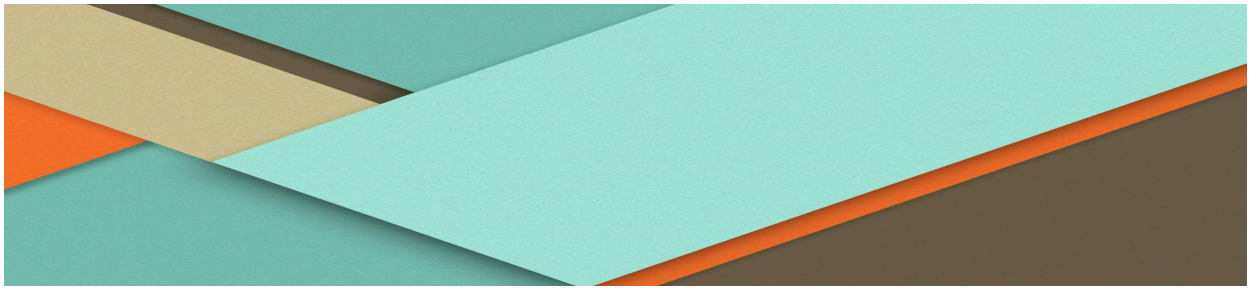


CSAI 301 - Project Phase 2

28.12.2024

—

By

Amin Gamal - 202202219

Anas Ahmad - 202202029

Ahmed Ibrahim - 202202064

Mohamed Ehab - 202201236

Autonomous Car Path Navigation - Reinforcement Learning

Overview

In the first phase of our project, we explored **informed and uninformed search algorithms** to find optimal paths for an agent. These algorithms relied on pre-defined rules and manual calculations for movement so we can say it was blind searching in the algorithms core.

In this phase, we shifted to a **Reinforcement Learning (RL)** approach using the **Q-Learning algorithm**. Unlike traditional search algorithms, Q-Learning allows the agent to **learn optimal paths dynamically** by interacting with the environment.

We also improved problem definition and visualization by remaking the whole problem using Pygame. This allowed real-time rendering of the agent's actions, dynamic updates,, unlike the last phase which was a more manual system where we summed up the plotted frames for each move in our tkinter gui.

Reinforcement Learning Approach

Learning Process

1. **Exploration vs. Exploitation:** The agent starts with random exploration and gradually shifts to exploiting learned policies using an **epsilon-greedy strategy**.
2. **Q-Value Updates:** State-action pairs are updated based on the Bellman equation.
3. **Training Episodes:** The agent runs for multiple episodes, learning incrementally with each iteration.
4. **Performance Metrics:** Training results are recorded, including average steps, total rewards, success rate, and training time.

Methodology

We applied RL using the following methodology:

1. **Environment:**
   The environment was represented as a grid layout with defined states representing the agent's position.
2. **Agent:**
   Move up/down/right/left.
3. **Reward Function:**
   The reward function encouraged goal-reaching and penalized bad actions:
   - A reward of **+150** for reaching the goal.
   - A penalty of **-100** for colliding with obstacles.
   - A penalty of **-0.5** to encourage efficient navigation.
4. **Q-Learning:**
   - **Initialization**: A Q-table is initialized to store action values.
   - **Learning Rule**: Q-values are updated using the Bellman equation.
5. **Training:**
   - The agent was trained over 150 episodes, each episode starting from(0, 0), some trials reached 300 episode with accuracy 80%.
   - Metrics such as total rewards, steps taken, and success rates were recorded to evaluate performance.

Used approach:

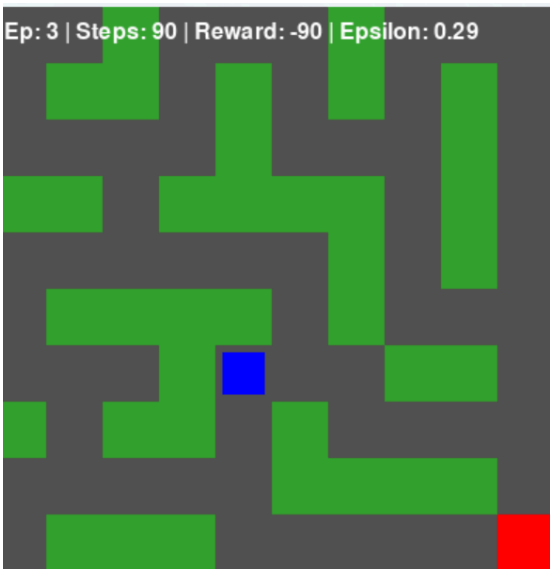
```
Initialize Q-table with zeros
Set hyperparameters: alpha, gamma, epsilon, epsilon_decay, min_epsilon

For each episode:
    Set initial state s = (0, 0)
    total_reward = 0

    For each step in the episode:
        Choose action a using ε-greedy strategy
        Perform action a and observe reward r and next state s'
        Update Q-value using:
            Q(s, a) = Q(s, a) + alpha * [r + gamma * max(Q(s', a')) - Q(s, a)]
        Set s = s'
        total_reward += r

        If goal state is reached or max steps are exceeded:
```

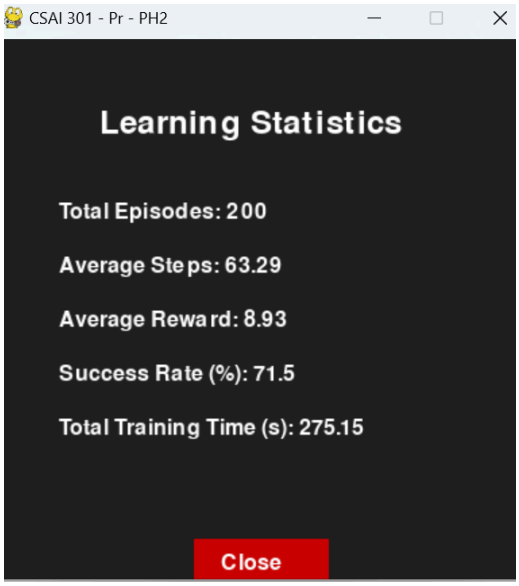
Our App

In this phase (new app):

- **Pygame** replaced Tkinter, enabling **real-time dynamic visualization**.
- The agent's movement and environment updates are shown live.
- A **HUD (Heads-Up Display)** displays essential metrics:
    - Current episode
    - Steps taken
    - Total reward
    - Exploration rate (epsilon)

- After training, a **Statistics Screen** summarizes:
    - Total episodes
    - Average steps
    - Average reward
    - Success rate
    - Total training time
- Training data is exported as a **CSV report** for further analysis.





Parameters and File Structure *("Can be set from `'main.py'` ")*

Parameter	Value
Episodes	100-300
Steps per Episode	Grid size (10*10)
Alpha (α)	0.1
Gamma (γ)	0.9
Epsilon (ε)	0.3
Epsilon Decay	0.99
Minimum Epsilon	0.1
FPS	60

- **Episodes:** Total training cycles.
- **Steps per Episode:** Maximum actions allowed per episode.
- **Alpha:** Learning rate.
- **Gamma:** Discount factor for future rewards.
- **Epsilon:** Controls exploration vs. exploitation.
- **FPS:** Controls the speed of visualization.

File Structure

1. **main.py**
    - Entry point for the project.
    - Handles training, visualization, and report generation.
2. **agent.py**
    - Q-Learning agent implementation.
    - Responsible for decision-making, learning, and Q-value updates.
3. **enviroment.py**
    - Defines the grid environment.
    - Manages state transitions, reward assignment, and obstacles.
4. **visualization.py**
    - Real-time visualization using Pygame.
    - Includes HUD updates and a final statistics screen.
5. **learning_report.csv**
    - Automatically generated after training.
    - Contains episode-wise metrics and summary statistics.

How to Use

- Install the requirements file: `pip install -r requirements.txt`
- Run the main.py: `python main.py`