_spots.size()==0)
            cur.value=-1000;
        else
        */
        /*
        if(second_valid_spots.size()==0)
        {
            if(depth==4);
            else if(depth%2==0)
                return 1000;
            else
                return -1000;
        }
        */

        int tmp;
        if(depth%2==0)
        {
            tmp=calculate(next,depth-1,p2,p1,second_valid_spots,alpha,beta);
            if(tmp>alpha)
                alpha=tmp;
```

```
        int tmp;
        if(depth%2==0)
        {
            tmp=calculate(next,depth-1,p2,p1,second_valid_spots,alpha,beta);
            if(tmp>alpha)
                alpha=tmp;
            cur.value=p1-p2-tmp;
        }
        else
        {
            tmp=-calculate(next,depth-1,p2,p1,second_valid_spots,alpha,beta);
            if(tmp<beta)
                beta=tmp;
            cur.value=p1-p2+tmp;
        }

        //cur.value=p1-p2-calculate(next,depth-1,p2,p1,second_valid_spots);
        cur.value+=estimate[t.x][t.y];
        cur.value-=2*second_valid_spots.size();
        //cur.value+=estimate[t.x][t.y];
        if(cur.value>max)
        {
            if(depth==4)
                ans=t;
            max=cur.value;
        }
        /*
        if(alpha<beta)
            return max;
        */
    }
    return max;
}
```

```
void write_valid_spot(std::ofstream& fout)
{
    //int n_valid_spots = next_valid_spots.size();
    //srand(time(NULL));

    // Choose random spot. (Not random uniform here)
    //int index = (rand() % n_valid_spots);
    //Point p = next_valid_spots.front();
    state current;
    for(int i=0;i<SIZE;i++)
        for(int j=0;j<SIZE;j++)
            current.first_board[i][j]=board[i][j];
    current.player=player;
    current.value=0;
    int statevalue=calculate(current,4,disc_count[player]+1,disc_count[3-player],next_valid_spots);
    //std::cout<<ans.x<<" "<<ans.y<<std::endl;
    // Remember to flush the output to ensure the last action is written to file.
    fout << ans.x << " " << ans.y << std::endl;
    fout.flush();
}
```