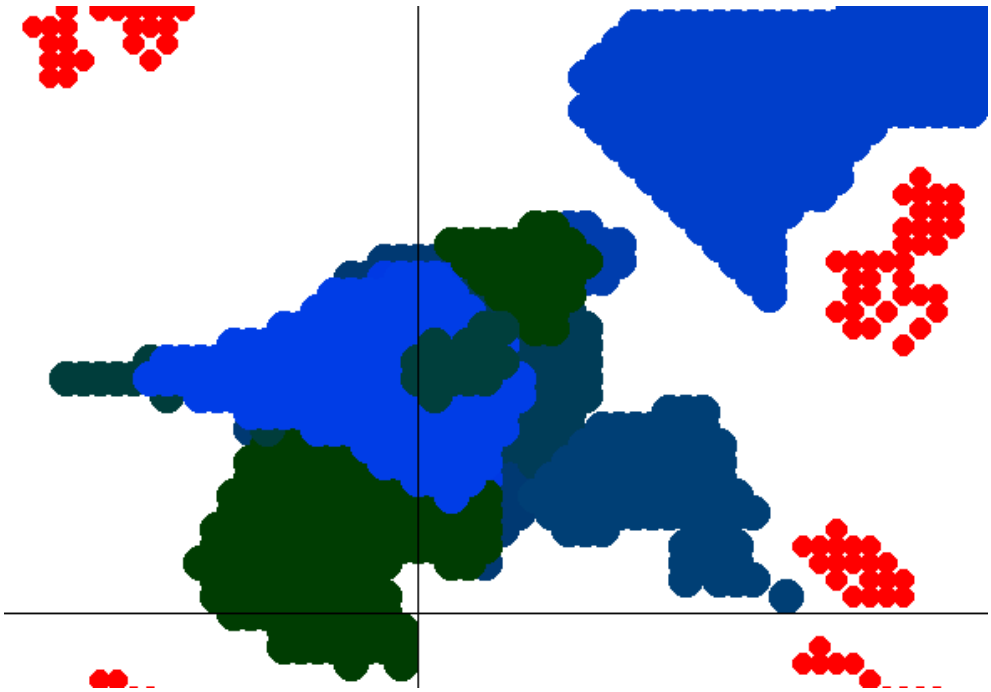
SORBONNE UNIVERSITÉ

POLYTECH SORBONNE

MAIN 4

Rapport de Projet C++ Smoke

Sélection naturelle sur des fumées vivantes



Achille BAUCHER, Ibtissam LACHHAB

16 janvier 2020

Table des matières

1 Introduction **2**

2 Classes générales **2**

2.1 Vue d'ensemble 2

2.2 Description détaillée 3

2.2.1 Utilitaires 3

2.2.2 Les Entités 3

2.2.3 Les jeux 3

3 Le Jeu standart **4**

3.1 Principe 4

3.2 Entités en présence 4

3.2.1 Mur (en noir) 4

3.2.2 Projectile (en rouge) 4

3.2.3 Fumee_test (en gris) 4

3.2.4 Fumee_esquive (nuances de violet à turquoise) 5

3.2.5 Esquive_vie 5

3.2.6 Mystherbe 5

3.3 Résultats et interprétation 5

4 Utilisation **6**

4.1 Documentation doxygen 6

4.2 Installation git 6

4.3 Jouer 6

4.3.1 Compilation 6

4.3.2 Changement des règles 6

4.3.3 Éxécution 6

4.4 Statistiques 7

4.5 Performances 7

4.6 Résumé 7

5 Remarques **8**

5.1 Types de conteneurs 8

5.2 Fuites Valgrind 8

6 Annexes **9**

1 Introduction

L'idée est de programmer une simulation informatique de différentes entités évoluant dans un milieu hostile, et qui se reproduisent régulièrement en léguant des attributs plus ou moins mutés à leurs enfants. Il serait ainsi possible d'étudier plusieurs facettes de la sélection naturelle, en observant par exemple quels types de comportements et de mutations sont susceptibles d'adapter le plus facilement l'espèce à l'environnement extérieur.

Les entités modélisées sont représentées par un ensemble de cases en 2 dimensions qui se meuvent en changeant de forme suggérant un mouvement de fumée ou de gaz qui a donné l'idée au projet.

2 Classes générales

Nous avons pensé l'architecture du programme de manière à ce qu'une partie du code pose les grands principes et ne soit pas changée par la suite. Pour chaque nouveau type d'entité avec des comportements différents ou de nouvelles règles du jeu, il suffit de créer les classes filles correspondantes et qui respectent les contraintes. On peut trouver les définitions de ces classes générales dans le dossier *src/general*.

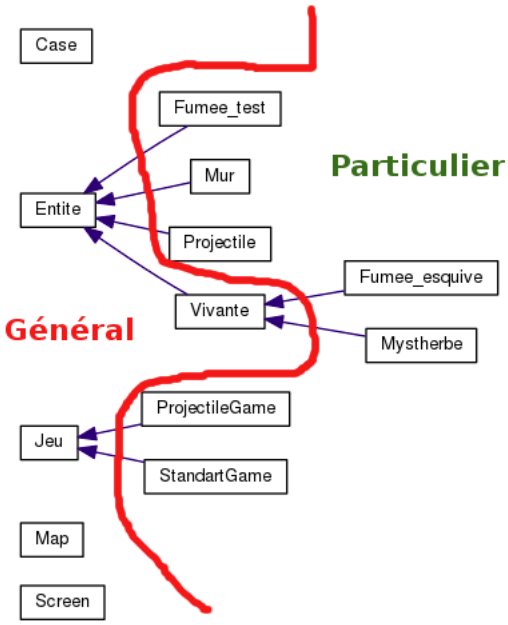


FIGURE 1 – UML résumé (Doxygen)

2.1 Vue d'ensemble

Ces classes générales se décomposent comme suit :

Le **Jeu** contient une liste d'**Entités** simples, et de **Vivantes** (héritées de **Entité**) qui peuvent se reproduire. A chaque tour, le **Jeu** se charge de faire avancer chacune d'entre elles d'un pas sur la carte. Cette carte est un objet **Map**, contenant un vecteur d'entiers en 2 dimensions. Le

Jeu sait à quelle entité correspond chacun de ses entiers. Les **Entités** possèdent une liste de **Cases**, simple objet chargé de stocker les deux coordonnées. Pour s'afficher sur un écran, ces différents objets appellent les fonctions de la classe **Screen**.

2.2 Description détaillée

Remarque : on peut trouver les diagrammes uml plus précis dans chaque descritpion de classe de la documentation Doxygen.

2.2.1 Utilitaires

Case : L'objet stocke des coordonnées en deux dimensions x et y. Elle contient des opérateurs de comparaison et de différence pour faciliter les opérations par la suite. Des fonctions utiles comme les extremums d'une liste de cases, les cases voisines ou une liste de cases contiguës de forme aléatoire ont aussi été ajoutées.

Map : Contient un vecteur de deux dimensions d'entiers, qui représentent les indices des entités. Elle contient des opérateurs d'affectation d'une valeur, d'affichage, mais aussi quantité de fonctions propres et amies très utilisées dans la suite. Parmis elles, la possibilité d'obtenir toutes les cases contigues à une autre, la possibilité d'obtenir la liste des listes de cases contigues dans une liste de cases, le renvoi d'une case aléatoire au milieu ou en bordure de la carte„ etc.

Screen : Se charge d'afficher les cases à l'écran avec la SDL. Même classe que le TP5, un constructeur et la possibilité d'afficher une case comme un disque ont été ajoutées.

2.2.2 Les Entités

Entité : Objet de base de la simulation, il est doté d'un identifiant unique et de la liste de ses cases. Il possède de nombreuses fonctions indiquant s'il est en activité, quelle est sa taille, son hostilité, ainsi que des opérations d'ajout et de suppression de **Cases**. De nombreuses méthodes amies, comme le calcul minimal de distances entre deux entités, ou avec toutes les entités d'un certain type, ont été ajoutées aussi. Le comportement de l'entité est défini dans la fonction *step()*, qui la fait avancer d'un pas.
Les classes filles de **Entité** utilisées sont dans le dossier *src/entites*

Vivante : Héritée de Entité, une vivante n'a pour fonction supplémentaire que celle de la reproduction. Elle indique lorsqu'elle doit se reproduire avec la fonction *should_greed()* et se reproduit en renvoyant ses enfants et en les ajoutant à la carte extérieure avec la fonction *get_children()*.
Les classes filles de **Vivante** utilisées sont dans le dossier *src/vivantes*.

2.2.3 Les jeux

La classe **Jeu** se charge de gérer le modèle et la simulation. Il contient toutes les Entités, la Map globale, le flux des statistiques. On peut lui ajouter, retirer des Entités vivantes ou non. Lors

de la simulation lancée par la fonction *sim()*, à chaque tour, il fait avancer toutes les entités d'un pas avec sa fonction *step()*, leur fait écrire leurs attributs dans son flux de statistiques, et les affiche sur un Screen.
Les classes filles de **Jeu** utilisées sont dans le dossier *src/jeux*.

3 Le Jeu standart

Nous envisagions d'abord de concevoir plusieurs types de simulations différentes, mais par manque de temps nous nous sommes concentrés sur un seul type assez général, le **Standart**. Le premier Jeu **ProjectileGame** que nous avons créé au début peut en fait être obtenu en réglant correctement les paramètres du **Standart**.

3.1 Principe

Entouré par un **Mur**, des **Projectiles** dangereux rebondissent dans tous les sens tandis que des **Fumee_test** toxiques se promènent aléatoirement. Dans ce contexte hostile, des **Fumee_esquive** et des **Esquive_vie** tentent de survivre malgré tout.

3.2 Entités en présence

3.2.1 Mur (en noir)

Le **Mur** est une **Entité** très simple, qui entoure une carte, s'affiche en noir, ne bouge pas et est hostile.

3.2.2 Projectile (en rouge)

Un **Projectile** est une **Entité** qui ne change pas de forme et qui se déplace avec une trajectoire rectiligne en rebondissant sur les bords de la carte. Sa vitesse et sa position initiale sont choisies aléatoirement au début du jeu. Il s'affiche en rouge et ne peut pas être discontinu.

3.2.3 Fumee_test (en gris)

La **Fumee_test** est la première **Entité** que nous avons écrite afin de tester un comportement de déplacement qui suggérait celui d'une fumée, mais qui aurait quand même une direction concrète. Elle a en mémoire deux **Cases** supplémentaires qui changent régulièrement et aléatoirement de place, *evil*, qu'elle doit fuir, et *good*, qu'elle doit suivre. Pour cela, elle parcourt d'abord sa frontière extérieure en ajoutant les bonnes **Cases**, puis sa frontière intérieure en supprimant les mauvaises. Plus précisément, une **Case** c a $\frac{dist(c,evil)}{dist(c,good)+dist(c,evil)}$ chances d'apparaître et $\frac{dist(c,good)}{dist(c,good)+dist(c,evil)}$ chances de disparaître.

3.2.4 Fumee_esquive (nuances de violet à turquoise)

Une **Fumee_esquive** est une **Vivante** chargée d'esquiver toutes les entités hostiles environnantes. Si elle en touche une, elle meurt immédiatement. À chaque tour, elle ajoute les cases de sa frontière extérieure les plus éloignées des fumées hostiles et retire les plus proches dans sa frontière. Elle possède de nombreux attributs fixés à sa naissance, comme le nombre de cases à ajouter et retirer à chaque tour, le nombre de cases en dessous et au dessus desquels on ne peut pas aller. À chaque reproduction, chacun des attributs sont mutés avec une distribution normale centrée sur celui de la classe mère, et dont la variance décroit avec l'âge de cette dernière. Cette astuce tend à favoriser les attributs des individus qui résistent plus longtemps.

3.2.5 Esquive_vie

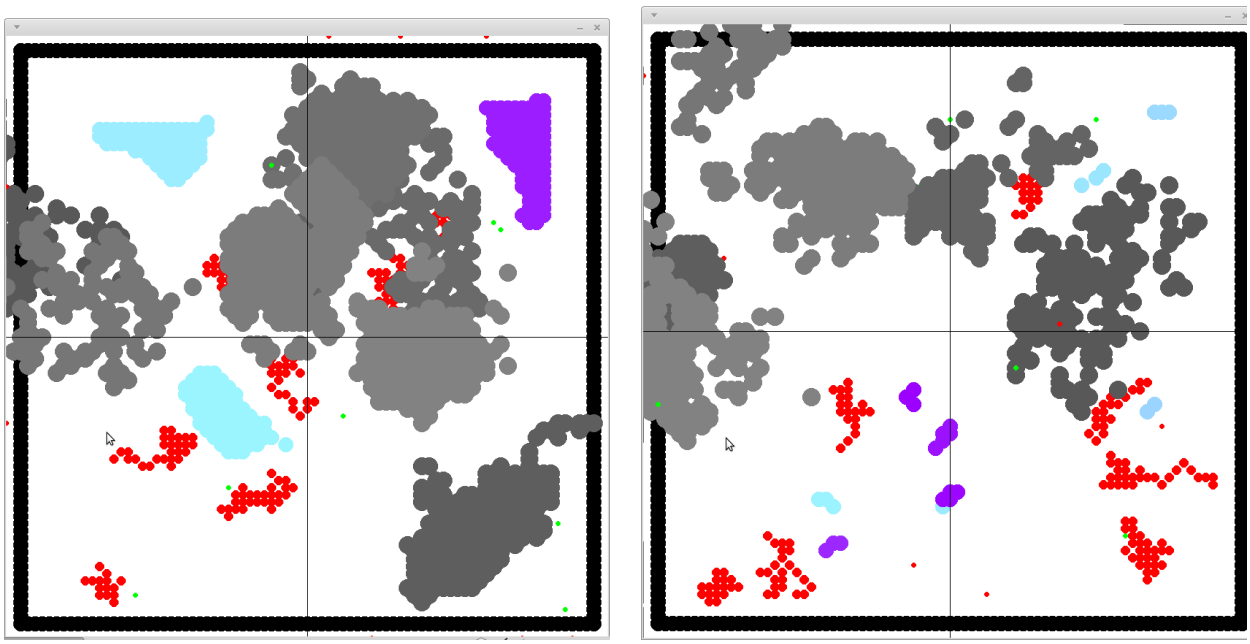
Une **Esquive_vie** est héritée de **Fumee_esquive** et n'ajoute qu'un seul point de différence, celui de posséder des points de vie qui augmentent en fonction de sa taille.

3.2.6 Mystherbe

Tentative malheureusement pas très intéressante pour l'instant. Voir plus sur la doc.

3.3 Résultats et interprétation

En plaçant des **Fumee_esquive** avec un nombre raisonnable de **Fumee_test** et de **Projectiles**, on peut remarquer au bout d'un certain temps une convergence des attributs vers une petite taille minimale et un nombre d'ajouts de cases très inférieur à celui des suppressions de cases, qui permet d'avoir la petite taille et l'agilité pour esquiver les autres fumées hostiles. Les règles par défaut en montrent un bon exemple. Plus amples graphiques (3) en annexe.



On peut voir ici une convergence vers des petites entités agiles. Remarque : parfois toutes les fumées meurent avant d'avoir pu converger, il fautparfois relancer plusieurs fois la simulation.

4 Utilisation

4.1 Documentation doxygen

Une documentation a été générée automatiquement avec Doxygen. Toutes les fonctions et classes y sont décrites en détail. Lien :

En local : darwiniansmoke/public/index.html
Sur le git : https://acquilles.frama.io/darwiniansmoke

Pour générer la doc, se placer dans darwiniansmoke et taper : make dox.

4.2 Installation git

Lien du git : https://framagit.org/Acquilles/darwiniansmoke.git

4.3 Jouer

4.3.1 Compilation

Se placer dans le dossier *src* et taper make. La compilation peut-être un peu longue (15 secondes).

4.3.2 Changement des règles

Le type de simulation qu'on peut lancer dépend d'un ensemble de règles, comme le nombre de projectiles, les caratéristiques des fumées initiales, la taille de la carte, etc. Ces règles sont modifiables très simplement :

On commence par choisir le type de règles qu'on veut changer, par exemple la règle du jeu **Standart** dans le dossier *src/standart*. Ensuite, on peut modifier les règles des différents éléments dans les fichiers *.csv* présents. On choisit dans le fichier *rules.csv* la taille de la carte, le nombre des différentes entités qu'on veut placer et le nom des fichiers contenant leurs caractéristiques. Ces fichiers se trouvent dans le même dossier, et les caractéristiques qu'ils présentent y sont expliquées. Celles-ci sont généralement comprises entre deux bornes qu'on choisit.

4.3.3 Éxécution

On se place dans le fichier *src*, dans lequel il y a l'exécutable. On le lance en ajoutant l'option du type de **Jeu** qu'on veut, par exemple :
./smoke standart

Pour faire avancer le jeu, appuyer sur Entrée dans le terminal, pour quitter taper q et pour restart taper r.

4.4 Statistiques

Afin de pouvoir observer les effets de la sélection naturelle à travers l'évolution statistique des attributs des entités vivantes, nous avons produit des graphiques. A chaque tour on enregistre les différents attributs (fonctions *write_stat()* de **Jeu** et de **Entité**) dans un fichier csv (le flux *stats* de **Jeu**) , qui peut-être lu simplement à la fin de la simulation avec Python. Nous pouvons ainsi afficher des graphiques avec matplotlib. Pour les obtenir, se placer dans *stats* et exécuter le code *plotstats.py* avec Python3. Avec une simulation **Standart** contenant 5 **Fumee esquive**, un **Mur**, 5 **Projectile** et 5 **Fumee test** on obtient les figures en Annexe (3).

Remarque : pour que les statistqiues ne soient pas faussées, il est préférable de ne mettre qu'un seul type d'entité dans la simulation, sinon ce ne snont pas les mêmes attributs.

4.5 Performances

Nous avons utilisé l'outil gprof pour comprendre quelles étaient les fonctions dans lesquelles l'exécution passaient le plus de temps. Pour nos premières simulations, ce sont clairement les calculs de distances entre les entités, car il faut pour chaque case de chaque entité calculer la distance avec chaque case de toutes les autres entités.

Pour utiliser gprof, se placer dans le répertoire *src* et taper :
make perf
qui compilera avec gprof. Exécuter ensuite avec l'option de votre choix.

4.6 Résumé

— **Installer :**
git clone https ://framagit.org/Acquilles/darwiniansmoke.git

— **Compiler :** Dans src
make

— **Changement de règles (optionnel) :**
cd src/<type de jeu>
Puis changer les règles correspondantes dans un éditeur de texte.

— **Exécution :**
./smoke <type de jeu> , par exemple ./smoke standart

Dans le terminal : rien pour avancer, **q** pour quitter et **r** pour restart.

— **Statistiques :** Dans stats
python3 statistiques.py
Ou ouvrir le Jupyter notebook dans le même fichier.

— **Performances :** Dans src
make perf

./smoke <type de jeu>
Ouvrir le fichier perf.txt

— **Doxygen :** Dans darwiniansmoke
make dox
Ouvrir public/index.html avec un navigateur

5 Remarques

5.1 Types de conteneurs

Afin de minimiser le temps de calcul, nous avons favorisé les passages par référence constante. Nous avons fait le choix des std : :list pour la majorité de nos conteneurs, car les utilisations principales sont l'ajout à la fin et le parcours sans ordre déterminé.

5.2 Fuites Valgrind

De nombreuses fuites apparaissent avec Valgrind, mais après vérification elles sont toutes dues à la bibliothèque SDL. Nous avons vérifié la destruction de toutes les instances que nous créons. Une partie du résultat de la commande :

valgrind --show-leak-kinds=all --leak-check=full

se trouve dans la Figure2.

```
==9911==
==9911== HEAP SUMMARY:
==9911==     in use at exit: 270,377 bytes in 1,236 blocks
==9911==   total heap usage: 60,688 allocs, 59,452 frees, 2,131,975 bytes allocated
==9911==
==9911== 2 bytes in 1 blocks are still reachable in loss record 1 of 327
==9911==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==9911==    by 0x66E990D: _XlcCreateLC (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x67066BF: _XlcDefaultLoader (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66F0EAD: _XOpenLC (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66F10BA: _XrmInitParseInfo (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66D8DDF: ??? (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66DC3FD: XrmGetStringDatabase (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66B8CE3: ??? (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66B8F8D: XGetDefault (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0xB056059: _XcursorGetDisplayInfo (in /usr/lib/x86_64-linux-gnu/libXcursor.so.1.0.2)
==9911==    by 0xB0560B8: XcursorSupportsARGB (in /usr/lib/x86_64-linux-gnu/libXcursor.so.1.0.2)
==9911==    by 0xB058B30: XcursorNoticeCreateBitmap (in /usr/lib/x86_64-linux-gnu/libXcursor.so.1.0.2)
==9911==
==9911== 2 bytes in 1 blocks are still reachable in loss record 2 of 327
==9911==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==9911==    by 0x59FF489: strdup (strdup.c:42)
==9911==    by 0x66E641F: _XlcResolveLocaleName (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66E9B17: ??? (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66E90A2: ??? (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66E98DB: _XlcCreateLC (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x67066BF: _XlcDefaultLoader (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66F0EAD: _XOpenLC (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66F10BA: _XrmInitParseInfo (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66D8DDF: ??? (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66DC3FD: XrmGetStringDatabase (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==    by 0x66B8CE3: ??? (in /usr/lib/x86_64-linux-gnu/libX11.so.6.3.0)
==9911==
```

FIGURE 2 – Fuites Valrgind dues à la SDL

6 Annexes

Quelques exemples de graphiques qu'on obtient avec les plotstats.py au cours d'une simulation standart dont la seule entité vivante est une Fumee_esquive (règle par défaut). On obtient

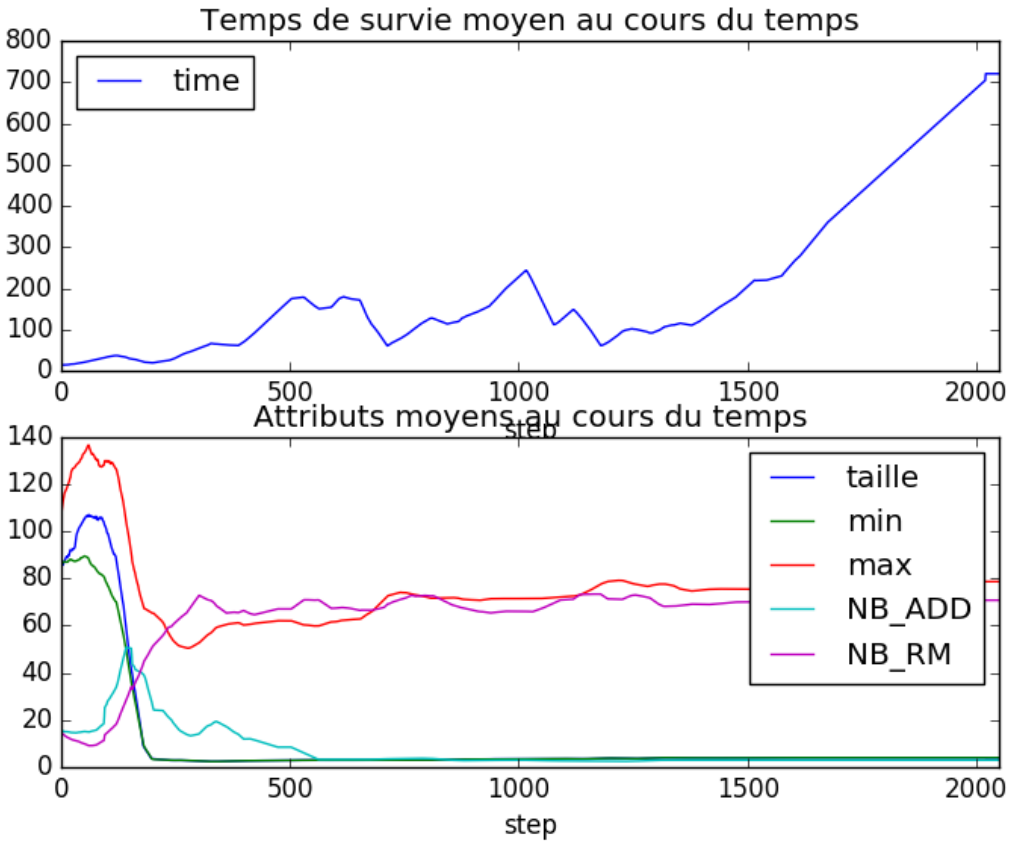


FIGURE 3 – Convergence globale des attributs.

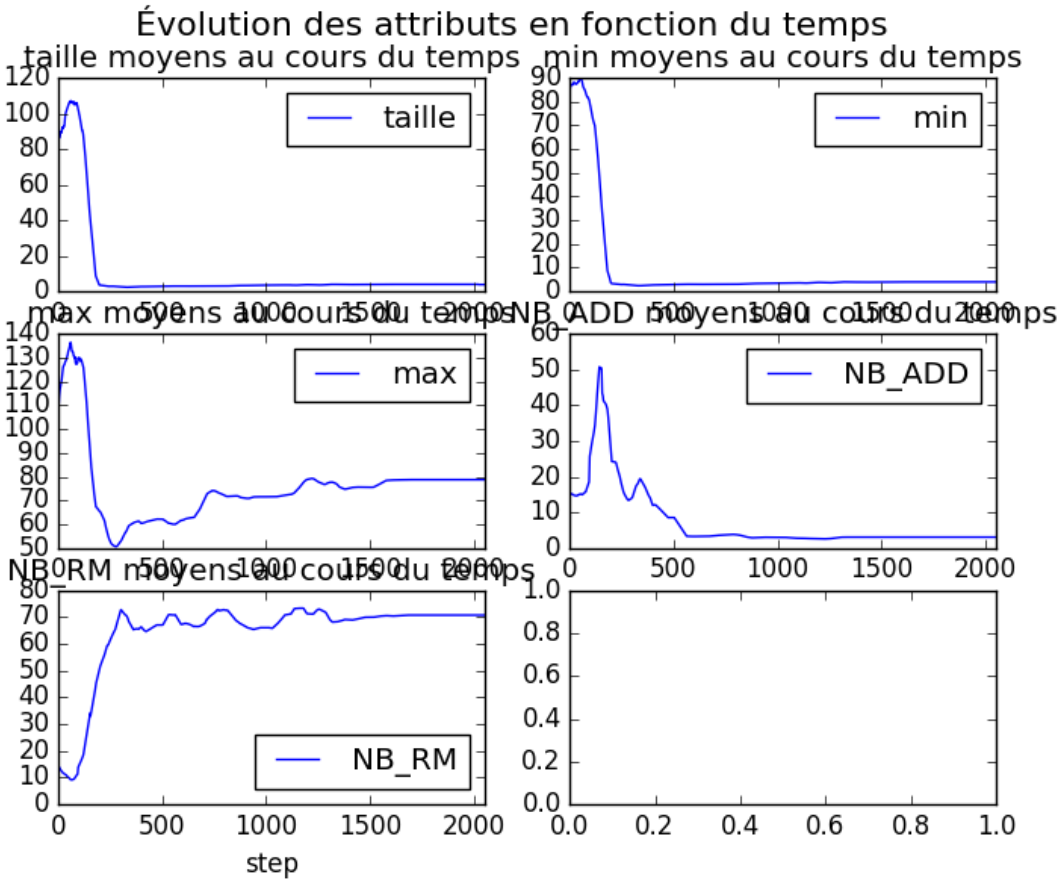


FIGURE 4 – Convergence précise des attributs

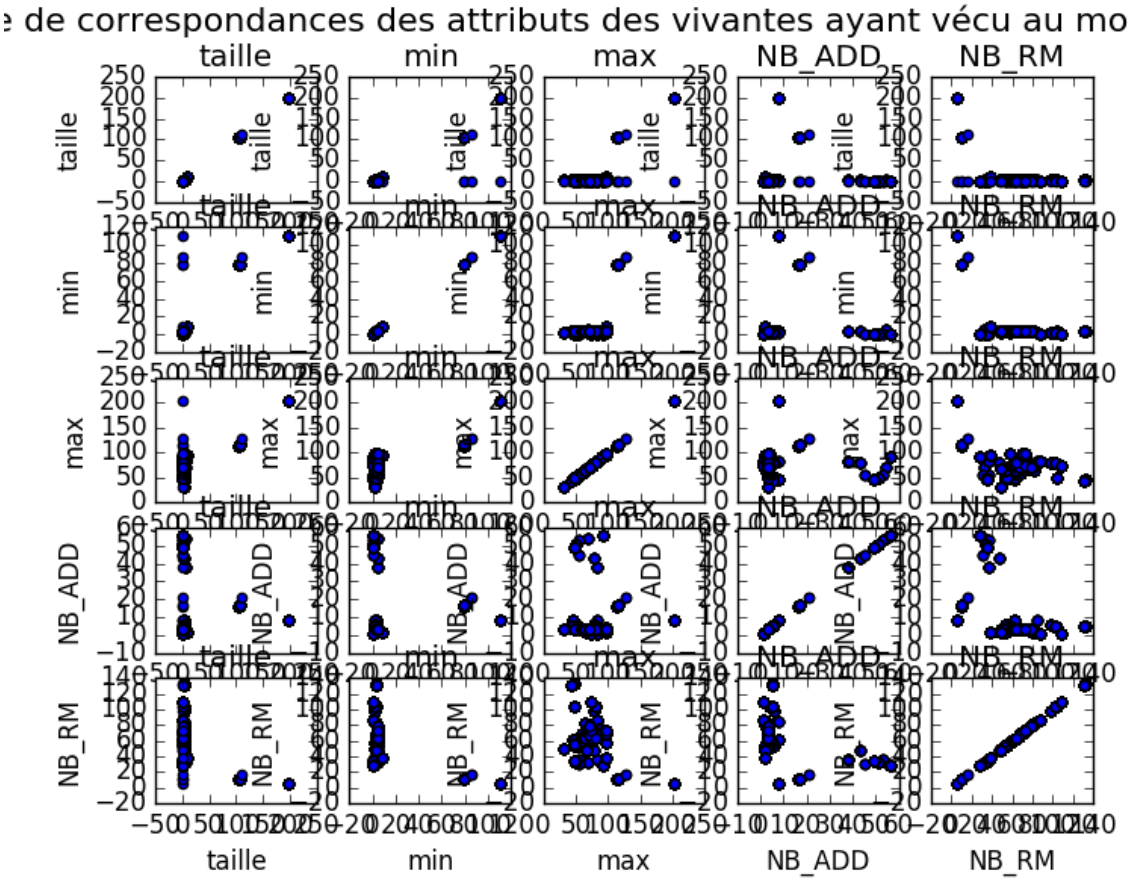


FIGURE 5 – Matrice de correspondance des attributs des vivantes ayant vécu au moins 50 steps.

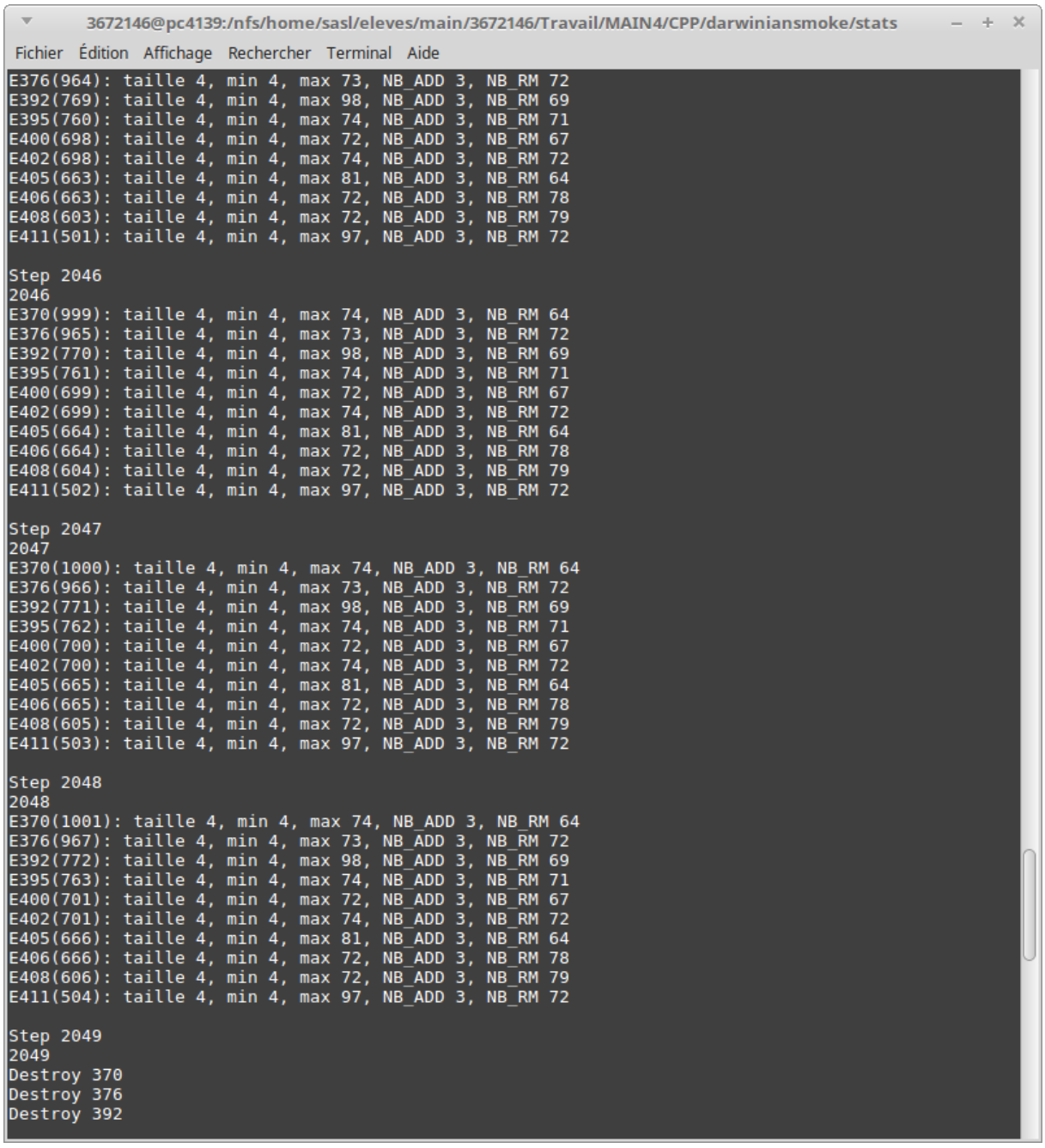


FIGURE 6 – Les attributs des Fummee_esquive à la fin de la simulation