

Tutorial 1: The Lunar Lander

Feature extraction and manual programming of an agent.

February 3, 2026

- The aim of this tutorial is to get familiar with the Lunar Lander environment, which will be used in some practical lessons of the course.
- The tutorial can be done in Linux, Windows or Mac.
- It is important to solve the exercises in order.

1 Introduction

The practical part of this course is based on the Lunar Lander environment from Gymnasium, a toolkit for solving reinforcement learning problems. We will use this environment to develop autonomous agents while exploring and understanding some machine learning concepts but we will still not use its reinforcement learning capabilities.

The objective of the game is to land a spacecraft safely on the landing pad located at coordinates (0, 0). The landing pad is marked by two flags. The lander starts at the top center of the screen with a random initial force applied to it. The player (or agent) can control the spacecraft using three engines:

- **Main engine:** Fires downward, slowing the descent
- **Left orientation engine:** Rotates the lander clockwise
- **Right orientation engine:** Rotates the lander counter-clockwise

The episode ends when the lander crashes (touches the ground with too much velocity or at a bad angle), lands successfully, or flies out of bounds. The score depends on how well the landing is performed:

- Landing on the pad: +100 to +140 points
- Each leg contact with ground: +10 points
- Firing main engine: -0.3 points per frame
- Firing side engines: -0.03 points per frame
- Crash or out of bounds: -100 points

During the course of the game, we will have access to the game state (observation), information we will use to generate an autonomous agent.

2 Python Programming Language and Gymnasium Environment

The Lunar Lander environment is accessed through Python using the Gymnasium library, so we will use this tutorial also to get familiar with this library.

2.1 About Gymnasium

Gymnasium (formerly OpenAI Gym) is a standard API for reinforcement learning and a diverse collection of reference environments. It is maintained by the Farama Foundation and provides a simple, unified interface to interact with many different environments.

The key concepts in Gymnasium are:

- **Environment:** The world in which an agent operates. In our case, the Lunar Lander simulation.
- **Observation:** Information about the current state of the environment that the agent receives (the 8 values described in Section 4).
- **Action:** A decision made by the agent that affects the environment (the 4 actions described in Section 5).
- **Reward:** A numerical signal that indicates how well the agent is performing.
- **Episode:** A sequence of interactions from an initial state until a terminal state (landing, crashing, or going out of bounds).

The basic interaction loop with a Gymnasium environment follows this pattern:

```
import gymnasium as gym

# Create the environment
env = gym.make("LunarLander-v3", render_mode="human")

# Reset to get initial observation
observation, info = env.reset()

# Game loop
while True:
    action = choose_action(observation) # Your agent decides
    observation, reward, terminated, truncated, info = env.step(action)

    if terminated or truncated:
        observation, info = env.reset() # Start new episode

env.close()
```

For more information about Gymnasium, visit the official documentation:

<https://gymnasium.farama.org/>

2.2 Installation

For executing the exercise code, you need to install the “gymnasium” library with the Box2D extra. You can do this using pip:

```
pip install "gymnasium[box2d]"
```

Or by creating a conda environment (recommended, see Appendix A for detailed instructions).

3 Code Execution

1. Download the Lunar Lander code from Aula Global (**LunarLander.py** file).
2. Open a terminal and navigate to the directory where the file is located.
3. Launch the game by executing the following command in the terminal:

```
python LunarLander.py
```

By default, the game starts with the lander being controlled with the keyboard.

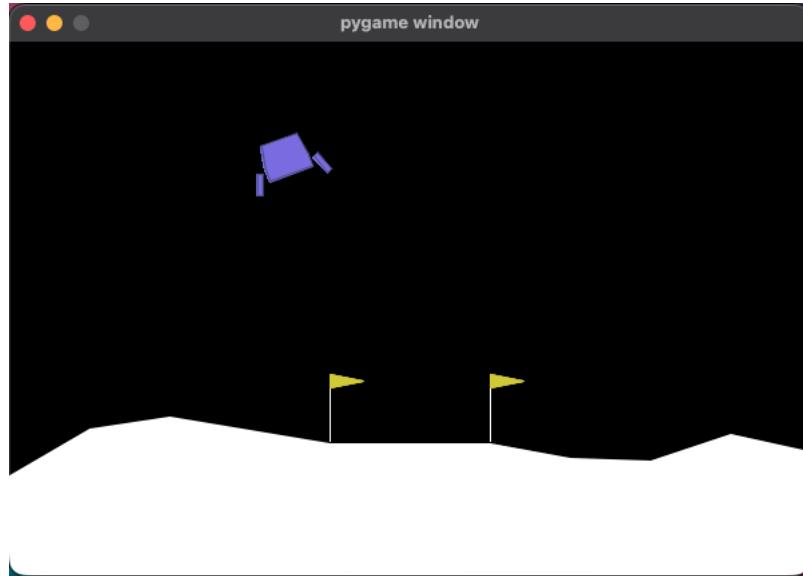


Figure 1: Lunar Lander interface. The lander must land on the pad between the two flags.

4 The Observation Space

The Lunar Lander environment provides an observation (state) consisting of 8 continuous values:

1. **obs[0] - X Position:** Horizontal position of the lander (0 is the center/landing pad)
2. **obs[1] - Y Position:** Vertical position of the lander (0 is ground level)
3. **obs[2] - X Velocity:** Horizontal velocity (positive = moving right)
4. **obs[3] - Y Velocity:** Vertical velocity (negative = falling down)
5. **obs[4] - Angle:** Angle of the lander in radians (0 = upright)
6. **obs[5] - Angular Velocity:** Rotation speed
7. **obs[6] - Left Leg Contact:** 1.0 if left leg is touching ground, 0.0 otherwise
8. **obs[7] - Right Leg Contact:** 1.0 if right leg is touching ground, 0.0 otherwise

5 The Action Space

The agent can choose from 4 discrete actions:

- **Action 0:** Do nothing
- **Action 1:** Fire left orientation engine (rotates clockwise)
- **Action 2:** Fire main engine (slows descent)
- **Action 3:** Fire right orientation engine (rotates counter-clockwise)

6 Exercises

Before starting the following tutorial, it is highly recommended that you open, read and understand the *LunarLander.py* file thoroughly to familiarize yourself with the code. After that, please run the game and answer the following questions.

1. What information is shown in the interface? And in the terminal?
2. What happens if you change the value of the GRAVITY variable?
3. In your opinion, which data from the observation could be useful to decide what action the lander should take on each tick? Explain your reasoning.
4. Complete the method called *print_line_data(game)* inside the file. This method should return a string with the information from the game state you consider relevant. Then, call this method from the main loop to write the information to a CSV file.

For each tick, you should store a line with all the considered information, splitting each data with commas. The first line of the file should contain the column headers.

Moreover, each time a new game starts, the new lines must be appended below the old ones. You should not rewrite the file when a new episode starts.

5. Program an “intelligent” lander using your own rules. To do so, complete the method *move_tutorial_1(game)*. Your new agent should control the lander to land safely on the pad, maximizing the score.

Consider the following tips for your implementation:

- Control the descent speed using the main engine (y_velocity should be small when landing)
- Keep the lander upright by controlling the angle (should be close to 0)
- Use horizontal position and velocity to center over the landing pad
- Remember that using engines costs points, so use them efficiently

6. The agent you just programmed does not use any machine learning technique. What advantages do you think machine learning may have to control the Lunar Lander?

7 Files to Submit

All the lab assignments **must** be done in groups of 2 people. You must submit a .zip file containing the required material through Aula Global before the following deadline: **Tuesday, February 10th at 17:00**. The name of the zip file must contain the last 6 digits of both student’s NIA, i.e., `tutorial1-123456-234567.zip`

The zip file must contain the following files:

1. A **PDF** document with:
 - Cover page with the names and NIAs of both students.
 - Answers to all the questions appearing in the Exercises section.
 - Description of the method implemented to save the game state on each tick.
 - Description of the behavior programmed for the **Tutorial 1 Agent**.
 - No code or code screenshots should be included in the document.
 - Conclusions.
2. Include in a folder called “material” the following files:
 - The modified **LunarLander.py** file with the method that saves the game state on each tick and the new tutorial 1 agent method implemented.
 - A sample CSV file generated by your data logging method.

Please, **be very careful and respect the submission rules**.

A Environment Setup with Conda

This appendix provides detailed instructions for setting up a Python environment using Conda, which is the recommended approach to avoid conflicts with other Python packages on your system.

A.1 Installing Miniconda

If you don't have Conda installed, download and install Miniconda (a minimal installer for Conda) from:

```
https://docs.conda.io/en/latest/miniconda.html
```

Follow the installation instructions for your operating system (Windows, macOS, or Linux).

A.2 Creating a Conda Environment

Open a terminal (or Anaconda Prompt on Windows) and run the following commands:

```
# Create a new environment named 'ml-lunar' with Python 3.10
conda create -n ml-lunar python=3.10 -y

# Activate the environment
conda activate ml-lunar

# Install the required packages
pip install "gymnasium[box2d]"
```

A.3 Using the Environment

Every time you want to work on this tutorial, you need to activate the environment first:

```
conda activate ml-lunar
```

Then you can run the Lunar Lander code:

```
python LunarLander.py
```

To deactivate the environment when you're done:

```
conda deactivate
```

A.4 Alternative: Using requirements.txt

If you have a `requirements.txt` file provided, you can install all dependencies at once:

```
conda activate ml-lunar
pip install -r requirements.txt
```

A.5 Troubleshooting

- **Box2D installation issues:** On some systems, you may need to install SWIG first. On Ubuntu/Debian: `sudo apt-get install swig`. On macOS with Homebrew: `brew install swig`.
- **Display issues on Linux:** If you encounter display errors, ensure you have the required graphics libraries installed and that you're not running in a headless environment.
- **Permission errors:** If you get permission errors during installation, do not use `sudo` with pip inside a conda environment. Instead, ensure your conda environment is properly activated.