421 - Decision Tree

The idea is to hack the 421 game to propose our own autonomous player.

The game is reachable on *replit.com*.

https://replit.com/teams/join/uwiarllynbhxlxftwgdkvfvwywrxrsvx-ChefProjetIA22

Understand the code

Open *playerSimple.py* file and identify the two main classes The first one **PlayerHuman** implements a HumanUI player modeled as a simple agent (with perception and action). It is based on **PlayerRandom** implementing a autonomous player returning a random action for each perceptive state of the game.

From these 2 classes, a simple main script is used to launch the game with a human or random player.

- Play few games to be sure to understand the game mechanism.

Generate your own Player

You will not modify the **421** implementation and so, work on your own *python* file.

Create a new python file by importing the game421 engine, copying the main function but by calling your own player (*MyPlayer*), then implement a very simple player.

Your file must look like:

```
# Agent as a very simple UI
class MyPlayer() :

    def __init__(self):
        self.results= []

    # AI interface :
    def wakeUp(self, numberOfPlayers, playerId, tabletop):
        self.scores= [ 0 for i in range(numberOfPlayers) ]
        self.id= playerId
        self.model= tabletop

    def perceive(self, turn, scores, pieces):
        self.turn= turn
        self.reward= scores[ self.id ] - self.scores[ self.id ]
        self.scores= scores
        self.dices= pieces

    def decide(self):
      return 'keep-keep-keep'

    def sleep(self, result):
      print( f'--- {str(result)}' )
      self.results.append(result)

# Activate default interface :
if __name__ == '__main__':
    main()
```

Player protocol

The player protocol followed by a game start by waking-up a player (method *wakeUp*). This step informs the player about its initial state and the possible actions during the game. Then the engine iteratively asks the player for an action (method *action()*), and inform the player about the reached situation (method *perceive()*). The reached game situation is composed of a game state and a gained

reward (or cost in case of negative reward). The methods *action()* then *perceive()* are called until the player reached a final state. Then at the end of the game, the player is virtually killed to inform in that the game end for him.

Developing a first AI

Now you are ready to propose your first AI.

The idea is to first draw the decision tree you want to implement and then implement it as a *if-then-else* script based on `self.dices` value list and `self.turn` value.

For convenience in 421 game, it is possible to build a state as a dictionary like this:

```
state= { "H":self.turn,  "D1":self.dices[0],  "D2":self.dices[1],  "D3":self.dices[2]
        }
```

You can try your AI by computing the average score after 1000 games (do not forget to remove the calls to `print` function).