Report: Dynamic Maze Game with Reinforcement Learning

---

1. Introduction

The **Dynamic Maze Game** integrates reinforcement learning and interactive gameplay to create a progressively challenging environment for both agents and users. Leveraging a **Deep Q-Learning (DQN)** framework, this project trains an AI agent to navigate a maze and dynamically increases the difficulty level, challenging users with larger grids and denser obstacles.

The game provides both a **visual experience using Tkinter** and an AI-driven learning pipeline, making it an engaging and educational exploration of reinforcement learning.

---

2. Features

1. **Interactive Gameplay:**
   - Users navigate the maze using `W`, `A`, `S`, and `D` keys to move up, left, down, and right, respectively.
   - Visual feedback provided with a dynamic **Tkinter-based GUI**.
2. **AI-Driven Agent:**
   - The agent learns optimal navigation strategies through **Deep Q-Learning**.
   - Trained to handle increasing levels of complexity with dynamic maze adjustments.
3. **Dynamic Difficulty:**
   - Grid size and obstacle density increase as levels are cleared.
   - Challenges both users and the AI agent to adapt to more complex environments.
4. **Color Coding:**
   - **Yellow:** Agent’s position.
   - **Green:** Goal.
   - **White:** Free space.
   - **Black:** Obstacles.

---

3. Algorithms Used

3.1. Reinforcement Learning with DQN

The AI agent uses **Deep Q-Learning** to learn optimal actions for navigating the maze:

1. **State Space:**
    - Represented as a flattened matrix of the maze grid.
    - Includes agent and goal positions along with obstacles.
2. **Action Space:**
    - Four discrete actions: `Up`, `Down`, `Left`, `Right`.
3. **Reward Mechanism:**
    - **+10:** Reaching the goal.
    - **-0.01:** Penalty for each step to encourage faster completion.

The Q-Learning update equation is:

$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max a' Q(s',a') - Q(s,a)]$

Where:

- Q(s,a)Q(s, a)Q(s,a): Current Q-value.
- α: Learning rate.
- γ: Discount factor.
- R: Immediate reward.
- s', a': Next state, action.

---

4. Implementation

4.1. Environment (`maze_env.py`)

The maze is modeled as a grid:

- **Reset Method:**
    - Initializes the grid with agent, goal, and obstacles.
    - Ensures the agent and goal are never at the same position.
- **Dynamic Difficulty:**
    - Grid size and obstacle density increase with each level.

Key methods include:

- `reset()`: Initializes the maze.
- `step(action)`: Updates the agent's position based on the chosen action.
- `increase_difficulty()`: Increases the grid size and obstacle density up to specified limits.

4.2. Visualization (`visualizer.py`)

The game is rendered using **Tkinter**:

- **Canvas Resizing:**
  - Dynamically adjusts to fit the current grid size.
- **Color-Coded Elements:**
  - **Yellow:** Agent.
  - **Green:** Goal.
  - **White:** Free space.
  - **Black:** Obstacles.

4.3. Training (`train_agent.py`)

Training uses the **DQN agent**:

- **Dynamic Networks:**
  - Adjusts input dimensions as maze size grows.
- **Memory Replay:**
  - Stores past experiences for batch training, improving stability.
- **Dynamic Levels:**
  - Increases maze size and complexity every `n` episodes.

4.4. User Interaction (`challenge_user.py`)

Provides interactive gameplay for users:

- Displays instructions, color meanings, and controls.
- Dynamically increases difficulty as users clear levels.
- Captures user input (`W`, `A`, `S`, `D`) to navigate the maze.

5. Dynamic Difficulty

5.1. Grid Size

Starts at `4x4` and increases up to `10x10`:

```
self.size = min(self.size + 1, 10)
```

5.2. Obstacle Density

Starts at `10%` and grows to `30%`:

```
self.obstacle_density = min(self.obstacle_density + 0.05, 0.3)
```

5.3. Reinitialization

Both the agent and environment are reinitialized to accommodate the new grid size.

---

6. Files and Responsibilities

File Name	Description
`maze_env.py`	Defines the maze environment.
`visualizer.py`	Handles the GUI rendering using Tkinter.
`train_agent.py`	Trains the AI agent using DQN.
`challenge_user.py`	Provides an interactive game mode for the user.
`dqn_agent.py`	Implements the DQN agent and replay memory.
`run_visual_game.py`	Visual demonstration of the trained agent.

---

7. Results

7.1. AI Performance

- Successfully trains to navigate mazes of increasing complexity.
- Completes mazes within fewer steps as training progresses.

7.2. User Interaction

- Engaging gameplay with dynamic difficulty adjustments.
- Visual feedback makes it intuitive to understand the game.

---

8. How to Run

8.1. Train the Agent

Run the following to train the AI:

```
python train_agent.py
```

8.2. User Gameplay

To play the game interactively:

```
python challenge_user.py
```

8.3. AI Demonstration

To watch the AI in action:

```
python run_visual_game.py
```

This report summarizes the working, algorithms, and implementation details of the **Dynamic Maze Game**. Copy this text into a document for a formatted, detailed explanation of the project. Let me know if further sections or enhancements are required!