COMP3211 Assignment 1: Boundary-Following Agents in Grid Mazes

Overview

In this assignment, you will implement two AI agents that navigate grid-based mazes. The agents must follow walls (boundary-following) to systematically explore the environment without any knowledge of the maze layout or goal locations.

Learning Objectives

By completing this assignment, you will: - Understand and implement **production rule systems** for decision-making - Design and implement **finite state machines** for agent behavior - Handle **limited perception** scenarios - Gain practical experience with agent-environment interaction

Assignment Tasks

You are required to implement **two agents**:

1. **Production Rules Agent**
   - Uses production rules for left-hand wall-following
   - Has full 3×3 perception window (can see all 8 surrounding cells)
   - Makes instant decisions based on current perception
2. **State Machine Agent**
   - Uses a 3-state finite state machine for left-hand wall-following
   - Has **limited perception** (can only see left, front, right)
   - Must maintain internal state to handle corners and wall-following

The Environment

- **Grid-based mazes** with walls, passages, and rewards (yellow coins)
- **Agent perception**: 3×3 observation window centered on agent
- **Three difficulty levels**: Simple, Medium, Hard
- **Goal**: Collect all rewards by following boundaries
- **Success criteria**: Collect all rewards within step limit

Environment Assumptions: 1. **Non-tight spaces**: All passages are at least 2 cells wide, ensuring the agent can navigate without getting stuck 2. **Boundary rewards**: All coins are placed along the walls that the agent will follow using the left-hand rule

Grading Criteria

Testing Protocol: - Your agents will be tested on **3 hidden test maps** (not the ones provided for practice) - Each hidden map has relative difficulty: Simple, Medium, or Hard - Each map will be tested **3 times** with different starting positions and orientations - Each test run has an **optimal step limit** based on the reference solution

Grading Scheme: - The total score for this assignment is 50. - Score for each test case = (Number of collected coins / Total number of coins) - Each test case contributes evenly to the total score.

Setup Instructions

Prerequisites

- Python 3.8
- MiniGrid package

Installation

```
# Create environment by Conda
conda create --name 3211A1 python=3.8
conda activate 3211A1

# Install required packages
pip install minigrid
```

Verify Installation

Test the environment with keyboard control:

```
python manual_control.py --env simple --view full
```

If you see a maze with a red triangle (agent) and can control it with arrow keys, you're ready to go!

Project Structure

```
A1/
  agents/
     __init__.py
     base_agent.py              # Base agent class (DO NOT MODIFY)
     keyboard_agent.py          # Manual control (for testing)
     naive_agent.py             # Random movement baseline (IMPLEMENTED)
     production_rules_agent.py  # TODO: IMPLEMENT THIS
     state_machine_agent.py     # TODO: IMPLEMENT THIS
  environments/
     __init__.py
     maze_env.py                # Maze environments (DO NOT MODIFY)
```

```
utils/
    __init__.py
manual_control.py                # Test with keyboard
visualize_steps.py               # Step-by-step visualization
README.md                        # This file
```

Key Files for Implementation

- **agents/production_rules_agent.py**: Implement `_decide_left_hand()` method
- **agents/state_machine_agent.py**: Implement `_decide_left_hand()` method with state transitions

---

Getting Started

Step 1: Understand the Environment

Important: Agent Orientation and Local Perception

The agent has a **direction/orientation** (facing up/down/left/right in the global maze). All observations and actions are **agent-relative**, not global:

- The 3×3 observation grid is what the agent sees **from its current perspective**
- "Front" (TF) means the cell in front of where the agent is currently facing
- "Left" (ML) means to the agent's left side, relative to its orientation
- Please check the comment in 'the Python files to view the visualized positions
- Actions (turn left, turn right, move forward) are relative to the agent's **current facing direction**
- When the agent turns, its perception rotates with it

Example: If the agent is facing east and turns left, it will then face north, and its "front" cell changes accordingly.

Run manual control to explore the maze and observe how perception changes with orientation:

```
# Simple maze
python manual_control.py --env simple --view full

# Medium maze, from agent's perspective
python manual_control.py --env medium --view agent

# Hard maze
python manual_control.py --env hard --view full
```

Keyboard Controls: - **Arrow Up / W**: Move forward - **Arrow Left / A**: Turn left - **Arrow Right / D**: Turn right - **R**: Reset environment - **Q / ESC**: Quit

View Options: - `--view full`: See entire maze (global view) - `--view agent`: See from agent's perspective (local 3×3 window that rotates with agent)

Step 2: Study the Baseline

Examine the NaiveAgent implementation to understand: - How to inherit from BaseAgent - How perception works (3×3 grid) - How to return actions

```
# Watch the naive agent (random movement)
python visualize_steps.py --agent naive --env simple --max-steps 200
```

Step 3: Read the Code Structure

Open the skeleton files to understand what you need to implement:

agents/production_rules_agent.py - Read the class docstring explaining the left-hand wall-following strategy - Study the `perceive()` method to understand the 3×3 grid layout - Review the TODO comments in `_decide_left_hand()` for implementation hints - Note the 3 production rules listed in the comments

agents/state_machine_agent.py - Read the class docstring explaining the 3 states - Study the limited `perceive()` method (only ML, TF, MR) - Review the TODO comments for state transition logic - Note the conditions for each state transition

Step 4: Implement Production Rules Agent

What you need to implement: - `_decide_left_hand(p)` method in production_rules_agent.py

Production Rules (in priority order): 1. IF (TF == WALL OR TR == WALL) THEN TURN_RIGHT 2. ELSE IF (ML == EMPTY AND BL == WALL) THEN TURN_LEFT
3. ELSE MOVE_FORWARD

Perception available: - Full 3×3 grid: TL, TF, TR, ML, MR, BL, BF, BR

Step 5: Implement State Machine Agent

What you need to implement: - `_decide_left_hand(p)` method in state_machine_agent.py - State transitions between FOLLOW_WALL, FIND_WALL, TURN_CORNER

Perception constraint: - Limited to 3 cells only: ML, TF, MR

States: - **FIND_WALL**: Initial state - agent searches for a wall to follow. Can also detect walls on the right and transition to turn appropriately. - **FOLLOW_WALL**: Operating state when wall on left is present, moving forward along wall - **TURN_CORNER**: Handling corners with multi-step turns

Startup note: The agent begins in FIND_WALL state to actively search for a wall, then transitions to FOLLOW_WALL once a left wall is detected. The agent can also detect right-side walls during search and react accordingly.

Step 6: Test Your Implementation

Visualize your agent's behavior:

```
# Test production rules agent
python visualize_steps.py --agent production_rules --env simple --max-steps 100

# Test state machine agent
python visualize_steps.py --agent state_machine --env simple --max-steps 100

# Compare on harder mazes
python visualize_steps.py --agent production_rules --env medium --max-steps 300
python visualize_steps.py --agent state_machine --env hard --max-steps 200
```

Playback controls: - **Space**: Play/Pause - **Right Arrow**: Step forward - **Left Arrow**: Step backward - **1-9 keys**: Set playback speed (1 = slowest, 9 = fastest) - **R**: Reset to beginning

---

Performance Expectations

Upon successful implementation, you should expect your agents collect all coins in the following numbers of steps:

Maze	Production Rules	State Machine
Simple	54 steps	54 steps
Medium	225 steps	225 steps
Hard	188 steps	188 steps

Common Issues and Tips

Debugging Tips

1. **Use visualization**: Watch your agent step-by-step to see where it fails
2. **Check rule order**: Rules must execute in exact priority order
3. **Verify state transitions**: Make sure you update `self.state` correctly

Getting Help

If your agent isn't working: 1. Compare with the NaiveAgent structure 2. Re-read the TODO comments in the skeleton code files 3. Review the production rules and state descriptions in this README 4. Use manual control to understand expected behavior 5. Test on Simple maze first before moving to harder mazes

---

Submission Guidelines

Submit the following files: 1. `agents/production_rules_agent.py` - Your production rules implementation 2. `agents/state_machine_agent.py` - Your state machine implementation

---

Additional Resources

- **MiniGrid Documentation**: https://minigrid.farama.org/ (though not that helpful for this assignment)
- **Finite State Machines**: Review lecture notes on FSM design
- **Production Systems**: Review lecture notes on rule-based systems

Good luck!