Projet Foot
2I013

Nicolas Baskiotis
nicolas.baskiotis@lip6.fr
http://webia.lip6.fr/~baskiotisn
http://github.com/baskiotisn/SoccerSimulator

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

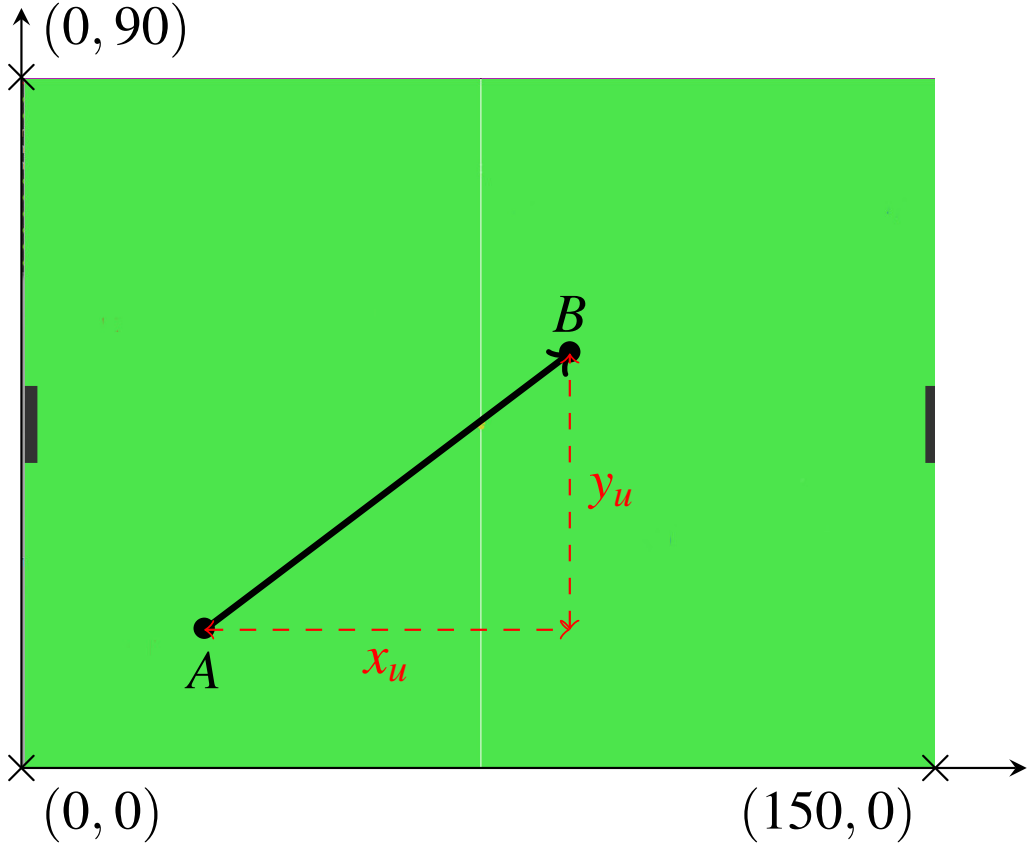
S2 (2016-2017)

Plan

Quelques rappels

Géométrie 2D

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

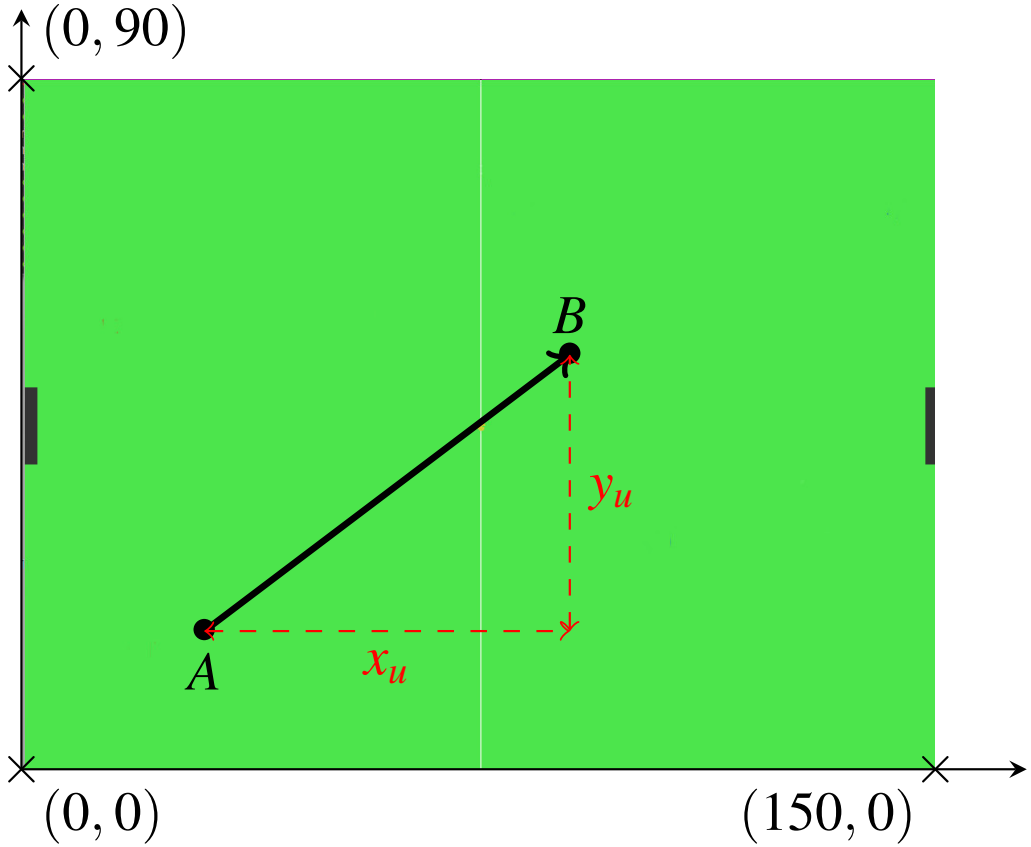


Quelques rappels

Géométrie 2D

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

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

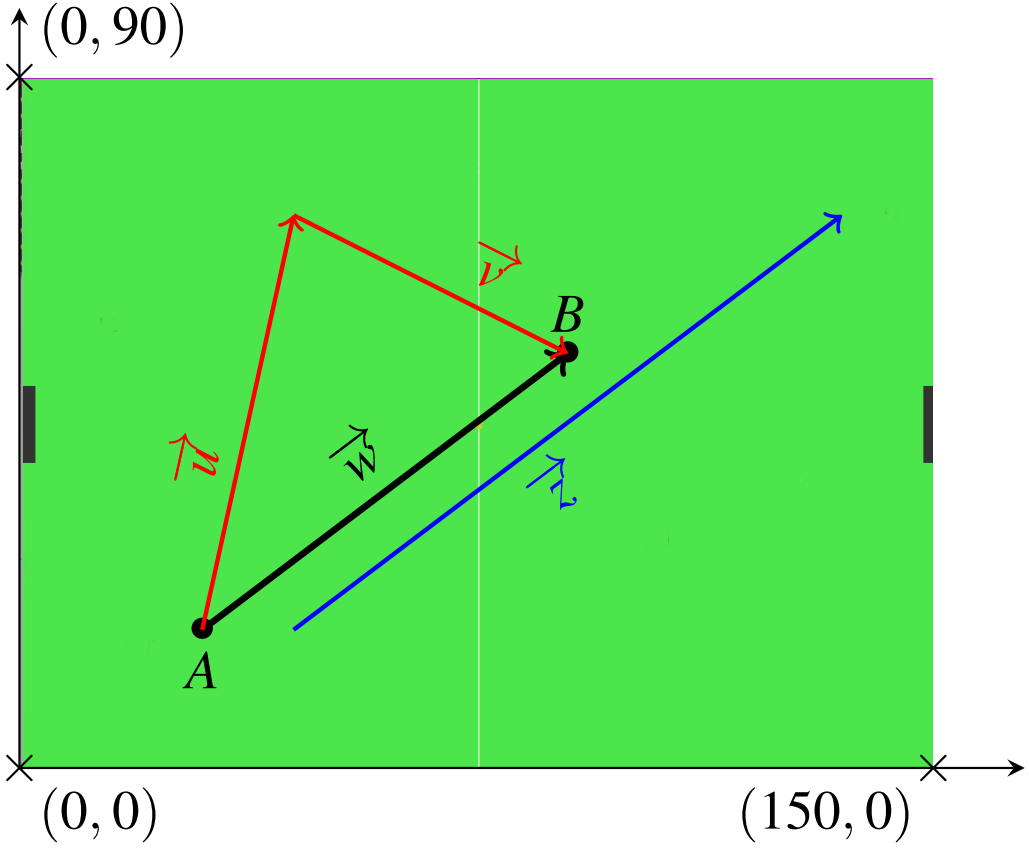


Quelques rappels

Opérations algébriques

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

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



Produit scalaire

Propriétés

$$\begin{aligned}
\vec{u}.\vec{v} &= x_u x_v + y_u y_v \\
&= \|\vec{u}\| \, \|\vec{v}\| \cos\theta \\
(\vec{u} + \alpha\vec{v}).\vec{w} &= \vec{u}.\vec{w} + \alpha\,\vec{v}.\vec{w}
\end{aligned}$$

$$\begin{aligned}
\|\vec{u}\| &= \sqrt{\vec{u}.\vec{u}} \\
&= \sqrt{x_u^2 + y_u^2} \\
\|\overrightarrow{\alpha u}\| &= \alpha \|\vec{u}\|
\end{aligned}$$

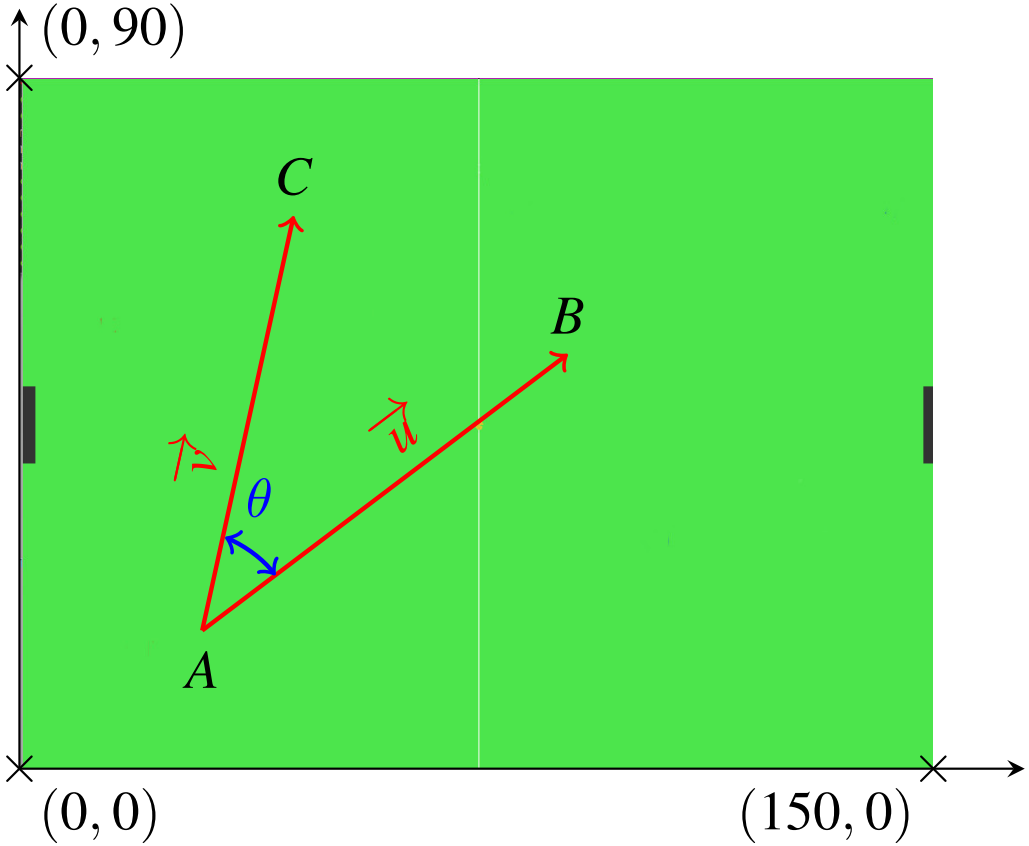
- $\vec{u}$ et $\vec{v}$ colinéaire

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

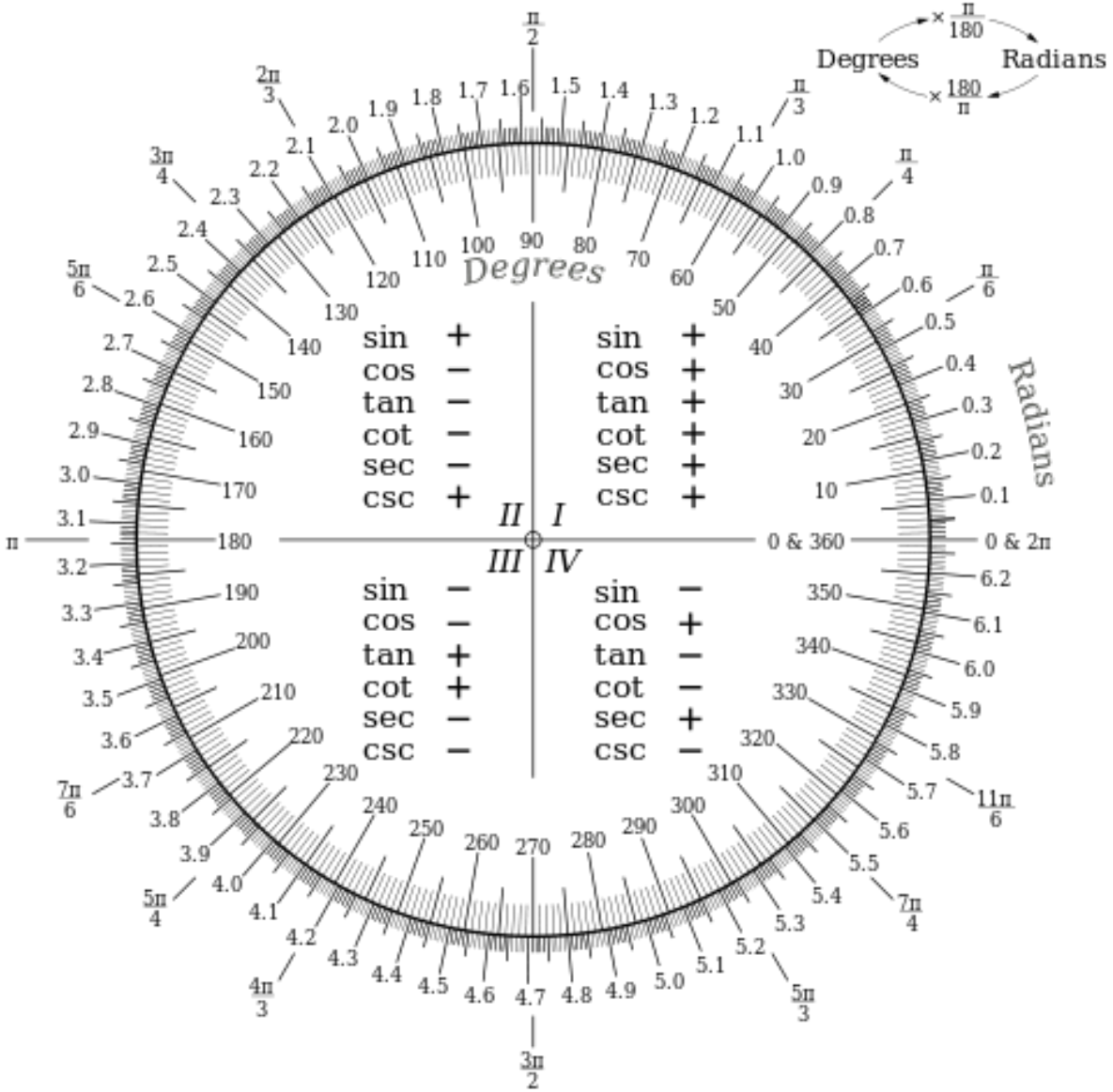
$\Leftrightarrow$ $\theta = 0,\ \vec{u}.\vec{v} = \|u\|\,\|v\|$

- $\vec{u}$ orthogonal à $\vec{v}$

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



Les angles


Degrees
Radians
× π/180
× 180/π
Degrees
Radians
II
I
III
IV
sin + cos − tan − cot − sec − csc +
sin + cos + tan + cot + sec + csc +
sin − cos − tan + cot + sec − csc −
sin − cos + tan − cot − sec + csc −
0 & 360
0 & 2π
π
π/2
π/3
π/4
π/6
2π/3
3π/4
5π/6
7π/6
5π/4
4π/3
3π/2
5π/3
7π/4
11π/6

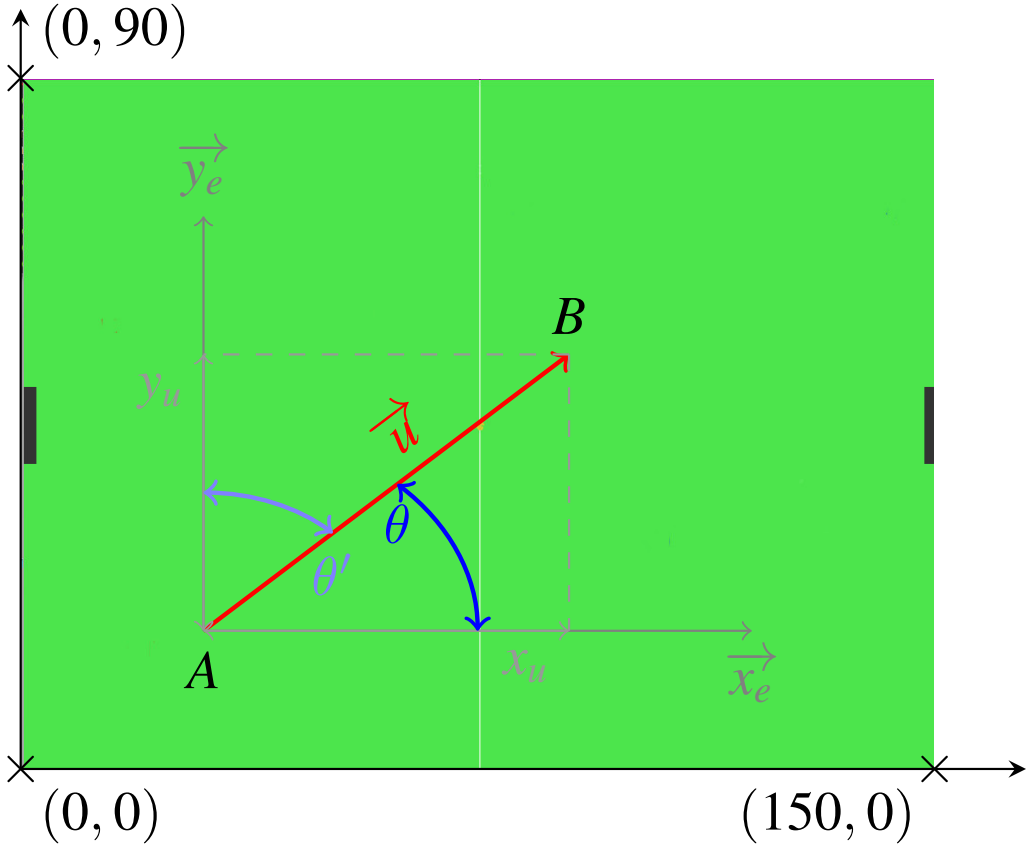
Décomposition dans la base normale

Coordonnées polaires

Rayon (norme) et angle à e_x

$$\begin{aligned}
\vec{u} \,.\, \vec{e_x} &= x_u \\
&= \| \vec{u} \| \cos\theta \\
&= \| \vec{u} \| \sin\theta' \\
\vec{u} \,.\, \vec{e_y} &= y_u \\
&= \| \vec{u} \| \cos\theta' \\
&= \| \vec{u} \| \sin\theta
\end{aligned}$$

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



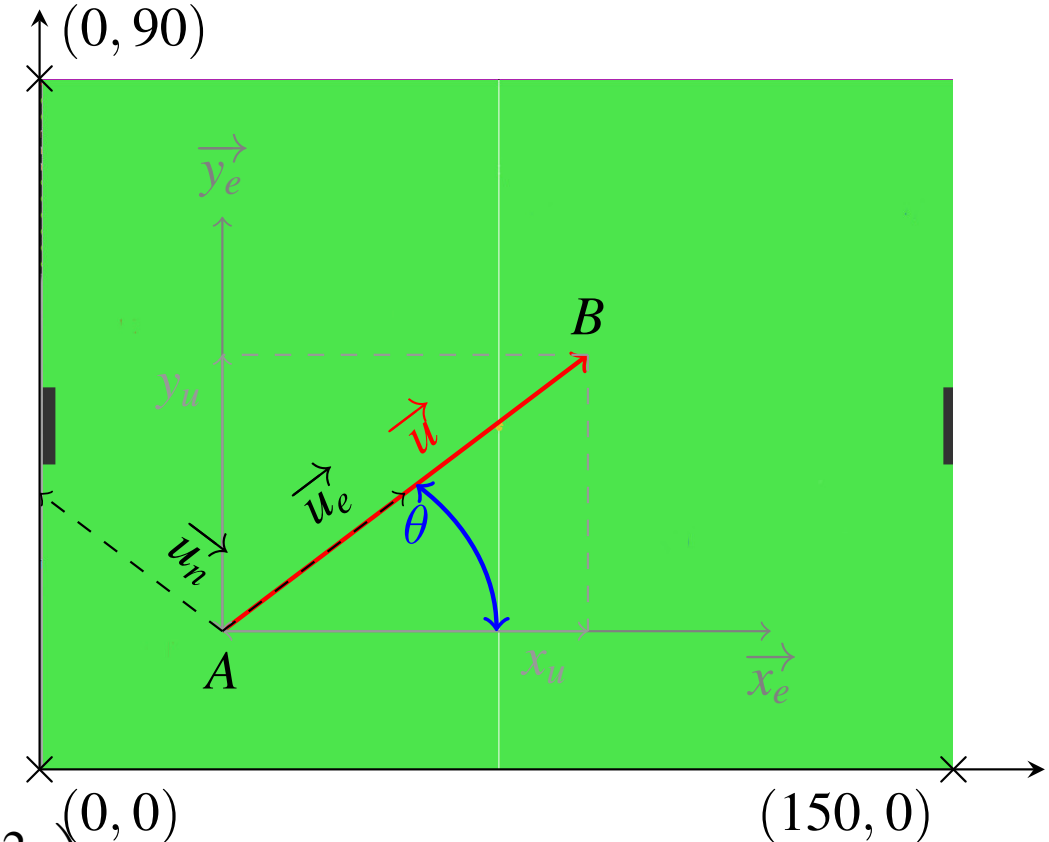
Changement de base

Engendré par un vecteur $\overrightarrow{u}$

Trouver $\overrightarrow{u_e}$ et $\overrightarrow{u_n}$, de norme 1 :

- $\overrightarrow{u_e}$ colinéaire à $\overrightarrow{u}$
- $\overrightarrow{u_n}$ normal à $\overrightarrow{u}$

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



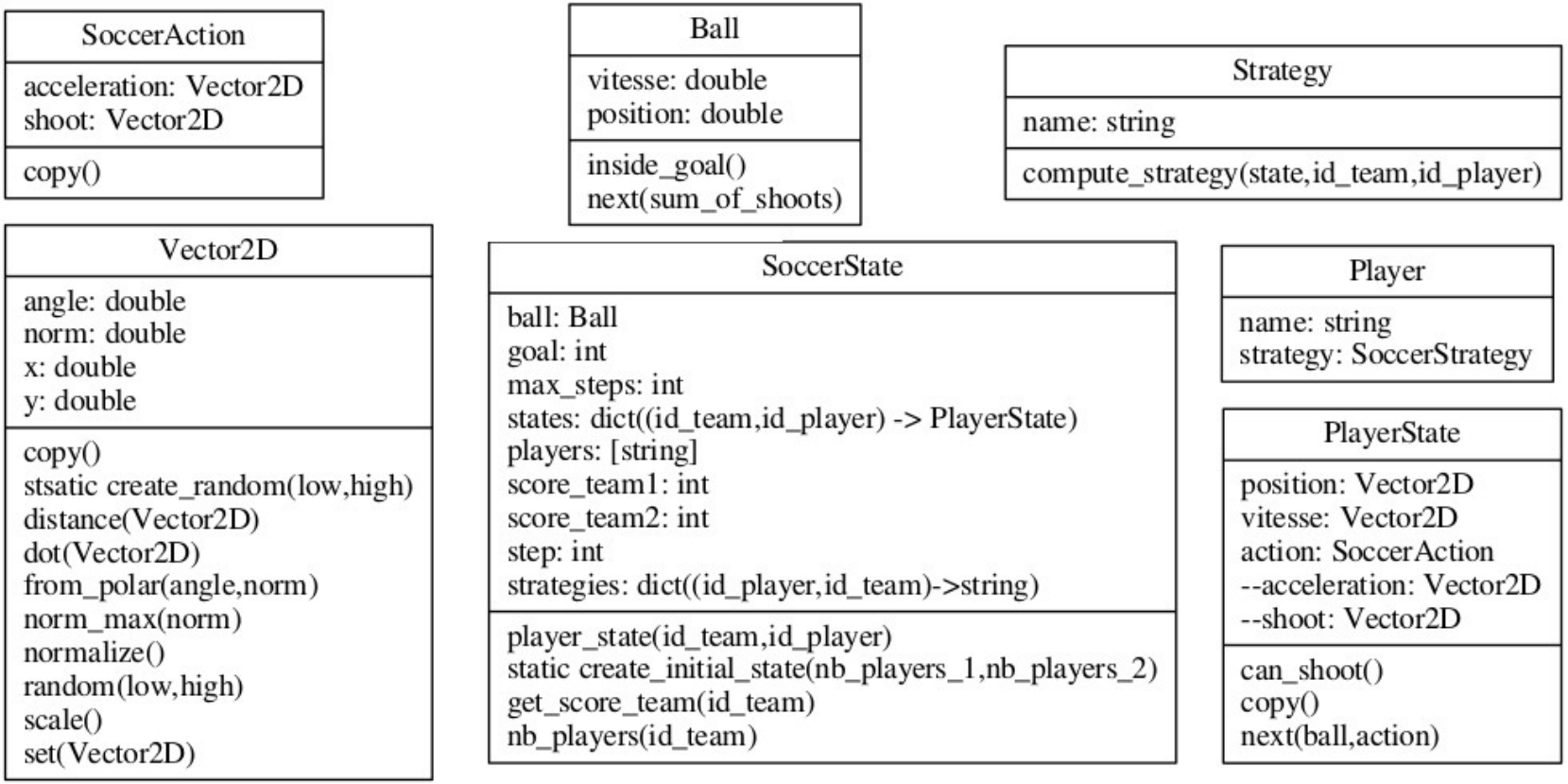
$$\begin{aligned} \overrightarrow{u_n} &= \begin{pmatrix} x_u \cos \pi/2 - y_u \sin \pi/2 \\ x_u \sin \pi/2 + y_u \cos \pi/2 \end{pmatrix} \\ \text{(cart.)} &= \begin{pmatrix} -y_u \\ x_u \end{pmatrix} \\ \text{(polaires)} &= (1, \theta + \pi/2) \end{aligned}$$

Plan

Les objets en présence (et leurs attributs)

- `Vector2D` : représente un point ou un vecteur;
- `MobileMixin` : représente un objet mobile (position, vitesse);
- `SoccerAction` : représente l'action d'un joueur (accélération, shoot);
- `Player` : représente un joueur (nom, stratégie);
- `Strategy` : représente une stratégie;
- `PlayerState` :représente un état d'un joueur (position, vitesse)
- `SoccerState` :représente un état du jeu (balle, joueurs, score)
- `SoccerTeam` : liste de joueurs et de leur stratégie
- `Simulation` : une simulation de match

Les objets en présence (et leurs attributs)

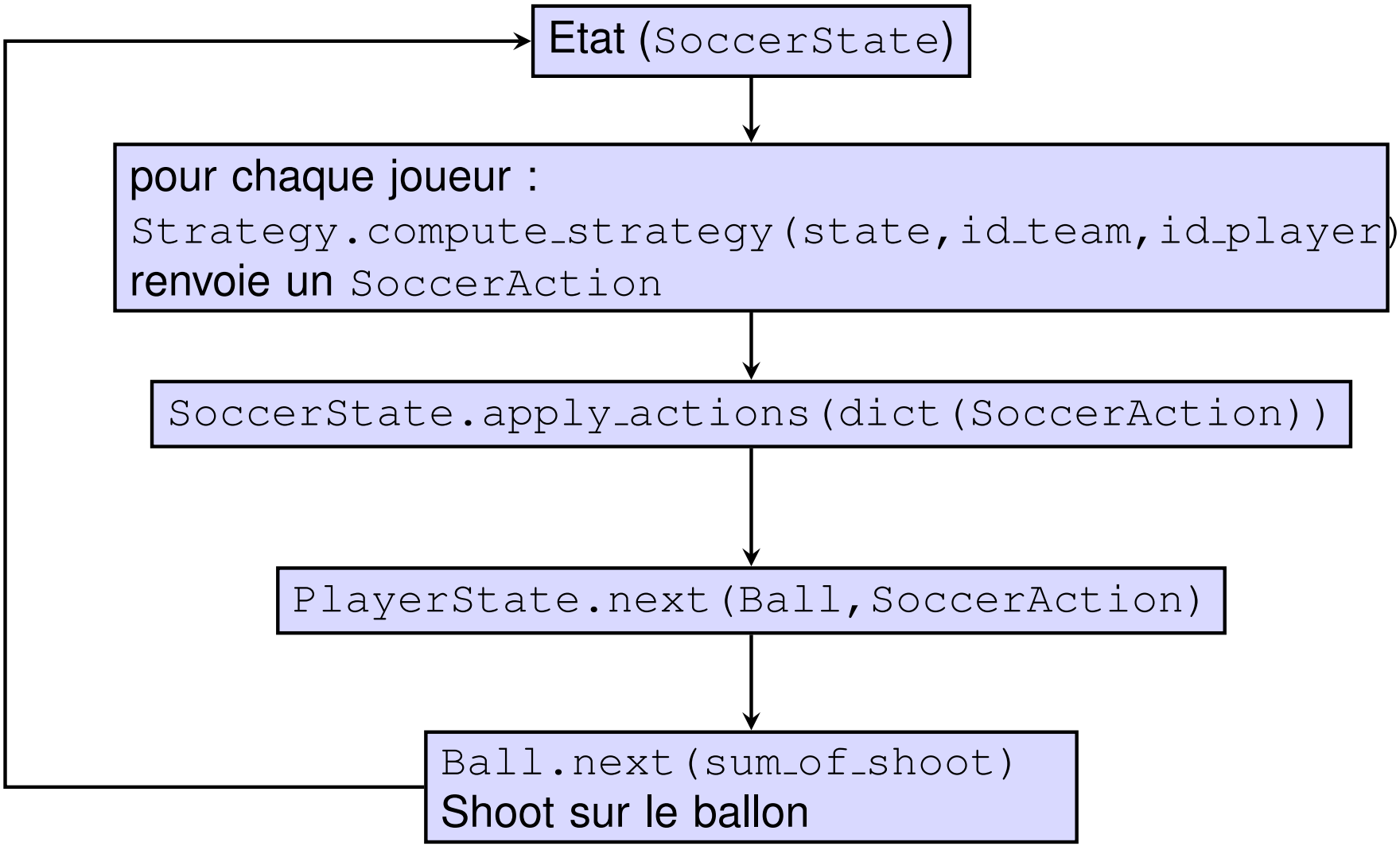


Les commandes utiles

Etant donné un état `state`

- `state.ball` : MobileMixin de la balle
- `state.ball.position` : la position de la balle (`state.ball.position.x, state.ball.position.y`)
- `state.ball.vitesse` : la vitesse de la balle
- `state.player_(idteam,idplayer)` : la configuration d'un joueur
- `state.player_state(idteam,idplayer)` : MobileMixin du joueur
- `state.player_state(idteam,idplayer).position` : position du joueur
- `state.player_state(idteam,idplayer).vitesse` : vitesse du joueur
- `state.players` : liste des clés `(idteam,idplayer)` de tous les joueurs

Le cœur du simulateur



Le moteur d'un joueur

```
def next(self, ball, action=None):
  if not (hasattr(action,"acceleration") and hasattr(action,"shoot")):
    action = SoccerAction()
    self.action = action.copy()
    self.vitesse *= (1 - settings.playerBrackConstant)
    self.vitesse = (self.vitesse +\
        self.acceleration).norm_max(settings.maxPlayerSpeed)
    self.position += self.vitesse
    if self.position.x < 0 or self.position.x > settings.GAME_WIDTH \
      or self.position.y < 0 \
      or self.position.y > settings.GAME_HEIGHT:
    self.position.x=max(0,min(settings.GAME_WIDTH,self.position.x))
    self.position.y=max(0,min(settings.GAME_HEIGHT,self.position.y))
    self.vitesse = Vector2D()
    if self.shoot.norm == 0 or not self.can_shoot():
      self._dec_shoot()
      return Vector2D()
    self._reset_shoot()
    if self.position.distance(ball.position)\
        >(settings.PLAYER_RADIUS+settings.BALL_RADIUS):
        return Vector2D()
    return self._rd_angle(self.shoot,(self.vitesse.angle-self.shoot.an
        self.position.distance(ball.position)/(settings.PLAYER_RADIUS+
```

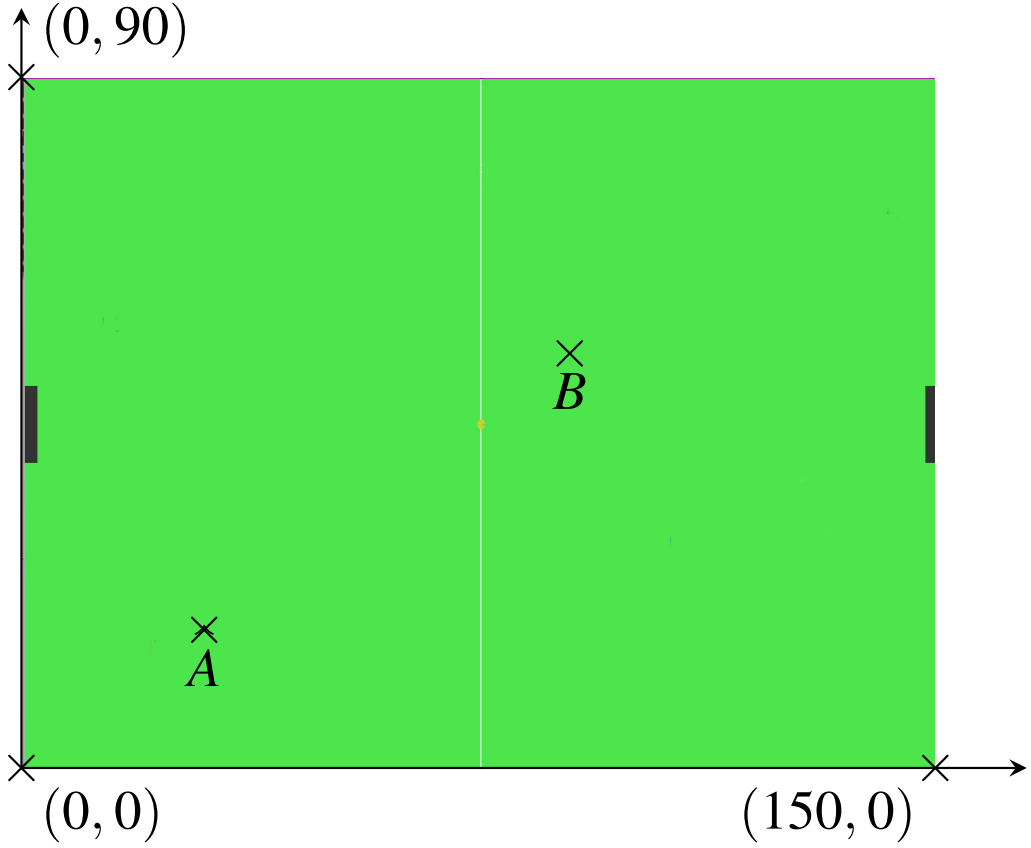
Le moteur du ballon

```
def next(self,sum_of_shoots):
  vitesse = self.vitesse.copy()
  vitesse.norm = self.vitesse.norm -\
                 settings.ballBrakeSquare*\
                 self.vitesse.norm**2-\
                 settings.ballBrakeConstant*self.vitesse.norm
  ## decomposition selon le vecteur unitaire de ball.speed
  snorm = sum_of_shoots.norm
  if snorm > 0:
    u_s = sum_of_shoots.copy()
    u_s.normalize()
    u_t = Vector2D(-u_s.y, u_s.x)
    speed_abs = abs(vitesse.dot(u_s))
    speed_ortho = vitesse.dot(u_t)
    speed_tmp = Vector2D(speed_abs * u_s.x\
       - speed_ortho * u_s.y,\
       speed_abs * u_s.y + speed_ortho * u_s.x)
    speed_tmp += sum_of_shoots
    vitesse = speed_tmp
  self.vitesse = vitesse.norm_max(settings.maxBallAcceleration)
  self.position += self.vitesse
```

Plan

Aller vers un point ?

- A position courante
- P : destination
- Quelle action ?

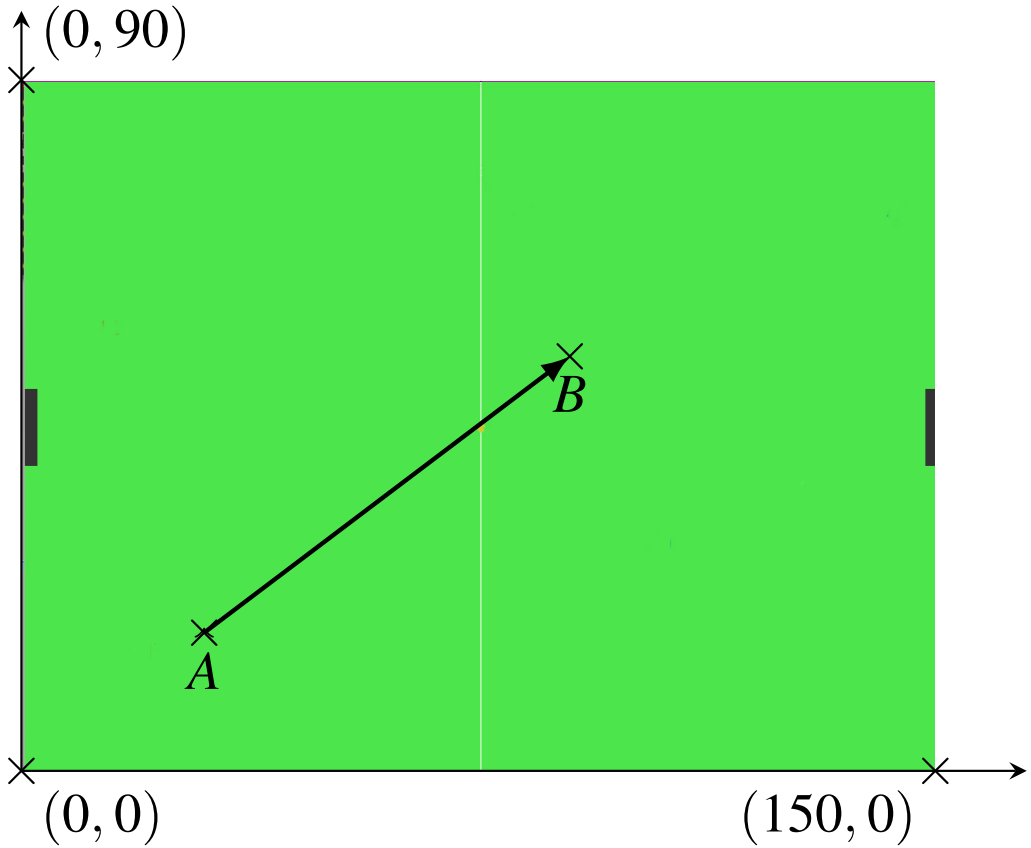


Aller vers un point ?

- A position courante
- P : destination
- Quelle action ?
- Vecteur vitesse:

$$\vec{v} = \overrightarrow{AB}$$

- Importance de la norme ?




```
def compute_strategy(self, state, id_team, id_player):
    return SoccerAction(state.ball.position
                -state.player_state(id_team,id_player).position,
                Vector2D())
```

Proposition de stratégie

Stratégie naïve

- Fonceur
- Goal
- ...

Proposition de stratégie

Stratégie naïve

- Fonceur
- Goal
- ...
- Comment choisir entre les différentes stratégies ?
- Comment le faire de manière élégante ?

$\Rightarrow$ Coder des petites fonctions légères et génériques !

Création d'une Toolbox

Dans un fichier séparé (par exemple `tools.py`

Inclure les fonctions usuelles pour :

- aller vers un point
- shooter vers le but
- trouver l'adversaire le plus proche
- ... (toutes les petites fonctions récurrentes dont vous aurez besoin)

Puis dans votre fichier

```
from tools import *
...
```

Réfléchissez à la structure de vos fonctions :

- Elles doivent être générique (situation miroir selon l'identifiant de l'équipe)
- Facile à manier.
- Possibilité d'encapsuler l'objet `SoccerState`.

Encapsuler un objet

Il s'agit

- d'enrichir un objet de nouvelles fonctionnalités;
- de *traduire* certaines de ses propriétés (par exemple objet miroir)
- d'en faciliter l'utilisation.

Exemple

```
class MyState(object):
  def __init__(self,state,idteam,idplayer):
      self.state = state
      self.key = (idteam,idplayer)
  def my_position(self):
      return self.state.player_state(*pkey).position
      #equivalent a self.player_state(self.key[0],self.key[1])
 def ball_position(self):
     return self.state.ball.position
 def aller(self,p):
     return SoccerAction(p-self.my_position(),Vector2D())
 def shoot(self,p):
     return SoccerAction(Vector2D(),p-self.state.my_position())
 def compute_strategy(self):
     return self.aller(p)=self.shoot(p)
```

Encapsuler un objet

Il s'agit

- d'enrichir un objet de nouvelles fonctionnalités;
- de *traduire* certaines de ses propriétés (par exemple objet miroir)
- d'en faciliter l'utilisation.

Exemple

```
class MyStrategy(Strategy):
      def __init__(self):
           Strategy.__init__(self,"Ma strat")
      def compute_strategy(self,state,idteamn,idplayer):
          return MyState(state,idteam,idplayer).compute_strategy()
#ou
class SousStrat(Strategy):
    def __init__(self,sous_strat):
        Strategy.__init__(self,sous_strat.name)
        self.strat=sous_strat
    def compute_strategy(self,state,idteam,idplayer):
        return self.strat(MyState(state,idteam,idplayer))

def fonceur(me):
    return me.aller(me.ball_position)+me.shoot(me.but_adv)
```