# AI Lab - Session 1

## Uninformed Search

Riccardo Sartea

University of Verona
Department of Computer Science

March $15^{th}$ 2019


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

- Install the *Anaconda* package manager for Python 3.7 from `https://www.anaconda.com/distribution/`
- On linux use "`sh Anaconda...version.sh`" to install and add the bin folder to PATH when asked
- On linux reload bashrc with *"source ∼/.bashrc"*
- Use the following snippet of code to download an create the Python source code and environment

Listing 1: Installation

```
sudo apt-get install git (may be required)
git clone https://github.com/SaricVr/ai-lab
cd ai-lab
conda env create -f ai-lab-environment.yml
```

Then to start the environment and work on your assignments:

Listing 2: Spin up

```
conda activate ai-lab
jupyter notebook
```

The last command will open your browser for you to start working

# Tutorial

To open the tutorial navigate with your browser to:
*session1/session1_tutorial.ipynb*

# Assignments

Your assignments for this session are at: *session1/session1.ipynb*

# Algorithms

In the following you can find pseudocode of the algorithms are required to implement in this session

# Breadth-First Search (BFS)

**Input:** $problem$
**Output:** $solution$

1: $node \leftarrow$ a node with STATE $= problem$.INITIAL-STATE
2: **if** $problem$.GOAL-TEST($node$.STATE) **then return** SOLUTION($node$)
3: $fringe \leftarrow$ QUEUE, with $node$ as the only element
4: $closed \leftarrow \emptyset$
5: **loop**
6: **if** IS-EMPTY($fringe$) **then return** FAILURE
7: $node \leftarrow$ POP($fringe$) ▷ Remove node from frontier
8: $closed \leftarrow closed \cup node$
9: **for each** $action$ **in** $problem$.ACTIONS($node$.STATE) **do**
10: $child \leftarrow$ CHILD-NODE($problem, node, action$)
11: **if** $child$.STATE **not in** $fringe$ **and** $child$.STATE **not in** $closed$ **then**
12: **if** $problem$.GOAL-TEST($child$.STATE) **then return** SOLUTION($child$)
13: $fringe \leftarrow$ INSERT($child, fringe$)

Note: this is a graph search version

# Iterative Deepening Search (IDS)

```
1:  function ITERATIVE-DEEPENING-SEARCH(problem)
2:      for depth ← 0 to ∞ do
3:          result ← DEPTH-LIMITED-SEARCH(problem, depth)
4:          if result ≠ CUTOFF then return result

5:  function DEPTH-LIMITED-SEARCH(problem, limit)
6:      return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

7:  function RECURSIVE-DLS(node, problem, limit)
8:      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
9:      if limit = 0 then return CUTOFF
10:     cutoff_occurred ← False
11:     for each action in problem.ACTIONS(node.STATE) do
12:         child ← CHILD-NODE(problem, node, action)
13:         result ← RECURSIVE-DLS(child, problem, limit − 1)
14:         if result = CUTOFF then cutoff_occurred ← True
15:         else if result ≠ FAILURE then return result
16:     if cutoff_occurred then return CUTOFF
17:     return FAILURE
```

Note: this is a tree search version