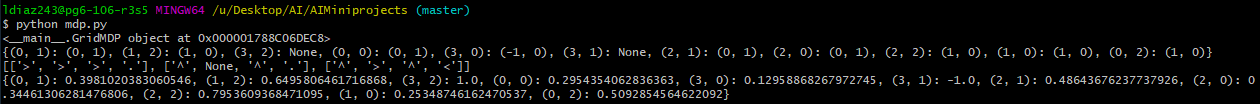
**Mini-Project 2: MDP**

For this mini-project, you are going to implement a simple simulation of robot path planning, called a grid world, and use value iteration to come up with policies to get the robot to navigate a maze. You are given two files ([mdp.pyPreview the document](https://fiu.instructure.com/courses/45004/files/7680903/download?wrap=1) and [utils2.pyPreview the document](https://fiu.instructure.com/courses/45004/files/7680904/download?wrap=1)) with an initial implementation that you need to understand and extend.

1. Questions about the code (30 points)

a) Run the code by executing python3 mdp.py. I ran this on Ubuntu 16.04 using Python 3.5. Show a snapshot of the code running.



b) Explain the relationship between the class MDP and the class GridMDP in mdp.py

c) Explain the following python functions from the sample code:

-vector\_add

-turn\_right

-turn\_left

-value\_iteration

-best\_policy

-expected\_utility

d) What happens with the found policy if the reward for the entries that have -0.04 are changed to -1, -0.3, or -0.02? Explain this.

2)  Extending the code (65 points)

Now, you will extend the sample code to accept other mazes of up to 15x15 squares.

Your program will find a policy for the robot such as the one presented in Figure 17.2. The maze layout will be provided as a simple text file, in which '%' means obstacles, 'P' is a location with a positive reward of +1 position, and 'N' means a negative reward -1. There will be one ‘N’ and one ‘P’ in the file. See the sample [mazePreview the document](https://fiu.instructure.com/courses/45004/files/7680902/download?wrap=1) file for an illustration. All the other locations besides obstacles, negative reward location, and positive reward location will have reward -0.04.

In each grid square, you can take one of four actions, North, South, East, or West, which moves you with 0.75 probability to the intended square and with probability 0.125 to both right angles of the intended location. See figure 17.1 in the book for an illustration.

Using the**value iteration** (Section 17.2) algorithm (you can modify as you need the implementation provided in mdp.py and util2.py), find an optimal policy for the agent.

Your program should run using Python 3.7. Your code can only import extra modules, but only if they are part of the standard python library. You can also use the provided code, please attach all the files needed to run your program.

Your program should run as follows:

python3 mdp.py maze.txt

For each maze, your program should include the obtained policy (this can be shown in the command line).

You will submit this mini-project through canvas. You will upload these files:

-the python files with your solution

-report.pdf - a project report on your implementation

You should include in your report the name of the team members (you can work in pairs).

Please describe the algorithms and data structures that you used for the implementation of value iteration. Show the run of 3 different mazes created by you. Answer the following questions on your report: How did you implement the transition function?  How did you implement the reward function? How did you implemented the Bellman update equation (17.6) in the book. How close is your implementation to the pseudo-code in figure 17.4?

[**PREVIOUS**](https://fiu.instructure.com/courses/45004/modules/items/2054022)