ĐẠI HỌC GIAO THÔNG VẬN TẢI

BÀI THU HOẠCH
AN NINH MẠNG

Giảng viên hướng dẫn: Phạm Thanh Hà
Lớp: Công nghệ thông tin 1 - K59
Sinh viên thực hiện:

Mã SV	Họ tên
181202289	Lê Quang Thọ

Hà Nội, 19 tháng 11 năm 2021

Nội dung

Mở Đầu

Trí tuệ nhân tạo ở đây là nói đến khả năng của máy khi thực hiện các công việc mà con người thường phải xử lý và khi dáng vẻ ứng xử hoặc kết quả thực hiệ n của máy là tốt hơn hoặc tương đương với con người thì ta gọi đó là máy thông m inh hay máy đó có trí thông minh.

Hay nói cách khác, đánh giá sự thông minh của máy không phải dựa trên nguyên l ý nó thực hiện nhiệm vụ đó có giống cách con người thực hiện hay không mà dựa trên kết quả hoặc dáng vẻ ứng xử bên ngoài của nó có giống với kết quả hoặc dáng vẻ ứng xử của con người hay không.

Các nhiệm vụ của con người thường xuyên phải thực hiện là: giải bài toán (tìm kiếm, chứng minh, lập luận), học, giao tiếp, thể hiện cảm xúc, thích nghi với m ôi trường xung quanh,

v.v., và dựa trên kết quả thực hiện các nhiệm vụ đó để kết luận rằng một ai đó có l à thông minh hay không.

1. BÀI TOÁN VÀ MÔI TRƯỜNG TRIỂN KHAI

1.1. Bài toán

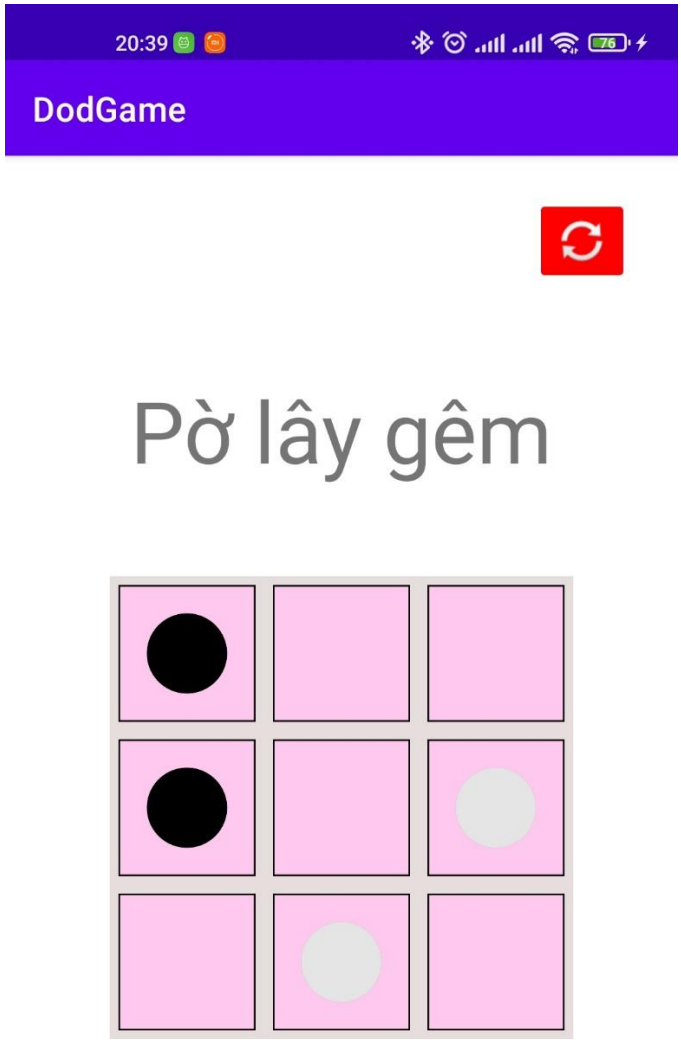
Bào toán đặt ra là phát triển một ứng dụng trò chơi DodGame trên chạy trên nền tảng Android có tích hợp trí tuệ nhân tạo AI.

Một ứng dụng có chế độ chơi giữa người và AI.

Phía AI sẽ được xây dựng dựa trên thuật toán Minimax để tìm nước đi tối ưu trong mỗi lượt đi và dành chiến thắng.

1.2. Môi trường triển khai

- IDE: Android Studio
- Ngôn ngữ: Java



Hình 1. Giao diện trò chơi

2. TRIỂN KHAI BÀI TOÁN

2.1. Các biến constant sử dụng trong bài toán

- Các constant sử dụng trong project sẽ được khai báo như sau với:
  + *CELL_BLACK_CAN_RUN* và *CELL_WHITE_CAN_RUN* là danh sách các hướng mà quân đen và quân trắng có thể di chuyển trong bàn cờ được đặt thành dạng số để thuận tiện trong việc sử dụng trong ma trận trạng thái của bàn cờ
  + *DEPTH* đây là biến xét độ sâu (độ khó) cho cây
  + *MAP_ROOT* là ma trận trạng thái mặc định ban đầu của một ván đấu
  + *POINTS_OF_WHITE* và *POINTS_OF_BLACK* là mảng lưu giá trị tương ứng với vị trí của các quân cờ để cho điểm phục vụ việc tính điểm và thuật toán MINMAX

```
public interface IMiniMax {
    int[] CELL_BLACK_CAN_RUN = {-3, 1, 3};
    int[] CELL_WHITE_CAN_RUN = {-1, -3, 1};
    int DEPTH = 5;
    int MAP_ROOT[] = {-1, 0, 0, -1, 0, 0, 0, 1, 1};
    int POINTS_OF_WHITE[] = {30, 35, 40, 15, 20, 25, 0, 5, 10};
    int POINTS_OF_BLACK[] =
                    {-10, -25, -40, -5, -20, -35, -0, -15, -30};
}
```

2.2. Lớp Minimax

- Lớp này xử lí logic chính thuật thoán Minimax.

```
public class Minimax {
    public int minVal(Node node, int depth) {...}

    public int maxVal(Node node, int depth) {...}

    public Node MiniMaxVal(Node node, int depth) {...}
}
```

- Hàm minVal

```
public int minVal(Node node, int depth) {
    int val = Integer.MAX_VALUE;

    if (depth == 0 || node.isNodeEnd()) {
        return node.getSumPoint();
    } else {
        for (Node node2 : node.getNextNodes()) {
            int t = maxVal(node2, depth - 1);
            if (val >= t) {
                val = t;
            }
        }
        return val;
    }
}
```

- Hàm maxVal

```
public int maxVal(Node node, int depth) {
    int val = Integer.MIN_VALUE;

    if (depth == 0 || node.isNodeEnd()) {
        return node.getSumPoint();
    } else {
        for (Node node2 : node.getNextNodes()) {
            int t = minVal(node2, depth - 1);
            if (val <= t) {
                val = t;
            }
        }
        return val;
    }
}
```

- Hàm MinimaxVal

```
public Node MiniMaxVal(Node node, int depth) {
    Node chooseNode = null;

    int val = Integer.MIN_VALUE;

    for (Node node2 : node.getNextNodes()) {
        int t = minVal(node2, depth - 1);
        if (val <= t) {
            val = t;
            chooseNode = node2;
        }
    }

    return chooseNode;
}
```

2.3. Lớp Node

```
public class Node {
    private ArrayList<Node> nextNodes;
    private int mapPoint[];
    private int depth;
    private boolean turnWhite;

    public Node(int[] mapPoint, int depth, boolean turnWhite)
{...}
    public ArrayList<Node> getNextNodes() {...}

    public int[] getMapPoint() {...}

    public int getSumPoint() {...}

    public boolean isNodeEnd() {...}

    public boolean isNodeEnd2() {...}

    private boolean canMove(int direction, int position) {...}
}
```

- Trong lớp này thực hiện xử lí đối tượng được gọi là Node (nghĩ là trạng thái của bàn cờ)
- Các lớp quan trọng trong lớp Node:
    - Hàm Constructor – hàm khởi tạo (`Node`)
    - Hàm `getSumPoint()` - hàm đánh giá mức độ lợi thế của quân trắng (quân AI)
    - 2 hàm `isNodeEnd()` và `isNodeEnd2()` - hàm kiểm tra xem trạng thái hiện tại của Node có phải là trạng thái kết thúc tức là 1 trong 2 bên không thể di chuyển (trắng hoặc đen chiến thắng)
    - Hàm `canMove(int direction, int position)` – kiểm tra xem tại vị trí được chọn (`direction`) và có được di chuyển tới vị trí (`direction`) hay không.
- Chi tiết hàm đánh giá (`getSumPoint()`)

```
public int getSumPoint() {
    int sum = 0;
    int numberBlack = 2;
    int numberWhite = 2;
    for (int i = 0; i < 9; i++) {
        if (this.mapPoint[i] == -1) {  // Quân Đen
            numberBlack--;
            sum += IMiniMax.POINTS_OF_BLACK[i];
            if (i < 6) {
                if (this.mapPoint[i + 3] == 1) {
                    sum -= 40;
                }
                if (i < 3 && this.mapPoint[i + 6] == 1) {
                    sum -= 30;
                }
            }
        } else {
            if (this.mapPoint[i] == 1) {  // Quân Trắng
                numberWhite--;
                sum += IMiniMax.POINTS_OF_WHITE[i];
                if (i % 3 != 0) {
                    if (this.mapPoint[i - 1] == -1) {
                        sum += 40;
                    }
                    if (i % 3 == 2 && this.mapPoint[i - 2] == -1) {
                        sum += 30;
                    }
                }
            }
        }
    }
    sum += (numberWhite - numberBlack) * 85;
    if (this.isNodeEnd() && !isNodeEnd2()) {
        if (this.turnWhite) {
            sum -= 100;
        } else {
            sum += 100;
        }
    }
    return sum;
}
```

- Hàm đánh giá sẽ dựa trên mảng chứa điểm lợi thế của hai bên – quân trắng và quân đen để tính toán và đưa ra kết quả đánh giá về trạng thái của quân trắng (quân AI).
- Đặc biệt, cải tiến thêm, ưu tiên cộng thêm điểm khuyến khích nếu quân cờ của cả hai bên thực hiện nước đi đưa quân của đội mình đi ra ngoài bàn cờ.

- Đối với hàm Constructor – hàm khởi tạo (`Node`)
  - Hàm này sẽ tự động sinh theo thuật toá đệ quy, sinh các trạng thái kề của trạng thái hiện tại cho đến khi không thể sinh thêm hoặc đạt đến độ sâu được quy định ở `IminiMax` (ở phần 1.1)

2.4. Thiết kế giao diện

Ở đây sẽ mô tả ngắn gọn về file giao diện trong project (activity_main.xml) và xử lí giao diện bên phía Java (MainActivity.java)

- 1 Group Layot (`ConstraintLayout`) bao toàn bộ. (activity_main.xml)
  - Một GridLayout 3x3 (đây chính là bàn cờ)
    - Bên trong sẽ chứa 9 item được gắn Tag và bắt sự kiện OnClick để khi click vào sẽ có phản hồi
  - 1 nút bấm sẽ hiện lên nếu quân được chọn có thể đi ra ngoài bàn cờ
  - 1 nút dùng để reset game – chơi lại từ đâu
- Xử lí tương tác của người dùng với giao diện (MainActivity.java)
  - Khi chương trình khởi chạy sẽ khởi tạo một bàn cờ mặc định dựa trên danh biến đã được quy định (phần 1.1)
  - Do quân trắng (quân AI) là quân được đi trước nên sẽ gọi tới hàm MinimaxVal trong lớp Minimax để tìm ra nước đi đầu tiên tối ưu. Sau đó sẽ chuyển lượt chơi cho quân đen
  - Sau khi quân đen (người dùng) thực hiện nước đi của mình thì sẽ chuyển lượt cho quân trắng (quân AI) và gọi hàm MinimaxVal
  - Mỗi khi đến lượt của quân trắng (quân AI) nếu nó còn có thể di chuyển thì sẽ gọi MinimaxVal để tìm nước đi.

3. MÃ NGUỒN BÀI TOÁN

Toàn bộ code của project được đẩy lên GitHub

- Link git: https://github.com/LeQuangTho/LearnAndroid/tree/main/DodGame

Click me