

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
KHOA CÔNG NGHỆ THÔNG TIN



# B Á O C Á O

ĐỒ ÁN MÔN HỌC: CƠ SỞ TRÍ TUỆ NHÂN TẠO  
CÁC THUẬT TOÁN TÌM KIẾM

Trợ giảng phụ trách : Nguyễn Duy Khánh (duykanhnguyen360@gmail.com)  
Nguyễn Ngọc Băng Tâm (bangtamnguyenn@gmail.com)

Sinh viên thực hiện : 20120128 - Nguyễn Thị Cẩm Lai  
20120547 - Võ Thành Phong

THÁNG 10/2022

## MỤC LỤC

<b>I. Tổng quan</b>	3
1. Thông tin nhóm	3
2. Mô tả về đề án	3
a) Bài toán đặt ra	3
b) Yêu cầu cụ thể của bài toán	3
c) Ngôn ngữ thực hiện	4
d) Tóm tắt các thuật toán đã sử dụng	4
3. Tự đánh giá	4
<b>II. Cơ sở lý thuyết</b>	5
A. Thuật toán tìm kiếm không có thông tin	5
1. Thuật toán BFS (Breadth First Search)	5
2. Thuật toán DFS (Depth First Search)	5
3. Thuật toán UCS (Uniform-Cost Search)	6
B. Thuật toán tìm kiếm có thông tin	7
1. Thuật toán tham lam (Greedy Best First Search)	7
2. Thuật toán A*	7
3. Giải thích các heuristic	8
<b>III. Cơ sở thực nghiệm</b>	9
1. Các bản đồ thực nghiệm	9
2. Yêu cầu kỹ thuật, giới thiệu chương trình, cách chạy và xem kết quả	9
3. Báo cáo trọng số và link video	10
a) Bản đồ không có điểm thưởng	10
b) Bản đồ có điểm thưởng	17
<b>IV. Tài liệu tham khảo</b>	20

# I. Tổng quan

## 1. Thông tin nhóm

Họ và tên	MSSV	Vai trò	Email
Võ Thành Phong	20120547	Nhóm trưởng	20120547@student.hcmus.edu.vn
Nguyễn Thị Cẩm Lai	20120128	Thành viên	20120128@student.hcmus.edu.vn

## 2. Mô tả về đề án

### a) Bài toán đặt ra

Cài đặt các thuật toán tìm kiếm đường đi từ vị trí xuất phát đến vị trí đích (vị trí thoát khỏi mê cung) cho tác nhân. Trong đó tác nhân chỉ có thể di chuyển lên, xuống, trái, phải với chi phí bằng nhau.

### b) Yêu cầu cụ thể của bài toán

Cài đặt các thuật toán sau:

- Thuật toán tìm kiếm không có thông tin:
  - Thuật toán tìm kiếm DFS (Depth First Search)
  - Thuật toán tìm kiếm BFS (Breadth First Search)
  - Thuật toán tìm kiếm UCS (Uniform-Cost Search)
- Thuật toán tìm kiếm có thông tin:
  - Thuật toán tìm kiếm tham lam (Greedy Best First Search)
  - Thuật toán tìm kiếm A\*

Đối với các thuật toán tìm kiếm có thông tin, tự định nghĩa hàm heuristic khác nhau (tối thiểu 2 hàm) và báo cáo kết quả.

Bản đồ trò chơi có 2 loại:

- Không có điểm thưởng: cài đặt 5 thuật toán trên để giải quyết. Tự thiết kế và báo cáo ít nhất 5 bản đồ, so sánh sự khác nhau giữa cách tìm đường đi với từng chiến lược cụ thể.

- Có điểm thưởng: tìm cách giải quyết và đề xuất chiến lược để tác nhân di chuyển sao cho chi phí đường đi là nhỏ nhất. Thiết kế bản đồ với tối thiểu 3 trường hợp: có 2, 5, 10 điểm thưởng. Nếu không tìm được lời giải tối ưu, hãy đề xuất chiến lược heuristic để giải quyết.

c) Ngôn ngữ thực hiện: python

d) Tóm tắt các thuật toán đã sử dụng

- Thuật toán tìm kiếm không có thông tin:
  - Thuật toán tìm kiếm DFS (Depth First Search)
  - Thuật toán tìm kiếm BFS (Breadth First Search)
  - Thuật toán tìm kiếm UCS (Uniform-Cost Search)
- Thuật toán tìm kiếm có thông tin:
  - Thuật toán tìm kiếm tham lam (Greedy Best First Search)
  - Thuật toán tìm kiếm A\*
- Đối với các thuật toán tìm kiếm có thông tin, có tự định nghĩa 3 hàm heuristic:
  - ManhattanDistance
  - EuclideanDistance
  - BreakingTie

### 3. Tự đánh giá

Tự đánh giá			Mức độ hoàn thành
<b>Bản đồ không có điểm thưởng</b>	Mức 1a	Thiết kế đầy đủ 5 bản đồ, và hoàn thành 3 thuật toán tìm kiếm không có thông tin (cài đặt thuật toán, báo cáo, hình vẽ minh họa) (Lai).	<b>100%</b>
	Mức 1b	Hoàn thành 2 thuật toán có thông tin (cài đặt thuật toán, báo cáo, hình vẽ minh họa) (Phong).	<b>100%</b>
<b>Bản đồ có điểm thưởng</b>	Mức 2a	Đề xuất thuật toán và các hàm heuristic phù hợp để giải quyết, thiết kế đủ 3 bản đồ và mô phỏng cách thuật toán chạy trên các bản đồ (Phong + Lai).	<b>100%</b>
	Mức 2b	Cài đặt thành công thuật toán (đảm bảo kết quả giống với mô phỏng ở mức 2a) (Phong).	<b>100%</b>

Điểm cộng	Kịch bản nâng cấp.	0%
	Video minh họa (Phong + Lai).	100%

## II. Cơ sở lý thuyết

### A. Thuật toán tìm kiếm không có thông tin

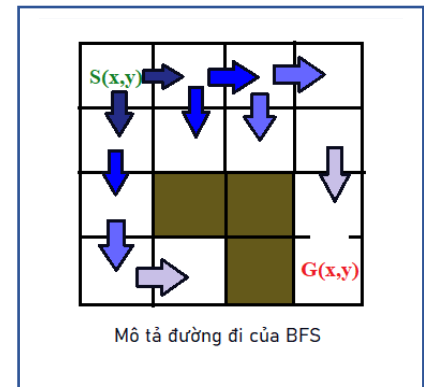
#### 1. Thuật toán BFS (Breadth First Search)

Breadth-First Search (Tìm kiếm theo chiều sâu) là một trong những thuật toán sử dụng chiến lược Tìm kiếm mù (Blind Search hay Uninformed Search) để tìm kiếm đường đi trong một không gian trạng thái cho trước.

Áp dụng với bài toán đặt ra, thuật toán BFS cho phép từ 1 ô nguồn loang rộng đến các ô kề với nó trong đồ thị dạng ma trận kích thước  $M * N$ . Khi trọng số mỗi cạnh nối - tương ứng với chi phí đi từ một ô đến ô kề nó đều bằng nhau (quy ước bằng 1) thì BFS cho ta đường đi có chi phí nhỏ nhất.

Mô tả thuật toán: Xuất phát từ một ô nguồn (S), di chuyển đồng thời tới tất cả các ô kề của nó (tương tự như vệt sóng lan, tất cả các nút ở độ sâu nhất định phải được mở trước khi mở các nút ở độ sâu tiếp theo) cho đến khi gặp ô đích (G) hoặc không còn ô nào để xét (đường đi tới G không tồn tại).

Cấu trúc dữ liệu cài đặt: Để thể hiện tính loang rộng của thuật toán, nhóm lựa chọn cài đặt BFS bằng deque (hàng đợi 2 đầu) để lưu trữ các ô kề đã được mở.

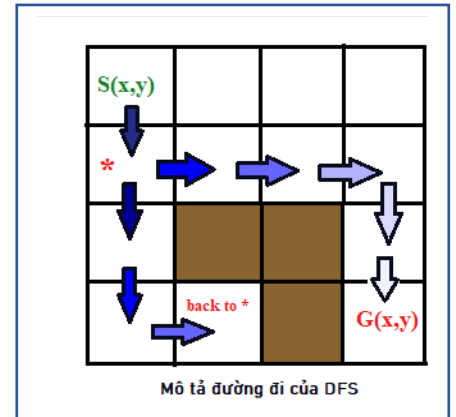


#### 2. Thuật toán DFS (Depth First Search)

Depth-First Search (Tìm kiếm theo chiều sâu) là một trong những thuật toán sử dụng chiến lược Tìm kiếm mù (Blind Search hay Uninformed Search) để tìm kiếm đường đi trong một không gian trạng thái cho trước.

Áp dụng đối với bài toán đặt ra, thuật toán DFS cho phép tìm kiếm đường đi từ một ô nguồn (S) đến các ô đích khác trong một đồ thị là ma trận kích thước  $M * N$ . Tuy nhiên không đảm bảo đường đi tìm được có chi phí nhỏ nhất.

Mô tả thuật toán: Xuất phát từ ô nguồn (S), ta chọn một hướng để bắt đầu đi và men theo nó (mở các nút sâu nhất trước) cho đến khi không thể đi được nữa. Lúc này ta sẽ quay trở lại theo hướng vừa tới và chọn đi theo một hướng khác. Tiếp tục men theo hướng đi này cho đến khi tới được ô đích (G) hoặc không thể đi được nữa thì lặp lại quá trình trên.



Cấu trúc dữ liệu cài đặt:

- Để thể hiện việc đi sâu vào một nhánh, nhóm lựa chọn cài đặt DFS bằng stack (ngăn xếp).
- Để tránh việc đi vào chu kỳ vô tận, mỗi ô trong bài toán chỉ được xét một lần duy nhất.

### 3. Thuật toán UCS (Uniform-Cost Search)

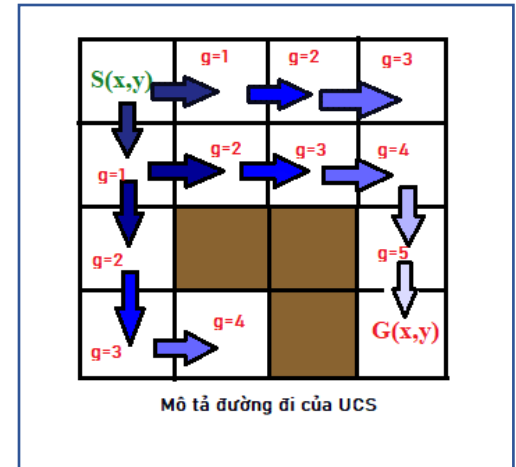
Uniform-Cost Search (Tìm kiếm chi phí thống nhất) là một trong những thuật toán sử dụng chiến lược Tìm kiếm mù (Blind Search hay Uninformed Search) để tìm kiếm đường đi trong một không gian trạng thái cho trước.

Áp dụng với bài toán đặt ra, thuật toán UCS cho phép tìm đường đi ngắn nhất từ ô nguồn (S) đến tất cả các ô khác trong đồ thị. Thuật toán tránh các đường đi có chi phí lớn, do đó thuật toán này đánh giá trạng thái hiện tại bằng hàm:  $f(n) = g(n)$  với  $f(n)$  là hàm đánh giá,  $g(n)$  là hàm chi phí đường đi từ nút nguồn (S) đến n và  $h(n)$  là hàm heuristic ước tính khoảng cách đến đích.



Mô tả thuật toán: Xuất phát từ ô nguồn (S), ta mở các ô lân cận và cập nhật chi phí từ ô xuất phát đến ô đó. Lần lượt chọn các đường đi ngắn nhất hiện tại để phát triển tiếp. Thực hiện cho đến khi gặp ô đích (G) hoặc không còn ô nào có thể đi.

Cấu trúc dữ liệu cài đặt: coi cấu trúc list là hàng đợi (tập biên) để tiện cho việc tìm nút có chi phí nhỏ nhất để mở và lấy nút đó ra khỏi hàng đợi.



## B. Thuật toán tìm kiếm có thông tin

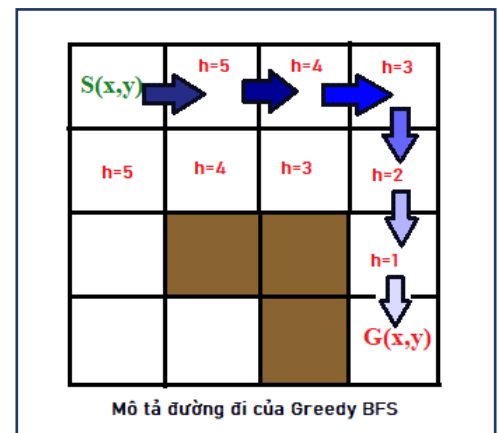
### 1. Thuật toán tham lam (Greedy Best First Search)

Greedy Best-First Search (Tìm kiếm tối ưu kiểu tham lam) là một trong những thuật toán sử dụng chiến lược Tìm kiếm có thông tin (Informed Search) để tìm kiếm đường đi trong một không gian trạng thái cho trước.

Áp dụng với bài toán đặt ra, thuật toán Greedy BFS cố gắng mở các nút được ước lượng gần với đích nhất. Do đó thuật toán này đánh giá trạng thái hiện tại chỉ dựa trên heuristic:  $f(n)=h(n)$  với  $f(n)$  là hàm đánh giá và  $h(n)$  là hàm heuristic.

Mô tả thuật toán: Xuất phát từ ô nguồn (S), tính chi phí heuristic của tất cả các ô kề với nó (chỉ duyệt đến ô kề có chi phí heuristic tốt nhất). Lặp lại quá trình trên cho đến khi gặp ô đích (G) hoặc không còn ô nào để xét (đường đi tới G không tồn tại).

Cấu trúc dữ liệu cài đặt: coi cấu trúc list là hàng đợi (tập biên) để tiện cho việc tìm nút có chi phí nhỏ nhất để mở và lấy nút đó ra khỏi hàng đợi.



### 2. Thuật toán A\*

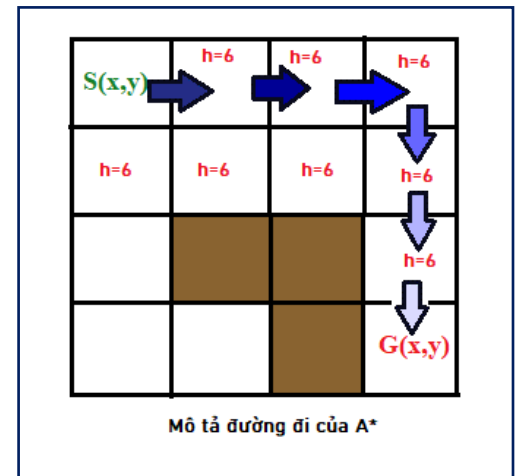
A\* là một trong những thuật toán sử dụng chiến lược Tìm kiếm có thông tin (Informed Search) để tìm kiếm trong một không gian trạng thái cho

trước. Thuật toán này là sự kết hợp giữa thuật toán Dijkstra và Greedy BFS để tối ưu hóa việc tìm kiếm đường đi.

Áp dụng với bài toán đặt ra, thuật toán A\* cũng cho phép từ một ô mở rộng đến ô gần đích nhất với hy vọng việc mở rộng như vậy sẽ dẫn đến lời giải một cách nhanh nhất. Thuật toán tích hợp heuristic và quá trình tìm kiếm, tránh các đường đi có chi phí lớn. Do đó thuật toán này đánh giá trạng thái hiện tại bằng hàm :  $f(n) = g(n) + h(n)$  với  $f(n)$  là hàm đánh giá,  $g(n)$  là hàm chi phí đường đi từ nút gốc đến n và  $h(n)$  là hàm heuristic ước tính khoảng cách đến đích.

Mô tả thuật toán: Xuất phát từ ô nguồn (S), tính chi phí heuristic của tất cả các ô kề với nó (chỉ duyệt đến ô kề có chi phí heuristic tốt nhất). Lặp lại quá trình trên cho đến khi gặp ô đích (G) hoặc không còn ô nào để xét (đường đi tới G không tồn tại).

Cấu trúc dữ liệu cài đặt: coi cấu trúc list là hàng đợi (tập biên) để tiện cho việc tìm nút có chi phí nhỏ nhất để mở và lấy nút đó ra khỏi hàng đợi.



### 3. Giải thích các heuristic

- ManhattanDistance:

- + Heuristic được dùng phổ biến cho đồ thị dạng lưới ô vuông.
- + Tồn tại một tham số D là một hyper parameter để tối ưu heuristic.
- + Chọn D là trọng số nhỏ nhất trong 4 hướng đi từ 1 ô.

+ Công thức:

function heuristic(node) =

dx = abs(node.x - goal.x)

dy = abs(node.y - goal.y)

return D \* (dx + dy)

- EuclideanDistance:

+ Công thức:



```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * sqrt(dx * dx + dy * dy)
```

+ Hệ số D vẫn được chọn là trọng số nhỏ nhất trong các trọng số của các hướng đi. Nếu bản đồ không có vật cản thì có xu hướng đi thẳng theo đường chéo đến đích.

- BreakingTie:

+ Công thức:

```
dx1 = current.x - goal.x
dy1 = current.y - goal.y
dx2 = start.x - goal.x
dy2 = start.y - goal.y
cross = abs(dx1 * dy2 - dx2 * dy1)
heuristic = cross * p
```

+ Hệ số p có thể chọn bằng (chi phí tối thiểu khi thực hiện một bước)/(độ dài đường đi tối đa dự kiến). Và việc tính cross dùng để kiểm tra độ thẳng hàng của điểm current so với start và goal. Khi chúng càng không thẳng hàng thì cross càng lớn.

### III. Cơ sở thực nghiệm

#### 1. Các bản đồ thực nghiệm

- Các bản đồ không có điểm thưởng:

- + input1.txt: Kích thước nhỏ, có tường và tồn tại đường đi.
- + input2.txt, input3.txt, input4.txt: Kích thước lớn, có tường, và tồn tại đường đi.
- + input5.txt: Không tồn tại đường đi.
- + input6.txt: Không có tường và tồn tại đường đi.

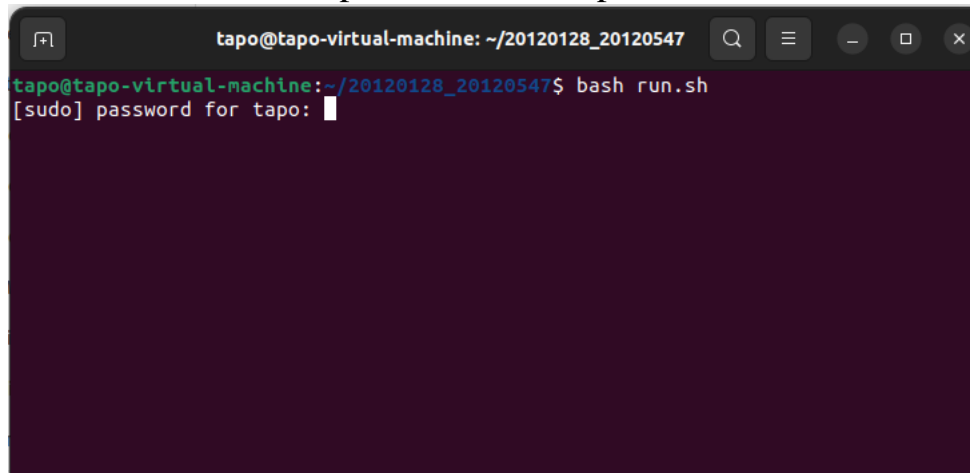
- Các bản đồ có điểm thưởng:

- + input1.txt: 2 điểm thưởng, có tường và tồn tại đường đi.
- + input2.txt: 5 điểm thưởng, có tường và tồn tại đường đi.
- + input3.txt: 10 điểm thưởng, có tường và tồn tại đường đi.

#### 2. Yêu cầu kĩ thuật, giới thiệu chương trình, cách chạy và xem kết quả

- Có cài đặt python phiên bản 3. trở về sau trong máy. Phiên bản cụ thể nhóm sử dụng là Python 3.10.0.

- Thư mục bài nộp gồm file run.sh, folder input chứa các bản đồ để thực hiện việc tìm đường đi, folder source chứa mã nguồn viết bằng ngôn ngữ Python3.
- Chạy file run.sh bằng dòng lệnh command **bash run.sh** trên terminal của các máy có hệ điều hành thuộc họ Unix hoặc trên power shell của các hệ thống chẳng hạn như WSL có hỗ trợ cho lập trình bash.
- **Lưu ý:** trong file run.sh có chứa **các câu lệnh cài đặt các python library** cần thiết nên quá trình thực thi file run.sh có thể tốn thời gian do quá trình cài đặt diễn ra.
- **Lưu ý:** mở terminal ở thư mục MSSV1\_MSSV2 để chạy được lệnh bash run.sh, file run.sh có chứa các câu lệnh cài đặt các library bằng lệnh sudo nên cần nhập mật khẩu để tiếp tục.



```
tapo@tapo-virtual-machine: ~/20120128_20120547
tapo@tapo-virtual-machine:~/20120128_20120547$ bash run.sh
[sudo] password for tapo: 
```

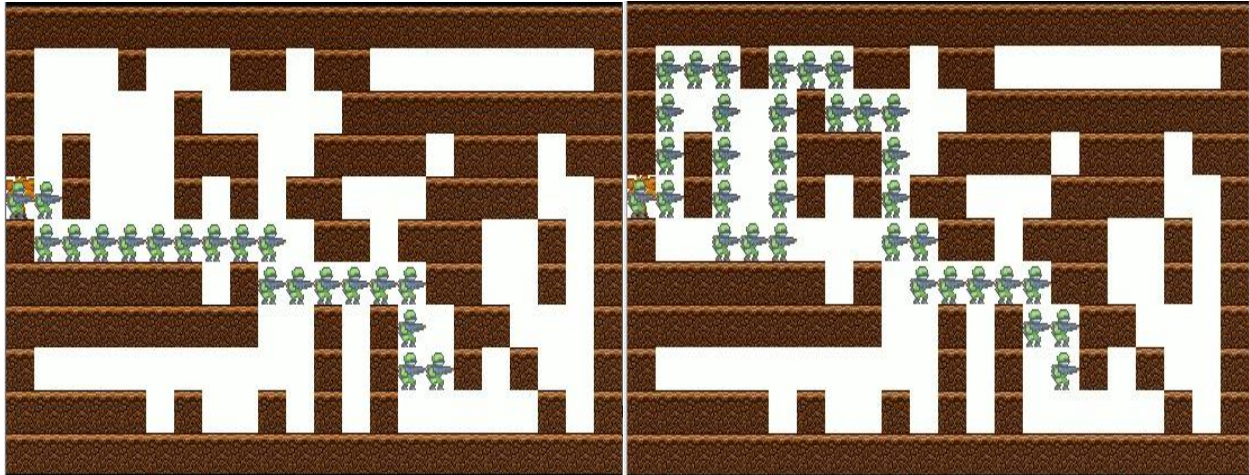
- Sau khi thực thi file run.sh, chương trình tạo ra folder output cùng cấp với folder input chứa kết quả của các thuật toán trong việc tìm đường đi trong các mê cung khác nhau. Folder có cấu trúc như yêu cầu trong đề án, trong đó các **folder con được đặt tên là tên các thuật toán** chứa 2 file: **file .txt** là chi phí tìm đường đi thoát khỏi mê cung và **file .mp4** là video lưu lại quá trình tìm đường đi.

### 3. Báo cáo trọng số và link video

*Số lượng bản đồ kết hợp với số lượng thuật toán tạo ra số bộ thực nghiệm lớn, do đó trong phần báo cáo sẽ chỉ trình bày một số bản đồ đại diện.*

#### a) Bản đồ không có điểm thưởng

- **Thực nghiệm 1:**



Map input1.txt: thuật toán bfs(bên trái), thuật toán dfs(bên phải).

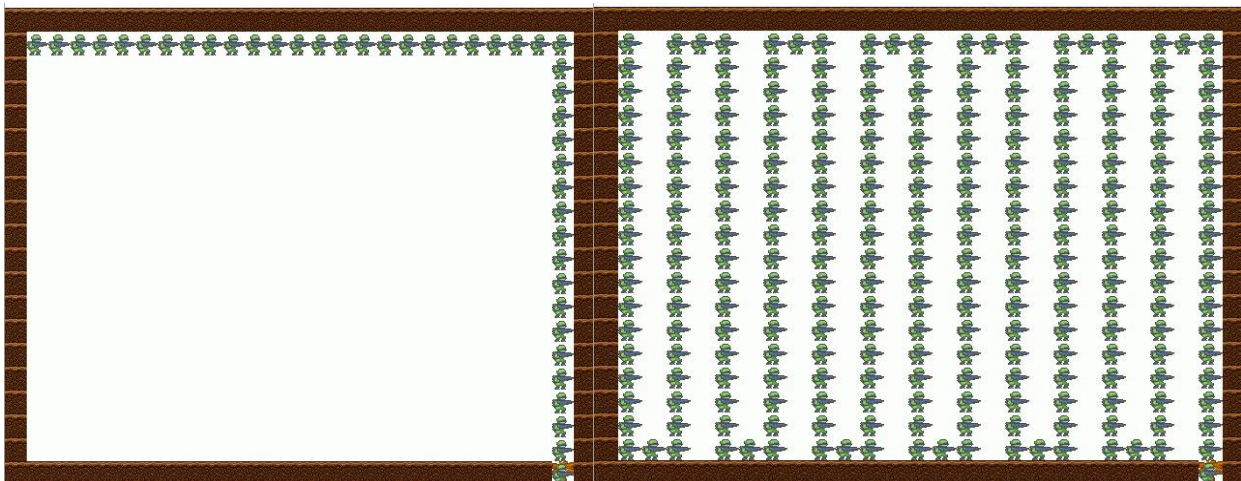
- Thực nghiệm trên cho thấy sự khác biệt về cách tìm đường đi giữa 2 thuật toán bfs và dfs:

+ Chi phí khi tìm theo bfs là: 19 và chi phí tìm theo dfs là: 33.

+ BFS chú trọng mở rộng theo chiều rộng trong khi DFS lại chú trọng mở rộng theo chiều sâu.

+ Chính vì điểm khác biệt ở cách lựa chọn nút để mở, BFS gần như phải duyệt qua toàn bộ ma trận để tìm được đường đi tới ô đích do xét hết tất cả các nút lân cận với nó trước, còn dfs có thể tìm ra một đường đi tới đích nhanh chóng hơn bfs nhưng đường đi thường dài và không tối ưu tốt bằng bfs.

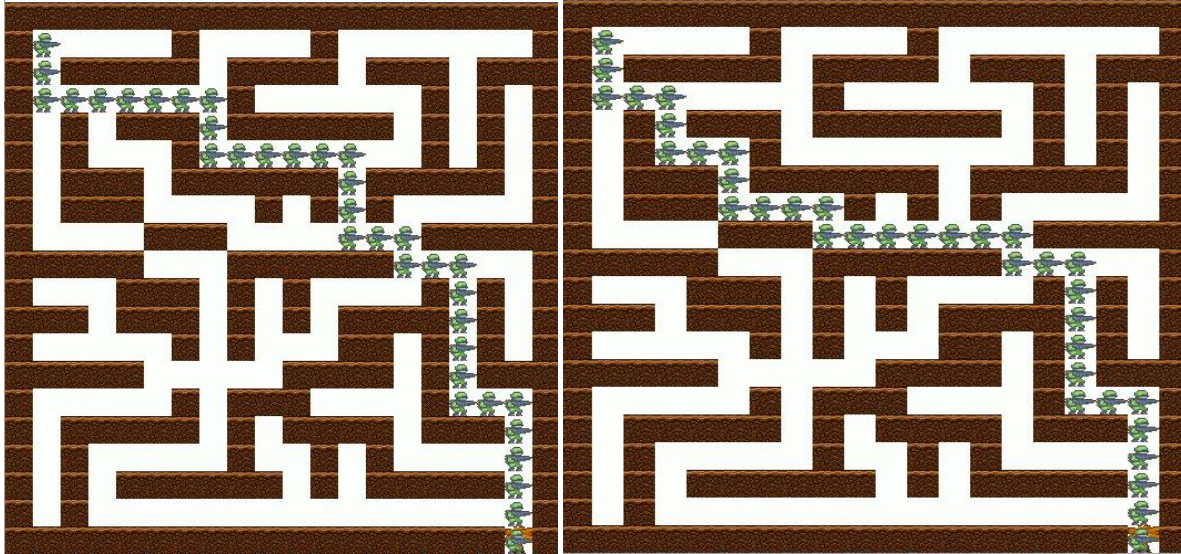
- **Thực nghiệm 2:**



Map input6.txt: thuật toán bfs(bên trái), thuật toán dsf(bên phải).

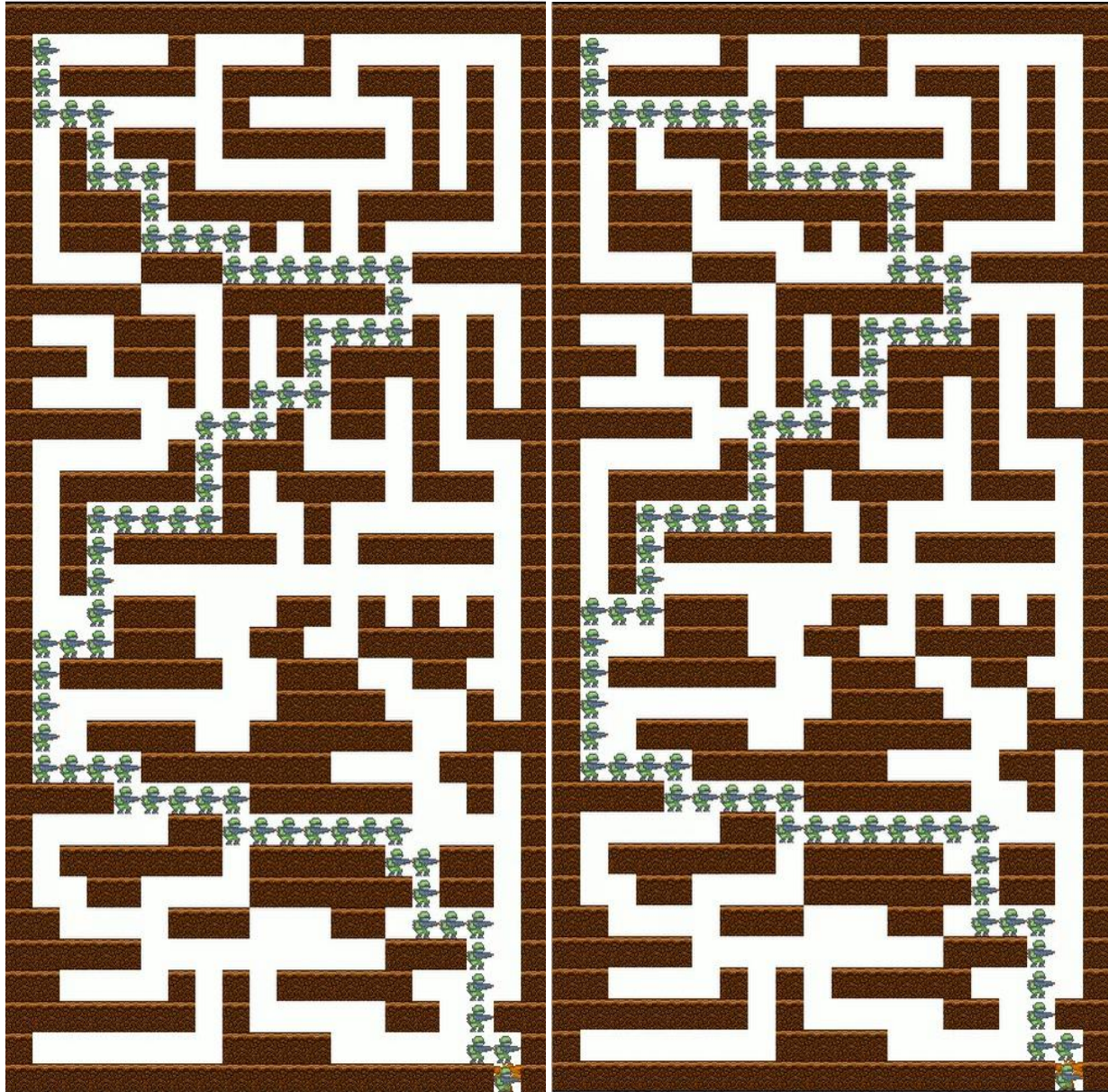


- Chi phí khi tìm theo bfs là: 42 và chi phí tìm theo dfs là: 246.
- Với bản đồ không có tường thì sự tối ưu càng thể hiện rõ hơn giữa bfs và dfs, nhưng tốc độ tìm ra đường đi đến đích của dfs luôn nhanh hơn, nó gần như tìm ra một đường đi hợp lệ tới đích sau vài lần thử đầu tiên. Tuy nhiên, đường đi thu được lại là một đường đi rất dài.
- **Thực nghiệm 3:**



Map input3.txt: thuật toán bfs(bên trái), thuật toán ucs(bên phải).

- So sánh giữa bfs và ucs:
  - + Chi phí khi tìm theo bfs là: 35 và chi phí tìm theo ucs là: 35.
  - + Mặc dù ucs thích hợp làm việc với các bản đồ có trọng số, còn bfs thì không nhất thiết bản đồ phải có trọng số nhưng kết quả giống nhau là do ucs dựa vào trọng số để quyết định nước đi tiếp theo, tuy nhiên trọng số của 4 hướng di chuyển đều bằng 1, do đó quyết định nút mở theo cấu trúc lưu trữ là hàng đợi, nên cách lựa chọn đường đi có phần giống bfs. Tất nhiên về tính tối ưu cho chi phí của đường đi, cũng giống như bfs, ucs tối ưu về chi phí hơn dfs nhưng cũng sẽ có thời gian tìm ra đường đi đến đích chậm hơn dfs.
- **Thực nghiệm 4:**



Map input4.txt: thuật toán ucs(trái), thuật toán greedy best first search (phải)

- So sánh giữa các thuật toán sử dụng cho các bản đồ có trọng số là UCS và Greedy Best First Search(GBFS):

+ Chi phí theo GBFS (Heuristic: EuclideanDistance): 1575.2046.

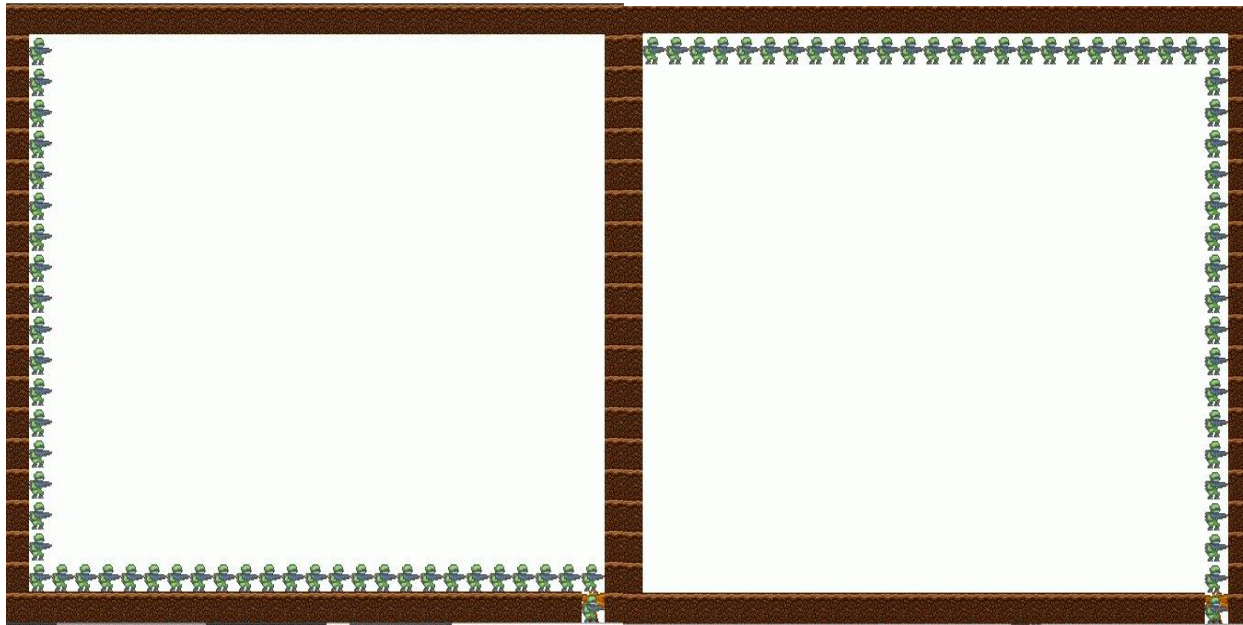
+ Chi phí theo UCS: 76.

+ Sự khác biệt chi phí là do ucs sử dụng trọng số giữa các hướng đi để quyết định nước đi tiếp theo, GBFS lại sử dụng ‘kinh nghiệm’ là heuristic về khoảng cách euclidean giữa tọa độ các ô.



+ Nhìn vào thực nghiệm: ucs có vẻ chiếm ưu thế hơn cả về mặt độ dài đường đi và thời gian tìm kiếm. Nguyên nhân là do GBFS sử dụng các heuristic về khoảng cách trong không gian sẽ ưu tiên việc đi đến ô nào mà có khoảng cách với đích ngắn nhất, do đó khi gặp vật cản GBFS sẽ không hoạt động tốt như ucs do ‘kinh nghiệm’ của nó chưa có cách tính chi phí vượt qua vật cản tối ưu, nếu không may mắn tìm ra đường đi vượt qua vật cản gần đó, có thể sẽ tốn thêm thời gian và chi phí để có thể vượt qua vật cản.

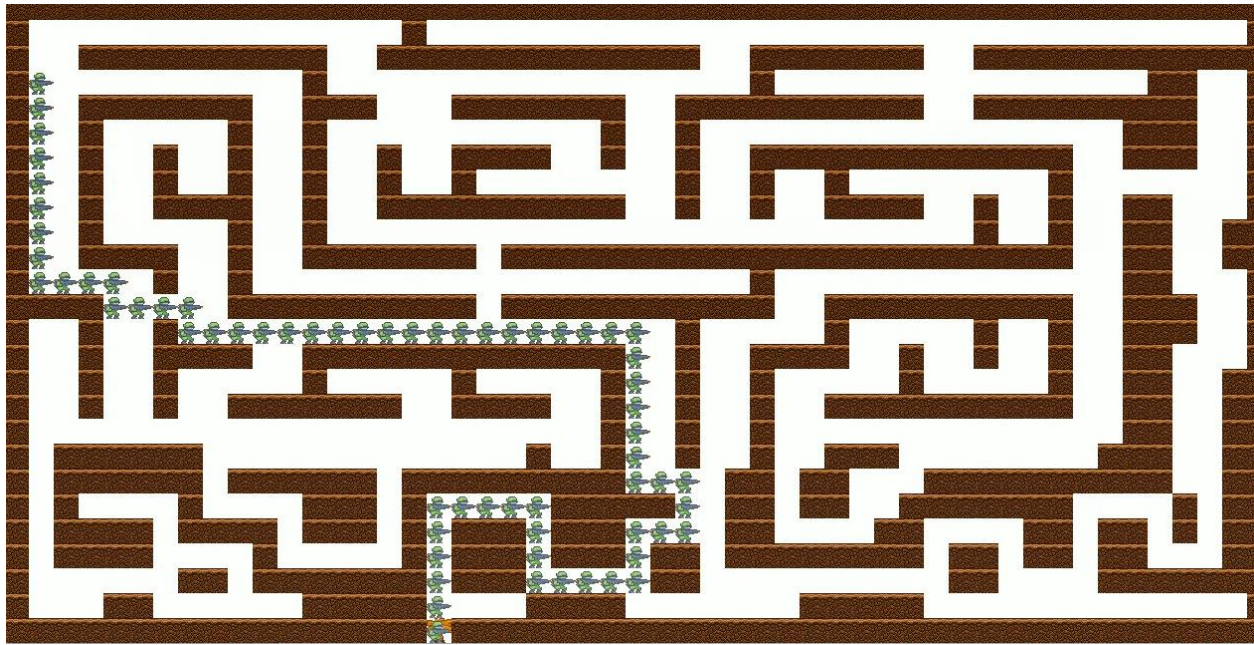
### - Thực nghiệm 5:



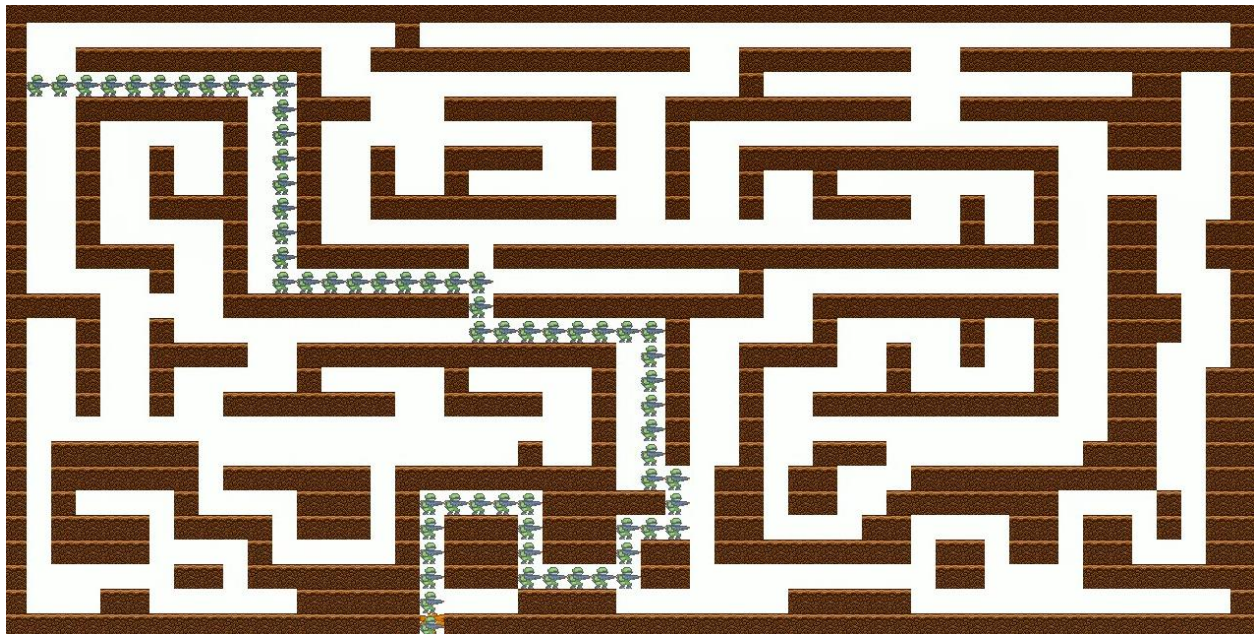
Map input6.txt: thuật toán ucs(trái), thuật toán A\*, heuristic:breakingtie (phải).

- Chi phí cả 2 thuật toán đều là 42. Do việc A\* tìm đường đi tiếp theo dựa vào cả trọng số và heuristic nhưng không cộng dồn heuristic qua các nước đi mà chỉ xét heuristic của nước đi kế tiếp, do đó heuristic ở điểm đích nếu tính dựa vào tọa độ của 2 điểm trong không gian sẽ luôn là 0. Nên chi phí đường đi là bằng nhau với những giá trị trọng số không đổi.
- Tuy nhiên UCS vẫn có xu hướng mở tất cả các nút để tìm đường đi đến đích, còn A\* mở theo hướng chính đến đích và tối ưu việc chọn đường đi bằng heuristic kèm theo nên thời gian tìm kiếm của A\* có thể nhanh hơn UCS.



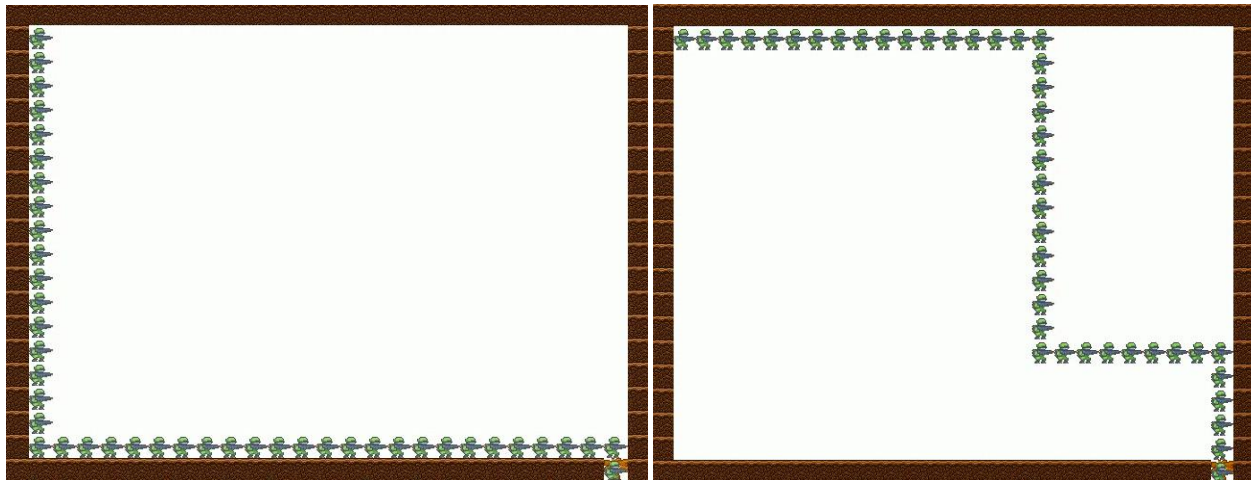


Áp dụng thuật toán UCS trên Map input2.txt, chi phí 64.

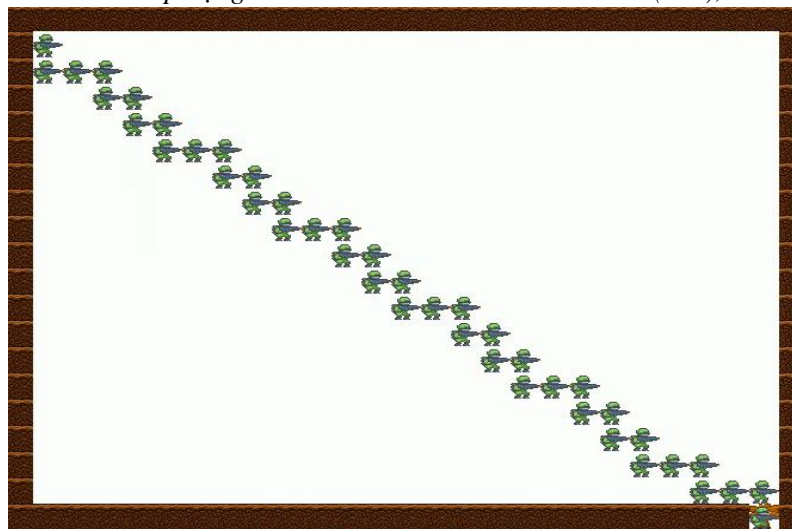


Áp dụng thuật toán A\*, heuristic: BreakingTie trên Map input2.txt, chi phí 64.

- **Thực nghiệm 6:** So sánh giữa các heuristic.



Thuật toán GBFS áp dụng các heuristic: *ManhattanDistance*(trái), *EuclideanDistance*(phải).



Thuật toán GBFS áp dụng heuristic *BreakingTie*.

- *ManhattanDistance* thông dụng cho các bản đồ dạng lưới ô vuông.
- *EuclideanDistance* thường thích hợp với những bản đồ ít chướng ngại vật và cho phép di chuyển theo đường chéo nhiều hơn, do khi đó khoảng cách Euclidean được tính theo đường thẳng nối 2 tọa độ (hiệu quả khi không có vật cản ở giữa). Với những bản đồ có quá nhiều chướng ngại vật thì *EuclideanDistance* sẽ chậm hơn so với *ManhattanDistance* do quy mô giữa hàm heuristic là khoảng cách Euclidean với quy mô của trọng số sẽ rất cách biệt, làm đường đi được mở rất vòng vo, ngoài ra cần sử dụng căn bậc hai trong tính khoảng cách Euclidean cũng nhằm mục đích giảm quy mô của heuristic.
- *BreakingTie*: Đối với những địa hình bằng phẳng, sẽ tồn tại trường hợp có nhiều đường đi bằng nhau cả về trọng số và heuristic cùng lúc

nếu sử dụng các heuristic tính khoảng cách trong không gian. Thuật toán phải khám phá hết tất cả những đường đi đó và lựa chọn lại đường đi phù hợp. Một cách giúp giải quyết vấn đề đó là thêm một hệ số cross vào heuristic để quyết định xem nước đi nào là gần nhất với đường đi thẳng từ nguồn đến đích. Thêm nữa ưu điểm của BreakingTie là khi gặp vật cản vẫn phải tìm đường vượt qua vật cản như những heuristic khác, nhưng khi đã vượt qua được sẽ tiến những bước tiếp theo rất nhanh ở những địa hình bằng phẳng.

#### b) Bản đồ có điểm thưởng

- **Đề xuất thuật toán giải quyết:** Áp dụng thuật toán Simulated Annealing, kết hợp UCS.

- Lý thuyết:

+ Simulated Annealing (SA) giúp tìm ra lời giải có thể được xem là tốt trong nhiều lời giải có thể có. Hay tìm ra lời giải tối ưu toàn cục (tối ưu nhất) giữa những lời giải tối ưu cục bộ.

+ Đầu tiên xem những ô điểm thưởng như là những điểm đón mà chưa quan tâm đến giá trị điểm thưởng mà những ô đó chứa. Đưa bài toán về ý tưởng của bài toán Người giao hàng (Traveling Salesman): đi qua tất cả các điểm đó với một chi phí tối ưu nhất. Sau khi đã có đường đi tối ưu nhất, sẽ tính những điểm thưởng vào tổng chi phí đường đi từ nguồn qua các điểm thưởng và về đích. Tất nhiên, nếu có một ô điểm thưởng đi qua nhiều hơn 1 lần trong quá trình tìm đường đi tối ưu thì chỉ tính một lần điểm thưởng. Cuối cùng nếu chi phí của chiến lược vừa nêu lớn hơn chi phí của việc dùng UCS với cải tiến là ăn những điểm thưởng có thể ăn gần với đường đi UCS thông thường thì ta sẽ chọn phương án sau.

+ Tư tưởng truyền thống của thuật toán SA:

\* Lấy ý tưởng từ quá trình luyện thép trong vật lý, thuật toán bắt đầu từ nhiệt độ  $T_0$  rất lớn. Tại thời điểm bắt đầu này, cấu trúc các phân tử còn khá rời rạc tương tự như lời giải rất chưa tối ưu.

\* Quá trình luyện thép bắt đầu và nhiệt độ được giảm từ từ. Ở mỗi lần nhiệt độ giảm, ta sinh ra ngẫu nhiên một lời giải mới từ lời giải đã có. Nếu lời giải mới có chi phí đường đi thấp hơn lời giải tốt nhất hiện tại ta chấp nhận lời giải mới này. Ngược lại, vẫn chấp nhận lời giải tồi với xác suất tỉ lệ nghịch với nhiệt độ và tuân theo công thức Boltzman. Việc chấp nhận này giúp vượt qua một lời giải tối ưu cục bộ. Xác suất chấp nhận tỉ lệ nghịch với nhiệt độ thì



càng về sau, một lời giải không tốt hơn sẽ càng khó được chấp nhận hơn. Quá trình luyện thép sẽ kết thúc khi đạt điều kiện dừng (có thể là nhiệt độ giới hạn, ....).

- Cài đặt: Điều quan trọng của việc cài đặt là tìm cách xác định một trạng thái lân cận trạng thái hiện có một cách hợp lí sau mỗi lần T thay đổi để tìm lời giải mới. Khác với ý tưởng ‘làm lạnh’ truyền thống trong đồ án này, nhóm cài đặt hàm tìm trạng thái tiếp theo như sau: Đối với một tập đường đi là tập các đỉnh thì tập các trạng thái có thể có sẽ là hoán vị của tập đỉnh đó, do đó:

+ Thay vì cho T là một số (nhiệt độ) lớn rồi sau đó giảm dần để tạo thành số lần lặp, thì đối với những bản đồ có số điểm thưởng nhỏ (chẳng hạn nhỏ hơn 6) thì T ban đầu bằng 0, và dùng một list chứa tất cả các hoán vị của tập tọa độ các điểm thưởng, hàm next\_path sẽ trả về trạng thái tiếp theo bằng cách duyệt qua hết list hoán vị nêu trên. Ở mỗi lần lặp sẽ tăng T lên 1 và dùng T làm chỉ số truy cập các phần tử trong list, khi T đã vượt ra khỏi độ dài list thì dừng. Trường hợp này sẽ chắc chắn tìm được trạng thái tốt nhất đối với số lượng điểm thưởng nhỏ, sinh hoán vị không là số quá lớn.

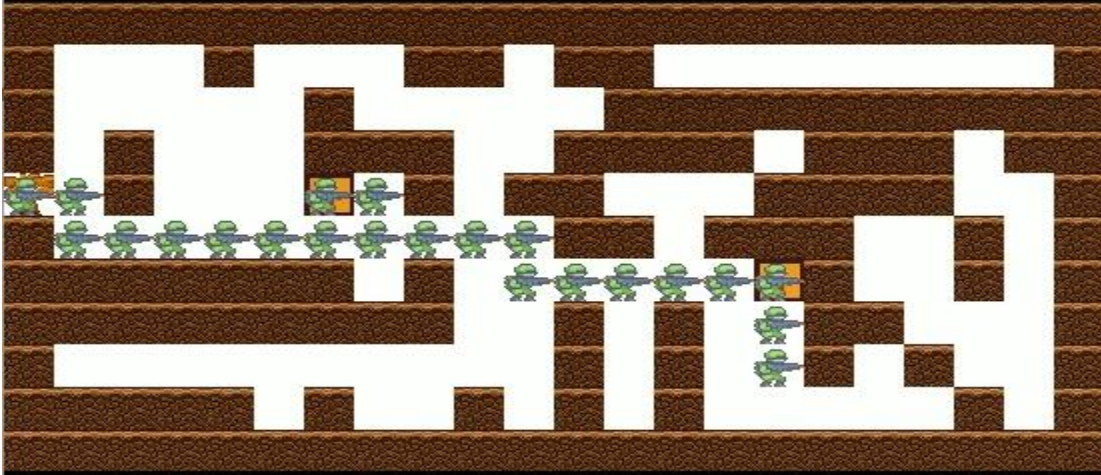
+ Đối với các trường hợp có số điểm thưởng lớn (chẳng hạn lớn hơn 6: giả sử trường hợp là 10 điểm thưởng thì  $10! = 3628800$  trường hợp) yêu cầu trong đồ án là mỗi thuật toán **không chạy quá 10s**, cho nên chiến lược tạo trạng thái mới bằng cách sinh hoán vị không khả thi cho các trường hợp này, do đó quay về quy trình ‘làm lạnh’ truyền thống và hy vọng việc sinh ngẫu nhiên trạng thái mới đạt được trạng thái tốt nhất. Ở mỗi lần lặp nhiệt độ (T) giảm 1 độ và xác suất chấp nhận lời giải tồi tuân theo phân phối Boltzman  $e^{\frac{-\delta}{T}}$  với  $\delta$  là độ chênh lệch giữa đường đi hiện tại và đường đi tốt nhất hiện có, T là nhiệt độ hiện tại.

+ Dùng UCS xác định đường đi ngắn nhất giữa các cặp điểm thưởng mà chúng ta quan tâm.

+ Cuối cùng kiểm tra với chi phí đường đi ucs cải tiến.

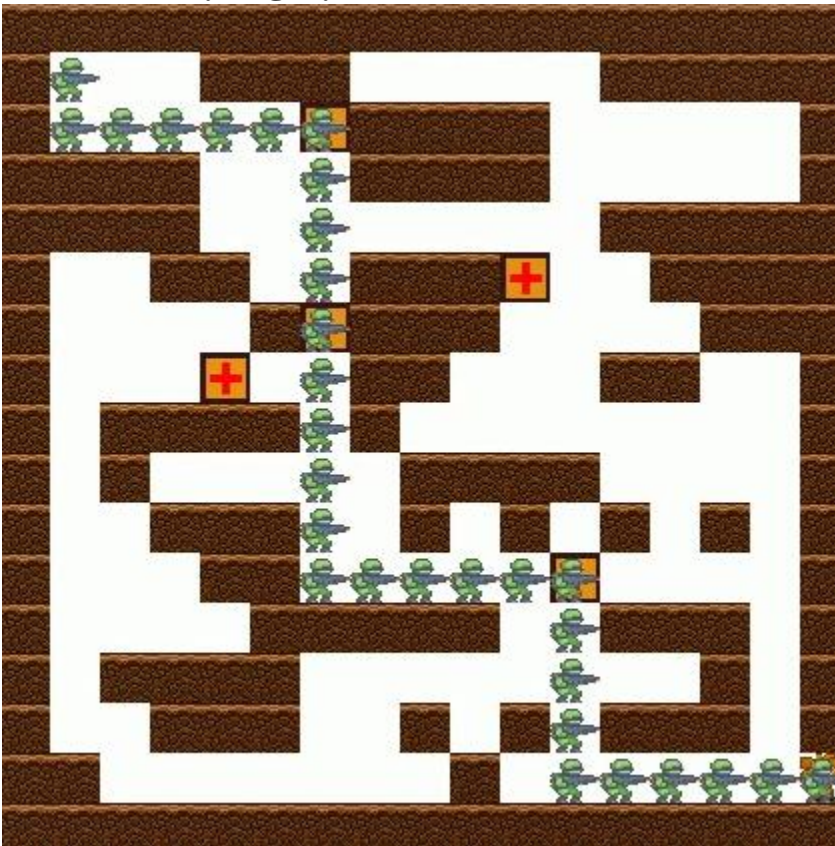
- Thực nghiệm:

+ **Thực nghiệm 1:**



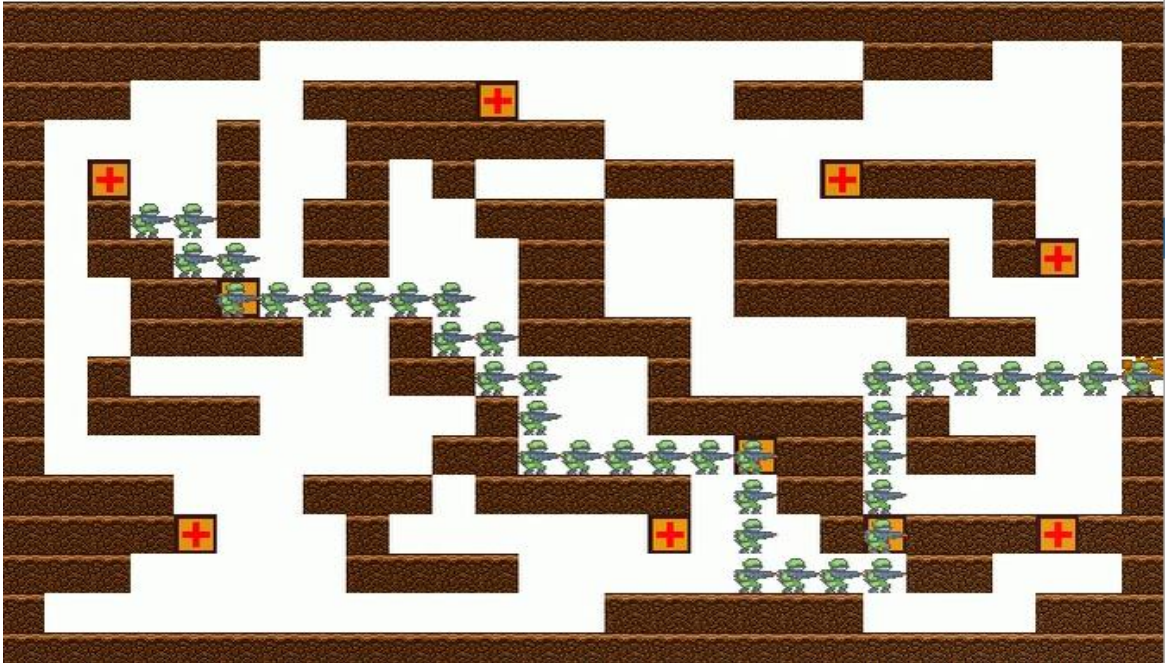
*Map có 2 điểm thưởng, chi phí là -4.*

## + Thực nghiệm 2:



Map có 5 điểm thưởng, chi phí là 8.

### + Thực nghiệm 3:



*Map có 10 điểm thưởng, chi phí 12.*

#### IV. Tài liệu tham khảo

- Hình ảnh sử dụng cho đồ án: <https://github.com/russs123/Shooter/tree/main/img>
- Heuristic: <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>
- Một số nguồn tham khảo khác:

<https://github.com/nbtam99/hcmus-path-finding>

<https://labs.flinters.vn/machine-learning/tim-kiem-trong-khong-gian-trang-thai-va-thuat-toan-simulated-annealing/>