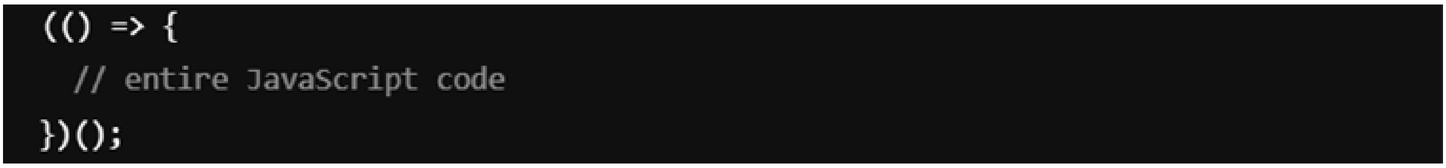
Overall structure of the program

Code reference



```
(() => {
  // entire JavaScript code
})();
```

Explanation

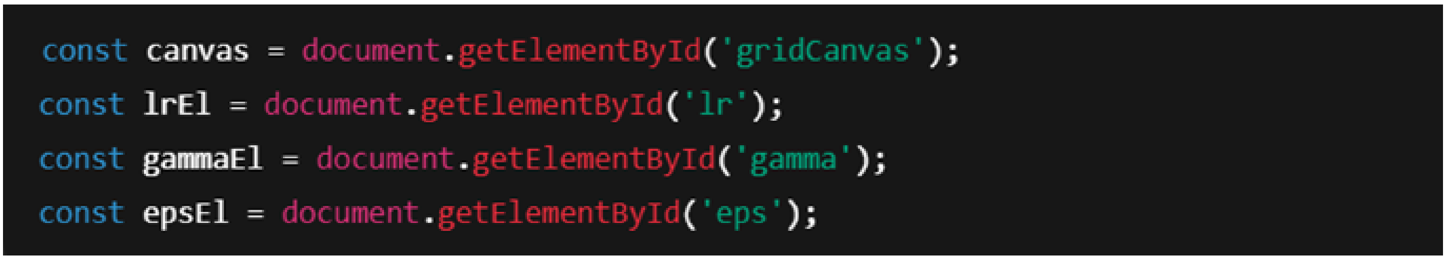
The entire JavaScript logic is wrapped inside an Immediately Invoked Function Expression (IIFE). This design choice ensures that:

- All variables remain local to the script
- No variables accidentally overwrite global browser variables
- The program behaves like a self-contained module

This is important in complex simulations like reinforcement learning, where many state variables (Q-table, agent position, grid, statistics) are constantly updated.

User interface & DOM references

Code reference



```
const canvas = document.getElementById('gridCanvas');
const lrEl = document.getElementById('lr');
const gammaEl = document.getElementById('gamma');
const epsEl = document.getElementById('eps');
```

Explanation

This section links HTML controls to JavaScript logic:

- Sliders control learning parameters (α, γ, ε)
- Buttons start, pause, and reset training
- The canvas is used for visualization

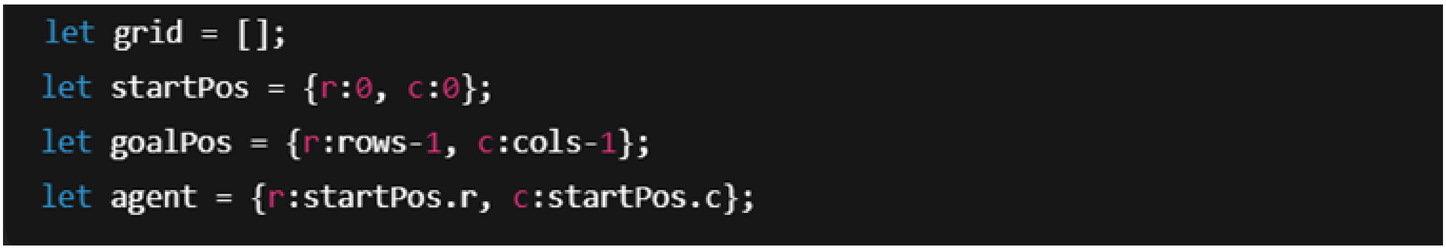
Because parameters are read live during training, the user can:

- Observe how learning behavior changes
- Experiment interactively with reinforcement learning concepts

This makes the project both educational and experimental.

Environment representation (Grid, agent, start, goal)

Code reference



```
let grid = [];
let startPos = {r:0, c:0};
let goalPos = {r:rows-1, c:cols-1};
let agent = {r:startPos.r, c:startPos.c};
```

Explanation

The environment is a finite Markov Decision Process (MDP) represented by a grid:

- Each cell is a state
- The agent occupies exactly one state at a time
- Obstacles represent forbidden or terminal states

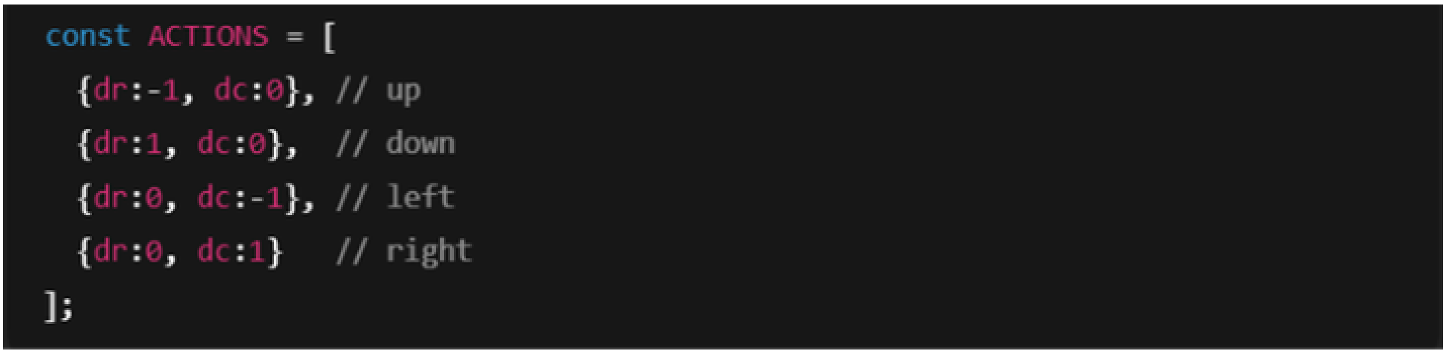
The start and goal positions define:

- Initial state distribution
- Terminal success condition

This abstraction is common in reinforcement learning and closely resembles classic problems like GridWorld.

Action space

Code reference



```
const ACTIONS = [
  {dr:-1, dc:0}, // up
  {dr:1, dc:0},  // down
  {dr:0, dc:-1}, // left
  {dr:0, dc:1}   // right
];
```

Explanation

The agent’s action space is discrete and finite:

- 4 possible actions per state

- Same actions are available in every state

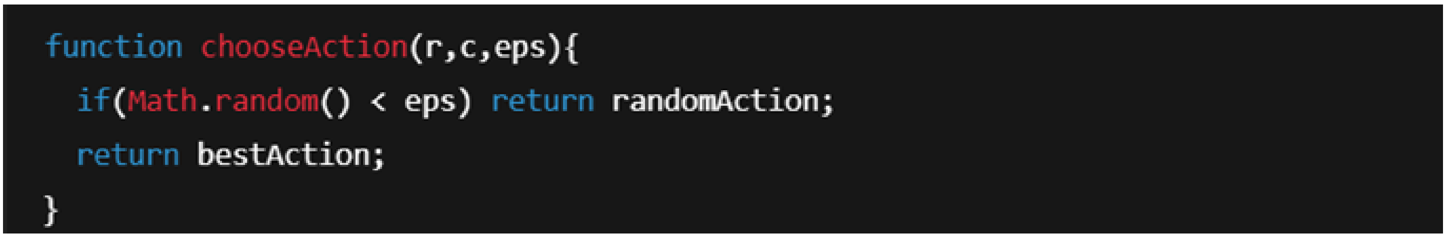
This simplifies learning because:

- The Q-table remains small and manageable
- The policy is easy to visualize with arrows

Each action deterministically changes the agent’s position, making the environment fully observable.

Q-table initialization

Code reference



```
function chooseAction(r,c,eps){
  if(Math.random() < eps) return randomAction;
  return bestAction;
}
```

Explanation

The Q-table stores the agent’s knowledge:

- One Q-value per (state, action) pair
- Initially zero → complete ignorance

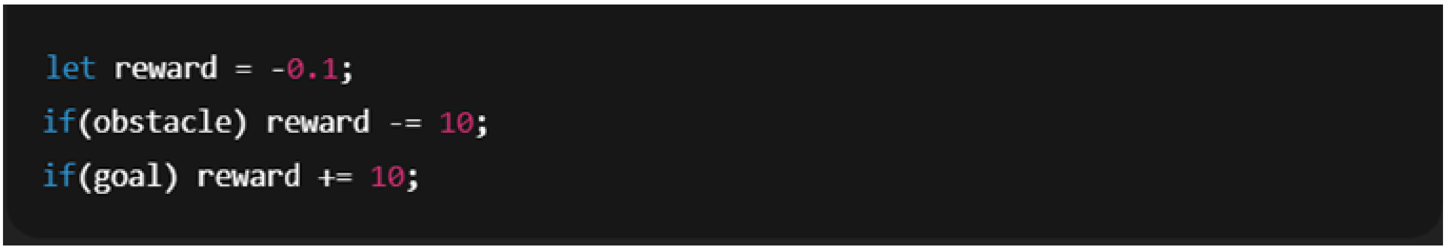
This is a tabular reinforcement learning approach, meaning:

- Learning is explicit and interpretable
- No neural networks are used
- The agent directly memorizes action values

This makes it ideal for learning and visualization.

ε-greedy action selection

Code reference



```
let reward = -0.1;
if(obstacle) reward -= 10;
if(goal) reward += 10;
```

Explanation

The ε-greedy policy balances:

- **Exploration** (trying unknown actions)
- **Exploitation** (using learned knowledge)

Early in training:

- ε is high → more random behavior

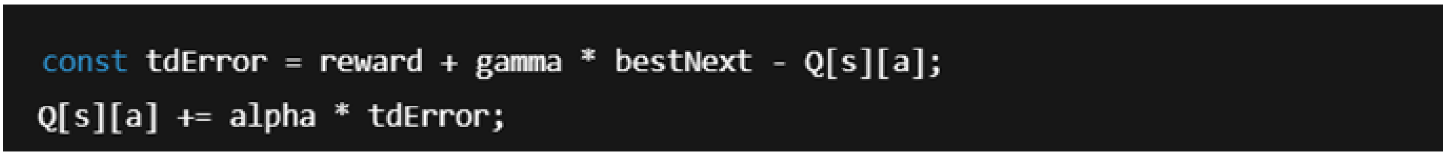
Later in training:

- ε decays → more optimal decisions

This mechanism prevents the agent from getting stuck in poor solutions too early.

Environment dynamics & rewards

Code reference



```
 const tdError = reward + gamma * bestNext - Q[s][a];
Q[s][a] += alpha * tdError;
```

Explanation

The reward function is carefully designed:

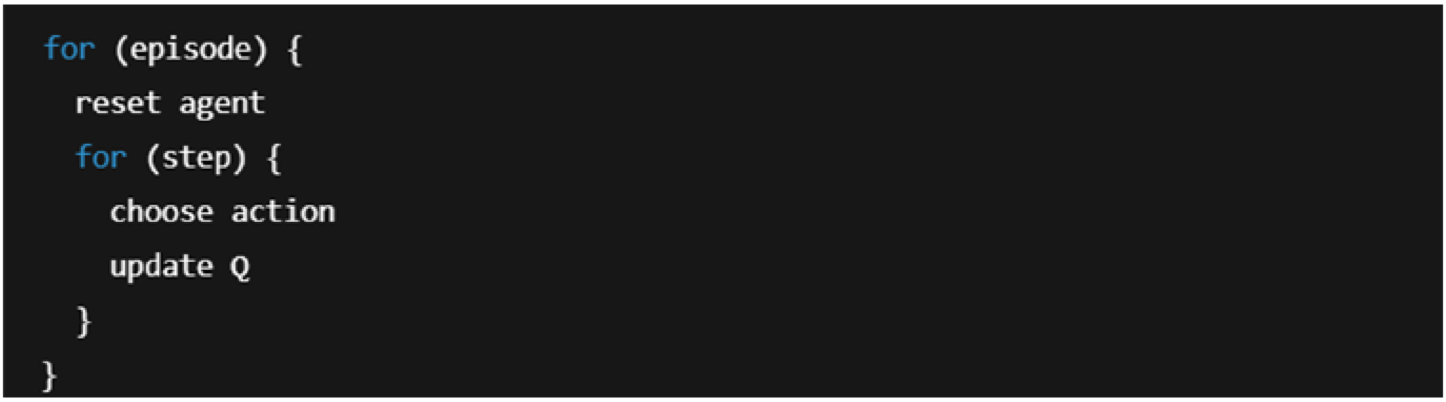
- Small step penalty encourages short paths
- Large negative reward discourages collisions
- Large positive reward defines success

This reward shaping guides learning and strongly affects:

- Convergence speed
- Path optimality
- Agent behavior

Q-learning update rule (core algorithm)

Code reference



```
for (episode) {
  reset agent
  for (step) {
    choose action
    update Q
  }
}
```

Explanation

This is the mathematical heart of the algorithm.

The agent:

1. Predicts the value of an action
2. Observes the real outcome
3. Updates its belief using the TD error

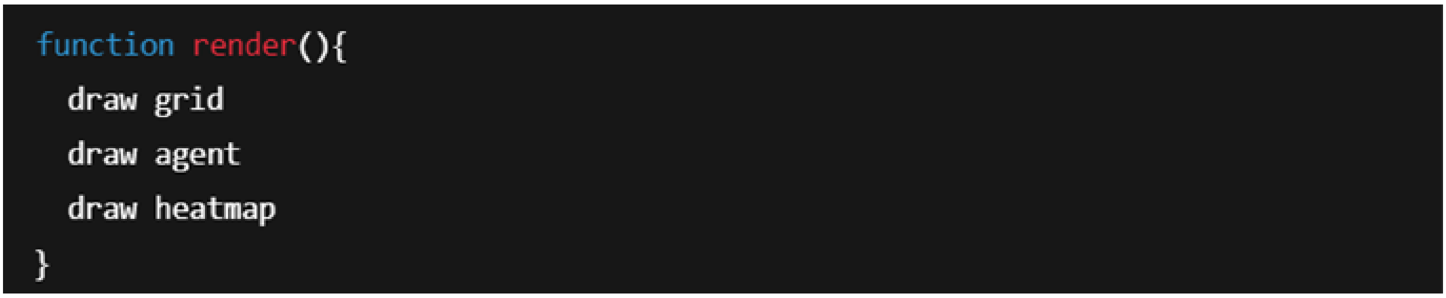
Q-learning is off-policy, meaning:

- It learns the optimal policy even while exploring

This guarantees convergence under standard assumptions.

Training loop (episodes & steps)

Code reference



```
function render(){
  draw grid
  draw agent
  draw heatmap
}
```

Extended explanation

Training is divided into:

- Episodes (complete trials)
- Steps (individual interactions)

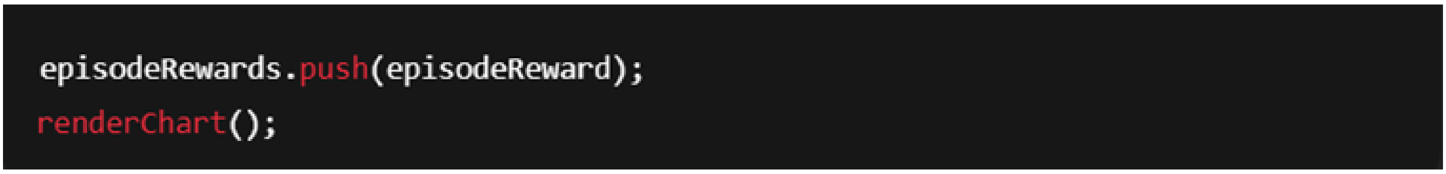
This structure:

- Allows performance tracking per episode

- Matches theoretical RL formulations
- Enables epsilon decay after each episode

Visualization of the environment

Code reference



```
episodeRewards.push(episodeReward);
renderChart();
```

Explanation

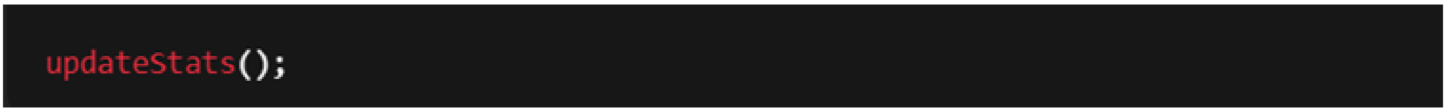
Visualization serves multiple purposes:

- Shows agent behavior in real time
- Makes policy learning visible
- Helps debug reward and parameter choices

The Q-heatmap directly reflects learned value estimates, which is rare in black-box models.

Learning curve (chart)

Code reference



```
updateStats();
```

Explanation

The learning curve:

- Displays total reward per episode
- Shows convergence trends
- Helps detect instability or poor parameter choices

The moving average smooths noise and reveals real progress.

Statistics & monitoring

Code reference

```
canvas.addEventListener('click', ...);
```

Explanation

Statistics provide quantitative insight:

- Success rate → policy reliability
- Average steps → path efficiency
- Best episode → peak performance

This bridges theory and observed behavior.

Interactivity

Code reference

```
canvas.addEventListener('click', ...);
```

Explanation

Real-time interaction:

- Forces the agent to adapt
- Demonstrates non-stationary environments
- Shows the importance of continuous learning

This is especially useful for demonstrations and teaching.

14. Global summary

Explanation

This project:

- Implements a complete tabular Q-learning system
- Respects theoretical RL foundations
- Adds strong visualization and interaction
- Is ideal for education, experimentation, and explanation

Webside Explanation

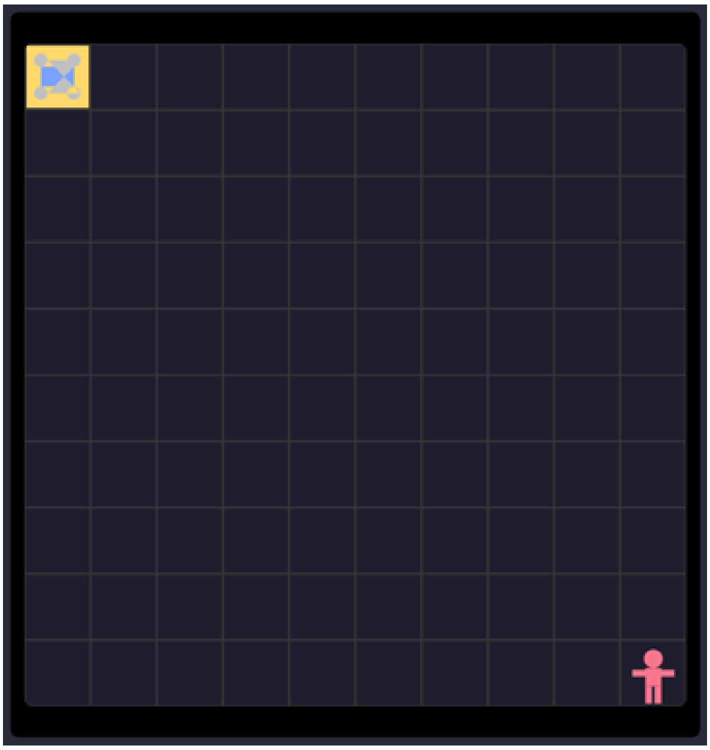
Grid Environment

This image shows the environment where the agent operates.

The grid consists of cells representing possible states.

The pink character is the goal, and the yellow cell in the top-left corner is the drone.

The agent starts at the top-left and must learn the optimal path, receiving rewards or penalties along the way.



Role:

To visualize how the agent explores and learns the best path to the goal.

Hyperparameters and Algorithm

This image shows the Q-learning hyperparameter panel.

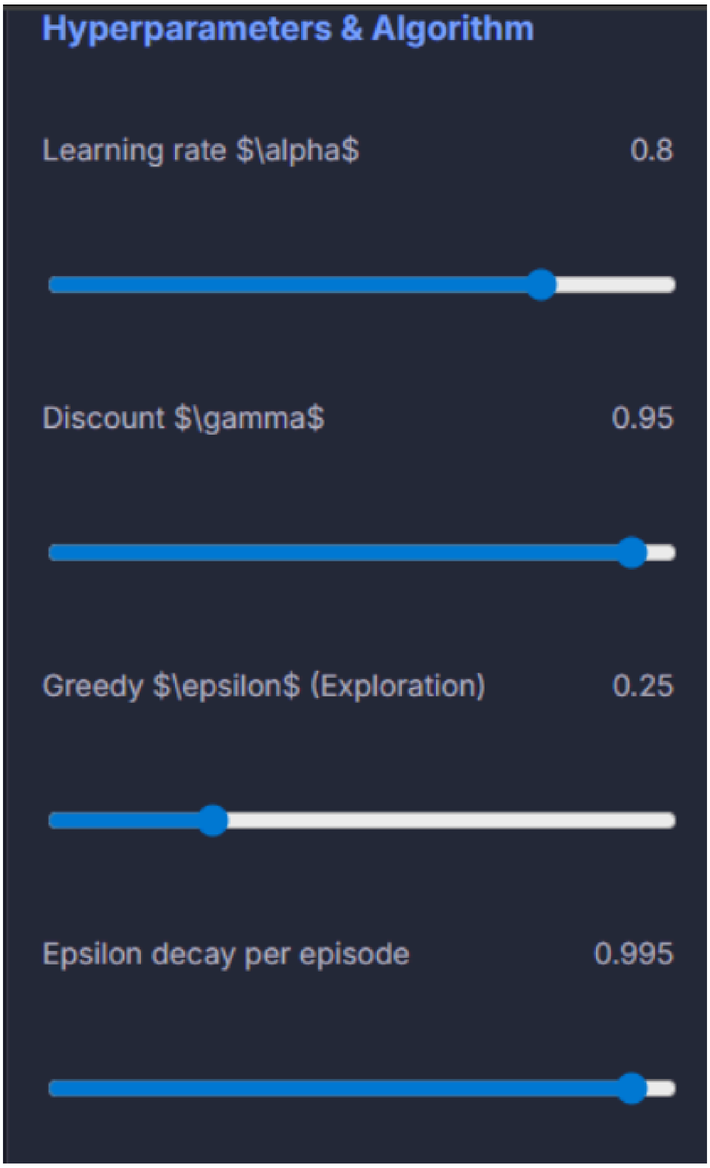
The learning rate (α) controls how fast the agent learns.

The discount factor (γ) sets the importance of future rewards.

Epsilon (ε) manages exploration; higher values increase randomness, while epsilon decay reduces it over time.

Role:

To tune the algorithm and observe how hyperparameters affect learning.



Hyperparameters & Algorithm

Learning rate α 0.8

Discount γ 0.95

Greedy ϵ (Exploration) 0.25

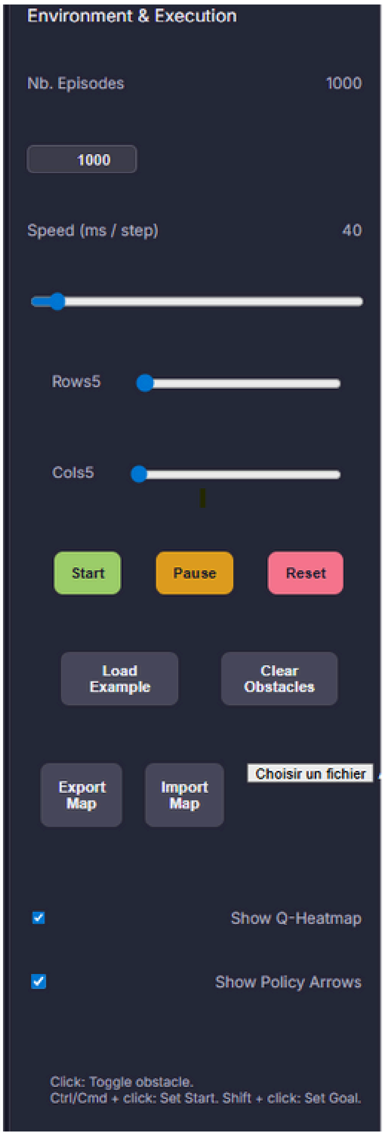
Epsilon decay per episode 0.995

Environment & Execution

This panel shows the model and grid parameters. It includes the number of episodes (1000), the simulation speed (40 ms per step), the grid size (5×5 displayed, although the code uses 10×10), and simulation controls (Start, Pause, Reset, etc.).

Role:

It allows configuring the environment (obstacles, start and goal positions) and adjusting the hyperparameters of the reinforcement learning algorithm: learning rate (α), discount factor (γ), and exploration rate (ϵ).



Environment & Execution

Nb. Episodes 1000

1000

Speed (ms / step) 40

Rows5

Cols5

☑ Show Q-Heatmap

☑ Show Policy Arrows

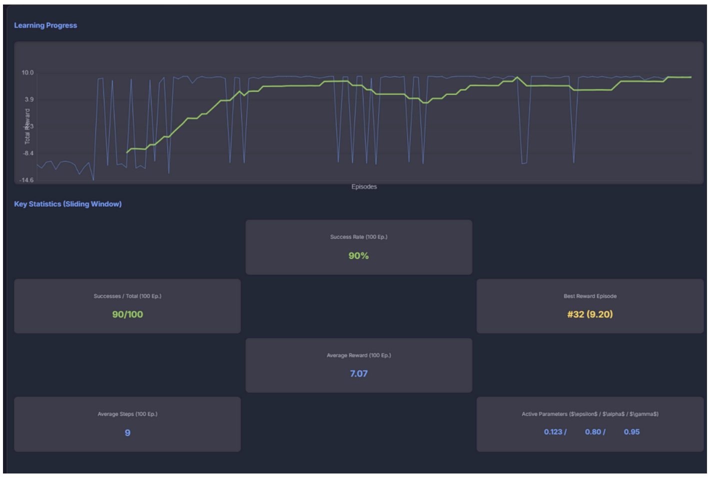
Click: Toggle obstacle.
Ctrl/Cmd + click: Set Start. Shift + click: Set Goal.

Image 2: Learning Progress

This panel presents the agent’s performance results, including a graph of rewards per episode and key statistics computed over the last 100 episodes (sliding window).

Role:

It shows whether the agent has successfully learned an optimal policy to achieve the goal.



Learning Progress

Key Statistics (Sliding Window)

Success Rate (100 Ep.)
90%

Successes / Total (100 Ep.)
90/100

Best Reward Episode
#32 (9.20)

Average Reward (100 Ep.)
7.07

Average Steps (100 Ep.)
9

Active Parameters (ϵ / α / γ)
0.123 / 0.80 / 0.95