**1.Presentation of the subject:**

**Topic: Car in Maze**

Environment: A maze is a matrix with size , which is randomly generated. The elements of the maze can be:

* Walls
* Some clear cells containing a random score in range . If the car moves to this cell, it will then become a cell with no score.
* Other clear cells with no score

The starting points and finish points are randomly initialized among the clear path cells: their coordinates are denoted as and with being the top-left corner of the matrix. The maze is set that there exists and exists only one shortest way from a point to another point in the maze. The car wants to get out of the maze with the final highest score, which is computed by dividing the total score it earns by the total steps it takes to get out of the maze.

The car agent can perform the following actions:

* Move forward
* Turn left
* Turn right
* Get the score (the car automatically takes the score when it moves to the cell with score, doesn’t cost another step to take it)

The program will have several outputs:

* Number of steps needed for the agent to get out of the maze and total score it earns through the paths
* The final score of the agent
* The percept sequence and corresponding actions of the agent.
* Execution time

**2.Description of the problem:**

- Detailed description purpose of the problem: (What kind of problem you are dealing with? What are the specificities of your problem?)

This is an extension of the maze problem. Instead of simply solving the maze, we added some score cells randomly and the problem became: get out of the maze and maximize the final score. This makes the problem more complicated, because the car agent now will have more roads to consider, not just finding the shortest way from start to end anymore. The car agent now can go a longer road to take a score in a cell, then come back and reach the end, this make it has more steps to go, but if that score is worth it, the final score will be greater than the normal road.

Specificities of our problem: with increasing maze size and the number of score cells, the problem become significantly difficult, because we have to consider much more cases (branching factor become hundreds). But with our algorithm, we can expand the maze to 100x100, and increase the number of score cells to 200! Here is an example of a big maze that we can solve

**A picture containing text, newspaper

Description automatically generated**

- PEAS formulation:

Performance measure:

* Get out of the maze
* Maximize the final score

Environment: Maze with:

* Wall cells
* Clear cells with score
* Clear cells with no score

Actuators

* Normal actuators to move: steering wheel, accelerator, brake, signal, horn, …
* Special system for auto collecting the score

Sensors

* Cameras to see 8 adjacent cells
* Special sensor to detect a cell with score
* Type of environment:

***Partially observable*** because the car agent is only able to see 8 adjacent cells

***Deterministic***, because the next state of the environment is completely determined by:

• the current state

• the action executed by the agent

***Single agent***: only one car agent

***Static***: The environment is unchanged during the time where the agent is making its next decision

***Discrete***: A limited number of distinct, clearly defined percepts and actions

***Episodic***: The agent's experience is divided into atomic "episodes", next episode does not depend on the actions taken in previous episodes

* Type of agent: Problem-solving agent

**3. Explaining the choice of algorithms to be used for solving the problem, and their parameterization**

**3.1 Maze generator:**

The maze is a matrix with size m x n where m, n is odd numbers.

First, we create a matrix with the properties: all (x, y) cells where x, y are odd numbers will be clear cells, the other will be wall cells.

Then, we randomly choose a clear cell. From that cell, we consider all clear cells that adjacent to the same wall cell with it. If that clear cell is

Our first approach to the problem is that we noticed that the maze has only one path without coming back from a cell to another, so when we found the shortest path from start cell to end cell (we called this “main path”), then the final path (the path with highest final score) must be based on this main path with additional sub-path to go to a score cell and then come back.

This is an example to make it easier to understand: A picture containing diagram

Description automatically generated

In our solution, we divide the problem into two sub-problems:

* In the first problem, we “discover” the maze, means that we find all simple paths from the start cell to the end cell or a clear cell with score
* In the second problem, we add the sub-paths to the main path to get the final path. This has some main tasks:
* First, from those paths in the first problem, we cut the main path off to get all the sub-paths (meaning that path go from nearest cell in the main path to a score cell).
* Then, we try to add these sub-paths to the main path. The final result will be the main path adding some sub-paths that maximize the final score

In the above example, the optimal path will be the main path adding the Sub2 (a sub-path) to have the highest final score.

**3.1.** **First part: Discover the maze**

In this part, we choose to use 3 algorithms: Depth-First Search, Breadth-First Search and A\* Search.

(The stop condition of 3 algorithm is that there are no cells containing score or the end cell)

***For Depth-First Search***, we use a list to implement. At the beginning, the list has only the start cell’s coordinate. Then, at each step, we remove and take from the list the last element and consider its valid directions that the car can continue (the directions towards the cells that is not walls or cells that already has coordinate in the list. If there exist some valid directions, we will add the corresponding coordinate, otherwise, we do not do anything and go to the next step.

***For Breadth-First Search***, we use a list to implement, let call it list A. At the beginning, the list has only a list of start cell’s coordinate. At each step, we remove and take the first element of list A, let call it list B. Then, with each valid direction of the last coordinate of list B, we create a list by adding the coordinate correspond to the direction and add to list A.

***For A\* Search,*** we use a list to implement, let call it list A. At the beginning, the list has only a list of start cell’s coordinate. At each step, we remove and take a list B that has minimal value g. Here, g = f + h, where f is equal to the length of the path correspond to list B and h is the heuristic function based on Manhattan distance of the last element of list B to the nearest scored cell of the end cell.

**3.2. Second part: Adding sub-paths to the main path to get the optimal solution**

In this part, we choose to use 2 methods: Brute Force and Best First Search algorithm.

**3.2.1 Brute force algorithm**

To add the sub-paths to the main path, our first try is do it by brute force. Like its name, we will gradually add a sub-path to the main path, and try all the combinations of sub-paths to get the final solution

**3.2.2 Best First Search algorithm**

The brute force will have significant running time when we increase the maze size and the number of points, as it will have much more sub-paths. So, we attempted to think another way and obtain a solution using the Best First Search algorithm. In this way, we will gradually add a priority sub-path among all sub-paths to the main path.

**- Finding the priority sub-path:**

We will prove that: When adding the sub-paths to the main path, if the main path has the highest value (scores/steps) then it is the optimal path, else we will gradually add the sub-path with the highest value to the main path until it has the highest value.

In other words, we will prove that, if a sub-path has the highest value among all paths (all sub-paths and the current main path), it must be in the final path.

Let be the total scores in the main path, is the number of steps in the main path, is the number of all sub-paths that we will consider

is the total scores and steps in the subpath, respectively.

Consider all the .

* If is the highest value, i.e. :

Whenever we add a sub-path to the main path, assume it is sub-path with

Our path now will have the value . This is smaller than , as:

Hence, if is the highest value, we cannot add a sub-path to have a higher value

* If is not the highest value:

Assume we have found the optimal value for the problem:

(Which means we have found sub-path that makes the final value be highest)

As is not the highest value, then there exists sub-path that has highest value . We prove the sub-path must be in sub-paths of the final solution

We prove this by the method of reduction to absurdity

Assume , which means the sub-path is not in sub-paths of the final solution

Among sub-paths of , there exists a sub-path that has the highest value

Hence, when adding other sub-path to to have the value will lead to

Since is the highest value among all sub-path, so

Therefore, is not the optimal value (because we can still add sub-path into it to make a higher value). So, our initial assumption is not true

Consequently, if is the highest value then sub-path must belong to sub-paths in the final solution

…

**4. Implementing the algorithms to be used for solving the problem**

**4.1. Implement the algorithms to generate the maze:**

- Ban đầu tạo mê cung 1 cách ngẫu nhiên nên mê cung rất k bình thường.

- Vì thế nên đã tham khảo những mê cung trên mạng, đọc hiểu code của ng ta..

- Tuy vậy nhưng mê cung vẫn rất đơn giản, .. cần phải cải tiến.

- ..

**4.2. Implement the algorithms for the second part:**

- Ban đầu không thể áp dụng những thuật toán đã học 1 cách trực tiếp. Cho dù là BFS, DFS thì cũng gần giống như liệt kê

- Nên cả nhóm nghĩ tới việc sử dụng những thuật toán dùng heuristic nhưng có vẻ không khả thi vì không tìm dc hàm đánh giá

- Nên đã chuyển hướng sang việc giải quyết 1 cách toán học rồi áp dụng vào code..

|  |  |  |
| --- | --- | --- |
| Tasks | Member 1 | Member 2 |
| **Programming** |  |  |
| DFS |  |  |
| BFS |  |  |
| A\* |  |  |
| Brute force |  |  |
| Nghĩ ra ý tưởng thuật toán |  |  |
| Chứng minh thuật toán |  |  |
| Implement thuật toán |  |  |
| Running random instances |  |  |
| Plot data |  |  |
| GUI |  |  |
| **Analytic** |  |  |
| Task 1 |  |  |
| Task 2 |  |  |
| Task 3 |  |  |
| Task 4 |  |  |
| Task 5 |  |  |
| Powerpoint |  |  |
| Demo video |  |  |

|  |  |
| --- | --- |
| Member | List of tasks |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

<https://scipython.com/blog/making-a> maze/?fbclid=IwAR1L9KWKoXo2pneBQnJX6mk9FpRyA5Vmi5u5u6sGV\_8n6PsKLOQ-0FLEIjo