Kick-off on Risky game

MAD - Decision Under Uncertainty

The second series of tutorials is based on a simple two players turn based strategic game **Risky**. The goal of this game is to take the control over a maximum of position, and to destroy the opponent armies.

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

Install

Risky requires the installation and the compilation of **HackaGames**

- Hackagames

Play Risky

The `play-risky.py` will launch the game with a simple opponent.

Go on Risky README file for more details about the game rules.

Automonous Player

In `game-risky` directory, the `simplePlayer.py` and the `confront.py` scripts provide a first random player and a launcher to confront 2 players in several games.

You can copy the `simplePlayer.py` in a new `myPlayer.py` file and modify the `confront.py` script to import `myPlayer` rather than `simplePLayer` (line modification: `from myPlayer import player`).

And that it, you are ready to develop our onw player.

State Space

Before to initialize a Q-Learning method, the idea is to define the state space, in a way it is possible to use it as keys in Q-Value dictionary. We propose a string of tuples of 3 values, one tuple per tabletop cell. The values represent respectively, `0 or 1` if the cell is owned by the player, `X` the strength of the unit on the cell and `0 or 1` if the units already moved from the last `sleep` action.

- Add a new method to your player `stateStr()` in a way it builds a list of tuples initialized on “0-0-0” for each node. Then change the value of the tuple, for each piece in the tabletop.

Example of stateStr method:

```python
def stateStr(self):
    states= ['0-0-0' for c in self.tabletop]
    for p in self.pieces :
        owner= '0'
        if p.owner == self.id :
            owner= '1'
        states[p.position]= owner + '-' + str(p.attributs[STRENGH]) +'-'+
        str(p.attributs[ACTIVATED])
    return '|'.join(states)
```

- Print the state value at the beginning of `decide method`. You also can change the heritance from `hg.PlayerVerbode` to `hg.Player` to minimize the printed information.

Apply Q-Learning

The goal of this exercise is to apply Q-Learning.

To notice that the `simplePlayer.py` script already defines the list of all the available actions to perform. The list is a list of lists and not a list of string and the list is incomplete (only one of the `move` action is defined per cell). The strength of the move is decided randomly if the move action is chosen.

However you have all the ingredients to start to investigate the use of Q-Learning on *Risky game*, in ‘perceive’ method.

```python
def perceive(self, turn, scores, pieces):
    last= self.stateStr()
    super().perceive(turn, scores, pieces)
    # ...
```

The difficulty consists in handling the combinatorial explosion of the state space. So the question is, ‘With strategy will you use to permit our player to learn how to rapidly win ?’