

# Projet Foot 2I013

Nicolas Baskiotis

`nicolas.baskiotis@lip6.fr`

Université Pierre et Marie Curie (UPMC)  
Laboratoire d'Informatique de Paris 6 (LIP6)

S2 (2015-2016)

# Description de l'UE

## Objectifs du cours

Apprendre :

- à faire un projet;
- à appréhender un nouvel environnement (Python);
- quelques outils (design pattern, interface graphique);
- petite introduction à l'apprentissage statistique et IA;
- faire un rapport et une soutenance.

**Ce n'est pas :**

- un cours approfondi de python,
- que du codage.

## Pré-requis

- notions d'algorithmique et de structure,
- de la motivation !

# Déroulement de l'UE

## En pratique

- 1h45 de cours le lundi 10h45-12h30;
- 3h30 de TME le lundi 16h-19h45;
- **web** : `http://webia.lip6.fr/~baskiotisn`
- **slides et code** :  
`http://github.com/baskiotisn/soccersimulator`
- **email** : `nicolas.baskiotis@lip6.fr`  
(mettre dans le titre [2I013])

## Évaluation

- CC : 70%
  - un partiel sur machine (à mi-parcours)
  - un rapport (à la fin)
  - le code (à la fin)
  - participation (tout le temps)
- Examen : 30%
  - soutenance orale (à la fin)
  - examen sur machine (à la fin).

# A propos du projet

## Objectif

- Développer des IAs (plus ou moins intelligentes) de joueurs de football

## Code fourni : le simulateur

- les règles du jeu
- la gestion des matchs
- une interface graphique simple

## Code demandé : implémentation des joueurs

- pour commencer, des joueurs simples
- puis des joueurs plus intelligents (notion d'apprentissage automatique)
- bonus : apprentissage avancé

# Championnat

## Organisation :

- a partir de la 2 ou 3ème semaine (selon l'avancement), chaque semaine une série de rencontre, tous les groupes rencontrent tous les groupes
- catégories : 1 contre 1, 2 contre 2, 4 contre 4

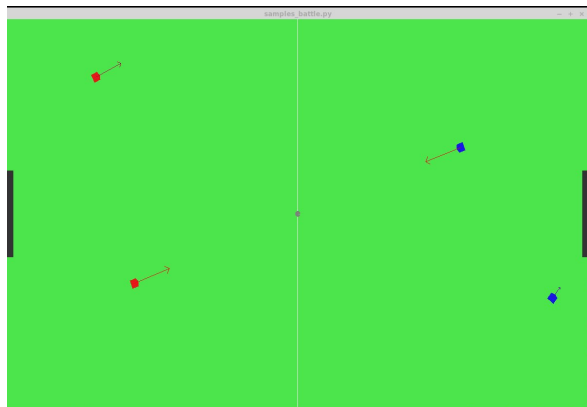
## Evaluation du controle continu

- classement dans le championnat, mais il ne suffit pas de gagner !
- prime aux joueurs les mieux pensés, justifiés,
- progression d'une semaine à l'autre,
- participation.

# Plateforme de simulation

## Besoins

Concept de :

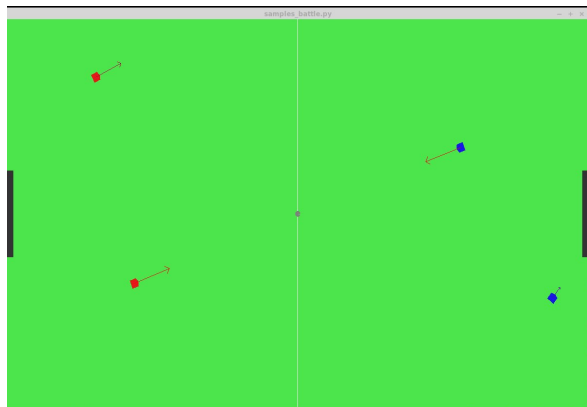


# Plateforme de simulation

## Besoins

Concept de :

- terrain
- ballon
- joueur
- équipe
- tournoi
- c'est tout ?

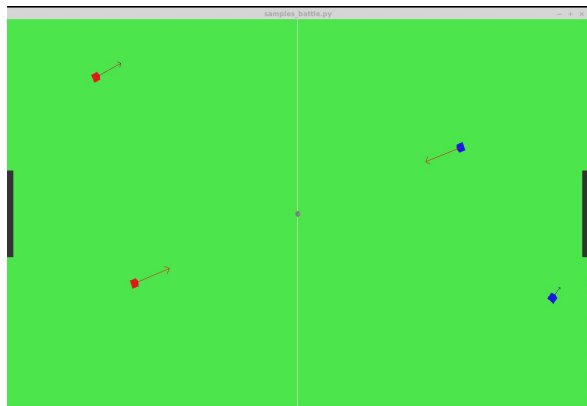


# Plateforme de simulation

## Besoins

Concept de :

- terrain
- ballon
- joueur
- équipe
- tournoi
- c'est tout ?

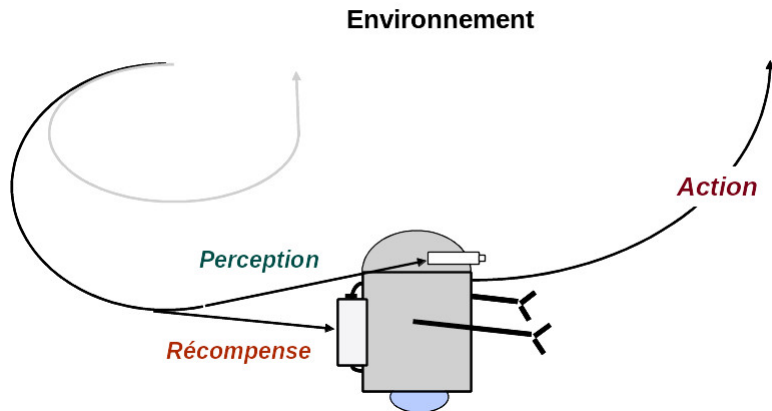


Un joueur est une coquille vide !

⇒ il faut lui donner les moyens de réagir



# Un joueur = un agent



# Modélisation agent

## Principe

- Environnement
  - tout ce qui est extérieur à l'agent
- Etat
  - ce que perçoit l'agent
- Action
  - ce que peut décider l'agent
- Récompense
  - donnée par l'environnement de l'agent à l'issue d'actions

## Exemples

- Jeu d'echecs
- Tetris
- Sudoku
- ... et le foot.

# Plan

**Résumé synthétique**

Petite intro python

Plateforme de la simulation

# Modélisation Agent : Foot

- Environnement = plateforme de simulation
- Agent : le joueur
- Action :
  - Déplacement
  - Tir
- Etat
  - Position/vitesse des joueurs
  - Terrain
  - Position/vitesse de la balle
- Récompense
  - +1 si match gagné
  - 0 si match nul
  - -1 si match perdu

# Plan

Résumé synthétique

**Petite intro python**

Plateforme de la simulation

# Python : un langage interprété

## Peut être exécuté

- en console : interaction direct avec l'interpréteur
- par exécution de l'interpréteur sur un fichier script : `python fichier.py`

## Opération élémentaire

```
# Affectation d'une variable
a = 3
# operations usuelles
(1 + 2. - 3.5), (3 * 4 / 2 ), 4**2
# Attention ! reels et entiers
1/2, 1./2
# Opérations logiques
True and False or (not False) == 2>1
# chaines de caracteres
s = "abcde"
s = s + s # concatenation
# afficher un resultat
print(1+1-2,s+s)
```

# Structures : N-uplets et ensembles

Liste d'elements ordonnes, de longueur fixe, non mutable : aucun element ne peut etre change apres la creation du n-uplet

```
c = (1,2,3) # creation d'un n-uplet
c[0],c[1]  # acces aux elements d'un couple,
c + c     # concatenation de deux n-uplet
len(c)    # nombre d'element du n-uplet
a, b, c = 1, 2, 3 # affectation d'un n-uplet de variables
```

```
s = set() # creation d'un ensemble
s = {1 ,2 ,1}
print(len(s)) #taille d'un ensemble
s.add('s')    # ajout d'un element
s.remove('s') # enlever un element
s.intersection({1,2,3,4})
s.union({1,2,3,4})
```

# Structures : Listes

Structure très importante en python. Il n'y a pas de tableau, que des listes (et des dictionnaires)

```
l = list() # creation liste vide
l1 = [ 1, 2 ,3 ] # creation d'une liste avec elements
l = l + [4, 5] #concaténation
zip(l1,l2) : liste des couples
len(l) #longueur
l.append(6)      # ajout d'un élément
l[3]            #accés au 4-eme element
l[1:4]          # sous-liste des elements 1,2,3
l[-1],l[-2]    # dernier element, avant-dernier element
sum(l)          # somme des elements d'une liste
sorted(l)       #trier la liste
l = [1, "deux", 3] # une liste composee
sub_list1 = [ x for x in l1 if x < 2] # liste comprehension
sub_list2 = [ x + 1 for x in l1 ] # liste comprehension 2
sub_list3 = [x+y for x,y in zip(l1,l1)] # liste comprehension 3
```



# Structures : Dictionnaires

Dictionnaires : listes indexées par des objets (hashmap), très utilisés également. Ils permettent de stocker des couples (cle,valeur), et d'accéder aux valeurs à partir des cles.

```
d = dict() # creation d'un dictionnaire
d['a']=1   # presque tout type d'objet peut etre
d['b']=2   # utilise comme cle, tout type d'objet
d[2]= 'c'  # comme valeur
d.keys()   # liste des cles du dictionnaire
d.values() # liste des valeurs contenues dans le dictionnaire
d.items()  # liste des couples (cle,valeur)
len(d)     #nombre d'éléments d'un dictionnaire
d = dict([ ('a',1), ('b',2), (2, 'c')]) # autre methode pour creer un
d = { 'a':1, 'b':2, 2:'c' } # ou bien...
d = dict( zip(['a','b',2],[1,2,'c'])) #et egalement...
d.update({'d':4,'e':5}) # "concatenation" de deux dictionnaires
```

# Boucles, conditions

Attention, en python toute la syntaxe est dans l'indentation : un bloc est formé d'un ensemble d'instructions ayant la meme indentation (meme nombre d'espaces precedent le premier caractère).

```
i=0
s=0
while i<10:  # boucle while
    i+=1      #indentation pour marquer ce qui fait parti de la boucle
    s+=i
s=0
for i in [1, 2, 3]: #boucle for
    j = 0          # indentation pour le for
    while j<i:     # boucle while
        j+=1       # deuxieme indentation pour le bloc while
        s = i + j
    s = s + s # retour a la premiere indentation, instruction du bloc
```

# Fonctions

```
def increment(x):      # definition d'une fonction par le mot-cle def
    return x+1         # retour de la fonction

y=increment(5)         # appel de la fonction

def somme_soustraction(x,y=2):
    # possibilite de donner une valeur par default aux parametres
    return x+y,x-y     # possibilite de retourner
                        # un n-uplet de valeurs,
                        # equivalent a (x+y,x-y)
xsom,xsub = somme_soustraction(10,5) #ou
res = somme_soustraction(10,5)
xsom == res[0],res[1]
```

# Fichiers

```
##Lire
f=open("/dev/null","r")
print(f.readline())
f.close()
```

```
#ou plus simplement
with open("/dev/null","r") as f :
    for l in f:
        print l
```

```
## Ecrire
f=open("/dev/null","w")
f.write("toto\n")
f.close()
```

```
#ou
with open("/dev/null","w") as f:
    for i in range(10):
        f.write(str(i))
```

# Modules

- Un module groupe des objets pouvant être réutilisés
  - module `math` : `cos, sin, tan, log, exp, ...`
  - module `string` : manipulation de chaîne de caractères
  - module `numpy` : librairie scientifique
  - modules `sys, os` : manipulation de fichiers et du système
  - module `pdb, cProfile` : debuggage, profiling
- importer un module : `import module [as surnom]` et accès au module par `module.fonction` (ou `surnom.fonction`)
- importer un sous-module ou une fonction : `from module import sousmodule`
- tout répertoire dans le chemin d'accès qui comporte un fichier `__init__.py` est considéré comme un module !
- tout fichier python dans le repertoire courant est considéré comme module : `import fichier` si le fichier est `fichier.py` (ou plus couramment `from fichier import *`)

# Les objets : très grossièrement

- c'est une structure : contient des variables stockant des informations
- contient des *méthodes* (fonctions) qui agissent sur ses variables,
- contient *un constructeur*, fonction spécifique qui sert à l'initialiser.
- le `.` sert à indiquer l'appartenance d'un objet/fonction à un autre objet :  
`obj.fun` est l'appel de la fonction `fun` de l'objet `obj`
- *self* indique l'objet lui-même

Un objet Agent pourrait être ainsi le suivant :

```
class Agent(object):
    def __init__(self,nom):
        self.nom = nom
        self.x = 0
        self.y = 0
    def agir(self,etat):
        action = None#faire qqe chose
        return action
    def get_position(self):
        return self.x, self.y
    def safficher(self):
        print ("Je_suis_s_en_d,%d"
              %(self.nom,self.x,self.y))

a = Agent("John") # creation
a.x, a.y = 1, 1 # déplacement
a.safficher() #equivalent a
Agent.safficher(a)
a.mavar = 4 #ajout d'une variable
```

# Plan

Résumé synthétique

Petite intro python

Plateforme de la simulation

# Plateforme : les objets en présence

- `Vector2D` : `x`, `y` et toutes les opérations vectorielles
- `MobileMixin` : base pour tous les objets mobiles, contient `position` et `vitesse`
- `SoccerAction` : contient `acceleration` et `shoot`, deux `Vector2D`
- `SoccerState` :
  - `player_state(self, id_team, id_player)` : contient `position` et `vitesse`
  - `ball:ball.position, ball.vitesse`
  - `score_team1, score_team2, get_score_team(self, i)`
  - `step` : numéro de l'état
- `Player` : joueur, contient un nom (`name`) et une stratégie (`strategy`)
- `AbstractStrategy` : modèle de stratégie, toute stratégie doit implémenter la méthode `compute_strategy(...)`
- `SoccerTeam` : liste des joueurs
- `SoccerMatch` : deux équipes et la durée du match
- `SoccerTournament` : tournoi



# Boucle d'action et stratégie

## Boucle d'action

- Pour `step` de 0 à `MAX_STEP`
  - calcul pour chaque jouer l'action selon l'état présent : méthode `compute_strategy(self, state, id_team, id_player)`
  - cette méthode doit renvoyer un objet `SoccerAction` correspondant à l'action
  - calcul du prochain état en fonction des actions des joueurs.

## Stratégie constante

```
class AbstractStrategy:
    def __init__(self, name):
        self.name = name
    def compute_strategy(self, state, id_team, id_player):
        return SoccerAction()
```

# Boucle d'action et stratégie

## Boucle d'action

- Pour `step` de 0 à `MAX_STEP`
  - calcul pour chaque jouer l'action selon l'état présent : méthode `compute_strategy(self, state, id_team, id_player)`
  - cette méthode doit renvoyer un objet `SoccerAction` correspondant à l'action
  - calcul du prochain état en fonction des actions des joueurs.

## Stratégie aléatoire

```
class RandomStrategy(AbstractStrategy):  
    def __init__(self):  
        AbstractStrategy.__init__(self, "Random")  
    def compute_strategy(self, state, player, teamid):  
        return SoccerAction(Vector2D.create_random(),  
                             Vector2D.create_random())
```

# Lancer une partie

```
import soccersimulator, soccersimulator.settings
from soccersimulator import AbstractStrategy, SoccerAction
from soccersimulator import SoccerTeam, SoccerMatch
from soccersimulator import Vector2D, Player, SoccerTournament
class RandomStrategy(AbstractStrategy):
    def __init__(self):
        AbstractStrategy.__init__(self, "Random")
    def compute_strategy(self, state, id_team, id_player):
        return SoccerAction(Vector2D.create_random(),
                             Vector2D.create_random())

team1 = SoccerTeam("team1", [Player("t1j1", RandomStrategy())])
team2 = SoccerTeam("team2", [Player("t2j1", RandomStrategy())])
team3 = SoccerTeam("team3", [Player("t3j1", RandomStrategy())])
match = SoccerMatch(team1, team2)
match.play() # ou
soccersimulator.show(match) # pour l'affichage graphique
tournoi = SoccerTournament(1)
tournoi.add_team(team1)
tournoi.add_team(team2)
tournoi.add_team(team3)
tournoi.play() #ou
soccersimulator.show(tournoi)
```

# Objectifs TME

- Installation de la plateforme
- Prise en main de python et de l'environnement
- Programmation du joueur aléatoire
- Programmation du joueur fonceur.

## Depot git

`https://github.com/baskiotisn/soccersimulator`

## Pour installer un module python manquant

```
pip install module --user
```

## Pour installer un module python stocké dans le repertoire courant

```
pip install -e . --user
```