Rapport TER

Simulation Stochastique

Écrit par Julien Maille - Paez et Manitas Bahri

2023 - 2024

I. Introduction

Ce projet vise à créer une simulation informatique, en C++, capable de traiter un fichier texte contenant des réactions biochimiques et des instructions spécifiques. L'objectif principal est de développer un modèle informatique qui peut reproduire de manière réaliste et aléatoire le mouvement des molécules à l'intérieur des vésicules cellulaires, en tenant compte des interactions complexes avec les enzymes.

Le diabète est une condition médicale dans laquelle le corps a des difficultés à réguler le taux de sucre dans le sang. Le sucre, également appelé glucose, est une source d'énergie essentielle pour le fonctionnement de notre corps. Pour que le sucre puisse être utilisé comme énergie, il doit pénétrer dans nos cellules avec l'aide de l'insuline, une hormone produite par le pancréas.

Dans le cas du diabète, le pancréas ne produit pas suffisamment d'insuline ou les cellules du corps ne répondent pas efficacement à l'insuline produite. Cela entraîne une accumulation de sucre dans le sang, ce qui peut provoquer divers problèmes de santé s'ils ne sont pas contrôlés.

Pour diagnostiquer le diabète, les médecins utilisent souvent des tests de dépistage qui mesurent la quantité de sucre dans le sang. Une méthode courante consiste à utiliser des réactions enzymatiques spécifiques. Les enzymes sont des protéines qui accélèrent les réactions chimiques dans notre corps. Dans un test de dépistage du diabète, des enzymes spécifiques réagissent avec le sucre présent dans un échantillon de sang. Si le sucre est présent en quantité anormale, la réaction enzymatique peut produire un changement de couleur observable. Ce changement de couleur peut être utilisé pour déterminer si une personne est susceptible d'avoir du diabète ou non.

II. Analyseur Lexical

Le lexer, ou analyseur lexical, est conçu pour lire le fichier d'entrée caractère par caractère et les convertir en une série de jetons. Chaque jeton représente une unité logique de l'information, telle qu'un identifiant, un nombre, une ponctuation spéciale, ou un mot-clé.

Le lexer joue un rôle fondamental dans la préparation des données pour le parsing en décomposant le texte d'entrée en jetons compréhensibles. Chaque jeton est soigneusement extrait en fonction des règles définies dans la fonction ‘*lex’*, permettant ainsi une analyse syntaxique précise dans les étapes suivantes du processus. Le design et la mise en œuvre de *lexer.hpp* assurent que toutes les données nécessaires sont disponibles et correctement formatées pour le parsing ultérieur par *parser.hpp*.

a. Les méthodes principales

- ***hash*** **:** Cette fonction calcule et retourne un indice basé sur la valeur hash d'une chaîne de caractères. Elle est utilisée pour identifier de manière unique les chaînes lors de leur insertion ou leur recherche dans une table de hachage.

- ***index*** **:** Cette fonction recherche une chaîne dans la table de hachage du lexer. Si la chaîne n'est pas trouvée et que l'insertion est demandée, elle ajoute la chaîne à la table.

- ***lex*** **:** C'est la fonction principale qui lit les caractères du fichier d'entrée et les convertit en jetons. Elle gère différents états pour identifier correctement les nombres, identifiants, ponctuations, etc.

- ***lex_all*** **:** Cette fonction appelle *lex* dans une boucle jusqu'à ce que la fin du fichier soit atteinte, stockant tous les jetons générés dans un vecteur.

- ***print_ul*** **:** Fonction utilitaire pour afficher les informations d'un jeton.

b. Le traitement de flux

Le lexer fonctionne en lisant chaque caractère du fichier d'entrée et en déterminant son type en fonction de l'état actuel du lexer et du caractère lu :

- **STD** : L'état initial où le lexer décide du prochain état en fonction du caractère.

- **NUM**, **IDENT**, **ARROW**, **COMMENT** : Ces états aident à construire des jetons pour les nombres, les identifiants, les commentaires, les flèches (->), etc.

c. Exemple de traitement

Considérons un exemple de ligne du fichier d'entrée : *'**init (E1) = 30;***'. Voici comment le lexer traite cette ligne :

- *'**init**'* est reconnu comme un mot-clé (KEYWORD).

- *'**(E1)**'* est traité où E1 est reconnu comme un identifiant (IDENT) et les parenthèses comme ponctuation (PONCT).

- *'**=**'* est reconnu comme un symbole d'assignation (PONCT).

- *'**30**'* est reconnu comme un nombre (NUM).

- *'**;**'* est reconnu comme un point-virgule terminant l'instruction (PONCT).

III. Analyseur Syntaxique

L'objectif principal du parser est de transformer une séquence de jetons générés par le lexer en une structure de données ou en instructions que le système peut utiliser pour exécuter des opérations spécifiques, telles que des simulations biochimiques. Le lexer analyse le texte source pour identifier et extraire des jetons, tandis que le parser analyse la séquence de ces jetons pour construire une représentation plus significative.

Le lexer commence par lire le texte source et segmente ce texte en une série de jetons, chacun représentant un élément syntaxique de base tel qu'un identifiant, un mot-clé, un symbole ou un nombre. Ces jetons sont ensuite passés au parser.

Le parser reçoit ces jetons et commence son processus d'analyse basé sur une série de règles prédéfinies qui déterminent la structure syntaxique attendue du langage en question. Pour notre projet, les principales structures reconnues par le parser comprennent des instructions et des réactions chimiques, chacune ayant des composants spécifiques tels que des enzymes, des substrats, des produits, des constantes de réaction, etc.

1. Exemple de Flux de Contrôle

Étape 1 - Réception des Jetons : Le parser obtient une liste de jetons du lexer. Ces jetons sont traités un par un pour vérifier leur conformité avec les règles syntaxiques attendues.

Étape 2 - Construction des Structures :

- Instructions : Pour une instruction, le parser vérifie le type de l'instruction (par exemple, INIT, DIAMETER, SPEED), extrait les valeurs numériques associées et les stocke dans une structure *'instr'*.

- Réactions Chimiques : Pour une réaction chimique, le parser extrait les identifiants de l'enzyme, du substrat et du produit, ainsi que les paramètres de la réaction comme les constantes de mM et kcat.

Étape 3 - Gestion des Erreurs : Si un jeton inattendu est détecté ou si les jetons ne suivent pas la séquence attendue, le parser génère une exception ou une erreur, informant le système d'une possible erreur syntaxique ou de format dans le texte source.

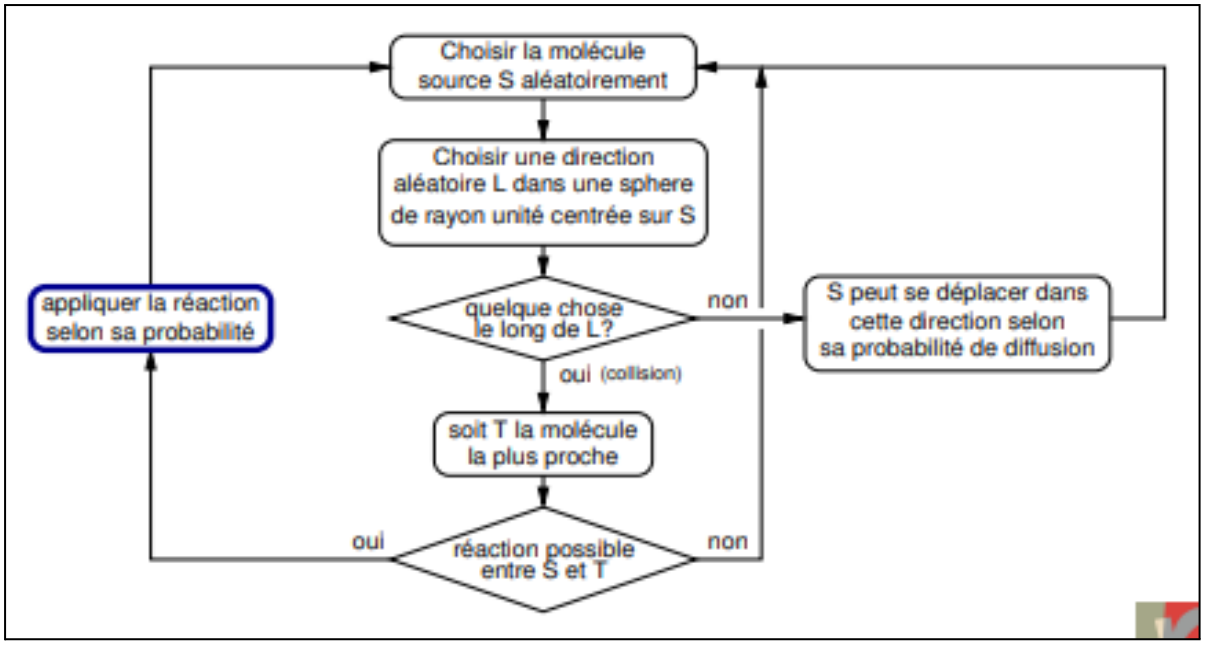
2. Importance de la Coordination entre Lexer et Parser

La coordination efficace entre le lexer et le parser est cruciale pour le traitement précis des données. Une mauvaise segmentation par le lexer peut entraîner des erreurs de syntaxe non détectées par le parser, ce qui peut conduire à des erreurs lors de la simulation ou de l'analyse des données. Par conséquent, il est essentiel que le lexer fournisse une séquence exacte et complète de jetons conforme aux attentes du parser.

Le parser, en collaboration avec le lexer, joue un rôle crucial dans la transformation des entrées textuelles en structures de données exploitables. Cette transformation est au cœur de la fonctionnalité de notre système, permettant des simulations précises et efficaces basées sur des données textuelles formatées. La robustesse du parser, sa capacité à gérer divers cas d'erreurs, et sa coordination avec le lexer sont donc essentielles pour assurer l'intégrité et la fiabilité du système dans son ensemble.

IV. Le solveur Entité - Centré

Le fichier *simulation.hpp* implémente une simulation biochimique centrée sur les entités, basé sur l'algorithme d'entité centrée, typiquement des molécules, au sein d'un espace confiné. Ce programme gère les interactions dynamiques entre molécules, réactions enzymatiques, et mouvements aléatoires suivant des paramètres de vitesse et de diamètre spécifiques à chaque molécule.



Algorithme Entité - Centré

1. Initialisation des molécules et des réactions

- Mapping des instructions : La fonction *__map_instructions* est utilisée pour construire une map associant chaque identifiant de molécule à ses attributs (nombre, diamètre, vitesse). Cette structure facilite la gestion des attributs lors de l'initialisation des molécules et lors des interactions entre elles.

- Initialisation des positions : La fonction *init_equidistant_positions* calcule des positions initiales équidistantes pour chaque molécule dans un espace sphérique (représentant une vésicule), en veillant à ce qu'elles ne se chevauchent pas et restent confinées à l'intérieur de la vésicule.

2. Simulation du mouvement et des réactions

- Mouvement : La méthode *__rand_movement* génère un nouveau déplacement aléatoire pour une molécule donnée basé sur son angle aléatoire et sa vitesse. Ce mouvement respecte les limites de la vésicule.

- Détection de collision : *__is_hit* vérifie si une molécule entre en collision avec une autre, ce qui pourrait potentiellement déclencher une réaction chimique.

- Réactions chimiques : *__is_reacting* détermine si deux molécules en collision doivent réagir selon les règles définies dans les réactions chargées. Les fonctions

__reacting_fusion et *__reacting_unfusion* gèrent les états post-réaction des molécules.

3. Calcul des probabilités de réaction

Lorsque deux molécules entrent en collision, nous déterminons le type de réaction en calculant trois probabilités à l'aide des méthodes suivantes : *__compute_probability_1*, *__compute_probability_2* et *__compute_probability_3*.

Ces méthodes évaluent les probabilités des différentes étapes des réactions enzymatiques en fonction des constantes cinétiques et d'autres paramètres biochimiques.

4. Déroulement de la simulation

- Le cycle principal de la simulation, est réalisé par la méthode *move_all_molecules*, met à jour la position de chaque molécule, vérifie les collisions, et gère les réactions. Ce cycle se répète pour chaque pas de temps jusqu'à la fin de la simulation.
- À chaque étape, la position des molécules est actualisée, les collisions sont détectées, et les réactions potentielles sont exécutées en fonction des probabilités de réaction.

Ce modèle de simulation centré sur l'entité fournit un cadre robuste pour étudier des interactions moléculaires complexes dans un environnement spatial confiné. L'utilisation de méthodes probabilistes pour la gestion des réactions permet de simuler de manière réaliste les processus biochimiques aléatoires tels que ceux observés dans les cellules vivantes. Ce programme peut être étendu ou adapté pour inclure différents types de molécules, réactions, ou conditions expérimentales, offrant ainsi une plateforme flexible pour la recherche en biochimie et biophysique.

V. Visualisation graphique 3D

Le fichier *view.hpp* fournit une implémentation complète pour la visualisation 3D des données de simulation dans notre projet de simulation de diabète. Ce fichier utilise la bibliothèque OpenGL avec GLUT pour créer et gérer une interface graphique interactive, permettant aux utilisateurs de visualiser et d'interagir avec les molécules en simulation.
Voici les points clés de cette implémentation:

1. Initialisation d'OpenGL

La méthode *init_opengl* configure l'environnement nécessaire pour l'affichage avec OpenGL. Elle initialise GLUT, définit le mode d'affichage (double buffering, couleur RGB, et gestion de la profondeur), crée une fenêtre avec un titre spécifique, et lie les fonctions de rappel pour le dessin, les clics de souris, et les pressions de touches. Elle configure également le viewport

et les matrices de projection pour la caméra, ce qui est crucial pour un affichage correct des objets en 3D.

2. Gestion de la simulation

La classe View utilise un singleton pour garantir qu'une seule instance gère l'état de la visualisation à travers le programme. Cela est utile pour maintenir la cohérence lors des callbacks. La méthode *update_simulation* est appelée périodiquement pour déplacer toutes les molécules et redessiner la scène, assurant ainsi une animation fluide.

3. Dessin de la scène

La méthode *draw_scene* est le cœur de la visualisation. Elle commence par effacer les tampons pour préparer le dessin d'une nouvelle image. Elle positionne la caméra, dessine la vésicule (si présente) et toutes les molécules. Elle gère aussi l'affichage d'une légende qui inclut le temps de simulation et les compteurs de différents types de molécules, ajoutant une couche informative importante pour les utilisateurs.

4. Détails du rendu

- Vésicule : Représentée comme une sphère semi-transparente, ce qui aide à visualiser les limites du domaine de simulation.
- Molécules : Chaque molécule est dessinée comme une sphère solide, avec une couleur spécifique attribuée pour distinguer les différents types.
- Légende : Affiche des informations clés telles que le temps écoulé et le nombre de chaque type de molécule, utile pour suivre l'évolution de la simulation.

5. Interaction Utilisateur

Des gestionnaires pour les événements de souris et de clavier permettent à l'utilisateur de manipuler la vue de la simulation. Les rotations et le zoom peuvent être ajustés avec la souris et le clavier, offrant une expérience interactive qui aide à examiner les données sous différents angles.

Ainsi le module *view.hpp* illustre une intégration efficace de techniques de visualisation 3D dans le cadre d'une simulation scientifique complexe. En utilisant OpenGL et GLUT, il offre non seulement une représentation visuelle des phénomènes moléculaires mais aussi une interaction en temps réel, essentielle pour l'analyse et la présentation de données dynamiques complexes.

VI. Conclusion

En conclusion, afin d'améliorer la précision et l'efficacité de notre simulation stochastique des réactions enzymatiques pour le test du diabète, l'exploration de nouvelles

approches algorithmiques est nécessaire. L'intégration d'un algorithme basé sur les zones pourrait représenter une avancée significative, permettant une gestion plus efficace du nombre croissant de molécules impliquées dans les réactions. Actuellement, la limitation majeure de notre programme réside dans son traitement moléculaire individuel, entraînant une charge de calcul importante lors des interactions, même entre des molécules éloignées. L'adoption d'un tel algorithme permettrait une meilleure représentation des interactions moléculaires en regroupant les molécules selon leur proximité spatiale, ouvrant ainsi la voie à des simulations plus réalistes et économiques des réactions enzymatiques.

Au cours de la réalisation de ce projet, nous avons été confrontés à plusieurs défis que nous avons réussi à surmonter avec succès. Parmi ceux-ci, l'implémentation du lexer et la gestion des identifiants dans la table de hachage ont représenté des étapes cruciales. Le développement du lexer a nécessité une compréhension approfondie des règles de la syntaxe du langage, ainsi qu'une attention particulière à la précision et à l'efficacité de la reconnaissance des tokens. De même, la gestion efficace des identifiants dans la table de hachage a exigé une conception soignée de la structure de données et une gestion rigoureuse des collisions pour garantir une résolution rapide et précise des noms de variables et de fonctions. En surmontant ces obstacles, nous avons pu avancer dans le développement de notre projet de manière cohérente et assurer la robustesse de notre modèle.

VII. Annexe

1. Prérequis

Dans la conception du projet, nous avons utilisé les bibliothèques OpenGL avec GLUT. Cela nous a permis de créer une fenêtre graphique où la simulation est affichée.

Pour installer la bibliothèque sous Linux, veuillez exécuter les commandes suivantes dans votre terminal :

$ sudo apt-get update
$ sudo apt-get install freeglut3-dev

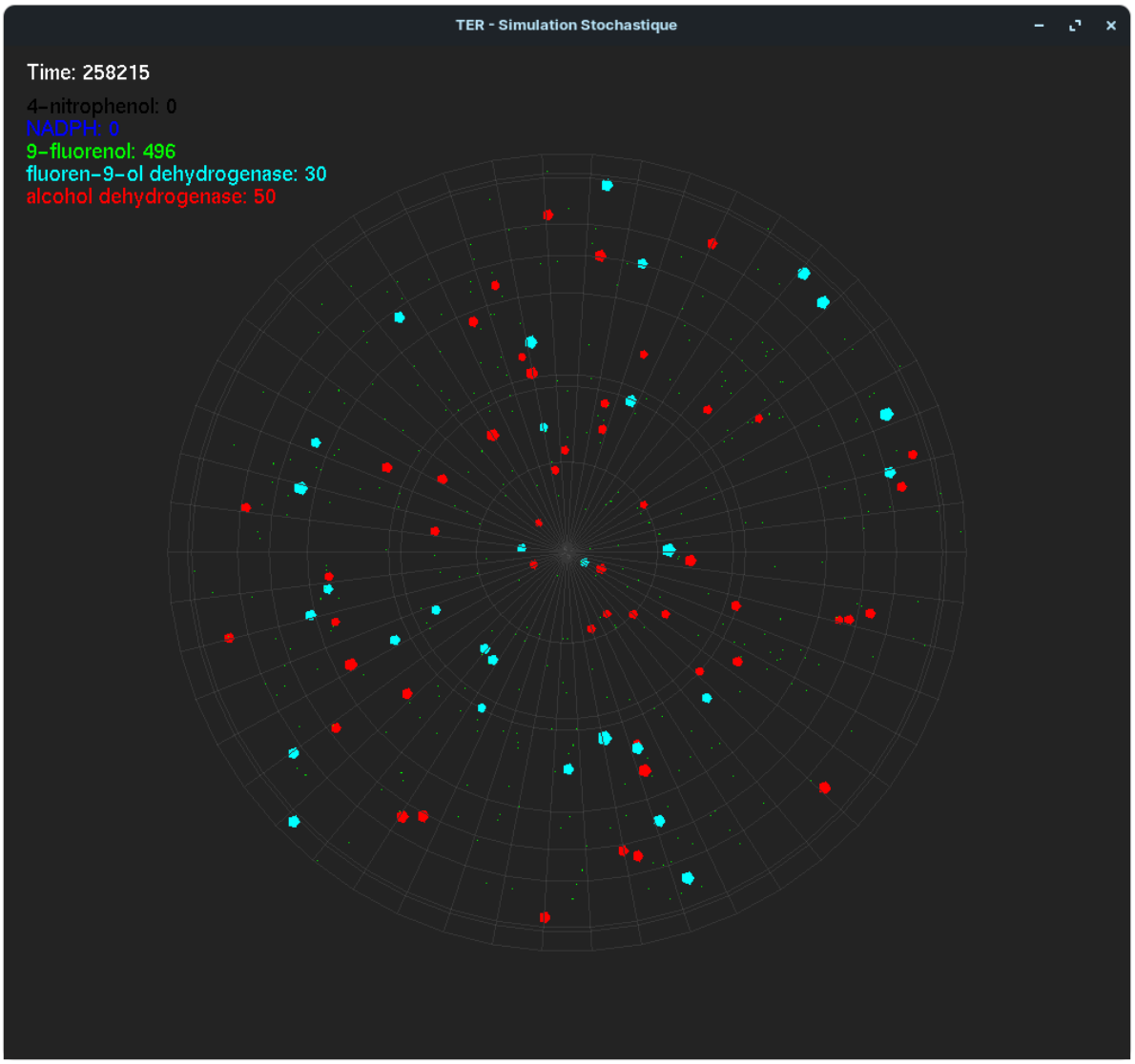
De plus, assurez-vous d'avoir C++11 installé sur votre système.

2. Lancement du programme

Pour lancer le programme, veuillez exécuter les commandes suivantes à la racine du projet.

$ g++ -std=c++11 main.cpp source/lexer.cpp source/parser.cpp source/simulation.cpp source/view.cpp -lGL -lGLU -lglut -o main
$ ./main “data/test.txt”

3. Image de la simulation



Temps: 250 215