Projet informatique : EvoArena

Simulation de vie artificielle et d’évolution génétique

Maxime YOU & Clément ROBIN

1. Introduction et objectifs

Nous avons réalisé EvoArena pour ce projet de C++. C'est une simulation en temps réel où des cellules luttent pour leur survie. L'objectif était de mettre en pratique la sélection naturelle et la reproduction via du code.

Pour la partie technique, nous avons intégré plusieurs notions complexes comme le multi-threading et l'utilisation de la librairie SDL2. Tout le projet est codé en orienté objet pour bien structurer le programme.

2. Architecture technique

2.1 Choix technologiques

Le projet est développé en **C++** pour bénéficier de la puissance de la programmation orientée objet tout en maintenant des performances élevées nécessaires à la simulation de centaines d'entités.

- **Bibliothèque graphique : SDL2** est utilisée pour le rendu 2D, la gestion des événements et le fenêtrage.
- **Extensions SDL :**
    - SDL2_gfx : Pour le dessin de formes géométriques (cercles, polygones).
    - SDL2_ttf : Pour l'affichage de texte (interface utilisateur).
    - SDL2_mixer : Pour la gestion audio (musique et effets sonores).
    - SDL2_image : Pour l'importation de textures.
- **Format de données :** Utilisation de **JSON** (nlohmann/json) pour la configuration externe des traits génétiques.

2.2 Structure du code

Le projet suit une architecture modulaire :

- **Core (Simulation) :** Gère toute la simulation. Gère la boucle de jeu, le cycle de vie des entités et l'utilisation des cœurs du processeur.
- **Entity (Entity, Projectile) :** Encapsule la logique des individus (ADN, statistiques dérivées, machine à états).
- **Graphics & UI (Graphics, Menu) :** Gère le rendu visuel, la caméra (Zoom/Pan) et les interactions utilisateur.
- **Managers (TraitManager) :** Singleton permettant de charger et distribuer les mutations spéciales définies en JSON.

3. Implémentation du moteur génétique

Le cœur du projet repose sur un algorithme génétique complet respectant les étapes de l'évolution.

3.1 L'ADN

Chaque entité possède un tableau de gènes (float geneticCode[14]) déterminant ses caractéristiques physiques et comportementales. Conformément au cahier des charges, il n'y a pas de "valeur absolue" : chaque avantage a un coût.

- Gène taille (Radius) : Augmente les PV Max et l'Armure, mais réduit la Vitesse et augmente la Hitbox.
- Gène armement : Détermine la spécialisation (Mêlée, Distance ou Soigneur).
- Gènes comportementaux : Bravoure (seuil de fuite) et Gourmandise (priorité à la nourriture).

3.2 Reproduction et hérédité

À la fin d'une génération (lorsqu'il ne reste que N survivants), une nouvelle population est générée :

1. Sélection : Les survivants sont sélectionnés comme parents, avec plus de probabilité pour les entités fertiles.
2. Crossover : L'enfant hérite d'une combinaison des gènes de ses deux parents (moyenne ou sélection aléatoire).
3. Mutation : Une faible probabilité (5%) modifie aléatoirement un gène pour introduire de la diversité et éviter la convergence vers un seul gène.

3.3 Système de traits

Nous avons enrichi le système génétique avec des "Traits" chargés depuis un fichier JSON. Cela permet l'apparition de mutations rares comme *"Gourmand"* (Vitesse ++, mais perte constante d'énergie) ou *"Myope"* (Vision réduite), ajoutant une couche stratégique **dynamique**.

4. Intelligence Artificielle et Comportement

Pour dépasser le comportement aléatoire, chaque entité est pilotée par une **machine à états** complexe :

1. Errance : Exploration aléatoire ou déplacement vers le centre de la carte.
2. Recherche : Activé par la faim (gène *Greed*). L'entité cherche activement de la nourriture pour restaurer sa Stamina.
3. COMBAT : Engagement d'une cible à portée.
    - *Mêlée :* Charge au contact.
    - *Distance :* Maintien d'une distance de sécurité ("Kiting").
    - *Soigneur :* Priorise le soin des alliés (basé sur la couleur) avant l'attaque.
4. Fuite : Activé si les PV sont bas (gène *Bravery*). L'entité fuit à l'opposé de la menace.

5. Optimisation technique : multithreading

Afin de gérer un grand nombre d'entités (300+) sans ralentissement, nous avons implémenté une architecture parallèle.

5.1 Modèle multi-threadé

La fonction de mise à jour (Simulation::update) détecte le nombre de cœurs logiques disponibles sur la machine. Le tableau des entités est découpé en partie groupe, et chaque partie est traité par un thread indépendant (std::thread).

5.2 Gestion de la mise en commun (Mutex)

L'accès aux ressources partagées (comme le vecteur de projectiles ou la résolution de collisions) représente une section critique. Nous avons utilisé des verrous d'exclusion mutuelle (std::mutex) pour garantir la Thread-safety et éviter les *Data Races*, assurant ainsi la stabilité de la simulation même sous forte charge.

6. Interface et Expérience Utilisateur

Pour faciliter l'analyse de la simulation, nous avons développé une interface complète :

- Caméra dynamique : Possibilité de zoomer (Molette) et de se déplacer (Clic droit) dans un monde plus grand que la fenêtre.
- Inspecteur d'entité : En cliquant sur une entité, un panneau latéral affiche son génome, ses parents (Généalogie), son état mental actuel et ses statistiques vitales.
- Infos visuel & audio : Effets de flash lors de la nutrition, barres de vie dynamiques, et ambiance sonore immersive.

7. Conclusion

La réalisation d'EvoArena nous a permis de mettre en application les concepts théoriques du module C++ avancé en produisant une simulation fonctionnelle et autonome. L'objectif de modéliser l'évolution génétique est atteint : le couplage entre l'algorithme génétique et les machines à états permet bien de faire émerger des comportements de survie logiques au fil des générations, sans intervention directe de l'utilisateur.

Sur le plan technique, ce projet a constitué un défi important en matière d'architecture logicielle et d'optimisation. La nécessité de gérer plusieurs centaines d'entités en temps réel nous a contraints à structurer rigoureusement le code et à implémenter une gestion parallèle via le multithreading. Finalement, EvoArena démontre notre capacité à concevoir un système complexe alliant la flexibilité de la programmation orientée objet aux contraintes de performance d'un moteur temps réel.

*Notez qu'une partie plus technique et pratique se trouve dans notre fichier README.md