COMP9414 Artificial Intelligence

GridWorldEnv Environment User Guide

Term 2, 2025

Version: 1.0
Last Updated: July 2025
Purpose: This guide provides comprehensive documentation for the `GridWorldEnv` class used in Assignment 2

1 Overview

The `GridWorldEnv` class implements a grid world environment for reinforcement learning experiments. This guide covers installation, usage, and troubleshooting for the environment provided in `env.py`.

2 Environment Specifications

The environment implements an 11×11 grid world with the following characteristics:

- **Grid Size:** 11×11 cells
- **State Space:** Discrete 2D coordinates (x, y) where $x, y \in \{0, 1, ..., 10\}$
- **Action Space:** 4 discrete actions
- **Goal Position:** $(10, 10)$ - bottom-right corner
- **Starting Position:** Random (avoiding obstacles and goal)
- **Episode Termination:** Only when goal is reached

2.1 Coordinate System

The environment uses a standard 2D coordinate system:

- Origin $(0, 0)$ is at the top-left corner
- X-axis increases downward (rows)
- Y-axis increases rightward (columns)

2.2 Obstacles

The environment contains 10 fixed obstacles arranged in two patterns:

L-shaped pattern (top-left area):

- Positions: $(2, 2)$, $(2, 3)$, $(2, 4)$, $(3, 2)$, $(4, 2)$

Cross pattern (centre):

◦ Positions: $(5, 4)$, $(5, 5)$, $(5, 6)$, $(4, 5)$, $(6, 5)$

2.3 Action Space

Action	Value	Effect
Up	0	Decrease x by 1
Down	1	Increase x by 1
Left	2	Decrease y by 1
Right	3	Increase y by 1

2.4 Reward Structure

Event	Reward
Reach Goal	+25
Hit Obstacle	-10
Normal Step	-1

3 Installation and Setup

3.1 Prerequisites

Ensure you have Python 3.7+ and the following packages installed:

```
pip install numpy pygame
```

3.2 File Structure

Your project directory should contain:

```
your_project/
|-- env.py                    # The environment file (provided)
|-- your_solution.ipynb       # Your Jupyter notebook solution
`-- images/                   # Required image directory
    |-- grid_agent.png        # Agent image
    |-- grid_goal_position.png # Goal position image
    `-- grid_obstacle.png     # Obstacle image
```

4 API Reference

4.1 Class: GridWorldEnv

4.1.1 Constructor

```
GridWorldEnv(seed=None)
```

Parameters:

– `seed` (int, optional): Random seed for reproducibility

Example:

```python
env = GridWorldEnv(seed=42)  # Reproducible environment
```

4.1.2 reset()

```python
reset() -> Tuple[int, int]
```

Resets the environment to a new episode.

Returns:

- `state`: Initial position as tuple (x, y)

Example:

```python
state = env.reset()  # Returns e.g., (3, 7)
```

4.1.3 step(action)

```python
step(action: int) -> Tuple[Tuple[int, int], float, bool, dict]
```

Executes one environment step.

Parameters:

- `action`: Integer in $\{0, 1, 2, 3\}$

Returns:

- `next_state`: New position (x, y)
- `reward`: Reward for this step
- `done`: Whether episode is finished
- `info`: Additional information (empty dict)

Example:

```python
next_state, reward, done, info = env.step(0)  # Move up
```

4.1.4 render()

```python
render(delay=0.1, mode="human", episode=1,
       learning_type="Q-learning",
       availability=None, accuracy=None)
```

Displays the environment using Pygame.

Parameters:

- `delay`: Time delay between frames (seconds)
- `mode`: Rendering mode (unused, for compatibility)
- `episode`: Current episode number for display

- `learning_type`: Algorithm name for display
- `availability`: Teacher availability (0.0-1.0)
- `accuracy`: Teacher accuracy (0.0-1.0)

Example:

```
env.render(delay=0.1, episode=50, learning_type="SARSA")
```

4.1.5 close()

```
close() -> None
```

Closes the Pygame window and cleans up resources.

Example:

```
env.close()  # Always call when done
```

5 Usage Examples

5.1 Jupyter Notebook Setup

Since your solution must be submitted as a Jupyter notebook (`.ipynb`), start your notebook with:

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm

# Import the environment
from env import GridWorldEnv

# Set random seed for reproducibility
np.random.seed(42)

# Configure plotting for notebook
%matplotlib inline
plt.style.use('default')
```

5.2 Basic Episode

```
from env import GridWorldEnv
import numpy as np

# Create environment
env = GridWorldEnv(seed=42)

```

```python
# Run one episode
state = env.reset()
done = False
total_reward = 0

while not done:
    # Random action (replace with your policy)
    action = np.random.randint(0, 4)

    # Take action
    next_state, reward, done, _ = env.step(action)
    total_reward += reward

    # Optional: Render
    env.render(delay=0.1)

    state = next_state

print(f"Episode finished with total reward: {total_reward}")
env.close()
```

5.3 Teacher-Student Framework

```python
# Rendering with teacher information
env.render(
    delay=0.05,
    episode=episode,
    learning_type="Q-learning with Teacher",
    availability=0.7,  # 70% availability
    accuracy=0.9       # 90% accuracy
)
```

6 Common Issues and Solutions

Issue	**Solution**
`ModuleNotFoundError: No module named 'pygame'`	Install pygame: `pip install pygame`
`FileNotFoundError: 'images/grid_agent.png'`	Ensure the `images/` folder exists in your project directory with all three PNG files
Display issues in WSL/SSH	Set environment variable: `export SDL_VIDEODRIVER=dummy` or run without rendering
Window not responding	Ensure you're calling `env.render()` in your loop; Pygame requires regular event processing
Slow training with rendering	Only render periodically (e.g., every 50-100 episodes) or increase the delay parameter

7 Important Notes

- Actions that would move the agent outside grid boundaries are ignored
- Actions that would move the agent into an obstacle are ignored, but the agent still receives the obstacle penalty (-10)
- The episode only terminates when the goal is reached; there is no maximum step limit in the environment itself
- Always use `seed` parameter for reproducible experiments
- The rendering is optional and can be disabled for faster training

7.1 Jupyter Notebook Requirements

- Submit your solution as a single `.ipynb` file
- Ensure all cells have been run and outputs are visible
- Include markdown cells to explain your approach
- Use `%matplotlib inline` for plots to display in the notebook
- The notebook should be self-contained and runnable from top to bottom
- Do not use `env.render()` in the final submission as it requires pygame display

8 Integration with Assignment Tasks

When implementing your reinforcement learning algorithms:

1. Initialise your Q-table with dimensions: `(11, 11, 4)`
2. Access Q-values using: `q_table[state[0], state[1], action]`
3. Track metrics as specified in the assignment requirements
4. The environment automatically handles boundaries and obstacles