TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TPHCM

VIỆN ĐÀO TẠO CLC

BÁO CÁO MÔN HỌC

MÔN HỌC KỸ THUẬT LẬP TRÌNH C++

BÁO CÁO HỌC KÌ II

2023 – 2024

GAME GREEDY ROBOTS

Sinh viện thực hiện: Quách Phú Thuận

MSSV: 2251120446

Lớp: CN22CLCE

Giảng viên hướng dẫn: Nguyễn Văn Huy

Hồ Chí Minh, 2023

MỤC LỤC

MỞ ĐẦU

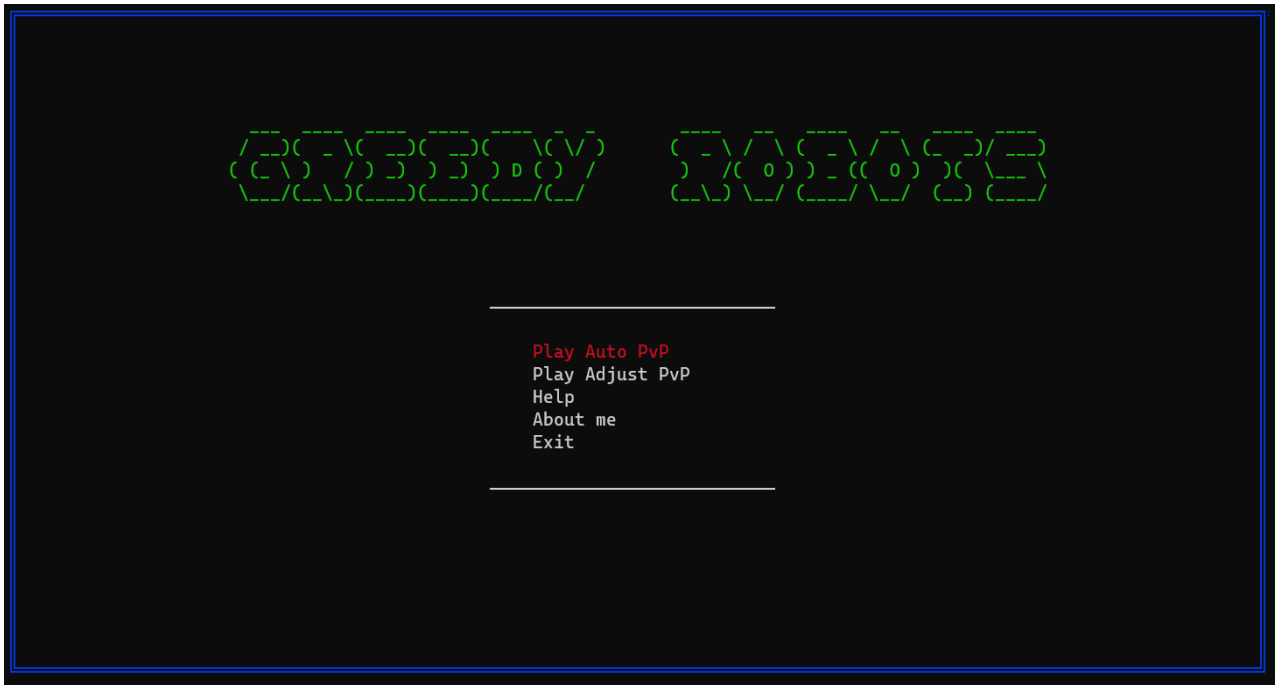
Nội dung cần báo cáo

1) Tất cả các kỹ thuật, thuận toán sinh viên đã tìm hiểu để thực hiện đồ án.

2) Mô tả các đoạn mã và các ý của từng chức năng trong đồ án.

3) Nêu rõ nguồn tham khảo

Giao diện Game :



GIỚI THIỆU CÁC CHỨC NĂNG GAME

1. Auto PvP : Robot tự động di chuyển dựa vào vị trí ban đầu người chơi nhập

+ Normal Mode: robot thắng khi có tổng lớn nhất, không được đi ô trùng nhau với robot khác, không di chuyển ra khỏi mê cung và mỗi lượt chỉ di chuyển một ô.

+ Intersection Mode: robot thắng khi có tổng lớn nhất, được phép đi ô trùng nhau với robot khác, không di chuyển ra khỏi mê cung. Xuất kết quả những ô mà hai Robots cắt nhau.

2. Adjust PvP: Robot di chuyển tùy ý theo 4 hướng (trên, dưới, trái, phải) tùy người dùng, robot thắng khi có tổng điểm lớn nhất, không được đi những ô trùng nhau, mỗi lượt chỉ di chuyển một ô.
3. Help: Hướng dẫn luật chơi và các phím tắt: ECS, Enter, P,... cho game
4. About me: Thông tin cá nhân của author
5. Exit: Thoát trò chơi, trước khi thoát sẽ hỏi người chơi có thực sự muốn thoát không

Phần 1: Kỹ thuật, kiến thức đã tìm hiểu và sử dụng

Báo cáo môn kỹ thuật lập trình C++

Game Greedy Robots được xây dựng bằng C++.

Game có 2 chế độ chính: tự động (Auto PvP) và điều chỉnh (Adjust PvP).

Auto PvP có 2 chế độ: Normal (di chuyển không trùng ô), Intersection (di chuyển trùng ô).

Kiến thức đã sử dụng trong game:

- Struct: để lưu tọa độ hai robots, bước đi và quãng đường robots di chuyển
- Đọc / Ghi file: lưu lại quãng đường robots và số bước vào file .txt để phòng trường hợp bị mất ván đấu khi đang chơi game
- Hàm (function): để tái sử dụng nhiều lần tại nhiều vị trí trong chương trình
- Cấp phát động: sử dụng kiến thức vector (mảng động tự co dãn tùy kích thước quãng đường trong game), con trỏ.
- Đệ quy: để tự gọi lại hàm di chuyển đến khi không thể di chuyển được nữa

Kỹ thuật chính để di chuyển robots:

- ☞ Tạo struct lưu vị trí (x, y) của hai robots, lưu steps và quãng đường robot đã đi qua. Tạo hàm bool passed[row][col] = {false}; ô nào đi qua thì sẽ đánh dấu là true để không phải đi lại ô đó.
- ☞ Tạo hàm printMaze để hiển thị vị trí hiện tại của robot trên ma trận lên màn hình.
- ☞ Tạo hàm bool isMoveValid để kiểm tra xem ô di chuyển có hợp lệ hay không, di chuyển không hợp lệ khi robot đi ra ngoài mê cung, đi những ô đã đi qua và đi vào ô robot khác.
- ☞ Tạo hàm di chuyển bằng cách lúc đầu tạo biến max_value = -1 và max_index = -1. Dùng vòng lặp để duyệt qua 4 ô xung quanh, nếu một trong 4 ô lớn hơn max_value thì max_value sẽ có giá trị mới bằng ô lớn hơn đó đồng thời sẽ đánh lại max_index. Nếu 4 ô xung quanh không di chuyển được nữa do max_value xung quanh = -1.

Các kỹ thuật phụ :

- Kỹ thuật chèn âm thanh, hiệu ứng cho game sinh động.
- Kỹ thuật tô màu kí tự và căn chỉnh màn hình console . (Dùng để làm giao diện và hiệu ứng thắng thua)
- Kỹ thuật xử lí tạo Menu Game .
- Kỹ thuật đệ quy (để quay trở về Menu và thực hiện tiếp)
- Sử dụng vòng lặp và lệnh Sleep() để tạo hiệu ứng chữ chớp
- Kỹ thuật ẩn con trỏ

Phần 2 : Mô tả code và các ý của từng chức năng code

Các hàm hỗ trợ để vẽ đồ họa console:

Hàm gotoxy để di chuyển đến một vị trí bất kì trên màn hình

```
void gotoxy (int column, int line){
	COORD coord;
	coord.X = column;
	coord.Y = line;
	SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),coord);
}
```

Hàm TextColor để thiết lập màu sắc cho code

```
void TextColor (int color){
	SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE) , color);
}
```

Hàm clrscr để xóa toàn bộ màn hình

```
void clrscr(){
	CONSOLE_SCREEN_BUFFER_INFO	csbiInfo;
	HANDLE		hConsoleOut;
	COORD Home = {0,0};
	DWORD		dummy;

	hConsoleOut = GetStdHandle(STD_OUTPUT_HANDLE);
	GetConsoleScreenBufferInfo(hConsoleOut,&csbiInfo);

	FillConsoleOutputCharacter(hConsoleOut,' ',csbiInfo.dwSize.X * csbiInfo.dwSize.Y,Home,&dummy);
	csbiInfo.dwCursorPosition.X = 0;
	csbiInfo.dwCursorPosition.Y = 0;
	SetConsoleCursorPosition(hConsoleOut,csbiInfo.dwCursorPosition);
}
```

Hàm hideCursor để ẩn con trỏ chuột

```
void hideCursor(){
```

```
    HANDLE consoleHandle = GetStdHandle(STD_OUTPUT_HANDLE);


    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(consoleHandle, &cursorInfo);
    cursorInfo.bVisible = false;
    SetConsoleCursorInfo(consoleHandle, &cursorInfo);
}
```

Các hàm để Robot di chuyển:

Hàm veBanCo để vẽ bàn cờ cùng với vị trí hiện tại của robot lên màn hình

Với maze là vector 2 chiều, Robot A đi qua ô nào thì sẽ đánh dấu X, Robot B là O để nhận biết ô đó đã đi qua cho dễ nhìn.

```
void veBanCo(const vector<vector<int> >& maze, int robot1X, int robot1Y, int robot2X, int robot2Y, int score1, int score2) {
gotoxy(0,0);
TextColor(10);
    int m = maze.size();
    int n = maze[0].size();
    cout << char(218);
    for (int i = 0; i < n - 1; i++) {
        cout << char(196) << char(196) << char(196) << char(194);
    }
    cout << char(196) << char(196) << char(196) << char(191) << endl;
    for (int i = 0; i < m - 1; i++) {
        cout << char(179);
        for (int k = 0; k < n - 1; k++) {
            if (i == robot1X && k == robot1Y) {
              TextColor(7);
               cout << " A ";
               TextColor(10);
              cout << char(179);
            } else if (i == robot2X && k == robot2Y) {
              TextColor(7);
```

```
            cout << " B ";
            TextColor(10);
          cout << char(179);
        } else if (maze[i][k] == -2) {
          TextColor(1);
            cout << " O ";
            TextColor(10);
          cout << char(179);
        } else if (maze[i][k] == -1) {
            TextColor(4);
            cout << " X ";
            TextColor(10);
          cout << char(179);
          } else {
          cout << " ";
         TextColor(13);
          cout << maze[i][k];
          cout << " ";
          TextColor(10);
          cout << char(179);
        }
    }
    if (i == robot1X && n - 1 == robot1Y) {
        TextColor(7);
        cout << " A ";
        TextColor(10);
          cout << char(179);
    } else if (i == robot2X && n - 1 == robot2Y) {
        TextColor(7);
        cout << " B ";
        TextColor(10);
```

```
            cout << char(179);
        } else if (maze[i][n-1] == -2) {
            TextColor(1);
             cout << " O ";
             TextColor(10);
            cout << char(179);
          } else if (maze[i][n-1] == -1) {
             TextColor(4);
             cout << " X ";
             TextColor(10);
            cout << char(179);
            } else {
          cout << " ";
          TextColor(13);
            cout << maze[i][n - 1];
            cout << " ";
            TextColor(10);
            cout << char(179);
        }
        cout << endl;
        TextColor(10);
        cout << char(195);
        for (int j = 0; j < n - 1; j++) {
            cout << char(196) << char(196) << char(196) << char(197);
        }
        cout << char(196) << char(196) << char(196) << char(180) << endl;
    }
    cout << char(179);
    for (int k = 0; k < n - 1; k++) {
        if (m - 1 == robot1X && k == robot1Y) {
            TextColor(7);
```

```
            cout << " A ";
            TextColor(10);
              cout << char(179);
        } else if (m - 1 == robot2X && k == robot2Y) {
            TextColor(7);
            cout << " B ";
            TextColor(10);
              cout << char(179);
        } else if (maze[m-1][k] == -2) {
            TextColor(1);
            cout << " O ";
            TextColor(10);
              cout << char(179);
        } else if (maze[m-1][k] == -1) {
            TextColor(4);
            cout << " X ";
            TextColor(10);
              cout << char(179);
              }
              else {
            cout << " ";
            TextColor(13);
              cout << maze[m - 1][k];
              cout << " ";
              TextColor(10);
              cout << char(179);
        }
    }
    if (m - 1 == robot1X && n - 1 == robot1Y) {
        TextColor(7);
        cout << " A ";
```

```
    TextColor(10);
        cout << char(179);
} else if (m - 1 == robot2X && n - 1 == robot2Y) {
    TextColor(7);
    cout << " B ";
    TextColor(10);
        cout << char(179);
} else if (maze[m-1][n-1] == -2) {
    TextColor(1);
    cout << " O ";
    TextColor(10);
        cout << char(179);
} else if (maze[m-1][n-1] == -1) {
    TextColor(4);
    cout << " X ";
    TextColor(10);
        cout << char(179);
        }
        else {
    cout << " ";
    TextColor(13);
        cout << maze[m - 1][n - 1];
        cout << " ";
        TextColor(10);
        cout << char(179);
}
cout << endl;
TextColor(10);
cout << char(192);

for (int i = 0; i < n - 1; i++) {
```

```
        cout << char(196) << char(196) << char(196) << char(193);
    }
    cout << char(196) << char(196) << char(196) << char(217);
    //Score cua 2 Robots
    gotoxy(87, 6);
    TextColor(4);
    cout << score1;
    gotoxy(87, 15);
    TextColor(1);
    cout << score2;
}
```

Hàm isMoveValid để kiểm tra vị trí di chuyển tới có hợp lệ hay không

```
bool isMoveValid(int currentX, int currentY, int newX, int newY, vector<vector<int> > maze, int robot1X, int robot1Y, int robot2X, int robot2Y) {
    if (newX < 0 || newX >= maze.size() || newY < 0 || newY >= maze[0].size()) {
        // Robot has gone outside the maze!
        return false;
    }
    if (maze[newX][newY] == -1 || maze[newX][newY] == -2) {
        // This position has been visited before!
        return false;
    }
    if ((newX == robot1X && newY == robot1Y) || (newX == robot2X && newY == robot2Y)) {
        // Robot cannot move into the other robot's cell!
        return false;
    }
    return true;
}
```

Hàm move để di chuyển robot, sau khi di chuyển sẽ tiếp tục đệ quy lại hàm xem còn di chuyển được không

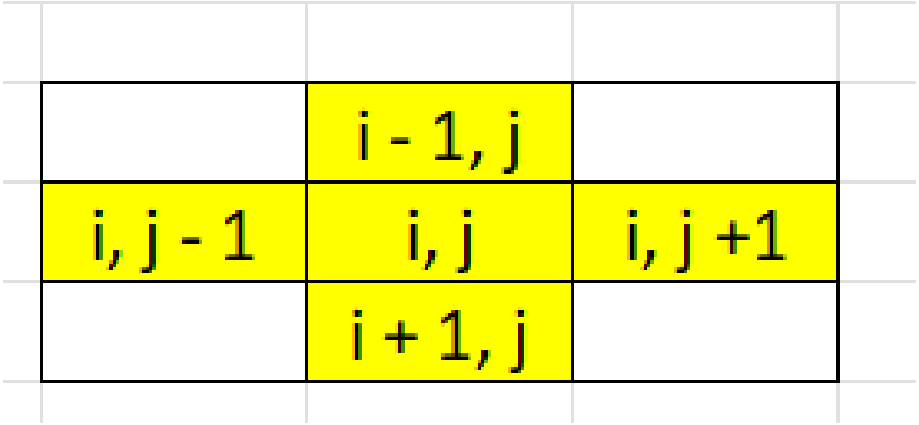
```
void move(robot& RobotA, robot& RobotB, vector<vector<int> >& maze, int numRows, int numCols) {
    int currentX, currentY, newX, newY;
```

```
// Robot A's turn
currentX = RobotA.x;
currentY = RobotA.y;
// Find the maximum value among the neighboring cells
int maxScore = -1;
int maxIndex = -1;
for (int i = 0; i < 4; i++) {
    newX = currentX + direction_x[i];
    newY = currentY + direction_y[i];

    if (isMoveValid(currentX, currentY, newX, newY, maze, RobotA.x, RobotA.y, RobotB.x, RobotB.y)) {
        if (maze[newX][newY] > maxScore) {
            maxScore = maze[newX][newY];
            maxIndex = i;
        }
    }
}
// Move Robot A to the cell with the maximum score
if (maxIndex != -1) {
    newX = currentX + direction_x[maxIndex];
    newY = currentY + direction_y[maxIndex];
    RobotA.x = newX;
    RobotA.y = newY;
    RobotA.steps++;
    gotoxy(93, 8);
        TextColor(4);
        cout << RobotA.steps;
    RobotA.score += maze[newX][newY];
    RobotA.path.push_back(maze[newX][newY]);
    maze[currentX][currentY] = -1;
    maze[newX][newY] = -1;
```

```
}
if (maxIndex == -1 || maxIndex == -2){
        calculateWinner(RobotA, RobotB);
        return;
}
// Print the maze after Robot A's move
veBanCo(maze, RobotA.x, RobotA.y, RobotB.x, RobotB.y, RobotA.score, RobotB.score);
Sleep(20);
        // Robot B's turn
currentX = RobotB.x;
currentY = RobotB.y;
// Find the maximum value among the neighboring cells
maxScore = -2;
maxIndex = -2;
for (int i = 0; i < 4; i++) {
   newX = currentX + direction_x[i];
   newY = currentY + direction_y[i];

   if (isMoveValid(currentX, currentY, newX, newY, maze, RobotA.x, RobotA.y, RobotB.x, RobotB.y)) {
      if (maze[newX][newY] > maxScore) {
         maxScore = maze[newX][newY];
         maxIndex = i;
      }
   }
}
// Move Robot B to the cell with the maximum score
if (maxIndex != -2) {
   newX = currentX + direction_x[maxIndex];
   newY = currentY + direction_y[maxIndex];
   RobotB.x = newX;
   RobotB.y = newY;
```

```
        RobotB.steps++;
        gotoxy(93, 17);
            TextColor(1);
            cout << RobotB.steps;
        RobotB.score += maze[newX][newY];
        RobotB.path.push_back(maze[newX][newY]);
        maze[currentX][currentY] = -2;
        maze[newX][newY] = -2;
    }
    // Print the maze after Robot B's move
    veBanCo(maze, RobotA.x, RobotA.y, RobotB.x, RobotB.y, RobotA.score, RobotB.score);
    Sleep(20);
    // Recursive call for the next round
    move(RobotA, RobotB, maze, numRows, numCols);
}
```

Hình minh họa về một trường hợp di chuyển của robot từ một vị trí (i, j) bất kì



TÀI LIỆU THAM KHẢO

1. Vẽ bàn cờ caro : https://www.youtube.com/watch?v=AOvXmLpucXk
2. Các lệnh chèn âm thanh , tô màu kí tự, ẩn trỏ chuột , thay đổi kích thước console trên các diễn đàn học lập trình.
3. Tài liệu hướng dẫn lập trình: Sách Effective Modern C++ của Scott Meyers

Link Source Code game Greedy Robots:

https://drive.google.com/drive/folders/1BNEc7c0WnOFovJ7joUol0jJfbejYpGrG?usp=sharing