# **亚马逊期末大作业报告**

1800094810 枚辉煌

1. 代码结构

代码由三个源文件组成：main.cpp、game.cpp、board.cpp，其中相应的头文件是game.h和board.h，定义Game和Board类型。Main只有开始游戏的功能而已。

// main.cpp

#include "pch.h"

#include "game.h"

#include <iostream>

int main()

{

    Game game; // create game

    while (game.isRunning)

    {

        game.ShowMenu();

    }

}

Class Game 是游戏的框架，包括显示菜单、处理玩家的输入、计算对应的棋步。

class Game {

public:

    bool isRunning;

    Game();

    void ShowMenu();

private:

    Board board;

    int startX, startY, resultX, resultY, obstacleX, obstacleY;

    map<pair<grid\_t, int>, bool> data; // save checked state in DeepSearch

    map<pair<grid\_t, int>, vector<cell>> step; // save move step for each state

    queue<pair<int, int>> q; // queue for bfs

    int dis[GRID\_SIZE][GRID\_SIZE];

    void HumanMove(); // handle human move

    void BotMove(); // calculate computer move

    int CalAreaDiff(int color); // calculate area difference

    int MoveArea(int color, int depth); // calculate mobility

    int GreedySearch(int color);

bool DeepSearch(int color);

}

Class Board 记录棋盘的状态，画棋盘，检测并且执行棋步。

class Board

{

private:

    grid\_t grid; // use 2D int array to save grid information

public:

    int turnID;

    Board();

    Board(bool isNewGame); // Board's constructor, isNewGame == false to load game from file

    int GetCell(int x, int y); // return cell information

    grid\_t GetGrid(); // return all grid

    vector<cell> pos[2]; // pos[0/1] is white/black chess location

    bool InGrid(int x, int y);

    bool OnStraghtLine(int x1, int y1, int x2, int y2);

    bool HaveObstacleBetween(int x1, int y1, int x2, int y2);

    void DrawLine(int length);

    void DrawBoard();

    void SaveBoard(); // save board information to SAVE\_FILE

    bool ValidMove(int x1, int y1, int x2, int y2, int x3, int y3, int color); // check valid move

    void ApplyMove(int x1, int y1, int x2, int y2, int x3, int y3, bool revert); // apply move to board, can revert move

    bool LoseState(int color); // check if color in lose state

};

1. 代码实验思路

首先，我采用Minimax算法，极大化自己最好的棋步并且极小化对方最好的棋步。在亚马逊棋，评估棋步的方法根据两个因素：领土和灵动性。具体在探索的过程，我试试所有可能的棋步，然后试试对方所有可能的棋步，最后计算自己的分数和对方的分数，选择两者差距最大的棋步。在游戏前15轮，我优先灵动性，用四个棋子通过一步能达到格子的个数来算分。在16轮到22轮，不限制棋步的数量，用领土来算分。具体如下:

// Calculate mobility after (depth) moves

int Game::MoveArea(int color, int depth)

{

    int area = 0;

    memset(dis, 0, sizeof dis);

    int t = color; if (t == -1) t = 0;

    for (int i = 0; i < board.pos[t].size(); i++)

    {

        q.push(board.pos[t][i]);

    }

    // Use BFS to calculate the area

    while (!q.empty())

    {

        int i, j;

        tie(i, j) = q.front(); q.pop();

        for (int k = 0; k < 8; k++)

            for (int d = 1; d < 8; d++)

            {

                int u = i + dx[k] \* d, v = j + dy[k] \* d;

                if (!board.InGrid(u, v)) break;

                if (board.GetCell(u, v) != 0) break;

                if (dis[u][v] == 0)

                {

                    dis[u][v] = dis[i][j] + 1;

                    area++;

                    if (dis[u][v] < depth)

                        q.push(cell(u, v));

                }

            }

    }

    return area;

}

关于领土和灵动性的差距计算：

int Game::CalAreaDiff(int color)

{

// in the first 15 turns, mobility is calculated by Queen's move

    if (board.turnID <= 15)

    {

        return MoveArea(color, 1) - MoveArea(-color, 1);

    }

    else // turn 16-22, calculate the number of cell each color can reach

    {

        return MoveArea(color, INF) - MoveArea(-color, INF);

    }

}

用Minimax算法来探索棋步：

// Use Minimax Algorithm, maximize self's best move and minimize opponent's best move

// Only calculate depth = 1

int Game::GreedySearch(int color)

{

    int rx1, ry1, rx2, ry2, rx3, ry3, best = -INF;

    int t = color; if (t == -1) t = 0;

    for (int i = 0; i < board.pos[t].size(); i++)

    {

        // iterate all chess's position x1 y1

        int x1 = board.pos[t][i].first, y1 = board.pos[t][i].second;

        for (int k1 = 0; k1 < 8; k1++)

        for (int d1 = 1; d1 < 8; d1++)

        {

            // move to cell x2 y2

            int x2 = x1 + dx[k1] \* d1, y2 = y1 + dy[k1] \* d1;

            if (!board.InGrid(x2, y2)) break;

            if (board.GetCell(x2, y2) != 0) break;

            for (int k2 = 0; k2 < 8; k2++)

            for (int d2 = 1; d2 < 8; d2++)

            {

                // shoot arrow to x3 y3

                int x3 = x2 + dx[k2] \* d2, y3 = y2 + dy[k2] \* d2;

                if (!board.InGrid(x3, y3)) break;

                if (board.GetCell(x3, y3) != 0 && !(x3 == x1 && y3 == y1)) break;

从23轮起，根据Game theory暴力探索所有可能棋盘状态。我用vector<vector<int>>类型记录棋盘状态以便用map标记探索过的状态。具体如下：

// apply move

                board.ApplyMove(x1, y1, x2, y2, x3, y3, false);

                int area;

// Bot's turn, calculate Player's best move

                if (BotColor == color)

                    area = -GreedySearch(-color);

                else // Player's turn, maximize area difference

                    area = CalAreaDiff(color);

                if (area > best)

                {

                    best = area;

                    if (color == BotColor) tie(rx1, ry1, rx2, ry2, rx3, ry3) = tie(x1, y1, x2, y2, x3, y3);

                }

                // revert move

                board.ApplyMove(x1, y1, x2, y2, x3, y3, true);

            }

        }

    }

    if (best == -INF)

    {

        if (color == BotColor) startX = startY = resultX = resultY = obstacleX = obstacleY = -1;

        return 0;

    }

    else

    {

        if (color == BotColor)

        {

            tie(startX, startY, resultX, resultY, obstacleX, obstacleY) = tie(rx1, ry1, rx2, ry2, rx3, ry3);

        }

        return best;

    }

}

// Use game theory to calculate all possible moves

bool Game::DeepSearch(int color)

{

    auto it = data.find(make\_pair(board.GetGrid(), color));

    // if this state's already calculated, return the value

    if (it != data.end()) return it->second;

    bool flag = false;

    int t = color; if (t == -1) t = 0;

    for (int i = 0; i < board.pos[t].size(); i++)

    {

        // iterate all chess's position x1 y1

        int x1 = board.pos[t][i].first, y1 = board.pos[t][i].second;

        for (int k1 = 0; k1 < 8; k1++)

        for (int d1 = 1; d1 < 8; d1++)

        {

            // move to cell x2 y2

            int x2 = x1 + dx[k1] \* d1, y2 = y1 + dy[k1] \* d1;

            if (!board.InGrid(x2, y2)) break;

            if (board.GetCell(x2, y2) != 0) break;

            for (int k2 = 0; k2 < 8; k2++)

            for (int d2 = 1; d2 < 8; d2++)

            {

                // put obstacle at x3 y3

                int x3 = x2 + dx[k2] \* d2, y3 = y2 + dy[k2] \* d2;

                if (!board.InGrid(x3, y3)) break;

                if (board.GetCell(x3, y3) != 0 && !(x3 == x1 && y3 == y1)) break;

                // apply move

                board.ApplyMove(x1, y1, x2, y2, x3, y3, false);

                if (!DeepSearch(-color)) flag = true;

                // revert move

                board.ApplyMove(x1, y1, x2, y2, x3, y3, true);

                // if this move can let (color) player win

                if (flag)

                {

                    auto state = make\_pair(board.GetGrid(), color);

                    step[state].emplace\_back(x1, y1);

                    step[state].emplace\_back(x2, y2);

                    step[state].emplace\_back(x3, y3);

                    break;

                }

            }

            if (flag) break;

        }

        if (flag) break;

    }

这些是AI主要算法的思路。

    auto state = make\_pair(board.GetGrid(), color);

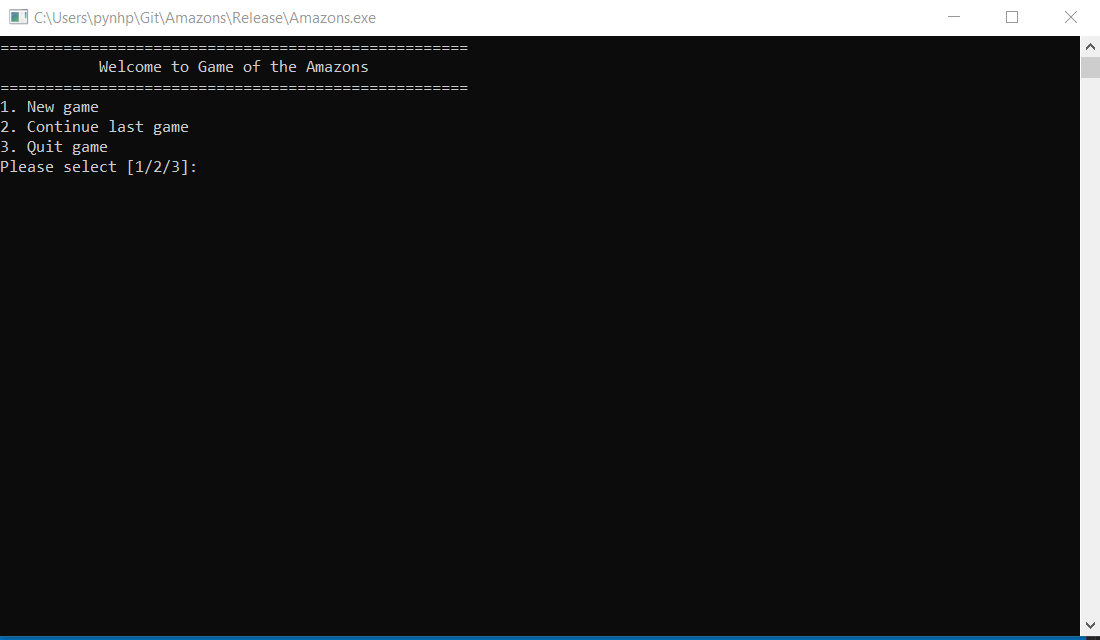
    data[state] = flag;

    return flag;

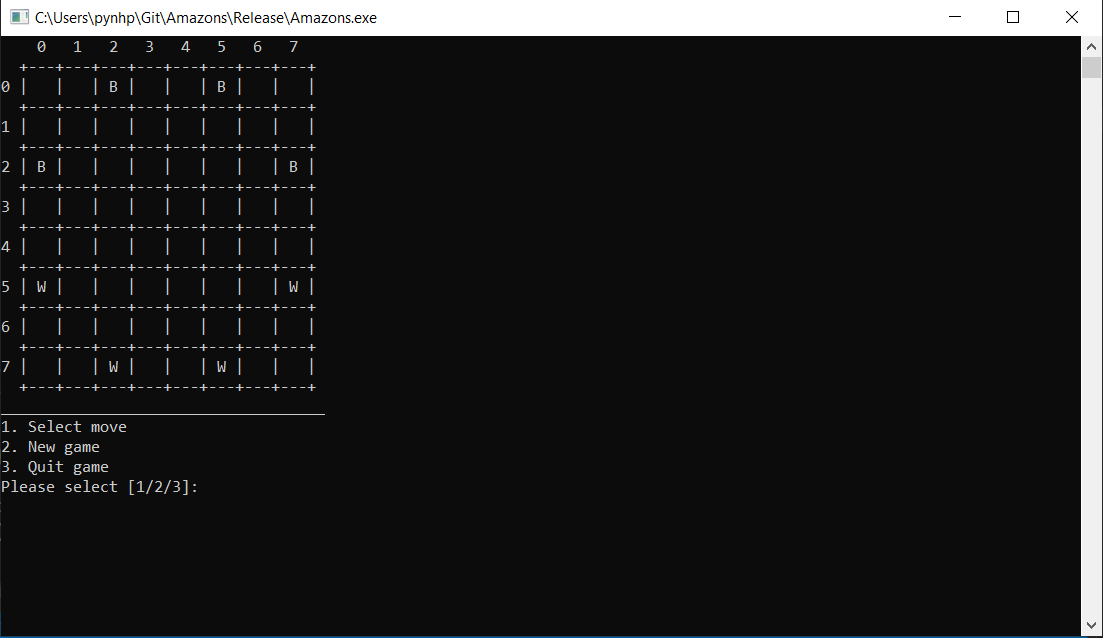
}

1. 代码运行过程

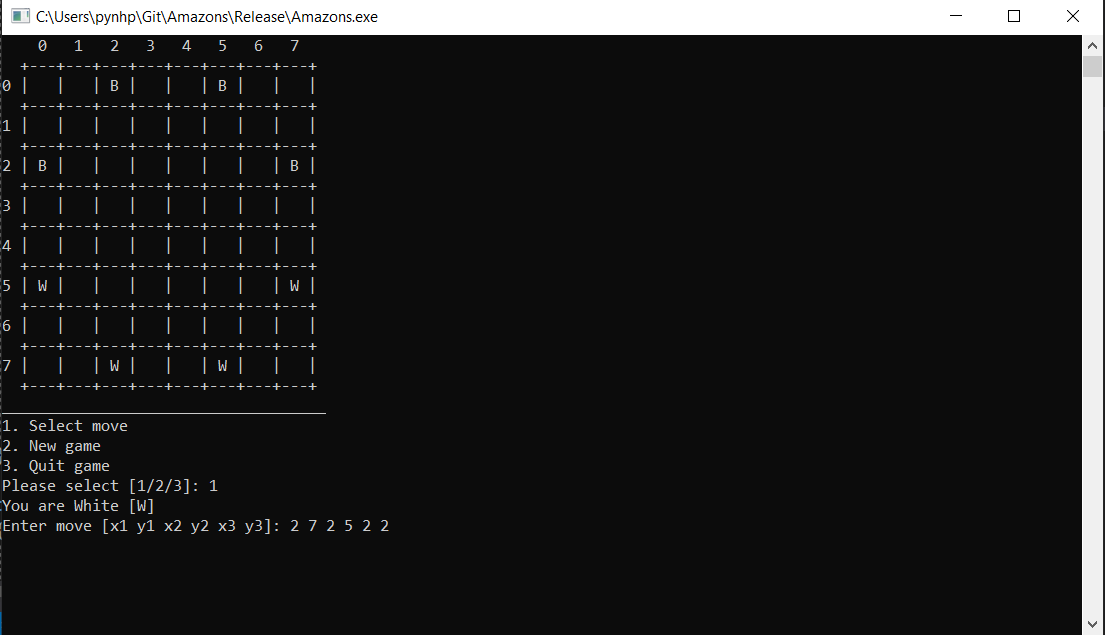
开始



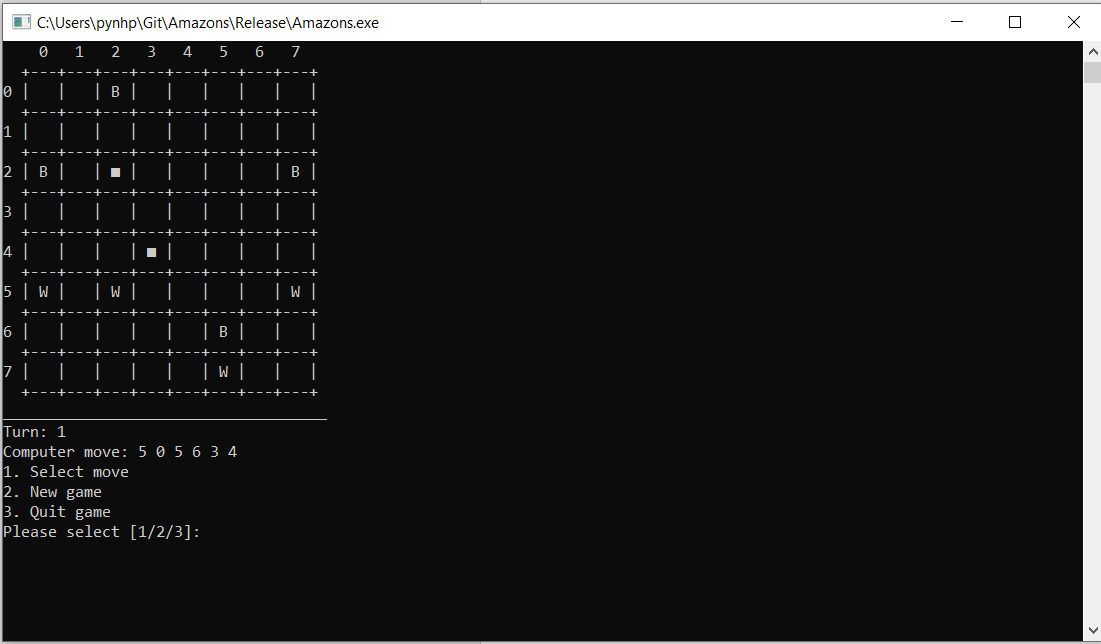
点击New game



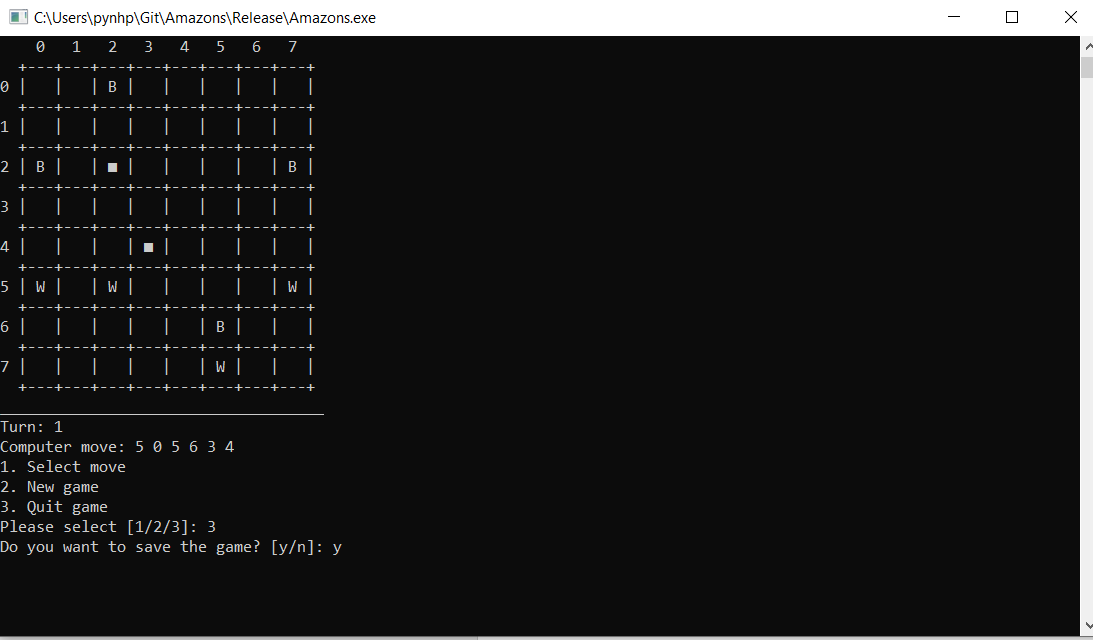
点击Select move



处理玩家输入并且计算棋步



存库棋盘



显示开始菜单以后，程序一直循环ShowMenu()，在ShowMenu() 里继续处理玩家的棋步和计算AI的棋步。

    switch (ops)

    {

    case 1: // Select move

        HumanMove();

        BotMove();

        break;

    case 2: // New game

        board = new Board(true);

        board.DrawBoard();

        break;

    case 3: // Quit game

        cout << "Do you want to save the game? [y/n]: ";

        char ans;

        cin >> ans;

        if (ans == 'y') board.SaveBoard();

        isRunning = false;

        break;

    default:

        break;

    }

其中HumanMove()不多说，重要的是BotMove()如下：

void Game::BotMove()

{

    if (board.LoseState(BotColor)) // Bot in lose state

    {

        cout << "You win!\n";

        isRunning = false;

        return;

    }

    if (board.turnID <= 23) // Use Minimax algorithm in the first 22 moves

    {

        GreedySearch(BotColor);

    }

    else // Search all possible moves

    {

        data.clear();

        step.clear();

        bool canWin = DeepSearch(BotColor);

        if (canWin)

        {

            auto state = make\_pair(board.GetGrid(), BotColor);

            tie(startX, startY) = step[state][0];

            tie(resultX, resultY) = step[state][1];

            tie(obstacleX, obstacleY) = step[state][2];

        }

        else

        {

            startX = startY = resultX = resultY = obstacleX = obstacleY = -1;

        }

    }

关于棋盘处理，检测合法棋步如下：

bool Board::ValidMove(int x1, int y1, int x2, int y2, int x3, int y3, int color)

{

    if (!InGrid(x1, y1) || !InGrid(x2, y2) || !InGrid(x3, y3)) return false;

    if (grid[x1][y1] != color || grid[x2][y2] != 0) return false;

    if (grid[x2][y2] != 0 && !(x3 == x1 && y3 == y1)) return false;

    if (!OnStraghtLine(x1, y1, x2, y2) || !OnStraghtLine(x2, y2, x3, y3)) return false;

    ApplyMove(x1, y1, x2, y2, x3, y3, false);

    bool flag = HaveObstacleBetween(x1, y1, x2, y2) || HaveObstacleBetween(x2, y2, x3, y3);

    ApplyMove(x1, y1, x2, y2, x3, y3, true);

    return !flag;

}

    if (startX == -1) // There is no move left for bot

    {

        cout << "You win!\n";

        isRunning = false;

        return;

    }

    else

    {

        board.ApplyMove(startX, startY, resultX, resultY, obstacleX, obstacleY, false);

        board.DrawBoard();

        cout << "Turn: " << board.turnID << "\n";

        cout << "Computer move: " << startX << ' ' << startY << ' ' << resultX << ' '

            << resultY << ' ' << obstacleX << ' ' << obstacleY << "\n";

    }

    if (board.LoseState(-BotColor)) // Player in lose state

    {

        cout << "Computer win!\n";

        isRunning = false;

        return;

    }

}

其他没有什么可说

1. 独特地方

- 严格地处理玩家的输入，比如在开始菜单：

    cout << "1. New game\n";

    cout << "2. Continue last game\n";

    cout << "3. Quit game\n";

    cout << "Please select [1/2/3]: ";

    string line;

    int ops;

    while (true)

    {

        getline(cin, line);

        if (line != "1" && line != "2" && line != "3")

        {

            cout << "Invalid input! Please re-select [1/2/3]: ";

        }

        else

        {

            stringstream st(line);

            st >> ops;

            break;

        }

    }

或者在Select move

getline(cin, line);

bool ok = true;

int space = 0;

for (int i = 0; i < line.size(); i++)

{

    if (line[i] == ' ') space++;

    else if (line[i] < '0' || line[i] > '9') ok = false;

}

// valid input only contains digit and has 6 numbers

if (!ok || space != 5)

{

    cout << "Invalid input! Please re-enter [x1 y1 x2 y2 x3 y3]: ";

}

* 利用vector具有comparision以便用map记录棋盘状态。
* 程序结构清楚，大概内容可以在头文件理解