TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO BÀI TẬP LỚN NHẬP MÔN TRÍ TUỆ NHÂN TẠO

PHẦN MỀM TỰ GIẢI SUDOKU BẰNG TRÍ TUỆ NHÂN TẠO

Nhóm: 26

Mã lớp học: 147728

Giảng viên hướng dẫn: Trần Thế Hùng

Danh sách sinh viên thực hiện:

STT	Họ và tên	MSSV	email
1	Lương Nguyễn Quốc Tùng	20215161	tung.lnq215161@sis.hust.edu.vn
2	Lê Anh Đạt	20215025	dat.la215025@sis.hust.edu.vn
3	Nguyễn Mạnh Linh	20215079	linh.nm215079@sis.hust.edu.vn
4	Nguyễn Minh Chiến	20215000	chien.nm215000@sis.hust.edu.vn
5	Ngụy Khắc Phi Long	20210966	long.nkp210966@sis.hust.edu.vn
6	Hoàng Văn Quang	20215124	quang.hv215124@sis.hust.edu.vn

BẢNG CHẤM ĐIỂM

Họ và tên	Nội dung thực hiện	Điểm
Lương Nguyễn Quốc Tùng	-Leader, phân công nhiệm vụ cho từng thành viên. -Nghiên cứu bài toán, xây dựng phương pháp, thảo luận ưu nhược điểm giữa 2 thuật toán. -Xây dựng thuật toán Heuristic, code thuật toán.	
Lê Anh Đạt	-Nghiên cứu bài toán, thảo luận ưu nhược điểm giữa 2 thuật toán sử dụng. -Xây dựng thuật toán Backtracking, code thuật toán.	
Nguyễn Mạnh Linh	-Xây dựng giao diện (80%). -Thiết kế controller xử lí solve và hint.	
Nguỵ Khắc Phi Long	-Xây dựng giao diện (20%). -Thiết kế controller xử lí unsolve và reset. -Tìm hiểu mở rộng với phương pháp học tăng cường (RL)	
Hoàng Văn Quang	-Nghiên cứu bài toán, thực nghiệm, đưa ra kết quả so sánh ưu nhược điểm giữa hai thuật toán Backtracking và Heuristic trên lý thuyết và thực nghiệm. -Tìm hiểu mở rộng với phương pháp học tăng cường (RL)	

Nguyễn Minh Chiến	-Đánh giá và kiểm thử chương trình -làm báo cáo	

Mục Lục

Lời nói đầu

Trong thời đại công nghệ 4.0, trí tuệ nhân tạo (AI) đã trở thành một trong những lĩnh vực nghiên cứu và ứng dụng quan trọng, mang lại nhiều thành tựu vượt bậc trong nhiều lĩnh vực khác nhau. Với khả năng học hỏi và tự động hóa các tác vụ phức tạp, AI đang dần thay đổi cách chúng ta tiếp cận và giải quyết các vấn đề từ đơn giản đến phức tạp trong cuộc sống hàng ngày.

Sudoku, một trò chơi giải đố với các quy tắc đơn giản nhưng đòi hỏi tư duy logic và sự tập trung cao, đã trở thành một thách thức thú vị cho nhiều người yêu thích giải đố. Sự kết hợp giữa trò chơi này và AI đã mở ra một hướng nghiên cứu mới, không chỉ giúp tự động hóa quá trình giải đố mà còn đóng góp vào việc phát triển các thuật toán và công nghệ AI hiện đại.

Xuất phát từ sự đam mê với công nghệ và mong muốn khám phá khả năng ứng dụng của AI trong việc giải các bài toán phức tạp, chúng em đã quyết định chọn đề tài "Phần mềm tự giải Sudoku bằng trí tuệ nhân

tạo" cho báo cáo nghiên cứu của mình. Đề tài này không chỉ giúp chúng em hiểu rõ hơn về các thuật toán và công nghệ AI, mà còn mang lại cơ hội thực hành và áp dụng kiến thức vào thực tế.

Trong suốt quá trình thực hiện đề tài, chúng em đã nhận được sự hỗ trợ và hướng dẫn tận tình từ thầy Trần Thế Hùng, sự giúp đỡ từ các sinh viên cùng lớp. Chúng em xin chân thành cảm ơn sự giúp đỡ quý báu này.

Chúng em hy vọng rằng báo cáo này sẽ mang lại những thông tin hữu ích và gợi mở những hướng nghiên cứu mới cho những ai quan tâm đến lĩnh vực trí tuệ nhân tạo và các ứng dụng của nó trong đời sống.

Xin trân trọng cảm ơn!

Chương I: Giới thiệu về đề tài và trí tuệ nhân tạo

1.1. Giới thiệu về trí tuệ nhân tạo

1.1.1 Định Nghĩa AI

Trí tuệ nhân tạo (AI - Artificial Intelligence) là lĩnh vực nghiên cứu và phát triển các hệ thống máy tính có khả năng thực hiện các nhiệm vụ thường đòi hỏi trí thông minh của con người. Các hệ thống AI có khả năng học hỏi từ dữ liệu, nhận biết và phân tích môi trường, đưa ra quyết định và hành động để đạt được các mục tiêu cụ thể.

Trí tuệ nhân tạo khác với việc lập trình logic trong các ngôn ngữ lập trình là ở việc ứng dụng các hệ thống học máy (machine learning) để mô phỏng trí tuệ của con người trong các xử lý mà con người làm tốt hơn máy tính.

Cụ thể, trí tuệ nhân tạo giúp máy tính có được những trí tuệ của con người như: biết suy nghĩ và lập luận để giải quyết vấn đề, biết giao tiếp do hiểu ngôn ngữ, tiếng nói, biết học và tự thích nghi,…

Tuy rằng trí thông minh nhân tạo có nghĩa rộng như là trí thông minh trong các tác phẩm khoa học viễn tưởng, nó là một trong những ngành trọng yếu của tin học. Trí thông minh nhân tạo liên quan đến cách cư xử, sự học hỏi và khả năng thích ứng thông minh của máy móc

1.1.2. Lịch Sử Phát Triển AI

AI đã trải qua nhiều giai đoạn phát triển kể từ khi xuất hiện vào những năm 1950:

- **1956**: Hội nghị Dartmouth, nơi thuật ngữ "trí tuệ nhân tạo" được chính thức giới thiệu bởi John McCarthy.
- **1960s - 1970s**: Giai đoạn phát triển các hệ chuyên gia, hệ thống AI được thiết kế để mô phỏng quyết định của con người trong các lĩnh vực cụ thể.
- **1980s - 1990s**: Xuất hiện các hệ thống học máy (machine learning) và mạng nơ-ron nhân tạo (neural networks).
- **2000s - nay**: Sự phát triển mạnh mẽ của học sâu (deep learning), một phân nhánh của học máy, cùng với sự gia tăng của dữ liệu lớn (big data) và sức mạnh tính toán.

1.1.3. Các Ứng Dụng Phổ Biến của AI

AI hiện nay được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau:

- **Y tế**: Chẩn đoán bệnh, phân tích hình ảnh y khoa, quản lý hồ sơ bệnh án.
- **Giao thông**: Xe tự lái, quản lý lưu lượng giao thông, hệ thống định vị thông minh.
- **Tài chính**: Phân tích thị trường, dự đoán rủi ro, phát hiện gian lận.
- **Giải trí**: Khuyến nghị nội dung, tạo nhạc và hình ảnh, trò chơi điện tử.
- **Dịch vụ khách hàng**: Chatbot, hỗ trợ trực tuyến, phân tích phản hồi của khách hàng.

1.2. Giới thiệu về sudoku

1.2.1. Lịch Sử và Nguồn Gốc của Sudoku

Sudoku là một trò chơi giải đố số học phổ biến trên toàn thế giới. Mặc dù trò chơi này trở nên nổi tiếng vào cuối thế kỷ 20, nguồn gốc của nó có thể được truy vết từ những năm 1700 với một trò chơi số học của Thụy Sĩ tên là "Latin Squares" do nhà toán học Leonhard Euler phát minh. Tuy nhiên, phiên bản hiện đại của Sudoku được phát triển bởi Howard Garns, một kiến trúc sư người Mỹ, vào năm 1979 dưới tên gọi "Number Place" và sau đó được phổ biến rộng rãi ở Nhật Bản vào những năm 1980 bởi công ty Nikoli, người đặt tên cho nó là "Sudoku".

2. Quy Tắc và Cách Chơi Sudoku

Sudoku là một bảng lưới 9x9, được chia thành 9 vùng nhỏ 3x3. Mục tiêu của trò chơi là điền các số từ 1 đến 9 vào các ô sao cho:

- Mỗi hàng ngang (row) có đủ các số từ 1 đến 9, không trùng lặp.
- Mỗi cột dọc (column) có đủ các số từ 1 đến 9, không trùng lặp.
- Mỗi vùng 3x3 có đủ các số từ 1 đến 9, không trùng lặp.

Bảng Sudoku bắt đầu với một số ô đã được điền số, và nhiệm vụ của người chơi là điền các số còn lại sao cho tuân thủ các quy tắc trên.

8			4		6			7
						4		
	1					6	5	
5		9		3		7	8	
				7				
	4	8		2		1		3
	5	2					9	
		1						
3			9		2			5

8	3	5	4	1	6	9	2	7
2	9	6	8	5	7	4	3	1
4	1	7	2	9	3	6	5	8
5	6	9	1	3	4	7	8	2
1	2	3	6	7	8	5	4	9
7	4	8	5	2	9	1	6	3
6	5	2	7	8	1	3	9	4
9	8	1	3	4	5	2	7	6
3	7	4	9	6	2	8	1	5

Chương II: Cấu trúc dữ liệu và giải thuật cho bài toán

2.1. Thuật toán đệ quy quay lui (Backtracking)

2.1.1. Thuật toán đệ quy quay lui (backtracking) là gì?

Quay lui hay Backtracking là một kĩ thuật thiết kế giải thuật dựa trên đệ quy. Ý tưởng của quay lui là tìm lời giải từng bước, mỗi bước chọn một trong số các lựa chọn khả dĩ và đệ quy. Người đầu tiên đề ra thuật ngữ này (backtrack) là nhà toán học người Mỹ D. H. Lehmer vào những năm 1950.

Quay lui dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng.

2.1.2. Tư tưởng

Dùng để giải bài toán liệt kê các cấu hình. Mỗi cấu hình được xây dựng bằng từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng.

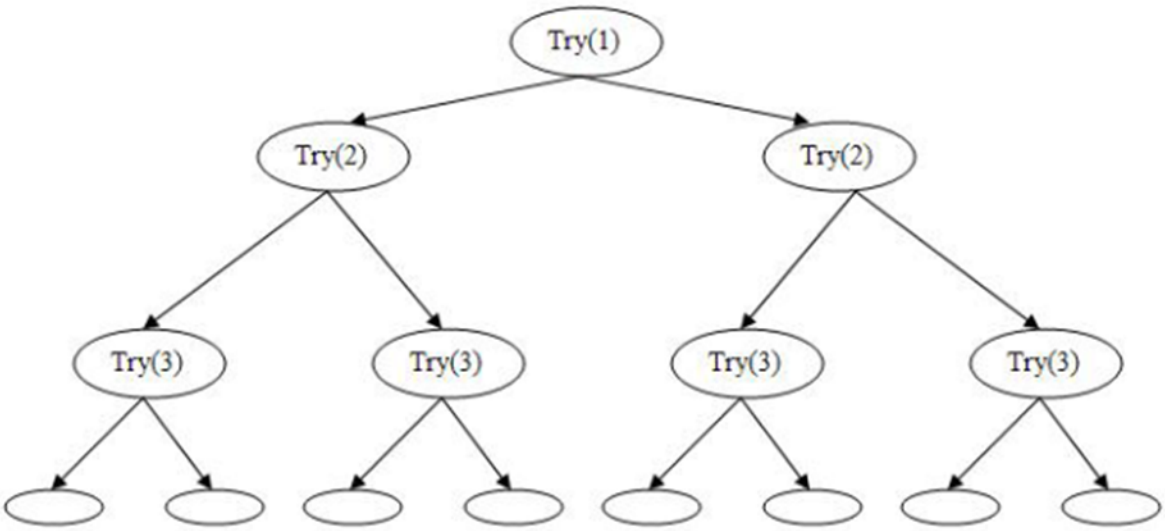
Các bước trong việc liệt kê cấu hình dạng X[1...n]:

Xét tất cả các giá trị X[1] có thể nhận, thử X[1] nhận các giá trị đó. Với mỗi giá trị của X[1] ta sẽ:

Xét tất cả giá trị X[2] có thể nhận, lại thử X[2] cho các giá trị đó. Với mỗi giá trị X[2] lại xét khả năng giá trị của X[3]...tiếp tục như vậy cho tới bước:

...
...

Xét tất cả giá trị X[n] có thể nhận, thử cho X[n] nhận lần lượt giá trị đó. Thông báo cấu hình tìm được.
Bản chất của quay lui là một quá trình tìm kiếm theo chiều sâu(Depth-First Search).



2.1.3. Áp dụng vào giải Sudoku

Trong trường hợp giải Sudoku, thuật toán này hoạt động như sau:

1. **Xác định ô trống đầu tiên trong lưới sudoku**
    - Duyệt qua từng ô trong lưới sudoku để tìm ra ô trống đầu tiên.
2. **Điền giá trị hợp lệ vào ô trống đó**

- Với mỗi ô trống, thử từng giá trị từ 1 đến 9.
- Kiểm tra xem giá trị đó có thỏa mãn các quy tắc của Sudoku không, tức là không có số trùng lặp trong hàng, cột và vùng 3x3 chứa ô đó.

- Nếu giá trị hợp lệ, điền giá trị vào ô đó và tiếp tục đệ qui vào ô trống tiếp theo.

3. Kiểm tra tính hợp lệ và quay lui

- Nếu không có giá trị nào thỏa mãn, quay lại ô trống trước đó và thử giá trị khác.
- Lặp lại quá trình này cho đến khi tìm ra một giải pháp hoặc không có giá trị nào thỏa mãn.

4. Lặp lại cho tất cả các ô trống

- Lặp lại các bước trên cho tất cả các ô trống trong lưới Sudoku cho đến khi tìm ra một giải pháp hoặc không có giải pháp nào tồn tại.

5. Kết thúc

- Thuật toán sẽ kết thúc khi tất cả các ô trong lưới đã được điền và thỏa mãn các quy tắc của Sudoku, hoặc nếu không có giải pháp nào tồn tại.

2.2. Thuật toán Heuristic

2.1.1. Thuật toán Heuristic là gì?

Heuristic là những tri thức rút ra từ kinh nghiệm và suy đoán của con người, nó có thể là những tri thức đúng hoặc sai, là những meta knowledge và thường là đúng.

Giải thuật Heuristic là một mở rộng của khái niệm thuật toán. Nó thể hiện cách giải bài toán với những đặc trưng sau:

-Thường tìm được lời giải tốt (nhưng không chắc chắn là tốt nhất).

-Giải bài toán theo giải thuật Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì thế mà chi phí thấp hơn.

-Giải thuật Heuristic thường khá tự nhiên, gần gũi với cách suy nghĩ và hành động của con người.

2.2.2. Áp dụng thuật toán Heuristic để giải Sudoku

Bằng cách tiếp cận heuristic để giải Sudoku bằng cách kết hợp ưu tiên (Priority Queue) và các quy tắc logic để giảm không gian tìm kiếm và tăng tốc độ giải quyết. Dưới đây là giải thích chi tiết về nguyên lý hoạt động của thuật toán này:

1. **Xác định các giá trị khả thi cho từng ô trống**: Các giá trị khả thi được tính toán và lưu trữ.
2. **Sắp xếp các ô trống theo độ khó**: Các ô có ít giá trị khả thi nhất được ưu tiên giải quyết trước.
3. **Thử và sai với các giá trị khả thi**: Sử dụng đệ quy và quay lui để thử các giá trị khả thi cho từng ô, tìm kiếm giải pháp cho toàn bộ bảng Sudoku.
4. **Quay lui khi gặp ngõ cụt**: Nếu không tìm được giá trị hợp lệ, thuật toán sẽ quay lui và thử các giá trị khác.

Bằng cách này, thuật toán heuristic giúp tăng tốc quá trình giải Sudoku bằng cách giảm không gian tìm kiếm và đưa ra các lựa chọn thông minh dựa trên các quy tắc logic đơn giản.

Chương III: Cài đặt và đánh giá thử nghiệm

3.1. Chương trình giải Sudoku bằng giải thuật đệ quy quay lui (backtracking)

```java
package ai;

import java.util.Queue;
import java.util.LinkedList;
import java.util.Stack;

public class BackTrackSolver implements SudokuSolver {
    protected int[][] sudoku;
    protected Queue<Integer[]> route;
    protected Stack<Integer[]> hints;

    public BackTrackSolver(int[][] sudoku)
    {
        this.sudoku = new int[9][9];
        for (int row = 0; row < 9; row++)
            for (int col = 0; col < 9; col++)
                this.sudoku[row][col] = sudoku[row][col];
        route = (Queue<Integer[]>) (new LinkedList<Integer[]>());
        hints = new Stack<Integer[]>();
    }



    public Queue<Integer[]> getRoute()
    {
        return route;
    }



    public Stack<Integer[]> getHints()
    {
        return hints;
    }
```

```
    public int[][] getSolution()
    {
        return sudoku;
    }



    protected boolean isValidPuzzle()
    {
        for (int row = 0; row < 9; row++)
            for (int col = 0; col < 9; col++)
            {
                if (sudoku[row][col] == 0)
                    continue;
                for (int i = 0; i < 9; i++)
                {
                    if (i != col && sudoku[row][i] == sudoku[row][col]) return
false;

                    if (i != row && sudoku[i][col] == sudoku[row][col]) return
false;

                    if (3 * (row/3) + i/3 != row && 3 * (col/3) + i%3 != col &&
sudoku[3 * (row/3) + i/3][3 * (col/3) + i%3] == sudoku[row][col]) return false;
            }
                    }
        return true;
    }

    protected boolean isValid(int value, int row, int col, int[][] sudoku_t)
    {
        for (int i = 0; i < 9; i++)
        {
            if (sudoku_t[row][i] == value) return false;

            if (sudoku_t[i][col] == value) return false;

            if (sudoku_t[3 * (row/3) + i/3][3 * (col/3) + i%3] == value) return
false;
        }
```

```
        return true;
    }



    public boolean solve()
    {
        if (!isValidPuzzle())
            return false;

        return solve(sudoku);
    }

    protected boolean solve(int[][] sudoku_t)
    {
        for (int row = 0; row < 9; row++)
            for (int col = 0; col < 9; col++)
            {
                if (sudoku_t[row][col] != 0)
                    continue;
                for (int num = 1; num <= 9; num++)
                {
                    if (isValid(num, row, col, sudoku_t))
                    {
                        sudoku_t[row][col] = num;
                        Integer[] cell = {row, col, num};
                        route.offer(cell);
                        if (solve(sudoku_t))
                        {
                            hints.push(cell);
                            return true;
                        }

                        sudoku_t[row][col] = 0;
                        Integer[] cell2 = {row, col, 0};
                        route.offer(cell2);
                    }
                }

                return false;
            }

        return true;
```

```
    }
}
```

3.1.1. Nguyên lý hoạt động của thuật toán đệ quy quay lui (backtracking) trong đoạn code

1. Khởi Tạo

- Khởi Tạo BackTrackSolver:
  - Sao chép bảng Sudoku vào mảng sudoku.
  - Khởi tạo hàng đợi route để lưu trữ các bước đã thực hiện.
  - Khởi tạo ngăn xếp hints để lưu trữ các gợi ý (các giá trị đúng đã điền vào).

2. Hàm isValid

- Kiểm Tra Giá Trị Hợp Lệ:
  - Hàm này kiểm tra xem giá trị value có thể được đặt vào ô (row, col) hay không.
  - Kiểm tra hàng, cột và ô vuông 3x3 để đảm bảo không có giá trị trùng lặp.

3. Hàm solve (Overloaded)

- Giải Quyết Sudoku:
  - Hàm này thực hiện đệ quy để giải quyết bảng Sudoku.
  - Duyệt qua từng ô trong bảng.
  - Nếu ô hiện tại không phải là ô trống (giá trị khác 0), tiếp tục với ô tiếp theo.
  - Nếu ô hiện tại là ô trống (giá trị là 0), thử điền các giá trị từ 1 đến 9.
  - Kiểm tra tính hợp lệ của mỗi giá trị với hàm isValid.

- Nếu giá trị hợp lệ, điền giá trị vào ô và gọi đệ quy solve với bảng Sudoku mới.
- Nếu không tìm được giải pháp, quay lui bằng cách xóa giá trị đã điền (gán lại là 0) và thử giá trị khác.

4. Hàm isValidPuzzle

- Kiểm Tra Tính Hợp Lệ của Bảng Sudoku Ban Đầu:
  - Hàm này kiểm tra xem bảng Sudoku ban đầu có hợp lệ hay không.
  - Kiểm tra từng ô để đảm bảo không có giá trị trùng lặp trong hàng, cột và ô vuông 3x3.

5. Hàm solve (Main)

- Bắt Đầu Giải Quyết Sudoku:
  - Kiểm tra tính hợp lệ của bảng Sudoku ban đầu.
  - Gọi hàm solve đệ quy để bắt đầu giải quyết Sudoku.

3.1.2. Chương trình giải Sudoku bằng thuật toán Heuristic

```
package ai;

import java.util.PriorityQueue;

public class HeuristicSolver extends BackTrackSolver {
    Value[][] values;
    private PriorityQueue<Value> blankCells;

    public HeuristicSolver(int[][] sudoku)
    {
        super(sudoku);
        values = new Value[9][9];
        blankCells = new PriorityQueue<Value>();
```

```
    }

    public boolean solve(int[][] sudoku_t)
    {
        updateValues();
        if (!blankCells.isEmpty())
        {
            Value vals = blankCells.poll();
            int row = vals.getRow(), col = vals.getCol();

            for (Integer num : vals.getValues())
            {
                if (isValid(num, row, col, sudoku_t))
                {
                    sudoku_t[row][col] = num;
                    Integer[] cell = {row, col, num};
                    route.offer(cell);
                    if (solve(sudoku_t))
                    {
                        hints.push(cell);
                        return true;
                    }

                    sudoku_t[row][col] = 0;
                    Integer[] cell2 = {row, col, 0};
                    route.offer(cell2);
                }
            }

            return false;
        }

        return true;
    }

    private void updateValues()
    {
        blankCells.clear();
        for (int row = 0; row < 9; row++)
            for (int col = 0; col < 9; col++)
            {
                if (sudoku[row][col] == 0)
```

```
                {
                    Value val = possibleValues(row, col);
                    blankCells.offer(val);
                    values[row][col] = val;
                }
            }
    }



    private Value possibleValues(int row, int col)
    {
        Value result = new Value(row, col);
        if (sudoku[row][col] != 0)
            return result;
        for (int i = 1; i <= 9; i++)
            if (isValid(i, row, col, sudoku))
                result.add(i);

        return result;
    }
}
```

3.1.1. Nguyên lý hoạt động của thuật toán Heuristic trong đoạn code

1. Khởi Tạo

- Khởi Tạo HeuristicSolver:
    - values là một mảng hai chiều lưu trữ các giá trị khả thi cho từng ô trống.
    - blankCells là một Priority Queue lưu trữ các ô trống theo thứ tự ưu tiên. Các ô này được sắp xếp dựa trên số lượng giá trị khả thi (số lượng càng ít thì ưu tiên càng cao).

2. Hàm solve

- Giải Quyết Sudoku:

- Hàm solve thực hiện đệ quy để giải quyết bảng Sudoku.
- Đầu tiên, nó cập nhật các giá trị khả thi cho từng ô trống bằng cách gọi updateValues.
- Sau đó, nếu còn ô trống, nó lấy ô có ít giá trị khả thi nhất từ Priority Queue blankCells.
- Nó thử từng giá trị khả thi cho ô đó. Nếu giá trị này hợp lệ (isValid), nó điền giá trị vào ô và tiếp tục giải quyết ô tiếp theo bằng cách gọi đệ quy solve.
- Nếu không tìm được giải pháp, nó quay lui bằng cách xóa giá trị đã điền (gán lại là 0) và thử giá trị khác.

3. Hàm updateValues

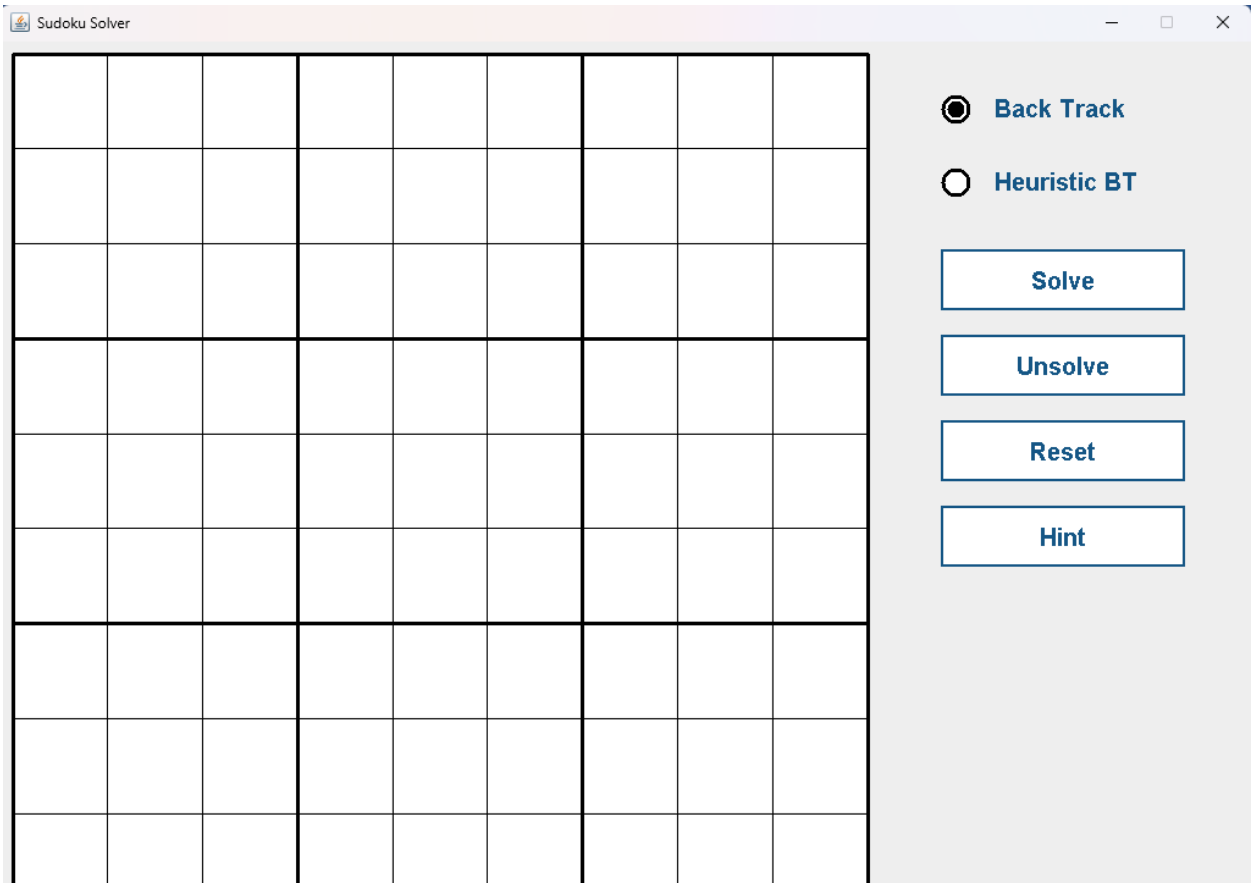
- Cập Nhật Các Giá Trị Khả Thi:
  - Hàm này duyệt qua toàn bộ bảng Sudoku và tìm các ô trống.
  - Với mỗi ô trống, nó gọi hàm possibleValues để tìm các giá trị khả thi cho ô đó.

4. Hàm possibleValues

- Tìm Các Giá Trị Khả Thi Cho Ô:
  - Hàm này kiểm tra từng giá trị từ 1 đến 9 xem có hợp lệ tại vị trí (row, col) hay không.
  - Nếu giá trị hợp lệ, nó thêm giá trị đó vào đối tượng Value.
  - Đối tượng Value chứa thông tin về vị trí ô và các giá trị khả thi.

  - Các ô trống và giá trị khả thi được thêm vào Priority Queue blankCells.

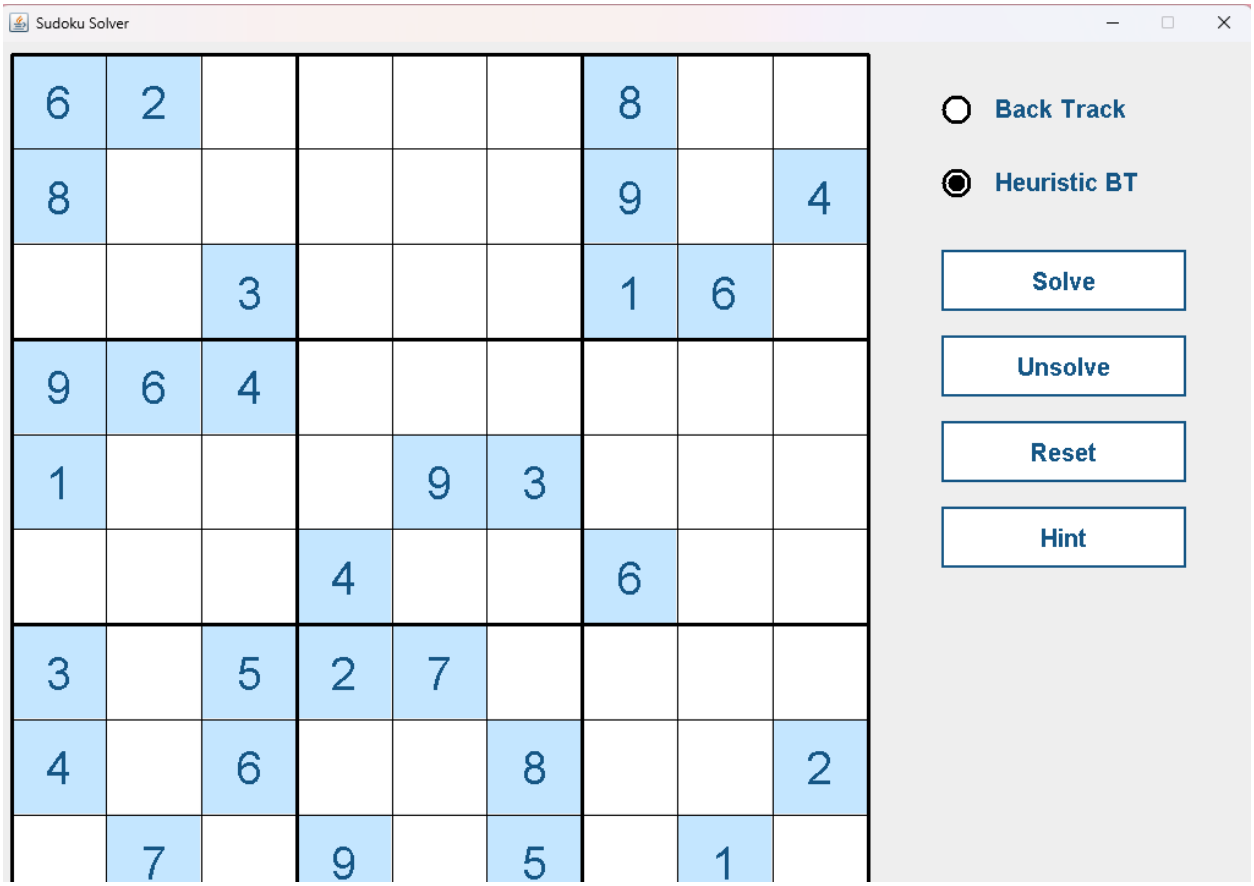
3.3. Giao diện phần mềm



3.3.1. Các chức năng chính

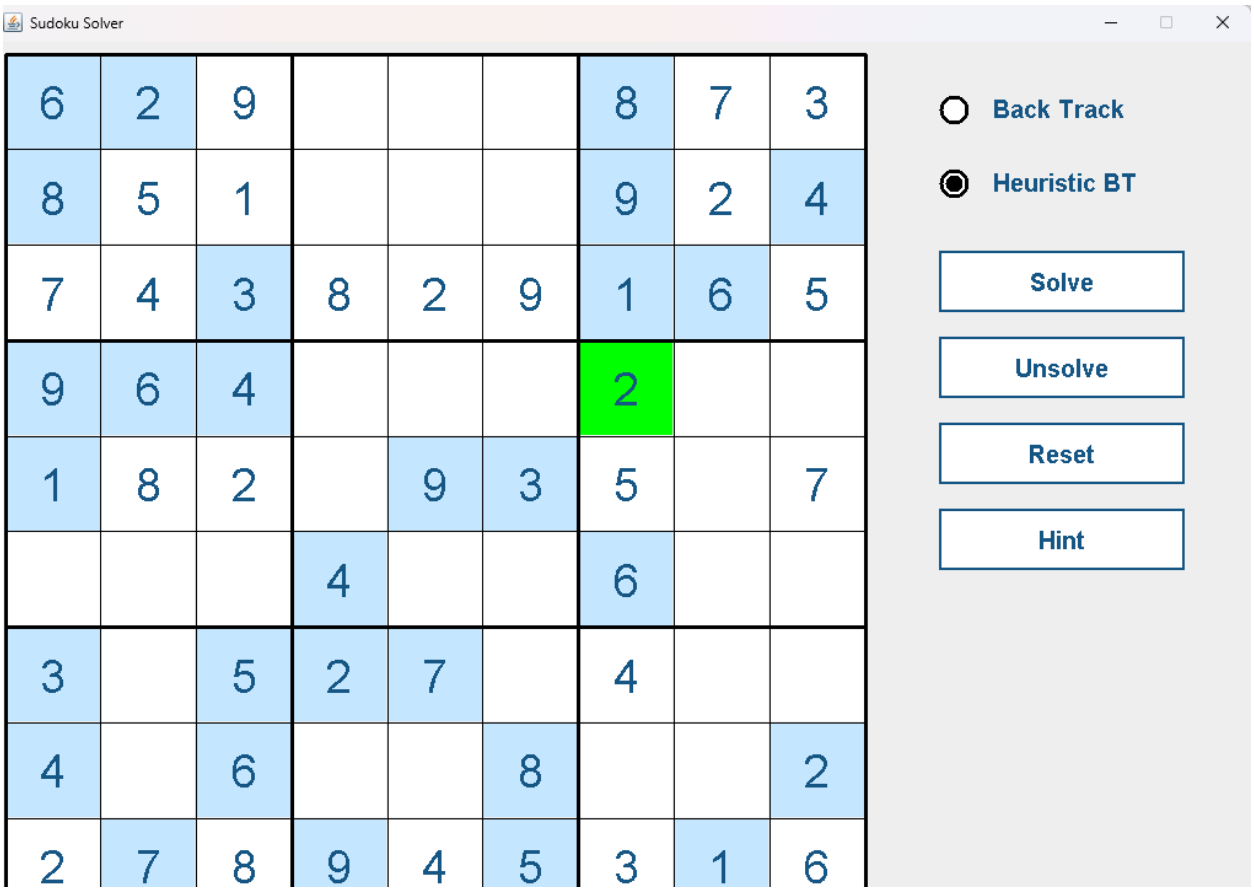
- **Back Track** và **Heuristic BT**: Lựa chọn thuật toán để giải bảng sudoku
- **Solve**: Sử dụng các thuật toán hoạt động để giải câu đố
- **Unsolve**: Xóa các ô đã được giải
- **Reset**: Xóa toàn bộ chữ số trong bảng
- **Hint**: hiện đáp án từng ô một, thứ tự dựa trên lộ trình của thuật toán đã chọn

3.3.2. Chọn đề bài

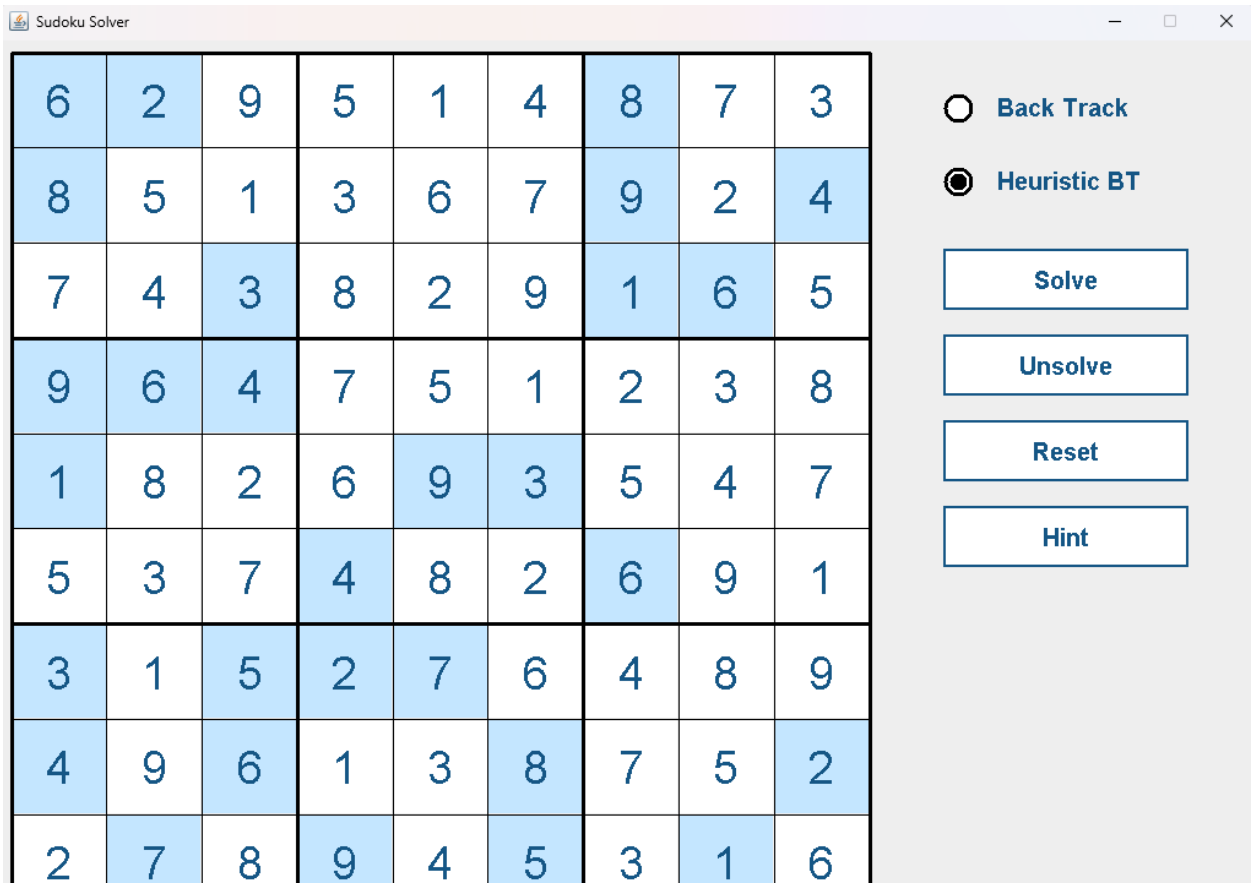


- Điền chữ số vào từng ô một để tạo đề bài
- Có thể tạo cùng một chữ số cho nhiều ô cùng một lúc

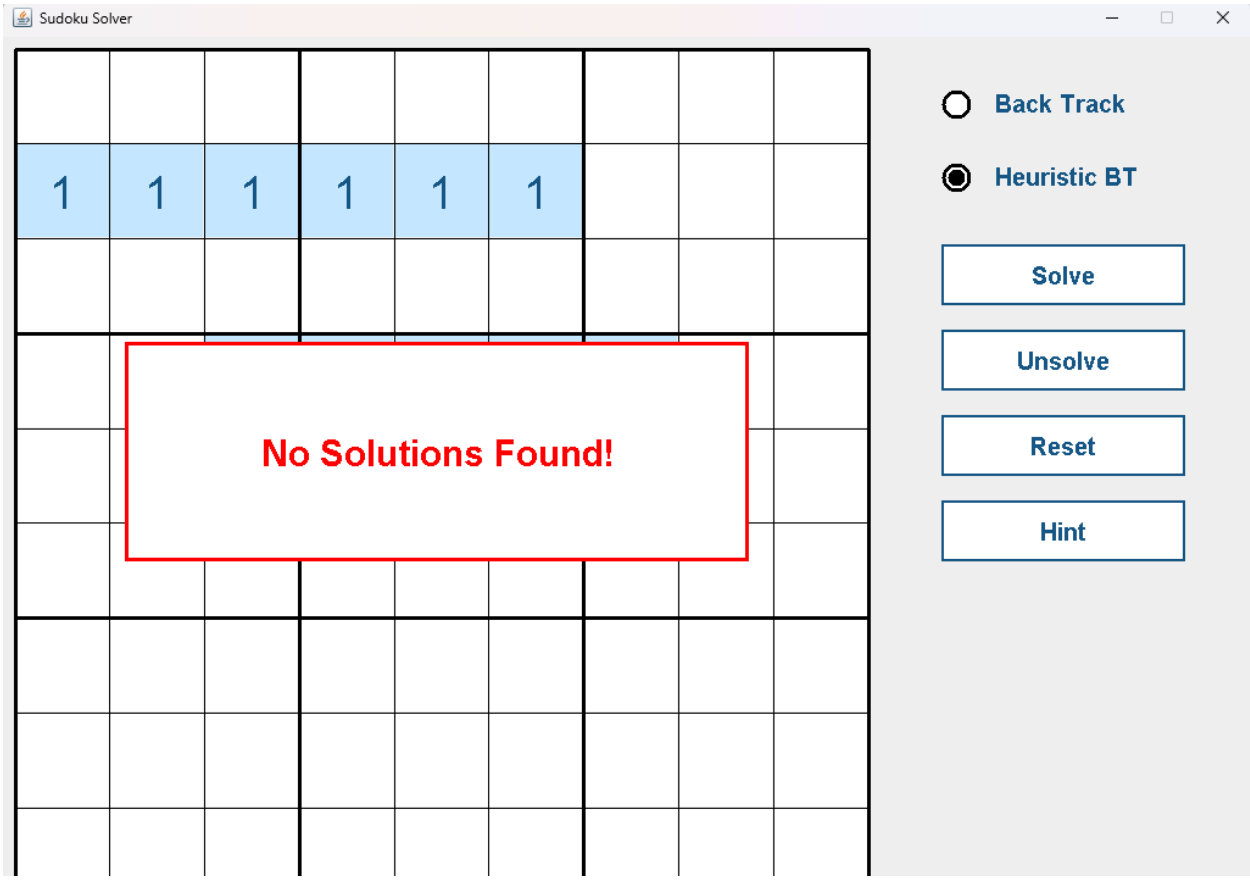
3.3.3. Giải bài toán



Quá trình giải bài toán



Sau khi đã giải xong



Dòng chữ "No Solutions Found!" hiện ra nếu bảng sudoku không hợp lệ

3.4. Phân tích kết quả

3.4.1.Phân Tích

Hiệu Quả của Thuật Toán Đệ Quy Quay Lui

2. Ưu Điểm:
    - Đơn Giản và Hiểu Quả: Thuật toán đệ quy quay lui dễ hiểu và dễ triển khai. Nó tìm kiếm tất cả các giải pháp khả thi bằng cách thử và sai.
    - Đảm Bảo Tìm Được Giải Pháp: Nếu có ít nhất một giải pháp cho bảng Sudoku, thuật toán này sẽ tìm ra nó.
3. Nhược Điểm:

- Hiệu Suất Thấp: Trong các trường hợp bảng Sudoku phức tạp, thuật toán này có thể mất nhiều thời gian do phải thử tất cả các khả năng có thể.
- Khả Năng Quay Lui Cao: Khi gặp các trường hợp cần quay lui nhiều, thuật toán sẽ phải lặp lại nhiều bước dẫn đến hiệu suất giảm.

3.4.2. Hiệu Quả của Thuật Toán Heuristic

4. Ưu Điểm:
   - Tăng Tốc Độ Giải Quyết: Thuật toán heuristic cải thiện tốc độ giải quyết bằng cách ưu tiên các ô trống có ít giá trị khả thi nhất.
   - Giảm Số Lần Quay Lui: Bằng cách giải quyết các ô khó trước, thuật toán giảm số lần thử sai và quay lui.
5. Nhược Điểm:
   - Độ Phức Tạp Cao Hơn: Việc triển khai thuật toán heuristic phức tạp hơn so với thuật toán đệ quy quay lui cơ bản.
   - Không Đảm Bảo Tìm Giải Pháp Nhanh Nhất: Dù tăng tốc độ giải quyết, không phải lúc nào thuật toán heuristic cũng tìm được giải pháp nhanh nhất do phụ thuộc vào chiến lược ưu tiên.

3.4.3. Kết Quả Thực Nghiệm

Để so sánh hiệu quả của hai thuật toán, chúng ta tiến hành các bài kiểm tra trên một tập hợp các bảng Sudoku có độ khó khác nhau, từ dễ đến khó. Kết quả thực nghiệm cho thấy:

6. Đối với các bảng Sudoku dễ:
   - Thuật toán đệ quy quay lui và heuristic đều giải quyết một cách nhanh chóng.
   - Thời gian giải quyết không có sự khác biệt đáng kể.

7. Đối với các bảng Sudoku khó:
    - Thuật toán heuristic giải quyết nhanh hơn so với đệ quy quay lui.
    - Số lần quay lui của thuật toán heuristic ít hơn so với thuật toán đệ quy quay lui.

3.4.4. Kết Luận

8. Độ Hiệu Quả Tổng Thể:
    - Thuật toán heuristic cho thấy hiệu quả vượt trội trong việc giải quyết các bảng Sudoku phức tạp so với thuật toán đệ quy quay lui.
    - Đối với các bảng Sudoku dễ, cả hai thuật toán đều hoạt động tốt và thời gian giải quyết không có sự khác biệt lớn.
9. Lựa Chọn Thuật Toán:
    - Nếu ưu tiên sự đơn giản và đảm bảo tìm được giải pháp, thuật toán đệ quy quay lui là một lựa chọn tốt.
    - Nếu cần tăng tốc độ giải quyết và giảm số lần quay lui cho các bài toán phức tạp, thuật toán heuristic là lựa chọn tối ưu hơn.

Đề Xuất Phát Triển

10. Cải Tiến Heuristic:
    - Kết hợp thêm các phương pháp tiên tiến như học máy để cải thiện độ chính xác của việc ước tính giá trị khả thi cho từng ô.
11. Kết Hợp Cả Hai Thuật Toán:

- Sử dụng thuật toán heuristic để giảm không gian tìm kiếm ban đầu và sau đó áp dụng đệ quy quay lui để tìm giải pháp cuối cùng.

12. Tối Ưu Hóa Mã Nguồn:
    - Cải tiến mã nguồn để tận dụng tối đa tài nguyên hệ thống, như song song hóa quá trình giải quyết hoặc tối ưu bộ nhớ.

Chương IV: MỞ RỘNG VỚI THUẬT TOÁN Q-LEARNING TRONG HỌC TĂNG CƯỜNG (RL) ĐỂ GIẢI SUDOKU

1. Giới thiệu về Q-Learning

Q-Learning là một thuật toán học tăng cường (Reinforcement Learning) không có mô hình (model-free) được sử dụng để tìm chính sách tối ưu nhằm tối đa hóa tổng phần thưởng (reward) nhận được qua thời gian. Để giải Sudoku, Q-Learning có thể được sử dụng để dạy một agent (tác tử) cách điền số vào các ô trống theo cách tối ưu nhất.

2. Cơ bản về Q-learning

Q-Value: Q-Learning sử dụng một bảng Q-Table để lưu trữ các giá trị Q cho các cặp trạng thái-hành động. Giá trị Q là dự đoán của agent về phần thưởng tương lai khi thực hiện một hành động nhất định trong một trạng thái nhất định.

Cập Nhật Q-Value: Q-Value được cập nhật bằng cách sử dụng phương trình Bellman:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Trong đó

- s là trạng thái hiện tại.
- a là hành động được thực hiện.
- r là phần thưởng nhận được sau khi thực hiện hành động aaa.
- s′ là trạng thái mới sau khi thực hiện hành động aaa.
- α là tốc độ học (learning rate).
- γ là hệ số giảm giá (discount factor), xác định tầm quan trọng của phần thưởng tương lai.

3. Áp Dụng Q-Learning vào Giải Sudoku

3.1. Mục tiêu (Goal):

- **Điền Đầy Đủ Các Ô Trống:** Điền tất cả các ô trống trong bảng Sudoku với các số từ 1 đến 9.
- **Tuân Thủ Các Quy Tắc Sudoku**: Đảm bảo rằng mỗi số từ 1 đến 9 chỉ xuất hiện duy nhất một lần trong mỗi hàng, mỗi cột và mỗi vùng 3x3.

3.2. Môi trường (Environment):

Bảng Sudoku: Một ma trận 9x9 đại diện cho trạng thái hiện tại của trò chơi.

3.3. Trạng Thái (State):

Trạng thái là trạng thái hiện tại của bảng Sudoku, đại diện bởi một ma trận 9x9 với các ô có giá trị từ 0 đến 9 (0 đại diện cho ô trống).

3.4. Hành Động (Action):

Hành động là việc điền một số từ 1 đến 9 vào một ô trống. Có thể đại diện hành động dưới dạng (row, col, num).

3.5. Phần Thưởng (Reward):

Phần thưởng có thể được thiết kế dựa trên việc hành động có hợp lệ hay không. Ví dụ:

- Nếu hành động hợp lệ, phần thưởng là +1.

- Nếu hành động không hợp lệ, phần thưởng là -1.
- Phần thưởng lớn hơn có thể được trao khi giải xong một bảng Sudoku hoàn chỉnh.

3.6. Trạng thái dừng (Terminal State):

Bảng Sudoku được giải xong: Đây là trạng thái mà tất cả các ô trong bảng Sudoku đều được điền đầy đủ và thỏa mãn các điều kiện của Sudoku (mỗi số từ 1 đến 9 xuất hiện duy nhất một lần trên mỗi hàng, cột, và vùng 3x3).

4.Phân tích thuật toán

4.1. Ưu Điểm

1. **Học Tập Thích Ứng**:

- Q-Learning cho phép tác tử học cách điền số vào bảng Sudoku dựa trên kinh nghiệm từ các tập huấn luyện.
- Thích ứng với các tình huống khác nhau và cải thiện hiệu suất qua thời gian.

2. **Tối Ưu Hóa Chính Sách**:

- Q-Learning tìm kiếm chính sách tối ưu để điền số vào các ô, đảm bảo tác tử luôn chọn hành động tốt nhất dựa trên giá trị Q.

4.2. Nhược Điểm

1. **Thời Gian Huấn Luyện**:

- Q-Learning cần thời gian huấn luyện dài để đạt được hiệu suất tốt, đặc biệt là với các bảng Sudoku phức tạp.
- Cần nhiều tập dữ liệu huấn luyện để tác tử học cách giải quyết bài toán hiệu quả.

2. **Không Bảo Đảm Giải Quyết Nhanh Chóng**:

Trong giai đoạn ban đầu, tác tử có thể không giải quyết bài toán một cách nhanh chóng do thiếu kinh nghiệm và cần nhiều lần thử nghiệm và học hỏi.

5. Kết Luận

Sử dụng Q-Learning để giải Sudoku là một phương pháp thú vị và tiềm năng, cho phép tác tử học từ kinh nghiệm và cải thiện qua thời gian. Mặc dù có thể không hiệu quả ngay từ đầu, việc áp dụng học tăng cường vào giải quyết Sudoku mở ra nhiều cơ hội để phát triển các phương pháp giải quyết bài toán thông minh và hiệu quả hơn.

Kết Luận

Tổng Kết về Phần Mềm Tự Giải Sudoku Bằng Trí Tuệ Nhân Tạo

Qua quá trình nghiên cứu và phát triển, chúng em đã triển khai hai thuật toán chính để giải quyết bài toán Sudoku: thuật toán đệ quy quay lui và thuật toán heuristic. Mỗi thuật toán đều có những ưu điểm và nhược điểm riêng, phù hợp với các loại bài toán và mục đích sử dụng khác nhau.

Đóng Góp của Thuật Toán Đệ Quy Quay Lui

13. **Đơn Giản và Hiệu Quả**:
    - Thuật toán đệ quy quay lui có cấu trúc đơn giản, dễ hiểu và dễ triển khai. Đây là một lựa chọn tốt cho những người mới bắt đầu hoặc cho các ứng dụng đơn giản.
    - Thuật toán đảm bảo tìm được giải pháp nếu có, và có thể áp dụng cho mọi bảng Sudoku.
14. **Hạn Chế về Hiệu Suất**:
    - Thuật toán này có thể mất nhiều thời gian và tài nguyên khi xử lý các bảng Sudoku phức tạp do số lần thử sai và quay lui cao.

Đóng Góp của Thuật Toán Heuristic

15. **Tăng Tốc Độ Giải Quyết**:
    - Thuật toán heuristic cải thiện đáng kể tốc độ giải quyết bằng cách sử dụng ưu tiên (Priority Queue) để chọn các ô trống có ít giá trị khả thi nhất. Điều này giúp giảm không gian tìm kiếm và tăng hiệu suất.

16. **Giảm Số Lần Quay Lui**:
    - Bằng cách giải quyết các ô khó trước, thuật toán heuristic giảm số lần thử sai và quay lui, từ đó cải thiện hiệu suất tổng thể, đặc biệt là với các bảng Sudoku có độ khó cao.
17. **Độ Phức Tạp Cao Hơn**:
    - Mặc dù hiệu quả hơn, việc triển khai thuật toán heuristic phức tạp hơn và yêu cầu nhiều công đoạn xử lý hơn so với thuật toán đệ quy quay lui cơ bản.

Kết Quả Thực Nghiệm

Thực nghiệm trên các bảng Sudoku với độ khó khác nhau cho thấy:

- Với các bảng Sudoku dễ, cả hai thuật toán đều hoạt động tốt và thời gian giải quyết không có sự khác biệt lớn.
- Với các bảng Sudoku khó và rất khó, thuật toán heuristic vượt trội hơn về tốc độ và số lần quay lui.

Đề Xuất và Phát Triển Tương Lai

18. **Cải Tiến Thuật Toán Heuristic**:
    - Có thể cải tiến thuật toán heuristic bằng cách kết hợp với các kỹ thuật học máy để ước tính giá trị khả thi cho từng ô chính xác hơn, từ đó tăng hiệu quả giải quyết.
19. **Kết Hợp Cả Hai Thuật Toán**:
    - Một hướng phát triển khác là kết hợp cả hai thuật toán: sử dụng heuristic để giảm không gian tìm kiếm ban đầu và sau đó áp dụng đệ quy quay lui để tìm giải pháp cuối cùng.
20. **Tối Ưu Hóa Mã Nguồn và Hiệu Năng**:
    - Tối ưu hóa mã nguồn để tận dụng tối đa tài nguyên hệ thống, như song song hóa quá trình giải quyết hoặc tối ưu bộ nhớ, để tăng hiệu quả và tốc độ giải quyết.

Kết Luận Cuối

Phần mềm tự giải Sudoku bằng trí tuệ nhân tạo là một ứng dụng thú vị và hữu ích, thể hiện khả năng của các thuật toán giải quyết bài toán trong lĩnh vực trí tuệ nhân tạo. Qua quá trình nghiên cứu và phát triển, chúng tôi đã triển khai hai thuật toán hiệu quả để giải quyết bài toán Sudoku, mang lại cái nhìn sâu sắc hơn về ứng dụng của các phương pháp này trong thực tế. Việc tiếp tục cải tiến và phát triển các thuật toán này sẽ mở ra nhiều cơ hội và ứng dụng mới trong lĩnh vực trí tuệ nhân tạo và giải quyết bài toán.

Với phần mềm này, người dùng có thể giải quyết nhanh chóng các bảng Sudoku từ dễ đến khó, học hỏi cách thức hoạt động của các thuật toán đệ quy và heuristic, cũng như ứng dụng chúng vào các bài toán thực tế khác.