TRƯỜNG ĐẠI HỌC VĂN LANG

KHOA CÔNG NGHỆ THÔNG TIN

TRƯỜNG ĐẠI HỌC VĂN LANG

**KHOA CÔNG NGHỆ THÔNG TIN**

Ảnh có chứa văn bản, hình mẫu

Mô tả được tạo tự động

**BÁO CÁO ĐỒ ÁN MÔN HỌC**

CHUYÊN NGÀNH: TRÍ TUỆ NHÂN TẠO

***Chủ đề:***

**TÌM HIỂU VÀ CÀI ĐẶT ỨNG DỤNG**

**GIẢI THUẬT BACKTRACKING**

**VÀO TRÒ CHƠI SUDOKU**

**NHÓM THỰC HIỆN: NONAME**

**GVHD: THẦY PHAN HỒ VIẾT TƯỜNG**

TP. Hồ Chí Minh – năm 2024

TP. Hồ Chí Minh – năm 2023

TP. Hồ Chí Minh – năm 2023

TP. Hồ Chí Minh – năm 2023

TP. Hồ Chí Minh, ngày… tháng … năm 2023

Sinh viên thực hiện

(Ký và ghi rõ họ tên)

Nguyễn Ngọc Thảo Vy

CÁC THÀNH VIÊN TRONG NHÓM

|  |  |  |
| --- | --- | --- |
| **HỌ VÀ TÊN** | **MSSV** | **CÔNG VIỆC** |
| TẠ GIA BẢO | 2274802010050 | A, C, F |
| PHAN ANH THƯ | 2274802010872 | A,B, D, E |
| VŨ ĐỨC HUY | 2174802010672 | A, C, F |
| TRẦN MINH HUY | 2274802010332 | A, B |

A: Tìm hiểu đề tài và khai thác

B: Triển khai code

C: Triển khai word

D: Triển Khai PPT

E: Điều chỉnh word và PPT

F: Kiểm tra code

Lời cảm ơn

Trước hết, chúng em xin gửi lời cảm ơn chân thành đến thầy **Phan Hồ Viết Tường**, người đã trực tiếp giảng dạy và hướng dẫn em trong môn *Trí tuệ nhân tạo và ứng dụng*. Nhờ sự tận tụy và nhiệt huyết của thầy, chúng đã có cơ hội khám phá và hiểu rõ hơn về những kiến thức nền tảng cũng như các ứng dụng thực tế của trí tuệ nhân tạo, lĩnh vực đầy tiềm năng và không ngừng phát triển.

Môn *Trí tuệ nhân tạo và ứng dụng* là một môn học đầy thú vị, mang lại kiến thức bổ ích và có tính thực tiễn cao, giúp chúng em hiểu rõ hơn về cách AI được ứng dụng trong cuộc sống hiện đại. Môn học đã trang bị cho chúng em nền tảng kiến thức vững chắc, gắn liền với nhu cầu thực tiễn. Tuy nhiên, do kiến thức còn hạn chế và thiếu kinh nghiệm thực tiễn, bài đồ án của chúng em khó có thể tránh khỏi các thiếu sót. Chúng em rất mong nhận được sự xem xét và góp ý từ thầy để có thể hoàn thiện đồ án tốt hơn.

Xin chân thành cảm ơn thầy!

Nhận xét của giảng viên

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

…………………………………………………………………………………………..

**MỤC LỤC**

[PHẦN MỞ ĐẦU 6](#_Toc182032914)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 7](#_Toc182032915)

[1. Khái niệm giải thuật backtracking 7](#_Toc182032916)

[2. Nguyên lý của backtracking hoạt động 7](#_Toc182032917)

[3. Ví dụ về backtracking 8](#_Toc182032918)

[4. Ứng dụng giải thuật backtracking 10](#_Toc182032919)

[5. Kết chương 11](#_Toc182032920)

[CHƯƠNG 2: ÁP DỤNG GIẢI THUẬT BACKTRACKING VÀO TRÒ CHƠI SUDOKU 12](#_Toc182032921)

[1. Bài toán giải quyết vấn đề gì 12](#_Toc182032922)

[2. Dữ liệu của bài toán 12](#_Toc182032923)

[3. Lập trình 13](#_Toc182032924)

[3.1 Solve.py: 13](#_Toc182032925)

[3.2 GUI.py 16](#_Toc182032926)

[4. Kết quả xuất ra 23](#_Toc182032927)

[5. Kết chương 23](#_Toc182032928)

[CHƯƠNG 3: KẾT LUẬN 24](#_Toc182032929)

[Tài liệu tham khảo 25](#_Toc182032930)

Mục lục hình ảnh

[Hình 1.1: Quy trình Backtracking 7](#_Toc181884543)

[Hình 1.2: Bảng game Sudoku 9](#_Toc181884545)

[Hình 1.3: Bảng game Sudoku đã giải 10](#_Toc181884546)

[Hình 2.1: Cách thức hoạt động Sudoku 12](#_Toc181884547)

[Hình 2.2: Backtracking dưới dạng cây 13](#_Toc181884548)

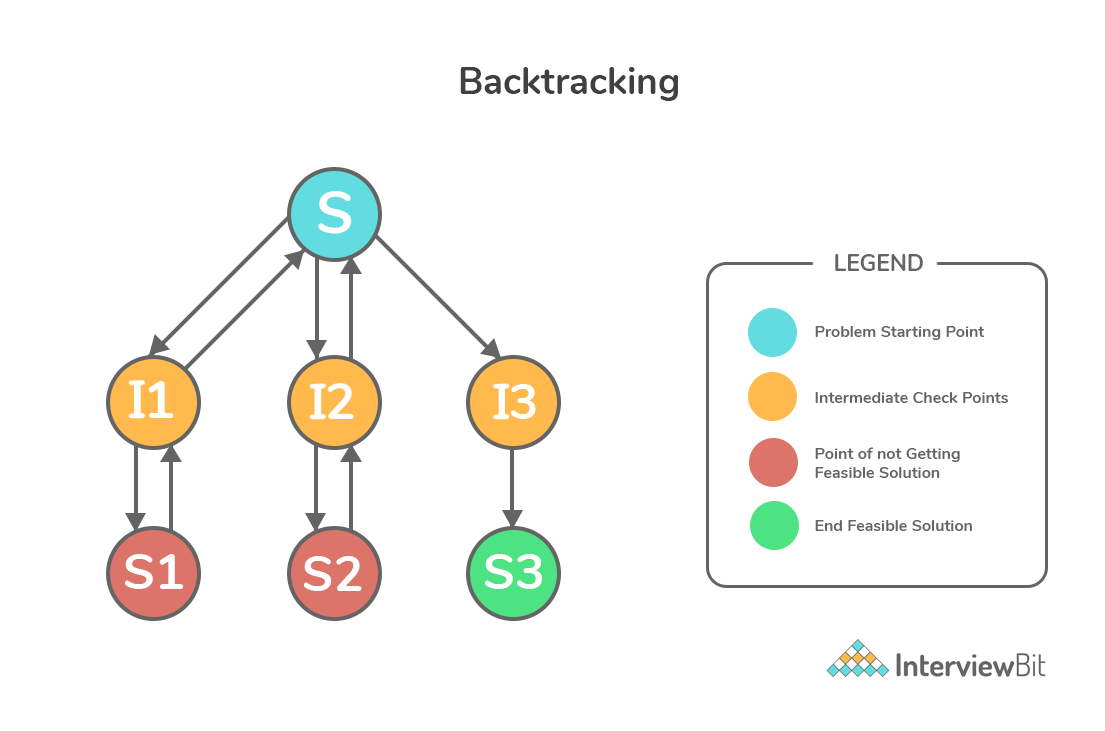
[Hình 2.3: Kết quả của lập trình 23](#_Toc181884549)

# **PHẦN MỞ ĐẦU**

Trong lĩnh vực AI và khoa học máy tính, thuật toán quay lui (backtracking) là một công cụ mạnh mẽ và hiệu quả để giải quyết các vấn đề phức tạp đòi hỏi tìm kiếm kết hợp và kiểm tra toàn bộ không gian lựa chọn. Phương pháp này thực hiện bằng cách thử từng bước, loại bỏ các lựa chọn không hợp lý nhằm tiết kiệm tài nguyên và tăng hiệu quả tìm kiếm. Nhờ vào điều đón đã biến thuật toán quay lui đặc biệt hữu dụng trong các bài toán như giải Sudoku, bài toán n-queen, tìm đường trong mê cung. Báo cáo sẽ phân tích chi tiết về thuật toán này, từ các nguyên lý cơ bản đến các bước triển khai, và minh họa qua **ứng dụng vào bài toán Sudoku** để hiểu rõ cách thuật toán giúp tối ưu hóa giải pháp.

# CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

## Khái niệm giải thuật backtracking

Quay lui là một kỹ thuật giải quyết các bài toán tìm kiếm theo từng bước bằng cách xây dựng các ứng viên cho các giải pháp, là một kĩ thuật thiết kế giải thuật dựa trên đê. Nó quay lại và bắt đầu lại theo một hướng khác khi nó đạt đến một trường hợp không hợp lệ của giải pháp, do đó có tên như vậy. Nó thường được sử dụng cho tổ hợp, tối ưu hóa và cũng như các bài toán tìm kiếm. [1]

Hình 1.1: Quy trình Backtracking [2]

## Nguyên lý của backtracking hoạt động

Thuật toán backtracking hoạt động dựa trên việc thử từng lựa chọn có thể và quay lại khi gặp điều kiện không thỏa mãn yêu cầu. Dưới đây là các bước chi tiết về quá trình thuật toán backtracking hoạt động:

Khởi tạo : Bắt đầu từ trạng thái ban đầu, khởi tạo các biến cần thiết để bắt đầu quá trình tìm kiếm bằng backtracking.

Kiểm tra điều kiện dừng: Xác định xem bài toán đã đạt đến điều kiện dừng hay chưa. Điều kiện dừng có thể là khi đã tìm được giải pháp hoặc đã thử hết các khả năng mà không tìm ra giải pháp. Nếu điều kiện dừng thỏa mãn, thuật toán sẽ trả về kết quả và kết thúc.

Lựa chọn và kiểm tra ràng buộc: Chọn một phương án từ không gian tìm kiếm và kiểm tra xem lựa chọn này có đáp ứng các ràng buộc của bài toán không. Nếu không thỏa mãn, thuật toán quay lại để thử lựa chọn khác.

Cập nhật trạng thái: Cập nhật trạng thái của bài toán theo phương án vừa chọn, có thể cập nhật các biến, mảng hoặc cấu trúc dữ liệu để phản ánh trạng thái hiện tại.

* Gọi đệ quy: Gọi đệ quy để tiếp tục tìm kiếm các lựa chọn kế tiếp. Quá trình này lặp lại cho đến khi tìm được giải pháp hoặc không còn lựa chọn khả thi nào.
* Quay lui: Khi đã thử hết các lựa chọn hoặc không tìm thấy giải pháp, thuật toán sẽ quay lại trạng thái trước đó và thử các lựa chọn khác. Bước quay lui này giúp khám phá các nhánh lựa chọn mới chưa được thử.
* Trả về kết quả: Khi quá trình backtracking hoàn tất, trả về giải pháp tìm thấy (nếu có) hoặc một thông báo rằng không tồn tại giải pháp nào.

Thuật toán backtracking sẽ tiếp tục quá trình này cho đến khi tìm được giải pháp hoặc xác nhận rằng không có giải pháp nào. Backtracking sẽ trở nên hiệu quả hơn khi áp dụng cắt nhánh và kiểm tra các điều kiện ràng buộc để loại bỏ những nhánh không cần thiết. Điều này giúp giảm khối lượng tính toán và tăng tốc độ tìm kiếm, đảm bảo hiệu quả khi giải quyết các bài toán tổ hợp phức tạp.

## Ví dụ về backtracking

* Để hiểu rõ hơn về thuật toán backtracking, chúng ta sẽ thử giải bài toán cụ thể: Giải Sudoku.. Dưới đây là một ví dụ về đầu vào và đầu ra của bài toán Sudoku:

Input: (Một bảng Sudoku có một số ô đã điền trước)

Ảnh có chứa trò chơi giải ô chữ, số, hình vuông, văn bản

Mô tả được tạo tự động

Hình 1.2: Bảng game Sudoku

Tìm ô trống đầu tiên: Thuật toán sẽ duyệt qua từng ô trong bảng để tìm ô trống (ô chứa số 0) đầu tiên cần được điền số.

Thử các giá trị từ 1 đến 9: Đối với ô trống, thuật toán sẽ thử điền lần lượt các số từ 1 đến 9 và kiểm tra xem số đó có hợp lệ hay không, theo quy tắc của Sudoku (không trùng trong hàng, cột, và vùng 3x3).

Kiểm tra ràng buộc: Đảm bảo rằng số vừa chọn không trùng với các số đã có trong hàng, cột và vùng 3x3 chứa ô hiện tại. Nếu số đó vi phạm các quy tắc của Sudoku, thuật toán sẽ thử số khác.

Gọi đệ quy: Nếu một số hợp lệ được điền vào ô trống, thuật toán sẽ đệ quy để tiếp tục giải các ô trống tiếp theo.

Quay lui nếu không hợp lệ: Nếu không có số nào trong khoảng từ 1 đến 9 hợp lệ cho ô trống, thuật toán sẽ quay lại ô trước đó để thử các lựa chọn khác, thay thế số đã chọn và tiếp tục đệ quy.

Hoàn thành giải pháp: Khi tất cả ô trống đã được điền mà không vi phạm quy tắc nào, thuật toán sẽ hoàn thành và trả về bảng Sudoku đầy đủ. Nếu không thể điền đủ, thuật toán sẽ báo rằng không tồn tại giải pháp hợp lệ.

Cách giải gồm các bước sau:

Bước 1: Thuật toán tìm ô trống đầu tiên (ô ở hàng 1, cột 3).

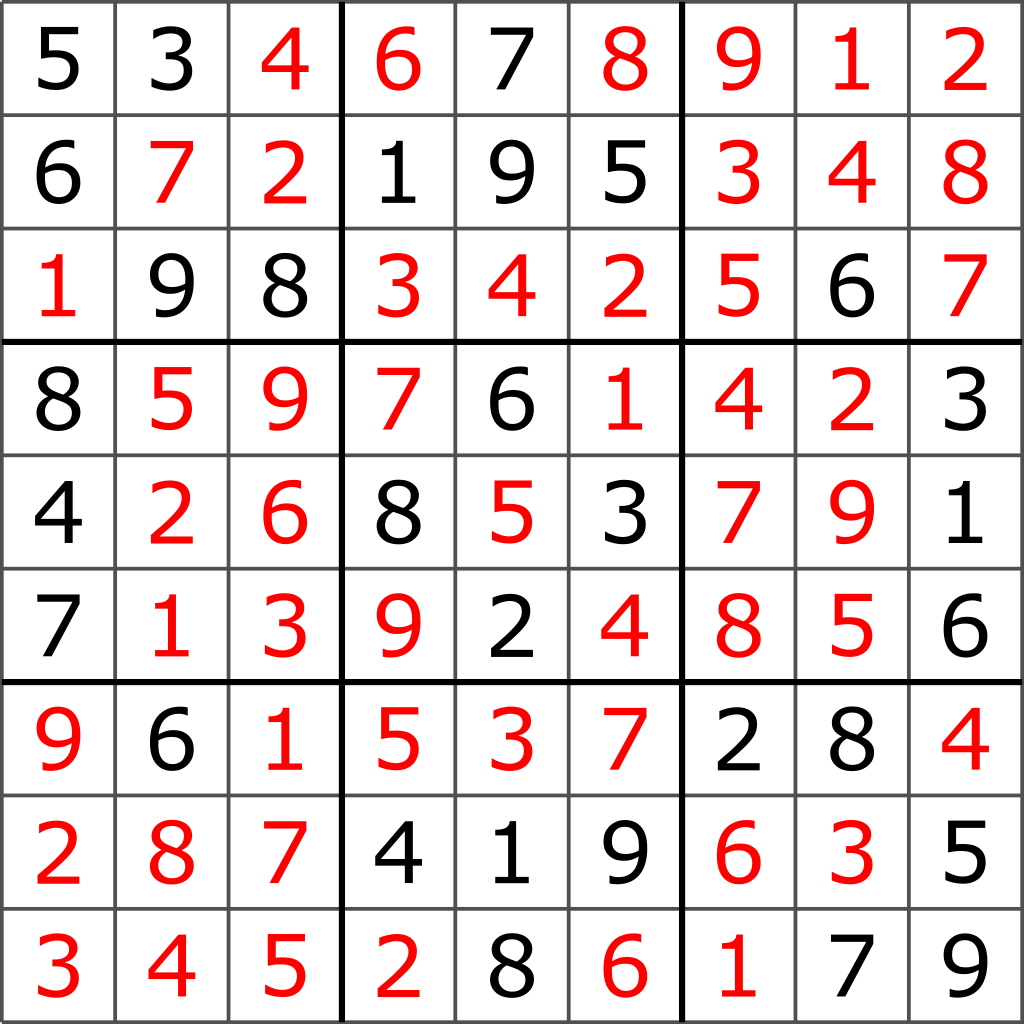
Bước 2: Thử điền các số từ 1 đến 9 vào ô đó và kiểm tra tính hợp lệ.

Bước 3: Giả sử số 1 không hợp lệ vì vi phạm quy tắc hàng/cột/ô 3x3, thuật toán sẽ thử số 2, rồi số 3, và tiếp tục như vậy cho đến khi tìm được số hợp lệ.

Bước 4: Nếu tìm thấy một số hợp lệ, thuật toán điền số đó vào ô và tiếp tục đến ô trống tiếp theo.

Bước 5: Nếu gặp ô mà không số nào từ 1 đến 9 hợp lệ, thuật toán sẽ quay lui về ô trước đó và thử một số khác.

Đây sẽ là kết quả của game Sudoku:



Hình 1.3: Bảng game Sudoku đã giải

## Ứng dụng giải thuật backtracking

Thuật toán Backtracking là một phương pháp mạnh mẽ trong việc giải quyết các bài toán tối ưu hóa và tổ hợp, đặc biệt khi cần tìm kiếm lời giải qua nhiều bước lựa chọn. Một số ứng dụng tiêu biểu của Backtracking bao gồm:

**Giải đố và trò chơi, xử lý văn bản và ngôn ngữ, tối ưu hóa lịch trình và kết hợp, tìm đường trong hệ thống định vị, các bài toàn đồ thị, giải bài toán điểm giao và tối ưu hóa, thiết kế mạch và sơ đồ điện tử.**

Cụ thể ở đây là áp dụng vào trò chơi Sudoku khi mà thuật toán sẽ điền số vào các ô trống và quay lui khi không thỏa mãn các quy tắc đã đặt ra.

## Kết chương

Khi chúng ta đối mặt với những bài toán đòi hỏi khả năng lựa chọn và tối ưu hóa, backtracking sẽ là một trong những công cụ đắc lực giúp tìm ra câu trả lời nhanh chóng và chính xác nhất.

# CHƯƠNG 2: ÁP DỤNG GIẢI THUẬT BACKTRACKING VÀO TRÒ CHƠI SUDOKU

## Bài toán giải quyết vấn đề gì

Đây là bài toán ràng buộc và đòi hỏi thuật toán tìm kiếm toàn bộ các đường đi có cả khả năng đạt được lời giải thỏa mã các yêu cầu.

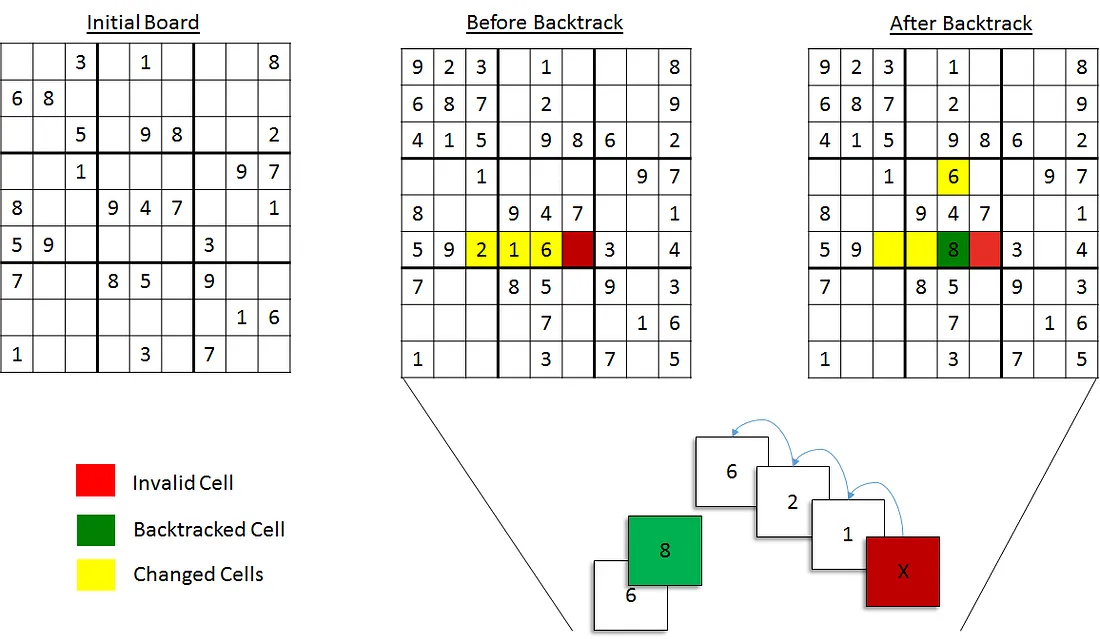
1. **Dữ liệu của bài toán**

Dữ liệu của bài toán là một bảng Sudoku 9x9 có tổng là 81 ô (9 hàng và 9 cột). Một số ô đã được điền ngẫu nhiên các số 1 đến 9 để trống các ô còn lại để người chơi điền. Trong đó có các ràng buộc như:

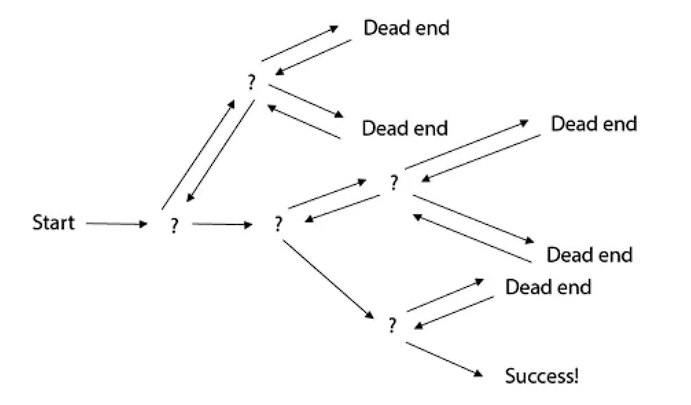
Hàng: Mỗi hàng phải chứa đủ các số từ 1 đến 9 mà không có số nào lặp lại.

Cột: Mỗi cột phải chứa đủ các số từ 1 đến 9 mà không có số nào lặp lại.

Ô Vuông 3x3: Mỗi ô vuông 3x3 (tổng cộng có 9 ô vuông) cũng phải chứa đủ các số từ 1 đến 9 mà không lặp lại. [3]



Hình 2.1: Cách thức hoạt động Sudoku [3]



Hình 2.2: Backtracking dưới dạng cây [3]

## Lập trình

Bài toán của chúng tôi sử dụng môi trường Visual Studio Code để thực hiện.

Dự án sẽ bao gồm 2 file chính:

### 3.1 Solve.py:

File **solve.py** này chứa mã Python để giải câu đố Sudoku bằng cách sử dụng giải thuật Backtracking:

* board = [
* [5,3,0 ,0,7,0, 0,0,0],
* [6,0,0 ,1,9,5, 0,0,0],
* [0,9,8 ,0,0,0, 0,6,0],
* [8,0,0 ,0,6,0, 0,0,3],
* [4,0,0 ,8,0,3, 0,0,1],
* [7,0,0 ,0,2,0, 0,0,6],
* [0,6,0 ,0,0,0, 2,8,0],
* [0,0,0 ,4,1,9, 0,0,5],
* [0,0,0 ,0,8,0, 0,7,9]
* ]
* def solve(bo)
* find = find\_empty(bo)
* if not find:
* return True
* else:
* row, col = find
* for i in range(1,10):
* if valid(bo, i, (row, col)):
* bo[row][col] = i
* if solve(bo):
* return True
* bo[row][col] = 0
* return False
* def valid(bo, num, pos):
* # Check row
* for i in range(len(bo[0])):
* if bo[pos[0]][i] == num and pos[1] != i:
* return False
* # Check column
* for i in range(len(bo)):
* if bo[i][pos[1]] == num and pos[0] != i:
* return False
* # Check box
* box\_x = pos[1] // 3
* box\_y = pos[0] // 3
* for i in range(box\_y\*3, box\_y\*3 + 3):
* for j in range(box\_x \* 3, box\_x\*3 + 3):
* if bo[i][j] == num and (i,j) != pos:
* return False
* return True
* def print\_board(bo):
* for i in range(len(bo)):
* if i % 3 == 0 and i != 0:
* print("- - - - - - - - - - - - - ")
* for j in range(len(bo[0])):
* if j % 3 == 0 and j != 0:
* print(" | ", end="")
* if j == 8:
* print(bo[i][j])
* else:
* print(str(bo[i][j]) + " ", end="")
* def find\_empty(bo):
* for i in range(len(bo)):
* for j in range(len(bo[0])):
* if bo[i][j] == 0:
* return (i, j)  # row, col
* return None
* print\_board(board)
* solve(board)
* print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")
* print\_board(board)

board: Biến board là một ma trận 9x9 đại diện cho bảng Sudoku cần được giải, trong đó các số 0 biểu thị các ô trống.

solve(bo): Hàm solve thực hiện giải thuật Backtracking để điền các ô trống trên bảng

* Hàm gọi find\_empty để tìm vị trí của một ô trống.
* Nếu không còn ô trống, hàm trả về True vì bảng đã hoàn tất.
* Nếu còn ô trống, hàm thử điền các số từ 1 đến 9 vào ô đó và kiểm tra tính hợp lệ bằng hàm valid.
* Nếu số hợp lệ, nó sẽ đặt số đó vào ô, gọi đệ quy solve(bo) tiếp tục cho các ô trống tiếp theo.
* Nếu việc điền không dẫn đến lời giải hợp lệ, ô đó được đặt lại là 0 và thử số tiếp theo.
* valid(bo, num, pos): Hàm valid kiểm tra xem một số num có thể điền vào vị trí pos trong bảng bo hay không:
* Kiểm tra xem num có trùng lặp trong hàng và cột không.
* Kiểm tra xem num có trùng lặp trong khối 3x3 chứa pos không.
* Nếu không vi phạm bất kỳ quy tắc nào, hàm trả về True; ngược lại trả về False.
* print\_board(bo): Hàm print\_board in bảng Sudoku ra màn hình với định dạng dễ đọc, chia các khối 3x3 bằng các dòng và cột.
* find\_empty(bo): Hàm find\_empty tìm và trả về tọa độ (hàng, cột) của ô trống đầu tiên (có giá trị 0). Nếu không còn ô trống, trả về None.

### 3.2 GUI.py

File GUI.py là chương trình Python tạo trò chơi Sudoku có giao diện đồ họa sử dụng thư viện pygame.

* # GUI.py
* import pygame
* from sovle import solve, valid
* import time
* pygame.font.init()
* class Grid:
* board = [
* [5,3,0 ,0,7,0, 0,0,0],
* [6,0,0 ,1,9,5, 0,0,0],
* [0,9,8 ,0,0,0, 0,6,0],
* [8,0,0 ,0,6,0, 0,0,3],
* [4,0,0 ,8,0,3, 0,0,1],
* [7,0,0 ,0,2,0, 0,0,6],
* [0,6,0 ,0,0,0, 2,8,0],
* [0,0,0 ,4,1,9, 0,0,5],
* [0,0,0 ,0,8,0, 0,7,9]
* ]
* def \_\_init\_\_(self, rows, cols, width, height, win):
* self.rows = rows
* self.cols = cols
* self.cubes = [[Cube(self.board[i][j], i, j, width, height) for j in range(cols)] for i in range(rows)]
* self.width = width
* self.height = height
* self.model = None
* self.update\_model()
* self.selected = None
* self.win = win
* def update\_model(self):
* self.model = [[self.cubes[i][j].value for j in range(self.cols)] for i in range(self.rows)]
* Grid là lớp đại diện cho toàn bộ bảng Sudoku với các phương thức hỗ trợ người chơi nhập số, kiểm tra đáp án, và vẽ bảng.
* \_\_init\_\_: Hàm khởi tạo, thiết lập các thông số của bảng như số dòng, cột, kích thước ô và cửa sổ game (win). Nó cũng tạo ra các ô vuông con (Cube) cho từng vị trí trên bảng.
* update\_model: Cập nhật lại trạng thái hiện tại của bảng Sudoku từ các ô (Cube).

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

Mô tả được tạo tự động

* place: Đặt một giá trị vào vị trí ô được chọn, kiểm tra tính hợp lệ và tự động giải nếu đúng.
* Kiểm tra tính hợp lệ: Trước khi đặt giá trị, phương thức sẽ gọi valid(bo, val, pos) để kiểm tra xem việc đặt giá trị này có vi phạm quy tắc nào không (trùng lặp trong hàng, cột, hoặc khối 3x3).
* Cập nhật bảng: Nếu giá trị hợp lệ, nó sẽ cập nhật ô tương ứng trong bảng và có thể thực hiện thêm các thao tác khác như lưu lại lịch sử hay hiển thị thông báo.
* sketch(self, val): Đánh dấu tạm thời giá trị val vào ô được chọn.
* Đánh dấu tạm thời: Thay vì đặt giá trị vĩnh viễn vào ô, phương thức này sẽ lưu giá trị tạm thời để người chơi có thể xem xét.
* Hỗ trợ quyết định: Giúp người chơi thử nghiệm các giá trị khác nhau mà không làm mất thông tin đã nhập.
* draw(self): Vẽ các đường lưới và ô trên cửa sổ.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

Mô tả được tạo tự động

* select(self, row, col): Chọn một ô theo tọa độ hàng và cột.
* Cập nhật trạng thái: Khi chọn ô, phương thức có thể thay đổi trạng thái để phản hồi lại sự chọn lựa của người dùng.
* Chọn ô cụ thể: Ghi nhớ tọa độ ô đã chọn để có thể thực hiện các thao tác tiếp theo (như place hoặc sketch).
* clear(self): Xóa giá trị tạm thời trong ô được chọn.
* Xóa giá trị tạm thời: Nếu ô đã chọn có một giá trị tạm thời, phương thức sẽ xóa nó đi, cho phép người chơi thử nghiệm với các giá trị khác.
* Trả về trạng thái ban đầu: Đưa ô trở về trạng thái chưa có giá trị nào.
* click(self, pos): Xử lý sự kiện nhấp chuột và xác định ô được nhấp.
* Xác định ô được nhấp: Tính toán tọa độ hàng và cột từ vị trí nhấp chuột trên giao diện người dùng.
* Cập nhật chọn lựa: Gọi select(row, col) để cập nhật ô được chọn.
* is\_finished(self): Kiểm tra xem trò chơi đã hoàn thành hay chưa.
* Kiểm tra bảng: Duyệt qua tất cả các ô trong bảng để xác định xem có ô nào trống hay không.
* Trả về kết quả: Nếu tất cả các ô đều đã được điền, phương thức sẽ trả về True, cho biết trò chơi đã hoàn thành.

Ảnh có chứa văn bản, ảnh chụp màn hình

Mô tả được tạo tự động

* solve(self): Giải quyết bảng Sudoku bằng cách sử dụng thuật toán đệ quy.
* solve\_gui(self): Giải quyết bảng với cập nhật giao diện người dùng.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

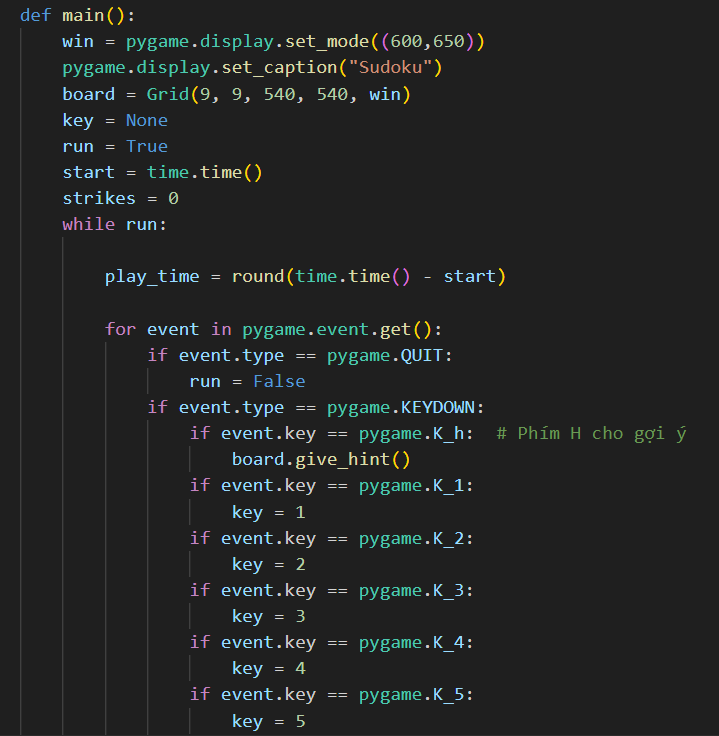
Mô tả được tạo tự động

* \_\_init\_\_(self, x, y, z, size): Khởi tạo một khối lập phương với tọa độ (x, y, z) và kích thước (size).
* draw(self): Vẽ giá trị hiện tại của ô Sudoku lên cửa sổ win.
* Sử dụng phông chữ "comicsans" để tạo văn bản với kích thước 40.
* Nếu ô có giá trị tạm thời (temp) khác 0 và giá trị chính thức (value) bằng 0, hiển thị giá trị tạm thời bằng màu xám.
* Nếu giá trị chính thức khác 0, hiển thị giá trị đó bằng màu đen.
* Nếu ô được chọn (selected là True), vẽ một hình chữ nhật màu đỏ quanh ô để làm nổi bật.
* draw\_change(self, win, g=True): Vẽ ô với hiệu ứng đặc biệt khi giá trị của nó thay đổi.
* Tương tự như draw, nhưng sẽ xóa nội dung của ô trước khi vẽ lại.
* Hiển thị giá trị của ô ở giữa ô bằng màu đen.
* Nếu tham số g là True, vẽ viền ô bằng màu xanh lá cây để chỉ ra rằng giá trị đã thay đổi thành công; nếu không, viền sẽ là màu đỏ để chỉ ra rằng giá trị không hợp lệ.
* set(self, val): Đặt một giá trị chính thức cho ô.
* set\_temp(self, val): Đặt một giá trị tạm thời cho ô.

Ảnh có chứa văn bản, ảnh chụp màn hình

Mô tả được tạo tự động

* find\_empty(bo): Tìm ô trống (giá trị bằng 0) trong bảng.
* valid(bo, num, pos): Kiểm tra xem số num có thể được đặt tại vị trí pos hay không.
* redraw\_window(win, board, time, strikes): Vẽ lại cửa sổ trò chơi với các thông tin như thời gian đã trôi qua, số lần sai (strikes) và bảng hiện tại.
* format\_time(secs): Định dạng thời gian đã trôi qua (tính bằng giây) thành chuỗi dễ đọc theo định dạng "phút, giây".

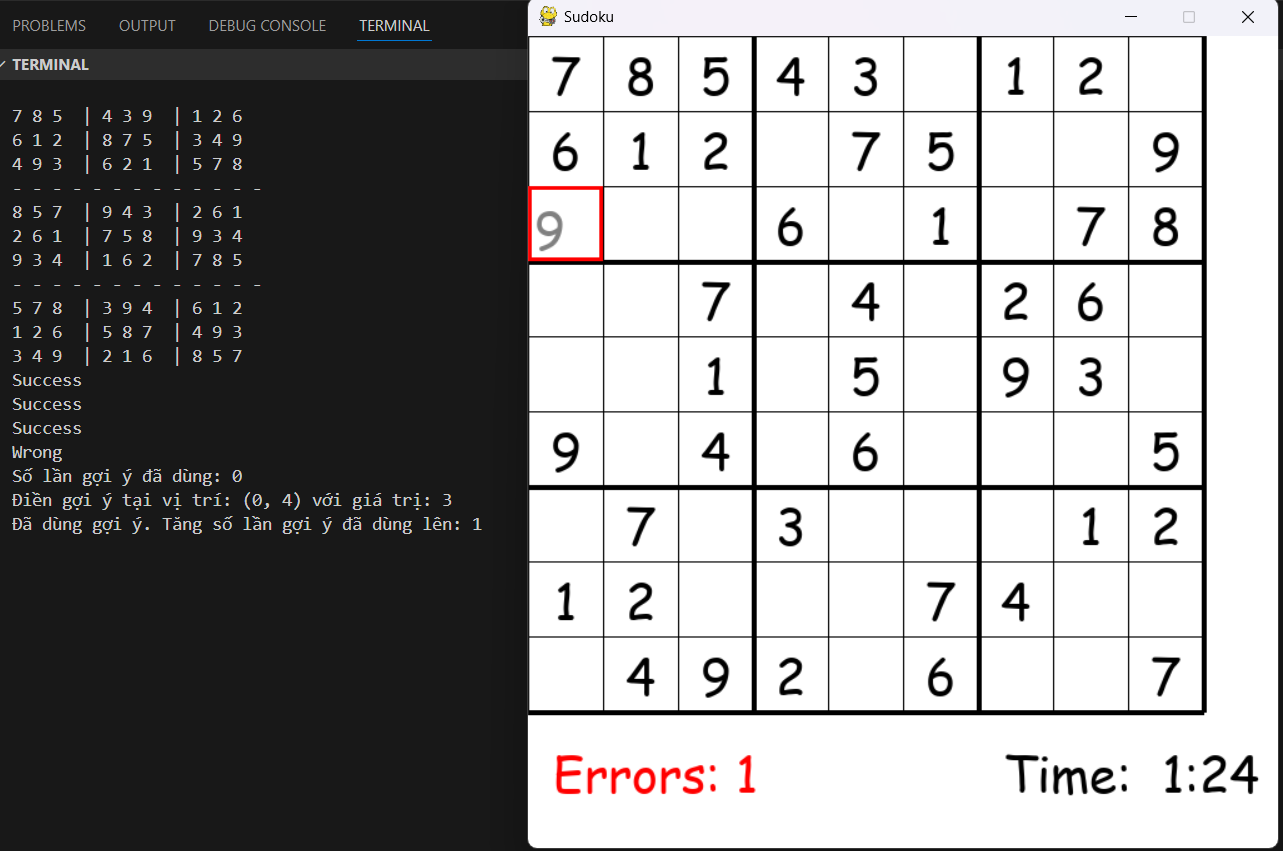


Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

Mô tả được tạo tự động

* Hàm main(): Khởi tạo cửa sổ trò chơi, tạo đối tượng Grid, xử lý các sự kiện từ bàn phím và chuột, vẽ lại cửa sổ, và kiểm tra các điều kiện thắng/thua.

## Kết quả xuất ra



Hình 2.3: Kết quả của lập trình

Khi chạy chương trình, chúng ta sẽ thấy một bảng Sudoku 9x9 với số đã cho và ô trống, thời gian chơi và số lần nhầm hiển thị ở góc dưới, cùng với các phím nhập để điền giá trị và thông báo "Success", "Wrong", “Số lần dùng gợi ý” và "Game over" tùy thuộc vào kết quả. Ở mục terminal là kết quả đúng của game Sudoku.

## Kết chương

Áp dụng giải thuật Backtracking vào trò chơi Sudoku cho phép chúng ta tìm kiếm giải pháp bằng cách thử nghiệm từng giá trị có thể cho từng ô trong bảng Sudoku, quay lại khi gặp phải sự mâu thuẫn và tiếp tục với các lựa chọn khác, việc giải quyết các bài toán tổ hợp phức tạp đã mở ra nhiều hướng đi cho các ứng dụng lập trình khác trong tương lai.

# CHƯƠNG 3: KẾT LUẬN

Áp dụng Backtracking vào Sudoku đã chứng rằng nó không chỉ đơn thuần là một phương pháp tìm giải pháp mà còn mở ra những cách hoàn toàn mới để phát triển trò chơi, tạo ra các bài toán có độ khó tăng lên và hỗ trợ cho các hệ thống tối ưu hóa tự động.

Qua thử nghiệm, chúng tôi thấy rằng Backtracking là một công cụ hữu ích với tiềm năng ứng dụng rộng rãi trong nhiều lĩnh vực công nghệ, tuy nhiên dự án còn nhiều thiếu sót và chưa hoàn thiện, dự rằng trong tương lai sẽ cải tiến trò chơi tốt hơn mang lại nhiều trải nghiệm thú vị cho người sử dụng.

# Tài liệu tham khảo

|  |  |
| --- | --- |
| [1] | T. C. CÁCH, “Tìm hiểu về Thuật toán quay lui (Backtracking),” 25 2 2023. [Trực tuyến]. Available: https://tuicocach.com/tim-hieu-ve-thuat-toan-quay-lui-backtracking/. |
| [2] | "InterviewBit," 17 11 2023. [Online]. Available: https://www.interviewbit.com/courses/programming/backtracking/. |
| [3] | P. Mistry, "Medium," Backtracking Sudoku, 19 2 2019. [Online]. Available: https://medium.com/@priyankmistry1999/backtracking-sudoku-6e4439e4825c. |
| [4] | "Algorithm to Solve Sudoku," 30 7 2024. [Online]. Available: https://www.geeksforgeeks.org/sudoku-backtracking-7/. |
| [5] | techacademy, “Leetcode: Valid Sudoku,” 18 1 2024. [Trực tuyến]. Available: https://techacademy.edu.vn/bai-36-leetcode-valid-sudoku/. |
| [6] | “download.vn,” 26 4 2019. [Trực tuyến]. Available: https://download.vn/cach-choi-sudoku-18200. |