Kick-off on 421

IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

MAD - Decision Under Uncertainty

The first series of tutorials is based on a customized version of the dice game **421**. The goal of this game is to make the best combination with 3 dice after 2 possibilities to roll again the dice or a subset of dice.

The proposed version of the game is implemented in **Python3**.

Install

First configure a proper **Python3** environment. We suggest working under Linux operating system and the editor **Visual Studio Code**.

- Open Visual Studio Code on a new directory (`mad-d2u` for instance).
- Open a terminal on Visual Studio Code
- Execute `python3` command in the terminal then `exit()` to test your configuration.

Then you can download the **421** game.

- Download game421 and unzip it `unzip hackagames-421.zip`
- Execute the game to test it: `python3 play.py`

Play 421

The simple *Python* implementation of **421** come with a very simple User Interface (UI) on your terminal. At each time step, the game state is printed then an action is asked until the game reach a final state (a horizon equals to 0).

The expected action format is 'a1-a2-a3' with a1, a2 and a3 as '*keep*' or '*roll*'. For example: `keep-keep-roll`, `roll-roll-roll`, `roll-keep-roll`, ...

Try the game until understand correctly the scoring mechanism.

First 421 AI

Understand the code

Open *player421.py* file, it implements **2** basic players for **421** game: *PlayerRandom* and *PlayerHuman*. *PlayerHuman* is the one you just played with. It prints the game state and wait for an action from a human player.

Both the players (and all the players we will implement) is based on **4** methods accordingly to an *Agent Based Model*:

- **wakeUp**: notify the player that a game start.
- **perceive**: inform on the state of the environment
- **decide**: must return the chosen action (always called after the perceive method)
- **sleep**: notify the player that the game is ended.

PlayerRandom provide a first AI implementation with a player choosing its action randomly. To try *PlayerRandom,* replace *PlayerHuman* by *PlayerRandom* in `play.py` script then run it `python3 play.py`.

Implement our own AI

Now you are ready to propose your first AI. First create a new python file (for example: `myfirstplayer.py`) create your own class player (typically by copying *PlayerRandom*) Modify the **decide** method to return coherent action to the game situation (state).

If you have any interrogation on the `Python` implementation, asks your teacher and do not stay in a gray zone.

You can try your AI by computing the average score after 10 000 games in `play.py` script.

Retro-engineering

A system state (the game state here) is composed of the overall variables describing the systems.

What is the variable defining the state of the 421 game ?

What is the possible value of each of the variables ?

The first notion of complexity over a discrete system to analyze and control is the size of the state space (the number of reachable states). As a first estimation, the state space can be defined as the Cartesian product over the state variables.

What is the size of the state space ?

The second notion of complexity relies on the action branching.

What is the number of possible actions?

From a given state, what is the maximal number of reachable next states ?