Agents and Tournaments

Session III - Introduction to ABM

Ali Seyhun Saral (Uni. Bologna)

14 June 2022

Recap

- Learned about python types
- Leaned about classes and instances
- Agents and their behaviour can be defined in a class

Prisoner's Dilemma Game

	Cooperate	Defect
Cooperate	(2,2)	(0,3)
Defect	(3,0)	(1,1)

Agents

- Our agents can remember only the last round (memory 1)
- Types:
  - Cooperator: Always cooperates
  - Defector: Always defects
  - Tit-for-Tat: Start with cooperating, then do what the opponent did last round
  - Random: Random strategy

Practice

Ex9_Agents1.ipynb

Implementing the Game

Function with two inputs

```
def pd_game(action1, action2):
    if action1 == 'C':
        if action2 == 'C':
            return (2,2)

    if action1 == 'D':
        if action2 == 'D':
            return (0,0)

    if action1 == 'C':
        if action2 == 'D':
            return (0,3)

    if action1 == 'D':
        if action2 == 'C':
            return (3,0)
```

. . .

```
pd_game('C', 'C')
```

```
(2, 2)
```

. . .

```
pd_game('C', 'D')
```

```
(0, 3)
```

Implementing the Game

Function with one input

```
def pd_game(actions):
    action1, action2 = actions

    if action1 == 'C':
        if action2 == 'C':
            return (2,2)

    if action1 == 'D':
        if action2 == 'D':
            return (0,0)

    if action1 == 'C':
        if action2 == 'D':
            return (0,3)

    if action1 == 'D':
        if action2 == 'C':
            return (3,0)
```

. . .

```
pd_game(['C', 'C'])
```

```
(2, 2)
```

. . .

```
pd_game(['C', 'D'])
```

```
(0, 3)
```

Tuples

- Tuples are like lists: an ordered sequence of elements
- Tuples are immutable
- You can replace the instances of a list with a tuple

```
pd_game(('C', 'D'))
```

```
(0, 3)
```

Lists might change by mistake

```
a = [1,2,3]
b = a
a[0] = 5
print(b)
```

```
[5, 2, 3]
```

. . .

- Copying the list:
- `b = a[:]`

```
a = [1,2,3]
b = a[:]
a[0] = 5
print(b)
```

```
[1, 2, 3]
```

Copying the list

```
import copy
copy.copy() # copy the list
copy.deepcopy() # copy the list and the elements
```

Game as a Dictionary

```
pd_game = {('C','C'): (2,2), ('D','D'): (1, 1), ('C','D'): (0,3), ('D','C'): (0,3)}
pd_game[('D','C')]
```

```
(0, 3)
```

Implementing the types

Type as a subclass

Type is defined as a "child" class. The usual workings (methods and properties) are inherited from the parent object and methods and properties belonging to that specific subclass is defined.

```
class Agent:
    def __init__(self):
        self.payoff = 10


class Cooperator(Agent):
    def respond(self, action=None):
        return 'C'


class Defector(Agent):
    def respond(self, action=None):
        return 'D'
```

Type as attribute

```
class Agent:
    def __init__(self, typ):
        self.payoff = 10
        self.typ = typ

    def respond(self, action):
        if self.typ == 'cooperator':
            return 'C'
```

```
        if self.typ == 'defector':
            return 'D'

        if self.typ == 'titfortat':
            if action is None:
                return 'C'

            if action == 'D':
                return 'D'
```