# Planning Lab - Lesson 1

## Uninformed Search

Luca Marzari and Alessandro Farinelli

University of Verona
Department of Computer Science

October 19, 2022


UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

# The OpenAI Gym Framework

## What is it

*Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball*

## What is it for

- An open-source collection of environments that can be used for benchmarks
- A standardized set of tools to define and to work with environments

## Where to find it

https://gym.openai.com

# Installation Process

During the lab lessons we will use Jupyter notebook files. In order to use these files you should install the following dependecies.

## Detailed guide for the installation process:

`https://github.com/LM095/Planning-Lab`

- Download the *Anaconda* package manager for Python 3.7 from `https://www.anaconda.com/distribution/#download-section`
- Install Conda on your system
- Open a terminal and digit:

```
> git clone https://github.com/LM095/Planning-Lab
> cd Planning-Lab
> conda env create -f tools/planning-lab-env.yml
> conda activate planning-lab
```

# Tutorial

To open the tutorial:

- Navigate to your local Planning-Lab folder.
- Ensure that you have activated the *planning-lab* conda environment and launch Jupyter Notebook (> jupyter notebook) from your folder
- Navigate with your browser to: *lesson_1/lesson_1_tutorial.ipynb*

# Assignments

- Your assignments for this lesson are at: *lesson_1/lesson_1_problem.ipynb*. You will be required to implement some Uninformed Search algorithms
- In the following you can find pseudocodes for such algorithms

# Uninformed Search: tree and graph search versions

**function** TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  **loop do**
    **if** the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    **if** the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

**function** GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  ***initialize the explored set to be empty***
  **loop do**
    **if** the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    **if** the node contains a goal state **then return** the corresponding solution
    ***add the node to the explored set***
    expand the chosen node, adding the resulting nodes to the frontier
      ***only if not in the frontier or explored set***

# Breadth-First Search (BFS): graph search version

**Require:** $problem$
**Ensure:** $solution$

1: $node \leftarrow$ a node with STATE $= problem$.INITIAL-STATE, PATH-COST $= 0$
2: **if** $problem$.GOAL-TEST($node$.STATE) **then return** SOLUTION($node$)
3: $frontier \leftarrow$ NODE-QUEUE
4: $explored \leftarrow \emptyset$
5: **while** Not IS-EMPTY($frontier$) **do**
6: $node \leftarrow$ REMOVE($frontier$) ▷ Remove last node
7: $explored \leftarrow explored \cup node.STATE$
8: **for each** $action$ **in** $problem$.ACTIONS($node$.STATE) **do**
9: $child \leftarrow$ CHILD-NODE($problem, node, action$)
10: **if** $child$.STATE **not in** $explored$ **or** $frontier$ **then**
11: **if** $problem$.GOAL-TEST($child$.STATE) **then return** SOLUTION($child$)
12: $frontier \leftarrow$ INSERT($child$)
**return** FAILURE

# Iterative Deepening Search (IDS): tree search version

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
      result ← DEPTH-LIMITED-SEARCH(problem, depth)
      if result ≠ cutoff then return result
```