In this project we will find the optimal path to pass through a maze under different conditions and using different methods. The methods are:

- Reinforcement Learning (RL)
- Information Reinforcement Learning (IRL)
- Using a simulation and evolution of the agents.

We would like to compare the solution each method achieves under various conditions.

**General remarks:**

- Efficiency, readability and good organization of your code will be a part of the grade. (Minimize the use of for-loops when possible, give meaningful names to variables, write small functions for small sections of the algorithm etc.).

- Try to write functions such that they can be used in more than one part of the project.


- The instructions are probably approximately self-contained but everything is based on material you saw in the lectures or on references given below.

- For my convenience, when you show the maze or any other function in the maze coordinates, always show it so that (1,1) is in the bottom left corner.

**Part A - Reinforcement Learning**

In this part we will find the solution for a maze problem using the *value iteration* algorithm. Given a Markovian transition model $P_{as}$, the expected value of the next reward $R_{ss'}$, and a set of actions $A$, this algorithm finds iteratively an optimal solution for the value function, by using the Bellman optimality equation as an update rule. This solution is optimal in the sense that the value of each state is the value obtained by the optimal policy(/ies), which is the policy whose expected return is greater than or equal to that of all other policies for all states. For more information read sections 3.8 and 4.4 in Sutton & Barto [http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html](http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html).

The value iteration algorithm (as it is given in S&B's book):
Initialize $V$ arbitrarily, e.g. $V(s) = 0$ for all $s \in S$

Repeat
\[ \Delta \leftarrow 0 \]
for each $s \in S$:
\[ v \leftarrow V(s) \]
\[ V(s) \leftarrow \max_{a \in A} \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')] \]
\[ \Delta \leftarrow \max (\Delta, |v - V(s)|) \]
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
\[ \pi(s) = \arg \max_{a \in A} \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')] \]

Note that $\pi_s(a)$ or $\pi(s,a)$ is a common notation for $Pr(a|s)$ (i.e. the policy). In the algorithm above we use a deterministic policy and $\pi$ denotes the action itself.

**The problem setup - applies to all parts**

- Initialize a maze using the lines below. This 10x12 matrix contains 0’s and 1’s. The agents are allowed to step only on 0 squares.
  \[
  \text{maze} = \text{zeros}(10,12); \\
  \text{maze}(1:7,4:9) = 1; \\
  \text{maze}(4,:) = 0; \\
  \]
- The allowed actions are up, right, down, left.
- When trying to move to a 1 square an agent gets a reward of -100 and stays in its place.
- When stepping to a 0 square the agent gets a reward of -1 (what is it good for?).
- When attempting to move outside the maze boundaries an agent gets a reward of -1000 and stays in its place.
- When reaching the goal state the agent gets a reward of 0.
- The goal state is (6,11).
- Initialize the reward matrix according to the requirements above. Note that $s$ is the current state, $s'$ is the next state, and $a$ is the action which takes (deterministically) from state $s$ to $s'$. Therefore $R^a_{ss'}$ can be simplified in our case to $R^a_s$.
- Initialize $P^a_{ss'}$ according to deterministic transitions. Define the goal state as an absorbing state ($P^a_{s\text{goal}*\text{goal}} = 1 \forall a$).

**The algorithm implementation and results**

- Implement the value iteration algorithm and find the optimal value function for the problem above.
- Show the value function after several different iteration counts.
- Run an agent according to the learned policy and show its path starting from (1,1).
Part B - Information Reinforcement Learning

In this part we will use the same problem setup as in part A, but we will use the INFO-RL algorithm by Tishby et al. to find the optimal solution given by

$$
\min_{\pi} F_{\pi}(s_0; \beta) = \min_{\pi} \left[ I_{\pi}(s_0) - \beta V_{\pi}(s_0) \right] = \min_{\pi} \left\{ \lim_{T \to \infty} E \left[ \sum_{t=0}^{T-1} \left( \log \frac{\pi_{s_t}(a_t)}{\rho_{s_t}(a_t)} - \beta R(s_t, a_t) \right) \right] \right\}
$$

where

- the parameter $\beta$ is a positive scalar which controls the tradeoff between information and value
- $\rho_s(a) = Pr(a|s)$ is some default policy (e.g. the random policy)
- $R(s, a)$ is the reward matrix (the same as $R^a_s$ in part A)
- The expectation is taken with respect to the probability of all future trajectories starting in the state $s_0$ and executing the policy $\pi$.

The INFO-RL algorithm is stated as follows: (for more details about the algorithm see http://eprints.pascal-network.org/archive/00009320/01/IRL.pdf)

```
Initialize $F(s) \leftarrow 0, \forall s \in S$

repeat
  for each $s \in S$
    $Z(s; \beta) \leftarrow \sum_a \rho_s(a) e^{\beta R(s, a) - E_{s'|s,a}[F(s' ; \beta)]}$
    $F(s; \beta) \leftarrow -\log Z(s; \beta)$
  end for
until $F$ has converged ($F^* \leftarrow F$)

for each $a \in A, s \in S$
  $\pi^* (a|s) \leftarrow \frac{\rho_s(a)}{Z(s; \beta)} \exp \{ \beta R(s, a) - E_{s'|s,a}[F^* (s', \beta)] \}$
return $\pi^*$
```

* Expected values are taken with respect to $P^a_{ss'}$, which is the same transition probability matrix as in part A.

The algorithm implementation and results

- Implement the INFO-RL algorithm above.
- Visualize the optimal free energy for different values of $\beta$ between 0.1 to 5. Distribute the $\beta$ values so that you won’t have “big” jumps in the next plots.
- Plot the expected value $V_\pi$ vs. the control information $I_\pi$ (each point is a different beta value).
- Visualize the optimal policy for different betas.
- Run an agent according to policies obtained in different betas and show its path starting from (1,1).
Part C - Simulating evolution of agents

In this part we will attempt to find the optimal path using a gradual evolution of agents, where in each generation only those who accumulated value above the defined threshold survive and contribute to the next generation. Specific values of the parameters are given, but you are more than invited to examine different values.

Algorithm requirements and parameters:

- Use the same problem setup as in part A.
- Initialize the threshold value to some very low number (e.g. $-10^8$).
- In each generation increase the required value in 5%.
- Set to 200 the maximal number of steps each agent is allowed to make in a single run. An agent finishes its run when it reaches the goal state or the maximal number of steps.
- Set the number of agents in each run to 100.
- Set the maximal number of generations to 100. Note that at some point there might not be agents that pass the value threshold. Consider how to handle this.
- After all the agents in each generation finished their run, generate a new stochastic policy by giving an equal weight to the paths of the agents that passed the current value threshold.

Results

- Visualize the policy learned in different generations.
- Show a sample path done by an agent in different generations.
- Plot the average accumulated value as a function of the generation number.
- Plot the average control information as a function of the generation number. Write explicitly how you calculated it (see the definition of the free energy in part B, take $\rho$ to be the random policy).
- Plot the average accumulated value as a function of the empirical control information.

Note: Don’t expect 100% perfect results, but try to explain your results.

Part D - Discussion and conclusions

Discuss your results and compare the solutions obtained by the different algorithms. Further analysis (e.g. different parameters, other mazes*, different learning/strategies in the simulation part), as well as examination of literature-based or well-explained changes to the algorithms would be welcome and will contribute to your grade. Issues for discussion may include types of optimality, do the methods give different/same solutions and when, robustness, behavior under different conditions, relation to statistical mechanics if you wish, or any other conclusions or findings which seem interesting to you. You can also try a more thorough analysis on a simplified problem set - one state with 3 actions (reward is on the actions), where you can also visualize the advance in the probability space.

* For example you can try this maze:
maze = zeros(10);
m1 = [ 0 1 0 0 1 1; 1 0 0 1 0 0; 0 1 0 0 1 1; 0 1 1 0 1 0; 0 0 1 0 0 0; 1 1 1 0 0 0];
maze(4:9,2:7) = m1;

with the goal state at (10,10).