Group 14

AI CAPSTONE PROJECT

PACMAN

INSTRUCTOR: Asso.Prof, Than Quang Khoat

Student names:

Nguyen Ngoc Dang 20200149

Nguyen Hop Phu 20205165

Tran Quang Nam 20200427

Nguyen Le Hung 20205156

CONTENT

Introduction
PacMan's game

Analysis Agent

Demo & Static

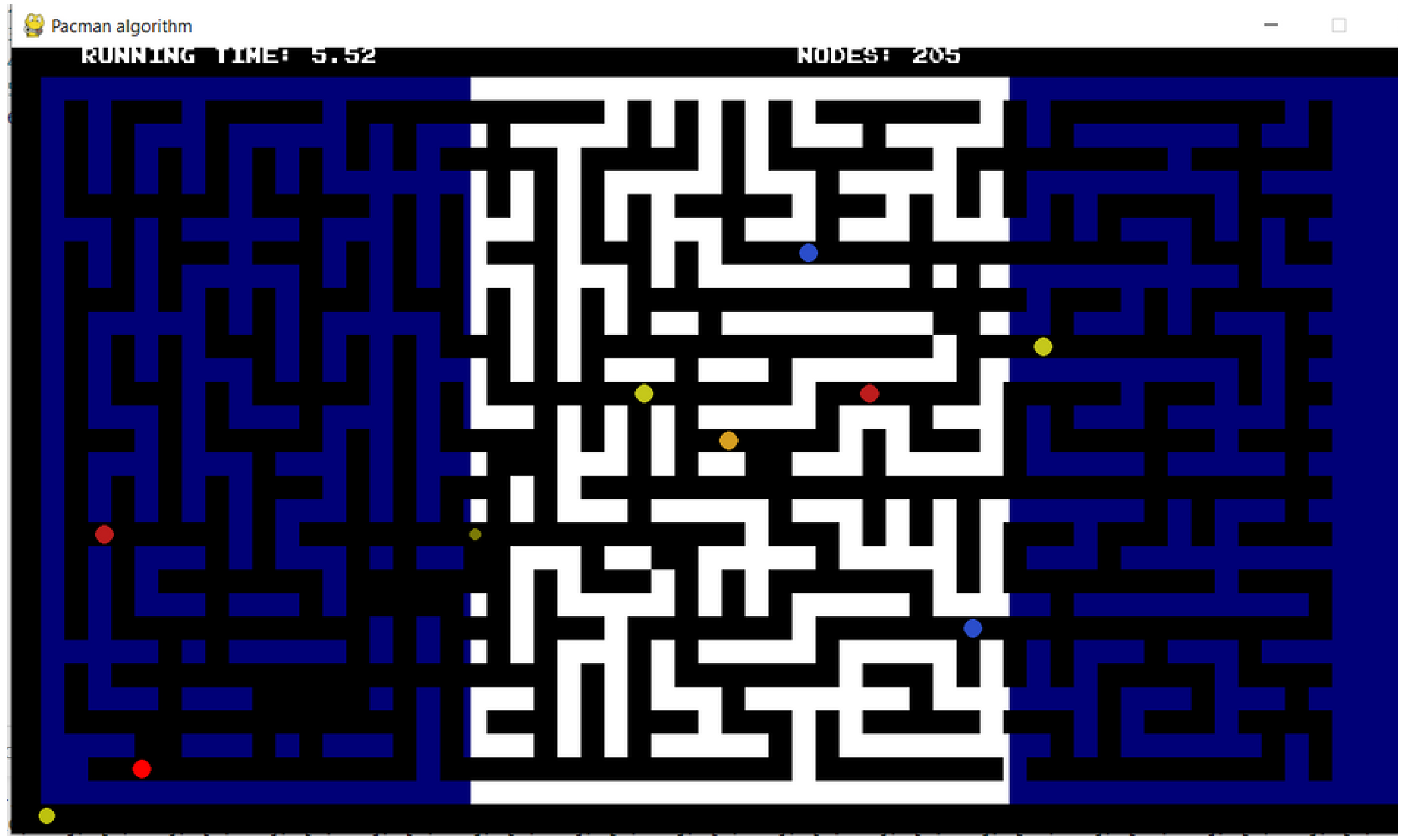
I. Introduction PacMan’s game

- Pacman is a famous Atari game developed back in 1979 by a nine-persons team
- Pacman was released in 1980 by the former Japanese developer and publisher of arcade video games Namco.
- The great success the game had at the time, made it survive until the present days becoming one of the greatest and influential video games of all time

I. Introduction
PacMan's game

Our program has:

- 30x55 matrix maps
- 7 ghosts
- Same coin per 2 round
- A Pacman

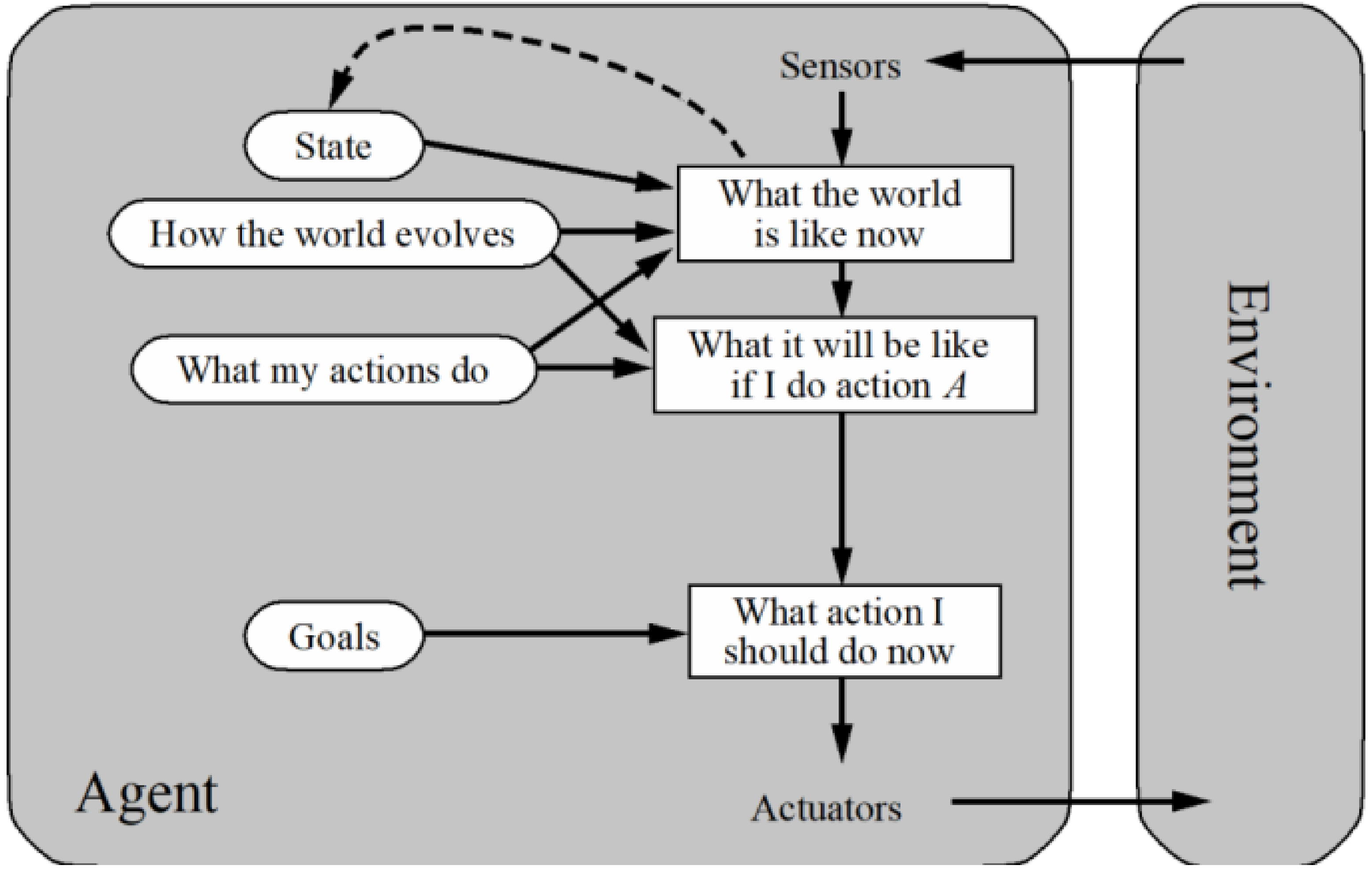


THE PEAS FORMULATION

- **Agent**: Pacman
- **Performance measure**: Execution time, number of steps to win the game
- **Environment**: Maze containing a random white dot, four ghosts
- **Actuators**: Arrow keys
- **Sensors**: Game screen

TYPES OF AGENT

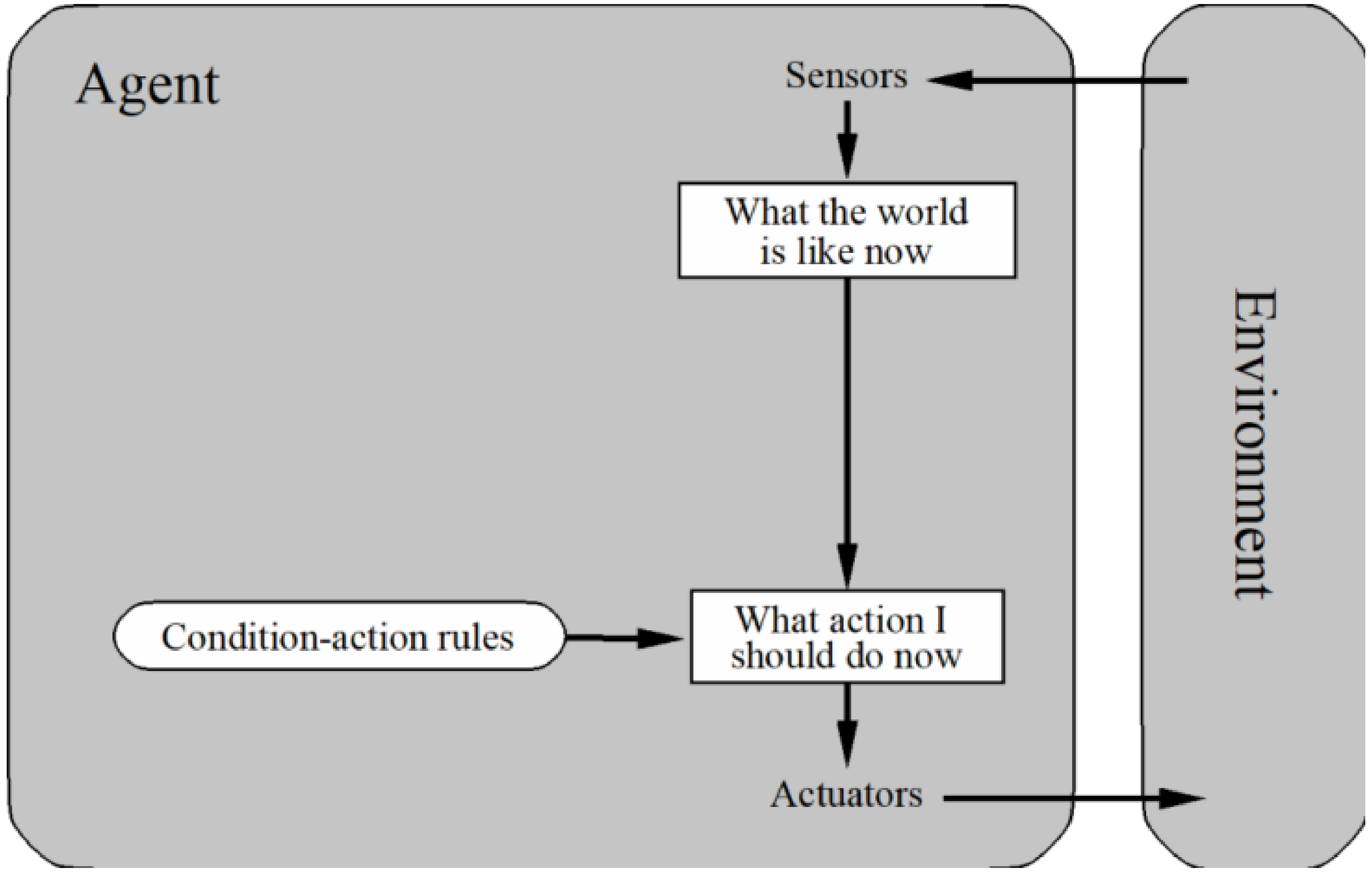
Pacman :
Goal-based agent



TYPES OF AGENT

Pacman :
Goal-based agent

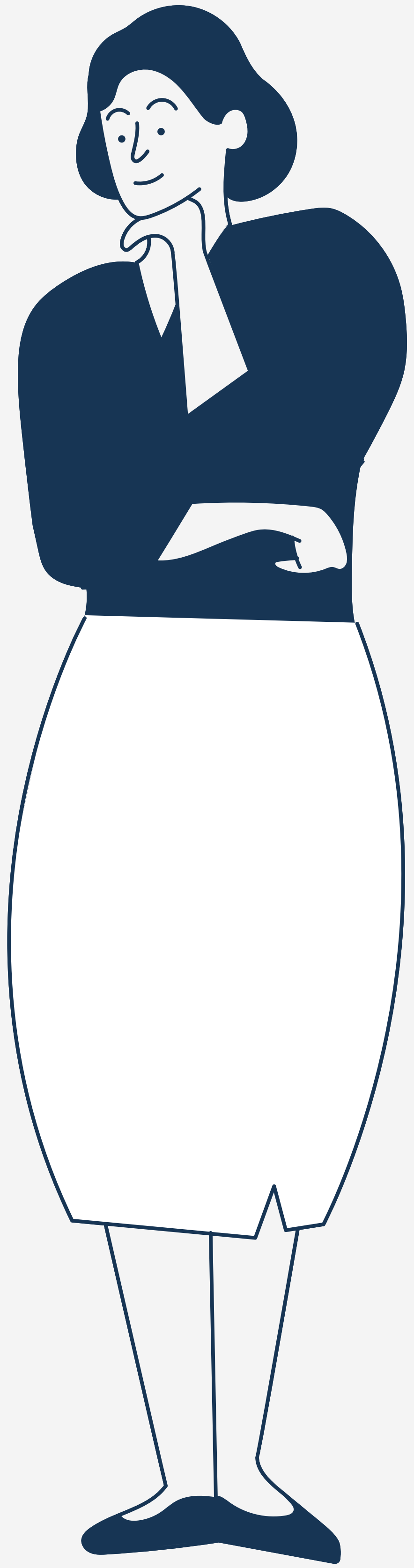
Ghost :
Simple flex agent



PROBLEM

How can Pacman eat coin without encountering ghosts if

- The movements of the ghost are pre -defined
- The movements of the ghost do not depend on Pacman's actions
- Pacman knows everything about the movement "rules" for the ghost



SOLUTION

BFS

Is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level

SOLUTION

A* SEARCH

Is a graph traversal and path search algorithm, which is often used in many fields of computer science due to its completeness, optimality, and optimal efficiency

BFS

❖**Breadth first search** is a level by level search. All the nodes in a particular level are expanded before moving on to the next level. It expands the shallowest node first. It uses a queue data structure to maintain a list of all the nodes which have been expanded.

❖ **COMPLEXITY**

• The time complexity of Breadth First Search is :

O(b^d**)** where b is the branching factor of the search tree and d is the depth at which the shallowest goal node is situated.

• The space complexity of Breadth First Search is :

O(b^d **)** i.e, it is dominated by the size of the frontier.

❖ **COMPLETE**

• BFS is complete

❖ **OPTIMAL**

• BFS is optimal

ALGORITHM

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element
explored ← an empty set
loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /*chooses the shallowest node in the frontier*/
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem,node,action)
        if child.STATE is not in explored or frontier then
            if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
            frontier ← INSERT (child,frontier)
```

A*

❖ **A* search** falls under the category of informed search strategy. This kind of search is also called best first search. A* search evaluates which node to combine using g(n) i.e, the cost to reach the node and h(n) i.e, the cost to get from the current node to the goal node, represented as :

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

❖ **COMPLEXITY**

The time complexity of A* depends on the heuristic. In the worst case of an unbounded search space it will be : O(b^d) where b is the branching factor and d is the depth at which the solution resides in the tree.

- The space complexity of A* is : O(b^d)

❖ **COMPLETE**

- BFS is complete

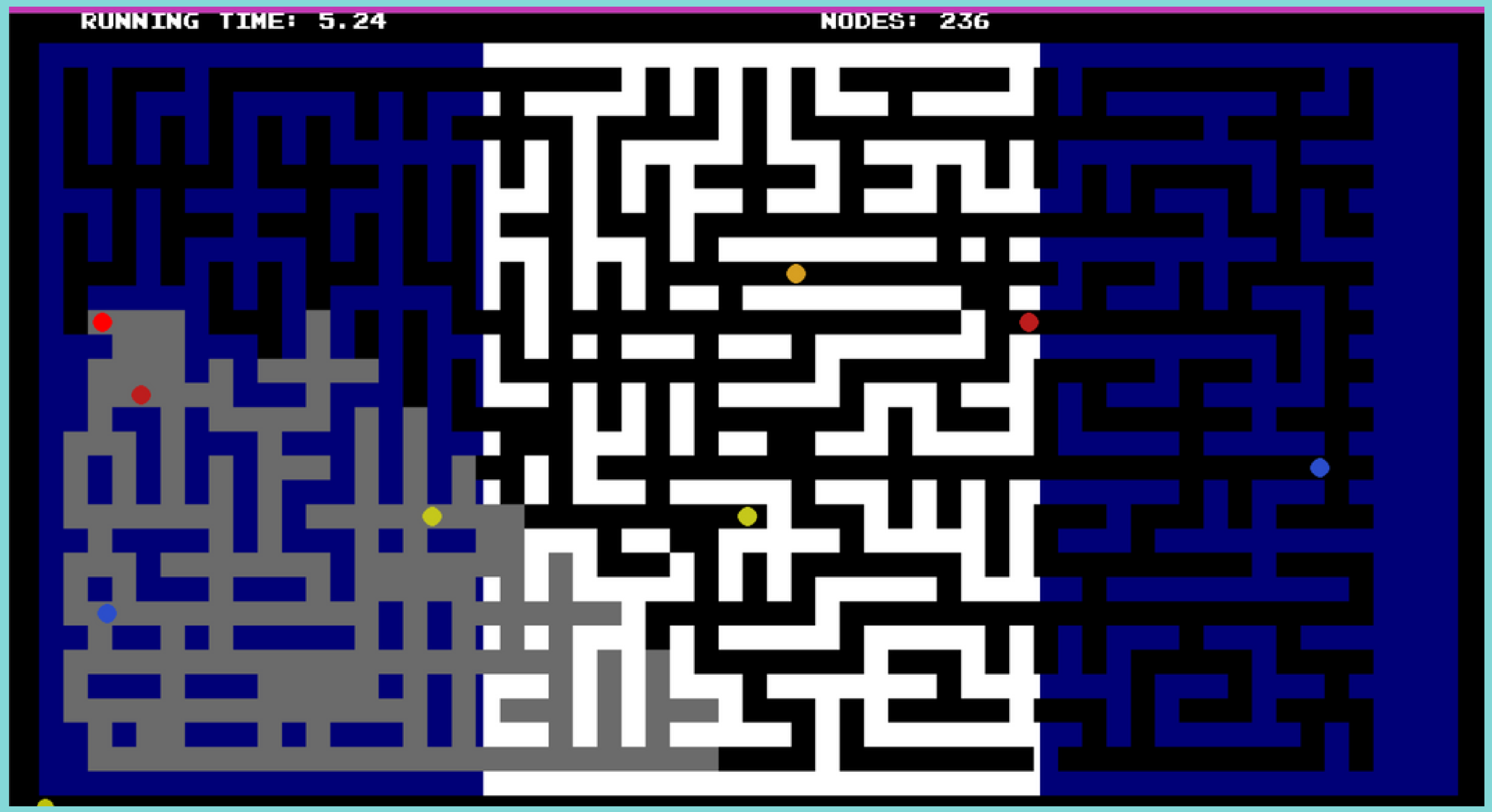
❖ **OPTIMAL**

- BFS is optimal

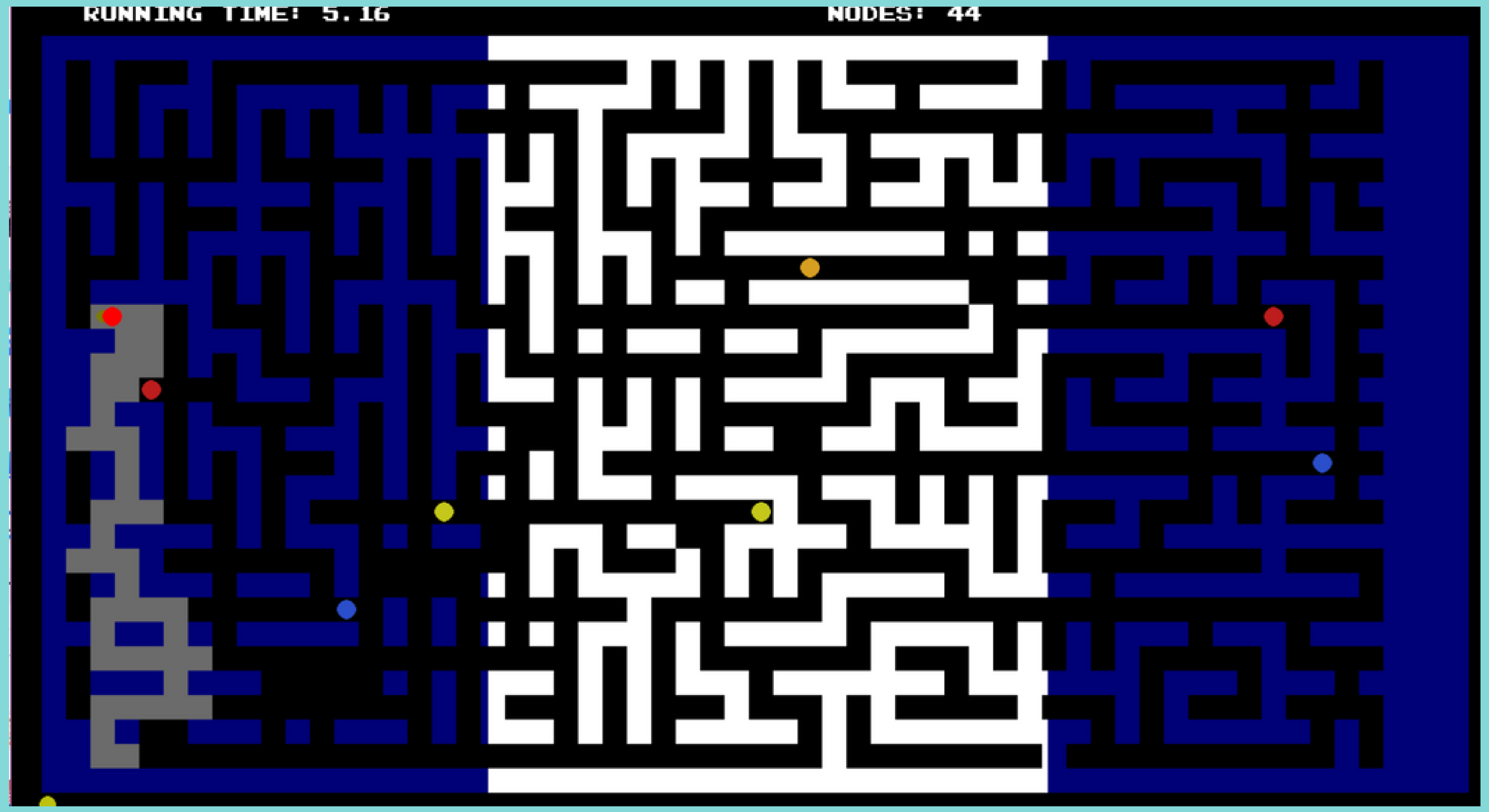
ALGORITHM

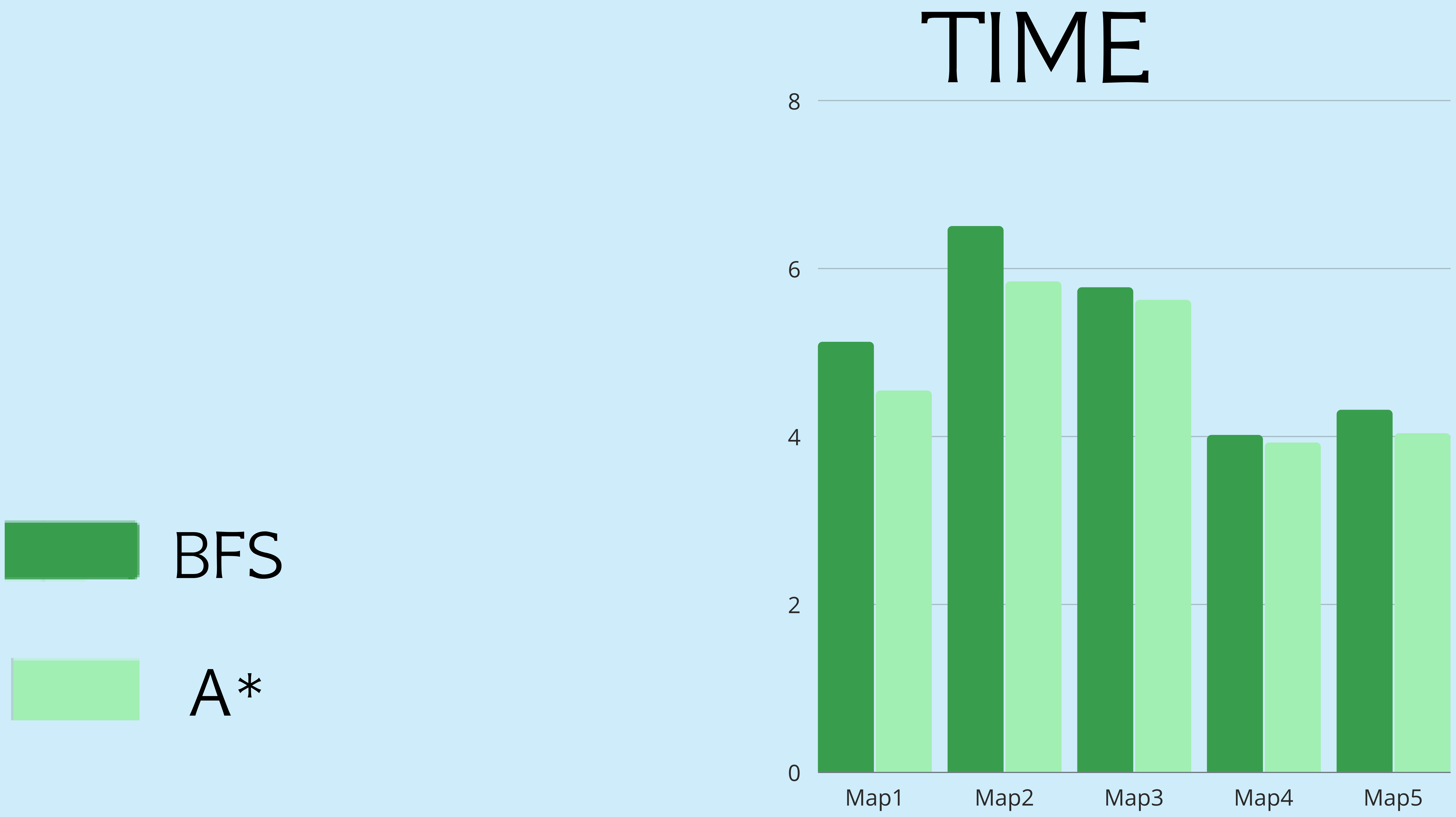
```
function reconstruct_path(cameFrom, current)
        total_path := current
        while current in cameFrom.Keys:
                current := cameFrom[current]
                totalPath.prepend(current)
        return totalPath
function A_Star(start, goal, h)
openSet := start
        cameFrom := an empty map
        gScore := map with default value of Infinity
        gScore[start] := 0
        fScore := map with default value of Infinity
        fScore[start] := h(start)
        while openSet is not empty
                current := the node in openSet having the lowest fScore[] value
                if current = goal
                        return reconstruct_path(cameFrom, current)
                openSet.Remove(current)
                closedSet.Add(current)
                for each neighbor of current
                        if neighbor in closedSet
                                continue
                        tentative_gScore := gScore[current] + d(current, neighbor)
                        if neighbor not in openSet
                                openSet.add(neighbor)
                        if tentative_gScore < gScore[neighbor]
                                cameFrom[neighbor] := current
                                gScore[neighbor] := tentative_gScore
                                fScore[neighbor] := gScore[neighbor] + h(neighbor)
        return failure
```

BFS

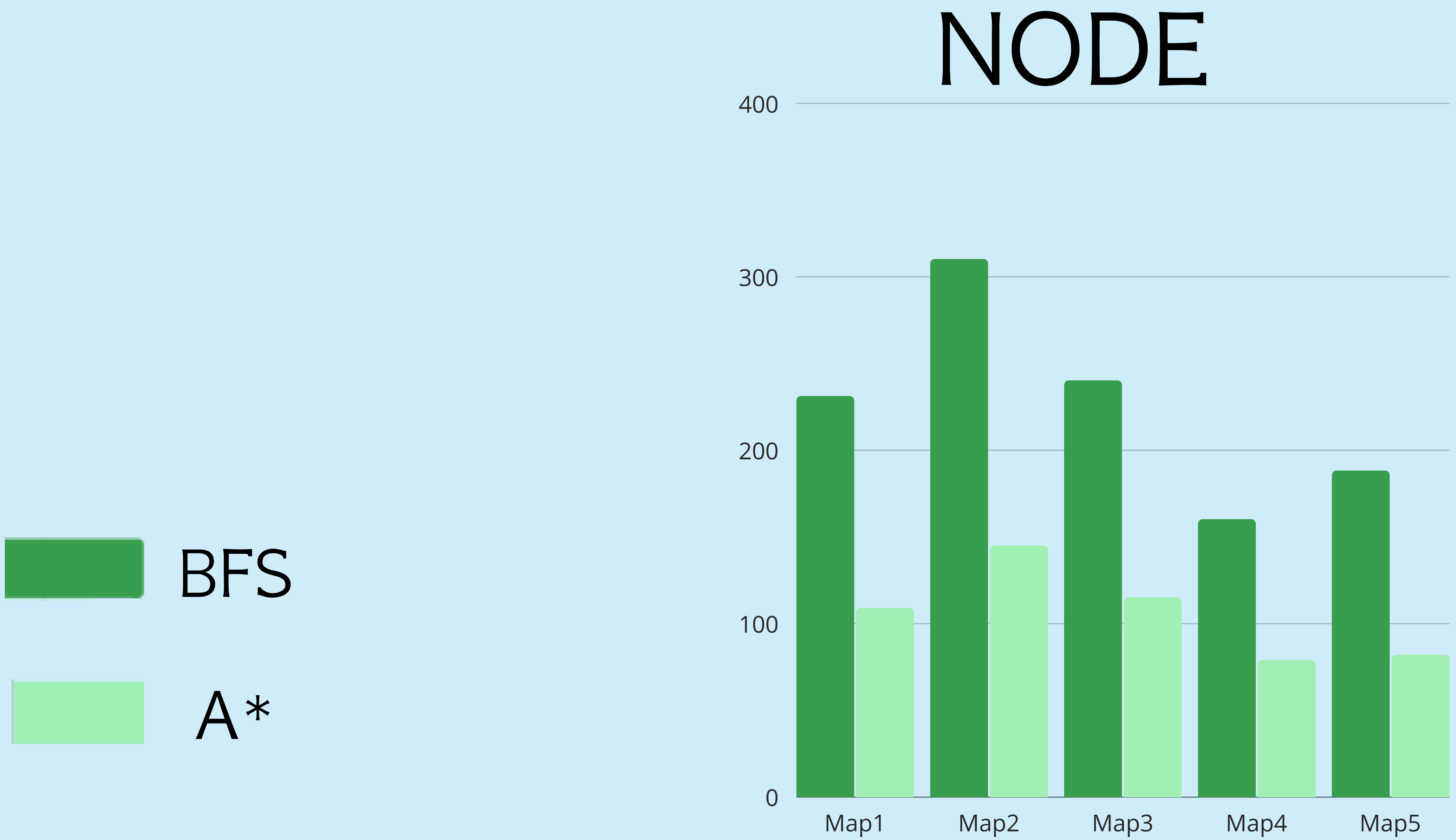


A*





TIME
8
6
4
2
0
Map1
Map2
Map3
Map4
Map5
BFS
A*



NODE
400
300
200
100
0
Map1
Map2
Map3
Map4
Map5
BFS
A*

BFS

The evaluation function

Consider some observation about the environment:

- The distance to the closest dot
- The distance to the closest ghost

=> **The value of a state**

Comment

Advantage:

- Simpliciity
- Always find out the shortest path
- Behaves pretty well in most mazes

Disadvantage:

- Take a lot of time
- Take a lot of memory

A*

The evaluation function

Consider some observation about the environment:

- The distance to the closest dot
- The distance to the closest ghost

=> **The value of a state**

Comment

Advantage:

- Optimize
- Always find out the shortest path

Disadvantage:

- DIfficutlt to design, somtime be stucked in loop trouble
- Some situation cannot find the shortest path

SOME OTHER ERRORS

Complexity of 2 algorithm O(b^d)
-> Difficult to find a solution in case the coin is placed too far from the pacman.

SOME OTHER ERRORS

Complexity of 2 algorithm O(b^d)
-> Difficult to find a solution in case the coin is placed too far from the pacman.

A* is highly stubborn; it will defy the quickest path.
-> When it becomes trapped in the terrain, it will be tough for A* to find a way out.


THANK YOU