

COMPUTATIONAL APPLICATIONS TO POLICY AND STRATEGY

EXERCISES WEEK 1

L. KLENNER, H. FUNG, C. COMBS


1. Rule-based Systems and Finite State Machines

In this section, you will build and evaluate simple rule-based decision-making algorithms for navigation (**1.1.**) You will evaluate a rule-based algorithm used for finding the Nash Equilibria in a 2x2 game and assess its limitations (**1.2.**) You will translate the fuzzy description of a task into computable decision rules and decompose a high-level task into computable sub-tasks (**1.3.**) Lastly, you will address various conceptual problems related to rule-based systems and finite state machines.

1.1. Rule-based Pathfinding

In this exercise, you need to design a rule-based algorithm that optimally navigates an autonomous supply vehicle to a desired location. The environment in which the vehicle is located is given as 5x3 grid-world, in which each state is denoted by a pair of x, y coordinates such as (1, 1) or (5, 3). The desired location is denoted as the *Goal*. The image of the autonomous vehicle denotes its starting position.

A simple version of the 5x3 world is given below.

3		↑			
2	←		→		
1		↓			Goal
	1	2	3	4	5

We make the following assumptions:

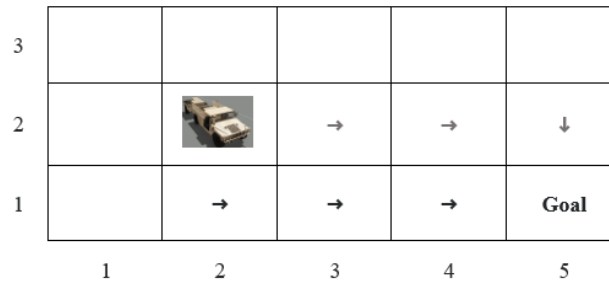
- The vehicle can move into **four directions**, right (→), left (←), up (↑), down (↓) and makes one move at each time step.
- The vehicle can only sense its current state and **cannot see into an adjacent state**, unless it enters that state. Above, the field of vision of the vehicle is shaded grey.
- As a designer, you have perfect information about the world, e.g., you can see into all of the 15 states of the world. However, **your knowledge of the world is restricted to what is explicitly specified**,

e.g. if the location of the goal is denoted as “?”, then you do not know where on the map the goal is located.

Your task is to find the optimal path to reach the goal and specify a decision rule that allows the vehicle to take this path.

1.1.1. Example

Before you start, let’s consider a simple example:



The vehicle starts at (2, 2). The goal is located at (5, 1). Arrows represent moves to be taken when in a given state.

There are two optimal paths for the vehicle:

- Path 1* = [(2, 2), (3, 2), (4, 2), (5, 2), (5, 1)]
- Path 2* = [(2, 2), (2, 1), (3, 1), (4, 1), (5, 1)]

Taking the first path, we can write a simple decision rule for the vehicle:

- Rule 1* = [right, right, right, down]

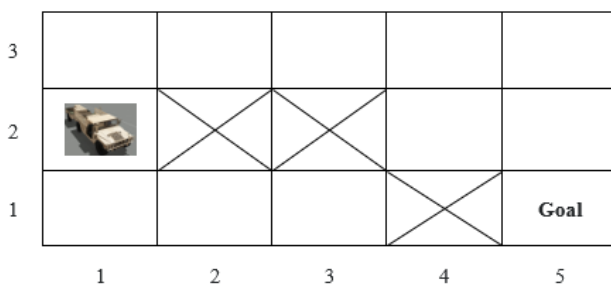
Note, that this rule is extremely simple, and for more complex environments we need to introduce additional elements, such as checking whether we have reached the goal.

1.1.2. Exercises

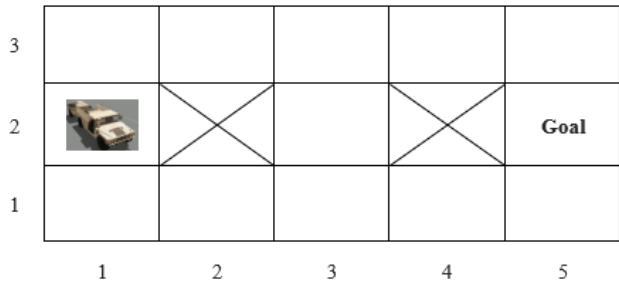
For each of the worlds, state the optimal path and a decision rule that allows the vehicle to take this path.

1.1.2.1. Pathfinding with obstacles

States with an X inside represent obstacles that cannot be crossed by the vehicle.



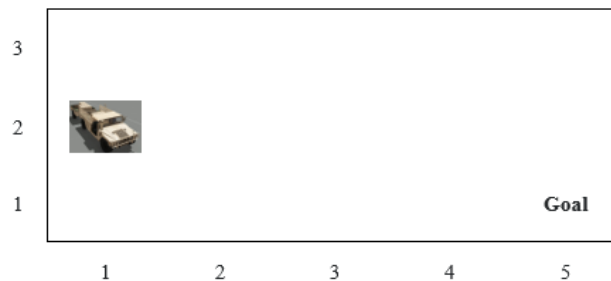
a



b

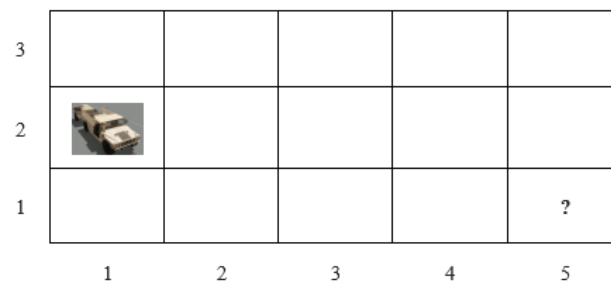
1.1.2.2. Pathfinding in continuous space

In a continuous space – as opposed to the discrete space of the 5x3 grid-world – there is no default distance covered by the vehicles’ movements. Also, more movements are possible. Note, that the labelled axes are for reference only and do not map to states in the world.



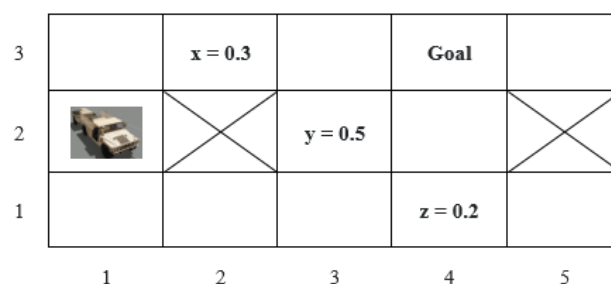
1.1.2.3. Pathfinding with an unknown goal

The location of the goal is unknown and can be in any of the 15 states. Here, it is useful to think about additional elements that you need to introduce to specify the decision rule.




1.1.2.4. Pathfinding with probabilistic combat and a known goal

Here, the environment is contested in states (2, 3), (3, 2) and (4, 1). In each of these states, the vehicle is exposed to adversarial fire with probabilities x , y and z , respectively. Assume for this and all of the following exercises that the vehicle will get destroyed if it is exposed to fire. Additionally, address whether your optimal path would change if the probability to take fire in each of the three states increases by 0.1 per time step (where one time step is equal to one move of the vehicle).




1.1.2.5. Pathfinding with probabilistic combat and an unknown goal

Same as in 1.1.2.4, but now the location of the goal is unknown. Again, assume that the vehicle cannot withstand fire. Additionally, address whether your optimal path would change if the probability to take fire in each of the three states increases by 0.1 per time step (where one time step is equal to one move of the vehicle).

3		$x = 0.3$?	
2			$y = 0.5$		
1				$z = 0.2$	
	1	2	3	4	5


1.1.2.6. Pathfinding with unknown probabilistic combat and a known goal

Same as in 1.1.2.4., but now the probabilities x , y , and z are unknown. Additionally, address whether your optimal path would change if the probability to take fire in each of the three states increases by 0.1 per time step (where one time step is equal to one move of the vehicle).

3		$x = ?$		Goal	
2			$y = ?$		
1				$z = ?$	
	1	2	3	4	5

1.1.2.7. Pathfinding with unknown probabilistic combat and an unknown goal

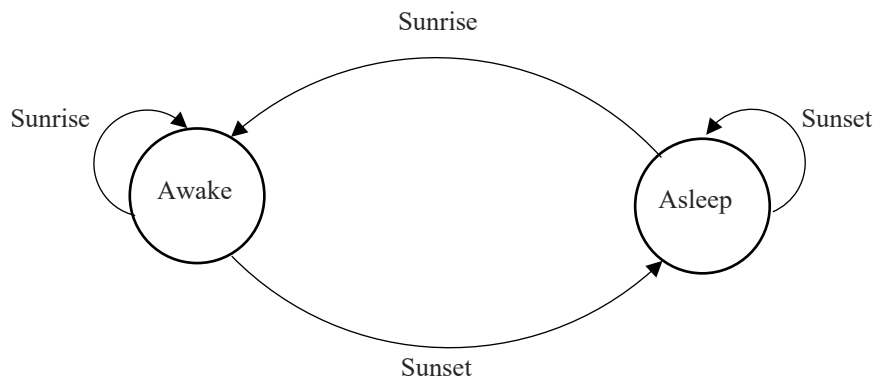
This environment is again contested in states (2, 3), (3, 2) and (4, 1). Now, however, neither the location of the goal nor probabilities x , y , and z are known. Additionally, address whether your optimal path would change if the probability to take fire in each of the three states increases by 0.1 per time step (where one time step is equal to one move of the vehicle).

3		$x = ?$?	
2			$y = ?$		
1				$z = ?$	
	1	2	3	4	5

1.2. Finite State Machines for Warning Systems and Radio Control

In this exercise, we want to develop our understanding of Finite State Machines (FSM) on two simple task, a warning system and radio control of our autonomous supply vehicle.

First, let's recall the basic idea of an FSM. An FSM is a machine that can be in a finite number of states and that changes its state based on inputs from the environment. You can think of a human's sleeping and waking habits as an FSM. There two states, "asleep" and "awake" and each of these states is activated by an input from the environment, such as "sunset" and "sunrise". If the environment tells the human that the sun sets, she will be in a sleepy state, etc. (unless there is coursework due but maybe even then). If the human moves from being asleep to being awake, we refer to this as a *state transition*. A regular day might thus have two state transitions, one for waking up and the other for falling asleep.



1.2.1. Exercises

1.2.1.1. Building an FSM Warning System Wearable

You are tasked with designing a wearable device for the Afghan National Army that guides the sleeping habits of soldiers by sending "wake" and "sleep" messages as well as sending them a warning signal in the case of an enemy attack or approaching deployment. Your task is to draw the FSM architecture of this device. Note, that the soldiers need to receive the warning signal both when they are sleep and awake. You can expand on the FSM architecture given above.

1.2.1.2. Building an FSM Radio Control

To connect our autonomous supply vehicle to a human supervisor, we want to build a radio control system that serves as a communication link between the supervisor and the vehicle. The radio control is implemented in the vehicle and has three states, "send coordinates", "return to last coordinates", "return to base". The radio control takes three inputs from the environment, "connection clear", "connection disrupted", "goal reached", respectively. Your task is to draw the FSM architecture of the radio control.

1.3. Rules for Finding the Nash Equilibrium

In this exercise, your task is to evaluate a rule-based algorithm for finding the Nash equilibrium in a prisoner's dilemma game. The goal of this exercise is to get you working on basic Python code and understand how code represents decision rules. If you're unfamiliar with game theory, you can grasp the basics necessary for this exercise through a quick online search. The code you will be working on is very simple but yields the correct

Nash equilibrium for the prisoner's dilemma. In general, more sophisticated algorithms such as *Support Enumeration* or *Lemke-Howson* exist for handling this task.

You are given the prisoner's dilemma in the following form:

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	3, 3	0, 5
	Defect	5, 0	1, 1

Your tasks are:

- Find the Nash equilibrium and write down your method for finding it
- Look at the code in the appendix and describe its method for finding the Nash equilibrium
- Briefly compare your method to that represented by the code

In addition, consider the game stated below:

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	0, 0	-1, 1
	Defect	1, -1	-1000, -1000

For this game, your tasks are:

- Find the Nash equilibrium
- Assess the code in the appendix and evaluate what Nash equilibrium it would return
- Briefly outline where you see flaws in the code

1.4. Decoding Rules from Fuzzy Information

In this exercise, your task is to decompose a fuzzy first-person account of a room-clearing operation into clear decision rules that allow an autonomous system to search the same building that is described in the account. The goal of this exercise is to improve our understanding of code as a representation of human decision-making that needs to mitigate the imperfect information, which we often only have on how humans make decisions. Your task is to understand the blueprint of the building that is described in the account given below and write a set of navigational decision rules for an autonomous system to completely search the building (e.g., enter every room inside the building). In defining these rules, you can follow the order of rooms given in the account or establish an order that you think is more optimal. If there is uncertainty about the precise details of the blueprint, the rules you specify should be robust enough to perform in the uncertain environment.

Account of clearing a building in a conflict zone:

“When entering the building, I first take a look around to get a sense of the features of the location. Most of the buildings over here start with a small hallway, with doors going in multiple directions. That can be challenging, because it allows for a lot of movement through the location. I take whatever door is closest to my entrance position first. In this operation here, the door let into a small room with an adjacent door. Nothing

in the room, but the adjacent one caught my attention. I moved up close next to the door and then turned sideways to enter the room in one motion. One person on the ground leaned against the wall at the back of the room, as if waiting for a long time. The translator got in and started talking in the dark. Then I am back at the hallway and the next door leads to the left. We're keeping the stairway at the end of the hallway secured. For now, no one has moved up to the second floor. The door leads into a wide room with multiple open doors, suggesting movement on this side of the building. This time I take the door on the left first. The room contains a table and a flat bed and the room is dark again and I move back into the main room. The other door is at the end of the room, leading through a short, angular hallway to a cubic-shaped room. Three family members are located in the corner to my right and we need to get the translator back in here. I'm moving to the hallway. Behind me someone returns from the third door in the main room but it's not the translator. ... We're moving up to the second floor now, lined up along the sides of the stairway. The stairway opens up into a wide room and there is white light flooding from the ceiling, touching the walls in a steady, symmetric shape. This is when we hear movements behind the wall on the left. Half the team takes the door in front of us and I take the one on the right, moving through a set of small, connected rooms until I reach its end. The room I'm in now is empty but for another bed and a set of chairs and a fan next to the bed that is still moving from side to side. I hear the confirmation coming in on the comms that the team has secured the asset on the other side of the second floor and I move back through the set of three rooms to the wide room and into the other door. There are only two rooms here and one of them is half hidden behind a metal door that seems impossible to close."

1.5. Conceptual problems

Briefly outline your answers to the following questions:

2. What is a rule-based system?
3. What tasks are generally conducive to being solved through a rule-based system? Why?
4. What are some *development* challenges of rule-based systems?
5. What are some *implementation* challenges of rule-based systems?

```

### Simple code for solving for the Nash equilibrium in a prisoner's dilemma

# initialize the game

player_1_utilities = [[3, 0], [5, 1]] # [["cooperate", "cooperate"], ["defect", "defect"]]
player_2_utilities = [[3, 5], [0, 1]] # [["cooperate", "defect"], ["cooperate", "defect"]]

error = []

# compare player 1's utilities across strategies

if player_1_utilities[0][0] > player_1_utilities[1][0]:
    player_1_mark_1 = "cooperate"
else:
    player_1_mark_1 = "defect"

if player_1_utilities[0][1] > player_1_utilities[1][1]:
    player_1_mark_2 = "cooperate"
else:
    player_1_mark_2 = "defect"

# compare player 2's utilities across strategies

if player_2_utilities[0][0] > player_2_utilities[0][1]:
    player_2_mark_1 = "cooperate"
else:
    player_2_mark_1 = "defect"

if player_2_utilities[1][0] > player_2_utilities[1][1]:
    player_2_mark_2 = "cooperate"
else:
    player_2_mark_2 = "defect"

# check for consistency across player 1's selected strategies

if player_1_mark_1 == player_1_mark_2:
    player_1_equilibrium_strategy = player_1_mark_1
else:
    print("Error. Could not determine Player 1's equilibrium strategy.")
    error.append("True")

# check for consistency across player 2's selected strategies

if player_2_mark_1 == player_2_mark_2:
    player_2_equilibrium_strategy = player_2_mark_1
else:
    print("Error. Could not determine Player 2's equilibrium strategy.")
    error.append("True")

if len(error) == 0:
    nash_equilibrium = [player_1_equilibrium_strategy, player_2_equilibrium_strategy]
    print(nash_equilibrium)

# Output = ["defect", "defect"]

```
