

# Assignment 2

Daniel Fišer, Jan Mrkos

Department of Computer Science,  
Faculty of Electrical Engineering,  
*mrkosja1@fel.cvut.cz*

April 10, 2022

## Content:

- Assignment 2 domain
- FF replan solution example
- Requirements
- Grading

## Assignment 2: Help Pepa get home

Pepa spent a nice evening out in the city and now he's trying to get home. He's waking up early, so he wants to get home ASAP. But he's not his best self; his coordination is not that good and might try to "gather his strength" if he finds a bar that's still open. Something that he will regret in the morning.

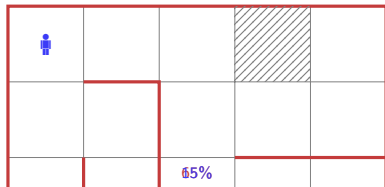
It's your task to help Pepa get home.



## Assignment 2 domain - movement

More formally, consider the following task:


- We have a rectangular city (maze) with walls and an agent, AKA Pepa.
- The agent can move up, down, left, and right.
- The agent can move if the movement is not blocked by a wall.
- But if the movement is blocked by a wall, the agent stays put.
- There is, however, a catch: Whenever the agent decides to go in one direction, he succeeds with probability of 70%, he will go sideways with probability of 15%.
- So for example, if the agent decides to go up, he ends up here with 70% probability, here with 15% probability, here with 15% probability.



## Assignment 2 domain - rewards

Another important part of a probabilistic planning problem is the reward structure:

- For reaching its goal, the agent receives reward of 200
- When agent moves onto a delay (open pub) square (strafed) it receives reward  $-50$
- Each transition the agent makes it receives reward  $-1$

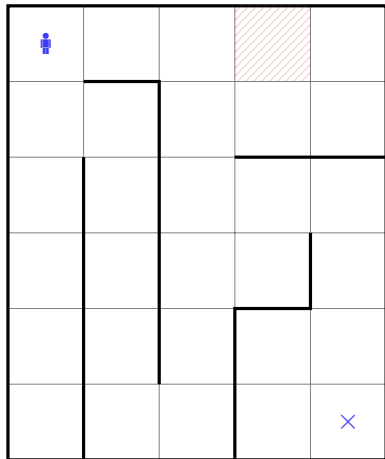
	-1	-1	-50	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	200

## Assignment 2 domain - objective

The goal of the agent is to maximize his reward at the end of the simulation.

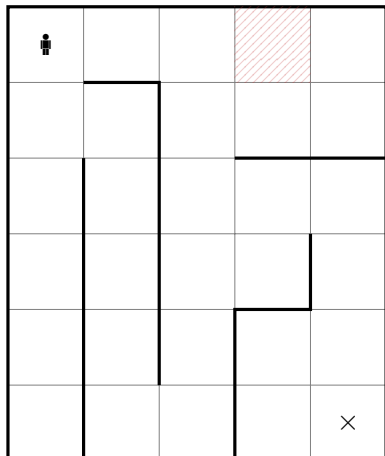
Regarding the rewards, also consider this:

- Moving into the wall is still a transition (even though the starting state is the same as the ending state).
- The rewards are associated with transitions between states, not with occupying certain squares.



# FF-replan example

- You will be allowed to solve the problem however you wish. Here, we will show you one way to solve it, which should also give you a general idea about how FF-Replan works.
- The idea is the following:
  - 1 We determinize the planning problem. In this case, we will simply assume that actions will not move the agent sideways or backwards, i.e., the action up will always move the agent up, down will always move the agent down, and so on. (Note that this corresponds to the “Most-likely-outcome determinization” .)
  - 2 We plan the path (e.g., using dijkstra algorithm).
  - 3 We execute the plan step by step (now with the prescribed probabilities of outcomes) and:
    - a If the result of the action agrees with the plan, then we continue with the next action in the plan.



# FF-replan example

So, for our example task, we start by finding a **plan** using dijkstra on the determinized problem.

We get: (**right**, **right**, **down**, down, right, right, down, down, down)

Now, we execute the plan.

The first action right moves the agent right, so we continue.

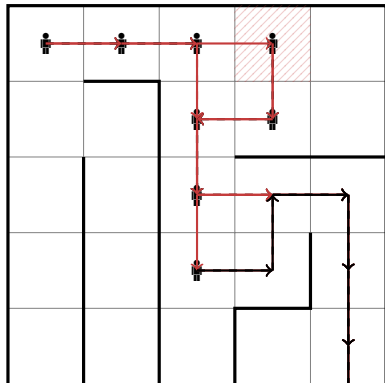
The second action right also moves the agent right, so we continue.

But, the third action down moves the agent right (this was the step sideways with 15% probability).

So, we discard the current plan.

And replan, again.

And we execute the plan (down, left, down, right, right, down, down, down) step by step. Plan: (**down**, **left**, **down**, **right**, right, down, down, down) Not let's say that the next three actions move in the correct direction





# Assignment 2 description - data

## The description of your assignment:

- 1 We will provide a dataset consisting of 25 mazes of various sizes in ascii format (on the course website).

The format of the data files is the following:

- Each file starts with two numbers specifying the number of rows and columns (in this order).
- Then the description of the maze follows in ascii: '#' corresponds to a wall, 'S' marks the starting position of the agent, 'D' marks the delay position, 'E' marks the end position, and the space character corresponds to a free space.
- The goal is to navigate the agent from 'S' to 'E' with up/down/left/right probabilistic actions as described before.
- An example of a data file describing a 7x7 maze is depicted on the right.
- All mazes are solvable (and there are no dead-ends).

```
7 7
#S#####
#      D#
### # #
# #   #
#   # #
#   # #
#####E#
```

# Assignment 2 description

## The description of your assignment:

- 1 We will provide a dataset consisting of 25 mazes of various sizes in ascii format (on the course website).
- 2 You have to come up with a solution to this problem and implement it in a programming language of your preference. (You can choose to solve it by both online planning as we described here, or by looking for a policy using value-iteration, or you can use any other reasonable approach.)
- 3 You have to write a short ( try to keep it around 3 pages) report where you have to:
  - a Describe the solution you have chosen and discuss possible shortcomings of your proposed solution.
  - b Experimentally evaluate your implementation on the given dataset.
- 4 You have to upload your code and your report in pdf to the upload system in one zip file before the deadline (which you can find on the course website).
- 5 After the deadline, your tutor will look at your solutions, read the reports and mark them with up to 20 points. Bear in mind, that the implementation part is only a part of your grade, so focus on writing the report clearly and concisely. The tutor will provide a

## Assignment 2 grading

You will be scored on the quality of your report and the methods you have implemented. In total, you can gain **maximum of 20 points**. Each technique you implement can get you following points:

- 1 FF-replan [max 5 points]
- 2 VI and derivatives [max 10 points, max 2 points for other additional VI version]
- 3 MCTS [max 15 points]

You can also use any other method of your choice, ask the tutor for valuation of a technique not on this list.

Regarding the scoring of the report itself, different aspects of it will have following weights:

- 1 Understandability, grammar, consistency of notation [0.3]
- 2 Method implementation and description [0.3]
- 3 Experimental evaluation of methods [0.3]
- 4 Overall quality [0.1]

However, your report **MUST** include both method description AND evaluations section. If either of these sections is missing, you will not get more than 10 points.

For example:

- good report using FF-replan only will get you at most 5 points.
- if you implement MCTS and VI (15+10) and write a well-written report (0.3) with with good description of your methods (0.3) and a poor evaluation section (0.1), you will get  $\max(20, (.3 + .3 + .1 + .0) * 25) = 15$  points.

Your report *MUST*:

- include concise descriptions of methods with references. Don't rewrite descriptions of algorithms known in the literature, only highlight changes made by you.
- Include concise boxplot comparison of reward of your implemented methods from multiple runs (with different seeds) on all mazes,
- include examples of paths taken by different methods on selected mazes, similarly to the illustrations in this document (which ones is up to you, possibly the E variants),
- have a conclusion in your report, summarizing the drawbacks and advantages of different methods, possible on different maze sizes or maze types.
- use spellcheck.

Consider following *suggestions*:

- Think about the experiments, consider what is the best way to evaluate and compare your algorithms. Should the boxplots be from all mazes at once? Or for separate classes of mazes? Try to look for patterns in your results and adjust your presentation accordingly.
- Apart from accumulated reward, you should probably report aggregate running times and number of steps your agent needed to reach the goal.
- We recommend using latex, but anything goes (MS Word, jupyter, html). If possible, try to submit a pdf.
- *Use spellcheck.*