

# Artificial Intelligent Systems: Models Final Projects

2021/2022

For all projects the candidate (s) is required to provide:

-the software source code (it is option but recommend to upload them on a specific GitHub repository)

-a brief project report containing:

- -the problem description and assumed hypothesis
- -algorithms and features of the adopted solution
- -discussion of complexity, efficiency issues
- -sample data and comparison experimental results

Further project proposal, or variations, can be proposed and agreed upon with the instructor.  
Project are assigne to **individual candidates**, or to **groups of max two students**:

## Programming Languages and Tools

The projects could use: differente software not provided such as: OpenCv, Watson Services, Google Tensorflow, Google Colab and other tool

For project in the Network Area we suggest tools such as:

**NetworkX**: Available in Python <https://networkx.org/>

**Snap**: Available in

C++ <http://snap.stanford.edu/snap/index.html>

Python <http://snap.stanford.edu/snappy/index.html>

**Graph-tool**: available in Python

<https://graph-tool.skewed.de/>

For project requiring Machine Learning or Reinforcement Learning components (recognizers/classifiers):

**Scikit-learn**: all types of classifier, clustering and regression algorithms (SVM, decision tree, ecc.), algoritmi di clustering e per la regressione. Available in Python

<https://scikit-learn.org/stable/>

**Keras**: for deep learning. Choose a backend (for example Tensorflow)

<https://keras.io/>

Among the many aailable tutorials for beginners:

<https://elitedatascience.com/keras-tutorial-deep-learning-in-python>

<https://machinelearningmastery.com/tutorial-nirst-neural-network-python-keras/>

## AREA: Social Graph and Complex Network

### Project: Epidemic Super Spreaders in Networks

The purpose is to implement a systems to simulate epidemic diffusion in a complex network and to detect super spreader nodes.

In the model the nodes represent individuals that, for each time step can become infected by contact with their network neighbor nodes.

A variation of the SIR model (Susceptible Infected Recover) infection diffusion is adopted, where at each time steps the epidemic diffusion is calculated with respect to system configuration parameters.

All nodes are initially in *Susceptible* state, and can become *Infected*. If a node become Infected it moves to state *Recovered* after  $t_{REC}$  time steps and cannot be infected. A *Recovered* node become again *Susceptible* after  $t_{SUS}$  time steps.

At each time step, for each infected node, there is a probability  $p_{trans}$  of transmitting the disease to each one of its non infected neighbors. The nodes transitions are synchronous, i.e. they all take place at the end of the time step.

- initial number and/or distribution of random infects
- $p_{trans}$  probability of transmission to a neighbors
- $t_{REC}$  time steps needed for an individual to heal and recover
- $t_{SUS}$  time steps after recovering needed for an individual to become susceptible again (consider a special maximum value of *infinite*)

The system should allow:

- to load any graph in csv format
- to configure the parameters
- to start and run one or more simulations **with same initial node states**
- to display statistics and metrics
- to display a graphical representation of the graph before, during or at the end of the simulation by coloring it node basing on their states

**The candidate should develop a technique to find and show the most relevant super spreader nodes in a specific graph through a series of random simulation.**

### Project: Opinion Diffusion in Networks

The purpose is to implement a systems to analyze opinion diffusion in a complex network.

In the model the nodes represent individuals that, are characterized by an *opinion state*, among a fixed number of choices. The opinions can be exchanged/transmitted to neighbor nodes.

At each time step the opinion diffusion is calculated with respect to system configuration parameters.

All nodes are initially in given *opinion* state, and can change. At each time step, for each node with opinion A, there is a probability  $p_{unchangeA}$  of not changing opinion, and a probability  $1 - p_{unchangeA}$  of adopting the opinion of one of its neighbors. If the individual is going to change opinion it will take with uniform probability the opinion of one of its neighbors (it holds a random tournament among opinions depending on the number of neighbors and their opinion).

Once an individual has adopted an opinion, it stays with that opinion without changing it for at least  $t_{STAY}$  time steps.

The nodes transitions are synchronous, i.e. they all take place at the end of the time step.

The systems should allow to configure:

- the set of possible *opinions*
- $p_{unchangeA}$  probability/resistence to change for each single opinion A
- $t_{STAY}$  time steps that an individual persist in an newly adopted opinion

The system should allow:

-to load any graph in csv format

-to configure the parameters, in particular the set of *opinions* and for each opinion the conservation parameters resistance to change

-to start and run one or more simulations **with same initial node states**

-to display statistics and metrics

-to display a graphical representation of the graph before, during or at the end of the simulation by coloring the nodes basing on their states, and to provide *opinion distribution* over a number of simulation on the same parameters and initial states.

**The candidate should develop a technique to find the most important nodes to block or help opinion diffusion.**

### Project: Social Graph Formation Model

We want to build and analyse a social graph formation model which reflects the following rules: when a node has a small number of neighbors (or none) it creates new links by *preferential attachment*, while when a node is more connected, it creates links by a *link prediction method*.

Given a set of  $N$  nodes and 4 initially connected among them nodes, and  $K$  iterations, at each iteration the  $N$  nodes are extracted with uniform probability and to each extracted node  $x$  a link is added:

- by link prediction, with probability  $p = \frac{|\Gamma(x)|}{|\Gamma(x)|+c}$  : to choose the node to connect to  $x$  the link prediction algorithm **Adamic-Adar** is applied to all non connected nodes at distance 2 from  $x$ . The nodes are ranked by Adamic-Adar and the maximum ranked link is added. If not such a node exists the link is chosen by preferential attachment.

or

- by *preferential attachment* with probability  $1-p$  to choose the node to link to  $x$

The system will allow: to configure the parameters and experiment different values for  $c$ , with  $0 < c \leq 1$  e  $c > 1$  and different values of  $N$ , and  $K$ , to save the created graph/s in csv format and compare and plot different network metrics them along different simulations such as: *degree distribution for increasing number of iterations*, *max connected component*, *network diameter*, *average path*, *connectivity coefficient*.

## AREA: Agents and Reinforcement Learning

### Agent in Labyrinth Walls

Design and implementation of a reinforcement learning environment, for training an agent using a **Q-Learning algorithm** in the framework of AI-Gym.

**Goal:** *the purpose of the project is to show that the agent, through reinforcement learning, can learn to move in the labyrinth without bumping on the walls.*

**Environment:** a  $m \times n$  grid of cells where each cell is either empty (white) or a wall (coloured), where the agent is located in a upper left position (es. cell 2,2 in figure) and the exit is lower right position and certain percentage of random walls cells represent the labyrinth

**Agent:** the agent available action are the four movement actions: *Up, Down, Left, Right*

**State:** the percept state returned from the environment is a representation of the Moore 8-neighborhood centered in agent position

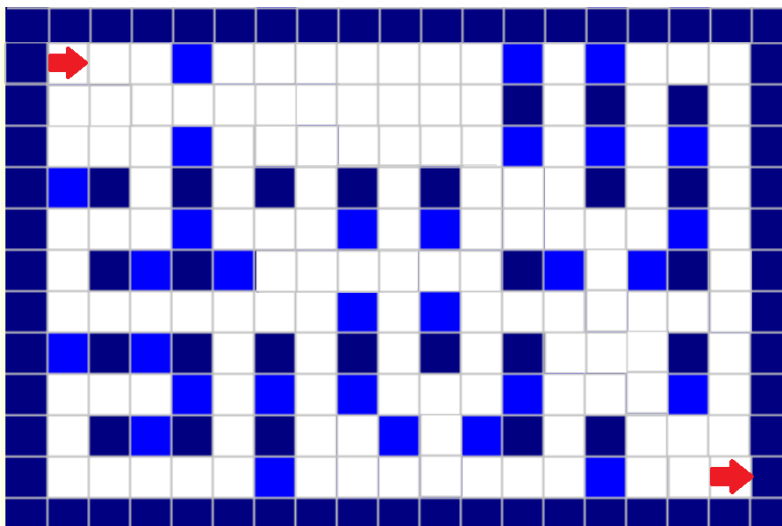
?	?	?
?	A	?
?	?	?

**Actions effect:** each actions has the effect of moving the corresponding position of the agent in the environment and returning the appropriate state and reward.

Moving toward a wall causes the agent bumping in the wall without changing position

**Reward:** each movement action has a reward of -1, bumping toward a wall has a reward of -5 reaching the final exit position has a reward of 10

**Stop condition:** the agent reach the exit cell or **maxK** actions are executed



The implementation of the environment should allow:  
to set the parameters:

- $m, n$  dimensions of the grid
- *percentage* of walls
- *maxK* maximum number of actions before end

The system should allow:

- Run and train the Qlearning reinforcement learning algorithm,
- Generating and trying different labirinths during training, showing the evolution of the accumulated reward
- Saving the Q(State, Action) matrix, Loading a saved Q matrix
- Executing the agent step-by-step on a given labirynth showing the reward

Note: the given rewards may be adjusted in the initial phases of testing the project to check convergence

## Agent Cleaning Leaves


Design and implementation of a reinforcement learning environment, for training an agent using a **Q-Learning algorithm** in the framework of AI-Gym.

**Goal:** *the purpose of the project is to show that the agent, through reinforcement learning, can learn to move in the map while cleaning leafs from the ground.*

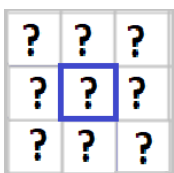
**Environment:** a  $m \times n$  grid of cells, where the agent is located in a upper left position (es. cell 2,2 in figure) and the exit is lower righth position and certain percentage of random *leafs* are distributed in the cells.

A random “wind” acts on the environment independently from the agent ,randomly moving the leaves **every two actions** steps. In case of collisions a leaf does not move.

**Agent:** the agent available action are the four movement actions: *Up, Down, Left, Right*, plus the

action *Suck* where if a leaf  is in the same cell position of the agent, it disappears from the ground and is stored in the agent

**State:** the percept state returned from the environment to the agent is of the Moore 9-neighborhood centered in agent position, where each cell is either empty or contains a leaf.

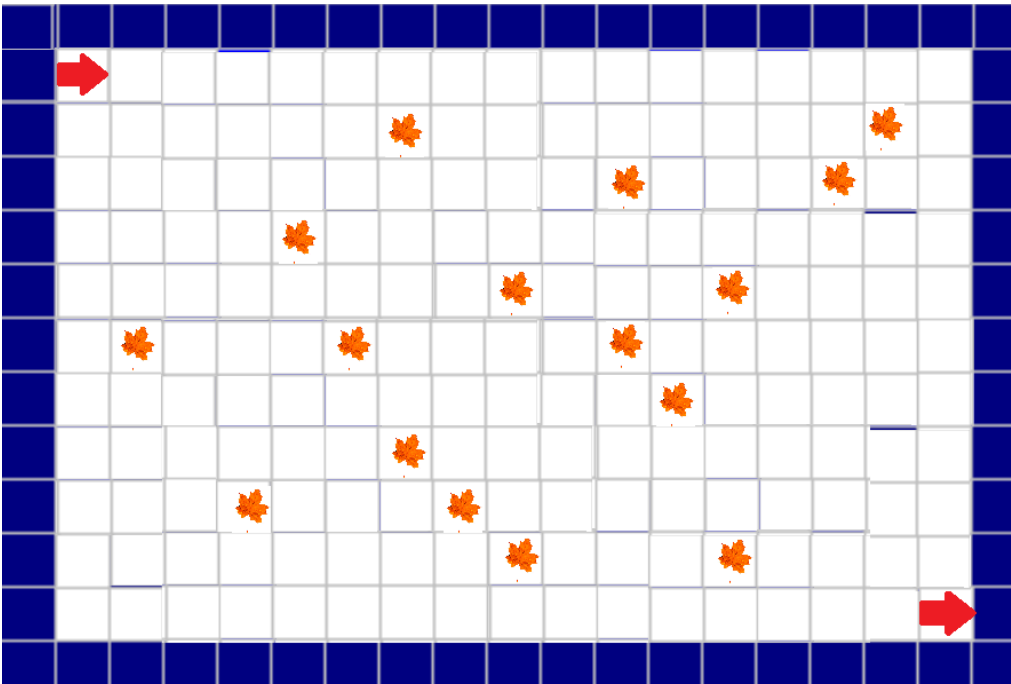


**Actions effect:** each action has the effect of moving the corresponding position of the agent in the environment and returning the appropriate state and reward.

Sucking a leaf, will have the effect of making the leaf disappear without changing the position of the agent, suck where a leaf is not present will not change the agent position.

**Reward:** each movement action has a reward of -1, sucking a leaf will have reward of +5, executing a sucking action where a leaf is not present will have a reward of -1, when the final position is reached a further reward of  $5 \times \text{number of collected leaves}$  is assigned to the agent.

**Stop condition:** the agent reach the exit cell, or **maxK** actions are executed



The implementation of the environment should allow:  
to set the parameters:

- $m, n$  dimensions of the grid
- *percentage* of leaves
- *maxK* maximum number of actions before end

The system should allow:

- Run and train the Qlearning reinforcement learning algorithm,
- Generating and trying different Cats distribution during training, showing the evolution of the accumulated reward
- Saving the Q(State, Action) matrix, Loading a saved Q matrix
- Executing the agent step-by-step on a given map showing the reward

Note: the given rewards may be adjusted in the initial phases of testing the project to check convergence

## Mouse Agent Patrolling Cats

Design and implementation of a reinforcement learning environment, for training an agent using a **DQN-Learning algorithm** in the framework of AI-Gym.

**Goal:** *the purpose of the project is to show that a Mouse agent, through reinforcement learning, can learn to move in the map for cheese, or wait, while avoiding Cats.*

In order to implement this project some further technical details may be needed in order to use the environment and to interface the DQN algorithm, students are suggested to contact prof. Milani (alfredo.milani@unipg.it) or Dott.Biondi (giulio.biondi@unipg.it) or Dott.Polticchia (mattia.polticchia@studenti.unipg.it).

**Environment:** a  $m \times n$  grid of cells, where the *Mouse* agent is located in a upper left position and the *Cheese* in a random cell. A certain number of *Cats* are located in random position and they patrol rows or column by moving up/down or left/right through the rows/column they are assigned. **Eat condition:** if a *Cat* is in the same cell where the *Mouse* is, in the next instant of time, *Cat* does not move, the mouse is eaten and the episode ends.

**Agent:** the agent available action are the four movement actions: *Up*, *Down*, *Left*, *Right*, plus the action *Stop* where the agent remain in the same cell for one step.

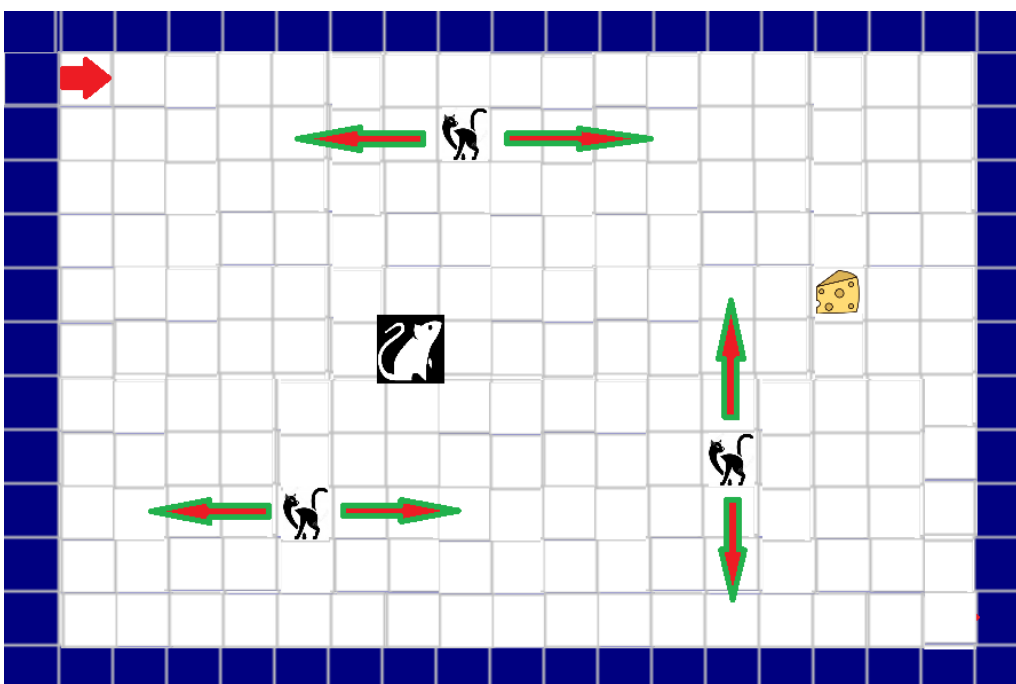
**State:** the percept state returned from the environment to the agent is *any representation* (image or character matrix) of the environment that the candidate consider suitable.

**Actions effect:** each action has the effect of moving the corresponding position of the agent in the environment and returning the appropriate state and reward.

Some actions will take place on the environment independently of the agent: *Cats* patrolling movements, eating *Mouse* condition. In case of conflict the effect of eating *Mouse* condition will prevail, in case of conflict on the *Cats* position break ties randomly.

**Reward:** each movement action has a reward of -1, *Mouse* reaching the *Cheese* has a reward +100, eating *Mouse* condition has a reward of -1000

**Stop condition:** the agent reach the *Cheese* cell, or **maxK** actions are executed or the *Mouse* is eaten



The implementation of the environment should allow to set the parameters:

- $m, n$  dimensions of the grid
- *number* of *Cats* with randomly assigned cell and patrolling direction (*row/column*)
- *maxK* maximum number of actions before end

The system should allow:

- Run and train the DQN reinforcement learning algorithm,
- Generating and trying different Cats distribution during training, showing the evolution of the accumulated reward
- Saving/Loading the learned DQN model
- Executing/Showing the agent with a learned DQN model on an initial state

Note: the given rewards may be adjusted in the initial phases of testing the project to check convergence



## Mouse Armed Agent and Hunting Cats

Design and implementation of a reinforcement learning environment, for training an agent using a **DQN-Learning algorithm** in the framework of AI-Gym.

**Goal:** *the purpose of the project is to show that a Mouse agent, through reinforcement learning, can learn to move in the map for cheese while avoiding or trying to combat Cats .*

In order to implement this project some further technical details may be needed in order to use the environment and to interface the DQN algorithm, students are suggested to contact prof. Milani (alfredo.milani@unipg.it) or Dott.Biondi (giulio.biondi@unipg.it) or Dott.Polticchia (mattia.polticchia@studenti.unipg.it).

**Environment:** a  $m \times n$  grid of cells, where the *Mouse* agent is located in a upper left position and the *Cheese* in a random cell. A certain number of *Cats* are located in random position and they have long sight and move toward the *Mouse* at each time step (alternating vertical and horizontal approaching direction). If a *Cat* move in the cell where is the *Mouse*, the mouse is eaten and the episode ends.

**Agent:** the agent available action are the four movement actions: *Up, Down, Left, Right*, plus the action *Shot* where the agent explodes arrows in of the four directions and two diagonals, possibly killing all the cats in the lines.

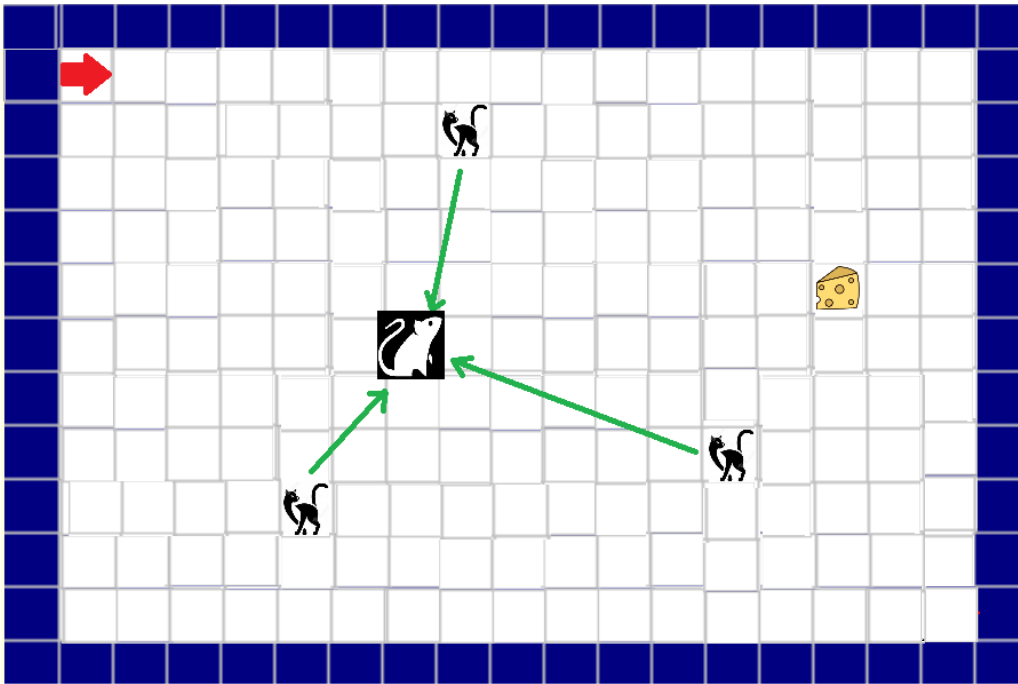
**State:** the percept state returned from the environment to the agent is *any representation* ( image or characters matrix) of the environment that the candidate consider suitable.

**Actions effect:** each action has the effect of moving the corresponding position of the agent in the environment and returning the appropriate state and reward.

Some actions will take place on the environment independently of the agent: *Cats* movements, eating *Mouse* condition. In case of conflict the effect of eating *Mouse* condition will prevail, in case of conflict on the *Cats* position break ties randomly.

**Reward:** each movement action has a reward of -1, *Mouse* reaching the *Cheese* has a reward +100, eating *Mouse* condition has a reward of -1000, action *Shot* has a reward of -10 if no *Cat* is killed, a reward of 10 for each killed cat.

**Stop condition:** the agent reach the *Chees* cell, or **maxK** actions are executed or the *Mouse* is eaten



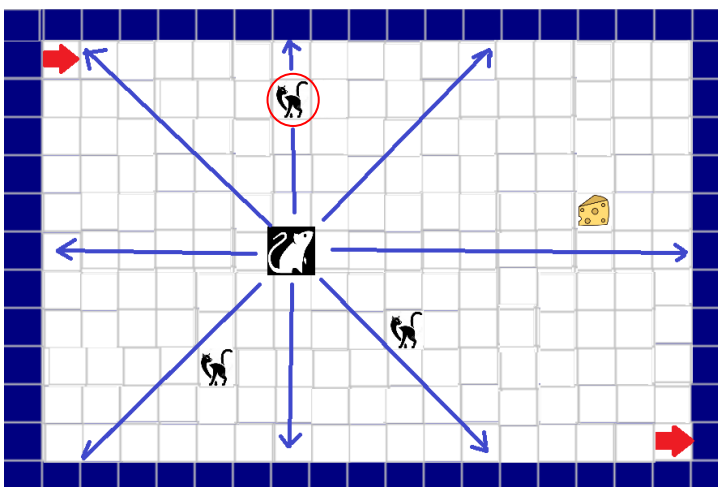
*Cats approaching Mouse*

The implementation of the environment should allow:  
to set the parameters:

- $m, n$  dimensions of the grid
- *number* of Cats with randomly assigned cells
- *maxK* maximum number of actions before end

The system should allow:

- Run and train the DQN reinforcement learning algorithm,
- Generating and trying different Cats distribution during training, showing the evolution of the accumulated reward
- Saving/Loading the learned DQN model
- Executing/Showing the agent with a learned DQN model on an initial state



**Example effect of action *Shot*.**

Note: the given rewards may be adjusted in the initial phases of testing the project to check convergence