Assignment 3 [MAI] - Final Project: Reinforcement Learning and DL Agents

Course Code and Name: COSC3145 Games and Artificial Intelligence Techniques

Assessment type: Code and output submission

Due Date: January 17, 2026, 11:59 PM

Type: Group Assignment

Late work: None

Weighting: 50%

Late submission penalty: Deduct 2 marks per day late unless special consideration has been granted. Submission after 5 days will be awarded 0 mark for this assignment.

Overview

Reinforcement learning is a key method used in real world autonomous systems including robotics, self driving cars, drone navigation, warehouse automation, adaptive game AI, traffic control, and simulation based training. This project gives you practical experience building such systems from the ground up.

You will create two interactive environments in Python and train agents that learn through interaction. **Part I** focuses on classical value based reinforcement learning inside a visual gridworld. These ideas form the basis of many real decision systems used in planning, logistics, routing, and simple autonomous navigation.

Part II expands to a real time Pygame arena where the agent learns movement, survival and progression using deep reinforcement learning with Stable Baselines3. This reflects the type of work done in modern game AI, simulation engineering, real time control, and robotics.

Masters students also complete **Part III**, which introduces research style experimentation similar to professional RL studies.

By the end of the assignment you will have designed simulation environments, defined state and reward models, trained agents, analysed learning behaviour with real logs, and demonstrated intelligent behaviour in visual interactive scenes. These are directly transferable skills to real world applications.

Assessment Details

Part I. Classical Reinforcement Learning (15 pts)

In this part you will implement Q learning and SARSA in a visual gridworld created in Python. Your gridworld must be implemented and visually rendered in Pygame and must support interaction and animation. Console or text based displays are not permitted.

1. Gridworld rules

- The agent can move: **up, down, left, right**.
- Rocks block movement. Attempting to move into one results in no movement.
- Stepping into fire or monsters results in immediate death.
- Apples give **+1 reward**.
- Keys give **no reward** but allow opening chests.
- Opening a chest gives **+2 reward**.
- The episode ends when all collectible rewards are obtained or the agent dies.
- After each agent action, monsters (if present) have a 40 percent chance of moving.

You may add helper functions if needed but must not alter rewards or mechanics. You must create multiple levels with different layouts.

Task 1: Implement Basic Q-Learning for Level 0

Level 0 contains only apples on the right side of the map.
Your Q-learning implementation must:

- Use an epsilon-greedy policy
- Update Q-values according to the Q-learning rule
- Use linear epsilon decay from epsilonStart to epsilonEnd
- Use random tie-breaking when multiple actions have equal value
- Successfully learn a shortest-path policy to the apples

Training parameters (episodes, alpha, gamma, epsilon ranges) will be provided in a config file.

Task 2: Implement Basic SARSA for Level 1

Implement SARSA using on-policy updates:

- Use the same exploration schedule as Q-learning
- Demonstrate that the learned policy differs from Q-learning, typically being more conservative around hazards

---

Task 3: Extend Q-Learning and SARSA to Levels 2–3

Levels 2–3 introduce:

- Multiple apples
- A key
- A chest

Task 4. Monster Levels (Levels 4 and 5)

What to implement

- Monsters that move after each agent action.
- Movement is probabilistic (for example 40 percent chance to move).
- Monsters use a simple pattern such as choosing randomly from allowed directions.
- If the player enters a monster tile or a monster moves into the player, the player dies.

What is required from the RL side

- Your Q learning and SARSA implementations must handle stochastic transitions.
- The agent should learn to avoid monsters while still completing objectives.

Evidence to include

- Working monster movement in your gridworld.
- Training curves showing learning behaviour on levels 4 and 5.

Task 5. Intrinsic Reward for Level 6

What to implement

Intrinsic reward:

$$r_i = \frac{\mathbf{intrinsicRewardStrength}}{\sqrt{n(s)+1}}$$

Where:

- n(s) = number of visits to the current state during the episode
- total reward = environment reward + intrinsic reward

Requirements

- Keep all environment rewards unchanged.
- Maintain a visit counter for each state per episode.
- Agent must use intrinsic reward during Q or SARSA updates.

Evidence to include in your report

- Training curves comparing learning with and without intrinsic reward.
- Short explanation of the improvement observed.

Part II. Deep Reinforcement Learning in a Pygame Arena (20 marks)

In this part you will design a real time Pygame arena and train deep RL agents inside it using Stable Baselines3. The simulation must be your own design and must be visually animated. The agent will learn from a continuous observation vector and train with a neural network based algorithm.

1- Environment requirements

Your Pygame scene must include:

- A controllable player ship with movement and shooting
- Enemy spawners that periodically create enemies
- Enemies that navigate toward the player
- Player health
- Enemy health
- Projectile collisions
- A phase system where destroying all active spawners progresses the simulation to the next difficulty level

The game must feel like a simplified action arena instead of a tile based grid. Movements should be continuous or semi continuous.
All elements must be visual on screen.

The episode ends when:

- The player dies
- A maximum time or step count is reached

2 Gym style API

Your environment must expose these methods:

- ***reset()*** returns an initial observation
- ***step(action)*** applies an action and returns observation, reward, done, info
- ***render()*** displays the scene for evaluation

3 Observation design

Your agent must receive a fixed size vector containing at least

- Player position
- Player velocity
- Player orientation if relevant
- Distance and relative direction to nearest enemy
- Distance and relative direction to nearest spawner
- Player health
- Current phase

You might consider not using pixels or screenshots. Feature vectors are preferred and they must be numeric and fixed size.

4 Action sets

You must implement two distinct control schemes and train a separate agent for each one.

Control style 1. Rotation movement and thrust

0. No action
1. Thrust forward
2. Rotate left
3. Rotate right
4. Shoot

Control style 2. Direct directional movement

0. No action
1. Move up
2. Move down
3. Move left
4. Move right
5. Shoot

Each style must produce a trained model and an evaluation script.

Each must produce a separate trained model.

5 Reward function

You must define a reward structure that encourages intentional progression, for example:

- Positive reward for destroying enemies
- Larger positive reward for destroying spawners
- Positive reward when progressing to the next phase
- Negative reward when taking damage
- Strong negative reward on death
- Optional shaping rewards must be justified

Reward design is part of the assessment. You might use dditional shaping with justification.

6 Deep RL training

You are advised to use **Stable Baselines3** with either:

- DQN
- PPO

Expectations:

- Use neural networks with at least one hidden layer
- Train using Stable Baselines3 while logging results to TensorBoard
- Tune hyperparameters in a meaningful way
- Save trained models in a folder named models
- Provide an evaluation script able to visually play the agent in the arena

Part III. Advanced RL Extension (5)

Masters students must complete one advanced pathway. Undergraduates skip this section.

Option A. Ablation study

Conduct three controlled ablations on your agent or environment.
Compare training curves and final performance.
Discuss which design choices matter most.

Option B. Second RL algorithm comparison

Train a second deep RL algorithm and compare learning behaviour, stability and final results.

Option C. Curriculum learning experiment

Design a curriculum that changes environment difficulty during training.
Evaluate whether it improves speed or stability.

A full writeup must be included in the report.

Report requirements (10 marks)

Maximum length: 10 pages including images, no appendix [this requirement is strict, everything outside of the 10th page will not be considered].

Your report must include:

1. Description of both environments
2. Observation design
3. Reward design
4. Hyperparameter exploration
5. Comparison of control sets
6. Evidence of training with logs and screenshots
7. Originality justification
8. Masters extension section if applicable

Submission requirements [STRICT]

You must submit exactly the 2 following items (Zip code, PDF Report including Video Demonstration):

1. A zip file containning:

- All gridworld code
- Arena simulation code
- RL training scripts
- Saved models
- TensorBoard logs
- Screenshots

2. Your report PDF

- Include all student numbers and contribution summary
- A Link to your Video Recording Demonstration

According to the recorded video demonstration:

- Each team must submit a video demonstrating their project. This video must clearly show that the implemented environments and agents work as described. Length: Max 10 minutes

Length and requirements:

- Maximum: 10 minutes
- All members of the group need to appear in the recorded video and must present at least 1 part.

What the video must show:

1. Gridworld (Part I)

- The gridworld visible in a Pygame window.
- The agent running using Q learning or SARSA on one level.
- Monsters or items behaving correctly (if applicable to that level).
- Evidence that the agent is following its learned policy, not random actions.

2. Arena Simulation (Part II)

- The real time Pygame arena running visually.
- The trained deep RL agent controlling the player.
- Enemies spawning and moving.
- Projectiles and collisions functioning.
- Phase progression occurring at least once.
- The agent demonstrating learned behaviour for **both control sets** (two short clips are fine).

Requirements for acceptability:

- The environment and agent behaviour must match the submission code.
- The models shown must be the ones included in the repository.
- The video must demonstrate actual gameplay, not static screens.

How to submit

- Upload the video to YouTube (unlisted) or saved int on your university OneDrive
- Provide a link in the PDF Report.

You may reuse any code or materials that you created in previous assignments or tutorials.

Appendix. Technical Expectations and Feasibility Guide

To ensure the project remains feasible on standard hardware, here is the technical expectations. The below is not a requirement but it is a feasible guide to help you.

Gridworld

- Use grids around 10 by 10 or 12 by 12
- Use simple shapes in Pygame
- Q learning and SARSA must run quickly

Arena simulation

- Window size around 800 by 600 or 960 by 680
- Use clear shapes for players, enemies and bullets
- Keep enemy count and spawner frequency manageable
- Physics should remain simple
- Render only during evaluation, not during long training runs

Observation vector

- Should contain around 10 to 30 float features
- Must be fixed size
- Must not use pixels or images

RL training

- Use Stable Baselines3
- Networks should be small MLPs
- Train headless for speed
- Total training typically between 100 thousand and 600 thousand timesteps
- Use TensorBoard for monitoring

These are not requierments but following these expectations ensures that your assignment will run efficiently on standard personal computers.

Learning Objectives Assessed

This assessment relates to the following learning outcomes:

- **[CLO1]:** Apply various AI techniques and tools in the context of games
- **[CLO2]:** Design and develop a gaming application, based on existing games engines
- **[CLO3]:** Work effectively in a team environment to develop a complex software system.

Ready for Life and Work

- **Enabling Knowledge**:
  By engaging in this assignment, you will enhance your skills by applying theoretical knowledge creatively and proactively in a practical context. You will demonstrate a comprehensive understanding of AI techniques and game development principles, including recent advancements in computer science and information technology, as you implement steering behaviors and finite state machines within a predator-prey game framework.
- **Critical Analysis**:
  You will develop the ability to rigorously examine and critically evaluate concepts related to computer science and data science as they pertain to game development. This involves analyzing the requirements and constraints of the game mechanics, modeling the interactions between characters, and understanding how these elements contribute to an engaging gameplay experience.
- **Problem Solving**:
  Your problem-solving skills will be refined as you tackle complex challenges inherent in game design and AI implementation. You will learn to design and develop software solutions that meet specific gameplay requirements, enabling you to synthesize appropriate techniques and tools that enhance the functionality and interactivity of the game, ultimately preparing you for real-world applications in the gaming industry.

Support Resources

This assessment requires that you meet RMIT's expectations for academic integrity. More information and advice on how to avoid plagiarism are available in the Getting Started module.

> Open **the academic integrity page**.

Additional library and learning resources are available to help with the assessment in this course

> Link to **Assignment Support**.

Submission Instructions

You need to submit a zip file on Canvas for this assignment as explained in the assignment description. Please note:

- Late submissions will incur a penalty of 10% of the total score possible per calendar
- If you do submit late, please let us know so that we download the correct version

- Information on applying for special consideration is available here: https://www.rmit.edu.au/students/my-course/assessment-results/special-consideration- extensions/special-consideration

Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarized, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods.
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source.
- Copyright material from the internet or databases.
- Collusion between students.

NOTE

This assignment specification may be corrected after release, either to fix mistakes or to provide more information. If so, the corrected version will be uploaded to Canvas as quickly as possible and an announcement will be posted.