COMP9414 Artificial Intelligence

Assignment 2: Reinforcement Learning

Term 2, 2025

Due: Week 9, Friday, 1 August 2025, 5:00 PM AEST
Worth: 25 marks (25% of final grade)
Submission: Electronic submission via Moodle

1 Problem Overview

In this assignment, you will implement and compare two reinforcement learning algorithms: **Q-learning** and **SARSA**, within a static grid world environment.

The grid world consists of an 11×11 grid in which the agent must navigate from a random starting position to a designated goal while avoiding fixed obstacles arranged in two distinct patterns.

You will first develop the Q-learning and SARSA algorithms, implementing action selection policies that allow the agent to choose actions using an epsilon-greedy approach to balance exploration and exploitation. To ensure fair comparison between the algorithms, you must use identical hyperparameters across all experiments.

After training the baseline agents, you will simulate **interactive reinforcement learning (IntRL)** by introducing a teacher-student framework. In this setup, a pre-trained agent (the teacher) provides advice to a new agent (the student) during its training. Each algorithm will teach its own type: Q-learning teachers will guide Q-learning students, and SARSA teachers will guide SARSA students.

The teacher's advice will be configurable in terms of its **availability** (probability of offering advice) and **accuracy** (probability that the advice is correct).

You will evaluate the impact of teacher feedback on the student's learning performance by running experiments with varying levels of availability and accuracy. By maintaining consistent hyperparameters across all four tasks, you can meaningfully compare how each algorithm performs with and without teacher guidance.

The goal is to understand how teacher interaction influences the learning efficiency of each algorithm and to determine which algorithm benefits more from teacher guidance under identical conditions.

2 Environment

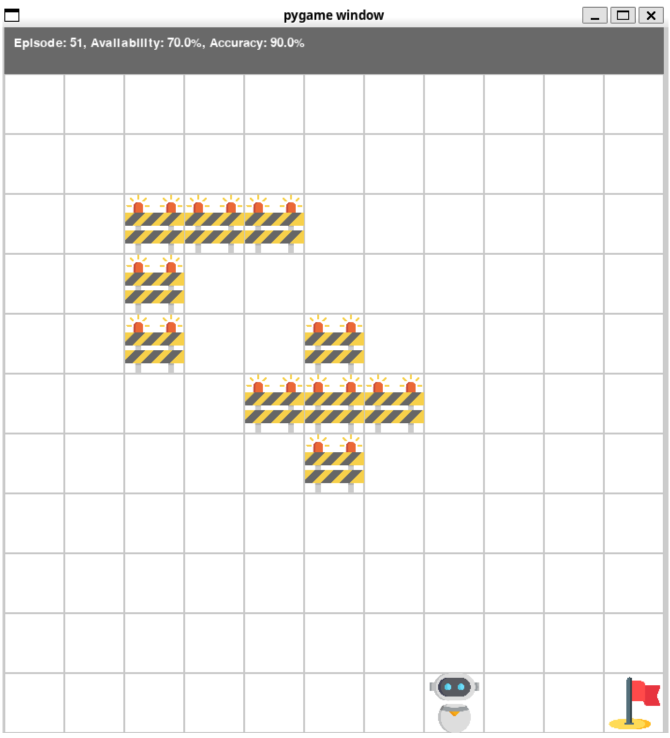
The environment is provided in the `env.py` file. This section provides a brief overview of the grid world environment you will be working with.

For detailed information about the environment including setup instructions, key functions, agent movement and actions, movement constraints, and the reward structure, please refer to the **Environment User Guide** provided separately as a PDF file. Ensure you familiarise yourself with the environment before proceeding with the assignment.

2.1 Environment Specifications

The environment has the following specifications:

- **Grid Size:** 11 × 11
- **Obstacles:** 10 cells arranged in two patterns:
  - L-shaped pattern: (2,2), (2,3), (2,4), (3,2), (4,2)
  - Cross pattern: (5,4), (5,5), (5,6), (4,5), (6,5)
- **Goal Position:** (10, 10)
- **Rewards:**
  - Reaching the goal: +25
  - Hitting an obstacle: -10
  - Each step: -1



Episode: 51, Availability: 70.0%, Accuracy: 90.0%

Environment Elements:

Agent
Grey robot that navigates the grid world

Goal
Red flag at position (10,10)
Rewards +25 points

Obstacles
Construction barriers
Penalty of -10 points

Status Bar Information:

- Episode number
- Teacher availability (%)
- Teacher accuracy (%)

Figure 1: The 11×11 grid world environment with visual elements. The agent must navigate from its starting position to the goal whilst avoiding the L-shaped and cross-shaped obstacle patterns.

3 Hyperparameter Guidelines

For this assignment, you should use the following baseline parameters:

- **Learning rate** (α): 0.1 to 0.5
- **Discount factor** (γ): 0.9 to 0.99
- **Epsilon**:
  - For exploration strategies with decay: Initial 0.8 to 1.0, Final 0.01 to 0.1
  - Decay strategy: Can use linear decay, exponential decay, or other decay strategies
  - For fixed epsilon (no decay): 0.1 to 0.3
- **Number of episodes**: 300 to 1000
- **Maximum steps per episode**: 50 to 100

You may experiment within these ranges, but must:

- Use identical parameters across all four tasks (Tasks 1–4) to ensure fair comparison between Q-learning and SARSA, both with and without teacher guidance
- Document your final chosen values and provide brief justification
- For Tasks 3 and 4 (teacher experiments), you may use fewer episodes to reduce computational time while maintaining meaningful results.

4 Task 1: Implement Q-learning

In this task, you will implement the Q-learning algorithm and train an agent in the provided environment.

Implementation Requirements

Your Q-learning implementation should:

- Train the agent for the specified number of episodes using the hyperparameters from Section 3
- Use epsilon-greedy action selection for exploration
- Update Q-values according to the Q-learning update rule

Metrics to Track

During training, track these metrics for each episode:

- **Total Rewards per Episode**: The cumulative reward accumulated during the episode
- **Steps per Episode**: The number of steps taken to complete the episode
- **Successful Episodes**: Whether the agent reached the goal

Required Outputs

After training is complete, you must produce the following:

- Generate a plot that displays the episode rewards over time. The plot should include:
  - Raw episode rewards (with transparency to show variance)
  - A moving average line (e.g., 50-episode window) for smoothing
  - A horizontal line at y=0 to indicate the transition between positive and negative rewards
  - Appropriate labels, title, and legend

  Figure 2 shows a sample of what your Q-learning performance plot should look like (generated with simulated data).

- Calculate and report the **Success Rate**, **Average Reward per Episode**, and **Average Learning Speed**, using the following formulas:
  - **Success Rate**:

    $$\text{Success Rate} = \left(\frac{\text{Number of Successful Episodes}}{N}\right) \times 100\% \tag{1}$$

    where N is the total number of episodes.
  - **Average Reward per Episode**:

    $$\text{Average Reward} = \frac{\sum_{i=1}^{N} R_i}{N} \tag{2}$$

    where R_i is the total reward in the i-th episode.
  - **Average Learning Speed**:

    $$\text{ALS} = \frac{1}{\frac{1}{N}\sum_{i=1}^{N} S_i} \tag{3}$$

    where S_i is the number of steps taken in the i-th episode.

- Keep track of the following outputs:
  - The three calculated metrics: average reward, success rate, and average learning speed
  - The trained Q-table, as it will be used as the teacher in Task 3

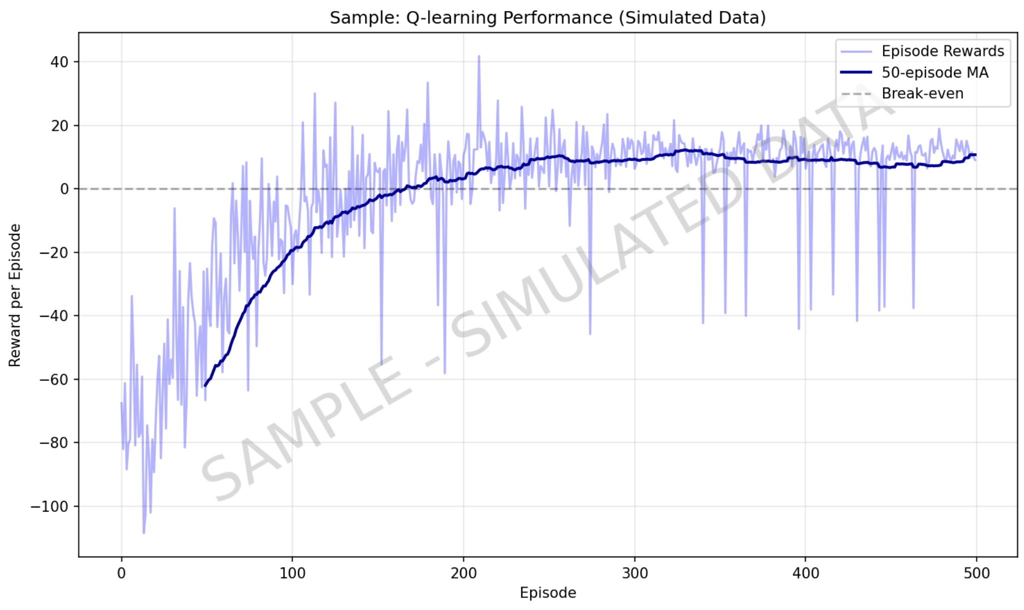


Figure 2: Sample Q-learning performance plot showing episode rewards and 50-episode moving average. This is generated with simulated data for demonstration purposes only.

5 Task 2: Implement SARSA

Important: You must use the same hyperparameters (learning rate, discount factor, epsilon, episodes, and maximum steps) that you chose in Task 1 to ensure fair comparison between Q-learning and SARSA.

In this task, you will implement the SARSA algorithm and train an agent in the provided environment.

Implementation Requirements

Your SARSA implementation should:

- Train the agent for the specified number of episodes using the same hyperparameters as Task 1
- Update Q-values according to the SARSA update rule

Metrics to Track

During training, track these metrics for each episode:

- **Total Rewards per Episode**: The cumulative reward accumulated during the episode
- **Steps per Episode**: The number of steps taken to complete the episode
- **Successful Episodes**: Whether the agent reached the goal

Required Outputs

After training is complete, you must produce the following:

- Generate a plot that displays the episode rewards over time. The plot should include:
  - Raw episode rewards (with transparency to show variance)
  - A moving average line (e.g., 50-episode window) for smoothing
  - A horizontal line at y=0 to indicate the transition between positive and negative rewards
  - Appropriate labels, title, and legend

  Figure 3 shows a sample of what your SARSA performance plot should look like (generated with simulated data).
- Calculate and report the **Success Rate**, **Average Reward per Episode**, and **Average Learning Speed** using the same formulas as in Task 1 (Equations 1, 2, and 3).
- Keep track of the following outputs:
  - The three calculated metrics: average reward, success rate, and average learning speed
  - The trained Q-table, as it will be used as the teacher in Task 4

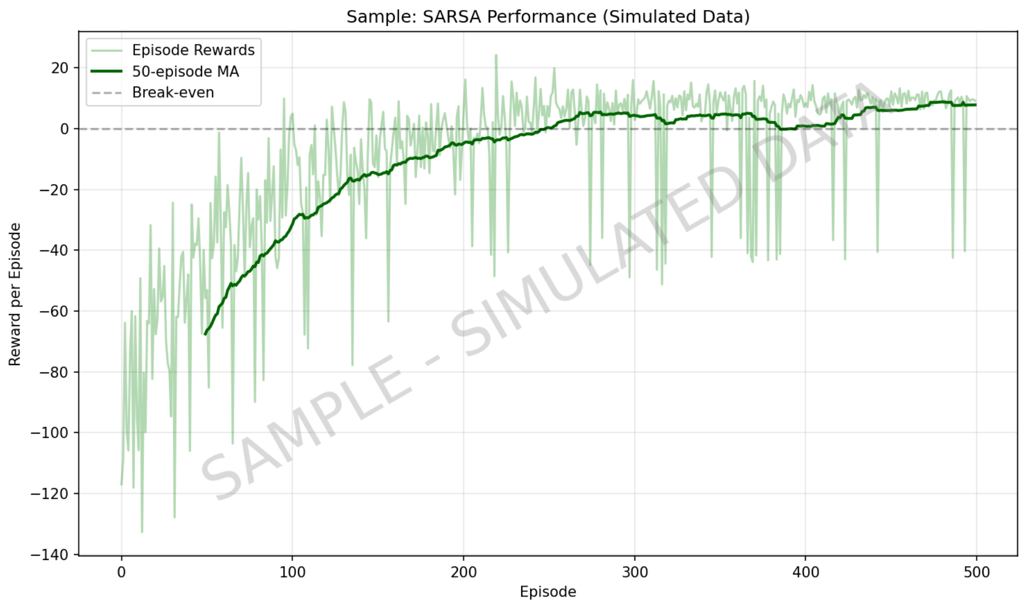


Figure 3: Sample SARSA performance plot showing episode rewards and 50-episode moving average. This is generated with simulated data for demonstration purposes only.

6 Baseline Comparison

After completing Tasks 1 and 2, you should compare the baseline performance of Q-learning and SARSA. This comparison will help you understand the fundamental differences between the two algorithms before introducing teacher guidance.

Creating the Comparison

Generate comparison visualisations that include:

- **Learning Progress Comparison**
  - Episode rewards for both Q-learning and SARSA (with transparency)
  - 50-episode moving averages for both algorithms
  - The y=0 reference line
  - Average reward values for each algorithm
- **Success Rate Comparison**
  - Rolling success rates (50-episode window) for both algorithms
  - Overall success rates for each algorithm
  - Success rate ranging from 0 to 100%

Figure 4 shows a sample baseline comparison plot (generated with simulated data).

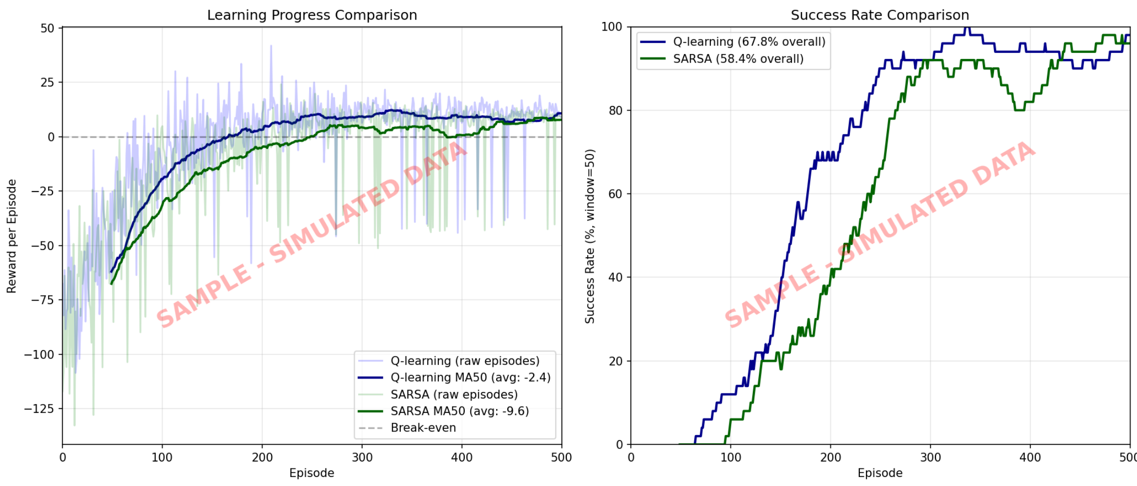


Figure 4: Sample baseline comparison showing Q-learning vs SARSA performance. Left: Episode rewards with moving averages. Right: Success rate over time. This is generated with simulated data for demonstration purposes only.

7 Teacher Feedback Mechanism

A **teacher feedback system** is a valuable addition to the training process of agents using Q-learning or SARSA. In this system, a pre-trained agent (the teacher) assists a new agent by offering advice during training. The advice provided by the teacher is based on two key probabilities:

- The **availability** factor determines whether advice is offered by the teacher at any step.

– The **accuracy** factor dictates whether the advice given is correct or incorrect.

7.1 How It Works

At each step, the system first determines whether the teacher provides advice (based on availability). If advice is given, it then determines whether the advice is correct (based on accuracy). These two checks ensure that advice is provided probabilistically and may not always be accurate.

The agent responds to the teacher's advice as follows:

- If the generated advice is **correct** (given the accuracy parameter), the agent follows the teacher's recommended action (the action with highest Q-value in the teacher's Q-table).
- If the generated advice is **incorrect**, the agent takes a random action, excluding the trainer's best action.
- If **no advice** is given, the agent continues its independent learning using its exploration strategy (epsilon-greedy).

Figure 5 illustrates the complete decision process for the teacher feedback mechanism.

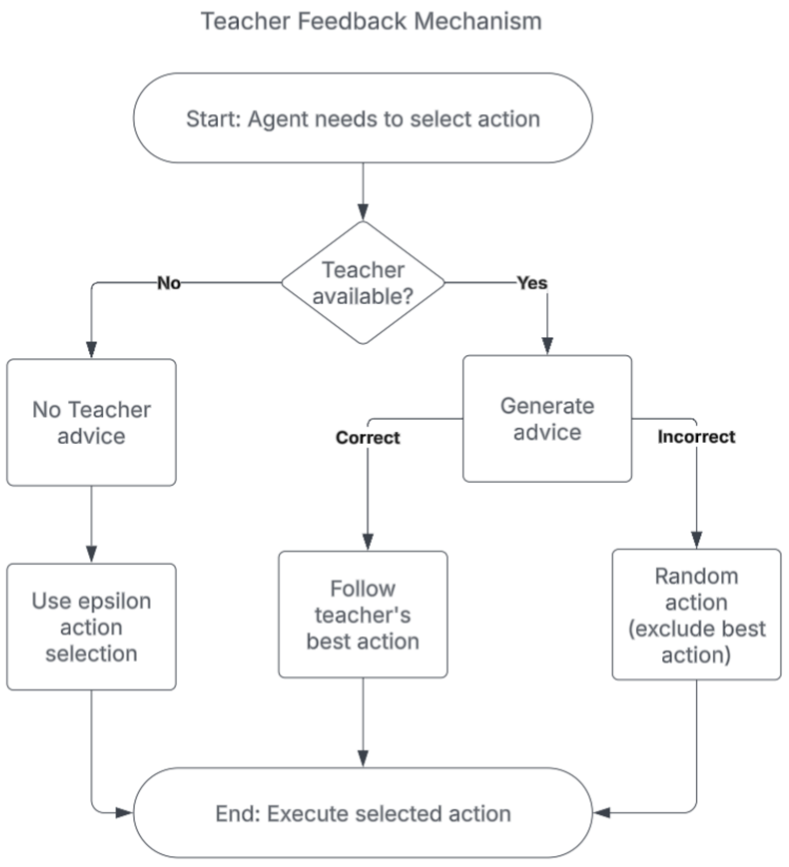

Teacher Feedback Mechanism
Start: Agent needs to select action
Teacher available?
No
Yes
No Teacher advice
Generate advice
Correct
Incorrect
Use epsilon action selection
Follow teacher's best action
Random action (exclude best action)
End: Execute selected action

Figure 5: Flowchart showing the teacher feedback mechanism. The student agent's action selection depends on two probability checks: availability (whether the teacher provides advice) and accuracy (whether the advice is correct).

8 Task 3: Teacher Advice Using Q-learning Agent

Important: You must use the same hyperparameters (learning rate, discount factor, epsilon, episodes, and maximum steps) that you chose in Task 1 to ensure fair comparison across all tasks.

In this task, you will implement the teacher-student framework where a pre-trained Q-learning agent (from Task 1) acts as a teacher to guide a new Q-learning student agent.

Implementation Requirements

Your implementation should:

- Load the trained Q-table from Task 1 to use as the teacher
- Train a new Q-learning student agent with teacher guidance
- Implement the teacher feedback mechanism as described in Section 7
- Test all combinations of teacher availability and accuracy parameters

Parameter Combinations

You must evaluate the following parameter combinations using nested loops:

- **Availability**: [0.1, 0.3, 0.5, 0.7, 1.0]
- **Accuracy**: [0.1, 0.3, 0.5, 0.7, 1.0]

This results in 25 different teacher configurations to test.

Metrics to Track

For each teacher configuration, track these metrics during training:

- **Total Rewards per Episode**: The cumulative reward accumulated during each episode
- **Steps per Episode**: The number of steps taken to complete each episode
- **Successful Episodes**: Whether the agent reached the goal

Required Outputs

After training with all parameter combinations, you must:

- Calculate performance metrics for each configuration:
    - **Success Rate** using Equation 1
    - **Average Reward per Episode** using Equation 2
    - **Average Learning Speed** using Equation 3
- Store all results in a structured format with the following data:
    - Availability

- Accuracy
- Avg Reward
- Success Rate (%)
- Avg Learning Speed

– Generate a heatmap visualisation showing average rewards for all teacher configurations:
  - X-axis: Availability values
  - Y-axis: Accuracy values
  - Colour intensity: Average reward achieved
  - Include appropriate colour bar and labels

Figure 6 shows a sample of what your teacher performance heatmap should look like (generated with simulated data).

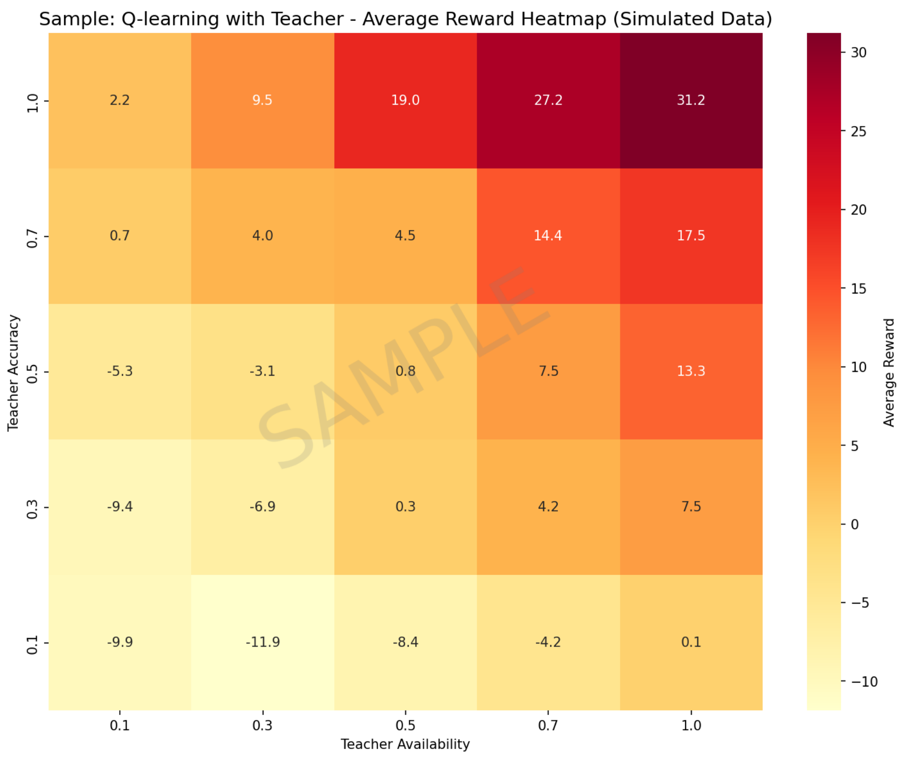


Figure 6: Sample heatmap showing Q-learning performance with different teacher configurations. Note that accuracy increases from bottom to top. This is generated with simulated data for demonstration purposes only.

9 Task 4: Teacher Advice Using SARSA Agent

Important: You must use the same hyperparameters (learning rate, discount factor, epsilon, episodes, and maximum steps) that you chose in Task 1 to ensure fair comparison across all tasks.

In this task, you will implement the teacher-student framework where a pre-trained SARSA agent (from Task 2) acts as a teacher to guide a new SARSA student agent.

Implementation Requirements

Your implementation should:

- Load the trained Q-table from Task 2 to use as the teacher
- Train a new SARSA student agent with teacher guidance
- Implement the teacher feedback mechanism as described in Section 7
- Test all combinations of teacher availability and accuracy parameters
- Use the same implementation structure as Task 3 for consistency

Parameter Combinations

You must evaluate the following parameter combinations using nested loops:

- **Availability**: [0.1, 0.3, 0.5, 0.7, 1.0]
- **Accuracy**: [0.1, 0.3, 0.5, 0.7, 1.0]

This results in 25 different teacher configurations to test.

Metrics to Track

For each teacher configuration, track these metrics during training:

- **Total Rewards per Episode**: The cumulative reward accumulated during each episode
- **Steps per Episode**: The number of steps taken to complete each episode
- **Successful Episodes**: Whether the agent reached the goal

Required Outputs

After training with all parameter combinations, you must:

- Calculate performance metrics for each configuration:
  - **Success Rate** using Equation 1
  - **Average Reward per Episode** using Equation 2
  - **Average Learning Speed** using Equation 3
- Store all results in a structured format with the following data:
  - Availability
  - Accuracy
  - Avg Reward

    - ◦ Success Rate (%)
    - ◦ Avg Learning Speed
- Generate a heatmap visualisation showing average rewards for all teacher configurations:
    - X-axis: Availability values
    - Y-axis: Accuracy values
    - Colour intensity: Average reward achieved
    - Include appropriate colour bar and labels

This will allow direct comparison with the Q-learning teacher results from Task 3 to determine which algorithm provides better teaching capabilities.

10 Testing and Discussing Your Code

After completing all four tasks, you should analyse and compare your results to understand the impact of teacher guidance on reinforcement learning performance.

10.1 Required Analysis

You must perform the following analysis to demonstrate your understanding:

1. Teacher Impact on Learning Curves

For selected teacher availability levels (e.g., 0.1, 0.5, 1.0), create plots showing how different teacher accuracies affect learning progress compared to the baseline. Each plot should show:

- Episode rewards (50-episode moving average) on the y-axis
- Episodes on the x-axis
- Multiple lines for different accuracy levels
- Baseline performance for reference

Figure 7 shows a sample comparison for Q-learning with 50% teacher availability.

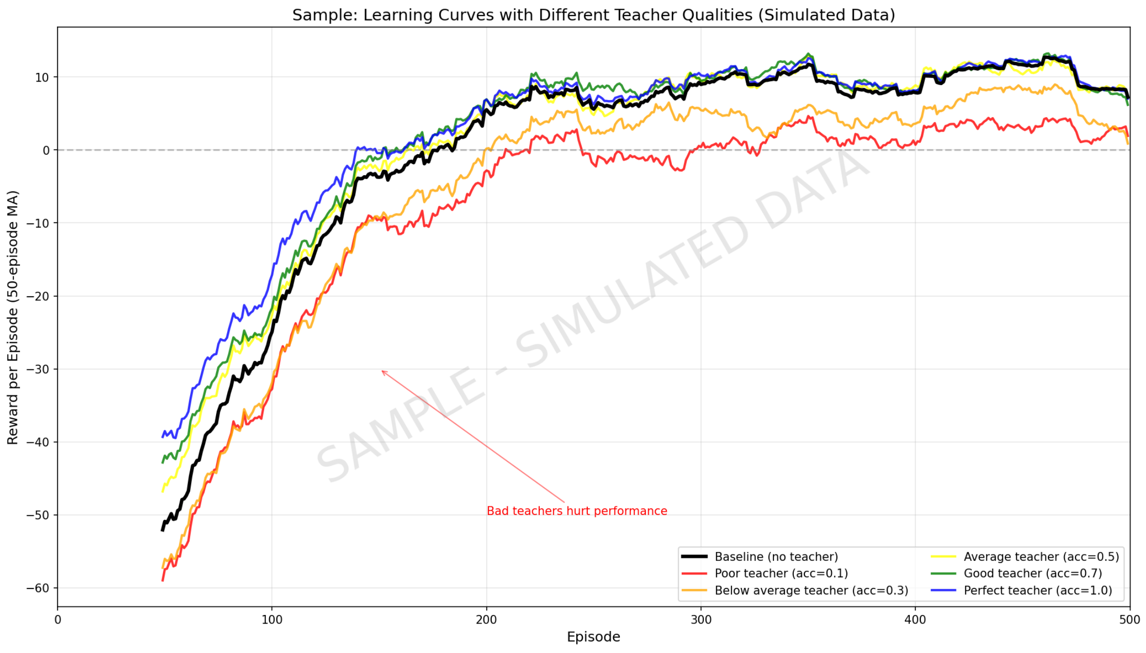


Figure 7: Sample comparison of teacher accuracy impact on Q-learning performance with 50% availability. Generated with simulated data.

2. Teacher Effectiveness Summary

Create a comprehensive analysis comparing how teacher guidance affects both algorithms:

- Generate learning curves showing Q-learning and SARSA performance with selected combinations of teacher availability and accuracy values
- Include baseline performance (no teacher) as a reference line
- Show multiple combinations of teacher availability (e.g., 0.1, 0.5, 1.0) and accuracy (e.g., 0.3, 0.7, 1.0) to demonstrate the range of teacher impact
- Use moving averages to smooth the learning curves for clarity

Figure 8 shows a sample teacher effectiveness summary analysis.

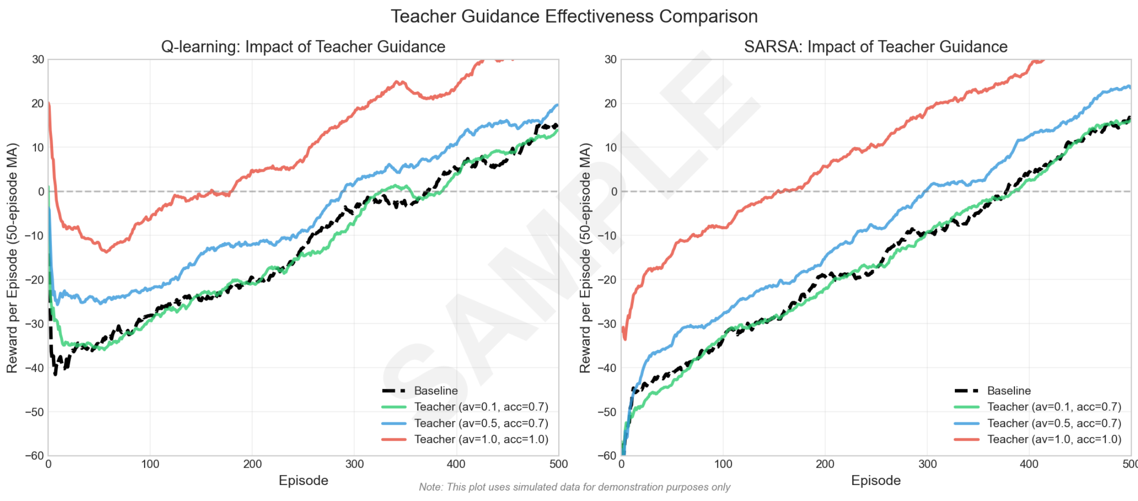


Figure 8: Sample teacher effectiveness summary showing the impact of different teacher configurations on both Q-learning and SARSA algorithms. This analysis helps identify optimal teacher parameters and compare algorithm responsiveness to guidance.

11 Marking Scheme

You will receive marks for each section as shown in Table 1.

Table 1: Marking scheme for the assignment.

Criteria	**Marks**
Implementation (4 marks total)	
Task 1: Q-learning implementation with performance plot and metrics (Success Rate, Average Reward, Average Learning Speed)	1 mark
Task 2: SARSA implementation with performance plot and metrics (Success Rate, Average Reward, Average Learning Speed)	1 mark
Task 3: Q-learning with teacher implementation and heatmap	1 mark
Task 4: SARSA with teacher implementation and heatmap	1 mark
Analysis and Discussion (20 marks total)	
Baseline comparison: Comparison plot and analysis	1 mark
Task 3 discussion: Analysis of Q-learning with teacher results	8 marks
Task 4 discussion: Analysis of SARSA with teacher results	8 marks
Teacher effectiveness summary: Comparative visualisation and insights	3 marks
Code Quality (1 mark total)	
Clear, well-commented, and organised code	1 mark
Total	**25 marks**

12 Resources Provided

The following resources are provided to support your implementation:

1. **`env.py`** – Grid World Environment Implementation
   - Contains the `GridWorldEnv` class implementing the 11×11 grid world
   - Includes obstacle positions, reward structure, and state transition logic
   - Provides optional visualisation using Pygame
   - Supports reproducible experiments through random seed parameter
2. **`Environment_Guide.pdf`** – Comprehensive Environment Documentation
   - Complete API reference for all environment methods
   - Usage examples and code snippets
   - Troubleshooting guide for common issues
   - Integration guidelines for your reinforcement learning algorithms
3. **`images/`** – Visualisation Assets (folder)
   - `grid_agent.png` – Agent representation
   - `grid_goal_position.png` – Goal location marker
   - `grid_obstacle.png` – Obstacle representation

Important: You must thoroughly read the Environment Guide before starting implementation. The guide contains essential information about:

- Coordinate system conventions
- Action space encoding
- Reward structure details
- Proper environment usage patterns
- Jupyter notebook setup requirements

All provided files must remain in their original directory structure for the environment to function correctly.

13 Submission Guidelines

13.1 Submission Requirements

You must submit your assignment through Moodle before the deadline. Your submission should consist of:

1. **Single Jupyter Notebook** (`.ipynb` format)
   - Filename format: `z1234567_COMP9414_Assignment2.ipynb` (replace with your zID)
   - Must contain complete implementations for all four tasks
   - All cells must be executed with outputs visible

- Include markdown cells explaining your approach and analysis

13.2 Notebook Structure Requirements

Your notebook must be organised as follows:

1. **Header Section**
   - Your name and zID
   - Assignment title
   - Brief description of contents
2. **Task Sections** (clearly labelled for each task)
   - Task 1: Q-learning Implementation
   - Task 2: SARSA Implementation
   - Task 3: Q-learning with Teacher
   - Task 4: SARSA with Teacher
3. **Each Task Section Must Include:**
   - Algorithm implementation code
   - Training execution and results
   - Required visualisations (plots, heatmaps)
   - Performance metrics calculations

13.3 Code Quality Standards

Your submitted code must adhere to the following standards:

- **Comments:** Add inline comments explaining complex logic
- **Variable Names:** Use descriptive, meaningful variable names
- **Code Structure:** Organise code into logical functions, avoid repetition
- **Reproducibility:** Set random seeds for consistent results

13.4 Pre-submission Checklist

Before submitting, ensure you have:

✓ Executed all cells in order (Kernel → Restart & Run All)

✓ Verified all plots and outputs are visible

✓ Included all required analysis and discussion text

✓ Checked that your notebook runs without errors on a clean kernel

✓ Saved the final version of your notebook

✓ Named your file correctly with your zID

13.5 Important Notes

! **Late Penalty:** 5% per day (or part thereof) will be deducted for late submissions

! **File Format:** Only `.ipynb` files will be accepted (not `.py` or `.pdf`)

! **Environment Rendering:** Do not include `env.render()` calls in your final submission as they may cause issues during marking

14 Deadline and Questions

Deadline: Week 9, Friday, 1 August 2025, 5:00 PM AEST.

Please use the forum on Moodle to ask questions related to the assignment. We will prioritise questions asked in the forum. However, you should not share your code to avoid making it public and possible plagiarism. If that's the case, contact the course admin Maryam Hashemi at `m.hashemi@unsw.edu.au` or the lecturer Francisco Cruz Naranjo at `f.cruz@unsw.edu.au` as an alternative.

Although we try to answer questions as quickly as possible, we might take up to 1 or 2 business days to reply; therefore, last-moment questions might not be answered timely.

For any questions regarding the discussion sessions, please contact your tutor directly. You can find your tutor's contact information in Table 2.

Table 2: COMP9414 25T2 Tutorials

No.	**Class ID(s)**	**Tutor**	**Email**
1	4374, 4383	Dr Jingying Gao	`jingying.gao@unsw.edu.au`
2	4375, 4376	Kiran Jeet Kaur	`kiran_jeet.kaur@unsw.edu.au`
3	4377, 4381	Leman Kirme	`l.kirme@unsw.edu.au`
4	4378, 4389	Xinyi Li	`xinyi.li17@student.unsw.edu.au`
5	4379, 4385	John Chen	`xin.chen9@student.unsw.edu.au`
6	4380, 4399	Abhishek Pradeep	`abhishek.pradeep@student.unsw.edu.au`
7	4382, 4394	Janhavi Jain	`j.jain@unsw.edu.au`
8	4384, 4393	Maher Mesto	`m.mesto@unsw.edu.au`
9	4386, 4391	Peter Ho	`peter.ho2@student.unsw.edu.au`
10	4387, 4390	Yixin Kang	`yixin.kang@student.unsw.edu.au`
11	4388, 4397	Jonas Macken	`j.macken@student.unsw.edu.au`
12	4392, 4405	Malhar Patel	`malhar.patel@unsw.edu.au`
13	4395, 4401	Ramya Kumar	`ramya.kumar1@unsw.edu.au`
14	4398, 4402	Zahra Donyavi	`z.donyavi@unsw.edu.au`
15	4396, 4403	Hadha Afrisal	`hadha.afrisal@unsw.edu.au`
16	4400, 4404	Joffrey Ji	`joffrey.ji@student.unsw.edu.au`

15 Late Submission Policy

UNSW has a standard late submission penalty of **5% per day** for all assessments where a penalty applies, capped at five days (120 hours) from the assessment deadline, after which a student cannot submit an assessment.

Work submitted after 5 days (120 hours) will not be accepted and will receive a mark of zero.

The penalty applies to the total available marks for the assignment. For example, if the assignment is worth 25 marks and is submitted 2 days late, the student will lose 2.5 marks (10% of 25).

Late work will not be accepted without penalty unless an extension has been previously granted. If you require an extension due to illness or misadventure, you must submit a Special Consideration request through UNSW's formal process with supporting documentation before the deadline.

16 Plagiarism Policy

Your programme must be entirely your own work. Plagiarism detection software may be used to compare submissions pairwise (including submissions from previous years), and serious penalties will be applied in cases of plagiarism, particularly for repeat offences.

Do not copy from others. Do not allow anyone to see your code. We encourage you to tackle this assignment independently to maximise your learning experience. Engaging fully with the assignment tasks will enhance your understanding of reinforcement learning concepts and algorithms.

Please refer to the UNSW Policy on Academic Integrity for further clarification: `https://student.unsw.edu.au/plagiarism`.