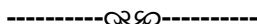




# **Đại học Khoa học Tự nhiên – ĐHQG TP.HCM**



## **Môn học**

Cơ sở trí tuệ nhân tạo

## **Báo cáo đồ án**

Project 1 – Ares's adventure

## **Giảng viên phụ trách:**

Tiến Sĩ Bùi Duy Đăng

Thầy Huỳnh Lâm Hải Đăng

Thầy Lê Nhựt Nam

TP Hồ Chí Minh, tháng 3 năm 2025

## Contents

<b>I. THÔNG TIN CHUNG:</b>	4
1. Thông tin thành viên:	4
2. Thông tin đồ án:	4
<b>II. PHÂN CHIA CÔNG VIỆC:</b>	5
<b>III. CHI TIẾT ĐỒ ÁN:</b>	5
1. Tổng quan về map (Testcase):	5
2. Một số lưu ý:	6
3. Thuật toán BFS:	7
a) Giới thiệu:	7
b) Chi tiết:	7
4. Thuật toán DFS:	7
a) Giới thiệu:	7
b) Chi tiết:	8
5. Thuật toán UCS:	8
a) Giới thiệu:	8
b) Chi tiết:	9
6. Thuật toán A Star ( $A^*$ ):	9
a) Giới thiệu:	9
b) Chi tiết:	9
7. Thuật toán Greedy BFS:	10
a) Giới thiệu:	10
b) Chi tiết:	10
8. Thuật toán Dijkstra:	10
a) Giới thiệu:	10
b) Chi tiết:	11
9. Thuật toán Swarm (Dijkstra + $A^*$ ):	11
a) Giới thiệu:	11

b)	Chi tiết: .....	11
10.	Kết quả: .....	12
a)	Map 1: .....	12
b)	Map 3: .....	13
c)	Map 6: .....	14
d)	Map 7: .....	15
e)	Map 10: .....	16
IV.	GUI: .....	17
V.	TỰ ĐÁNH GIÁ: .....	19
1.	Về nhóm: .....	19
2.	Về thuật toán: .....	19
VI.	REFERENCE: .....	20
VII.	LINK: .....	20

# **I. THÔNG TIN CHUNG:**

## **1. Thông tin thành viên:**

- Khoa: Công nghệ Thông tin (CLC)
- Lớp: 23CLC09
- Số lượng thành viên: 4
  - o Nguyễn Hữu Kiến Phi – 23127242
  - o Lê Hoàng Long – 23127412
  - o Lê Nguyễn Nhật Khánh – 23127206
  - o Lê Trọng Đạt – 23127169

## **2. Thông tin đề án:**

- Tên: Ares's adventure (Project 1)
- Ngày bắt đầu: 10/02/2025
- Ngày hết hạn: 03/03/2025
- Sơ qua về đề án:
  - o Tạo mê cung với chướng ngại vật (đá) và kho báu theo các cấp độ khác nhau từ dễ đến khó. Đá sẽ có trọng số.
  - o Mục tiêu giải là Ares (nhân vật trong game) từ vị trí bắt đầu phải làm sao để đẩy các cục đá về vị trí Switch.
  - o Viết cách giải đồ mê cung với các thuật toán sau đây
    - Breadth-First Search (BFS).
    - Depth-First Search (DFS).
    - Uniform Cost Search (UCS).
    - A\* Search.
    - Greedy Best-First Search, Swarm (Dijkstra + A\*), Dijkstra (Bonus).
- Làm GUI cho người dùng tương tác.

## II. PHÂN CHIA CÔNG VIỆC:

THÀNH VIÊN	CÔNG VIỆC	MỨC ĐỘ HOÀN THÀNH
Nguyễn Hữu Kiến Phi	BFS, UCS, A*, Swarm, Dijkstra, kiểm tra report, code phần kiểm tra '+', '*'	100%
Lê Hoàng Long	Greedy BFS, Swarm (sửa lỗi), chỉnh code để các thuật toán chạy với map, tính trọng số (file Weight.py), quay video	100%
Lê Nguyễn Nhật Khánh	GUI, hỗ trợ (file Function.py) tạo testcase (7-10), check testcase (7-10), requirement	100%
Lê Trọng Đạt	DFS, tạo testcase (1-6), check testcase (1-6), sửa code phần '+' và '*', readme, report	100%

## III. CHI TIẾT ĐỒ ÁN:

### 1. Tổng quan về map (Testcase):

- Map có 5 cấp bậc khác nhau từ dễ tới khó, tạo bằng file text.
- Input 1-2 là cấp 1, tức là map đơn giản, trong map chỉ có 1 cục đá.
- Input 3-5 là cấp 2, map bắt đầu phức tạp hơn, trong map sẽ có 2 cục đá.
- Input 6 là cấp 3, trong map sẽ có 3 cục đá kèm theo đó là trường hợp người hoặc đá đứng trên ô Switch.
- Input 7,8 là cấp 4, map sẽ phức tạp hơn, trong map sẽ có 4 cục đá và trường hợp người hoặc đá đứng trên ô Switch.
- Input 9,10 là cấp 5, map lớn, phức tạp, trong map sẽ có 5 cục đá và trường hợp người hoặc đá đứng trên ô Switch.
- **Các map được dùng để trình bày kết quả trong báo cáo là map 1, 3, 6, 7 và 10.**

## 2. Một số lưu ý:

- Trong mỗi thuật toán sẽ có 1 đoạn code nhằm xác định vị trí bắt đầu của Ares, vị trí các hòn đá tạo thành **trạng thái ban đầu**.

```
# Tìm vị trí của Ares và các viên đá
ares = None
stones = {}
for r in range(rows):
    for c in range(cols):
        if matrix[r][c] == "@":
            ares = (r, c)
        elif matrix[r][c] == "$":
            stones[(r, c)] = stone_weight[len(stones)]
        elif matrix[r][c] == "+":
            ares = (r, c)
            matrix[r][c] = "."
        elif matrix[r][c] == "*":
            stones[(r, c)] = stone_weight[len(stones)]
            matrix[r][c] = "."
```

- “@” là biểu tượng Ares, “\$” là biểu tượng đá.
- Cần xác định thêm trường hợp “+” tức Ares đứng trên vị trí Switch (vị trí cần đưa đá tới) và “\*” tức đá đã ở sẵn trên vị trí Switch nếu không chương trình sẽ không thể xác định Ares hoặc đá.
- Quy ước:
  - o “u,d,l,r” lần lượt là lên, xuống, trái, phải.
  - o “U,D,L,R” là lên, xuống, trái, phải nhưng là vừa di chuyển vừa đẩy đá.

```
directions = [(-1, 0, "u"), (1, 0, "d"), (0, -1, "l"), (0, 1, "r")] # Lên, xuống, trái, phải
node_count = 0
```

- Đây là khai báo hướng di chuyển trong tất cả các file thuật toán.

### 3. Thuật toán BFS:

#### a) Giới thiệu:

- Là thuật toán duyệt theo chiều rộng, BFS sử dụng queue để lưu trữ, tránh việc lặp lại.
- Ưu điểm:
  - Đảm bảo lời giải tìm thấy là ngắn nhất.
  - Không bị sót trường hợp.
  - Tối ưu với những bản đồ nhỏ.
- Nhược điểm:
  - Cần dùng nhiều bộ nhớ (lưu trữ những trường hợp đã chạy qua).
  - Không hiệu quả với những bản đồ lớn, phức tạp.

#### b) Chi tiết:

- Sau khi xác định được trạng thái ban đầu, tạo queue để lưu. Đồng thời tạo tập hợp visited (kiểu dữ liệu set) nhằm lưu những vị trí đã đi qua, tránh bị lặp lại.
- Chương trình lặp sẽ để lấy trạng thái trong queue để kiểm tra, phát triển theo code khai báo đã nói bên trên, tuần tự: lên (u), xuống (d), trái (l), phải (r), Không ưu tiên hướng nào trước, nếu cả 4 hướng đều khả thi sẽ đưa vào hàng đợi queue.
- Sau khi đưa vào queue (gọi là mức 1), sẽ xét tiếp mức 2, tức là chương trình đưa vào queue vị trí mới từ chỗ ban đầu đi lên, tiếp đến là vị trí mới từ chỗ ban đầu đi xuống, tiếp đến là vị trí mới từ chỗ ban đầu qua trái, tiếp đến là vị trí mới từ chỗ ban đầu qua phải. Xong mức 2, tiếp tục qua mức 3,...
- Điều kiện dừng là queue không còn giá trị để kiểm tra hoặc tất cả các viên đá đã vào vị trí Switch.
- Bắt đầu mỗi vòng lặp sẽ kiểm tra điều kiện như trên, nếu chưa thỏa thì tiếp tục lấy giá trị trong queue để xét tiếp 4 hướng. Lặp tới khi nào thỏa điều kiện thì ngừng.

### 4. Thuật toán DFS:

#### a) Giới thiệu:

- Là thuật toán duyệt theo chiều sâu, thường được sử dụng stack. Nếu BFS lưu trạng thái khả thi để lấy ra test bằng queue thì DFS lưu trạng thái đang xử lý bằng stack.
- Ưu điểm:
  - Mất ít thời gian để tìm đường đi.
  - Không sử dụng quá nhiều bộ nhớ.
  - Phù hợp với những map đơn giản.

- Nhược điểm:
  - o Mất ít thời gian nhưng không đảm bảo đường đi ngắn nhất, thậm chí nghèo nàn.
  - o Thường không phù hợp cho việc tìm đường đi, đặc biệt là những map phức tạp, lớn.

#### **b) Chi tiết:**

- Sau khi xác định trạng thái ban đầu sẽ tạo stack để lưu, kèm thêm visited để lưu những vị trí đã duyệt, tránh lặp lại.
- Chương trình sẽ lấy trạng thái cuối cùng của stack để kiểm tra. Thứ tự code khai báo là lên, xuống, trái, phải, tuy nhiên stack hoạt động theo LIFO (Last in first out), tức là sẽ lấy ra các phần tử cuối để xử lý trước (xử lý theo tuần tự phải, trái, xuống, lên).
- Tuy nhiên, khác với BFS ở chỗ DFS không xét hết mức 1 mà lấy phần tử cuối của stack xong xét tiếp luôn mức 2, nếu cần thiết mới quay đầu lại. Đó là lí do DFS không thể tối ưu bằng BFS, hoặc thậm chí nghèo nàn.
- Điều kiện dừng là khi tất cả các hòn đá đều được đặt đúng vị trí Switch.
- Lặp tới khi nào điều kiện dừng thỏa, nếu chưa thỏa sẽ tiếp tục lấy trạng thái cuối cùng trong stack để xử lý tiếp.

### **5. Thuật toán UCS:**

#### **a) Giới thiệu:**

- UCS là một thuật toán tìm kiếm giống BFS, tuy nhiên nó duyệt theo đường đi có chi phí thấp nhất chứ không phải theo chiều rộng. Phương pháp này sử dụng hàng đợi ưu tiên (priority queue) để chọn trạng thái có chi phí thấp nhất.
- Ưu điểm:
  - o Phù hợp cho những bài toán có chi phí di chuyển khác nhau.
  - o Đảm bảo tìm được đường đi có chi phí thấp nhất.
- Nhược điểm:
  - o Khi chi phí di chuyển của đường đi như nhau thì sẽ không tối ưu bằng BFS (Phải sắp xếp priority queue).
  - o Nếu bản đồ quá lớn, phức tạp sẽ bị quá tải bộ nhớ.



### b) Chi tiết:

- Về hàng đợi ưu tiên priority queue, 1 phần tử lưu vào sẽ bao gồm chi phí (cost) cùng vị trí Ares và vị trí viên đá. UCS sẽ duyệt để lấy ra phần tử có cost nhỏ nhất để xét 4 hướng. Cứ như thế cho tới khi thỏa điều kiện tất cả các viên đá nằm trên vị trí Switch thì ngừng thuật toán.
- Tuy nói là duyệt theo đường đi có chi phí thấp nhất, nhưng ở trong đồ án này, chi phí di chuyển là bằng nhau và bằng 1. Chính vì thế thuật toán UCS trong trường hợp này chạy tương tự với BFS.

## 6. Thuật toán A Star (A\*):

### a) Giới thiệu:

- Đây là một thuật toán dựa trên sự đánh giá heuristic (có thể gọi là hàm ước lượng) để tìm kiếm, được ưa chuộng cho việc tìm đường đi. Sử dụng hàng đợi ưu tiên để lưu trữ dữ liệu.
- $f(n) = g(n) + h(n)$ : Đây được gọi là heuristic function,  $g(n)$  là chi phí từ trạng thái ban đầu đến trạng thái  $n$ ,  $h(n)$  là hàm ước lượng chi phí từ  $n$  đến đích.  $F(n)$  là tổng chi phí dự đoán tốt nhất từ đầu đến đích.
- Khi mở rộng, A\* sẽ lấy phần tử có giá trị  $f(n)$  thấp nhất để xét duyệt.
- Ưu điểm:
  - o Nhanh nếu heuristic đủ tốt.
  - o Tối ưu chi phí, tốc độ.
  - o Ứng dụng rộng rãi.
- Nhược điểm:
  - o Tiêu tốn rất nhiều bộ nhớ.
  - o Nếu bản đồ quá lớn sẽ không hiệu quả.

### b) Chi tiết:

- Heuristic function: Tính khoảng cách từ tất cả những viên đá tới vị trí Switch gần nhất, công thức như sau.

```
def heuristic(stone_pos, goal_positions):  
    return sum(min(abs(r - gr) + abs(c - gc) for gr, gc in goal_positions) for r, c in stone_pos)
```

- Giả sử có viên đá ở (1,2) và Switch là (3,2) thì  $g(n) = |1-3| + |2-2| = 2$ .

- Đầu tiên chương trình sẽ thêm trạng thái ban đầu vào hàng chờ ưu tiên (priority queue) sau đó xét duyệt các vị trí, tính tổng cost và vị trí của các vị trí được xét duyệt rồi lưu vào priority queue. Sau đó chương trình sẽ lấy phần tử có cost nhỏ nhất để xét như đã nói.

## 7. Thuật toán Greedy BFS:

### a) Giới thiệu

- Đây là thuật toán tìm kiếm dựa trên hàm heuristic (  $h(n)$  ), cụ thể là sẽ tìm kiếm dựa vào phần tử có giá trị  $h(n)$  nhỏ nhất. Không tính thêm  $g(n)$  như A\*. Dùng hàng đợi ưu tiên (priority queue) để lưu trữ dữ liệu.
- Ưu điểm:
  - o Rất nhanh nếu heuristic tốt.
- Nhược điểm:
  - o Không đảm bảo tối ưu.

### b) Chi tiết:

```
def heuristic(stone_pos, goal_positions):
    return sum(min(abs(r - gr) + abs(c - gc) for gr, gc in goal_positions) for r, c in stone_pos)
```

- Đây là hàm tính  $h(n)$ , khoảng cách từ tất cả những cục đá đến điểm Switch gần nhất.  $h(n) = |x1-x2| + |y1-y2|$  giống với hàm của A\* đã đề cập và giải thích phía trên.
- Sau khi trạng thái đầu tiên được thêm vào priority queue và visited thì bắt đầu vòng lặp. Từ vị trí ban đầu đó xét các nước có thể đi được, nếu không trùng với visited thì tính  $h(n)$  của các nước đi sau đó nhét vào queue, chương trình sẽ lấy phần tử có  $h(n)$  nhỏ nhất để tiếp tục xét.
- Điều kiện dừng vòng lặp là tất cả viên đá đều đã ở trên ô Switch.

## 8. Thuật toán Dijkstra:

### a) Giới thiệu:

- Đây là một trong những thuật toán cổ điển dùng để giải quyết những bài toán tìm đường đi từ 1 điểm cho trước trong đồ thị có trọng số (trọng số không được âm), thậm chí được ứng dụng rộng rãi trong thực tế. Thường dùng priority queue để lưu trữ dữ liệu.

- Ưu điểm:
  - o Đảm bảo đường đi ngắn nhất.
- Nhược điểm:
  - o Nếu đồ thị, map quá lớn sẽ tốn bộ nhớ, đặc biệt chạy chậm nếu không dùng priority queue.

### **b) Chi tiết:**

- Sau khi đưa trạng thái đầu tiên vào priority queue, xét 4 hướng từ trạng thái đó để lấy hướng đi phù hợp.  $g(n)$  sẽ được cộng, tuy nhiên trường hợp này giống UCS, vì các hướng đi đều có trọng số là 1 nên cũng hoạt động giống với BFS.
- Các hướng đi hợp lệ được lưu vào priority queue, vì đều bằng nhau hết nên sẽ được xét lần lượt giống với BFS.
- Lặp tới khi nào thỏa điều kiện các hòn đá nằm trên ô Switch sẽ ngừng, nếu không có lời giải thì thông báo.

## **9. Thuật toán Swarm (Dijkstra + A\*):**

### **a) Giới thiệu:**

- Còn được gọi là thuật toán bầy đàn, có nhiều dạng Swarm, tuy nhiên trong đồ án này dùng loại kết hợp giữa thuật toán Dijkstra và thuật toán A\*. Tức vừa dùng heuristic  $h(n)$  như A\* vừa xem xét cả tổng chi phí  $g(n)$  như Dijkstra nhằm tối ưu kết quả.
- Ưu điểm:
  - o Có thể dùng thêm hệ số điều chỉnh giúp chương trình chạy nhanh hơn.
  - o Cân bằng giữa thời gian, độ chính xác.

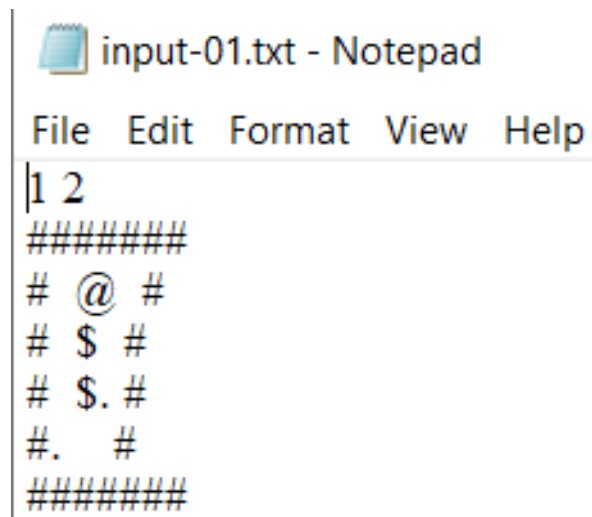
### **b) Chi tiết:**

```
def manhattan_distance(pos1, pos2):
    return abs(pos1[0] - pos2[0]) + abs(pos1[1] - pos2[1])
```

- Ở thuật toán này chúng ta tính heuristic bằng khoảng cách Manhattan (từ mỗi viên đá tới vị trí Switch gần nhất), công thức là  $h(n) = |x1-x2| + |y1-y2|$ .
- Sau khi thêm trạng thái ban đầu vào priority queue, bắt đầu vòng lặp thuật toán. Nó sẽ lấy phần tử có  $f(n) = h(n) + g(n)$  nhỏ nhất từ hàng đợi ưu tiên để mở rộng 4 hướng. Sau khi mở rộng, tính toán heuristic ở vị trí mới rồi thêm tiếp vào priority queue, lặp đi lặp lại.
- Điều kiện dừng chính là khi các hòn đá đã ở trên vị trí Switch.

## 10. Kết quả:

### a) Map 1:



```
input-01.txt - Notepad
File Edit Format View Help
1 2
#####
# @ #
# $ #
# $. #
#. #
#####
```

BFS

Steps: 10, Weight: 6, Node: 1003, Time (ms): 31.02, Memory (MB): 0.41

rdLulDDRdL

DFS

Steps: 182, Weight: 48, Node: 1908, Time (ms): 38.01, Memory (MB): 0.3

rrddLrruullldddRRllluurrrdddlUruullDldddrrruuulDDrdLruulllddrRluRllluurrrdddLLrruLrruullldddRRlUr  
rrdLruullulddRRlllddrRllluurrrdLrddLLrruulLdluRRddLrruuLrddllulluurrDrrddllluR

UCS

Steps: 10, Weight: 6, Node: 973, Time (ms): 23.99, Memory (MB): 0.31

rdLLulDDrR

A\*

Steps: 10, Weight: 6, Node: 234, Time (ms): 10.0, Memory (MB): 0.07

rdLLulDDrR

GBFS

Steps: 24, Weight: 8, Node: 213, Time (ms): 5.99, Memory (MB): 0.02

ldRurDDuullddRuurrdddLLL

Swarm

Steps: 10, Weight: 6, Node: 2087, Time (ms): 177.0, Memory (MB): 0.4

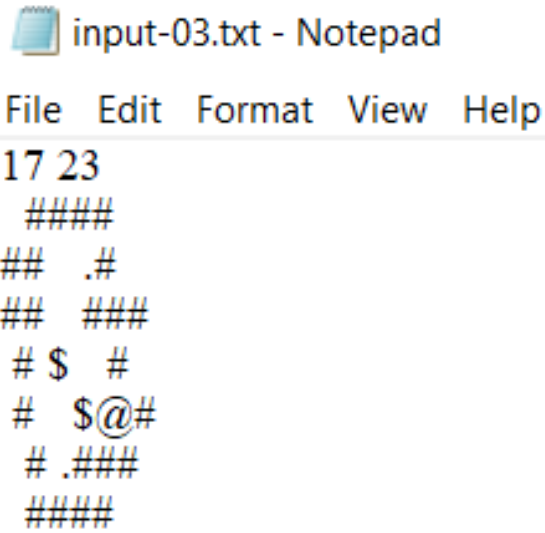
rdLLulDDrR

Dijkstra

Steps: 10, Weight: 6, Node: 1878, Time (ms): 44.0, Memory (MB): 0.37

rdLLulDDrR

**b) Map 3:**



```
input-03.txt - Notepad
File Edit Format View Help
17 23
####
## .#
## ####
# $ #
# $@#
# .####
####
```

BFS

Steps: 11, Weight: 114, Node: 759, Time (ms): 15.03, Memory (MB): 0.1

LulDIUUhRR

DFS

Steps: 37, Weight: 228, Node: 1555, Time (ms): 27.0, Memory (MB): 0.22

LLuulldRRlldRhuurrdDrruLLdluRdrUUluR

UCS

Steps: 11, Weight: 114, Node: 682, Time (ms): 11.97, Memory (MB): 0.07

LulDIUUhRR

A\*

Steps: 11, Weight: 114, Node: 164, Time (ms): 6.0, Memory (MB): 0.03

LulDIUUhRR

GBFS

Steps: 19, Weight: 114, Node: 111, Time (ms): 3.0, Memory (MB): 0.02

LulDuullddRdrUUluR

Swarm

Steps: 11, Weight: 114, Node: 2148, Time (ms): 183.0, Memory (MB): 0.34

LulDIUUhRR

Dijkstra

Steps: 11, Weight: 114, Node: 2148, Time (ms): 47.0, Memory (MB): 0.34

LulDIUUhRR

### c) Map 6:

```

input-06.txt - Notepad
File Edit Format View Help
11 22 33
#####
#@      #
# $      #####
# ##   ## ## $   .#
#      #####
#      .#
# $#####
### # #   ##   #####
### # #   ##
### ##   ##
###.#   ##
#####   ##

```

BFS

Steps: 38, Weight: 418, Node: 61167, Time (ms): 1764.0, Memory (MB): 11.21

rrrrDDDldDDDDuuuurRRRRRRRRRuruRRRRRRR

DFS

Steps: 172, Weight: 440, Node: 663, Time (ms): 11.0, Memory (MB): 0.07

rrrrrrrrrrrrddRRRRRRRllllllldllllllllluurrrRRRRRRRRRlllllllddrddDlлуurrrrrrrrruuururDlllllllddrddD  
DDuuullluurrrrrrrrrrrUdllllllllluurrrrrrrrrurDDD

UCS

Steps: 38, Weight: 418, Node: 59724, Time (ms): 1839.0, Memory (MB): 11.06

rrrrDDDldDDDDuuuurRRRRRRRRRuruRRRRRRR

A\*

Steps: 38, Weight: 418, Node: 15602, Time (ms): 805.03, Memory (MB): 3.01

rrrrDDDldDDDDuuuurRRRRRRRRRuruRRRRRRR

GBFS

Steps: 104, Weight: 462, Node: 5444, Time (ms): 235.0, Memory (MB): 1.09

rrrrDDDuulllddrddDDDDuuuuuuuuurrrrrdddddLLulluuurrrrrrrrrrrddRRRRRRRllllllluulllllllllddddRRRRR  
RRRRRR

Swarm

Steps: 42, Weight: 418, Node: 191545, Time (ms): 42558.42, Memory (MB): 30.74

rdddrddDDDDuuuuuuuuurrrRRRRRRRRRurDDDuRRRRRRR

Dijkstra

Steps: 42, Weight: 418, Node: 191543, Time (ms): 8421.73, Memory (MB): 30.73

rdddrddDDDDuuuuuuuuurrrRRRRRRRRRurDDDuRRRRRRR

## d) Map 7:



input-07.txt - Notepad

File Edit Format View Help

77 66 55 33

#####

# #####

# \$ ##

##\$#\*. #

# \$ .. #

# @ ## #

#####

BFS

Steps: 96, Weight: 1914, Node: 291164, Time (ms): 8866.75, Memory (MB): 48.14

luRRRRuuLDrdLLdluRRRuulldRDuRRdrrddluRuLuLLddRlldluRRRuulDDurddlldluRRluuerrrdLrrddluRuL

DFS

Steps: 396, Weight: 5500, Node: 293083, Time (ms): 8332.22, Memory (MB): 44.64

ruuuuulLrrddlldluRRRRurrdLLruullLrrddllLdluRRRRurrdluRuLuulldRRDRdruLLdLLdluRRRRurrdluRuLuLLDRl  
uLrrDrddluRuLdlLruullldRRRdRdruLuLrddllLrruulLrrddlldluRRRRluurdrdLLrruulDldRlLrrurduuulLrddRlldluRR  
RuulldRRRlIDurddrrddluRuLuLLRdRlulDrrrDrddluRuLuulldRlldluRRRlUdrruurrdrddluLLLuurrDullulDrrrdrddluLl  
dluRRRlUdrruulDrrrDulllIDurddlldluRRluuerrrdLrrddluRuL

UCS

Steps: 96, Weight: 1914, Node: 291287, Time (ms): 9631.44, Memory (MB): 48.03

luRRRRuuLDrdLLdluRRRuulldRDuRRdrrddluRuLuLLddRlldluRRRuulDDurddlldluRRluuerrrdLrrddluRuL

A\*

Steps: 96, Weight: 1914, Node: 288142, Time (ms): 14887.56, Memory (MB): 48.02

luRRRRuuLDrdLLdluRRRuulldRDuRRdrrddluRuLuLLddRlldluRRRuulDDurddlldluRRluuerrrdLrrddluRuL

GBFS

Steps: 134, Weight: 2200, Node: 56842, Time (ms): 2786.73, Memory (MB): 11.43

luRRRRurrdluRuLuLDRurDullLulldRRRdrdLLdluRRRuulldurddlldluRuurrdrddluRuLuLLddRluuLulldRddRRuulDD  
urddlldluRRluuerrrdLrrddluRuL

Swarm

Steps: 96, Weight: 1914, Node: 287648, Time (ms): 41792.11, Memory (MB): 48.13

luRRRRuuLDrdLLdluRRRuulldRDuRRdrrddluRuLuLLddRlldluRRRuulDDurddlldluRRluuerrrdLrrddluRuL

Dijkstra

Steps: 96, Weight: 1914, Node: 287646, Time (ms): 12604.6, Memory (MB): 48.12

luRRRRuuLDrdLLdluRRRuulldRDuRRdrrddluRuLuLLddRlldluRRRuulDDurddlldluRRluuerrrdLrrddluRuL

**e) Map 10:**

```
input-10.txt - Notepad
File Edit Format View Help
14 16 12 99 109
#####
### #
# @ #
### $.#
#.$$.#
# # . ##
#$ * $.#
# #
#####
```

BFS

Steps: 28, Weight: 163, Node: 151427, Time (ms): 4994.0, Memory (MB): 45.47

rrrddldRuuuulldRDrddllllUU

DFS

Steps: 56, Weight: 421, Node: 1010594, Time (ms): 30007.62, Memory (MB): 175.64

rrrddldRlLrruuuulldRluurrdDlDDrddllllURlUdrdrrruuUdddlU

UCS

Steps: 28, Weight: 163, Node: 143558, Time (ms): 5009.99, Memory (MB): 45.22

rrrddldRuuuulldRDrddllllUU

A\*

Steps: 28, Weight: 163, Node: 40508, Time (ms): 2598.99, Memory (MB): 14.46

rrrddldRuuuulldRDrddllllUU

GBFS

Steps: 44, Weight: 361, Node: 274854, Time (ms): 28193.4, Memory (MB): 59.82

ruurddldRuuuulldRDrddllllURldlUUdrdruuL

Swarm

Steps: 28, Weight: 163, Node: 19725, Time (ms): 6221.34, Memory (MB): 7.33

rrrddldRuuuulldRDrddllllUU

Dijkstra

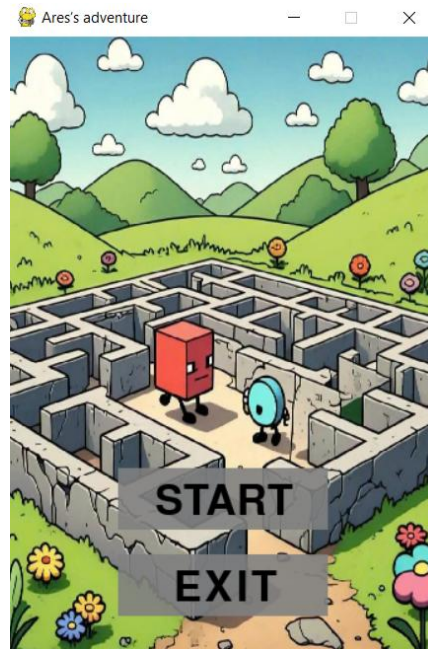
Steps: 28, Weight: 163, Node: 19725, Time (ms): 1382.2, Memory (MB): 7.24

rrrddldRuuuulldRDrddllllUU



#### IV. GUI:

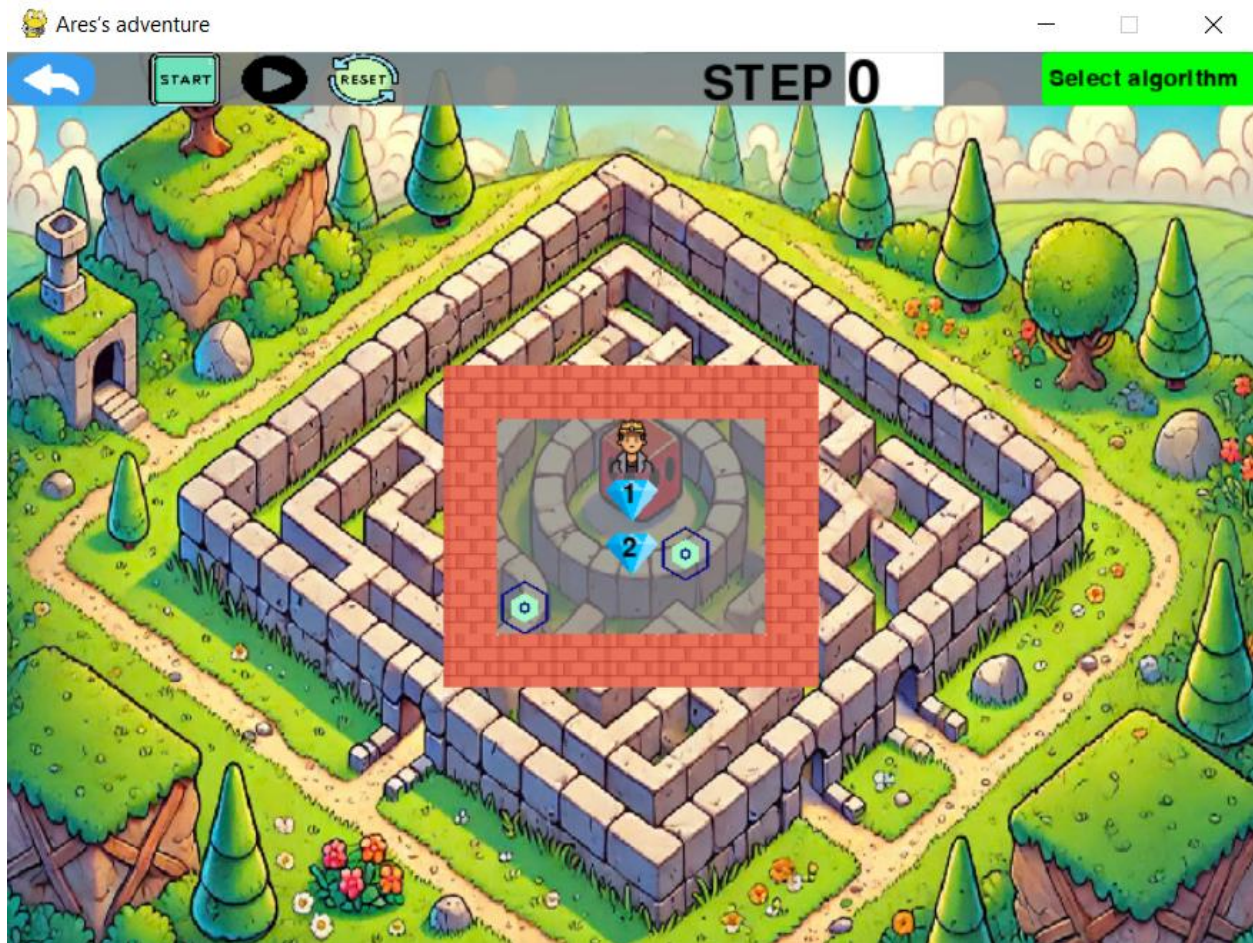
- Đây là giao diện giúp người sử dụng tương tác với trò chơi.
- Màn hình chính:



- Bấm start sẽ bắt đầu trò chơi, nếu bấm exit thì sẽ tắt, dừng chương trình. Sau khi bấm start, người dùng sẽ được chọn map.



- Ví dụ người dùng chọn map 1 thì nó sẽ hiện tiếp giao diện này, tùy theo loại map từ dễ tới khó mà chương trình load nhanh hay chậm, map càng khó và phức tạp thì thời gian load càng lâu, tối đa 2 phút.



- Người dùng có thể chọn thuật toán muốn duyệt bằng cách nhấp chuột vào ô “Select algorithm” sau đó bấm start, nếu người dùng chưa chọn thuật toán mà bấm start thì mặc định sẽ duyệt bằng BFS.
- Nút dừng/tiếp tục.
- Reset dùng để đặt lại, đưa trò chơi về trạng thái ban đầu.
- Step là đếm số bước đi hiện tại.
- Người dùng có thể quay lại trang chọn bản đồ bằng cách nhấn mũi tên nằm ở góc trên bên trái giao diện.
- Đồng thời ngay sau lúc nhấn chọn map, 1 file output sẽ được xuất ra bao gồm thời gian chạy, bộ nhớ, số bước đi của tất cả các thuật toán áp dụng vào map đó (Như đã chụp ở bên trên phần kết quả).

## V. TỰ ĐÁNH GIÁ:

### 1. Về nhóm:

- Đối với nhóm em, khó khăn lớn nhất là code phần kiểm tra trường hợp Ares hoặc đã đứng trên Switch. Trường hợp Ares đứng trên Switch mất khá nhiều thời gian để sửa lỗi vì gián đoạn ở phần không tìm được Ares trên bản đồ (lí do vì code phần kiểm tra và sửa lại kí tự sau khi Ares di chuyển chưa hợp lý).
- Những phần khác nhóm em hoàn thành tốt.
- Qua dự án này, chúng em có được nhiều kiến thức hơn đối với những thuật toán trong việc tìm kiếm đường đi, hiểu được cách dùng tối ưu nhất có thể cho từng trường hợp.

### 2. Về thuật toán:

- Thuật toán DFS hoạt động không tối ưu bằng các thuật toán khác, thậm chí có những lúc tìm đường ngoằn ngoèo. Không nên dùng thuật toán này trong tìm đường.
- Các thuật toán cần dùng tới trọng số như UCS, hay Dijkstra sẽ hiệu quả hơn nếu chi phí đường đi là khác nhau, trong trường hợp này, chi phí đường đi là như nhau nên dùng thuật toán BFS sẽ tối ưu nhất.
- Số node tạo ra ở các thuật toán chưa tối ưu.
- Bảng tổng kết:

Thuật toán	Tốc độ	Tối ưu	Yêu cầu bộ nhớ	Nhược điểm
BFS	Trung bình	Có	Cao	Không hiệu quả với bản đồ lớn
DFS	Nhanh	Không	Thấp	Đường đi có thể ngoằn ngoèo
UCS	Trung bình	Có	Cao	Chỉ nhanh khi trọng số khác nhau
A*	Nhanh	Có	Cao	Hàm heuristic phải tốt
Greedy BFS	Rất nhanh	Không	Thấp	Không đảm bảo đường tìm được tốt nhất
Dijkstra	Chậm	Có	Cao	Chỉ nhanh khi trọng số khác nhau
Swarm	Nhanh	Cân bằng	Cao	Cần điều chỉnh tham số

## **VI. REFERENCE:**

- Chatgpt
- Deepseek
- [https://github.com/pinkuuuu/path-finding-AI-19\\_21-hcmus](https://github.com/pinkuuuu/path-finding-AI-19_21-hcmus)
- [https://github.com/xxtnguyn/AISearch\\_AresAdventure](https://github.com/xxtnguyn/AISearch_AresAdventure)

## **VII. LINK:**

- Video demo: <https://m.youtube.com/watch?v=vKhrWJpEdPk>
- Link Github: [http://github.com/GinKzZ/Project\\_Ares\\_Adventure](http://github.com/GinKzZ/Project_Ares_Adventure)