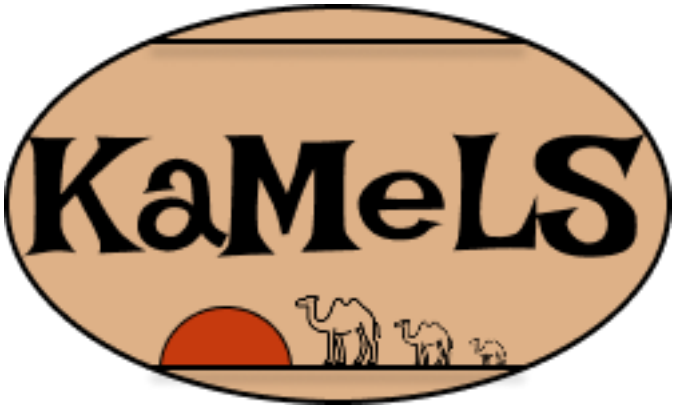
Rapport de Seconde Soutenance - Projet S4

Hayvolution

Groupe : KaMeLS



Kylian Djermoune Maxime Trimboli Lino Joninon Simon Campredon

Table des matières

1 Introduction

1.1 Rappel du projet

Hayvolution est un projet de simulation d'évolution d'écosystème qui possède ses propres règles, un jeu de la vie beaucoup plus poussé. Au sein de cet environnement représenté en caractères ASCII, différentes espèces animales évoluent entre elles. L'objectif est de réaliser une simulation que l'utilisateur pourrait entièrement personnaliser ; à savoir l'environnement et les animaux, pour voir comment ses choix influencent le déroulé de la simulation. Les animaux ayant tous des caractéristiques différentes, pouvoir observer leurs évolutions au fil des générations et essayer de les prédire aura un intérêt certain pour l'utilisateur. Dans ce projet se mélangeront des notions de génération aléatoire, que ce soit pour générer un environnement au début de la simulation mais aussi pour provoquer des mutations au sein des espèces d'une génération à l'autre. Un des objectifs du projet sera de simuler l'intelligence des animaux, pour cela nous utiliserons plusieurs algorithmes couplés, idéalement nous aimerions que les espèces s'adapte aussi de par leurs comportements, et que donc ces adaptations soient transmissibles aux générations suivantes. Enfin, nous souhaitons à terme avoir des interfaces graphiques agréables pour voir se dérouler la simulation, ainsi qu'une interface dédiée à des statistiques sur la simulation pour observer globalement l'évolution des espèces génération par génération.

1.2 Objet d'étude

1.2.1 Aspect technique

Pour ce projet nous devons aborder des problématiques de codage qui nous permettent de mettre à l'épreuve nos connaissances mais également de rester réalistes dans nos capacités actuelles. Nous avons donc décidé d'approfondir ce que nous avons appris au semestre précédent et de s'essayer à de nouveaux aspects de la programmation. Pour l'interface nous réutiliserons ce que nous avons appris avec GTK ainsi que SDL. Cependant notre vision finale du projet nous demandera peut-être de s'essayer à d'autres moyens d'affichage. L'environnement affiché, lui, sera lu à partir d'une matrice. Nous allons pouvoir utiliser la manipulation de matrice que nous connaissons. Par la suite, il nous faudra également afficher les animaux, leurs interactions et l'environnement, ceci se fera à travers des structs et du code plus classique mais non moins important au projet. Enfin, le comportement des animaux doit-être défini. Dans un premier temps il sera fait à partir d'un algorithme intelligent mais ensuite nous souhaitons le remplacer par un réseau de neurones. Cette partie du projet est certes ambitieuse mais nous sommes confiants que l'expérience accumulée grâce au projet OCR nous sera d'une grande aide. Le site internet est une bonne occasion pour parfaire et maintenir nos compétences en HTML. Finalement, ce projet à la différence du projet OCR ne contient pas des parties clairement découpées mais un ensemble de tâches que nous nous sommes fixées. Il sera donc impératif de pouvoir communiquer clairement et efficacement afin d'éviter les confusions ou les malentendus tout au long du projet. Ce point est encore plus crucial car les différentes

parties, telles que celles des animaux et celle de l'environnement se chevauchent. L'organisation de notre projet est donc bien plus importante que pour les projets précédent, ceci est dû au fait que les différentes soutenances ne contiennent plus des objectifs prédéfinis mais des objectifs que nous nous sommes donnés nous-mêmes. L'esprit d'équipe est donc un aspect sur lequel il nous faudra travailler davantage.

1.2.2 Intérêt culturel

Cette simulation permettra de récréer, de manière bien plus simpliste, des phénomènes qui ont été constatés dans la vie réelle. L'évolution des prédateurs au fil des générations grâce à la sélection naturelle pour filtrer ceux dont les caractéristiques facilitent la chasse d'autres animaux. Et il en va de même pour les herbivores qui s'adapteront au fur et à mesure pour être capable de survivre à ces menaces. Les différents environnements présents pourront également, selon le déroulé des simulations, mener à la spéciation des espèces (divers groupes d'animaux possédant un ancêtre commun mais ayant des traits et caractéristiques bien différentes) ou encore observer une convergence évolutive de deux espèces non semblables originellement.

1.3 Présentation du groupe

Simon Campredon
Pour le projet OCR de S3, je me suis occupé du réseau de neurones, cela a été bien difficile mais j'ai tout de même de bons souvenirs car il m'a permis de me découvrir une nouvelle passion pour l'intelligence artifi-

cielle. C'est pourquoi ce projet est pour moi important car il me permettra possiblement d'élargir mes connaissances déjà acquises ou bien d'en découvrir de nouvelles.

Kylian Djermoune

Je suis impatient de travailler sur ce projet de S4, l'idée d'un simulateur d'écosystème m'ayant directement emballé. La partie dont je m'occupe, l'environnement et notamment sa génération procédurale m'apporteront des compétences nouvelles en programmation. Je me suis occupé de la partie de création de l'environnement selon une taille variable et du début de la génération aléatoire des éléments au sein de cet environnement.

Lino Joninon

Lors de projets que nous avons effectués précédemment, j'ai senti un manque d'organisation dans les groupes dont je faisais partie. Nous obtenions un résultat fini, mais qui aurait pu être fait plus facilement et plus rapidement avec plus de communication et d'organisation. Donc, en plus de développer mes compétences de codage, je souhaite approfondir et affiner mes capacités à travailler et à interagir au sein d'une équipe. Ma contribution pour le projet est l'interface et lier les différentes parties avec l'interface. Comparer au projet OCR une difficulté supplémentaire dans l'interface est l'affichage constant de l'environnement.

Maxime Trimboli

Nous allons de nouveau réaliser un projet libre, et après le projet S2 et l'expérience que nous avons acquise en programmation, j'ai hâte de voir ce que nous allons produire. De plus, un des enjeux de ce projet est de pouvoir reproduire des concepts évolutifs réels et de les modéliser, et j'ai hâte de revenir à mes cours de SVT du lycée pour voir ce que nous pourrons rajouter à la simulation.

2 Tâches effectuées

2.1 Environnement

Pour la première soutenance, nous nous étions occupés de la base quant à la création et la gestion de la partie relative à l'environnement. Nous utilisions une struct possédant trois attributs ; une matrice de caractères dont la longueur et la largeur sont également des attributs de la structure. Cependant, la matrice de caractère n'étant finalement pas viable pour lier les animaux et l'environnement, nous l'avons donc remplacée par une matrice de struct Case. Ces struct Case ont, elles aussi, trois attributs ; un pointeur vers un animal, un caractère représentant le décor (arbres) et un caractère représentant le biome. Ainsi, sur une case de coordonnées (x, y) peuvent se retrouver trois éléments. Cela a amené à la modification de la fonction « print_environment » qui désormais peut être appelée avec un paramètre influant sur la couche à afficher.

Différentes fonctions utilisables sur les environnements sont disponibles.

— -new_environment : La fonction « newEnvironment » qui permet de créer un Environment, s'assurant d'allouer l'espace nécessaire à celui-ci puis d'initialiser ses attributs length et width selon les deux arguments passés. En ce qui concerne l'initialisation de son attribut mat, elle fait appel à la fonction « createMat ».

— -createMat : Cette fonction crée simplement la matrice de caractères en allouant dynamiquement l'espace nécessaire à celle-ci selon les deux arguments passés à la fonction.

— -print_environment : Cette fonction a donc été modifiée pour pouvoir afficher soit les animaux, soit le décor (arbres), soit les biomes (désert, etc) ou alors la couche la plus haute n'étant pas vide pour chaque case, dans cet ordre : animal - décor - biome.

— -free_environment : Une simple fonction libérant l'espace alloué à l'environnement et la matrice de caractères lui étant associée.

— -fill_environment : Cette fonction, actuellement en cours de modification, permet de remplir la matrice de caractères de l'environnement passée en argument. A l'aide d'un algorithme de génération de bruit de Perlin qui utilise une table de permutation, elle pourra générer des biomes de manière procédurale et cohérente. En fonction de la valeur de bruit assignée à chaque case après l'algorithme, un biome sera choisi.

— -spawn_animals : Elle est appelée au début de la simulation pour faire apparaître les animaux au sein de l'environnement. Elle utilise

```c
// Print the layer of the environment
// Mode = 0: Animals
// Mode = 1: Decors (Trees, Mountains)
// Mode = 2: Biomes
// Mode != {0, 1, 2}: Animals if it exists, else Decors if it exists, else Biomes
void print_environment(Environment* env, int mode)
{
    for (size_t i = 0; i < env->length; i++)
    {

        for (size_t j = 0; j < env->width; j++)
        {
            if(mode == 0)
            {
                if(env->mat[i][j].animal != NULL && env->mat[i][j].animal->status != STATE_DEAD)
                    printf("%c ", env->mat[i][j].animal->species);
                else
                    printf(" ");
            }
            else if(mode == 1)
            {
                if(env->mat[i][j].tree != NULL)
                    printf("T");
                else
                    printf(" ");
            }
            else if (mode == 2)
                printf("%c ", env->mat[i][j].biome);
            else
            {
                if(env->mat[i][j].animal != NULL && env->mat[i][j].animal->status != STATE_DEAD)
                    printf("%c", env->mat[i][j].animal->species);
                else if(env->mat[i][j].tree != NULL)
                    printf("T");
                else
                    printf("%c ", env->mat[i][j].biome);
            }
        }
        printf("\n");
    }
}
```

Figure 1 – Fonction print_environment

la liste chaînée des espèces de base. Elle fait appel autant de fois que souhaité (selon le paramètres *populationNumbers) à la fonction « reproduce » qui s'assure de créer les animaux autour du premier sans qu'ils se retrouvent sur une même case.

2.2 Arbres, listes et files

Comme nous l'avions décidé à la dernière soutenance nous avons implémenté la sauvegarde des animaux sous forme d'arbres. Au lieu de garder en mémoire tous les animaux sous une forme de liste linéaire, cette

```c
void spawn_animals(Environment* env, List* species, int* populationNumbers)
{
        species = species->next;
        int n = 0;
        while(species)
        {
                for(int i = 0; i < populationNumbers[n]; i++)
                {
                        Animal* an = (Animal*)species->element;
                        reproduce(an, env);

                }
                species = species->next;
                n++;
        }
}
```

Figure 2 – Fonction spawn_animals

implémentation en arbre nous permet de garder l'information de toute la lignée d'animaux.

Nous avons rajouté un attribut Sons, qui est une liste chaînée dans la struct animals et à laquelle on va rajouter les fils d'un animal lorsque celui-ci se reproduit. Comme nous l'avions anticipé, nous ne pouvons plus free un animal dès qu'il meurt. Nous avons donc aussi rajouté un attribut booléen state qui permet de marquer les animaux morts pour ne pas les faire jouer et ne plus les considérer dans le déplacement des autres animaux et dans l'affichage de l'environnement.

Nous avons donc aussi du retravailler sur notre fichier list.c, qui contenait une implémentation de listes chaînées avec sentinelles. Nous avons passé le type de la valeur de chaque élément de la liste en (void*) car nous allons avoir besoin de faire des listes d'arbres notamment. Nous avons aussi dû déterminer dans quel ordre faire jouer nos animaux : nous avions deux choix ; le premier, faire jouer tous les animaux d'une même espèce, puis ceux de l'espèce suivante, comme une sorte de parcours profondeur.

Ensuite, un parcours largeur de l'arbre ce qui correspond à faire jouer les plus vieilles générations d'abord indistinctement de leur espèce. Dans un élan de respect pour les personnes âgées, nous avons choisi la deuxième option.

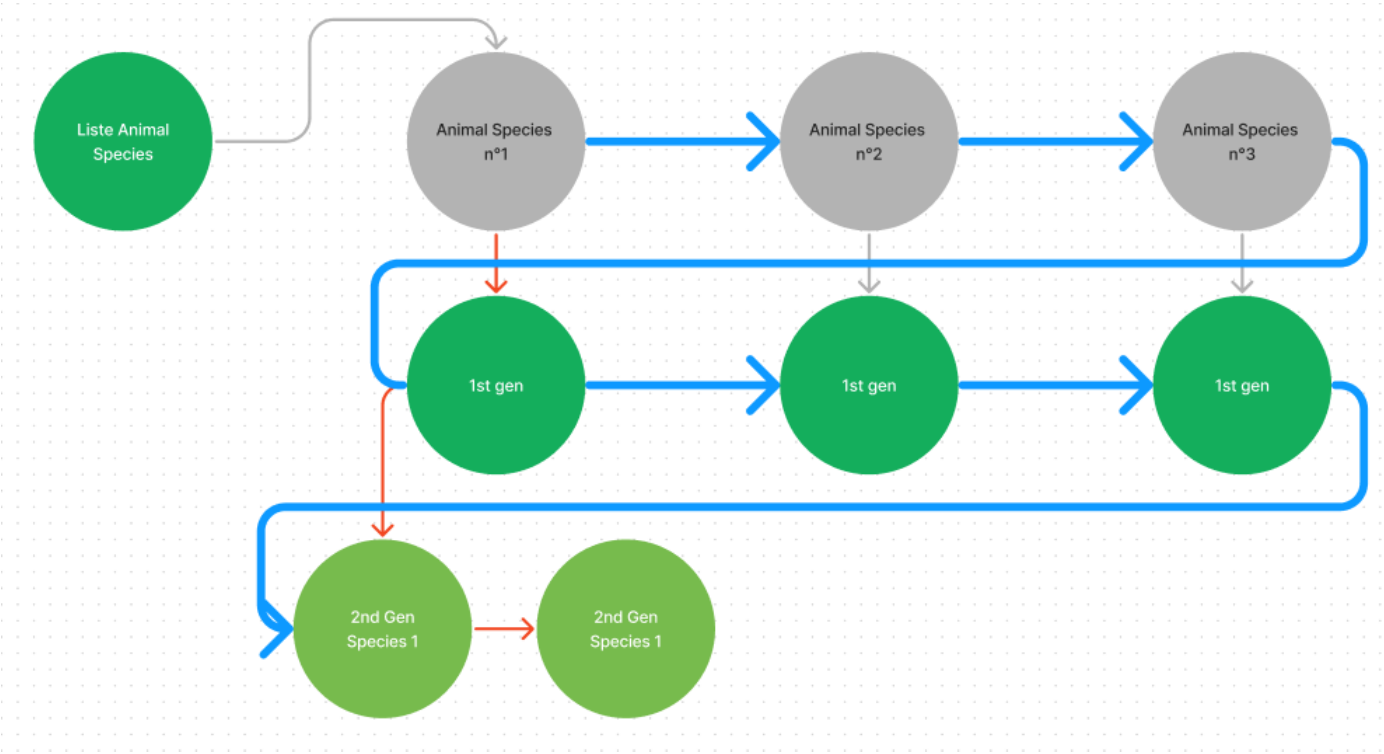


Figure 3 – En bleu le parcours retenu

Comme nous l'avions appris avec Mme Chaoued en SUP, un bon parcours largeur se doit d'utiliser une queue. Nous avons donc utilisé la struct list et créé des fonctions à part queue_push, queue_pop et queue_empty, nous avons utilisé une implémentation avec sentinelle et circulaire (la sentinelle pointe sur l'élément le plus récent qui lui-même pointe sur le plus vieux).

2.3 Interaction

En ce qui concerne les interactions entre les êtres vivants, nous avons dû faire plusieurs choix. Tout d'abord, nous avons implémenter la fonction « launch_interaction » qui permet à un animal de savoir si un autre animal se trouve à coté. Si un animal carnivore se retrouve face à un herbivore (se trouve à une case adjacente à ce dernier), les deux animaux ne disposent pas de mouvement dans le tour mais l'animal carnivore est obligé d'attaquer l'animal herbivore. On lance donc notre fonction « get_fight », qui va tout d'abord calculer avec un pourcentage de chance si l'herbivore parvient à s'enfuir de son agresseur et donc s'en aller dans la direction opposée à ce dernier en augmentant sa vitesse par 2. Ensuite, s'il n'arrive pas à s'échapper, l'animal attaquant va essayer de le tuer avec de nouveau un certain pourcentage de chance qui varie en fonction de la différence de force entre les deux animaux. En ayant pour objectif de refléter la réalité, si l'animal attaquant rate son attaque, il pourra réessayer au tour suivant.

2.4 Déplacement

2.4.1 Choix du mouvement des animaux

Pour permettre à un carnivore de chasser un herbivore, ou inversement qu'un herbivore fuit un carnivore, nous avons implémenté la fonction « possible_interaction » qui permet à un animal de voir à un certain nombre de cases autour de lui selon sa variable « sense », afin de décider quel déplacement faire. Simplement, le programme analyse toutes les cases qui se trouvent à une certaine distance de perception de l'animal

et va choisir la meilleure option lorsqu'un animal est rencontré, ou alors déterminer une direction aléatoire à prendre si aucun autre animal n'est perçu.

2.4.2 Vitesse de déplacement

La vitesse d'un animal dans notre simulation définit le nombre de case que l'animal pourra parcourir en un déplacement. Nous avons donc implémenté une fonction permettant de calculer le nombre de déplacement de l'animal en fonction de sa vitesse. Il faut noter que la caractéristique de vitesse est divisé par dix pour trouver le nombre de déplacement garanti, et le reste sera un taux de chance de se déplacer d'une case de plus. Par exemple :

```
Fox->speed = 27
=> Nombre de déplacements garantis: 2
=> Chance de se déplacer d'une case supplémentaire: 70% (27 modulo 10 * 10)
```

Figure 4 – Calcul du déplacement

2.4.3 Le mouvement

Pour terminer la partie sur le mouvement des animaux, il faut évidement une fonction de déplacement dans l'environnement. Elle prend en paramètres les coordonnées de la destination que l'animal cherche à atteindre, ainsi que la vitesse de l'animal. La première étape de la fonction est de calculer la distance en abscisse et la distance en ordonnée de la position actuelle de l'animal et de sa destination. Elle va ensuite déplacer d'une case ce dernier selon l'axe avec la plus grande différence tout en

faisant attention à ne pas dépasser les limites de notre environnement, et également ne pas marcher sur un animal déjà existant. Finalement, elle va répéter ces opérations le nombre de cases souhaité. Cette fonction présente encore quelques problèmes de gestion de l'environnement qui crée parfois des erreurs de segmentations qui seront à régler pour la prochaine soutenance.

2.4.4 Gestion des tours

Afin de tester nos fonctions, il nous a fallu créer notre fonction principale de gestion de tour pour chaque animal. Comme nous l'avons expliqué précédemment, les animaux les plus vieux joueront en premier leur tour, ce qui correspond dans notre architecture à un parcours largeur de nos animaux et de leurs enfants. C'est le rôle des fonctions « play » et « gen_end ». La première effectue le parcours largeur en faisant jouer chaque animal son tour en appelant les bonnes fonctions. La deuxième est appelée à la fin d'une génération, choisissant la mort, la survie ou la reproduction de l'animal.

2.4.5 Modification des anciennes fonctions

Nous pouvons noter pour ce rapport quelques majeurs changements dans les fonctions qui existaient précédemment :

— kill : Afin de respecter l'implémentation en arbre de l'évolution de nos êtres vivants, ils ne seront plus supprimés d'une liste contenant tous les animaux, au contraire ils seront toujours présents dans le programme mais définis comme morts afin qu'ils ne soient pas affichés dans la simulation. Ainsi, il est possible de garder une trace

de l'évolution de chaque animal et de ses enfants.

— reproduce : Cette fonction est restée globalement la même mais permet maintenant de placer chaque enfant dans l'environnement.

— life-or-death : A la place de gérer des cas pour chaque espèce, nous avons modifié la fonction afin que les décisions se fassent peu importe l'espèce de l'animal.

Afin de pouvoir nous repérer dans la simulation, nous utilisons une fonction qui permet d'afficher notre environnement dans le terminal. Vous pouvez voir sur la figure ci-dessous un exemple où les renards sont représentés par le caractère 'f', les moutons par 's', et les arbres par 'T' :

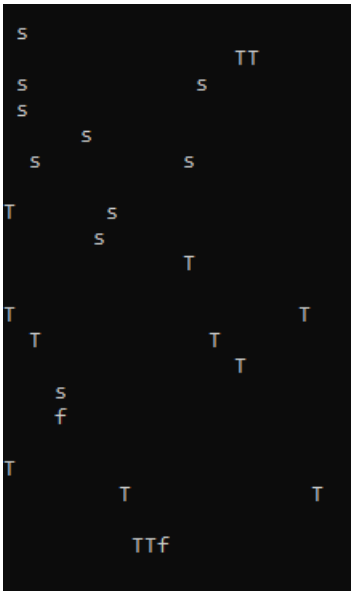


Figure 5 – Exemple de simulation

2.5 Interface

L'interface que nous allons utiliser ressemble beaucoup à celle que l'on a fait pour le projet OCR mais en plus poussée et plus approfondie.

2.5.1 Bouton et fonctionnalité

L'interface est divisée en plusieurs parties : à droite nous avons une série de boutons, au milieu et à la gauche un affichage montrant la simulation dans lequel les animaux interagissent. L'un des boutons mènera à un écran de statistiques affichant les informations concernant les animaux dans la simulation ainsi que des informations concernant la simulation actuel. L'affichage de la simulation montrera les animaux dans leur environnement, elle devra être mise à jour à chaque tour. A chaque tour les animaux doivent se déplacer, disparaître ou se dupliquer au fil des tours. Si nous avons du temps nous souhaitons rajouter l'option de toggle différents filtres sur l'affichage afin de mettre en avant un élément de la simulation tel que les biomes ou les animaux ou encore la nourriture disponible. Les boutons à droite servent à contrôler le temps dans la simulation, ce temps est traduit en tour et en génération. Les tours sont un temps durant lequel chaque animal aura son tour joué, il se déplacera, attaquera, mangera et ainsi de suite. Une génération est un nombre de tour à définir (peut varier entre 10 et 25 tours) qui sera la valeur de temps principale avec laquelle l'utilisateur interagira. Le bouton Start Simulation lance la simulation et sera renommé ensuite Play. Le bouton Pause met en pause la simulation une fois que la génération actuelle se termine. Le bouton Play Step sera seulement actif lorsque la simulation est en Pause, elle jouera une seule génération avant de mettre le jeu en Pause. Le bouton Play entame la prochaine génération de la simulation et ne s'arrêtera pas tant que l'utilisateur n'appuie pas sur Pause ou Stop Simulation. Le bouton Stop simulation arrête la simulation entièrement,

c'est-à-dire que tout le progrès qui a été effectué dans la simulation sera effacé et tous les éléments seront free. Le bouton Statistiques affiche un nouvel écran contenant des informations sur les animaux tels que leur vitesse, leur force, leur sens et leur régime. D'autres informations sur les animaux peuvent être ajoutées avec le temps. Des informations sur l'environnement seront également affichées telles que le nombre d'animaux restants de chaque espèce. Le dernier bouton Quit permet de quitter le programme.

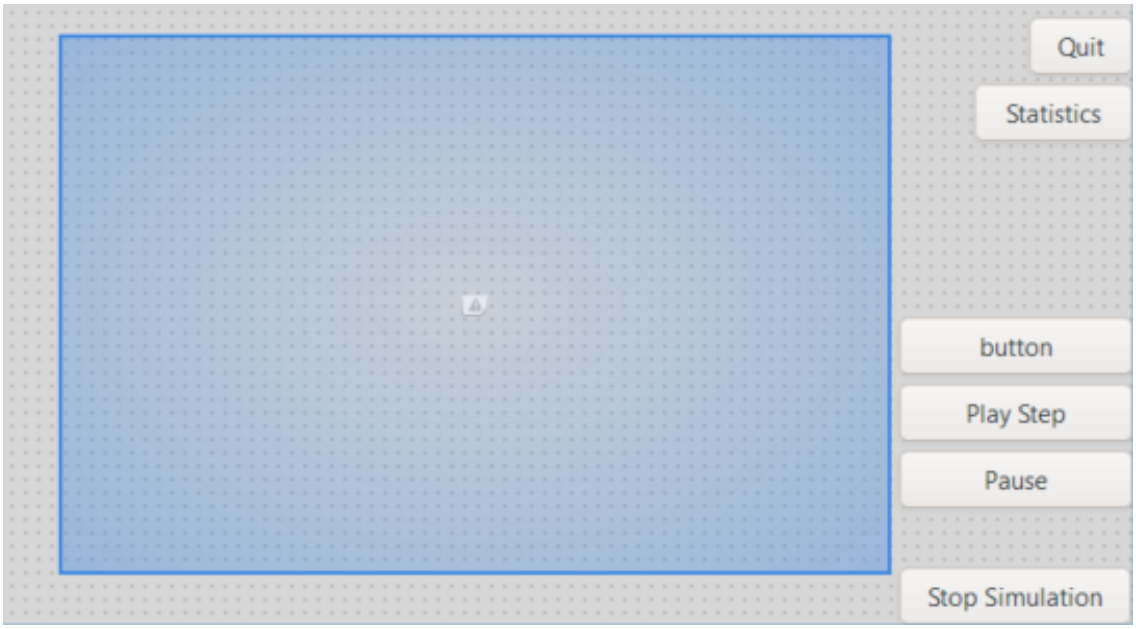


Figure 6 – état actuel de l'interface

2.5.2 Lecture de la matrice et bouton

La fonction de lecture de la matrice est plutôt simple à appliquer et le seul point important est de savoir ce que signifient les caractères que la fonction lit dans la matrice. Il est impératif que la matrice ou plutôt

l'environnement obéisse à des règles précises. Cela est nécessaire pour la lisibilité et la clarté de l'image affichée. Une deuxième tâche est la représentation des symboles de la matrice avec des sprites. Il faut que le sprite de la montagne soit lisible qu'il soit une montagne au milieu du désert ou une chaîne de montagnes. Ensuite les boutons devront tous être reliés avec les fonctions correspondantes.

2.5.3 Problèmes rencontrés

L'un des problèmes que l'on prévoit est la vitesse d'affichage qui variera en fonction du nombre d'animaux afficher. L'affichage et le code qui gère les interactions des animaux risque d'être si rapide que l'utilisateur ne puisse plus voir le déplacements et le chemin que les animaux ont pris. Pour remédier à cela nous rajouterons un temps de latence pour laisser l'utilisateur profiter. Ce temps de latence pourra également être modifié plus tard pour accélérer les tours et les générations. Mais un autre problème est le cas inverse ou le nombre trop élevée d'animaux ralentit la simulation. Actuellement la seule solution est d'optimiser le travail le mieux possible. Un problème récurrent est la gestion de librairies. En effet, en fonction de l'ordinateur sur lequel nous travaillons, le téléchargement de librairie et des dépendances varie grandement. La librairie la plus récalcitrante étant GTK. Ce problème nous empêche de tester proprement nos fonctions sur tous nos appareils. Le logiciel Glade que nous avons utilisé pour l'affichage est limité dans son utilisation. Cela nous oblige à réfléchir d'avantage à notre façon d'agencer notre interface afin de la rendre compréhensible et efficace.

2.6 Statistiques

Pour aider l'utilisateur à visualiser ce qu'il se passe dans l'interaction, j'ai réalisé des fonctions qui sont pour l'instant textuelles et qui permettent de suivre l'évolution de différentes statistiques. Tout d'abord, maintenant que les arbres ont leurs propres struct, nous pourrons aussi suivre l'évolution de la quantité de nourriture disponible sur la carte en temps réel. Aussi, nous allons suivre des statistiques sur les animaux, comme le nombre d'individus par espèce, le nombre de générations, le nombre de reproductions à chaque nouvelle génération, ainsi que le suivi de l'évolution des statistiques moyennes de chaque espèces : (speed, sense, strenght) et leurs total_food. Toujours dans l'idée de représenter des mécanismes réels de l'évolution, nous avons pensé à une façon de représenter la divergence évolutive. Au début d'une nouvelle génération si un individu a évolué de manière significativement différente du reste des individus de son espèce, on pourrait dire qu'il crée sa propre espèce. Cependant nous n'avons toujours pas intégré ces statistiques là dans l'interface.

2.7 Site web

Le site web était déjà assez avancé lors de la dernière soutenance. Comme il nous l'avait été conseillé par les ASMs, nous avons host le projet sur Github pages. Le site comporte actuellement une page de présentation du projet, une page pour télécharger le projet, le rapport de soutenance, ainsi qu'un lien pour accéder à l'organisation GitHub de

Kamels. Il y a aussi une page de présentation du groupe ainsi qu'une page de Documentation. Nous allons essayer de fournir la page documentation avec un suivi des principales nouveautés du projet et y mettre des captures d'écran des exemples de simulation en cours.

3 Conclusion

3.1 Etat d'avancement du projet

Tâche - Date	Prévisions 2ème soutenance	Avancement réel
Interface	70%	50%
Statistiques	50%	50%
Animaux	80%	65%
Environnement	70%	55%
Interactions	50%	75%

3.2 Problèmes rencontrés

Pour cette seconde soutenance, nous avons concentré nos efforts sur la partie centrale du projet, c'est-à-dire l'assemblage des parties animaux et environnement afin d'obtenir une simulation affichant des animaux capables de déplacements et d'interactions. De ce fait, les parties plus spécifiques du projet, tel que la gestion de l'interface, ou la génération procédurale de l'environnement ont un peu moins avancé.

3.3 Taches restantes à effectuer

Pour ce qui est de l'environnement, il faudra finir l'implémentation de l'algorithme de bruit utile pour la génération cohérente des différents biomes, puis implémenter les différences que ceux-ci impliquent sur la création des arbres.

En ce qui concerne les animaux, il reste à corriger les problèmes qui persistent encore ainsi que l'implémentation de la gestion de la nourriture, notamment l'ajout de la nourriture pour les herbivores.

Pour l'interface finale que nous souhaitons obtenir, il nous faut apporter un peu de couleur à cet interface monochrome, ainsi que raffiner l'ergonomie de notre affichage. Si le temps nous le permet nous souhaitons rajouter l'option de créer ces propres animaux, cela signifie un écran de création d'animaux. Nous nous sommes également penchés sur la possibilité d'intégrer un zoom dans la fenêtre de la simulation.

3.4 Conclusion

En conformité avec la conclusion du précédent rapport nous avons su rattraper notre retard et atteindre nos objectifs pour cette soutenance. Il ne nous reste maintenant plus que la soutenance finale et nous allons essayer de présenter un projet abouti et pertinent.