

**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**SCHOOL OF INFORMATION COMMUNICATION AND TECHNOLOGY**

**-----□□&□□-----**



**Introduction to Artificial Intelligence**

**REPORT CAPSTONE PROJECT**

**SEARCHING IN PACMAN**

**Instructor: Asoc.Prof. Than Quang Khoat**

**Group 14**

**Student names:**

|  |  |
| --- | --- |
| **Nguyen Ngoc Dang** | **20200149** |
| **Nguyen Hop Phu** | **20205165** |
| **Tran Quang Nam** | **20200427** |
| **Nguyen Le Hung** | **20205156** |

***Hanoi, July 2022***

***Hà Nội, tháng 10 năm 2012***

**Abstract**. Video games are an interesting field of study for many artificial intelligence researchers, since many different AI methods can be studied and tested with them, and later those investigations can be applied to many other situations. In this paper we use searching algorithm to make a bot can find coins in the shortest paths similar with PacMan. This is a report presented based on what we have learned and done. The main topic of this report is about the use of 2 algorithms BFS and A\* Search in the game Pac-Man. In particular, we use the BFS algorithm and A\* algorithm for finding and then we make a comparision between them about the times and nodes.

# INTRODUCTION

Video games are an interesting field of study for many Artificial Intelligence (AI) researchers, since they provide complex but controlled environments in which many AI techniques can be studied and tested. Video games are also very interesting to compare different AI techniques and learn their strengths and weaknesses.

In this work we use one of the most popular video games of all time: Pac-Man.

Pac-Man (パックマン Pakkuman) is an arcade game developed by Namco and first released in Japan on May 22, 1980. Popular and popular from the moment it was released to this day, Pac-Man is considered a classic game and became an icon of 80s pop culture.

Although Pac-Man may seem simple compared to some other kind of games, its special features make it a perfect target for AI researchers. Firstly, this game is simple enough to be quickly understood and it does not require a very powerful machine to be run. That's why we decided to use this game to make Capstone Project. However, we will have a little different adjustment from the original PacMan



Pac-Man Game

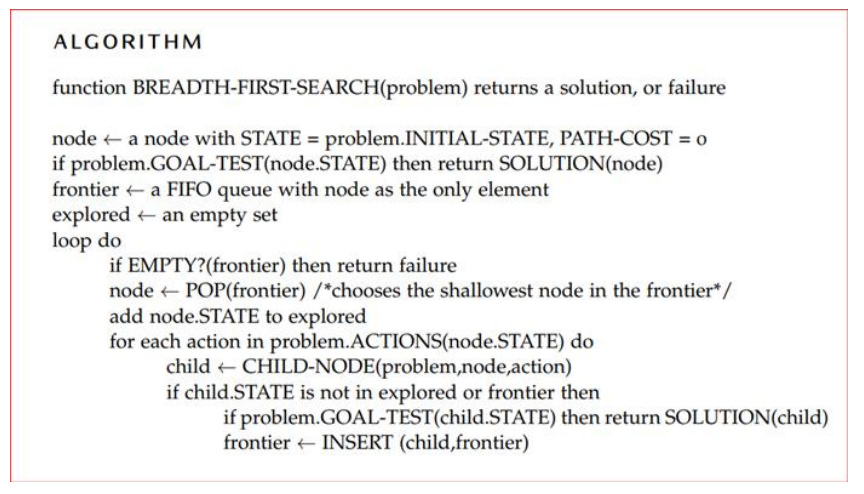
# PROBLEM DESCRIPTION

Searching path is a regular problem in the real life. There are many algorithms to solve those problems but most of students do not have practical knowledge for those. In this project, our task is appling two algorithm that are BFS and A\* into the Pacman game so that we can find out not only the diffirence but also the limit of them.

# ALGORYTHM

❖ **Breadth First Search**

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. Whereas DFS uses a LIFO queue, BFS is achieved very simply by using a FIFO queue (which is also called Queue) for the frontier. This is the pseudo code of the algorithm:

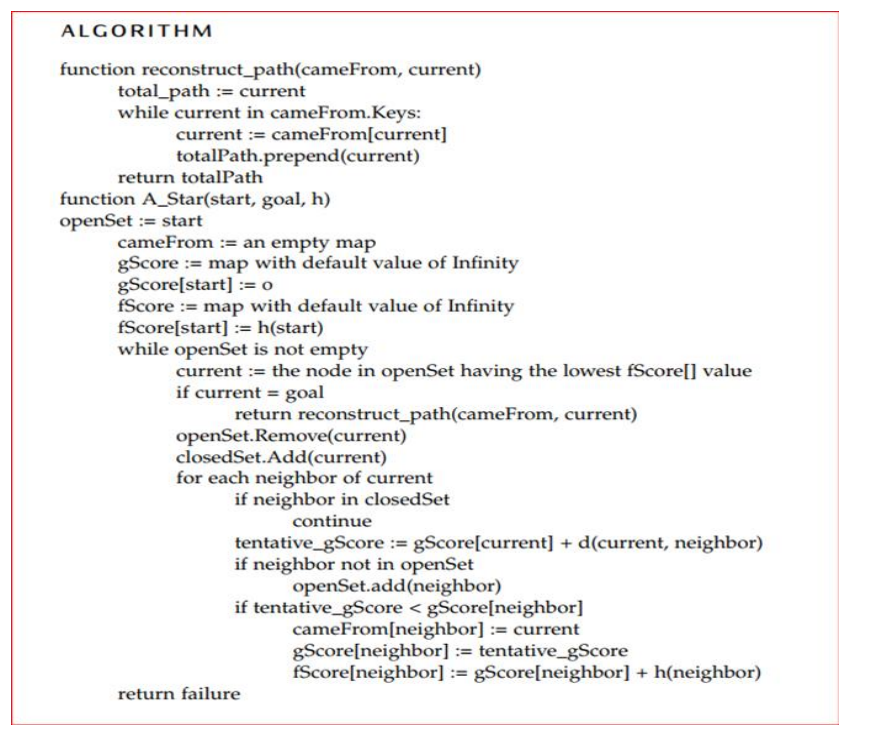


❖ **A\* Search**

One of the most widely known form of best-first search is called A\* search. It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

**f(n) = g(n) + h(n)**

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have f(n) = estimated cost of the cheapest solution through n Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of g(n) + h(n) This is the pseudo code of the algorithm:



# DESCRIPTION OF THE PROGRAM

❖ Problem formulation:

➢ PEAS formulation:

• Performance measure: Time, Node

• Environment: Maze containing a white dot, four ghosts

• Actuators: Arrow keys

• Sensors: Game screen

❖ Types of environments:

➢ Fully observable

➢ Deterministic

➢ Multi - Agent

➢ Continuous

➢ Episodic

➢ Dynamic

❖ Type of agent:

➢ Pac-Man: Goal-based agent

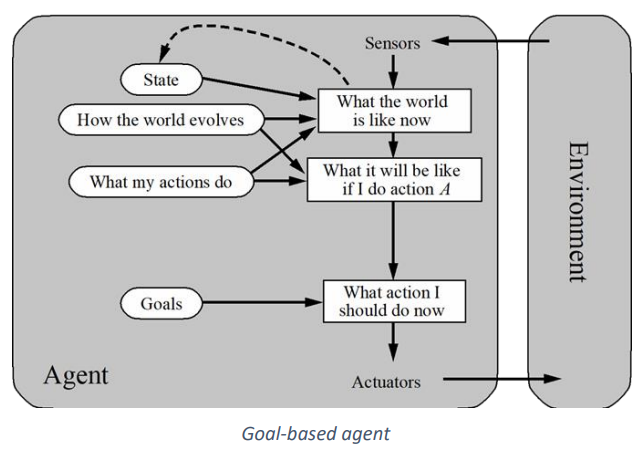
• Knowing about the current state is not always enough

to decide what to do

• Situations that are desirable must be specified (goal)

• Usually requires search and planning to find action

sequences achieving goal



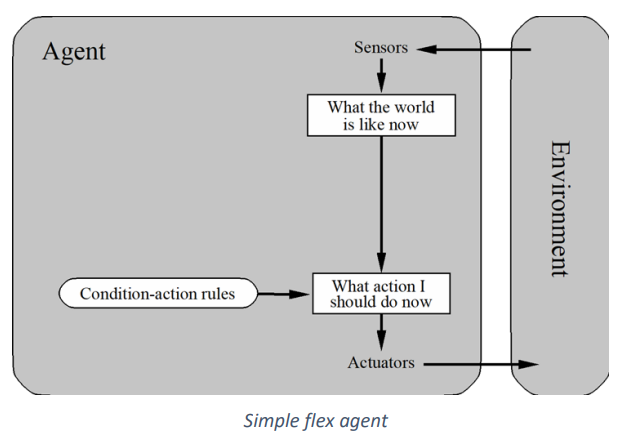
➢ Ghost: Simple flex agent

• Simple, but very limited intelligence

• Works only if the correct decision can be made on the

basis of the current percept (fully observability)

• Infinite loops in partially observable environment

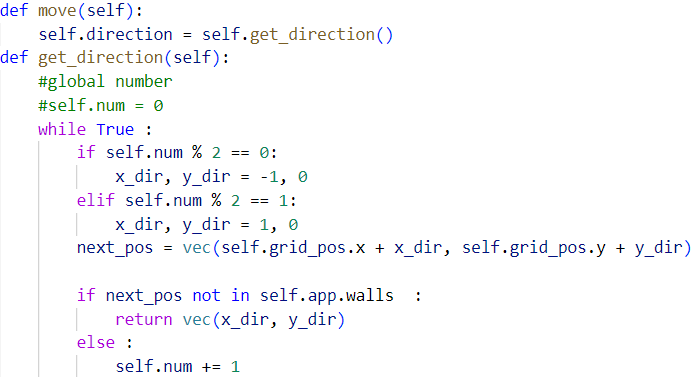


* 1. Main object

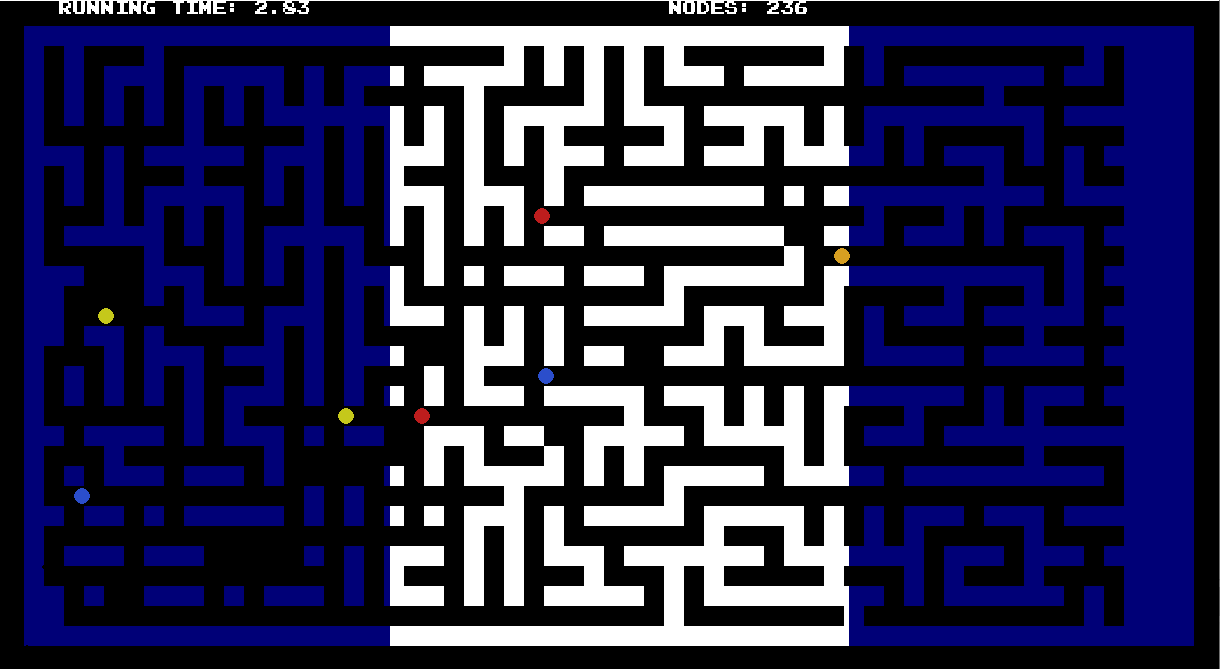
There are two main objects in this program which are

class Player and class Enemy.

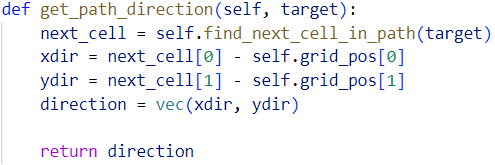
class Enemy are only movable obstacles, so we set them to move only horizontally by the wall.



The special thing is that the player knows the enemy's movements and can completely predict their moves. When the player touches the enemy, the game is immediately lost. We have created 7 randomly emenies in the file text.txt to have a difficult level for the player

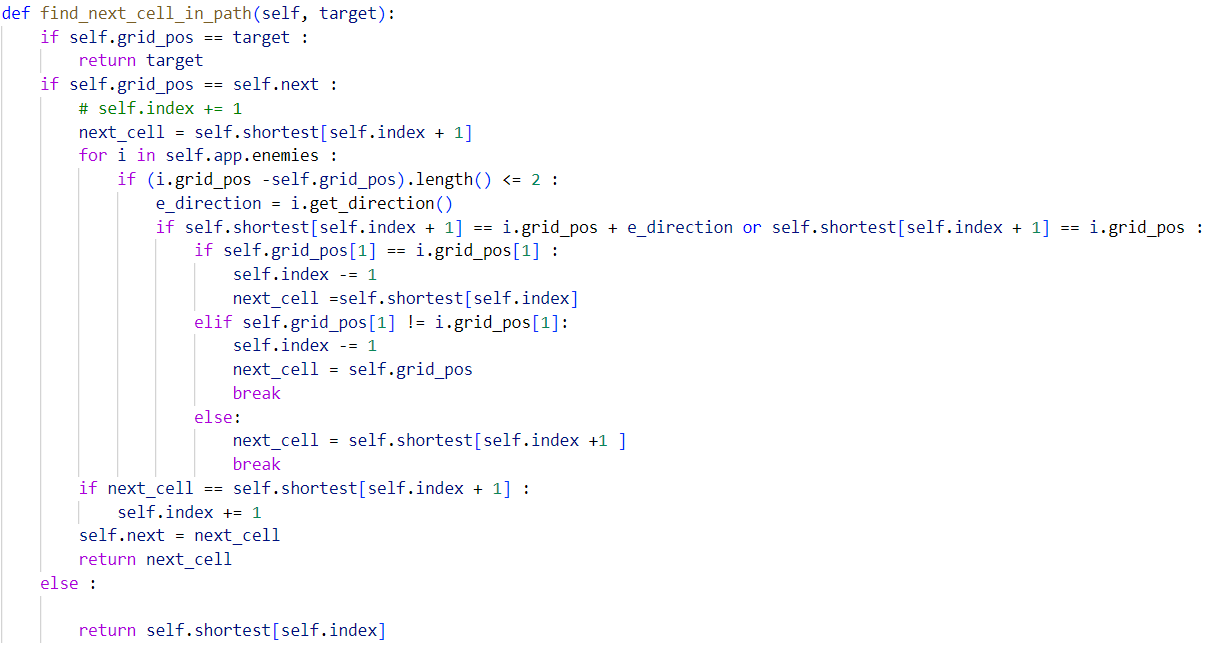


class Player has the function of setting move:



*self.grid\_pos is the position of player at that time*

The ways to avoid enemies and at the same time find the shortest path to the destination.



This object has the function of setting moves, ways to avoid enemies and at the same time find the shortest path to the destination. This class is set up to run 2 algorithms BFS and A\* in 2 similar environments, from which we can easily compare the similarities and differences of these 2 algorithms.

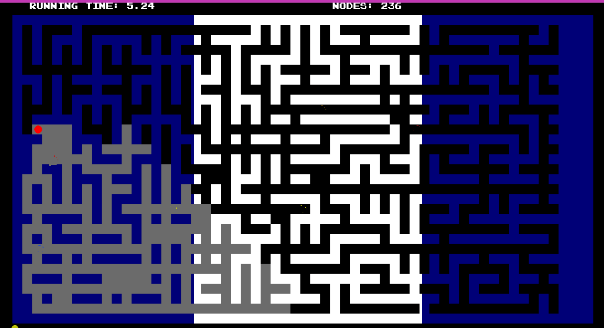
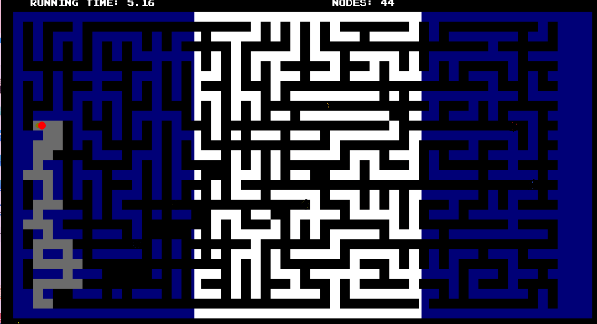
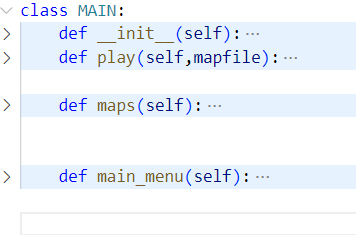
 

Figure : BFS Figure 2: A\*

BFS will traverse 4 sides around the player's current position, put those tiles in a queue array, and will check for its existence in the queue. If it does not appear, it will update the current position to that cell. As for the A\* function, we use the heuristic Euclid Distance to evaluate the priority. then browsing and adding to the queue is similar to BFS, except that when the node position is closer to the destination, it will add that node to the beginning of the queue and if it is farther from the destination, then you add and end the queue. However, because of the appearance of the enemy, it is necessary to add a method to avoid the enemy when knowing the enemy's position and way. That's why we created a method find\_next\_cell\_in\_path() to avoid collisions between enemies and players**.** There, if an enemy gets close to the player in the horizontal direction, the player will automatically move back 1 node until they cannot retreat. In case the player moves towards an enemy running parallel to the front, the player will be assigned to stand back and wait for the enemy to pass and then proceed to continue.

* 1. Asistance class

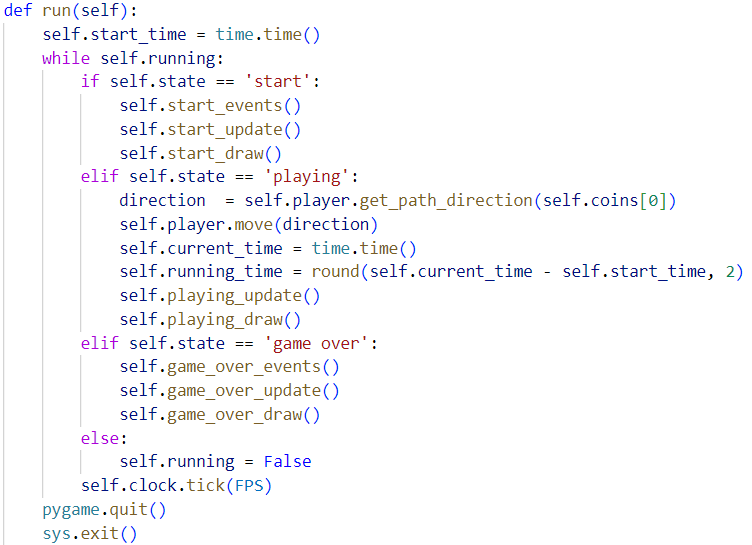
class MAIN: This class is the body of the program, is the menu that displays the game modes with 1 Play mode as the default. 2 Map is to choose the display modes of special situations that we choose to demo.



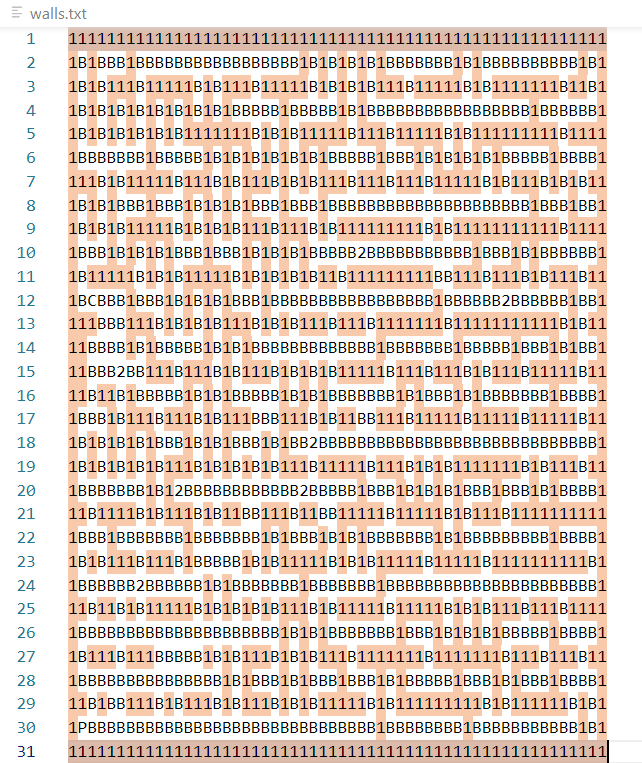
class App: This class is the association of the program with the main classes. Create loops to run the program. Specifically, the def load(self,mapfile): method will initialize the program, browse the map information in the map txt file to initialize.



Then the def run(self): method will run to create a small loop. Look at Player states to decide whether to run the BFS or A\* algorithm.



* 1. Execution process



*1: wall B: path 2: location of the enmy*

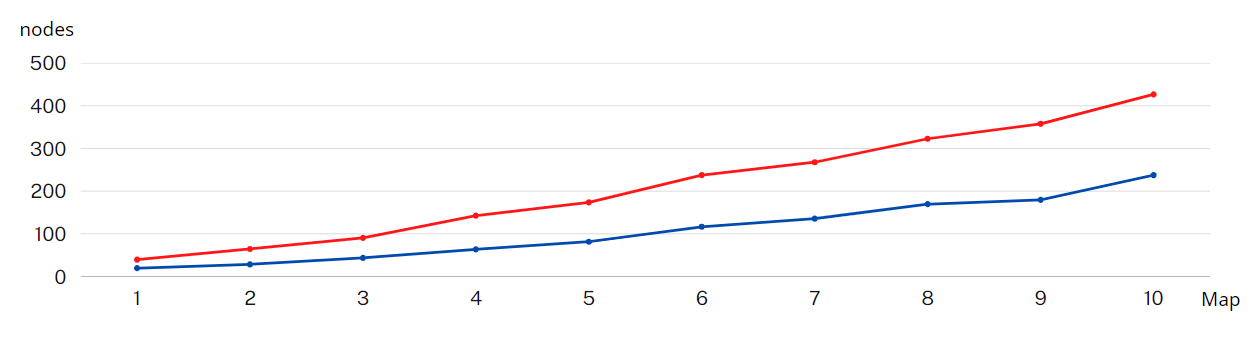
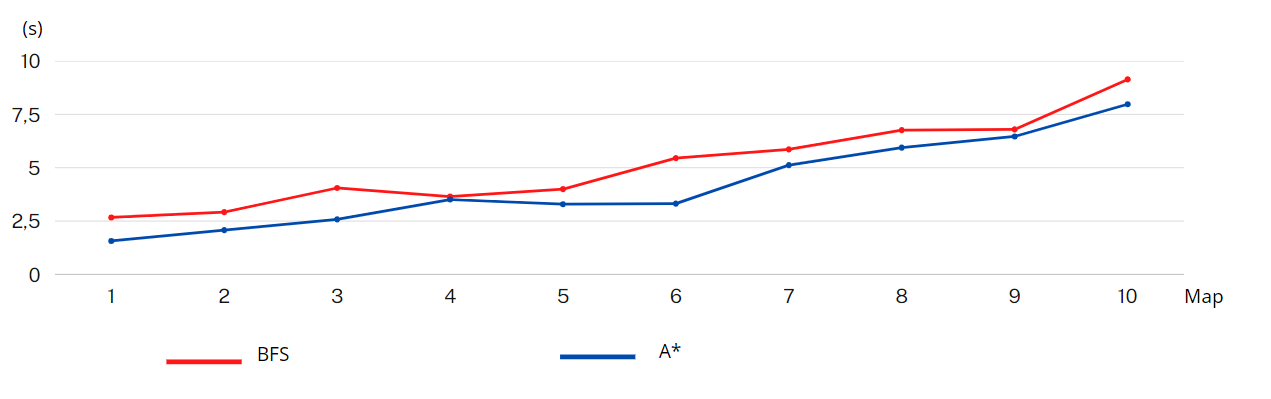
*P: player's starting position C: target position*

Firstly, we create the map by browsing the wall.txt file then we create a grid matrix to store the information. We use the functions available in the pygame library to set up the game's framework and a user-friendly interface. The next step is to pass the objects to the main class App to run the main program. Initialize a loop to run 2 states 'BFS' and 'A\*' simultaneously on the same environment. The loop will be run until the algorithm finds the path to the destination self.coins = [] .

# COMPARE

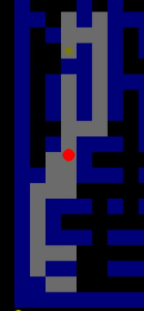
After setting up 5 maps, we ran the program and got this result. BFS algorithm runs more time than A\* algorithm. At the same time, the number of nodes that BFS saves for browsing is also much larger than that of the A\* algorithm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MAP | NODE | | TIME(s) | |
| BFS | A\* | BFS | A\* |
| 1 | 40 | 20 | 2.68 | 1.58 |
| 2 | 65 | 29 | 2.93 | 2.09 |
| 3 | 91 | 44 | 4.06 | 2.59 |
| 4 | 143 | 64 | 3.66 | 3.52 |
| 5 | 174 | 82 | 4.01 | 3.3 |
| 6 | 238 | 117 | 5.46 | 3.33 |
| 7 | 268 | 136 | 5.87 | 5.13 |
| 8 | 323 | 170 | 6.77 | 5.95 |
| 9 | 358 | 180 | 6.8 | 6.47 |
| 10 | 427 | 238 | 9.14 | 7.98 |

From the average results, we can conclude that A\* Search

algorithm will be faster and more optimal than BFS. The reason is because BFS extends to all sides, while A\* prioritizes expansion near coinThat proves the limitation in terms of time and capacity consumption of BFS. This happens because BFS will traverse all the surrounding nodes to expand without a specific direction. It opens in a circle with a larger until the destination is within that circle, then stops. This has the advantage that BFS will always find the shortest path to the destination with any different map structure. The drawback is that the amount of memory used to store it will be too much when you need to traverse a large map and it will be very time consuming. However, in some cases, the distance found by A\* is not the shortest distance found by BFS. Any deviation due to the hueristic function of A\* is misleading in itself.



In this case BFS runs perfectly fine and sees the shortest path. But A\* chooses to take a path that according to the calculated Heuristic function is shorter. Specifically, it chose to browse to the right instead of the left with a shorter path, which is the limitation of A\*.

In summary, the advantage of the BFS algorithm is that it is easy to program, always finds the shortest path regardless of the map, does not fall into a loop error. The drawback is that it consumes the resources of the computer and the user when doing it. The path cannot be found due to the large number of nodes. and for A\*, the search time is faster than that of BFS. The number of storage nodes is also much less, an extremely optimal algorithm. However, there is still a deviation when finding the shortest path due to the limitation of the heuristic function.

# ASSIGNED TASK

|  |  |
| --- | --- |
| Member | Task |
| Nguyễn Lê Hùng | Create the App() class (main) and perform the project.  Write report |
| Nguyễn Hợp Phú | Create the App() class (sub) and create the wall file. Setting the GUI for program(main) |
| Nguyễn Ngọc Đăng | Create the Player() class and improve the algorithm |
| Trần Quang Nam | Create Enemy() class and GUI (sub) for program. |

# REFERENCES

• Lecture: IT3160E Introduction to Artificial Intelligence/Asoc.Prof. Than Quang Khoat

• [How to create a vector in Python using NumPy - GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-vector-in-python-using-numpy/#:~:text=In%20order%20to%20create%20a%20vector%2C%20we%20use,way%20is%20using%20np.array%20with%20the%201-D%20list.)

• [Python GUI - GeeksforGeeks](https://www.geeksforgeeks.org/python-gui-tkinter/#:~:text=Python%20offers%20multiple%20options%20for%20developing%20GUI%20%28Graphical,a%20GUI%20using%20tkinter%20is%20an%20easy%20task.)

Content

s

**TABLE OF CONTENTS**

[INTRODUCTION 1](#_Toc108634894)

[1. PROBLEM DESCRIPTION 2](#_Toc108634895)

[2. ALGORYTHM 2](#_Toc108634896)

[3. DESCRIPTION OF THE PROGRAM 4](#_Toc108634897)

[4. COMPARE 10](#_Toc108634898)

[5. ASSIGNED TASK 12](#_Toc108634899)

[6. REFERENCES 12](#_Toc108634900)