Dell Hackathon 23 API Documentation

The hackathon is divided into two phases, the first or initial phase, and a final phase.

A. Initial Phase:
   The contesters will all be trying to solve one maze, trying to rescue as much as of rescue items that fall through their exploration to exit the maze
B. Final Phase:
   In the second phase, the contester should be able to solve other teams' phases, to enter the final phase, each contester must first:
   a. Pass the initial phase
   b. Submit a "Valid" maze
   c. His agent must be able to solve his own submitted maze

APIs and Functionality

Definitions of attributes sent from agent to server:

1. **"agentId"**: The Id sent privately to each team; each identifies and is unique to the whole team, it is sent with each request to identify the team trying to solve the maze.
2. **"action"**: The move the robot or agent is trying to take, it is either one of "**N**" for north, "**W**" for west, "**E**" for east, "**S**" for south.
   The action should always be sent in a **Capital** Case, one of the 4 characters, not following the above details will result in the server replying in Wrong action.
3. **"solution"**: Solution provided whenever the agent steps over a rescue item (riddle) , it is provided through the solve API
4. **"riddleType"**: the type of riddle the agent is trying to solve, it is sent from server to agent at the move response if they step over a riddle, it should be sent back when tries to solve the riddle they are stepping over.

APIs:

The sever exposes for the user 4 main APIs:

1- Init

This API is responsible for starting the maze that the agent or contestant should be trying to solve.

- **API:** POST /*init*
- The request should contain "**agentId**"
- Example Request:

```
{
"agentId": "team_1_agent",
}
```

- The response will contain the initial state containing **position** (current position), **distances**, **directions**
- Example Response:

```
{
        'directions': [[0, 0], [1, -1], [0, 0], [1, 0]],
                'distances': [-1, 10, -1, 6],
                'position': [0, 0]
}
```

2- Move

This API is used to move to the next step in the maze.

- **API:** POST /move
- The request should contain "**agentId**" and "**action**"

- Example Request:

```
{
        "agentId": "team_1_agent",
        "action": "N"
}
```

- The response will contain the state containing **position** (current position), **distances**, **directions**, **rescuedItems**, **riddleType** and **riddleQuestion**

- Example Response without a riddle in the current position:

```
{
        'directions': [[0, 0], [1, -1], [0, 0], [1, 0]],
        'distances': [-1, 10, -1, 6],
        'position': [0, 0],
        'rescuedItems': 0,
        'riddleQuestion': None,
        'riddleType': None
}
```

- Example Response with a riddle in the current position:

```
{
        'directions': [[0, 0], [1, -1], [0, 0], [1, 0]],
        'distances': [-1, 10, -1, 6],
        'position': [0, 0],
        'rescuedItems': 2,
        'riddleQuestion': "110110010010011000",
        'riddleType': ' cipher'
}
```

3- Solve

The API is used to solve the riddle in the current step/position in the maze.

- **API:** POST /solve
- The request should contain "**agentId**" , "**riddleType**" and "**solution**"
- Example Request:

```
{
        "agentId": "team_1_agent",
        "riddleType": "cipher",
        "solution": "cipher_solution"
}
```

- The response will contain the same state as before solving attempt (you are still in the same position) containing **position** (current position), **distances**, **directions**, **rescuedItems** increased by one if riddle is successfully solved, **riddleType** with None, **riddleQuestion** with None
- Example Response:

```
{
        'directions': [[0, 0], [1, -1], [0, 0], [1, 0]],
        'distances': [-1, 10, -1, 6],
        'position': [0, 0],
        'rescuedItems': 1,
        'riddleQuestion': None,
        'riddleType': None
}
```

4- Leave

This API is used to quit the simulation; the server calculates the agent score and saves the attempt.

- API: POST /leave
- The request should contain "**agentId**"
- Example Request:

```
{
        "agentId": "team_1_agent",
}
```

- The response will be either 200 if the agent already trying to solve a current maze of 400 in case of there is no agents with the specified ID

Server Triggered Leaves

Whenever an agent calls init API, and a maze is generated, and a state is sent to the agent:

a- The agent has a time limit of maximum 15 minutes to solve the maze
b- The agent has a time limit for each move and the next one of 30 seconds
c- The agent has a maximum number of moves which is 5000 steps (5000 moves)

If any of the above conditions is broken, the agent history is saved as an attempt, and it leaves the maze.

Condition For Saving Maze Attempts

The agent's or team's attempt is only saved in the Database if he passes 50 steps inside the maze, else it won't be considered as an attempt.

Flow

A- First the agent starts by calling the init API with the agent ID.
B- The agent starts moving around the maze trying to find rescue items and how to exit the maze
C- If the agent steps on a riddle, he is allowed to solve the riddle.
D- If the agent tries to solve the riddle, the exact state is returned with information in rescued items whether he was able to solve it right or wrong
E- The agent calls leave API if he finds out that he is ready to leave, or a server trigger leave is called to exit that specific agent.
F- The score is calculated with all the agent history in the database and is saved as an attempt if it satisfies the condition for saving a maze.