NOVEMBER, 2024

N-puzzle, Sudoku, Path Finding

Báo cáo Bài tập lớn 1

GIẢNG VIÊN HƯỚNG DẪN

Vương Bá Thịnh

NHÓM LỚP

L01

Thành viên

2110120	Hoàng Minh Hải Đăng
2113055	Bùi Tiến Dũng
2111118	Nguyễn Hồ Nhật Hà
2211909	Trương Thành Long
2112198	Nguyễn Thái Sơn

Mục lục

01

N-puzzle

1.1 - Giới thiệu bài toán

Bài toán N-Puzzle với N = 15

Mô tả

Là trò chơi xếp hình cổ điển, trong đó có một bảng vuông với một ô trống và các ô còn lại chứa các số từ 1 đến N.

Mục tiêu

Sắp xếp các số theo thứ tự tăng dần từ trái sang phải và từ trên xuống dưới, bằng cách di chuyển các ô trống.

1.2 - Biểu diễn trạng thái

12 14 6 4 2 5 7 1 0 3 8 15 10 9 11 13

Danh sách biểu diễn trạng thái

Các trạng thái

Biểu diễn thông qua một danh sách có kích thước N + 1.

Phần tử trong danh sách

- Có giá trị x, x ∈ 0,...N.
- Lần lượt chứa giá trị x đôi một khác nhau.

Phần tử 0

- Biểu diễn vị trí của ô trống.
- Là phần tử duy nhất được phép di chuyển.

1.3 - Trạng thái khởi đầu & mục tiêu

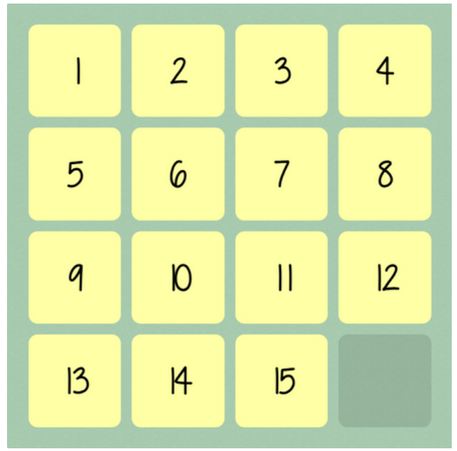
Trạng thái khởi đầu

Là trạng thái đầu tiên mà đề bài đưa ra

Đề bài 15-puzzle

Trạng thái mục tiêu

Là trạng thái mà các số trên bảng được sắp xếp theo thứ tự tăng dần từ trái sang phải, từ trên xuống dưới và ô trống sẽ nằm ở vị trí cuối cùng trong bảng.



Trạng thái mục tiêu của 15-puzzle

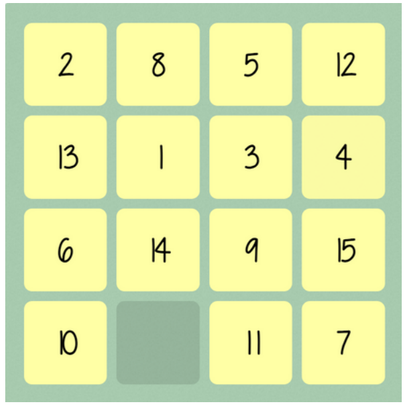
1.4 - Luật di chuyển

Trạng thái của trò chơi thay đổi khi thay đổi vị trí của ô trống trên bảng. **Luật di chuyển** là tập hợp các quy tắc sao cho việc di chuyển ô trống trong hình vuông là **hợp lệ**, các quy tắc bao gồm:

1. Nếu ô trống đang ở hàng trên cùng của hình vuông, nó không thể di chuyển theo hướng đi lên.
2. Nếu ô trống đang ở sát lề bên phải của hình vuông, nó không thể di chuyển sang phải.
3. Nếu ô trống đang ở sát lề bên trái của hình vuông, nó không thể di chuyển sang trái.
4. Nếu ô trống đang ở hàng dưới cùng của hình vuông, nó không thể di chuyển xuống dưới.

1.4 - Luật di chuyển

Không thể di chuyển lên trên



Không thể di chuyển xuống dưới

Không thể di chuyển sang trái

Không thể di chuyển sang phải

1.4 - Luật di chuyển

Để giải thuật tìm kiếm hoạt động hiệu quả hơn, ta thêm vào một số **luật di chuyển bổ sung** để tránh việc sinh ra các trạng thái đã thực hiện trước đó. Các luật này không cho phép ô trống di chuyển theo hướng ngược lại để trở về trạng thái trước đó.

1. Nếu trạng thái hiện tại được tạo thành từ việc di chuyển ô trống sang bên trái, ô trống hiện tại không được di chuyển sang bên phải.

2. Nếu trạng thái hiện tại được tạo thành từ việc di chuyển ô trống sang bên phải, ô trống hiện tại không được di chuyển sang bên trái.

3. Nếu trạng thái hiện tại được tạo thành từ việc di chuyển ô trống lên trên, ô trống hiện tại không được di chuyển xuống dưới.

4. Nếu trạng thái hiện tại được tạo thành từ việc di chuyển ô trống xuống dưới, ô trống hiện tại không được di chuyển lên trên.

1.5 - Hàm sinh trạng thái khởi đầu

Để đảm bảo bài toán N-puzzle được đưa ra **có thể giải được (có tồn tại lời giải)**, cần phải hiện thực **hàm sinh trạng thái khởi đầu** cho bài toán này. Hàm sinh được hiện thực dựa trên ý tưởng như sau:

1. **Sinh ngẫu nhiên** một trạng thái của bài toán N-puzzle với kích thước k được chỉ định (N = k*k - 1).

2. **Kiểm tra** xem với trạng thái ban đầu đó, bài toán có tồn tại lời giải hay không. Nếu có, trả về trạng thái này làm trạng thái ban đầu của bài toán, nếu không thì thực hiện lại bước 1.

1.5 - Hàm sinh trạng thái khởi đầu

Để xác định bài toán N-puzzle (ví dụ 8-puzzle, 15-puzzle) có lời giải hay không, ta cần xem xét hai yếu tố chính:

Số lần nghịch đảo (inversions)

"Nghịch đảo" xảy ra khi một số lớn hơn nằm trước một số nhỏ hơn (ngoại trừ ô trống) khi các ô được đọc theo thứ tự từ trên xuống dưới và từ trái qua phải.

Vị trí của ô trống

Đối với puzzle có kích thước k*k, vị trí của ô trống cũng ảnh hưởng đến tính giải được của bài toán.

Quy tắc xác định một puzzle là có thể giải được hay không như sau:

Kích thước k lẻ (ví dụ 8-puzzle)

Có thể giải được nếu tổng số nghịch đảo là số chẵn.

Kích thước k chẵn (ví dụ 15-puzzle)

Có thể giải được nếu:

- Số nghịch đảo lẻ, ô trống nằm trên hàng chẵn từ dưới lên.
- Số nghịch đảo chẵn, ô trống nằm trên hàng lẻ từ dưới lên.

1.6 - Hàm sinh trạng thái mục tiêu

Trạng thái mục tiêu của bài toán N-Puzzle là trạng thái mà các số trên bảng được sắp xếp theo thứ tự tăng dần từ trái sang phải, từ trên xuống dưới và ô trống sẽ nằm ở vị trí cuối cùng trong bảng. **Trạng thái mục tiêu của bài toán được sinh ra như sau:**

```
# Goal state: [1,2,...,k*k - 1,0]
self.goal = list(range(1, size * size)) + [0]
```

1.7 - Giải thuật

Bài toán được giải bằng **giải thuật DFS** có cơ chế hoạt động như sau:

1

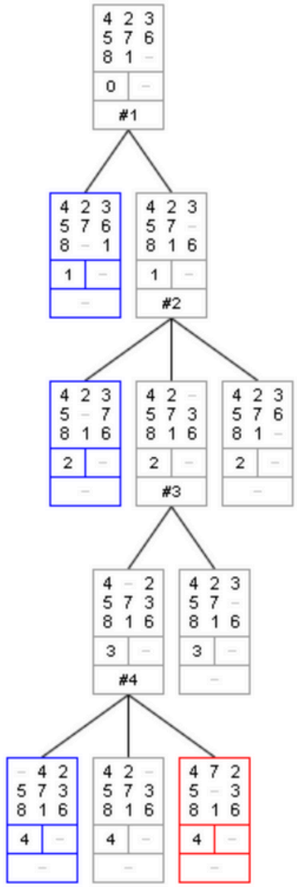
Bắt đầu từ nút gốc giải thuật sẽ tiến hành so sánh với trạng thái mục tiêu.

Nếu đã thỏa mãn trạng thái mục tiêu thì dừng tìm kiếm, ngược lại thì sinh ra các nút tiếp theo và đưa các nút đó vào stack.

2

Tiếp tục pop từng nút trong stack để tiến hành kiểm tra như ở bước 1 cho đến khi đạt được trạng thái mục tiêu hoặc khi không còn trạng thái nào để kiểm tra nữa.

1.7 - Giải thuật



Minh họa quá trình tìm kiếm bằng DFS

1.7 - Giải thuật

Một số nhận xét sau khi giải bài toán bằng DFS:

1

DFS chỉ có thể giải được các bài toán N-puzzle với N = 8, với các bài toán có kích thước lớn hơn thì số trạng thái sẽ tăng nhanh chóng theo cấp số nhân nên việc giải quyết bằng giải thuật DFS trên máy tính cá nhân là không khả thi.

2

DFS hoạt động theo cơ chế duyệt theo chiều sâu, nghĩa là khi không có giới hạn về độ sâu thì nó sẽ duyệt mãi theo một nhánh cho tới khi đạt đến trạng thái mục tiêu. Việc này dẫn đến lời giải của giải thuật DFS cho bài toán N-puzzle chứa số lượng bước di chuyển là rất lớn (lên đến hàng trăm ngàn bước) và không khả thi khi áp dụng tìm lời giải cho các bài toán ngoài thực tế.

1.7 - Giải thuật

Để giải quyết các nhược điểm của DFS, nhóm dùng **giải thuật cải tiến của** DFS là IDS để giải bài toán N-Puzzle.

Giải thuật IDS có cơ chế hoạt động như sau:

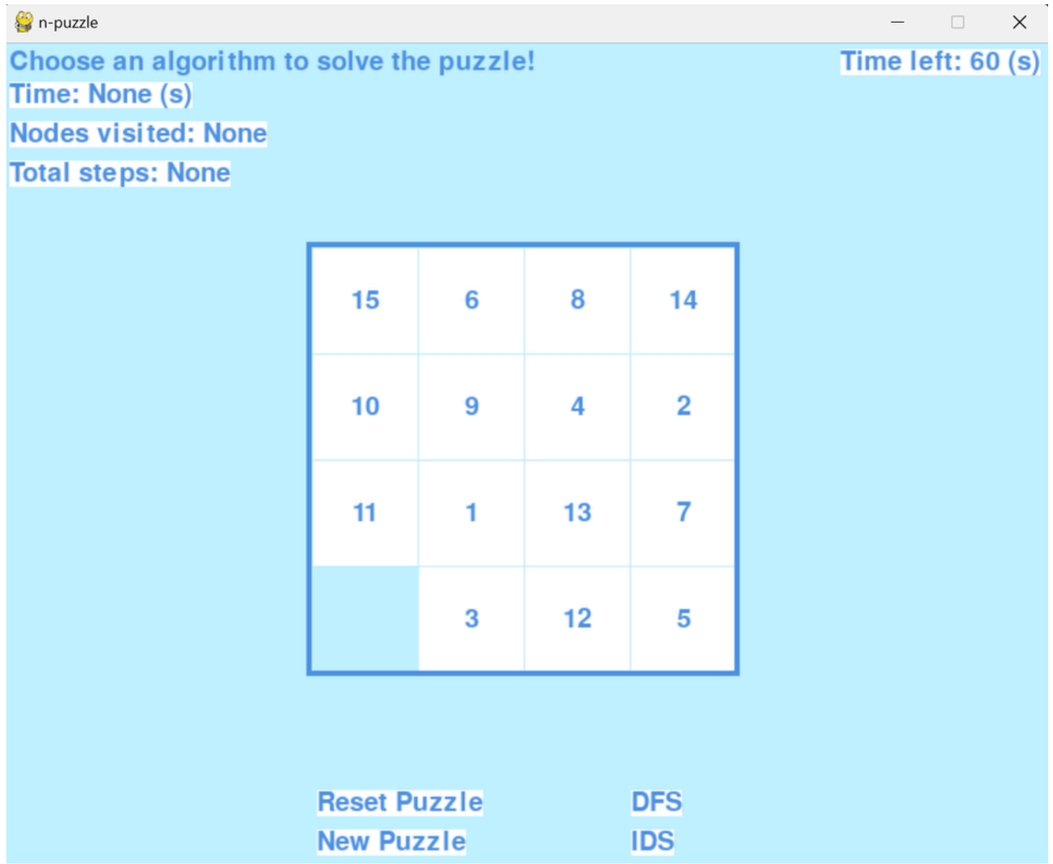
1. Thực thi giải thuật DLS (DFS có giới hạn độ sâu) với độ sâu tăng dần từ 1 tới độ sâu tối đa.
2. Nếu tìm thấy lời giải, trả về lời giải và kết thúc thuật toán. Nếu không, tiếp tục thực hiện DLS với độ sâu tăng lên 1.

- IDS có **giới hạn độ sâu tìm kiếm** nên lời giải tìm được chứa **số bước di chuyển ít hơn** rất nhiều so với DFS và có thể chấp nhận được.
- IDS lặp lại giải thuật với nhiều độ sâu giới hạn khác nhau cho đến khi tìm thấy lời giải ở một độ sâu thích hợp, có thể **tránh được việc lặp vô hạn** trong những bài toán có số trạng thái khổng lồ như 15-puzzle.

1.8 - Giao diện trò chơi

Trên giao diện, người dùng có 4 lựa chọn là:

- **New Puzzle:** Tạo game mới dựa trên kích thước k mà người dùng nhập vào.
- **Reset Puzzle:** Reset bài toán hiện tại về trạng thái ban đầu.
- **DFS**: Giải bài toán bằng giải thuật DFS.
- **IDS:** Giải bài toán bằng giải thuật IDS



Giao diện trò chơi N-Puzzle

02

Sudoku

2.1 Giới thiệu bài toán

2.2 Biểu diễn trạng thái

2.3 Trạng thái khởi đầu & mục tiêu

Luật di chuyển

2.4 Hàm sinh lượng giá trạng thái

2.5 Hàm sinh trạng thái khởi đầu

2.6 Giải thuật di truyền

2.7 Giao diện trò chơi

2.1 - Giới thiệu bài toán

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

Đề bài Sudoku mức độ dễ

Mô tả

- Là 1 trò chơi giải đố, với mục tiêu là điền hết tất cả các ô trong lưới 9 * 9 sao cho mỗi hàng, mỗi cột, mỗi khối con 3 * 3 đều chứa tất cả các số nguyên từ 1 đến 9 và mỗi số chỉ xuất hiện đúng một lần.
- Với mỗi bài toán hợp lệ, chỉ có **duy nhất một lời giải** được tìm ra.
- **Độ khó phụ thuộc vào số lượng các số** đã được cho trước.

2.1 - Giới thiệu bài toán

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

Bài toán Sudoku mức độ dễ

Mô tả

- Được coi là **bài toán NP-complete**, nghĩa là không có thuật toán giải quyết nhanh chóng cho mọi trường hợp.
- Ở trạng thái bắt đầu, có thể sinh ra được khoảng 6.67 * 10^21 nước đi mà chỉ tồn tại một nước đi hợp lệ trong đó.
- Nếu số ô cho trước ít hơn hoặc bằng 16, bài toán không thể đảm bảo có duy nhất một lời giải.

2.2 - Biểu diễn trạng thái

Không gian trạng thái

- Bao gồm tất cả các bảng 9x9, với mỗi ô trong bảng có thể chứa một số từ 1 đến 9.
- Kích thước rất lớn, vì có 9^81 trạng thái có thể có.
- Giải thuật di truyền chỉ làm việc với một tập hợp con nhỏ của không gian trạng thái ở mỗi thế hệ, được gọi là quần thể (population).

Định nghĩa trạng thái bài toán

- Một trạng thái được biểu diễn bằng một bảng 9x9, trong đó mỗi ô chứa một số từ 1 đến 9.
- Một bảng đầy đủ có 81 ô đã được điền đầy đủ, có thể hợp lệ hoặc không hợp lệ.
- Trong bài toán này, một trạng thái được biểu diễn bằng một ma trận 9x9.

2.3 - Trạng thái khởi đầu & mục tiêu Luật di chuyển

Trạng thái khởi đầu

- Bảng Sudoku đã cho trong đề bài, với một số ô được điền giá trị và các ô trống được đánh dấu bằng số 0.
- Các ô đã điền được xem là "cố định" và không được phép thay đổi trong suốt quá trình tiến hóa.

Trạng thái mục tiêu

- Bảng Sudoku đầy đủ và hợp lệ.
- Mỗi hàng, cột và khối 3x3 đều chứa các số từ 1 đến 9 mà không có số trùng lặp theo quy luật Sudoku.

Luật di chuyển

Là cách điền một số từ 1 đến 9 vào một ô trống trong bảng theo luật của Sudoku sao cho mỗi hàng, cột và khối 3x3 đều chứa các số từ 1 đến 9 mà không có số trùng lặp.

2.4 - Hàm lượng giá trạng thái

Hàm lượng giá **đánh giá mức độ phù hợp** của mỗi cá thể bằng cách **đo lường mức độ gần đạt đến trạng thái hợp lệ** của Sudoku. Hàm lượng giá được tính bằng cách **đếm số lượng các số duy nhất** từ 1 đến 9 trong mỗi hàng, cột và khối 3x3:

1. Tổng điểm của tất cả các hàng, cột và khối được chuẩn hóa trong khoảng từ 0 đến 1.

2. Với mỗi hàng, cột và khối 3x3, nếu có đủ các số từ 1 đến 9 mà không bị trùng lặp, cá thể đạt điểm tối đa cho phần đó.

2.4 - Hàm lượng giá trạng thái

Công thức tính điểm

Công thức tính điểm cho mỗi hàng/cột/khối 3*3: $score = \frac{1.0/N}{9}$

Với N là số lượng các ký tự số duy nhất trong hàng/cột/khối tương ứng.

Khi này mỗi phần sẽ có điểm tối đa là 1/9 và tổng điểm tối đa tất cả hàng, cột và khối có thể đạt được là 1.0

Điểm lượng giá

Điểm lượng giá (fitness) của một cá thể bằng tích tổng điểm của các hàng, cột và khối 3x3 trong bảng Sudoku của cá thể đó.

Một cá thể có điểm lượng giá đạt giá trị 1.0 nghĩa là bảng Sudoku của cá thể đó hoàn toàn hợp lệ, không có số nào bị lặp lại trong các hàng, cột hoặc khối 3x3.

2.5 - Hàm sinh trạng thái khởi đầu

Hàm sinh được hiện thực dựa trên kỹ thuật **heuristic** kết hợp với việc giảm dần số lượng lựa chọn của các ô để tạo bảng Sudoku hoàn chỉnh và thuật toán **backtracking** để tạo ra bảng Sudoku khởi đầu:

1. Khởi tạo danh sách các ô trong bảng Sudoku và tạo một bảng Sudoku trống.
2. Duyệt từng ô, chọn ô có ít giá trị khả thi nhất, dựa trên số lượng giá trị còn lại có thể điền vào ô đó.
3. Lựa chọn ngẫu nhiên một giá trị hợp lệ để điền vào ô. Sau khi điền, loại bỏ giá trị này khỏi các ô liên quan (cùng hàng, cột hoặc khối 3x3).
4. Tiếp tục cho đến khi tất cả các ô trong bảng đều được điền giá trị hợp lệ.
5. Xóa các ô trống dựa trên cấp độ, xác định bài toán có duy nhất một lời giải.

2.5 - Hàm sinh trạng thái khởi đầu

Để đảm bảo bảng Sudoku sinh ra hợp lệ, quá trình **kiểm tra và điền số** vào bảng phải theo các nguyên tắc sau:

1. Mỗi số trong một hàng, cột và khối 3x3 phải là duy nhất, không được lặp lại.

2. Đảm bảo rằng việc lựa chọn ô có ít lựa chọn nhất sẽ giảm thiểu khả năng gặp phải các tình huống không thể điền được giá trị sau này.

3. Đảm bảo chỉ có duy nhất một lời giải cho bài toán Sudoku.

2.5 - Hàm sinh trạng thái khởi đầu

Đề bài Sudoku được sinh ra sẽ được phân độ khó theo ba cấp độ:

- **Dễ (Easy):** chứa 30-35 ô trống.
- **Trung bình (Medium):** chứa 36-45 ô trống.
- **Khó (Hard)**: chứa 46-55 ô trống.

```
def generate_sudoku(sudoku, difficulty):
    if difficulty == "EASY":
        num_holes = random.randint(30, 35)
    elif difficulty == "MEDIUM":
        num_holes = random.randint(36, 45)
    elif difficulty == "HARD":
        num_holes = random.randint(46, 55)
    else:
        raise ValueError("Invalid difficulty level")

    remove_cells(sudoku, num_holes, difficulty)
    return sudoku
```

2.6 - Giải thuật di truyền

Để giải bài toán Sudoku bằng giải thuật di truyền, ta sẽ định nghĩa một số lớp (class) hỗ trợ như sau:

1. **Candidate:** Lớp biểu diễn một ứng viên trong quần thể.
2. **Population:** Lớp biểu diễn cho một quần thể gồm nhiều ứng viên
3. **GivenPuzzle:** Lớp con của Candidate, đại diện cho bảng Sudoku đã cho ở đề bài với một số ô được điền trước.
4. **Crossover:** Lớp thực hiện việc lai ghép giữa các ứng viên để tạo ra ứng viên mới.
5. **Tournament:** Lớp biểu diễn cho một vòng đấu trong quần thể để lựa chọn ứng viên cho quá trình lai ghép.

2.6 - Giải thuật di truyền

Bài toán được giải bằng giải thuật di truyền thông qua phương thức solve() của class Sudoku với các bước sau:

1. Khởi tạo quần thể ban đầu bằng cách sử dụng phương thức seed() của lớp Population.

2. Lặp qua từng thế hệ với số thế hệ tối đa được giới hạn bởi thuộc tính generations của class Sudoku.

3. Trong mỗi thế hệ, lựa chọn ra ứng viên tốt nhất (dựa trên điểm fitness) và kiểm tra xem đó có phải là lời giải của bài toán hay không. Nếu phải, trả về lời giải bài toán và kết thúc chương trình.

2.6 - Giải thuật di truyền

4 Nếu không phải là giải pháp, tiến hành tạo ra thế hệ kế tiếp bằng cách lựa chọn, lai ghép và đột biến. Quá trình này được lặp lại cho đến khi đạt điều kiện kết thúc.

5 Trong quá trình lặp, tỷ lệ đột biến được điều chỉnh dựa trên quy tắc 1/5 của Rechenberg.

6 Nếu quần thể trở nên "stale" (nghĩa là không có sự thay đổi, cải thiện trong quần thể sau một số thế hệ liên tiếp), giải thuật sẽ tạo ra một quần thể mới bằng cách thực hiện lại quá trình khởi tạo quần thể ban đầu.

7 Cuối cùng, nếu không có lời giải nào được tìm thấy sau khi đã thực hiện một số lượng thế hệ nhất định, giải thuật sẽ thông báo không có lời giải nào được tìm thấy.

2.7 - Giao diện trò chơi

1

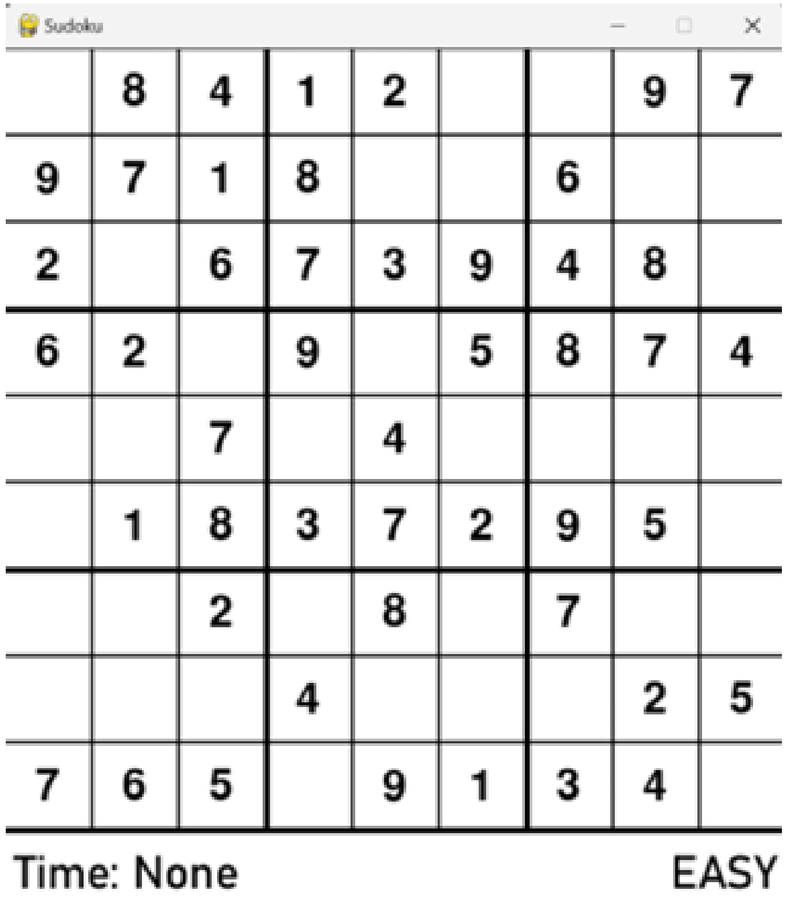
- Lựa chọn độ khó.
- Giao diện hiển thị 3 mức độ khó gồm Easy, Medium, Hard.

Giao diện trò chơi Sudoku

2.7 - Giao diện trò chơi

2

- Sau khi lựa chọn độ khó, trò chơi sẽ sinh ra đề bài tự động theo độ khó người dùng lựa chọn và hiển thị lên giao diện.
- Để giải đề bài Sudoku bằng giải thuật di truyền, người dùng nhấn vào phím Space trên bàn phím.

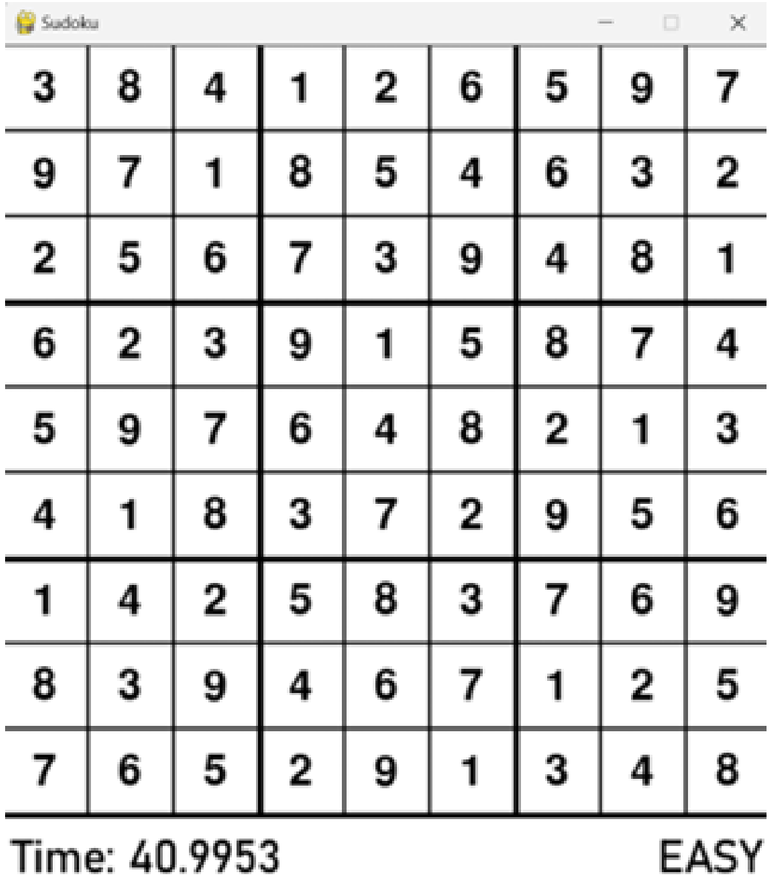


Giao diện trò chơi Sudoku

2.7 - Giao diện trò chơi

3

- Sau khi giải thuật di truyền được thực hiện xong và có lời giải, giao diện trò chơi sẽ được tự động điền vào các số còn thiếu trên bảng Sudoku dựa vào lời giải và hiển thị thời gian mà giải thuật đã dùng để tìm ra lời giải.

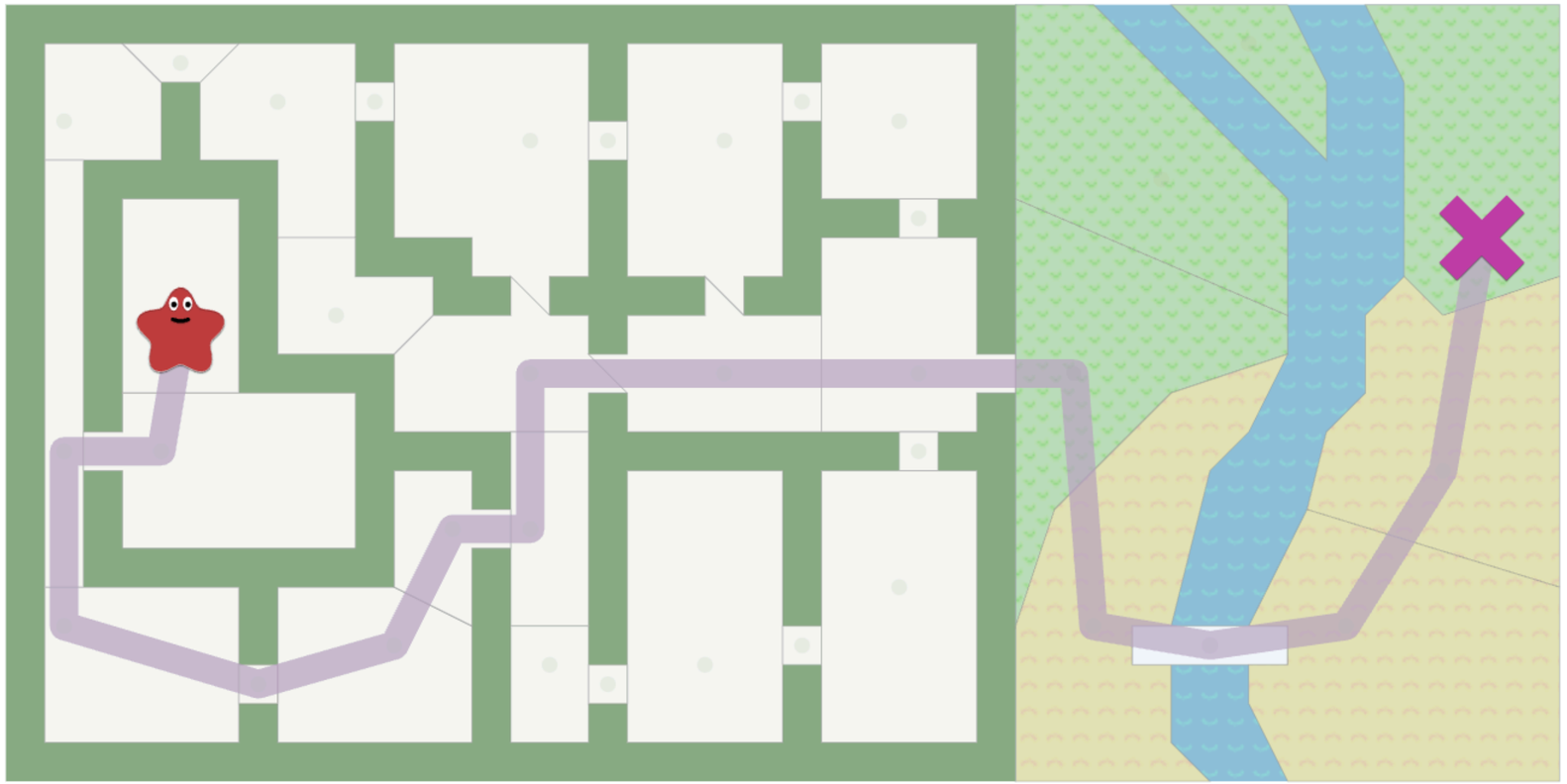


Giao diện trò chơi Sudoku

03

Path Finding

3.1 - Giới thiệu bài toán

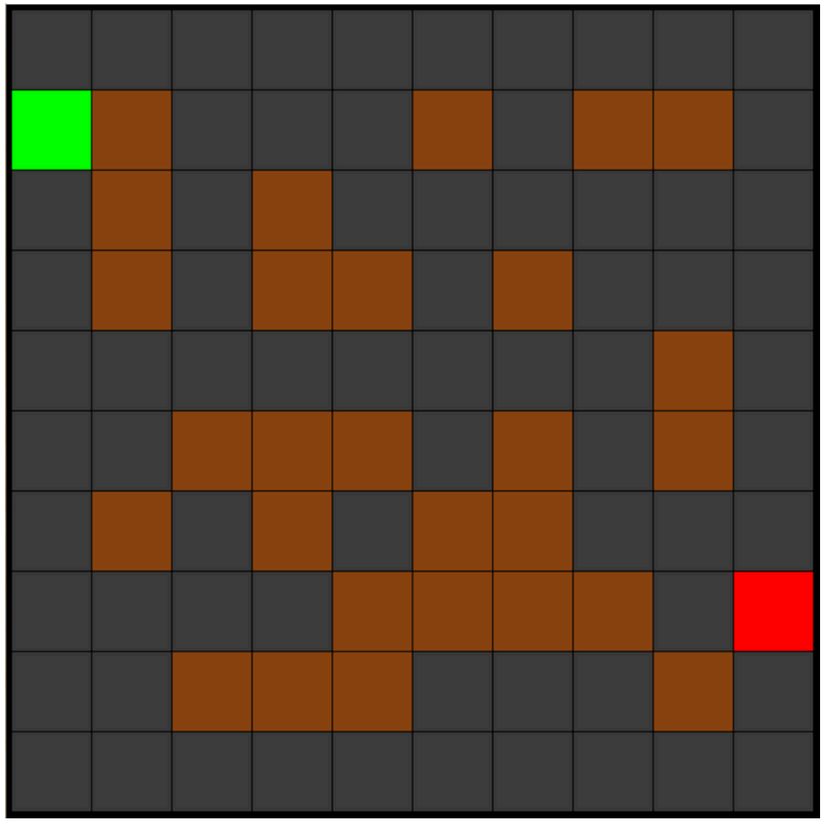


Bài toán tìm đường

Mô tả

Bài toán tìm đường là một trong những vấn đề cơ bản trong lập trình, trí tuệ nhân tạo và khoa học máy tính. Mục tiêu của nó là **tìm ra đường đi ngắn nhất hoặc có hiệu quả nhất (chi phí thấp nhất)** từ điểm xuất phát đến điểm đích trong một không gian nhất định. Thông thường, không gian này được biểu diễn dưới dạng một **lưới ô vuông** hoặc **đồ thị**.

3.2 - Biểu diễn trạng thái



Một đề bài của bài toán tìm đường

Không gian trạng thái

- Ở bài tập lớn lần này, nhóm em sử dụng **lưới ô vuông** (N hàng và N cột) để biểu diễn bài toán.
- Mỗi ô trong lưới sẽ đại diện cho một vị trí **có thể đi qua** hoặc **không thể đi qua** (vị trí của vật cản).

Định nghĩa trạng thái bài toán

Được biểu diễn là một vị trí (x,y), thoả mãn:

- Nằm trong không gian trạng thái: $0 \leqslant x < N$ và $0 \leqslant y < N$
- Không trùng với vị trí của vật cản

3.3 - Trạng thái khởi đầu và mục tiêu

Trạng thái khởi đầu

- Là một trạng thái (ngẫu nhiên hoặc được xác định cụ thể) đảm bảo nằm trong không gian trạng thái của bài toán và không trùng với vị trí của vật cản.

Kết quả mong muốn

- Là đường đi ngắn nhất từ trạng thái bắt đầu đến trạng thái mục tiêu.

Trạng thái mục tiêu

- Là một trạng thái (ngẫu nhiên hoặc được xác định cụ thể) đảm bảo nằm trong không gian trạng thái của bài toán và không trùng với vị trí của vật cản và không trùng với trạng thái bắt đầu.

3.4 - Luật di chuyển

Trạng thái của trò chơi thay đổi khi đối tượng thực hiện một lượt di chuyển. Ở bài toán này, nhóm quy định trong mỗi **một lượt** thì đối tượng chỉ có thể **di chuyển vào một ô liền kề** vị trí hiện tại **theo các hướng nhất định**.

1. Lên trên (đi vào ô phía trên)
2. Xuống dưới (đi vào ô phía dưới)
3. Sang trái (đi vào ô bên trái)
4. Sang phải (đi vào ô bên phải)

Lưu ý:

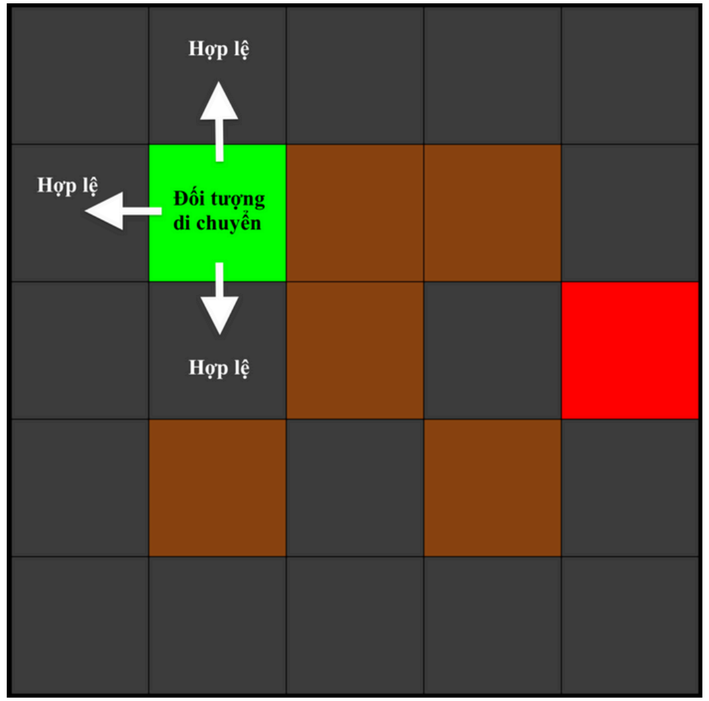
Để thoả mãn là một bước di chuyển hợp lệ, trạng thái mới của bước di chuyển đó phải đảm bảo:

- Theo đúng các hướng được quy định.
- Trạng thái mới nằm trong không gian trạng thái.
- Trạng thái mới không trùng với vị trí của vật cản.

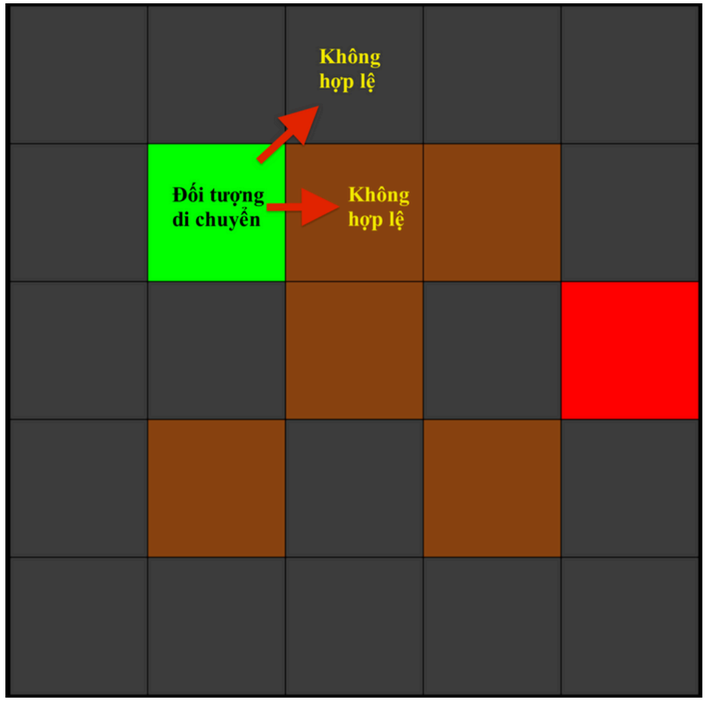
Nếu bước di chuyển là không hợp lệ, đối tượng sẽ không được thực hiện bước di chuyển đó.

3.3 - Luật di chuyển

Sau đây là ví dụ về một số bước di chuyển.



Các bước di chuyển hợp lệ
(Lên, Xuống, Sang trái)



Các bước di chuyển không hợp lệ
(Sang phải, Đường chéo)

3.5 - Giải thuật A*

Ở bài toán này, nhóm sử dụng **giải thuật A*** để tìm được đường đi ngắn nhất cho lời giải của bài toán.

Thuật toán A* kết hợp ưu điểm của cả thuật toán Dijkstra và Greedy Best-First Search. Nó sử dụng cả:

- **Chi phí thực tế** từ trạng thái bắt đầu đến trạng thái hiện tại.
- **Chi phí ước lượng** từ trạng thái hiện tại đến trạng thái mục tiêu (kết thúc).

Do đó, giải thuật A* vừa đảm bảo tìm ra **đường đi ngắn nhất** và vừa đảm bảo **sự hiệu quả**.

1

Mỗi một bước bước di chuyển sẽ có chi phí là 1 đơn vị. Do đó, chi phí thực tế là **tổng các bước di chuyển đến trạng thái hiện tại.**

2

Chi phí ước lượng **là giá trị ước lượng dựa vào kinh nghiệm (heuristic)**.

Một số công thức ước lượng:

- Khoảng cách **Manhattan**
- Khoảng cách **Euclidean**

3.6 - Giao diện trò chơi

```
Choose Auto mode? (y/n)
(y means randomize map, other key means read from input file): y
Auto mode activated.
Do you want write testcase generated and path into file (y/n)
(y means write into file, other key means do not write): y
Please enter the file name: TestCase1
Please enter map size (e.g., '5' for a 5x5 grid): 10
Mê cung được tạo sau 1 lần lặp.
Write map successfully into ./testcase/input/input_testcase1.text
Start: (0, 2)
End: (6, 8)
Evaluated nodes: 13
Path was written into ./testcase/output/output_testcase1.text
```

1

Khi khởi chạy trò chơi, người chơi được hỏi một số câu hỏi sau:

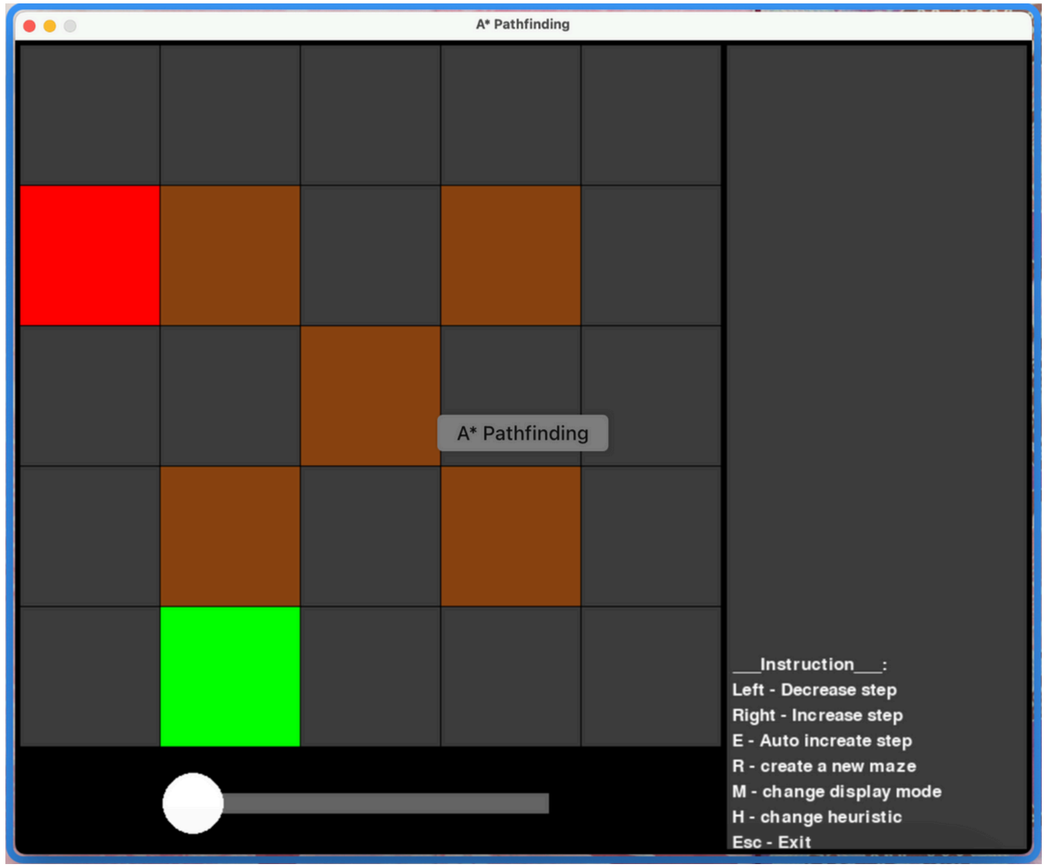
- Sử dụng chế độ tạo bài toán ngẫu nhiên hay sử dụng bài toán có sẵn?
- Có lưu bài toán lại hay không?
- Đặt tên cho bài toán để lưu lại?

Giao diện trò chơi Sudoku

3.6 - Giao diện trò chơi

2

- Giao diện bài toán là một lướt các ô vuông có kích thước (N * N)
- Trạng thái bắt đầu là vị trí của ô màu xanh.
- Trạng thái mục tiêu (kết thúc hoặc đích đến) là ô có vị trí màu đỏ.
- Các ô màu nâu đại diện cho vị trí có vật cản.
- Các ô còn lại đại diện cho các vị trí có thể di chuyển được.
- Giao diện có phần ‘Instruction’ để hướng dẫn thao tác với trò chơi.
- Có một thanh kéo để hỗ trợ việc di chuyển.

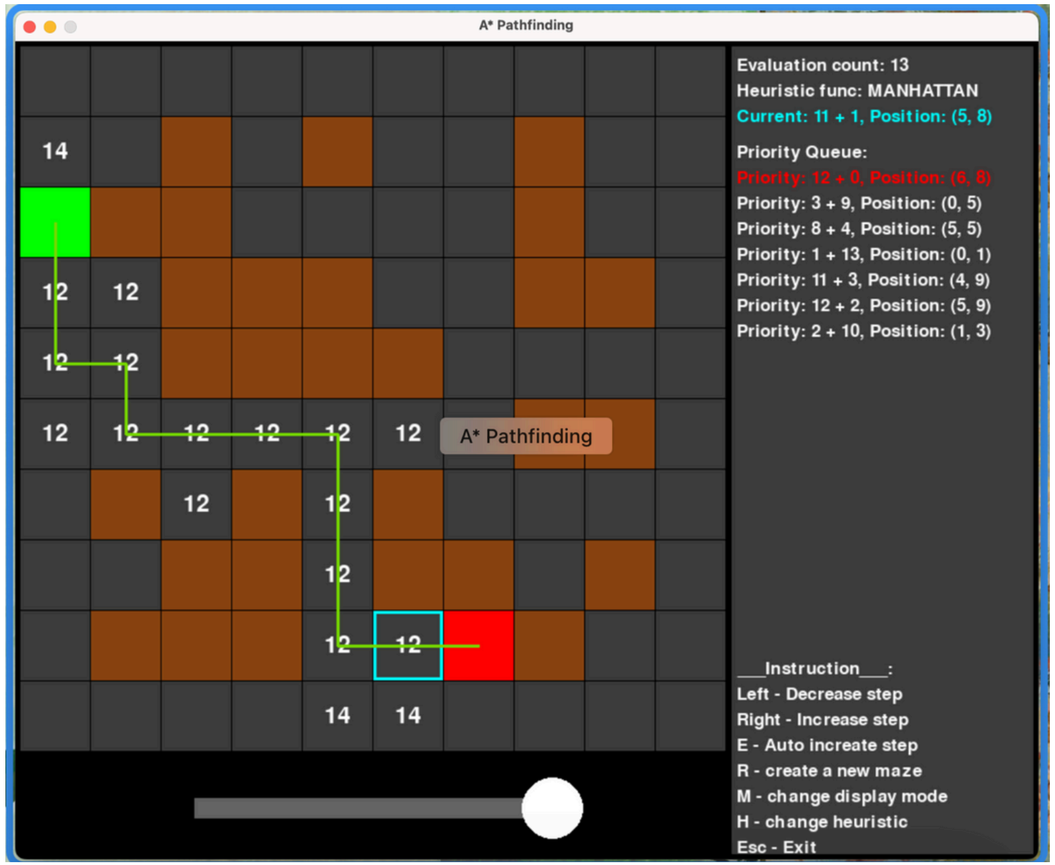


Giao diện trò chơi PathFinding
(Bắt đầu)

3.6 - Giao diện trò chơi

3

- Giao diện hiển thị đường đi ngắn nhất từ vị trí bắt đầu đến vị trí kết thúc bằng đường nối màu xanh.
- Các ô đã được lượng giá sẽ hiển thị giá trị lượng giá.
- Bên phải lưới ô vuông, giao diện hiển thị các thông tin cơ bản của thuật toán A*.



Giao diện trò chơi PathFinding
(Kết thúc)

THANK YOU FOR LISTENING