Amazons

From Botzone Wiki

Contents

Amazons

简介 Amazons(亚马逊棋),是1988年由阿根廷人WalteZamkauska发明的双人棋类游戏，由国际象棋中后的走法衍生而来，属于两人零和完备信息博弈。目前亚马逊棋是ICGA国际机器博弈锦标赛和全国大学生计算机博弈大赛的比赛项目之一。

游戏规则

棋盘：由N*N的方格组成，黑方在棋盘上方，白方在棋盘下方(坐标从0开始，先x后y, 原点在左上角，本游戏N=8)

棋子：每方有4个棋子（4个Amazons）,初始位置分别位于（0,2),(2,0),(5,0),(7,2)和(0,5),(2,7),(5,7),(7,5)

1.每个棋子都相当于国际象棋中的皇后,它们的行棋方法与皇后相同,可以在8个方向（上、下、左、右、左上、左下、右上、右下）上任意行走，但不能穿过阻碍

2.当轮到一方行棋时，此方只能而且必须移动4个Amazons中的一个，并在移动完成后，由当前移动的棋子释放一个障碍，障碍的释放方法与棋子的移动方法相同（8个方向，但不能穿过障碍），同样障碍的放置也是必须的

3.当某方完成某次移动后，对方4个棋子均不能再移动时，对方将输掉比赛

4.每次开局位于棋盘上方的玩家(黑方)先手;

5.整个比赛中双方均不能吃掉对方或己方的棋子或障碍

游戏交互方式

本游戏与Botzone上其他游戏一样，使用相同的交互方式：Bot#交互

具体交互内容

本游戏支持两种交互方式。推荐初学者选择简单交互。

简单交互

简单交互的基础是：Bot#简化交互

其中，本游戏的request行和response行都为如下格式:

```
x0 y0 x1 y1 x2 y2
```

黑方的第一回合的request是:

```
-1 -1 -1 -1 -1 -1
```

- x0：选择的棋子横坐标
- y0：选择的棋子纵坐标
- x1：棋子移动终点横坐标
- y1：棋子移动终点纵坐标
- x2：释放的障碍横坐标
- y2：释放的障碍纵坐标

举例：在样例对局 (https://www.botzone.org.cn/match/5bd137c30681335cc1f4d093)中的第5回合，黑方得到的输入是:

```
3
-1 -1 -1 -1 -1 -1
```

```
5 0 3 2 6 5
0 5 4 5 3 4
0 2 0 3 3 0
2 7 2 5 1 4
```

- 3表示这是自己的第3回合（也就是第3次轮到自己决策；上文的第5回合表示这是全游戏的第5步）。
- 第2、4、6行是对手的决策。
- 第1、3行是自己的决策。

JSON 交互

每回合Bot收到的request不是字符串，而是一个JSON对象，表示对方落子位置。格式如下：

```
{
                    "x0": Number, // 起点横坐标
                    "y0": Number, // 起点纵坐标
                    "x1": Number, // 终点横坐标
                    "y1": Number, // 终点纵坐标
                    "x2": Number, // 释放的障碍横坐标
                    "y2": Number  // 释放的障碍纵坐标
}
```

Bot所需要输出的response也是格式相同的JSON对象，表示自己选择移动的棋子的起点、终点和障碍的位置。

如果是黑方的第一回合，则 request 为 {"x0": -1, "y0": -1, "x1" : -1, "y1" : -1, "x2" : -1, "y2" : -1}。

游戏样例程序

推荐初学者使用简单交互的样例程序。请记得在提交Bot时勾选“使用简单交互”。

简单交互的 C++ 样例程序

```cpp
// 亚马逊棋（Amazons）简单交互样例程序
// 随机策略（由zhouhy修改）
// 作者：dgf123/syys
// 游戏信息：http://www.botzone.org/games#Amazons


#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>


#define GRIDSIZE 8
#define OBSTACLE 2
#define judge_black 0
#define judge_white 1
#define grid_black 1
#define grid_white -1
```

```cpp
using namespace std;

int currBotColor; // 我所执子颜色（1为黑，-1为白，棋盘状态亦同）
int gridInfo[GRIDSIZE][GRIDSIZE] = { 0 }; // 先x后y，记录棋盘状态
int dx[] = { -1,-1,-1,0,0,1,1,1 };
int dy[] = { -1,0,1,-1,1,-1,0,1 };

// 判断是否在地图内
inline bool inMap(int x, int y)
{
    if (x < 0 || x >= GRIDSIZE || y < 0 || y >= GRIDSIZE)
        return false;
    return true;
}


// 在坐标处落子，检查是否合法或模拟落子
bool ProcStep(int x0, int y0, int x1, int y1, int x2, int y2, int color, bool check_only)
{
    if ((!inMap(x0, y0)) || (!inMap(x1, y1)) || (!inMap(x2, y2)))
        return false;
    if (gridInfo[x0][y0] != color || gridInfo[x1][y1] != 0)
        return false;
    if ((gridInfo[x2][y2] != 0) && !(x2 == x0 && y2 == y0))
        return false;
    if (!check_only)
    {
        gridInfo[x0][y0] = 0;
        gridInfo[x1][y1] = color;
        gridInfo[x2][y2] = OBSTACLE;
    }
    return true;
}

int main()
{
    int x0, y0, x1, y1, x2, y2;

    // 初始化棋盘
    gridInfo[0][(GRIDSIZE - 1) / 3] = gridInfo[(GRIDSIZE - 1) / 3][0]
        = gridInfo[GRIDSIZE - 1 - ((GRIDSIZE - 1) / 3)][0]
        = gridInfo[GRIDSIZE - 1][(GRIDSIZE - 1) / 3] = grid_black;
    gridInfo[0][GRIDSIZE - 1 - ((GRIDSIZE - 1) / 3)] = gridInfo[(GRIDSIZE - 1) / 3][GRIDSIZE - 1]
        = gridInfo[GRIDSIZE - 1 - ((GRIDSIZE - 1) / 3)][GRIDSIZE - 1]
        = gridInfo[GRIDSIZE - 1][GRIDSIZE - 1 - ((GRIDSIZE - 1) / 3)] = grid_white;


    int turnID;
    cin >> turnID;

    // 读入到当前回合为止，自己和对手的所有行动，从而把局面恢复到当前回合
    currBotColor = grid_white; // 先假设自己是白方
    for (int i = 0; i < turnID; i++)
    {
        // 根据这些输入输出逐渐恢复状态到当前回合

        // 首先是对手行动
        cin >> x0 >> y0 >> x1 >> y1 >> x2 >> y2;
        if (x0 == -1)
            currBotColor = grid_black; // 第一回合收到坐标是-1, -1，说明我是黑方
        else
            ProcStep(x0, y0, x1, y1, x2, y2, -currBotColor, false); // 模拟对方落子

        // 然后是自己当时的行动
        // 对手行动总比自己行动多一个
        if (i < turnID - 1)
        {
            cin >> x0 >> y0 >> x1 >> y1 >> x2 >> y2;
```

```
            if (x0 >= 0)
                ProcStep(x0, y0, x1, y1, x2, y2, currBotColor, false); // 模拟己方落子
        }
    }

    // 做出决策（你只需修改以下部分）

    // 这里枚举了所有可能的下法，以便之后随机用……

    int beginPos[3000][2], possiblePos[3000][2], obstaclePos[3000][2];
    int posCount = 0, choice;
    for (int i = 0; i < GRIDSIZE; ++i) {
        for (int j = 0; j < GRIDSIZE; ++j) {
            for (int k = 0; k < 8; ++k) {
                for (int delta1 = 1; delta1 < GRIDSIZE; delta1++) {
                    int xx = i + dx[k] * delta1;
                    int yy = j + dy[k] * delta1;
                    if (gridInfo[xx][yy] != 0 || !inMap(xx, yy))
                        break;
                    for (int l = 0; l < 8; ++l) {
                        for (int delta2 = 1; delta2 < GRIDSIZE; delta2++) {
                            int xxx = xx + dx[l] * delta2;
                            int yyy = yy + dy[l] * delta2;
                            if (!inMap(xxx, yyy))
                                break;
                            if (gridInfo[xxx][yyy] != 0 && !(i == xxx && j == yyy))
                                break;
                            if (ProcStep(i, j, xx, yy, xxx, yyy, currBotColor, true))
                            {
                                beginPos[posCount][0] = i;
                                beginPos[posCount][1] = j;
                                possiblePos[posCount][0] = xx;
                                possiblePos[posCount][1] = yy;
                                obstaclePos[posCount][0] = xxx;
                                obstaclePos[posCount++][1] = yyy;
                            }
                        }

                    }
                }

            }
        }
    }

    int startX, startY, resultX, resultY, obstacleX, obstacleY;
    if (posCount > 0)
    {
        srand(time(0));
        choice = rand() % posCount;
        startX = beginPos[choice][0];
        startY = beginPos[choice][1];
        resultX = possiblePos[choice][0];
        resultY = possiblePos[choice][1];
        obstacleX = obstaclePos[choice][0];
        obstacleY = obstaclePos[choice][1];
    }
    else
    {
        startX = -1;
        startY = -1;
        resultX = -1;
        resultY = -1;
        obstacleX = -1;
        obstacleY = -1;
    }

    // 决策结束，输出结果（你只需修改以上部分）
    cout << startX << ' ' << startY << ' ' << resultX << ' ' << resultY << ' ' << obstacleX << ' ' << obstac
```

```
    return 0;
}
```

JSON 交互的 C++ 样例程序

本地编译方式请参看JSONCPP。

```
// 亚马逊棋（Amazons）样例程序
// 随机策略
// 作者：dgf123/syys
// 游戏信息：http://www.botzone.org/games#Amazons

//#include "stdafx.h"

#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
#include "jsoncpp/json.h"


#define GRIDSIZE 8
#define OBSTACLE 2
#define judge_black 0
#define judge_white 1
#define grid_black 1
#define grid_white -1

using namespace std;

int currBotColor; // 我所执子颜色（1为黑，-1为白，棋盘状态亦同）
int gridInfo[GRIDSIZE][GRIDSIZE] = { 0 }; // 先x后y，记录棋盘状态
int dx[] = { -1,-1,-1,0,0,1,1,1 };
int dy[] = { -1,0,1,-1,1,-1,0,1 };

// 判断是否在地图内
inline bool inMap(int x, int y)
{
    if (x < 0 || x >= GRIDSIZE || y < 0 || y >= GRIDSIZE)
        return false;
    return true;
}


// 在坐标处落子，检查是否合法或模拟落子
bool ProcStep(int x0, int y0, int x1, int y1, int x2, int y2, int color, bool check_only)
{
    if ((!inMap(x0, y0)) || (!inMap(x1, y1)) || (!inMap(x2, y2)))
        return false;
    if (gridInfo[x0][y0] != color || gridInfo[x1][y1] != 0)
        return false;
    if ((gridInfo[x2][y2] != 0) && !(x2 == x0 && y2 == y0))
        return false;
    if (!check_only)
    {
        gridInfo[x0][y0] = 0;
        gridInfo[x1][y1] = color;
        gridInfo[x2][y2] = OBSTACLE;
    }
    return true;
}
```

```
int main()
{
    int x0, y0, x1, y1, x2, y2;

    // 初始化棋盘
    gridInfo[0][(GRIDSIZE-1)/3] = gridInfo[(GRIDSIZE - 1) / 3][0]
                = gridInfo[GRIDSIZE - 1 - ((GRIDSIZE - 1) / 3)][0]
                = gridInfo[GRIDSIZE-1][(GRIDSIZE - 1) / 3] = grid_black;
    gridInfo[0][GRIDSIZE - 1 - ((GRIDSIZE - 1) / 3)] = gridInfo[(GRIDSIZE - 1) / 3][GRIDSIZE - 1]
                = gridInfo[GRIDSIZE - 1 - ((GRIDSIZE - 1) / 3)][GRIDSIZE - 1]
                = gridInfo[GRIDSIZE - 1][GRIDSIZE-1-((GRIDSIZE-1)/3)] = grid_white;

    // 读入JSON
    string str;
    getline(cin, str);
    Json::Reader reader;
    Json::Value input;
    reader.parse(str, input);

    // 分析自己收到的输入和自己过往的输出，并恢复状态
    int turnID = input["responses"].size();
    currBotColor = input["requests"][(Json::Value::UInt) 0]["x0"].asInt() < 0 ? grid_black : grid_white; // 
    for (int i = 0; i < turnID; i++)
    {
        // 根据这些输入输出逐渐恢复状态到当前回合
        x0 = input["requests"][i]["x0"].asInt();
        y0 = input["requests"][i]["y0"].asInt();
        x1 = input["requests"][i]["x1"].asInt();
        y1 = input["requests"][i]["y1"].asInt();
        x2 = input["requests"][i]["x2"].asInt();
        y2 = input["requests"][i]["y2"].asInt();
        if (x0 >= 0)
            ProcStep(x0, y0, x1, y1, x2, y2, -currBotColor, false); // 模拟对方落子
        x0 = input["responses"][i]["x0"].asInt();
        y0 = input["responses"][i]["y0"].asInt();
        x1 = input["responses"][i]["x1"].asInt();
        y1 = input["responses"][i]["y1"].asInt();
        x2 = input["responses"][i]["x2"].asInt();
        y2 = input["responses"][i]["y2"].asInt();
        if (x0 >= 0)
            ProcStep(x0, y0, x1, y1, x2, y2, currBotColor, false); // 模拟己方落子
    }

    // 看看自己本回合输入
    x0 = input["requests"][turnID]["x0"].asInt();
    y0 = input["requests"][turnID]["y0"].asInt();
    x1 = input["requests"][turnID]["x1"].asInt();
    y1 = input["requests"][turnID]["y1"].asInt();
    x2 = input["requests"][turnID]["x2"].asInt();
    y2 = input["requests"][turnID]["y2"].asInt();
    if (x0 >= 0)
        ProcStep(x0, y0, x1, y1, x2, y2, -currBotColor, false); // 模拟对方落子
    int beginPos[3000][2], possiblePos[3000][2], obstaclePos[3000][2];
    int posCount = 0, choice;
    for (int i = 0; i < GRIDSIZE; ++i) {
        for (int j = 0; j < GRIDSIZE; ++j) {
            for (int k = 0; k < 8; ++k) {
                for (int delta1 = 1; delta1 < GRIDSIZE; delta1++) {
                    int xx = i + dx[k] * delta1;
                    int yy = j + dy[k] * delta1;
                    if (gridInfo[xx][yy] != 0 || !inMap(xx, yy))
                        break;
                    for (int l = 0; l < 8; ++l) {
                        for (int delta2 = 1; delta2 < GRIDSIZE; delta2++) {
                            int xxx = xx + dx[l] * delta2;
                            int yyy = yy + dy[l] * delta2;
                            if (!inMap(xxx, yyy))
                                break;
                            if (gridInfo[xxx][yyy] != 0 && !(i == xxx && j == yyy))
```

```
                            break;
                        if (ProcStep(i, j, xx, yy, xxx, yyy, currBotColor, true))
                        {
                            beginPos[posCount][0] = i;
                            beginPos[posCount][1] = j;
                            possiblePos[posCount][0] = xx;
                            possiblePos[posCount][1] = yy;
                            obstaclePos[posCount][0] = xxx;
                            obstaclePos[posCount++][1] = yyy;
                        }
                    }

                }
            }

        }
    }

    // 做出决策（你只需修改以下部分）

    int startX, startY, resultX, resultY, obstacleX, obstacleY;
    if (posCount > 0)
    {
        srand(time(0));
        choice = rand() % posCount;
        startX = beginPos[choice][0];
        startY = beginPos[choice][1];
        resultX = possiblePos[choice][0];
        resultY = possiblePos[choice][1];
        obstacleX = obstaclePos[choice][0];
        obstacleY = obstaclePos[choice][1];
    }
    else
    {
        startX = -1;
        startY = -1;
        resultX = -1;
        resultY = -1;
        obstacleX = -1;
        obstacleY = -1;
    }

    // 决策结束，输出结果（你只需修改以上部分）

    Json::Value ret;
    ret["response"]["x0"] = startX;
    ret["response"]["y0"] = startY;
    ret["response"]["x1"] = resultX;
    ret["response"]["y1"] = resultY;
    ret["response"]["x2"] = obstacleX;
    ret["response"]["y2"] = obstacleY;
    ret["debug"]["posCount"] = posCount;
    Json::FastWriter writer;
    cout << writer.write(ret) << endl;
    return 0;
}
```

参考资料

1. An evaluation function for the game of amazons (http://download.springer.com/static/pdf/793/chp%253A10.1007%252F978-0-387-35706-5_19.pdf?originUrl=https%3A%2F%2Flink.springer.com%2Fchapter%2F10.1007%2F978-0-387-35706-5_19&token2=exp=1498449804~acl=%2Fstatic%2Fpdf%2F793%2Fchp%25

253A10.1007%25252F978-0-387-35706-5_19.pdf%3ForiginUrl%3Dhttps%253A%252F%252Flink.springer.com%252Fchapter%252F10.1007%252F978-0-387-35706-5_19*~hmac=e36a00e70de2eb80542cf63998960074b96dc038035ee2d103ec37c916a12b90)

2. Amazons Discover Monte−Carlo (http://download.springer.com/static/pdf/483/chp%253A10.1007%252F978-3-540-87608-3_2.pdf?originUrl=https%3A%2F%2Flink.springer.com%2Fchapter%2F10.1007%2F978-3-540-87608-3_2&token2=exp=1498450011~acl=%2Fstatic%2Fpdf%2F483%2Fchp%25253A10.1007%25252F978-3-540-87608-3_2.pdf%3ForiginUrl%3Dhttps%253A%252F%252Flink.springer.com%252Fchapter%252F10.1007%252F978-3-540-87608-3_2*~hmac=b22f42fcaa1f5f5f33ff91142164b8c146985a4ca722fba3ee4b8a31c1d213e1)

3. The Monte-Carlo Approach in Amazons (http://www.mi.parisdescartes.fr/~bouzy/publications/KIB-MCAmazons-CGW07.pdf)