BIGORRE Noé, CAPMARTIN Léo, DURFORT Alexis, MASSET Aurélien

Groupe Segfault

Rapport de soutenance 2

Introduction

Le groupe Segfault est constitué de 4 membres de Strasbourg et Toulouse. A cause de circonstance hors de notre contrôle, nous avons tous dû couper court à notre semestre à l'étranger et revenir en France pour participer au semestre 4 d'EPITA. C'est dans ce contexte qu'a vu le jour le groupe Segfault.

Le but de notre projet est de réaliser un simulateur d'évolution accompagné d'un système d'édition de la simulation pour permettre de recréer une multitude de scénarios avec un unique programme.
Pour cette soutenance, nous avons bien avancé sur ce que nous avions prévu. Depuis la dernière soutenance, nous avons peaufiné et amélioré ce que nous avons déjà implémenté. De plus, nous avons ajouté les parties manquantes telles que les statistiques ou le lien entre l'interface et la simulation.

Sommaire/

Introduction

I. Moteur

II. Agents
    A. Mouvement
    B. Reproduction
    C. Maladies

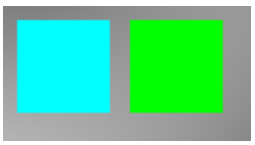
III. Le Terrain
    A. Génération de la nourriture
    B. Interaction entre les agents et le terrain

IV. Interface
V. Statistiques
    A. Généralités
    B. Enregistrement des données
VI. Site web

Conclusion

I. Moteur

Lors de la précédente soutenance, notre moteur nous permettait de mettre la simulation en pause, d'accélérer la vitesse ou de la ralentir. Cependant les boutons permettant d'effectuer ces actions étaient monochromes, ce qui n'aide évidemment pas à la compréhension du fonctionnement de notre application.



Anciennes versions des boutons

Pour remédier à ce problème nous avons décidé d'utiliser des images permettant de savoir rapidement quel est le rôle d'un bouton.

Différentes versions des nouveaux boutons

Lors de l'implémentation nous nous sommes heurté à un problème majeur. Lorsque nous mettions en pause la simulation, lors du clic sur un bouton, la nouvelle image de celui-ci venait se superposer sur l'ancienne image au lieu de la remplacer. Il est important de noter que ce problème intervient uniquement lorsque la simulation est en pause.

Exemple de boutons qui se superposent

Le problème vient du fait que nous dessinons toutes les frames sur la même image que celle qui est rendue à l'écran. Lorsque la simulation est en marche, la nouvelle image vient recouvrir celle des boutons et nous affichons ensuite les boutons par-dessus. Cependant dès que nous mettons en pause la simulation, nous ne mettons pas à jour la simulation et donc nous ne redessinons pas de nouvelle image, par conséquent au moment de dessiner les boutons, ils viennent s'afficher au-dessus des précédents.

Pour corriger le problème, nous utilisons donc deux images : une pour dessiner toutes les mises à jour chaque frame et une qui sera affichée toutes les frames. Pour ce faire, l'image des mises à jour est dessinée sur celle qui est affichée puis nous dessinons les boutons sur l'image à afficher. Le fait de réaliser ces deux étapes quoi qu'il arrive (que la simulation soit en pause ou non) nous permet d'éviter ce bug.

Nous avons aussi ajouté un bouton "stop" permettant, comme son nom l'indique, d'arrêter la simulation.

Bouton "stop"

II. Agents

A. Mouvement

Le fonctionnement des mouvements des agents n'a fondamentalement pas changé, la fonction *agentBehavior* va choisir si l'agent peut se reproduire, sinon, s'il peut se nourrir et s'il ne peut pas faire ces deux choses-là, il va alors se "balader" en quelque sorte. Et c'est justement cette fonction que nous avons modifiée, auparavant cette fonction prenait un point aléatoirement dans le terrain et faisait avancer l'agent vers celui-ci. Une fois que l'agent avait atteint ce point, un nouveau était choisi et ainsi de suite. L'inconvénient de cette technique est que l'agent avait souvent besoin de traverser une grande partie du terrain avant d'atteindre son point ce qui, de notre point de vue,

n'était pas très naturel. Régler ce problème était heureusement assez simple, au lieu de prendre un point aléatoirement sur le terrain, on va prendre un point aléatoirement autour de l'agent pour qu'il puisse changer de point plus souvent et éviter de parcourir tout le terrain inutilement. Cependant un inconvénient de cette technique est que les agents auront plus tendance à rester autour de leur position de départ.

B. Reproduction

Pour la reproduction, nous avions déjà implémenté une grosse partie de cet aspect durant la dernière soutenance. Pour rappel, la reproduction peut se faire de deux façons différentes, la première façon est la reproduction sexuée, où deux agents vont produire un ou plusieurs agents qui vont avoir des caractéristiques différentes de leurs parents selon deux manières différentes. Soit les caractéristiques suivent un pattern d'héritage, c'est-à-dire que chaque caractéristique de l'enfant sera la caractéristique du parent 1 ou 2, soit elles suivent un pattern de dérive génétique ou les caractéristiques dérivent légèrement par rapport au parent 1 ou au parent 2. Nous nous étions arrêtés là pour la première soutenance et n'avions pas fait la deuxième façon qui est la reproduction asexuée des agents. Nous l'avons alors implémenté pour cette soutenance, cela n'a pas été aussi difficile que pour la reproduction sexuée étant donné que nous avons principalement utilisé des fonctions déjà écrites. Le principe est le même que celui d'avant, sauf que nous n'avons pas deux différents agents donc nous ne pouvons qu'effectuer la dérive génétique sur la descendance de l'agent ce qui facilite grandement les choses.

C. Maladies

L'implémentation des maladies dans les agents est faite par le biais de la *struct sickness* qui permet de contrôler les différentes maladies et les interactions entre elles.

Cette structure comporte les éléments suivants :

Le nom de la maladie	
l'infectiosité	Représente la chance qu'une maladie se propage à un agent à proximité.
La sévérité	représente le coût additionnel en énergie qu'un agent doit fournir à cause de la maladie.
La mortalité	représente la chance qu'un agent a de mourir après une action.
canInfect	la liste des agents que la maladie peut infecter.

Les différentes maladies des agents sont représentées dans une liste chainée se comportant comme un ensemble pour éviter les duplications de maladies sur un même agent.

III. Terrain

A. Génération de la nourriture

Pour la première soutenance, nous nous étions arrêtés à l'ajout d'un système de bruit de perlin pour créer un terrain où nos agents peuvent évoluer. Pour cette soutenance nous avons donc ajouté la génération de la nourriture au terrain. L'idée est la suivante, nous allons en premier temps générer des arbres aléatoirement sur le terrain, puis à chaque tick de la simulation, ces arbres vont générer des fruits qui pourront être mangés par les agents herbivores.

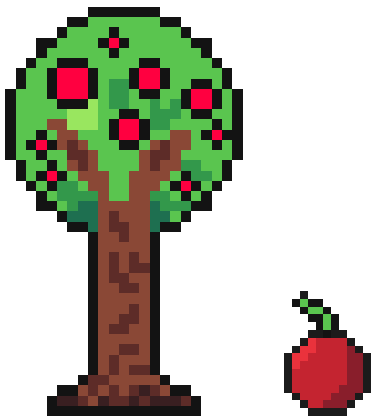
Pour l’implémentation nous avons utilisé plusieurs structures : d’abord la structure pour les arbres qui possède 4 caractéristiques :

Xpos	Position en X
Ypos	Position en Y
radius	Rayon d'apparition des fruit
currentFood	Nombre de fruit actuellement autour de l’arbre

Nous avons ensuite une deuxième structure pour les fruits qui possède comme caractéristique:

Xpos	Position en X
Ypos	Position en Y
energyToGive	Quantité d’énergie a donné à l'agent si il mange ce fruit
lifeSpan	Durée de vie du fruit

Nous avons une dernière structure qui n’est ni plus ni moins qu'une liste pour accéder au différent fruit actuellement sur le terrain.
Pour dessiner les arbres et les fruit nous avons aussi créer des sprite nous même:



B. Interaction entre les agents et le terrain

Le lien entre le terrain et l'agent a été ajouté au niveau du climat, ce qui permet d'augmenter le coût en énergie des actions quand l'agent se trouve dans un climat trop différent de son climat préféré.

Plus la différence entre le climat et celui préféré de l'agent est important, plus le coût additionnel en énergie est important, car on calcule le carré de la différence entre les deux. Ainsi, pour de petites différences, le changement du coût est relativement faible, mais pour de grandes différences, le coût peut sérieusement grimper.

IV. Interface

Afin de pouvoir créer et modifier facilement des agents, nous avons, à l'aide de Glade, et de Gtk-3.0, confectionné une interface utilisateur simple et efficace.

À gauche se situe une liste des unités, plantes et maladies déjà existantes, ainsi qu'un bouton permettant d'en créer une nouvelle. Enfin, un bouton permettant de lancer la simulation est également présent.

À droite se situent, sur 2 colonnes, les informations d'un agent, d'une plante ou d'une maladie qui sont modifiables. Ces dernières sont développées dans leurs parties respectives, développées précédemment.

Les boutons sont bels et bien reliés, lançant des signaux permettant de faire l'action demandée. Lors de la création, un bouton final a été rajouté, afin que l'on ne récupère les informations choisies qu'à ce moment-là pour envoyer le tout vers un constructeur.

L’interface reliant le signal du bouton pour lancer la simulation à une fonction.

```
void call_run(GtkButton *button, gpointer user_data)
{
```

```
GtkButton* run = GTK_BUTTON(gtk_builder_get_object(builder, "Run"));
g_signal_connect(run, "clicked", G_CALLBACK(call_run), NULL);
```

Extraits des Signal handler de notre interface

```
<object class="GtkButton" id="Run">
  <property name="label" translatable="yes">Run Simulation</property>
  <property name="visible">True</property>
  <property name="can-focus">True</property>
  <property name="receives-default">True</property>
  <property name="valign">start</property>
  <property name="margin-start">4</property>
  <property name="margin-end">4</property>
  <property name="margin-bottom">2</property>
</object>
```

Extrait du code de notre interface, correspondant au signal traité précédemment

Les menus déroulants se mettent à jour en fonction des nouveaux éléments créés, ainsi que de leur type *ItemType*, nouveau paramètre ajouté, propre au comportement de l’interface, afin qu’elle puisse séparer les agents des plantes et des maladies.

ItemType est un paramètre choisissable lors de la création d’un nouvel agent. Il n’est plus modifiable après. Ainsi, il est impossible de transformer une plante en maladie.

V. Statistiques

A. Généralités

Notre projet donne aux statistiques une grande importance : en effet, sans cette partie, notre projet n'aurait que peu d'intérêt.
À chaque intervalle de temps de x secondes, nous enregistrons l'état de la simulation ce qui permet par la suite de pouvoir constater les évolutions de nos espèces. Pour ce faire, nous avons une liste chainée dans laquelle, à chaque intervalle de temps, nous enregistrons une copie de l'état de la simulation. Il nous suffit alors de parcourir cette liste pour avoir une idée de l'évolution de la simulation.

B. Enregistrement des données

Une fois la simulation terminée, nous parcourons la liste pour enregistrer proprement toutes les données dans un fichier CSV qui pourra ensuite être lu par un logiciel comme excel (ou équivalent) pour effectuer tous les traitements que l'utilisateur souhaite.

```
time (secondes),1,2,3,4,5,6,7

espece 1
gene 1,56,64,12,0,-22,-44,-66
gene 2,56,84,23,21.333333333333329,4.8333333333333286,-11.666666666666671,-28.166666666666671
...

espece 1
gene 1,56,64,12,0,-22,-44,-66
gene 2,56,84,23,21.333333333333329,4.8333333333333286,-11.666666666666671,-28.166666666666671
...
```

Données après le traitement dans un document CSV

	A	B	C	D	E	F	G	H	I
1	time (secondes)	1	2	3	4	5	6	7	
2									
3	espece 1								
4	gene 1	56	64	12	0	-22	-44	-66	
5	gene 2	56	84	23	21,33333	4,833333	-11,66667	-28,16667	
6	...								
7									
8	espece 1								
9	gene 1	56	64	12	0	-22	-44	-66	
10	gene 2	56	84	23	21,33333	4,833333	-11,66667	-28,16667	
11	...								

Données affichées dans un tableau excel

VI. Site Web

Pour notre site, nous avons décidé de réaliser un design s'inspirant d'un terminal Linux.
https://leocapmartin.github.io/segfaultsite.github.io/

Plusieurs commandes sont disponibles permettant de naviguer sur le site. En voici une liste non exhaustive :

- help : permet d'afficher les commandes importantes ainsi que leur description.
- about : permet d'afficher une courte description du projet ainsi que de notre groupe
- clear : permet d'effacer tout ce qui est affiché sur le terminal
- files : permet d'afficher plusieurs liens permettant de télécharger nos rapports ou notre projet
- github : redirige vers notre page github et affiche le lien du github si l'utilisateur à un bloqueur de pop-up
- logo : affiche le logo de notre groupe
- welcome : affiche le message de bienvenue de notre site

Pour développer notre site nous avons utilisé du HTML, CSS et JavaScript très simple.

Conclusion

L’ambiance au sein du groupe est très bonne et nous avons pu nous organiser sans problème pour nous répartir efficacement les tâches. Malgré quelques contretemps, nous sommes globalement fiers de notre travail et tout ce que nous imaginions initialement a été implémenté avec succès.