2I013 Groupe 1

Projet Foot

Maxime Sangnier – Nicolas Baskiotis
maxime.sangnier@lip6.fr

2019

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

Outline

Séance précédente

Bilan du TME

Vous avez :

- organisé votre code à l'aide d'un **module** et de scripts ;
- décomposé vos stratégies de manière à les rendre génériques (si vous décidez à un moment de changer votre façon de courir, il ne faut rien toucher à la description ou aux stratégies) ;
- créé le fichier `tournament.py`
- affiné les joueurs d'attaque et de défense (positionnement, dribble...).

Fichier `tournament.py`

```
def get_team(nb_players):
    team = SoccerTeam(name="Maxime's Team")
    if nb_players == 1:
        team.add("Striker", StrikeStrategy())
    ...
    return team
if __name__ == '__main__':
    team1 = get_team(1)
    team2 = get_team(2)

    simu = Simulation(team1, team2)
    show_simu(simu)
```

Tournoi 1v1

Équipe	Points	Matchs (gagnés, nuls, perdus)	Buts (marqués, encaissés)
Nasser's Team (NasserChaker)	18	(6,0,0)	(37,3)
Nicolas / Téo (Bari0th)	9	(3,0,3)	(48,18)
Maxime's Team (GuillaumedeCumont)	9	(3,0,3)	(29,39)
equipe 3701195 (ECentaure)	0	(0,0,6)	(0,54)

Table 1 – Tournoi 1v1.

Équipes non-qualifées

Toutes celles qui n'apparaissent pas.

Création des équipes

Étudiants non-inscrits dans un groupe

(3602824, "D'HERVAIS RIWAN"),
(3520461, 'FIOCRE SIMON'),
(3804546, 'GUELIANE BELKACEM'),
(3412857, 'HASSANALY AHIL'),
(3520237, 'KLINGSHEIM NICOLAS'),
(3700379, 'SUBASI ROZA')

Sur github

- choisissez un compte **github** ;
- partagez votre dépôt avec l'autre ;
- renseignez le formulaire : `https://goo.gl/forms/kLOdERD8LOpEpGwu1`

Motifs de conception

Motifs de conception

Someone has already solved your problems

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (C. Alexander)

Pourquoi ?

- Solutions propres, cohérentes et saines
- Langage commun entre programmeurs
- C'est pas seulement un nom, mais une caractérisation du problème, des contraintes,. . .
- Pas du code/solution pratique, mais une solution générique à un problème de design.

Un très bon livre :

Head First Motifs de conception, E. Freeman, E. Freeman, K. Sierra, B. Bates, Oreilly

Motifs de conception

Quelques Principes

- Surtout pour les langages fortement typés, structurés (**Java** par exemple)
- Identifier les aspects de votre programme qui peuvent varier/évoluer et les séparer de ce qui reste identique
- Penser de manière générique et non pas en termes d'implémentations
- Composer plutôt qu'hériter (plus flexible) !

En avez-vous déjà vu ?

Motifs de conception

Quelques Principes

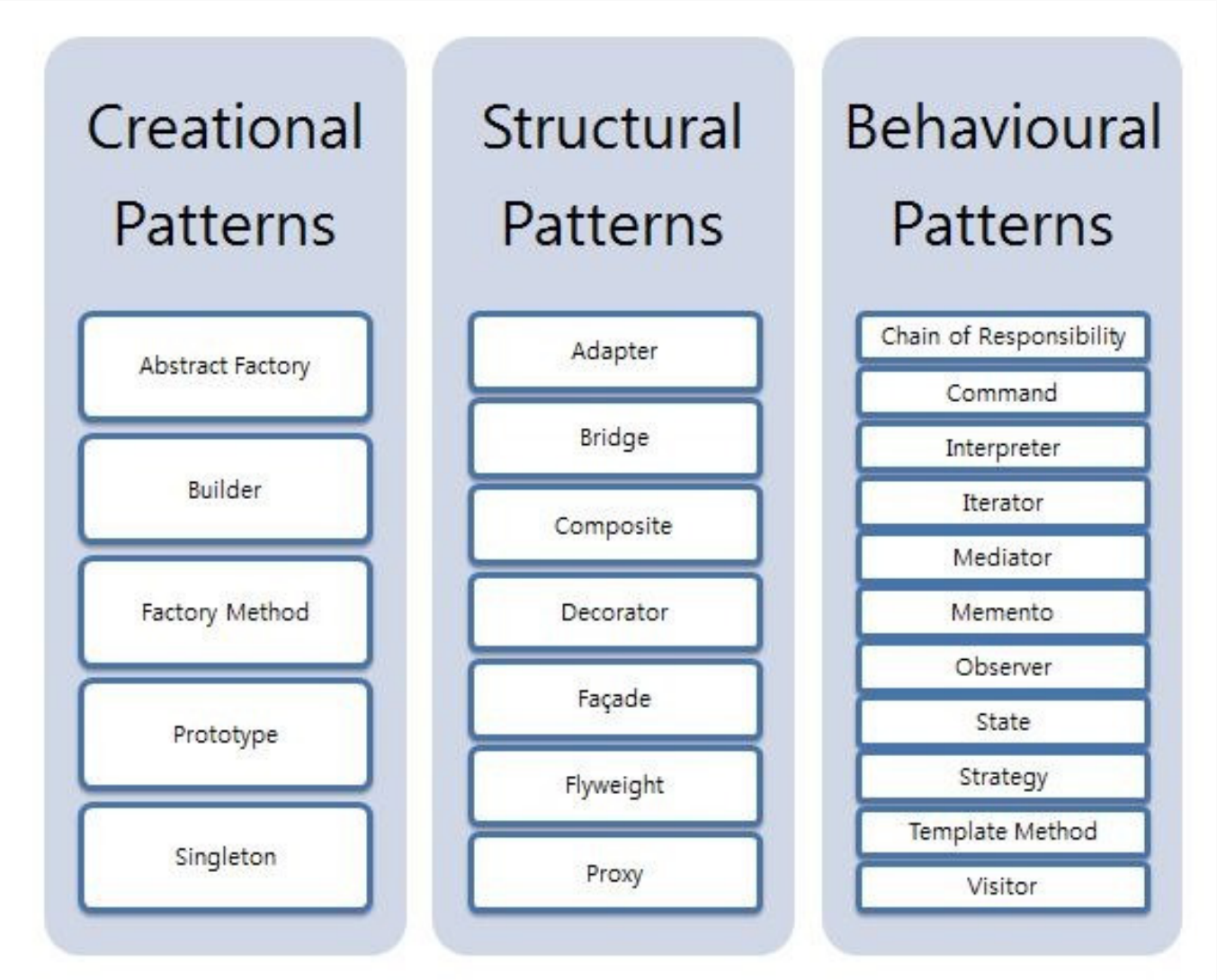
- Surtout pour les langages fortement typés, structurés (**Java** par exemple)
- Identifier les aspects de votre programme qui peuvent varier/évoluer et les séparer de ce qui reste identique
- Penser de manière générique et non pas en termes d'implémentations
- Composer plutôt qu'hériter (plus flexible) !

En avez-vous déjà vu ?

3 grandes classes

- *Creational :* Comment créer des objets
- *Structural :* Comment interconnecter des objets
- *Behavioral :* Comment faire une opération donnée

Une liste non exhaustive



Motifs de création

En Python, il n'y en a pas vraiment (sauf le singleton). Pour créer un objet d'une certaine manière, il suffit de faire une fonction.

```
def get_random_vec(x,y):
    return Vector2D.create_random(x,y)
def from_polar(x,y):
    return Vector2D.from_polar(0,2)
def from_cartesien(x,y):
    return Vector2D(x,y)
def get_null():
    return Vector2D()
  ...
```

Quelques caractéristiques de Python

Dans un objet :

- `def __init__(self,*args,**kwargs)`
  - `args` : arguments non nommés (`args[0]`)
  - `kwargs` : arguments nommés (`kwargs[''nom'']`)
- `__getattr__(self,name)` : appelé quand `name` n'est pas trouvé dans l'objet
- `__getattribute__(self,name)` : appelé pour toute rercherche de `name`

- Propriété : pour interroger de manière dynamique

```
class MyClass:
  @property
  def name(self): return ...
...
a = MyClass()
a.name # plutot que a.name()
```

En Python, pas d'erreur de typage, uniquement à l'exécution !

Python : Duck Typing

If it looks like a duck and quacks like a duck, it's a duck !

Typage dynamique

- La sémantique de l'objet (son type) est déterminée par l'ensemble de ses méthodes et attributs, dans un contexte donné
- Contrairement au typage nominatif où la sémantique est définie explicitement.

Concrétement

```
Class Duck:
  def quack(self):
     print("Quack")
Class Personne:
  def parler(self):
    print("Je␣parle")
donald  = Duck()
moi = Personne()
autre = "un␣canard"
try:
  donald.quack()
  moi.quack()
  autre.quack()
except AttributeError:
    print("ce␣n'est␣pas␣un␣canard"
```

Motifs structuraux : Adapter (et si je veux que ce soit un canard ?)

- Il suffit d'y ajouter une méthode qui le fait se comporter comme un canard.
- Toutes les autres méthodes doivent être disponibles !

```
class PersonneAdapter:
    def __init__(self,obj):
        self._obj = obj
    def __getattr__(self,attr):
        if attr == "quack":
            return self.parler()
        return attr(self._obj,attr)

moi = DuckAdapter(Personne())
moi.duck()
```

Motifs structuraux : Decorator (très similaire à Proxy et Adaptor)

Comment ajouter des fonctionnalités de manière dynamique à un objet

Exemple : tirer au but

```
class Decorator:
      def __init__(self,state):
         self.state = state
      def __getattr__(self,attr):
         return getattr(self.state,attr)
class Shoot(Decorator):
      def __init__(self,state):
          Decorator.__init__(self,state)
      def shoot(self,p):
         return SoccerAction(Vector2D(...))
class Passe(Decorator):
      def __init__(self,state):
          Decorator.__init__(self,state)
      def passe(self,p):
         return SoccerAction(Vector2D(...))

mystate = Shoot(Passe(state))
```

Motifs structuraux : Decorator (peut changer le comportement d'une fonction)

Exemple : modifier la passe

```
  class MeilleurPasse(Decorator):
     def petite_passe(self,p):
         return SoccerAction(...)
     def passe(self,p):
        if (condition):
           return self.petite_passe(p)
        return self.state.passe(p)
mystate = MeilleurPasse(Passe(state))
```

Motifs de comportement : Iterator

Pouvoir parcourir une liste d'éléments sans connaître l'organisation interne des éléments

Un itérateur est un objet qui dispose

- d'une méthode `__iter__(self)` qui renvoie l'itérateur
- d'une méthode `next(self)` qui renvoie la prochaine valeur ou lève une exception `StopIteration`

Un itérateur peut être renvoyé par une fonction grâce à `yield`.

```
class Counter:
    def __init__(self,low,high):
        self.current = low
        self.high = high
    def __iter__(self):
        return self
    def next(self):
        if self.current > self.high:
            raise StopIteration
        else:
            self.current+=1
            return self.current-1
for c in Counter(3,8):
    print(c)
```

```
def counter(low,high):
    current = low
    while current <= high:
        yield current
        current += 1

for c in counter(3,8):
    print(c)
```

Motifs de comportement : Chain of responsability

Chaque bout de code ne doit faire qu'une et une seule chose

Quand beaucoup d'actions complexes doivent être appliquées, il vaut mieux multiplier des petites fonctions en charge de chaque action que faire une unique grosse fonction.

```
class ContentFilter(object):
  def __init__(self, filters=None):
        self._filters = list()
        if filters is not None:
            self._filters += filters

  def filter(self, content):
        for filter in self._filters:
            content = filter(content)
        return content
filter = ContentFilter([offensive_filter, ads_filter, video_filter])
filtered_content = filter.filter(content)
```

Motifs de comportement : State (ou Proxy dans la version simple)

Changer le comportement d'une fonction en fonction de l'état interne du système.

Proxy quand il n'y a pas d'état interne.

```
class Implem1:
 def f(self):
   print("Je␣suis␣f")
 def g(self):
   print("Je␣suis␣g")
 def h(self):
   print("Je␣suis␣h")

   class Implem2:
 def f(self):
   print("Je␣suis␣toujours␣f.")
 def g(self):
   print("Je␣suis␣toujours␣g.")
 def h(self):
   print("Je␣suis␣toujours␣h.")
```

```
class State_d:
 def __init__(self, imp):
   self._implem = imp
 def changeImp(self, newImp):
   self._implem = newImp
 def __getattr__(self, name):
   return getattr(self._implem, name)


def run(b):
  b.f()
  b.g()
  b.h()
b = State_d(Implem1())
run(b)
b.changeImp(Implem2())
run(b)
```

Motifs de comportement : Strategy

Le motif Strategy définie une famille d'algorithmes, les encapsule et les rend interchangeables. Il permet de faire varier l'algorithme de manière dynamique et indépendante :

- Lorsqu'on a besoin de différentes variantes d'un algorithme.
- Lorsqu'on définie beaucoup de comportements à utiliser selon certaines situations

```
class StrategyExample:
      def __init__(self,func):
         self.compute_strategy = func
       @property
       def name(self):
            if hasattr(self.func,"name"):
                return self.func.name
            return self.func.__name__
def passe(state,id_team,id_player):
    return fait_une_passe()
def cours(state,id_team,id_player):
    return cours_versr()

stratCours = StrategyExample(cours)
stratCours.name = "cours"
stratPasse = StrategyExample(passe)
stratPasse.name = "passe"
```

Motifs de comportement : Observer

Simulation

- Une simulation possède une liste d'observateurs (`listeners`).
- A chaque événement marquant, tous les observateurs sont avertis par l'appel d'une fonction.
- Il est possible d'ajouter à la volée ou de supprimer des observateurs de la simulation.

Actions déclenchées lors d'une simulation

- `begin_match(team1,team2,state)` : au début de la simulation
- `end_match(team1,team2,state)` : à la fin
- `begin_round(team1,team2,state)` : au début de chaque engagement
- `end_round(team1,team2,state)` : à chaque but marqué
- `update_round(team1,team2,state)` : à chaque fin de tour

Optimisation d'une stratégie

Optimisation d'une stratégie

Quelle stratégie ?

- Shoot.
- Défense.
- Goal.
- Rejoindre la balle.
- Etc.

Optimisation d'une stratégie

Quelle stratégie ?

- Shoot.
- Défense.
- Goal.
- Rejoindre la balle.
- Etc.

Dépendent de degrés de liberté :

- puissance de tir ;
- direction ;
- déclenchement de l'action ;
- position de repos ;
- vitesse ;
- etc.

Les degrés de liberté

Comment fixer ces degrés de liberté ?

- Connaître le fonctionnement exact du simulateur.
- Savoir par expérience.
- Etc. . .

Les degrés de liberté

Comment fixer ces degrés de liberté ?

- Connaître le fonctionnement exact du simulateur.
- Savoir par expérience.
- Etc. . .

Ce n'est pas suffisant ! Cela ne prend pas en compte l'aléa.

Les degrés de liberté

Comment fixer ces degrés de liberté ?

- Connaître le fonctionnement exact du simulateur.
- Savoir par expérience.
- Etc. . .

Ce n'est pas suffisant ! Cela ne prend pas en compte l'aléa.

Nécessité d'un choix heuristique, valable en moyenne.

Plan d'expérience

Définir :

1. une action à optimiser ;
2. un modèle ;
3. les paramètres du modèle à optimiser ;
4. un critère ;
5. les conditions environnementales.

Plan d'expérience

Définir :

1. une action à optimiser ;
2. un modèle ;
3. les paramètres du modèle à optimiser ;
4. un critère ;
5. les conditions environnementales.

Procédure :

1. Étant donné une valeur de paramètre, évaluer le critère dans différentes conditions environnementales.
2. Moyenner ces évaluations.
3. Maximiser l'évaluation moyenne du critère par rapport aux paramètres à optimiser.

Paramètres maximisant

Si l'on dispose d'un paramètre discret :

Paramètres maximisant

Si l'on dispose d'un paramètre discret : recherche exhaustive.

Si l'on dispose d'un paramètre continu :

Paramètres maximisant

Si l'on dispose d'un paramètre discret : recherche exhaustive.

Si l'on dispose d'un paramètre continu :

- discrétisation ;
- nécessite des bornes ;
- nécessite un pas de discrétisation.

Paramètres maximisant

Si l'on dispose d'un paramètre discret : recherche exhaustive.

Si l'on dispose d'un paramètre continu :

- discrétisation ;
- nécessite des bornes ;
- nécessite un pas de discrétisation.

Si l'on dispose de plusieurs paramètres :

Paramètres maximisant

Si l'on dispose d'un paramètre discret : recherche exhaustive.

Si l'on dispose d'un paramètre continu :

- discrétisation ;
- nécessite des bornes ;
- nécessite un pas de discrétisation.

Si l'on dispose de plusieurs paramètres : recherche en grille.

Paramètres maximisant

Procédure :

1. Discrétiser les paramètres continus.
2. Pour chaque possibilité de valeurs des paramètres, . . .
3. Générer des conditions environnementales aléatoirement, et évaluer le critère.
4. Calculer le critère moyen.
5. Retourner l'ensemble des valeurs des paramètres associé à la valeur maximale du critère.

Exemple : le shoot

Exemple du tir (simple)

1. Action : shoot.
2. Modèle :
   - direction : $\vec{u} = \overrightarrow{\text{GOAL}} - \overrightarrow{\text{PLAYER}}$;
   - shoot = `Vector2D(shoot=`$\alpha \vec{u}$`)`.

Exemple du tir (simple)

1. Action : shoot.
2. Modèle :
   - direction : $\vec{u} = \overrightarrow{\text{GOAL}} - \overrightarrow{\text{PLAYER}}$;
   - shoot = `Vector2D(shoot=`$\alpha \vec{u}$`)`.
3. Paramètre : $\alpha \in [0, 2]$, continu.
4. Critère : nombre de buts marqués.
5. Conditions environnementales :
   - position de la balle sur le terrain ;
   - position du joueur par rapport à la balle (angle) ;
   - vitesse du joueur ;
   - etc.

Procédure

Procédure :

1. Discrétiser le paramètre $\alpha \in \{0, 0.1, 0.2, \ldots, 2\}$.
2. Pour chaque valeur de α, ...
3. Pour `essai` allant de 1 à ...
   - placer le ballon au hasard sur le terrain ;
   - placer le joueur sur le ballon ;
   - appliquer la stratégie avec ladite valeur de α ;
   - enregistrer s'il y a eu but ou non.
4. Calculer le nombre moyen de but (nombre de buts / nombre d'essais).
5. Retourner la valeur de α avec le meilleur ratio nombre de buts / nombre d'essais.

Exemple de code

```
class GoTestStrategy(Strategy):
    def __init__(self, strength=None):
        Strategy.__init__(self, "Go-getter")
        self.strength = strength

    def compute_strategy(self, state, id_team, id_player):
        s = SuperState(state, id_team, id_player)
        move = Move(s)
        shoot = Shoot(s)
        return move.to_ball() + shoot.to_goal(self.strength)
```

Exemple de code

```
class GoTestStrategy(Strategy):
    def __init__(self, strength=None):
        Strategy.__init__(self, "Go-getter")
        self.strength = strength

    def compute_strategy(self, state, id_team, id_player):
        s = SuperState(state, id_team, id_player)
        move = Move(s)
        shoot = Shoot(s)
        return move.to_ball() + shoot.to_goal(self.strength)
```

```
from socceria import GoalSearch, GoTestStrategy

expe = GoalSearch(strategy=GoTestStrategy(),
                  params={'strength': [0.1, 1]})
expe.start()
print(expe.get_res())
print(expe.get_best())
```

Exemple de code

```
class GoTestStrategy(Strategy):
    def __init__(self, strength=None):
        Strategy.__init__(self, "Go-getter")
        self.strength = strength

    def compute_strategy(self, state, id_team,
        s = SuperState(state, id_team, id_play
        move = Move(s)
        shoot = Shoot(s)
        return move.to_ball() + shoot.to_goal(
```

```
from socceria import GoalSearch, GoTestStrateg

expe = GoalSearch(strategy=GoTestStrategy(),
                  params={'strength': [0.1, 1]
expe.start()
print(expe.get_res())
print(expe.get_best())
```

```
/home/maxime/pysoccer/bin/python3.5 /home/maxime/bitb
{'strength': 0.1}    Crit: 1   Cpt: 1
{'strength': 0.1}    Crit: 1   Cpt: 2
{'strength': 0.1}    Crit: 1   Cpt: 3
{'strength': 0.1}    Crit: 2   Cpt: 4
{'strength': 0.1}    Crit: 2   Cpt: 5
{'strength': 0.1}    Crit: 3   Cpt: 6
{'strength': 0.1}    Crit: 3   Cpt: 7
{'strength': 0.1}    Crit: 3   Cpt: 8
{'strength': 0.1}    Crit: 3   Cpt: 9
{'strength': 0.1}    Crit: 4   Cpt: 10
{'strength': 0.1}    Crit: 4   Cpt: 11
{'strength': 0.1}    Crit: 5   Cpt: 12
{'strength': 0.1}    Crit: 6   Cpt: 13
{'strength': 0.1}    Crit: 7   Cpt: 14
{'strength': 0.1}    Crit: 7   Cpt: 15
{'strength': 0.1}    Crit: 8   Cpt: 16
{'strength': 0.1}    Crit: 9   Cpt: 17
{'strength': 0.1}    Crit: 9   Cpt: 18
{'strength': 0.1}    Crit: 9   Cpt: 19
{'strength': 0.1}    Crit: 9   Cpt: 20
{'strength': 1}    Crit: 0   Cpt: 1
{'strength': 1}    Crit: 0   Cpt: 2
{'strength': 1}    Crit: 0   Cpt: 3
{'strength': 1}    Crit: 0   Cpt: 4
{'strength': 1}    Crit: 0   Cpt: 5
{'strength': 1}    Crit: 0   Cpt: 6
{'strength': 1}    Crit: 0   Cpt: 7
{'strength': 1}    Crit: 0   Cpt: 8
{'strength': 1}    Crit: 0   Cpt: 9
{'strength': 1}    Crit: 1   Cpt: 10
{'strength': 1}    Crit: 1   Cpt: 11
{'strength': 1}    Crit: 1   Cpt: 12
{'strength': 1}    Crit: 2   Cpt: 13
{'strength': 1}    Crit: 3   Cpt: 14
{'strength': 1}    Crit: 3   Cpt: 15
{'strength': 1}    Crit: 3   Cpt: 16
{'strength': 1}    Crit: 4   Cpt: 17
{'strength': 1}    Crit: 4   Cpt: 18
{'strength': 1}    Crit: 4   Cpt: 19
{'strength': 1}    Crit: 4   Cpt: 20
{(('strength', 1),): 0.2, (('strength', 0.1),): 0.45}
(('strength', 0.1),)
```

Deuxième exemple de tir

1. Action : shoot.
2. Modèle :
   - direction : $\overrightarrow{u} = \overrightarrow{\text{GOAL}} - \overrightarrow{\text{PLAYER}}$;
   - $\overrightarrow{u} \leftarrow \frac{\overrightarrow{u}}{\| \overrightarrow{u} \|}$;
   - $p \leftarrow$ position dans la grille ;
   - shoot = `Vector2D(shoot=`$\alpha_p \overrightarrow{u}$`)`.

Deuxième exemple de tir

1. Action : shoot.
2. Modèle :
   - direction : $\overrightarrow{u} = \overrightarrow{\text{GOAL}} - \overrightarrow{\text{PLAYER}}$;
   - $\overrightarrow{u} \leftarrow \frac{\overrightarrow{u}}{\| \overrightarrow{u} \|}$;
   - $p \leftarrow$ position dans la grille ;
   - shoot = `Vector2D(shoot=`$\alpha_p \overrightarrow{u}$`)`.
3. Paramètres : $\alpha_1, \alpha_2, \cdots \in [0, 100]$, continu.
4. Critère : nombre de buts marqués.
5. Conditions environnementales :
   - position de la balle sur le terrain ;
   - position du joueur par rapport à la balle (angle) ;
   - vitesse du joueur ;
   - etc.

Procédure

Procédure :

1. Discrétiser les paramètres $\alpha_1, \alpha_2, \cdots \in \{0, 10, 20, \ldots, 100\}$.
2. Pour chaque position p sur la grille, ...
   - Pour chaque valeur de α_p, ...
   - Pour `essai` allant de 1 à ...
     - placer le ballon au hasard dans la cellule de la grille ;
     - placer le joueur sur le ballon ;
     - appliquer la stratégie avec ladite valeur de α_p ;
     - enregistrer s'il y a eu but ou non.
   - Calculer le nombre moyen de but (nombre de buts / nombre d'essais).
   - Retourner la valeur de α_p avec le meilleur ratio nombre de buts / nombre d'essais.

Troisième exemple de tir (réfléchi)

De quoi dépend le succès ?

Troisième exemple de tir (réfléchi)

De quoi dépend le succès ? $\rightarrow$ De la force.

De quoi dépend la force ?

Troisième exemple de tir (réfléchi)

De quoi dépend le succès ? $\rightarrow$ De la force.

De quoi dépend la force ? $\rightarrow$ De la distance aux cages.

Troisième exemple de tir (réfléchi)

De quoi dépend le succès ? $\rightarrow$ De la force.
De quoi dépend la force ? $\rightarrow$ De la distance aux cages.

1. Action : shoot.
2. Modèle :
   - direction : $\overrightarrow{u} = \overrightarrow{\text{GOAL}} - \overrightarrow{\text{PLAYER}}$;
   - $d \leftarrow \|\overrightarrow{u}\|$ (distance aux cages) ;
   - $\overrightarrow{u} \leftarrow \frac{\overrightarrow{u}}{\|\overrightarrow{u}\|}$;
   - shoot = `Vector2D(shoot=`$f_\alpha(d)\overrightarrow{u}$`)` ;
     - $f_\alpha(x) = 1 - e^{-\alpha x}$

Troisième exemple de tir (réfléchi)

De quoi dépend le succès ? $\rightarrow$ De la force.
De quoi dépend la force ? $\rightarrow$ De la distance aux cages.

1. Action : shoot.
2. Modèle :
   - direction : $\overrightarrow{u} = \overrightarrow{\text{GOAL}} - \overrightarrow{\text{PLAYER}}$;
   - $d \leftarrow \|\overrightarrow{u}\|$ (distance aux cages) ;
   - $\overrightarrow{u} \leftarrow \frac{\overrightarrow{u}}{\|\overrightarrow{u}\|}$;
   - shoot = `Vector2D(shoot=`$f_\alpha(d)\overrightarrow{u}$`)` ;
     - $f_\alpha(x) = 1 - e^{-\alpha x}$
     - $f_\alpha(x) = \frac{\mathbf{1}}{\mathbf{1}+e^{-\alpha(x-\mathbf{30})}}$

Troisième exemple de tir (réfléchi)

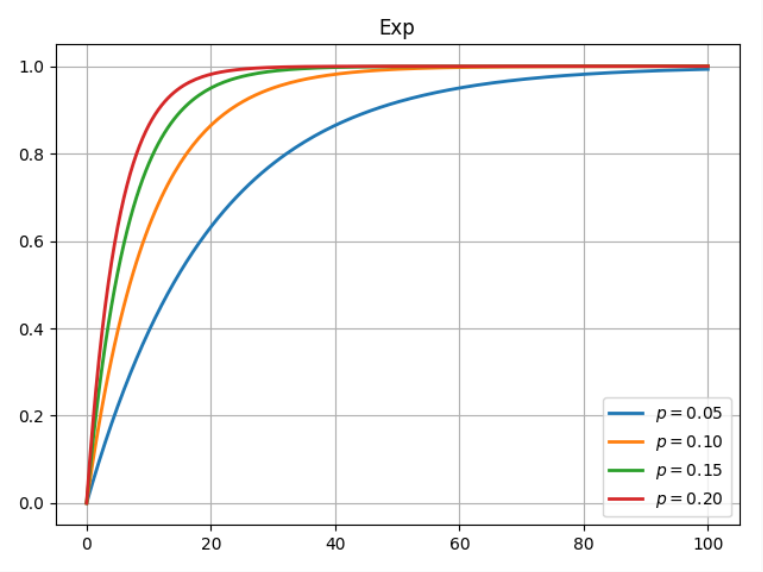
De quoi dépend le succès ? $\rightarrow$ De la force.
De quoi dépend la force ? $\rightarrow$ De la distance aux cages.

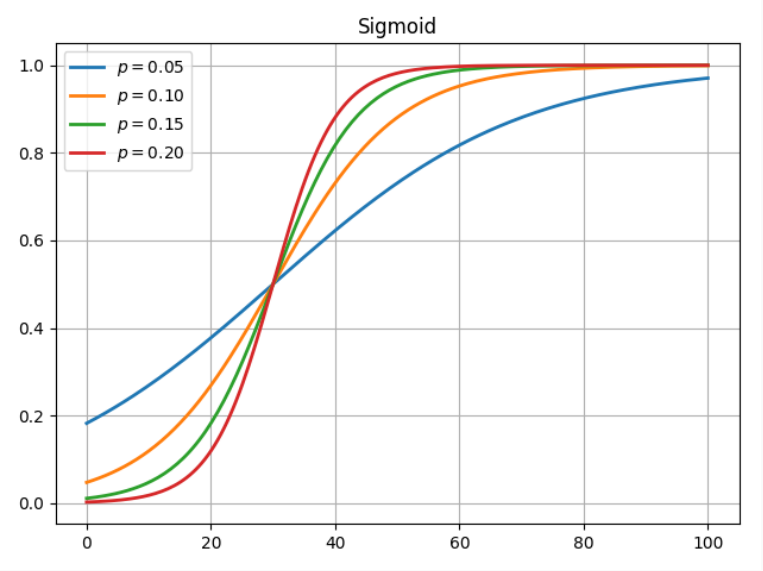
1. Action : shoot.
2. Modèle :
   - direction : $\overrightarrow{u} = \overrightarrow{\text{GOAL}} - \overrightarrow{\text{PLAYER}}$;
   - $d \leftarrow \|\overrightarrow{u}\|$ (distance aux cages) ;
   - $\overrightarrow{u} \leftarrow \frac{\overrightarrow{u}}{\|\overrightarrow{u}\|}$;
   - shoot = `Vector2D(shoot=`$f_\alpha(d)\overrightarrow{u}$`)` ;
     - $f_\alpha(x) = 1 - e^{-\alpha x}$
     - $f_\alpha(x) = \frac{1}{1+e^{-\alpha(x-30)}}$
3. Paramètre : $\alpha \in [0.05, 0.2]$, continu.
4. ...

Troisième exemple de tir (réfléchi)

De quoi dépend le succès ? $\rightarrow$ De la force.
De quoi dépend la force ? $\rightarrow$ De la distance aux cages.

1. Action : shoot.
2. Modèle :
   - direction : $\overrightarrow{u} = \overrightarrow{\text{GOAL}} - \overrightarrow{\text{PLAYER}}$;
   - $d \leftarrow \|\overrightarrow{u}\|$ (distance aux cages) ;
   - $\overrightarrow{u} \leftarrow \frac{\overrightarrow{u}}{\|\overrightarrow{u}\|}$;
   - shoot = `Vector2D(shoot=`$f_\alpha(d)\overrightarrow{u}$`)` ;
     - $f_\alpha(x) = 1 - e^{-\alpha x}$
     - $f_\alpha(x) = \frac{1}{1+e^{-\alpha(x-30)}}$
3. Paramètre : $\alpha \in [0.05, 0.2]$, continu.
4. ...





En pratique

La classe simulation

Méthodes implémentées par les observateurs :

- `begin_match(team1,team2,state)` : déclenchée au début du match. Peut être utilisé pour initialiser les paramètres de la procédure d'optimisation.

La classe simulation

Méthodes implémentées par les observateurs :

- `begin_match(team1,team2,state)` : déclenchée au début du match. Peut être utilisé pour initialiser les paramètres de la procédure d'optimisation.
- `end_match(team1,team2,state)` : déclenchée à la fin du match.

La classe simulation

Méthodes implémentées par les observateurs :

- `begin_match(team1,team2,state)` : déclenchée au début du match. Peut être utilisé pour initialiser les paramètres de la procédure d'optimisation.
- `end_match(team1,team2,state)` : déclenchée à la fin du match.
- `begin_round(team1,team2,state)` : déclenchée à chaque coup d'envoi. Peut être utilisé pour régler les conditions environnementales du jeu (placement aléatoire du ballon).

La classe simulation

Méthodes implémentées par les observateurs :

- `begin_match(team1,team2,state)` : déclenchée au début du match. Peut être utilisé pour initialiser les paramètres de la procédure d'optimisation.
- `end_match(team1,team2,state)` : déclenchée à la fin du match.
- `begin_round(team1,team2,state)` : déclenchée à chaque coup d'envoi. Peut être utilisé pour régler les conditions environnementales du jeu (placement aléatoire du ballon).
- `end_round(team1,team2,state)` : déclenchée à chaque but. Peut être utilisé pour calculer le critère et passer à la prochaine valeur du paramètre à optimiser.

La classe simulation

Méthodes implémentées par les observateurs :

- `begin_match(team1,team2,state)` : déclenchée au début du match. Peut être utilisé pour initialiser les paramètres de la procédure d'optimisation.
- `end_match(team1,team2,state)` : déclenchée à la fin du match.
- `begin_round(team1,team2,state)` : déclenchée à chaque coup d'envoi. Peut être utilisé pour régler les conditions environnementales du jeu (placement aléatoire du ballon).
- `end_round(team1,team2,state)` : déclenchée à chaque but. Peut être utilisé pour calculer le critère et passer à la prochaine valeur du paramètre à optimiser.
- `update_round(team1,team2,state)` : déclenchée à chaque pas de jeu. Peut être utilisé pour contrôlé le temps alloué à chaque essai.

Exemple de code

```
class GoalSearch(object):
    def __init__(self, strategy, params, simu=None, trials=20,
                 max_steps=1000000, max_round_step=40):
        self.strategy = strategy
        self.params = params.copy()
        self.simu = simu
        self.trials = trials
        self.max_steps = max_steps
        self.max_round_step = max_round_step

    def start(self, show=True):
        if not self.simu:
            team1 = SoccerTeam("Team␣1")
            team2 = SoccerTeam("Team␣2")
            team1.add(self.strategy.name, self.strategy)
            team2.add(Strategy().name, Strategy())
            self.simu = Simulation(team1, team2, max_steps=self.max_steps)
        self.simu.listeners += self

        if show:
            show_simu(self.simu)
        else:
            self.simu.start()
```

Exemple de code

```
def begin_match(self, team1, team2, state):
    self.last_step = 0  # Step of the last round
    self.criterion = 0  # Criterion to maximize (here, number of goals)
    self.cpt_trials = 0  # Counter for trials
    self.param_grid = iter(ParameterGrid(self.params))  # Iterator for the
    self.cur_param = next(self.param_grid, None)  # Current parameter
    if self.cur_param is None:
        raise ValueError('no parameter given.')
    self.res = dict()  # Dictionary of results
```

Exemple de code

```
def begin_match(self, team1, team2, state):
    self.last_step = 0  # Step of the last round
    self.criterion = 0  # Criterion to maximize (here, number of goals)
    self.cpt_trials = 0  # Counter for trials
    self.param_grid = iter(ParameterGrid(self.params))  # Iterator for the
    self.cur_param = next(self.param_grid, None)  # Current parameter
    if self.cur_param is None:
        raise ValueError('no parameter given.')
    self.res = dict()  # Dictionary of results

def begin_round(self, team1, team2, state):
    ball = Vector2D.create_random(low=0, high=1)
    ...

    # Player and ball postion (random)
    self.simu.state.states[(1, 0)].position = ball.copy()  # Player positio
    self.simu.state.states[(1, 0)].vitesse = Vector2D()  # Player accelerat
    self.simu.state.ball.position = ball.copy()  # Ball position

    self.last_step = self.simu.step  # Last step of the game

    # Set the current value for the current parameters
    for key, value in self.cur_param.items():
        setattr(self.strategy, key, value)
```

Exemple de code

```
def end_round(self, team1, team2, state):
    # A round ends when there is a goal of if max step is achieved
    if state.goal > 0:
        self.criterion += 1  # Increment criterion

    self.cpt_trials += 1  # Increment number of trials

    print(self.cur_param, end="␣␣␣␣")
    print("Crit:␣{}␣␣␣Cpt:␣{}".format(self.criterion, self.cpt_trials))

    if self.cpt_trials >= self.trials:
        # Save the result
        self.res[tuple(self.cur_param.items())] = self.criterion * 1. / sel

        # Reset parameters
        self.criterion = 0
        self.cpt_trials = 0

        # Next parameter value
        self.cur_param = next(self.param_grid, None)
        if self.cur_param is None:
            self.simu.end_match()
```

Exemple de code

```
def update_round(self, team1, team2, state):
    # Stop the round if it is too long
    if state.step > self.last_step + self.max_round_step:
        self.simu.end_round()
```

Exemple de code

```
def update_round(self, team1, team2, state):
    # Stop the round if it is too long
    if state.step > self.last_step + self.max_round_step:
        self.simu.end_round()

def get_res(self):
    return self.res

def get_best(self):
    return max(self.res, key=self.res.get)
```

Objectifs du TME

Objectifs du TME

- organiser son code à l'aide des motifs de conception ;
- paramétrer les stratégies (tir, défense, anticipation de la position de la balle. . .) ;
- optimiser les stratégies par recherche en grille.

A chaque TME

Mettre à jour le dépôt contenant le simulateur :

```
cd [DOSSIER DU SIMULATEUR]
git pull
```