

2I013 Groupe 1

Projet Foot

Maxime Sangnier – Nicolas Baskiotis

`maxime.sangnier@lip6.fr`

2019

Laboratoire d'Informatique de Paris 6 (LIP6)
Sorbonne Université

Description de l'UE

Introduction au projet

Plateforme de la simulation

Géométrie vectorielle

Objectifs du TME

Description de l'UE

Objectifs du cours

Apprendre :

- à réaliser un projet informatique personnel ;
- quelques outils (interface graphique, motifs de conception) ;
- ce que sont l'apprentissage statistique et l'intelligence artificielle ;
- à préparer un rapport et une soutenance.

Ce n'est pas

- un cours approfondi de Python,
- que du code.

Pré-requis

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

En pratique

- 1h45 de cours le lundi 10h45-12h30 en 23-24.203 (Maxime Sangnier) ;
- 3h30 de TME le lundi 16h-19h45 en 14-15.303 (Elina Thibeau-Sutre & Maxime Sangnier) ;
- slides et code sur **github** : <https://github.com/sangnier/Soccer>

Évaluation

- contrôle continu : 70%
 - participation (seul(e), tout le temps)
 - partiel écrit (seul(e), à mi-parcours)
 - rapport, soutenance et code (en binôme, à la fin)
- examen : 30%
 - examen sur machine (seul(e), à la fin).

Introduction au 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 (notions de plan expérimental, d'apprentissage statistique)
- bonus : apprentissage avancé

Organisation :

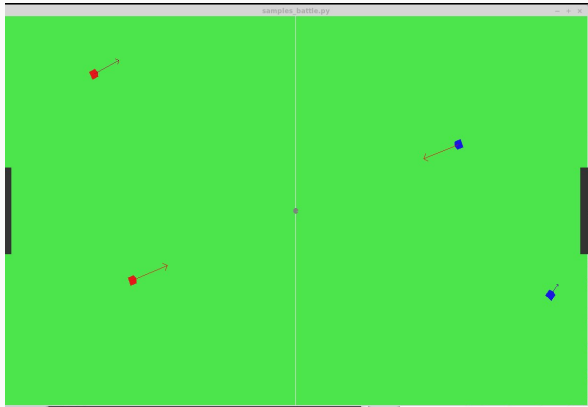
- a partir de la 2 ou 3ème semaine (selon l'avancement), chaque semaine une série de rencontres, tous les groupes rencontrent tous les groupes
- catégories : 1 contre 1, 2 contre 2 (et plus tard 4 contre 4)
- des challenges points d'étapes pour vérifier la qualité de vos joueurs (tirer un but, récupérer la balle, ...)

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.

Besoins

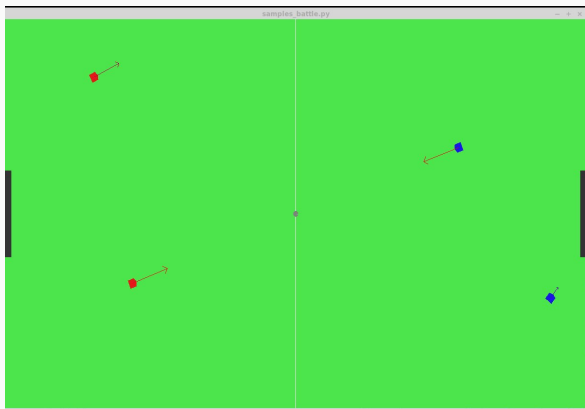
Concept de :



Besoins

Concept de :

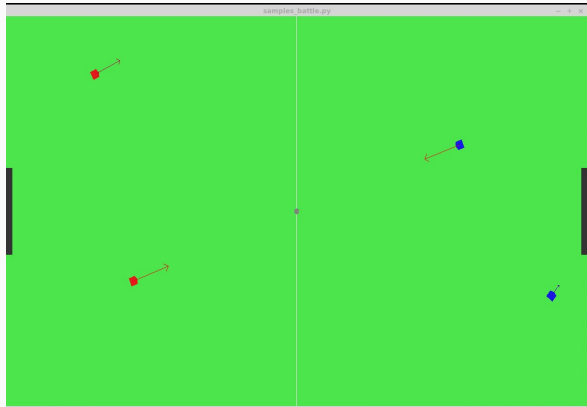
- terrain
- ballon
- joueur
- équipe
- tournoi
- c'est tout ?



Besoins

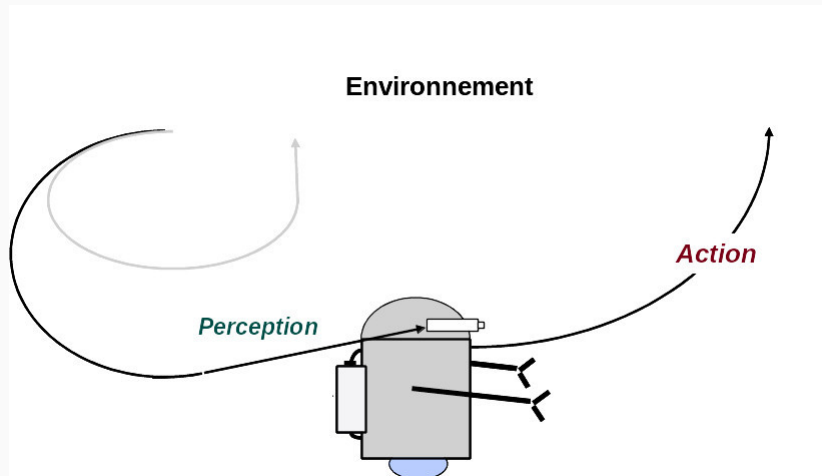
Concept de :

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



Un joueur est une coquille vide !

⇒ il faut lui donner les moyens de réagir



Principe

- Environnement
 - tout ce qui est extérieur à l'agent
- État
 - ce que perçoit l'agent
- Action
 - ce que peut décider l'agent

Exemples

- Jeu d'échecs
- Tetris
- Sudoku
- flappy bird
- ... et le foot.

- Environnement = plateforme de simulation
- Agent : le joueur
- Action :
 - Déplacement
 - Tir
- État
 - Position/vitesse des joueurs
 - Terrain
 - Position/vitesse de la balle

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`
- `PlayerState` : état d'un joueur (`vitesse`, `accélération`, `position`, `shoot`)
- `SoccerState` :
 - `player_state(self, id_team, id_player)` : renvoie l'état du joueur
 - `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`)
- `Strategy` : modèle de stratégie, toute stratégie doit implémenter la méthode `compute_strategy(...)`
- `SoccerTeam` : liste des joueurs
- `Simulation` : permet de lancer un bout de match entre deux équipes.
- `SoccerTournament` : tournoi.

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

Boucle d'action

- Pour step de 0 à MAX_STEP
 - calcul pour chaque joueur 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(Strategy):  
    def __init__(self):  
        Strategy.__init__(self, "Random")  
    def compute_strategy(self, state, id_team, id_player):  
        return SoccerAction(Vector2D.create_random(),  
                             Vector2D.create_random())
```

Lancer une partie

```
from soccersimulator import Strategy, SoccerAction, Vector2D
from soccersimulator import SoccerTeam, Simulation, Player, show_simu

class RandomStrategy(Strategy):
    def __init__(self):
        Strategy.__init__(self, "Random")
    def compute_strategy(self, state, id_team, id_player):
        return SoccerAction(Vector2D.create_random(),
                             Vector2D.create_random())

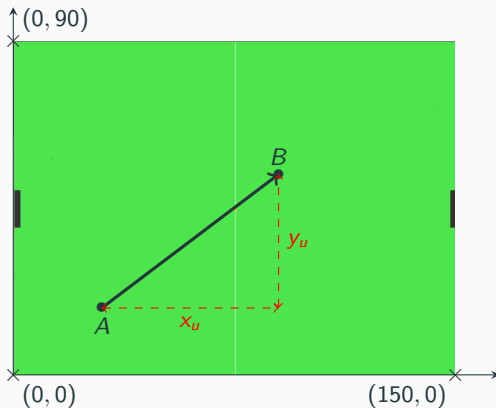
## Creation d'une equipe
pyteam = SoccerTeam("PyTeam")
## Ajout d'un joueur
pyteam.add("PyPlayer", RandomStrategy())
## Creation d'une deuxieme equipe et ajout d'un joueur (autre possibilite p
thon=SoccerTeam("ThonTeam", [Player("ThonPlayer", RandomStrategy())])

## Creation d'une simulation de 2000 pas de temps
match = Simulation(pyteam, thonteam, max_steps = 2000)
match.start() # ou
show_simu(match) # pour l'affichage graphique
```

Géométrie vectorielle

Géométrie 2D

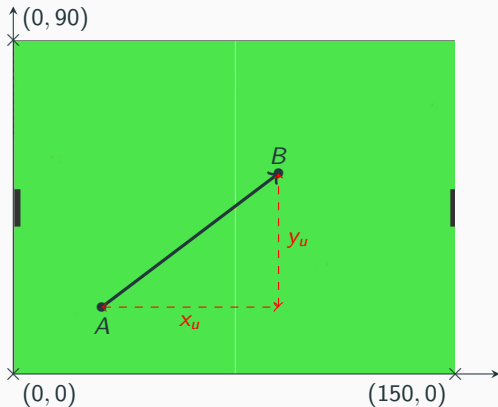
- Un point :
 $A : (x_A, y_A) \in \mathbb{R}^2$
- Un vecteur :
 $\mathbf{u} = (x_u, y_u) \in \mathbb{R}^2$
- Vecteur entre 2 points :
 $\overrightarrow{AB} = (x_B - x_A, y_B - y_A)$



Géométrie 2D

- Un point :
 $A : (x_A, y_A) \in \mathbb{R}^2$
- Un vecteur :
 $\mathbf{u} = (x_u, y_u) \in \mathbb{R}^2$
- Vecteur entre 2 points :
 $\overrightarrow{AB} = (x_B - x_A, y_B - y_A)$

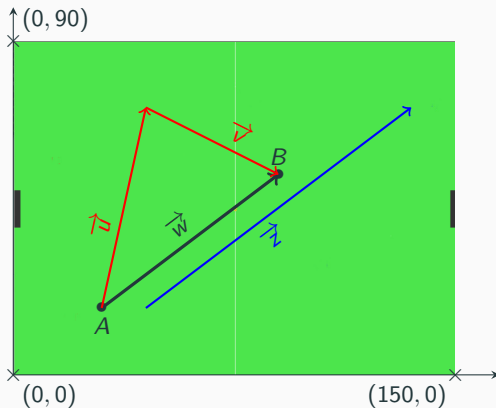
Un vecteur caractérise un déplacement, une vitesse, une accélération : une *norme* (puissance, force) et un *angle* (direction).



Opérations algébriques

$$\begin{aligned}\vec{w} &= \vec{u} + \vec{v} \\ \begin{pmatrix} x_w \\ y_w \end{pmatrix} &= \begin{pmatrix} x_u \\ y_u \end{pmatrix} + \begin{pmatrix} x_v \\ y_v \end{pmatrix} \\ &= \begin{pmatrix} x_u + x_v \\ y_u + y_v \end{pmatrix}\end{aligned}$$

$$\begin{aligned}\vec{z} &= a\vec{w} \\ &= a \begin{pmatrix} x_w \\ y_w \end{pmatrix} \\ &= \begin{pmatrix} ax_w \\ ay_w \end{pmatrix}\end{aligned}$$



Propriétés

$$\begin{aligned}\vec{u} \cdot \vec{v} &= x_u x_v + y_u y_v \\ &= \|\vec{u}\|_{\ell_2} \|\vec{v}\|_{\ell_2} \cos \theta\end{aligned}$$

$$(\vec{u} + \alpha \vec{v}) \cdot \vec{w} = \vec{u} \cdot \vec{w} + \alpha \vec{v} \cdot \vec{w}$$

$$\begin{aligned}\|\vec{u}\|_{\ell_2} &= \sqrt{\vec{u} \cdot \vec{u}} \\ &= \sqrt{x_u^2 + y_u^2}\end{aligned}$$

$$\|\alpha \vec{u}\|_{\ell_2} = \alpha \|\vec{u}\|_{\ell_2}$$

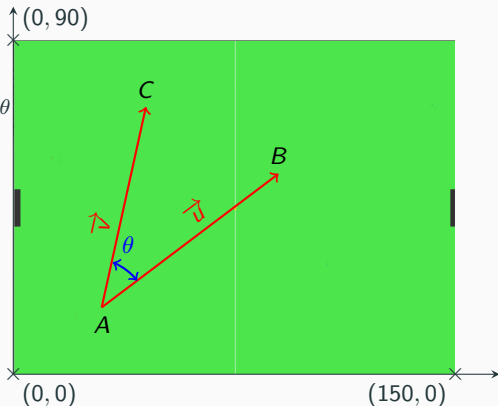
- \vec{u} et \vec{v} colinéaire

$$\Leftrightarrow \vec{u} = \alpha \vec{v}$$

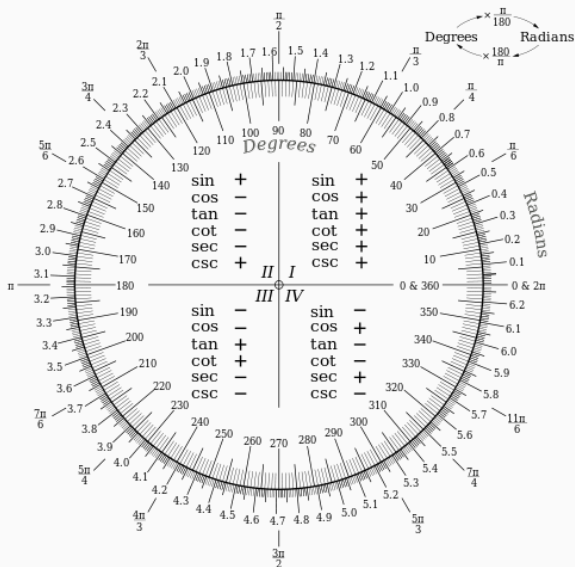
$$\Leftrightarrow \theta = 0, \vec{u} \cdot \vec{v} = \|\vec{u}\|_{\ell_2} \|\vec{v}\|_{\ell_2}$$

- \vec{u} orthogonal à \vec{v}

$$\Leftrightarrow \vec{u} \cdot \vec{v} = 0, \theta = \pm \pi/2$$



Les angles



Décomposition dans la base normale

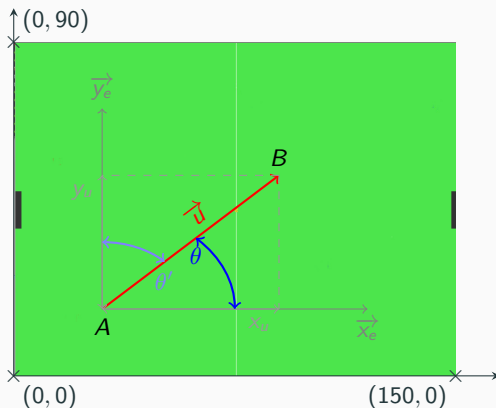
Coordonnées polaires

Rayon (norme) et angle à \mathbf{e}_x

$$\begin{aligned}\vec{u} \cdot \vec{e}_x &= x_u \\ &= \|\vec{u}\|_{\ell_2} \cos \theta \\ &= \|\vec{u}\|_{\ell_2} \sin \theta'\end{aligned}$$

$$\begin{aligned}\vec{u} \cdot \vec{e}_y &= y_u \\ &= \|\vec{u}\|_{\ell_2} \cos \theta' \\ &= \|\vec{u}\|_{\ell_2} \sin \theta\end{aligned}$$

cartésiennes	polaires
$\begin{pmatrix} x_u \\ y_u \end{pmatrix}$	$\begin{pmatrix} u_r = \ \vec{u}\ _{\ell_2} \\ u_\theta = \theta \end{pmatrix}$



Changement de base

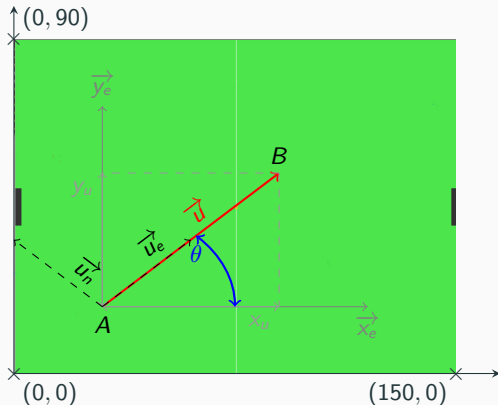
Engendré par un vecteur \vec{u}

Trouver \vec{u}_e et \vec{u}_n , de norme 1 :

- \vec{u}_e colinéaire à \vec{u}
- \vec{u}_n normal à \vec{u}

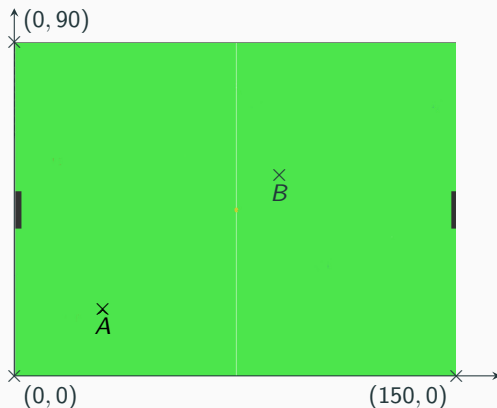
$$\begin{aligned}\vec{u}_e &= \frac{\vec{u}}{\|\vec{u}\|_{\ell_2}} \\ (\text{cart.}) &= \left(\frac{x_u}{\sqrt{x_u^2 + y_u^2}}, \frac{y_u}{\sqrt{x_u^2 + y_u^2}} \right) \\ (\text{polaires}) &= (1, \theta)\end{aligned}$$

$$\begin{aligned}\vec{u}_n &= \begin{pmatrix} x_u \cos \pi/2 - y_u \sin \pi/2 \\ x_u \sin \pi/2 + y_u \cos \pi/2 \end{pmatrix} \\ (\text{cart.}) &= \begin{pmatrix} -y_u \\ x_u \end{pmatrix} \\ (\text{polaires}) &= (1, \theta + \pi/2)\end{aligned}$$



Aller vers un point

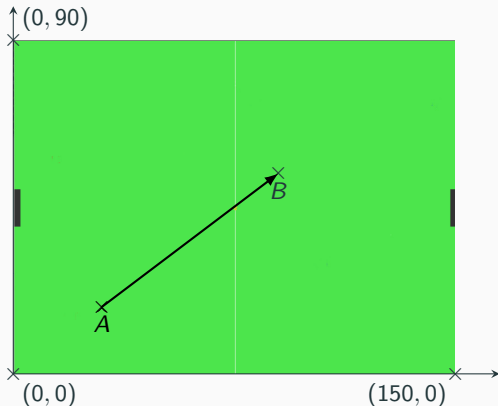
- A : position courante
- B : destination
- Comment quantifier le déplacement : $B = A + \vec{??}$?



Aller vers un point

- A : position courante
- B : destination
- Comment quantifier le déplacement : $B = A + \vec{?}$?
- Déplacement :

$$\begin{aligned}\vec{AB} &= \begin{pmatrix} x_{AB} \\ y_{AB} \end{pmatrix} \\ &= \begin{pmatrix} x_B \\ y_B \end{pmatrix} - \begin{pmatrix} x_A \\ y_A \end{pmatrix} \\ &= \begin{pmatrix} x_B - x_A \\ y_B - y_A \end{pmatrix}\end{aligned}$$



Objectifs du TME

Objectifs du TME

- prendre en main **git** ;
- créer un compte **github** ;
- installer la plateforme de simulation ;
- prendre en main Python et l'environnement ;
- programmer un joueur aléatoire ;
- programmation un joueur fonceur.

Dépôt git

`https://github.com/sangnier/Soccer`

Pour paramétrer le proxy

Ajouter dans `.bashrc` : `export https_proxy=proxy:3128`

Pour installer un module python manquant

`pip install [MODULE] --user`

Pour installer un module python stocké dans le repertoire courant

`pip install . -e --user`