Projet Foot
2I013

Nicolas Baskiotis
nicolas.baskiotis@lip6.fr

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

S2 (2014-2015)

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 infos)
- 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 possible en IHM : interface graphique évoluée (en 3D ou autre)

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

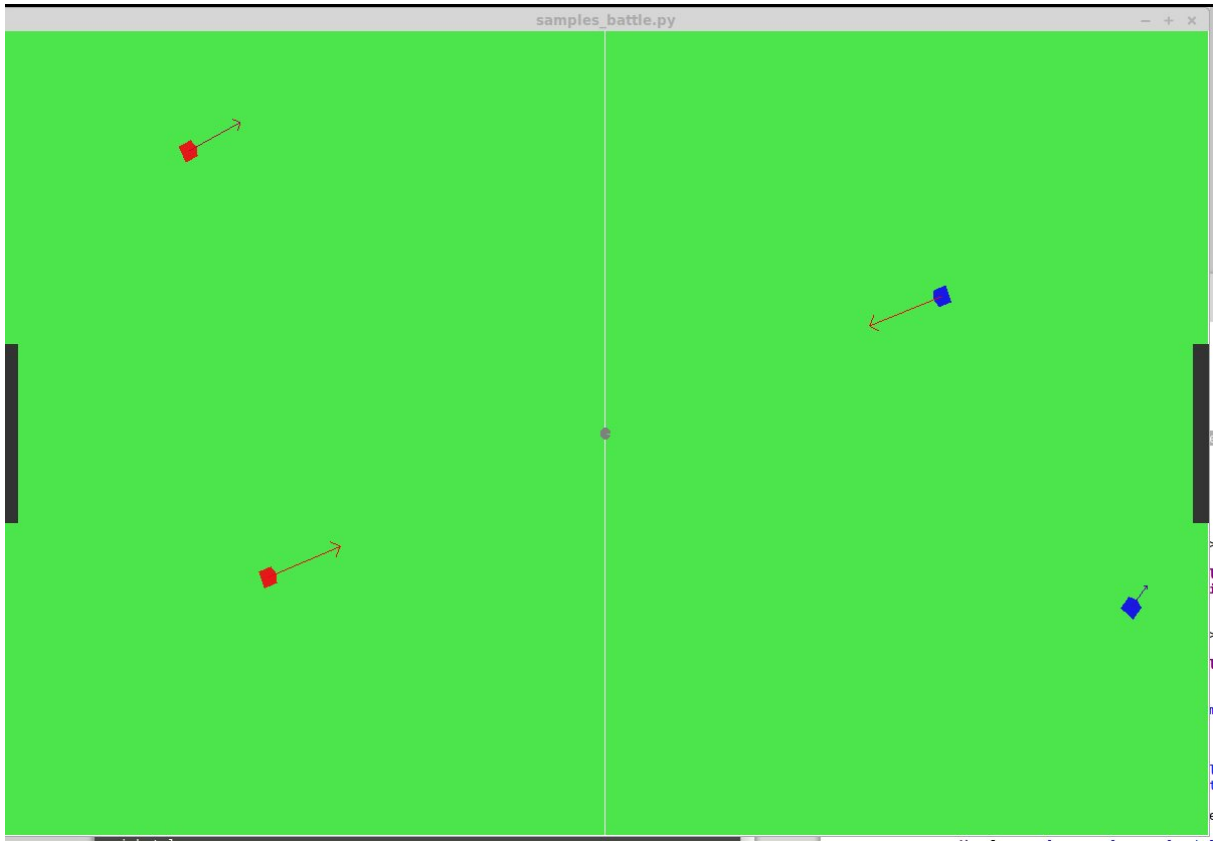
- classement dans le championnat, mais il ne suffit pas de gagner !
- prime aux joueurs les mieux pensés, justifiés,
- progression d'une semaine à l'autre,
- participation.

Plateforme de simulation

Besoins

Notion de :

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

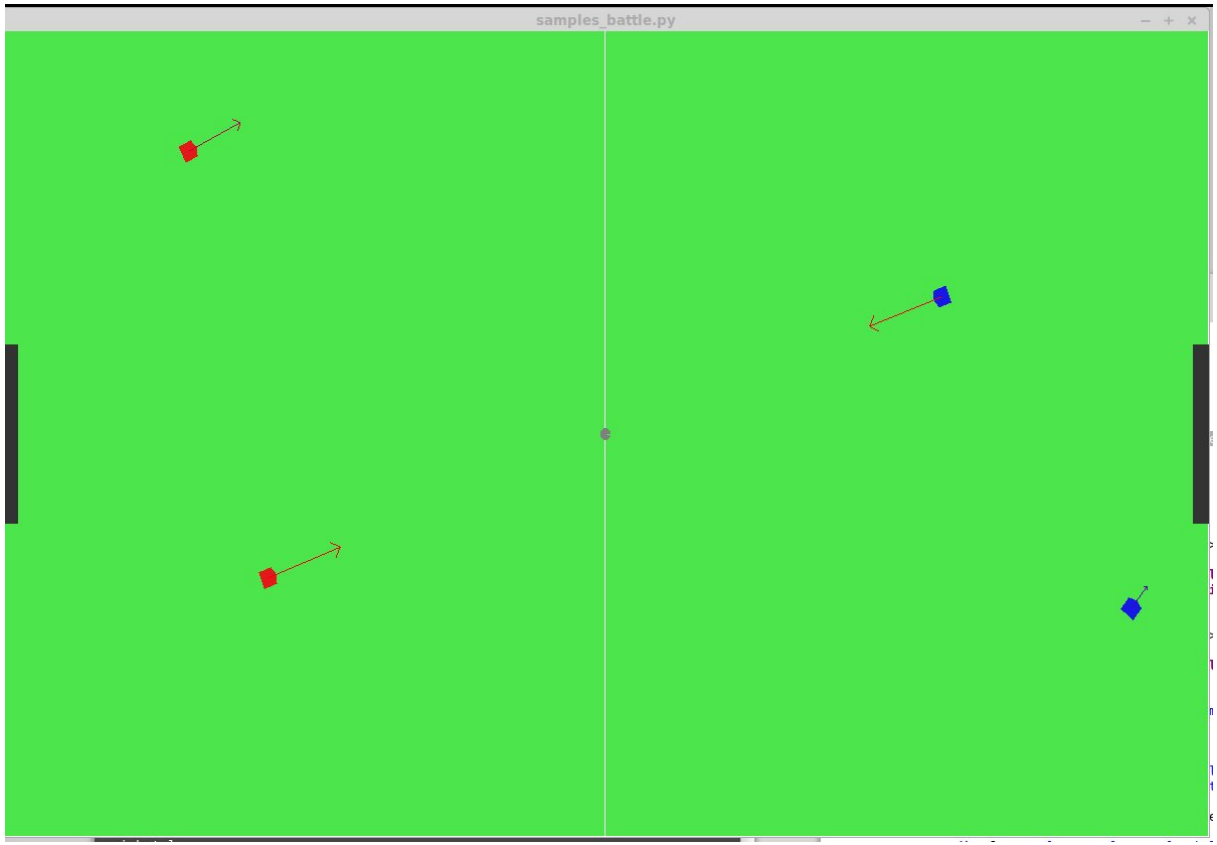


Plateforme de simulation

Besoins

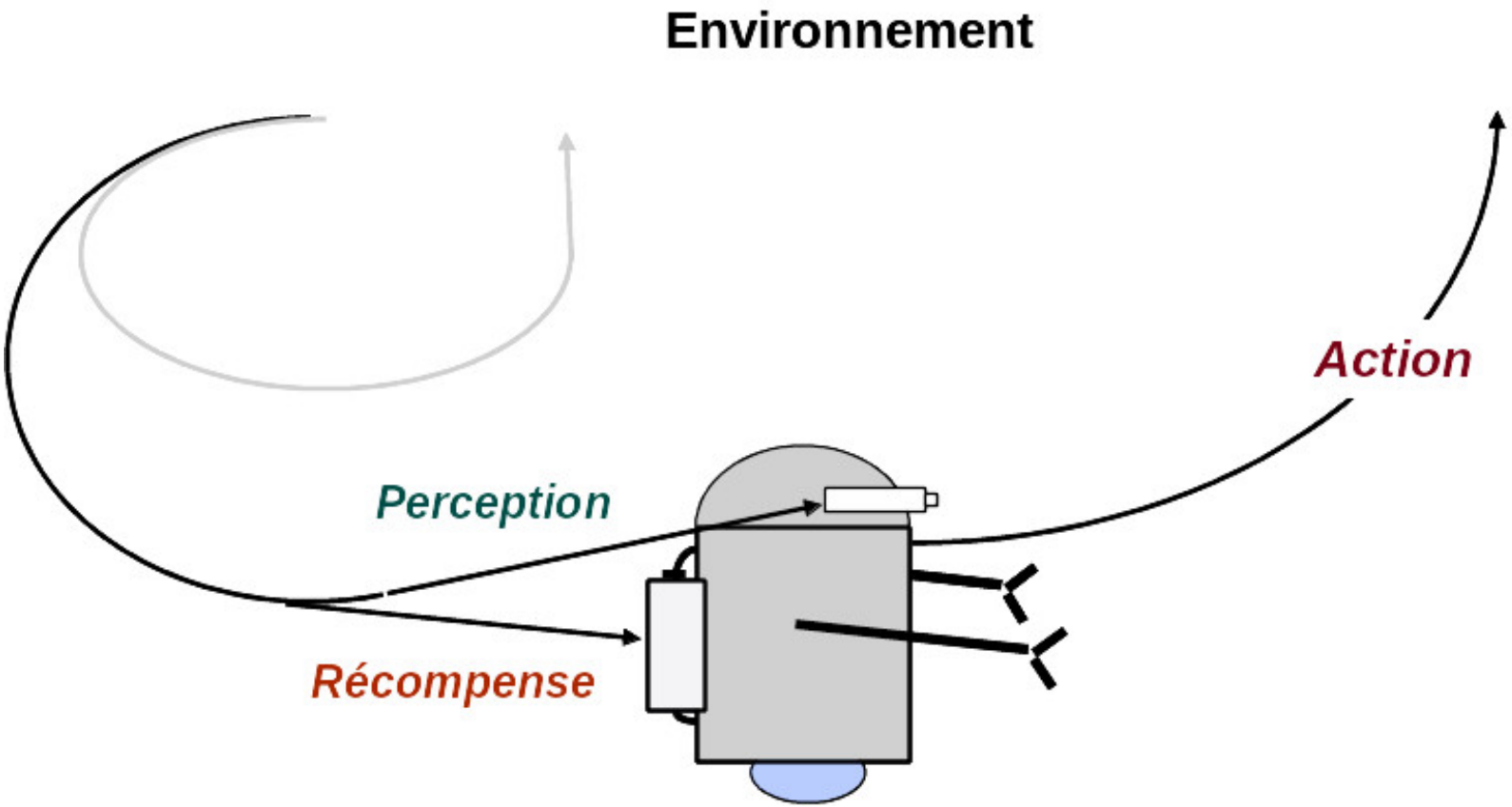
Notion de :

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



Un joueur est une coquille vide !
$\Rightarrow$ 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.

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

Plateforme : architecture (modélisation objet)

`soccer_base.py`

- constantes du jeu
- outils (`Vector2D` pour les maths, gestion des scores)

`soccerobj.py`

- `SoccerBall` : la balle
- `SoccerPlayer` : un joueur
- `SoccerTeam` : une équipe

`mdpsoccer.py`

- `SoccerAction` : une action
- `SoccerState` : un état du jeu
- `SoccerBattle` : un match

`strategies.py`

- `SoccerStrategy` : une stratégie (abstraite !)

`interface.py`

- contient une interface graphique, une interface texte et d'autres outils
- plus tard...

Un objet simple : la balle !

```
class SoccerBall(object):
    def __init__(self,position=Vector2D(),speed=Vector2D()):
        self.position=position
        self.speed=speed
    @property
    def angle(self):
        return self.speed.angle
    def copy(self):
        return SoccerBall(self.position.copy(),self.speed.copy())
    def __str__(self):
        return "Ball : %s %s"% (self.position,self.speed)
```

- creer un vecteur : `vect = Vector2D(0,0)`
- creer une balle : `maballe = SoccerBall(Vector2D(0,0),Vector2D(0,0))`
- position de la balle : `maballe.position`
- vitesse de la balle : `maballe.vitesse`

Des objets plus complexes

```
class SoccerState:
 def __init__(self,team1,\
  team2,ball):
   self.team1=team1
   self.team2=team2
   self._winning_team=0
   self.ball=ball
   self._width=GAME_WIDTH
   self._height=GAME_HEIGHT
     ...
```

```
class SoccerTeam:
  def __init__(self,name,soccer_club=None):
   self._name=name
   self._exceptions=[]
   self._players=dict()
  def compute_strategies(self,state,teamid):
   res=dict()
   for p in self.players:
    action=p.compute_strategy(state,teamid)
    res[p.name]=action
    return res
```

```
class SoccerPlayer(object):
    def __init__(self,name,strat=None):
        self._name=name
        self.position=Vector2D()
        self.angle=0.
        self.speed=0.
        self._num_before_shoot=0
        self._strategy=None
    def compute_strategy(self,state,teamid):
        if self._strategy:
            return self.strategy.compute_strategy(state,self,teamid)
        raise PlayerException('Pas de strategie définie pour le joueur
```

Une stratégie en détail

```
class SoccerAction(object):
    def __init__(self,acceleration=Vector2D(),shoot=Vector2D()):
        self.acceleration=acceleration
        self.shoot=shoot

class SoccerStrategy:
    def __init__(self,name):
        self.name=name
    def start_battle(self,state):
        raise NotImplementedError,"start_battle"
    def finish_battle(self,won):
        raise NotImplementedError,"finish_battle"
    def compute_strategy(self,state,player,teamid):
        raise NotImplementedError,"compute_strategy"
    @property
    def name(self):
        return self.name
    def copy(self):
        raise NotImplementedError,"copy"
    def create_strategy(self):
        raise NotImplementedError,"create_strategy"
```

Stratégie aléatoire

Action aléatoire :

```
pos = Vector2D.create_random()
shoot = Vector2D.create_random()
action = SoccerAction(pos,shoot)
```

Agent aléatoire :

```
class RandomStrategy(SoccerStrategy):
    def __init__(self):
        self.name="Random"
    def start_battle(self,state):
        pass
    def finish_battle(self,won):
        pass
    def compute_strategy(self,state,player,teamid):
        pos = Vector2D.create_random()
        shoot = Vector2D.create_random()
        return SoccerAction(pos,shoot)
    def copy(self):
        return RandomStrategy()
    def create_strategy(self):
        return RandomStrategy()
```

Lancer une partie

```
from soccersimulator import  Vector2D, SoccerBattle, SoccerPlayer, Soc
from soccersimulator import PygletObserver,ConsoleListener,LogListener
class RandomStrategy(SoccerStrategy):
    def __init__(self):
        self.name="Random"
    def start_battle(self,state):
        pass
    def finish_battle(self,won):
        pass
    def compute_strategy(self,state,player,teamid):
pass
    def copy(self):
        return RandomStrategy()
    def create_strategy(self):
        return RandomStrategy()
team1=SoccerTeam("team1")
team2=SoccerTeam("team2")
team1.add_player(SoccerPlayer("t1j1",RandomStrategy()))
team2.add_player(SoccerPlayer("t2j1",RandomStrategy()))
team1.add_player(SoccerPlayer("t1j2",RandomStrategy()))
team2.add_player(SoccerPlayer("t2j2",RandomStrategy()))
battle=SoccerBattle(team1,team2)
obs=PygletObserver()
obs.set_soccer_battle(battle)
pyglet.app.run()
```

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`