Artificial Intelligence - Assessed Exercise (deadline: March 16, 5pm UK time)
(Workload: ~ 20 hours, it should not take you much more than that !!!)

1 Problem Statement (Python Programming + Short Report)

Your task is to design, implement, evaluate and document three virtual agents which are (potentially) able to reach a goal in a custom Open AI Gym environment derived from Frozen. Thus, you will need to install and understand the workings of the Open AI Gym environment to be able solve the task (hint: see AI (H) Lab 2).

The specific environment/problem under consideration is a grid-world with a starting position (S), obstacles (H) and a final goal (G). The task is to get from S to G. The environment is defined in uofgsocsai.py via the class LochLomondEnv including documentation relating to the setting, specific states, parameters etc. An example of how to instantiate the environment and navigate it using random actions is provided in lochlomond_demo.py .

You must consider three agent types: a senseless/random agent, a simple agent and a reinforcement agent based on the requirements listed in the following sections. Your agents will be tested against other instances of the same problem type, i.e., you can not (successfully) hard-code the solution. You will have access to eighth specific training instances of the environment determined by the value of a single variable problem_id.

Your agents and findings should be documented in a short (max 1000 word) technical report accompanied by the actual implementation/code and evaluation scripts.

2 Tasks

You should provide a design, implementation, evaluation and documentation of three different agents; each with its own set of specific requirements:

2.1 Task I: Senseless/Random agent [5% of marks]

You should provide a solution for an agent without sensory input which takes random actions. You should initialise the environment as follows:

```
env = LochLomondEnv(problem_id=problem_id, is_stochastic=True, reward_hole=0.0)
```

where you can use problem_id∈[0:7] to evaluate the performance over different instances of the same problem.

Purpose: This agent should be used as a naive baseline. Hint: A basic senseless/random agent is already partly provided in lochlomond_demo.py (albeit with output computation of the performance measure...).

Requirements:
-<u>Sensors</u>: None (/random/full; it doesn't matter...)
-<u>Action</u>: Discrete
-<u>State-space</u>: No prior knowledge (i.e. it has not got a map)
-<u>Rewards/goal</u>: No prior knowledge (does not know where the goal is located)

2.2 Task II: Simple Agent -- A* Search [10% of marks]

You should provide an agent based on a tree/graph-search and justify its use for solving the particular the problem (**using A* search with a suitable heuristic**) assuming that the task environment is fully known and observable. You should initialise the environment as follows:

```
env = LochLomondEnv(problem_id=[0-7], is_stochastic=False, reward_hole=0.0)
```

where you can use problem_id $\in$[0:7] to evaluate the performance over different instances of the same problem - and to fine-tune your agent to make sure it generalises. **We recommend you use existing code (from e.g. the AIMA toolbox) to solve this part)**.

Purpose: This agent is used as an ideal baseline to find the optimal path under ideal circumstances. Hint: if you have attended the Lab sessions you will easily be able to reuse most of the code to solve this part (a parser that maps from env.desc to the lab 3 format will be made available).

Requirements:
-<u>Sensors</u>: Oracle (i.e. you're allowed to read the location of all object in the environment e.g. using env.desc)
-<u>Actions</u>: Discrete and noise-free.
-<u>State-space</u>: Fully observable a priori .
-<u>Rewards/goal</u>: Fully known a priori (you are allowed to inform the problem with the rewards and location of terminal states)

2.3 Key Task III: Reinforcement learning agent -- Q-Learning [85% of marks]

You should provide a reinforcement learning agent to solve the problem with minimal assumptions about the environment (see below). The choice of the RL agent is up to you but **we highly recommend using tabular Q-learning**. Regardless, the choice should be well-justified and meet the list of requirements listed below. You should instantiate the environment as follows:


```
env = LochLomondEnv(problem_id=[0-7], is_stochastic=True, reward_hole=[YOUR-CHOICE]) ,
```


where you can use problem_id $\in$[0:7] to evaluate the performance over different instances of the problem - and to fine tune your agent to make sure it generalises. You are encouraged to write your own code but use of 3rd party implementations (e.g. the AIMA toolbox) is allowed if you demonstrate a sufficient understanding of the algorithms in the accompanying documentation and cite appropriately.

Requirements:
-<u>Sensors</u>: Perfect information about the current state and thus available actions in that state; no prior knowledge about the state-space in general
-<u>Action</u>: Discrete and noisy. The requested action is only carried out correctly with a certain probability (as defined by uofgsocsai.py).
-<u>State-space</u>: No prior knowledge, but partially observable/learn-able via the sensors/actions.
-<u>Rewards</u>: No prior knowledge, but partially observable via sensors/actions.

Notice: You can restart and replay the same instance of the problem multiple times and maintain the knowledge obtained across repetitions/episodes. <u>The reward should in this case be reported as;</u>
a) the average across all restarts/repetitions and,
b) the single best policy (typically after a long learning phase).

3 Submission

You should include the following three items in your submission:

3.1 Implementation & Code

Your implementation - containing the three different agents (along with any dependencies, except the Open AI Gym) - should be uploaded to Moodle as a zip-file containing the source code. You must provide three separate and executable python scripts/programs named: run_random.py, run_simple.py and run_rl.py which takes as (only required) argument the problem_id . Each script/program should include training/learning phases (including possible repetitions/episodes of the problem) and output a text file (named after the agent, e.g. ”random”) with any relevant information. Hint: A template will be provided via Moodle.

3.2 Experiment & Evaluation

An important aspect of RL is assessing and comparing the performance of agents and different policies. To document the behavior of your agents you should design a suitable set of (computer) experiments which produces a relevant set of graphs/tables to document the behavior of your RL agent (e.g. average performance measure vs number of episodes, etc) and compares its performance against the baselines. The evaluation strategy should be implemented in a single Python script (a wrapper) run_eval.py which runs you entire evaluation, that is, it should call your agents, collate the results and produce the figures/tables you have included in your report. The run_eval.py should be submitted via Moodle alongside your implementation.

3.3 Short Report (1000 words)

You should document your work results in a short technical report of (aim for 1000 words; excluding figures, tables, captions, references and appendices). **Appendices may be used to provide extra information** to support the data and arguments in the main document, e.g., detailed simulation results but should not provide crucial information required to understand the principle of the solution and the outcome. <u>You can include as many references as you see fit.</u>

Requiremnts:

- Focus is at efficient, yet accurate and precise description of the implemented agents and learning algorithms in the given context of the task. We mark your understanding of these agents / learning algorithms and want to express the importance of their evaluation. You should The report should be submitted via Moodle as a pdf file alongside your implementation and evaluation scripts.

4 Marking Scheme

The assessment is based on the degree to which your submission (implementation, evaluation script and report) concisely, correctly and completely addresses the following aspects:

Analysis [15%]
Introduction/motivation and correct PEAS analysis (including task environment characterisation).

Method/design [15%]
Presentation of relevant theory and methods (for all your agents) with proper use of citations and justification for choosing specific methods (referencing your analysis).

Implementation [25%]
The code for all the agents (not in the report!) should be well-documented, follow best-practices in software development and follow the outlined naming convention. The report must contain a presentation of relevant aspects of the implementation.

Experiments / Evaluation
[20%] Evaluation script/program to reproduce the results (i.e. graphs/tables) adhering to requirements
[20%] Relevant presentation of the evaluation strategy, metrics and the obtained simulation results. A suitable presentation and comparison of the performance of the agent with other agents as evaluated across a suitable number of problem variations (e.g. using graphs/tables).

Discussion and conclusion [5%]
- Including a critical reflections on the outcome/results

The weighting of the senseless, simple and RL agent - for all marking criterion - is 5,10 and 85 %, respectively. So allocate your own time accordingly and remember that this exercise counts 25% of your overal grade in AI, so it is very important.

Evaluation of Agents – Tipps you should consider

Issue: A large choice of different models with many algorithmic choices to be made (even for a simple problem)...

<u>For example</u>:

- Which method / model ?
- Parameters changes the results, which setting is better ?
- Initialisation changes the result (e.g. randomness in the environment), which is better ?
- Number of iterations changes the results, converged ?

For standard search based methods, theoretical computing science has been able to prove complexity, optimality for many (e.g. A-star) under well-defined conditions.

In dealing with **real-world AI problems** we are often often left with asymptotic bounds depending on some property of the problem. This effectively means that we often need to do a large amount of empirical evaluation (we can still do theoretical stuff asymptotically):

<u>Aspects</u>

- **Completeness**
  - Will the procedure find a solution if there is one ?
- **Optimality**
  - Does it find the optimal solution, i.e. policy ?
- **Complexity**
  - Space: How much memory does the procedure require ?
  - Time: How quickly does it solve it?
    - I) Learning
    - II) Policy estimation / execution

- **Robustness**
  - Is the method/parameter choice robust?
    - Robust for the same problem (e.g. repeat the experiment)
    - Does it generalize to new problem of the same type
      - e.g. can we apply the same procedure for learning and policy estimation to variations of teh same problem (e.g. different start and goal states)?
    - Does it generalize to new problem of the same type
      - e.g. can we apply the same procedure for learning and policy estimation to variations ?

Evaluation setting

Aspects to consider when evaluating reinforcement agents

- <u>Problem formulation/modelling</u>:
  - e.g. is MDP suitable
  - e.g. is the discount factor appropriate?
  - e.g. are the exploration parameters appropriate?
- <u>Validate simulations settings</u>
  - e.g. number of iterations leading to convergence
- <u>Evaluate performance across relevant random aspects (e.g. random initialisations, random effects in the environment)</u>
  - e.g. consider average performance and variance across repetitions
- <u>Generalization across problem variations...</u>
  - e.g. have your agent learned to solve all mazes in the world
- Generalization across problem types/domains/tasks...
  - e.g. have you agent learned to solve fundamentally different tasks
- Generalization: idealy, we'd need to select our model, algorithm and parameters on a held-out problemtask/variant in order to estimate the agents ability to generalize

And we should always evaluate based on a set of well-defined metrics / performance measures...

Performance Metrics

Observed reward:

- per episode/trial (e.g mean and variance)
- per action (e.g mean and variance)
- ...

Utility loss (Q(s,a) loss):

$\mathbf{U}$ is a vector of the utilities $\mathbf{U} = [U(s0), U(s1), \ldots U(sn)]^{\top}$

$$E = \left\|\hat{\mathbf{U}} - \mathbf{U}\right\|_2^2 = \sum_{\forall s} \left(\hat{U}(s) - U(s)\right)^2$$

$$E_{RootMeanSqure} = \sqrt{\frac{1}{N_S} \sum_{\forall s} \left(\hat{U}(s) - U(s)\right)^2}$$

$$E_{\infty} = \left\|\hat{\mathbf{U}} - \mathbf{U}\right\|_{\infty} = \max_{s} \left(\hat{\mathbf{U}} - \mathbf{U}\right)$$

Policy loss: The most the agent can loose by executing a suboptimal policy compared to the optimal one

$$\|U^{\pi_i} - U\|_{\infty} < \epsilon \quad then \ \|U^{\pi_i} - U\|_{\infty} < 2\,\epsilon\gamma/(1-\gamma)$$

... but what is ϵ ??

Goal loss:

- Number of times an agent reaches the goal ....