Programmation 2 : OSCAR

Pastureau Romain, Rodriguez Charlotte, L3 MIASHS

December 14, 2015

Brève description

La classe principale est World. Elle lit le fichier texte, et à partir de celui-ci crée les agents (instances de la classe Agent) et les différents états (instances des classes Mineral, Vegetal ou Animal) que peuvent prendre ces agents.

Le passage d'un agent d'un état à un autre sera dépendant de l'évolution de ses variables (âge, poids, proximité d'autres agents, etc). Une façon de visualiser les données que nous traitons est de considérer un graphe qui représente l'agent et dont les noeuds sont les états qu'il peut prendre.

La simulation consiste à faire vivre les agents. Nous enregistrons donc au temps t l'état du monde et faisons vivre chaque agent dans dans ce monde de l'instant t. Une fois que nous sommes au temps t+1 pour tous les agents, il faut résoudre les éventuels conflits résultant d'agents étant allés sur la même case du monde. Et le temps avance ainsi, tant qu'il reste des agents 'vivant' dans le monde.

Selon son ordre (animal, végétal, minéral) et ses caractéristiques, l'agent pourra ou non effectuer certaines actions (bouger, se reproduire), sera ou non sensible à certains champs, et émettra ou non certains champs.

Ainsi, à partir des caractéristiques propres à chaque agent, Oscar nous permet d'observer le comportement global qui émerge de la 'population'.

Pour la partie graphique, réalisée sur Pygame, nous générons d'abord une liste aléatoire d'images contenues dans le sous-dossier "soil" qui constituera un sol unique aux motifs non-répétés. Ensuite, nous ajoutons par-dessus les sprites des agents au temps T indiqué en bas à droite de l'interface. Les trois boutons permettent respectivement de revenir d'un pas en arrière (fonctionnalité non-achevée), de lancer la simulation (en incrémentant le temps toutes les 200 ms) et d'avancer d'un pas.

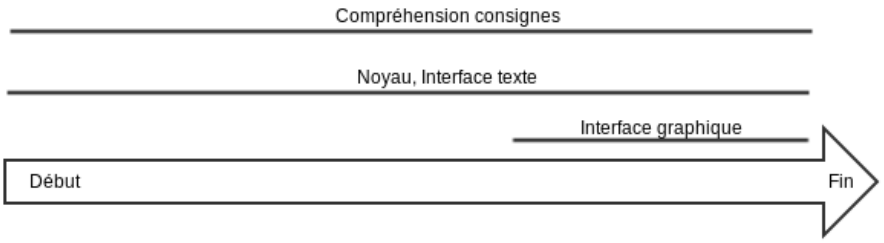

Figure 1: Chronologie

Fichiers

- Noyau oscar_noyau.py
- Interface texte oscar_texte.py
- Interface graphique oscar_graphique.py
- Créateur des agents du jeu de la vie CreationFicJDV.py
- Autres ezCLI.py, ezTK.py
- Fichiers texte de paramétrage (commencent par 't_' pour l'interface texte, par 'g_' pour l'interface graphique)
  - pour le jeu de la vie : jdv_36x19.txt, jdv_base.txt
  - autres : animal_cailloux.txt, animal_cailloux_vege_vieillesse.txt, breed.txt, male_femelle.txt, trace_seul.txt, trace_seul_vieillesse.txt.
- Images

Pour tester le programme Ouvrir le fichier oscar_texte.py ou ou le fichier oscar_graphique.py et faire run.

Difficultés et perspectives

Les difficultés que nous avons rencontrée ici sont surtout centrées autour de la compréhension de la consigne et du format des données du fichier texte.

L'implémentation que nous avons faite a pour défaut d'être lente à l'exécution (d'autant plus que le monde est peuplé d'agents). Une prochaine étape serait donc de l'optimiser pour diminuer de temps d'exécution. Cela pourrait notamment passer une nouvelle méthode de calcul de l'étendue des champs (ne la calculer que sur l'étendue du monde qui nous intéresse ; ou bien la calculer une unique fois à chaque temps t pour ne modifier que la contribution de l'agent qui 'appelle ce calcul').

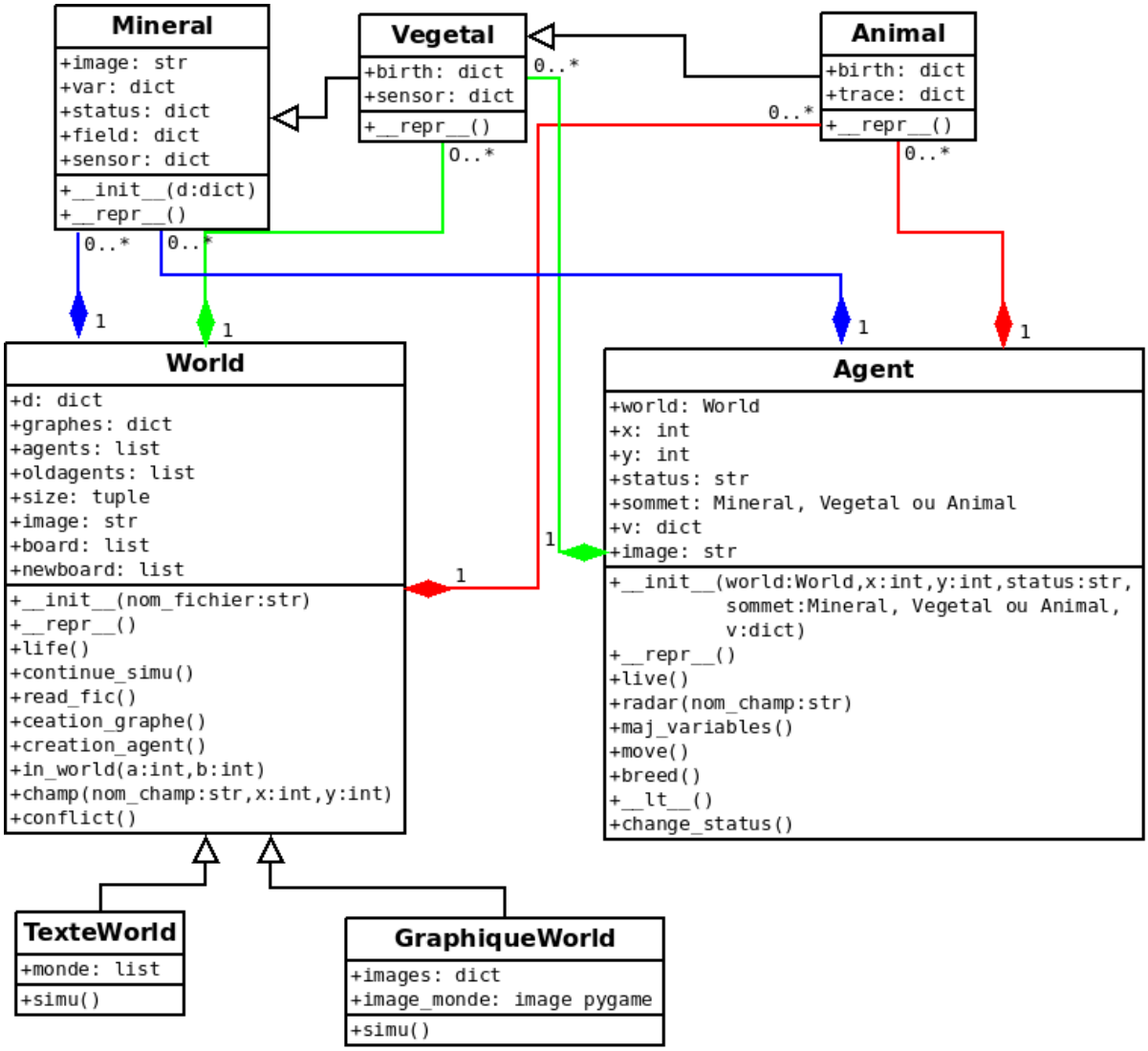
Mineral
+image: str
+var: dict
+status: dict
+field: dict
+sensor: dict
+__init__(d:dict)
+__repr__()
Vegetal
+birth: dict
+sensor: dict
+__repr__()
Animal
+birth: dict
+trace: dict
+__repr__()
0..*
1
World
+d: dict
+graphes: dict
+agents: list
+oldagents: list
+size: tuple
+image: str
+board: list
+newboard: list
+__init__(nom_fichier:str)
+__repr__()
+life()
+continue_simu()
+read_fic()
+ceation_graphe()
+creation_agent()
+in_world(a:int,b:int)
+champ(nom_champ:str,x:int,y:int)
+conflict()
Agent
+world: World
+x: int
+y: int
+status: str
+sommet: Mineral, Vegetal ou Animal
+v: dict
+image: str
+__init__(world:World,x:int,y:int,status:str,
sommet:Mineral, Vegetal ou Animal,
v:dict)
+__repr__()
+live()
+radar(nom_champ:str)
+maj_variables()
+move()
+breed()
+__lt__()
+change_status()
TexteWorld
+monde: list
+simu()
GraphiqueWorld
+images: dict
+image_monde: image pygame
+simu()

Figure 2: UML

Par ailleurs, le fait de devoir écrire un fichier texte de paramétrage rend l'utilisation plus compliquée. D'une part parce qu'il est difficile de visualiser ce que l'on écrit sous ce format, d'autre part parce qu'une erreur est vite commise. Il serait donc intéressant de créer une interface graphique qui écrira elle même le fichier texte puis lancera la programme. Elle pourrait faciliter la compréhension et l'écriture des relations d'attraction répulsion (field, sensor). Elle expliciterait les passages d'un status à un autre.

Une autre chose à travailler davantage est le contenu des fichiers texte : travailler les caractéristiques des agents de façon à reproduire des comportement observés dans le monde réel.