# 6ELEN018W - Applied Robotics
# Lecture 10: The Gymnasium RL Environment

Dr Dimitris C. Dracopoulos

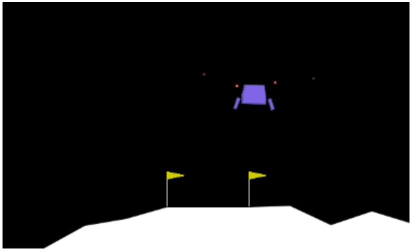
## Installing Gymnasium

Execute in a a terminal (inside Anaconda, if you are using Anaconda):

```
pip install swig
pip install gymnasium[toy_text]
pip install gymnasium[box2d]
```

You can follow the link below for more detailed instructions: https://github.com/Farama-Foundation/Gymnasium

## The Lunar Lander Problem



A classic rocket trajectory optimization problem.

- Decision: Fire engine on or off.

Action Space:

- 0: do nothing
- 1: fire left orientation engine
- 2: fire main engine
- 3: fire right orientation engine

## The Lunar Lander Problem (cont'd)

Observation Space:
An 8-dimensional vector:

- The coordinates of the lander in $x$ and $y$
- Linear velocities in $x$ and $y$
- Its angle
- its angular velocity
- 2 booleans that represent whether each leg is in contact with the ground or not.

## The Lunar Lander Problem (cont'd)

Rewards:
After every step a reward is granted. The total reward of an episode is the sum of the rewards for all the steps within that episode.
For each step, the reward:

- is increased/decreased the closer/further the lander is to the landing pad.
- is increased/decreased the slower/faster the lander is moving.
- is decreased the more the lander is tilted (angle not horizontal).
- is increased by 10 points for each leg that is in contact with the ground.
- is decreased by 0.03 points each frame a side engine is firing.
- is decreased by 0.3 points each frame the main engine is firing.

The episode receive an additional reward of -100 or +100 points for crashing or landing safely respectively.
$\longrightarrow$ An episode is considered a solution if it scores at least 200 points.

## The Lunar Lander Problem (cont'd)

Episode Termination:

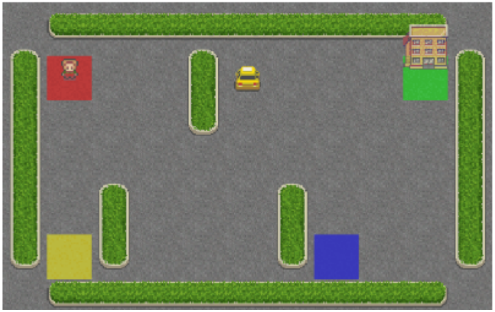
- the lander crashes (the lander body gets in contact with the moon)
- the lander gets outside of the viewport ($x$ coordinate is greater than 1);

## Example of Robot Agent (Random Policy) for Lunar Landing

```python
import gymnasium as gym

#env = gym.make("LunarLander-v3", render_mode="human")
env = gym.make("LunarLander-v3", render_mode="rgb_array")
observation, info = env.reset()

total_reward = 0
episode_over = False
while not episode_over:
    # agent random policy that uses the observation and info
    action = env.action_space.sample()
    observation, reward, terminated, truncated, info = env.step(action)

    print(f"Action: {action} - Observation: {observation}, reward: {reward}")
    total_reward += reward
    episode_over = terminated or truncated

print(total_reward)
env.close()
```

## The Taxi Driver Problem

trying to find the optimum path to a passenger location, pick up the passenger and then travel to a goal location and drop-off the passenger.



## The Taxi Driver Problem (cont'd)

The discrete environment is a 5x5 maze.

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

- There are four designated pick-up and drop-off locations (Red, Green, Yellow and Blue) in the 5x5 grid world. The taxi starts off at a random square and the passenger at one of the designated locations.
- The goal for the robot is to drive the taxi to the passenger's location, pick up the passenger, move to the passenger's desired destination, and drop off the passenger. Once the passenger is dropped off, the episode ends.

## The Taxi Driver Problem (cont'd)

Action Space:

- 0: Move south (down)
- 1: Move north (up)
- 2: Move east (right)
- 3: Move west (left)
- 4: Pickup passenger
- 5: Drop off passenger

## The Taxi Driver Problem (cont'd)

Rewards at each time step:

- -1 per step unless other reward is triggered.
- $+20$ delivering passenger.
- -10 executing “pickup” and “drop-off” actions illegally.
- An action that results a noop (no operation), like moving into a wall, will incur the time step penalty.

Episode end:

- Termination: 1. The taxi drops off the passenger.
- Truncation: The length of the episode is 200.

## The Taxi Driver Problem (cont'd)

Observation is returned as an integer that encodes the corresponding state, calculated by:

$$((taxi\_row * 5 + taxi\_col) * 5 + passenger\_location) * 4 + destination$$

**Encoding**:

```
def encode(taxi_row, taxi_col, pass_loc, dest_idx):
    # possible values for each argument: (5) 5, 5, 4
    i = taxi_row
    i *= 5
    i += taxi_col
    i *= 5
    i += pass_loc
    i *= 4
    i += dest_idx
    return i
```

## The Taxi Driver Problem (cont'd)

Observation is returned as an integer that encodes the corresponding state, calculated by:

$$((taxi\_row * 5 + taxi\_col) * 5 + passenger\_location) * 4 + destination$$

**Decoding**:

```
def decode(i):
    out = []

    out.append(i  4)
    i = i // 4
    out.append(i  5)
    i = i // 5
    out.append(i % 5)
    i = i // 5
    out.append(i)
    assert 0 <= i < 5

    return reversed(out)
```

# Example of Robot Agent (Random Policy) for the Taxi Driver Problem

```
import gymnasium as gym

env = gym.make("Taxi-v3", render_mode="rgb_array")
observation, info = env.reset()

total_reward = 0
episode_over = False
while not episode_over:
    # robot random policy that uses the observation and info
    action = env.action_space.sample()
    observation, reward, terminated, truncated, info = env.step(action)

    print(f"Action: {action} - Observation: {observation}, reward: {reward}")
    total_reward += reward
    episode_over = terminated or truncated

print(f'\n--> Total reward: {total_reward}')
env.close()
```