

VIETNAM NATIONAL UNIVERSITY,  
HO CHI MINH CITY

UNIVERSITY OF SCIENCE  
FACULTY OF INFORMATION TECHNOLOGY

---

# Project 01: Ares's Adventure Report

---

CS14003 – INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Ngo Nguyen The Khoa 23127065  
Bui Minh Duy 23127040  
Nguyen Le Ho Anh Khoa 23127211

March 3, 2025

# Contents

<b>1</b>	<b>Group Information</b>	<b>2</b>
<b>2</b>	<b>Project Information</b>	<b>3</b>
<b>3</b>	<b>Work assignment table</b>	<b>3</b>
<b>4</b>	<b>Screenshots</b>	<b>4</b>
4.1	Breadth First Search . . . . .	4
4.2	BFS Algorithms . . . . .	4
4.3	Depth First Search . . . . .	5
4.4	Uniform Cost Search . . . . .	6
4.5	A* Search with heuristic . . . . .	7
<b>5</b>	<b>References</b>	<b>9</b>

# 1 Group Information

- **Subject:** Introduction to Artificial Intelligence.
- **Class:** 23CLC09.
- **Lecturer:** Bui Duy Dang, Le Nhut Nam.
- **Team members:**

No.	Fullname	Student ID	Email
1	Ngo Nguyen The Khoa	23127065	<a href="mailto:nntkhoa23@clc.fitus.edu.vn">nntkhoa23@clc.fitus.edu.vn</a>
2	Bui Minh Duy	23127040	<a href="mailto:bmduy23@clc.fitus.edu.vn">bmduy23@clc.fitus.edu.vn</a>
3	Nguyen Le Ho Anh Khoa	23127211	<a href="mailto:nlhakhoa23@clc.fitus.edu.vn">nlhakhoa23@clc.fitus.edu.vn</a>

- **Tools:**
  - **Git**, **GitHub**: Source code version control.
  - **DrawIO**: UML drawing.
  - **CapCut**: Video editing.
  - **ChatGPT**, **Gemini** and **DeepSeek**.
  - **Visual Studio Code**: Code editing (Python, Latex).

## 2 Project Information

- **Name:** Ares.
- **Developing Environment:** Visual Studio Code (Windows).
- **Programming Language:** Python.
- **Libraries and Tools:**
  - **rye:** A comprehensive project and package management solution for Python.
  - **CustomTkinter:** GUI library.

## 3 Work assignment table

No.	Task Description	Assigned to
1	desc	member

## 4 Screenshots

iiiiiii HEAD

### 4.1 Breadth First Search

Breadth First Search (BFS), as the name says, explores the search space in the increasing order of the depth and the costs of traveling from one state to another is assumed to be a positive number. Typically, this algorithm is often associated with the concept of stack and queue and pushing and popping from the stack. Due to the larger states explored at shorter depths, the space complexity is very high of about  $O(b^d)$  and the time complexity is  $O(b^d)$ .

#### Pseudo Code

---

**Algorithm 1** Breadth First Search (*state*, *maxdepth*, *maxtimeout*)

---

```

1: queue  $\leftarrow$  starting position of Sokoban
2: cost  $\leftarrow$  cost of moves
3: while queue is not empty do
4:   Remove the first element of queue
5:   if Ares crates on target then
6:     break
7:   else
8:     if Is deadlock or depth  $\geq$  maxdepth or time  $\geq$  maxtimeout then
9:       pick next solution
10:    else
11:      Get valid moves for Sokoban
12:      for all move do
13:        Find next state
14:        Put it in queue with current cost + 1
15:      end for
16:    end if
17:  end if
18: end while
19: return moves

```

---

#### Implementation

### 4.2 BFS Algorithms

#### Introduction

Breadth-First Search (BFS) is a fundamental graph traversal algorithm used to explore nodes in a graph or tree data structure. It operates by visiting all nodes at the current depth level ("breadth") before moving to nodes at the next depth level. BFS guarantees the shortest path in an unweighted graph, making it highly useful in applications like network routing, social network analysis, and puzzle solving.

## Algorithm Overview

### 1. Key Principles

**Queue Data Structure:** BFS uses a queue (First-In-First-Out principle) to track nodes to be explored.

**Layer-by-Layer Exploration:** Nodes are visited level by level, starting from the root (or source node).

**Visited Tracking:** A visited array or hash map prevents reprocessing nodes, avoiding cycles.

### 2. Steps of BFS

#### (a) **Initialize:**

Select a starting node (root).

Create a queue and add the root node.

Mark the root as visited.

#### (b) **Loop Until Queue is Empty:**

Dequeue the front node.

Process the node (e.g., print its value).

Enqueue all adjacent unvisited nodes and mark them as visited.

#### (c) **Terminate:** When the queue is empty, traversal is complete.

## 4.3 Depth First Search

Depth First Search (DFS) is a special case of backtracking search algorithm. The search starts from the root and proceeds to the farthest node before backtracking. The difference between this and the backtracking is that this stops the search once a goal is reached and does not care if it is not minimum. The space complexity is  $O(D)$  and the time complexity is  $O(b^D)$  which is very high.

## Pseudo Code

---

### Algorithm 2 Depth First Search (*state*, *maxdepth*, *maxtimeout*)

---

```

1: stack  $\leftarrow$  starting position of Sokoban
2: while stack is not empty do
3:   if Ares crates on target then
4:     break
5:   else
6:     if Is deadlock or depth  $\geq$  maxdepth or time  $\geq$  maxtimeout then
7:       pick next solution
8:     else
9:       Get valid moves for Sokoban
10:      for all move do
11:        Find next state
12:        Put it in stack
13:      end for
14:    end if
15:  end if
16: end while
17: return moves

```

---

## Implementation

### 4.4 Uniform Cost Search

For any search problem, Uniform Cost Search (UCS) is the better algorithm than the previous ones. The search algorithm explores in branches with more or less same cost. This consist of a priority queue where the path from the root to the node is the stored element and the depth to a particular node acts as the priority. UCS assumes all the costs to be non negative. While the DFS algorithm gives maximum priority to maximum depth, this gives maximum priority to the minimum cumulative cost.

## Pseudo Code

---

**Algorithm 3** Uniform Cost Search (*state*, *maxtimeout*)
 

---

```

1: priority queue  $\leftarrow$  starting position of Sokoban
2: cost  $\leftarrow$  cost of moves
3: while queue is not empty do
4:   Remove the highest priority element of queue
5:   if Ares crates on target then
6:     break
7:   else
8:     if Is deadlock or time  $\geq$  maxtimeout then
9:       pick next solution
10:    else
11:      Get valid moves for Sokoban
12:      for all move do
13:        cost  $\leftarrow$  cost of move
14:        Add cost to current move
15:        Update queue with new cost and new state
16:      end for
17:    end if
18:  end if
19: end while
20: return moves

```

---

## Implementation

### 4.5 A\* Search with heuristic

A\* algorithm is one of the popular technique used in path finding and graph traversals. This algorithm completely relies on heuristics for computing the future cost of a problem. This algorithm is equivalent to the uniform cost search with modified edge cost. This heuristics is chosen according to the case where the algorithm is implemented, thus emphasizing the importance of domain knowledge. This algorithm is consistent if the modified cost is greater than zero.



---

**Algorithm 4** A\* Search with heuristic (*state*, *maxdepth*, *maxtimeout*)

---

```
1: priority queue  $\leftarrow$  starting position of Sokoban
2: cost  $\leftarrow$  cost of moves
3: Fix the Heuristics
4: while queue is not empty do
5:   Remove the highest priority element of queue
6:   if Ares crates on target then
7:     break
8:   else
9:     if Is deadlock or depth  $\geq$  maxdepth or time  $\geq$  maxtimeout then
10:      pick next solution
11:    else
12:      Get valid moves for Sokoban
13:      for all move do
14:        cost  $\leftarrow$  cost of move with heuristics
15:        Add cost to current move
16:        Update queue with new cost and new state
17:        Find Heuristics of new state
18:      end for
19:    end if
20:  end if
21: end while
22: return moves
```

---

**Implementation**

## 5 References

1. [Rye documentation](#)