Université de Lille

Simulation centrée individu

[Sous-titre du document]

Célestine SAUVAGE - Sami HALABI
23/10/2018

Table des matières

1 Bilan

1.1 Résumé

Dans ce répertoire vous trouverez nos trois implémentations des TP1, 2 et 3. Chaque implémentation de core se trouve dans un répertoire spécifique du TP. Chaque package à son propre fichier SMA et Properties.json.

Pour permettre une approche orientée objet, nous avons dû délaisser l'aspect performance de notre implémentation.

En effet, de nombreux calculs ont été délégués aux agents, permettant leur indépendance. Néanmoins, cela augmente le nombre de comparaisons entre les cases de l'environnement.

Hormis ce léger problème l'ensemble des tps est fonctionnel et implémente toutes les fonctionnalités demandées.

1.2 Implémentation

Pour chaque TP, on a un diagramme UML qui décrit son architecture (voir partie implémentation de chaque TP).

Pour le scheduling des agents durant un tour, 3 ordonnancements sont possibles :

1. Tous les agents agissent toujours dans le même ordre.
2. Chaque tour, on mélange la liste des agents aléatoirement.
3. Chaque tour, on tire n agents aléatoirement qui vont agir pendant ce tour.

1.3 Installation

Pour faire fonctionner notre TP, il faut installer les modules pynput, numpy et matplotlib, pour python3. Utiliser python3 pour l'exécution de nos TP.

1.4 Lien Répertoire

https://github.com/lrdoz/Simulation_centre_individus.git

2 TP1 - Particules

2.1 Objectif

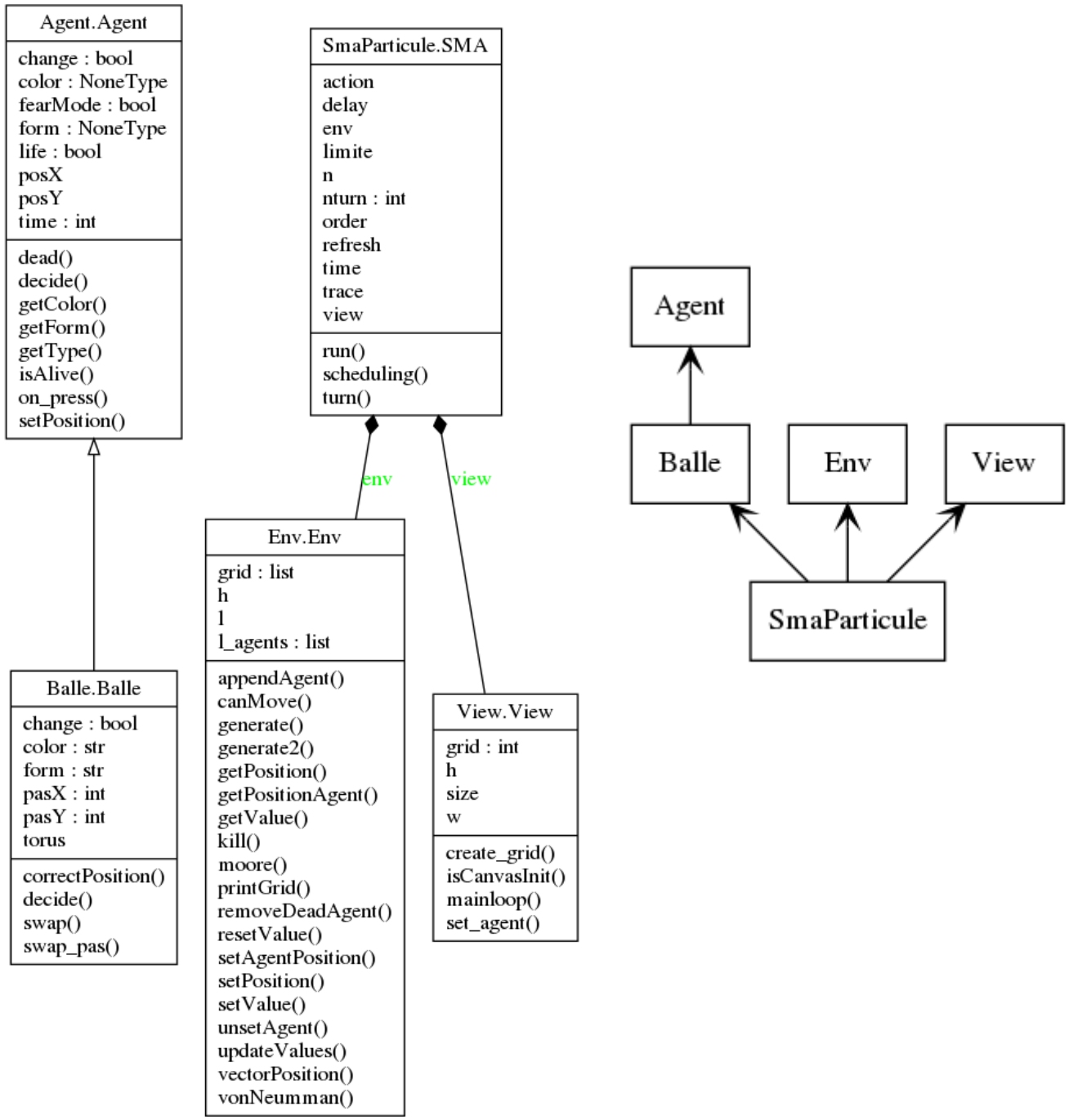
L'objectif du TP est de simuler un environnement contenant des agents se déplaçant et interagissant avec d'autres agents. Les agents sont des billes, possédant un vecteur de déplacement, et se bougeant dans leur environnement.

L'environnement peut être torique ou non torique, càd que les boules peuvent rebondir (ou non) contre les murs.

Lorsque deux agents se collisionnent, différents comportements peuvent être là aussi adoptés (voir question 3).

Le fichier Properties.json gère les différentes options pour lancer la simulation (se référer aux instructions du TP).

2.2 Implémentation



2.3 Exécution

1. Pour exécuter, faire :

```
make particule
```

2.4 Question

1. Combien d'agents peut-on raisonnablement faire tourner simultanément ? (Fréquence d'un tour par population)
2. Tracer quelques courbes montrant l'évolution du nombre de collisions par tic, pour le même nombre de billes mais selon les tailles d'environnement.
3. Variation sur le comportement :
   - Si une bille est là où je dois aller, je ne fais rien
   - Si une bille est là où je dois aller, j'inverse ma direction
   - Si une bille est là où je dois aller, j'inverse ma direction avec l'autre

   Que se passe-t-il dans ces différents cas ? Quel est le comportement qui fournit le meilleur rendu ?

2.5 Réponse

1. Raisonnablement : ~ 80.000 agents, avec un taux de rafraichissement de 2 images par seconde.
2. Pour afficher
3. Comportement :
   - Premier comportement : Les billes forment des tas, en effet elles s'immobilisent lors d'une collision.
   - Deuxième comportement : Les billes immobiles le resteront et les autres se déplaceront toujours sur le même vecteur.
   - Troisième comportement : Les billes ont un comportement plus aléatoire et naturel.
4. Bonus : On a voulu tester d'autres comportements, on a donc essayé d'appliquer le comportement 2 et 3, de manière aléatoire.

   Les billes se déplacent visuellement en groupe, mais quelques fois dévient de leur trajectoire.

Le comportement avec un meilleur rendu visuel est le quatrième comportement, car il est plus linéaire, mais garde un aspect aléatoire.

Cependant, le comportement trois est plus naturel, les billes ont un déplacement arbitraire au sein de l'environnement.

2.6 Rendu visuel

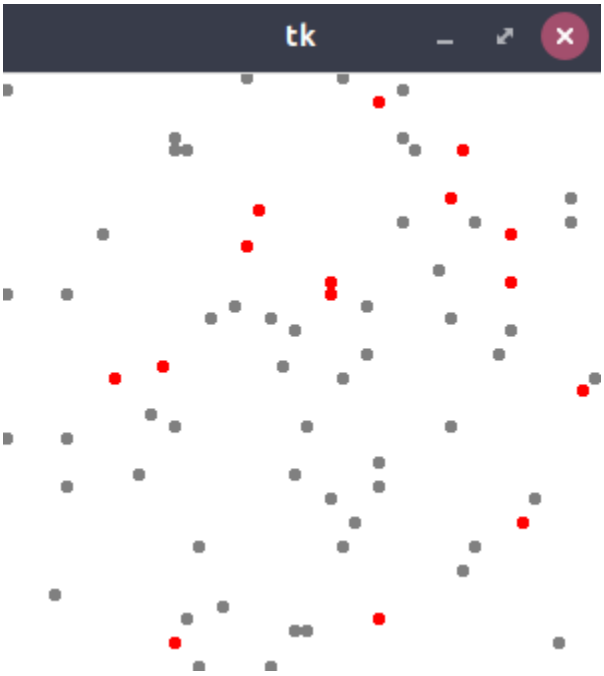


*Figure 1 Env 50*50 avec 70 balles, rouges après plusieurs tics*

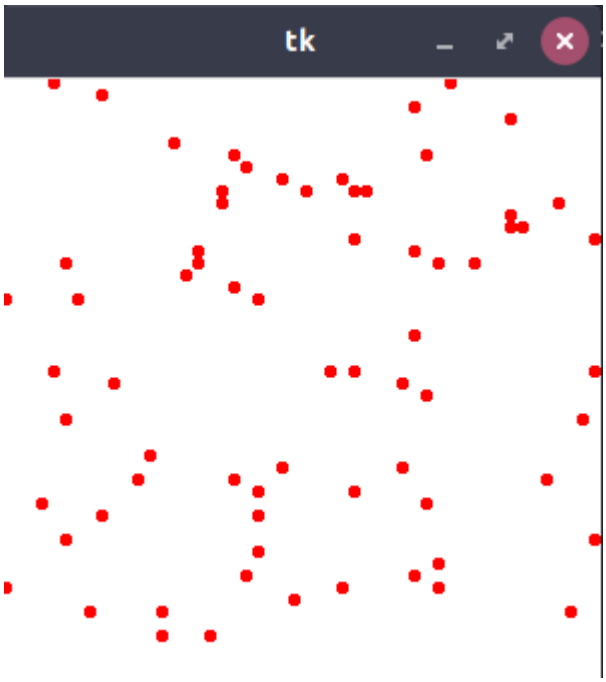


*Figure 2 Env 50*50 avec 70 balles, grises au début*

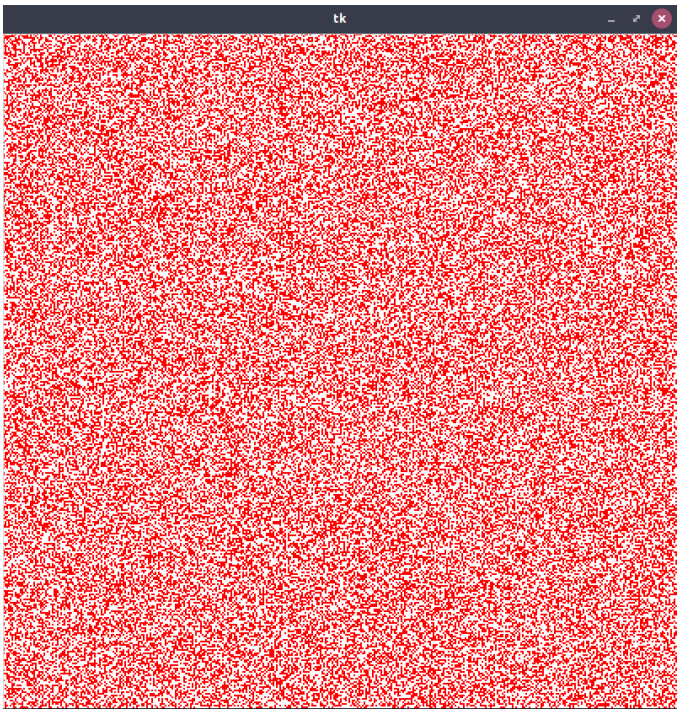


*Figure 3 Env 400*400, 80 000 agents*

3 TP2 - Shark Vs Fish

3.1 Objectif

L'objectif du TP est de simuler un environnement **proie prédateur**, pour se faire nous avons 2 agents. *Les poissons* : peuvent se déplacer et se reproduire dans l'environnement.

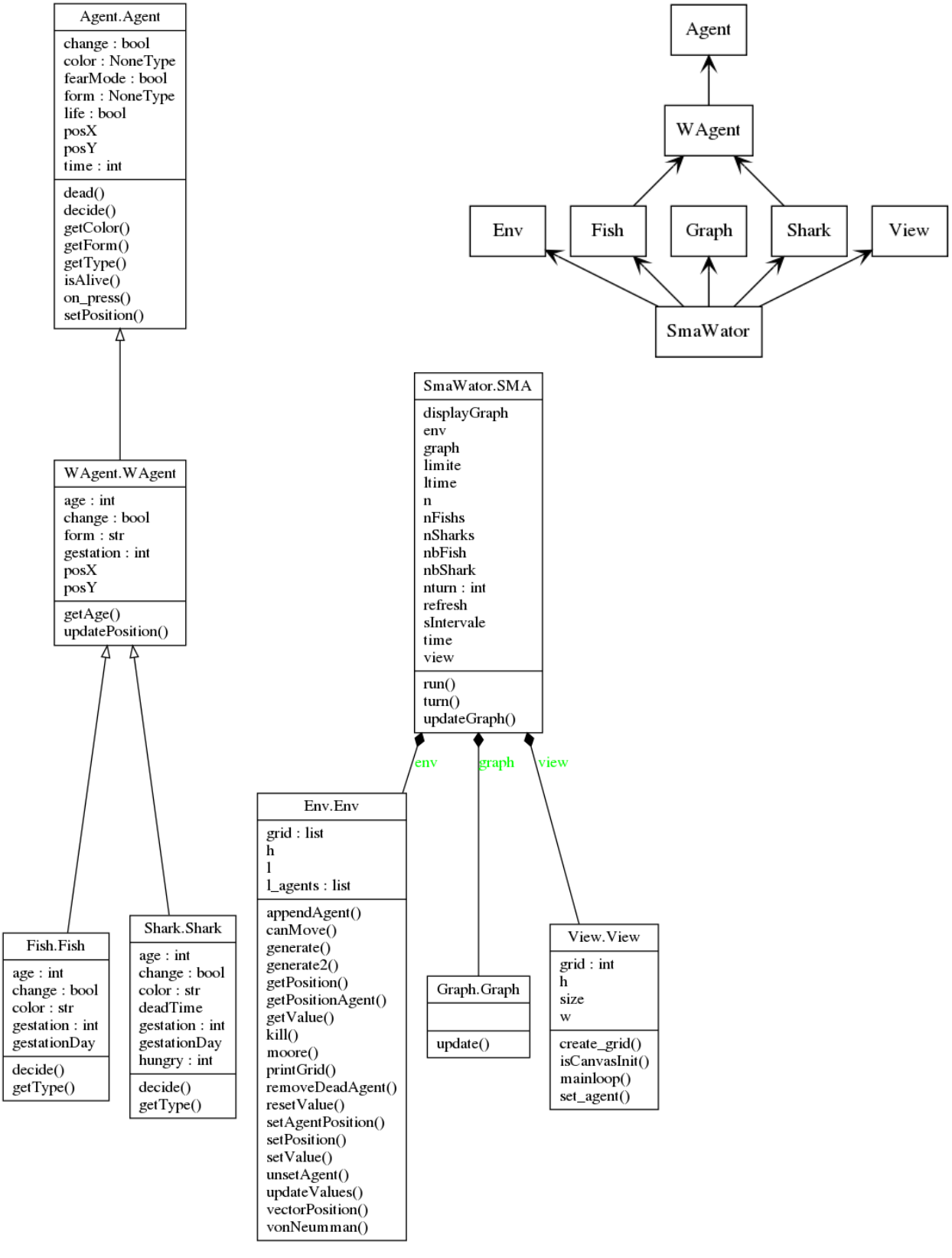
Les requins : peuvent se déplacer, se reproduire et/ou manger les poissons dans l'environnement.

3.2 Implémentation

Pour la gestion des naissances/décès, on va avoir une liste des agents dans SMA.

Naissance : Lors d'une naissance, l'enfant prend la place de son parent après que celui-ci est bougé. Il est ajouté également dynamiquement en fin de liste, il va donc agir pendant le même tour que son parent.

Décès : Pour pouvoir mettre à jour la vue, un agent qui décède n'est pas directement supprimé de la liste des agents. Chaque agent a donc un attribut `life`, qui permet de savoir si un agent est mort ou non. Une fois que tous les agents ont agi dans le tour. On supprime les points représentant les agents morts de la vue, puis on les enlève de la liste des agents.


Agent.Agent
change : bool
color : NoneType
fearMode : bool
form : NoneType
life : bool
posX
posY
time : int
dead()
decide()
getColor()
getForm()
getType()
isAlive()
on_press()
setPosition()
Agent
WAgent
Env
Fish
Graph
Shark
View
SmaWator
WAgent.WAgent
age : int
change : bool
form : str
gestation : int
posX
posY
getAge()
updatePosition()
SmaWator.SMA
displayGraph
env
graph
limite
ltime
n
nFishs
nSharks
nbFish
nbShark
nturn : int
refresh
sIntervale
time
view
run()
turn()
updateGraph()
env
graph
view
Env.Env
grid : list
h
l
l_agents : list
appendAgent()
canMove()
generate()
generate2()
getPosition()
getPositionAgent()
getValue()
kill()
moore()
printGrid()
removeDeadAgent()
resetValue()
setAgentPosition()
setPosition()
setValue()
unsetAgent()
updateValues()
vectorPosition()
vonNeumman()
Fish.Fish
age : int
change : bool
color : str
gestation : int
gestationDay
decide()
getType()
Shark.Shark
age : int
change : bool
color : str
deadTime
gestation : int
gestationDay
hungry : int
decide()
getType()
Graph.Graph
update()
View.View
grid : int
h
size
w
create_grid()
isCanvasInit()
mainloop()
set_agent()

3.3 Exécution

1.Pour l'exécution, faire :

```
make wator
```

3.4 Question

1. Est-il préférable d'initialiser les agents avec les mêmes valeurs pour les 3 différents compteurs ou initialiser ces compteurs aléatoirement ?
2. Testez différentes variantes comportementales. Quels sont les comportements qui donnent les meilleures dynamiques ?
   - Une action à chaque tic : soit manger, soit se reproduire, soit bouger
   - Se reproduire en bougeant
   - Se reproduire en mangeant

   Vous fournirez notamment la Courbe d'évolution du nombre de Fish et de Shark et de la courbe d'évolution du nombre de Fish sur le nombre de Shark pour chacun de ces cas.

3.5 Réponse

1. Il n'est pas préférable d'initialiser les mêmes valeurs, en effet selon la taille de la grille et la disposition, les requins vont disparaitre, et / ou les poissons. Il est plus préférable d'adapter les valeurs à l'environnement.
2. Comportement :
   - Premier comportement : Dans ce comportement les poissons forment des attroupements, qui grandissent et se réduisent au cours du temps, sans jamais disparaître. En effet, les requins sont moins « voraces » et peuvent ne pas manger des poissons certains tours.
   - Deuxième comportement : *Comportement par défaut dans tous les cas car plus facile à gérer pour les naissances.*
   - Troisième comportement : On constate la formation de nombreux, vague de requin poursuivant un amas de poisson jusqu'à son extermination.

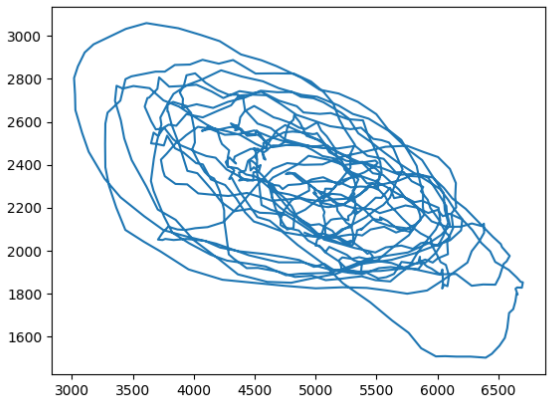


Figure 5 Courbe décrivant un comportement aléatoire

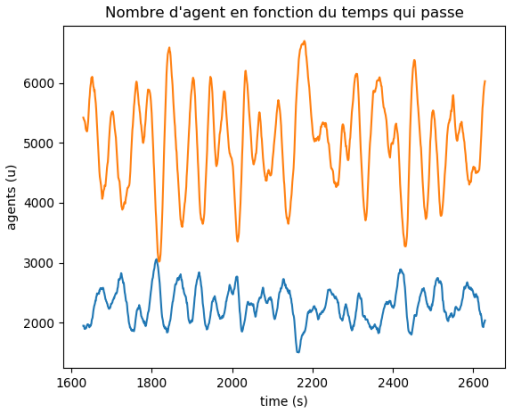


Figure 4Courbe du nombre de requin et poisson au cours du temps pour un comportement aléatoire

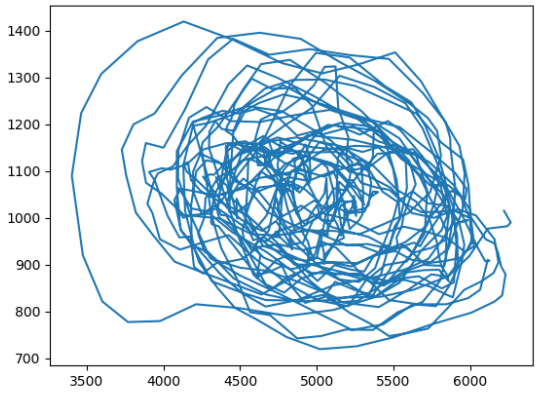


Figure 7Courbe représentant requin sur poison, pour le comportement 3

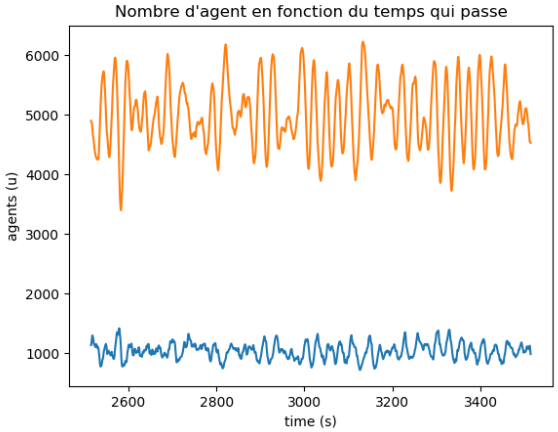


Figure 6Courbe du nombre de requin et poisson au cours du temps pour le comportement 3

3.6 Rendu visuel

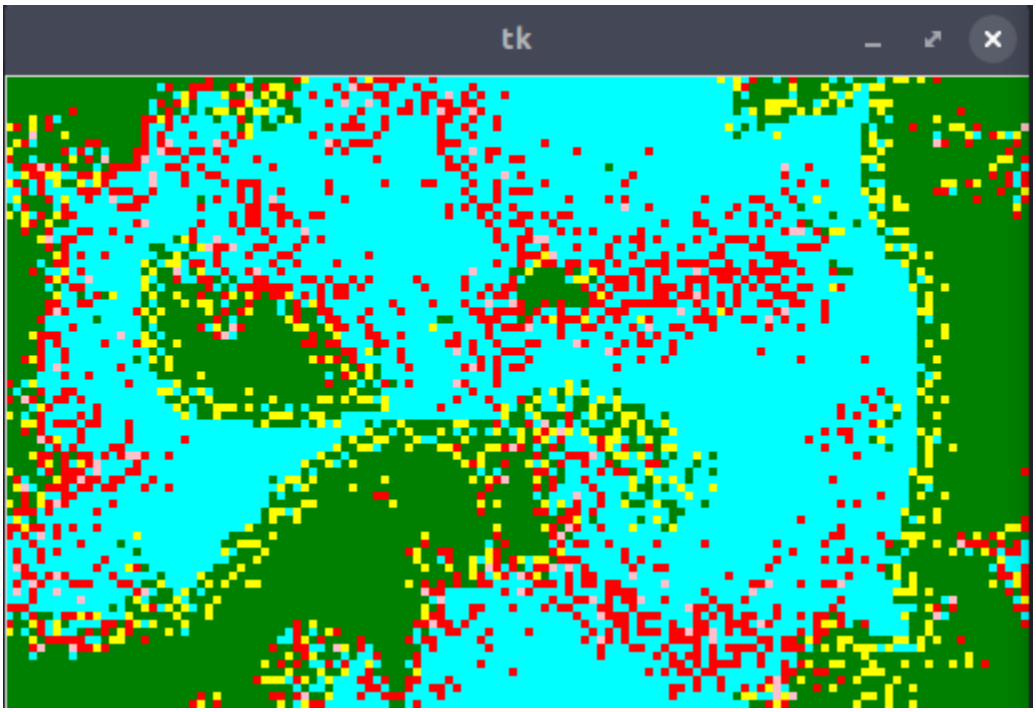


Figure 8 Requins vs Poissons. Les poissons sont jaunes à la naissance et deviennent verts. Les requins sont roses à la naissance et deviennent rouges.

4 TP3 - Pack Man

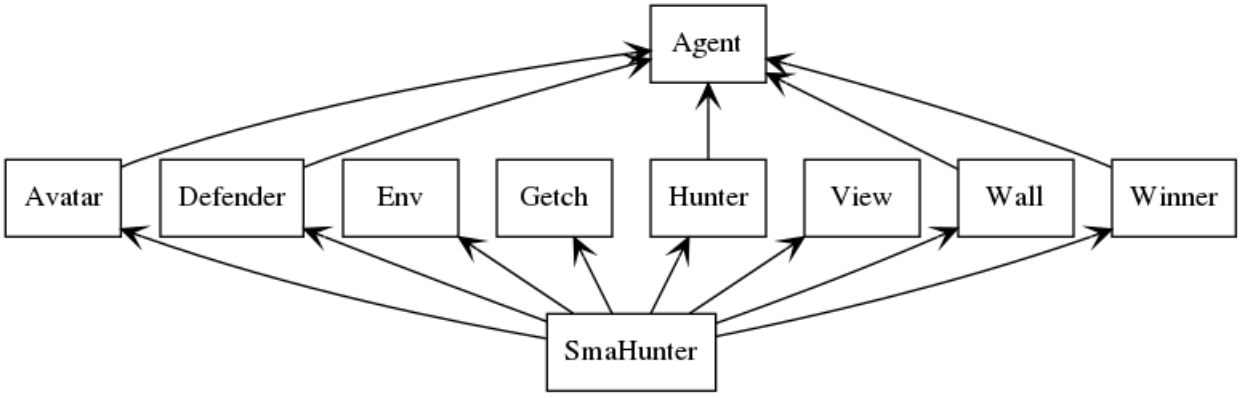
4.1 Objectif

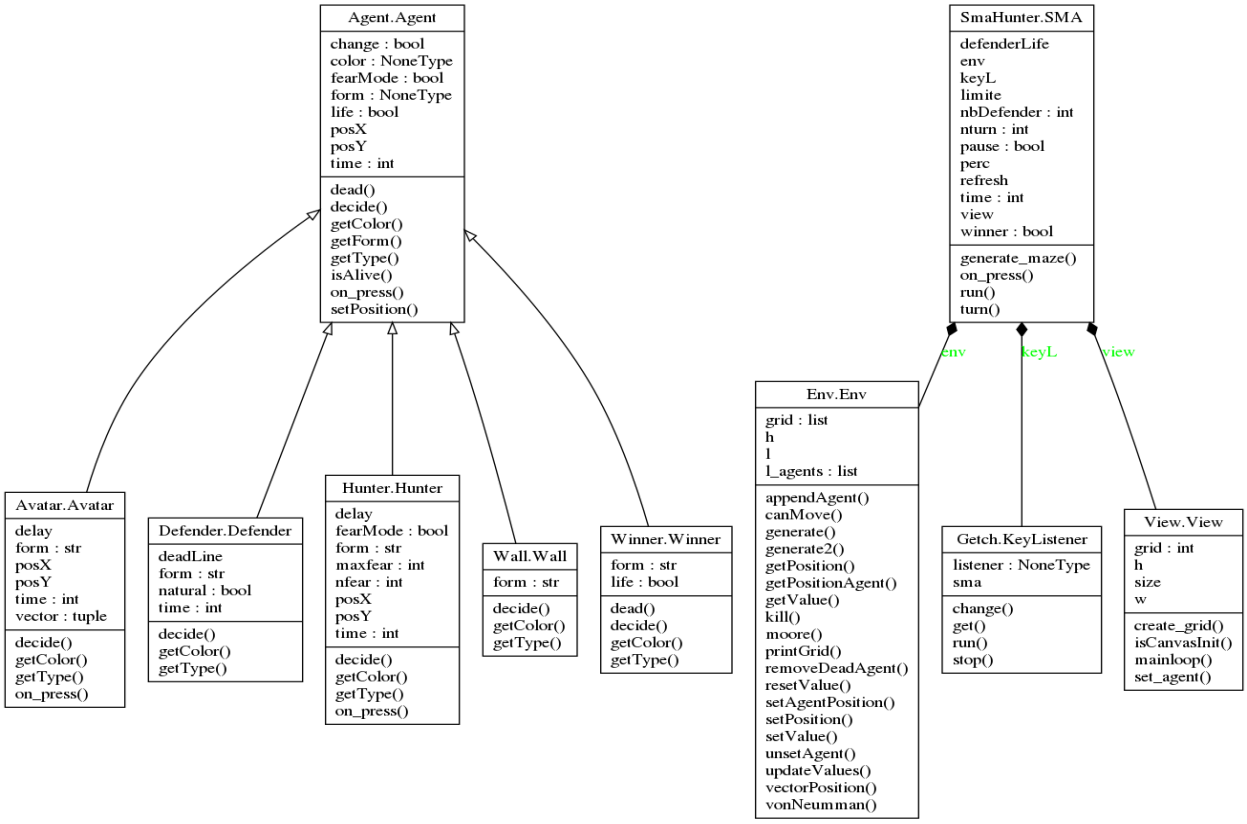
L'objectif du TP est de mettre un système temps réel, avec une interaction humaine sur le l'environnement.

Pour se faire, nous créer un agent pouvant être maitrisé par un individu, et des agents trackant l'humain.

L'individu, peut se déplacer dans l'environnement, manger certain agent, et mourir.

4.2 Implémentation





4.3 Exécution

1. Pour l'exécution, faire :

```
make hunter
```

4.4 Rendu visuel

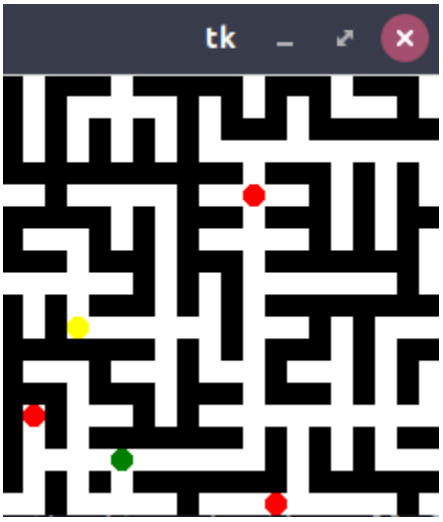


Figure 10 Génération entière d'un labyrinthe

Figure 9 Génération partielle du labyrinthe