Seveyrat Camille L2 MIASHS
Boccaccio Melissa S3

Compétence en Informatique

Niveau Expert Projet
Implémentation d'un Système Multi-Agents

Sommaire

I. Présentation du projet:

Pour ce projet, nous avons décidé de faire notre système multi-agent sur la création des étoiles. Ce système multi-agent ne représente pas exactement comment les étoiles se créent dû à la complicité de la chose, mais nous avons fait en sorte qu'il se rapproche le plus de la réalité tout en maintenant un affichage dynamique. Tout d'abord, nous allons expliquer comment les étoiles se forment dans la réalité, puis faire le lien avec nos modifications et nos simplifications pour créer un programme le plus ressemblant.

Cette explication est importante car lors de la création du projet, la première chose que nous avons fait est de bien comprendre le phénomène d'évolution de la formation des étoiles grâce à de nombreuses recherches, et cela facilitera aussi la compréhension.

Explications:

La nébuleuse (nuage de gaz et de poussière) va se fragmenter dû à la contraction gravitationnelle en plusieurs cœurs protostellaires. La phase protostellaire est un stade précoce dans le processus de formation d'une étoile. Il s'agit d'une région dense, qui, en accumulant de la matière, forme une proto-étoile, celle-ci étant principalement constituée d'Hydrogène et d'Hélium. Lorsque la proto-étoile atteint une certaine température et une certaine densité dans son noyau, la fusion d'atomes d'Hydrogène va créer un atome d'Hélium : c'est la fusion nucléaire. Pendant cette fusion, il y a une augmentation de la pression et une perte de masse ce qui crée une énergie thermique, et qui amène à que l'étoile s'allume.

Création des étoiles pour le projet:

Le panel représente une partie de nébuleuse. Dans le panel, au commencement, on y retrouve des atomes d'Hélium et des atomes d'Hydrogène, environ 25% d'atomes d'Hélium et 75% d'atomes d'Hydrogène. Cette proportion restera la même tout au long de la simulation grâce à la création progressive d'atomes.

Les atomes d'Hydrogène et d'Hélium sont attirés entre eux grâce à la force d'attraction gravitationnelle que l'on a configurée. Les atomes se rejoignant, un nuage dense se crée pour former, à un certain nombre d'atomes (et donc une certaine masse) accumulés, une proto-étoile. Nous n'avons pas créé d'agent "coeur protostellaire" pour simplifier notre système multi-agent. Nous aurions pu utiliser la formule de la masse critique de Jeans (masse à partir de laquelle commence l'effondrement de ce nuage dense pour former une proto-étoile), mais celle-ci n'était pas compatible avec le reste de la simulation. Nous avons donc défini nous-mêmes une masse critique.

Suite à cela, la proto-étoile va attirer d'autres atomes, toujours grâce à la force d'attraction gravitationnelle, et atteint un certain stade, la fusion nucléaire se produira. Pour cela, nous avons utilisé la formule de la température d'un système astrophysique. Lorsque la température atteint un seuil critique, que nous avons défini nous-mêmes, la fusion nucléaire s'opère, formant ainsi une étoile.

Nous avons défini un cycle de vie pour les étoiles de deux manières différentes : lorsque deux étoiles se rencontrent, les deux meurent et sont enlevées de la simulation, et lorsqu'une étoile atteint une certaine masse, elle "explose" et meurt.

On utilise seulement 5 agents (dont un alien qui sera là pour le pathfinding) pour ce projet, il y en a beaucoup plus en réalité, mais ce sont les agents les plus importants pour réaliser et comprendre comment la création d'étoile se fait.

Grâce au système multi-agent, nous pouvons voir comment se réalise la création de clusters et donc par la suite la création de proto-étoiles et d'étoiles.

Ce système sert donc à comprendre comment la formation d'étoiles se fait, comprendre quels sont les agents dans cette longue équation, quelles sont les relations (mathématiques) entre eux, tout cela de façon simplifiée !

II. Les agents

1. Listage des agents

Dans ce projet, on retrouve 5 agents tous dotés d'un nom, d'une position (x et y), d'une vitesse, d'une accélération et d'une masse. Tous nos agents sont attirés entre eux par la force d'attraction gravitationnelle.

L'Alien: L'Alien ne nous sert que pour le pathfinding, seule sa position change (de manière aléatoire) au cours de la simulation. Son objectif est d'atteindre ces positions cibles tout en évitant les étoiles et les proto-étoiles.

L'Hélium: L'hélium a une vitesse, une accélération et une position qui changent constamment au cours de la simulation. Seule sa masse est constante. Dans le panel, il se déplace donc grâce à la force d'attraction gravitationnelle par rapport à tous les autres agents. Il est représenté par un point de couleur cyan. Nous en générons au début de façon aléatoire et tout au long de la simulation pour créer un cycle. Il représente un quart des atomes présents dans le Panel.

L'Hydrogène: L'hydrogène a en majorité les mêmes caractéristiques que l'Hélium. Il est lui représenté par un point de couleur bleu. Il représente trois quarts des atomes présents dans le Panel.

La proto-étoile: Les proto-étoiles ne sont pas générées dès le début dans le Panel, mais arrivent lorsqu'une région dense d'atomes atteint une masse critique. Leur vitesse, leur accélération et donc leur position ne sont pas constantes. Contrairement à la réalité, nous avons décidé de lui attribuer une masse constante pour simplifier l'affichage. Leur mouvement n'est aussi influencé que par la force d'attraction gravitationnelle. C'est la phase intermédiaire entre les atomes et l'étoile. Elles changent de couleur au fur et à mesure de la simulation (du orange au rouge), proportionnellement à leur température qui augmente.

L'étoile: Tout comme les proto-étoiles, les étoiles ne sont pas générées dès le début, mais elles arriveront à la suite de la fusion nucléaire. Leur mouvement est défini de la même manière que pour la proto-étoile, à la différence que leur masse augmente. Mais ici également, pour rendre l'affichage plus visible, la masse qu'elles accumulent n'est pas conforme à la réalité mais est "modérée" à l'aide d'une formule d'interpolation linéaire. Elle est représentée par un point jaune et sa taille augmente proportionnellement à la masse qu'elle accumule.

2. Les relations existants entre eux - Formules mathématiques :

La première formule que nous avons utilisée est l'accélération gravitationnelle, dérivée de la loi de l'attraction gravitationnelle de Newton, que nous avons définie pour chaque agent :

$$a_1 = G \cdot \frac{m_2}{r^2}$$

- a_1 est la particule de masse m_1, qui interagit avec a_2, qui est la particule de masse m_2
- G est la constante gravitationnelle, qui vaut environ $6{,}674 \times 10\text{-}11\ N \cdot m^2.kg^{-2}$
- m2 sont les masses de l'objet avec laquelle la particule de masse m1 interagit (en kilogrammes, kg).
- r est la distance entre les centres des deux objets (en mètres, m).

Par la suite, nous avons utilisé les équations de la vitesse et de la position correspondantes pour chaque agent également :

- v(t) est la vitesse de la particule à un instant t (en m/s).
- v0 est la vitesse initiale de la particule (en m/s).
- a est l'accélération de la particule (en m/s²).
- t est le temps écoulé (en secondes, s).

$$x(t) = x_0 + v_0 \cdot t + \frac{1}{2} a \cdot t^2$$

- x(t) est la position de la particule à un instant t (en m/s).
- v0 est la vitesse initiale de la particule (en m/s).
- a est l'accélération de la particule (en m/s²).
- t est le temps écoulé (en secondes, s).

Pour finir, nous avons utilisé la formule de la température d'un système astrophysique. Lorsque la température de la proto-étoile (ou plus précisément la région dans laquelle se trouve la proto-étoile) atteint un certain seuil, la fusion nucléaire s'opère :

$$T = \frac{\mu \cdot m_H \cdot G \cdot \text{totalMass}}{k_B \cdot 10^6}$$

- T est la température de la proto-étoile (en K)
- μ est la masse moléculaire moyenne (sans unité), environ 2.33
- G est la constante gravitationnelle ($6{,}674 \times 10\text{-}11\ N \cdot m^2.kg^{-2}$)
- totalMass est la masse de la proto-étoile (en kg)
- kB est la constante de Boltzmann ($1{,}38.10^{-23}$ J/K)
- 10^6 est un facteur de conversion pour obtenir la température en mégakelvins (MK).

III. Le programme

Le programme est en java avec l'utilisation de l'IDE Intellij, on utilise A* pour le pathfinding de l'alien, on utilise des threads et swing pour le panel.

Java est un langage de programmation orienté objet, c'est-à-dire qu'il est un modèle de programmation qui repose sur le concept de classes et d'objets. Pour ce projet nous avons plusieurs classes: des classes contenant les agents, des classes pour le mouvement des agents, deux classes pour l'affichage dans le panneau (le Panel et une classe pour le flash lumineux), une classe pour la transformation des agents et deux autres pour le pathfinding. Toutes ces classes sont dans un package: package fr.um3.projet;

1. Les classes et leurs méthodes

1.1. Les agents

Agent:

Dans cette classe, on instancie, dans un constructeur, un nom et une position. On y retrouve également une méthode run() et stop(). La méthode run() est utilisée pour définir ce que le thread doit faire lorsqu'il est exécuté. Elle fait partie de l'interface runnable. Si le thread est interrompu, une exception est levée pour signaler le problème. Tous les agents ont hérité de cette classe.

Hydrogen et Helium:

On instancie dans ces deux classes les atomes d'Hydrogène et d'Hélium, qui comprennent une position, une vitesse, une accélération et une masse.

ProtoEtoile:

Ici est instanciée la proto-étoile, qui comprend elle aussi une position, une vitesse, une accélération et une masse mais aussi une température et une couleur, car celles-ci changent au fur et à mesure.

- Méthode **updateColor**:

Cette méthode retourne un type Color. On y définit une température minimale (environ la température initiale de la proto-étoile) et maximale (environ la température critique de la proto-étoile). Grâce à celles-ci, on calcule un ratio entre 0 et 1, 0 qui correspond à la couleur orange et 1 qui correspond à la couleur rouge. On "calcule" ensuite la nuance de couleur par rapport à ce ratio et finalement, on retourne celle-ci.

Etoile:

L'étoile comprend également une position, une vitesse, une accélération et une masse mais elle est également définie par un état "Flashing", qui, s'il est vrai, déclenche un flash lumineux.

1.2. Leur mouvement

Position:

Dans cette classe position on y crée une position x et une autre y, car le panel est en 2D. Chaque agent a une position, qui est random et qui n'est pas fixe.

- Méthode **distanceTo**:

Elle va servir à calculer la distance entre un autre agent, elle nous est utile pour le pathfinding.

- Méthode **equals**:

Elle compare deux objets pour vérifier s'ils sont égaux si leurs coordonnées X et Y sont identiques. Cette méthode equals est complémenté par **hashCode**, c'est une méthode qui génère un entier pour organiser et rechercher efficacement des objets. Cette implémentation assure que deux objets qui sont égaux (via equals) ont le même code de hachage.

Velocity:

Tout comme la position, la vitesse est définie par un x et un y.

Acceleration:

La classe accélération est également définie par un x et un y, et elle possède une méthode qui calcule l'accélération d'un agent par rapport à une liste d'autres agents.

- Méthode **GravitationalAcceleration**:

Cette méthode retourne un type Acceleration et prend en paramètre un agent cible (sur lequel s'appliquera l'accélération retournée) et une liste de tous les agents qui influencent le déplacement de l'agent cible. On utilise ici la formule de l'accélération gravitationnelle définie plus tôt, et on normalise les vecteurs selon X et Y, qui composent l'accélération. On retourne finalement l'accélération de l'agent cible.

1.3 L'affichage

TriggerFlash:

Cette classe génère un flash lumineux temporaire en utilisant un thread qui définit la durée du flash.

- Méthode **triggerFlash**:

Elle prend en paramètres une Etoile, une liste d'étoiles et un Panel. Lorsqu'elle est appelée, elle met l'étoile en paramètre en mode Flashing pendant 350 ms en utilisant un Timer, elle le désactive ensuite et enlève l'étoile de la liste des agents et du Panel.

Panel:

La classe panel est responsable de l'initialisation, de l'affichage et de la mise à jour des positions des agents (Hydrogène, Hélium, ProtoEtoile, Etoile, et Alien) dans un panneau graphique.
Cette classe comporte un grand nombre de méthode:

- Méthode **generateRandomTarget**:

Elle retourne une Position calculée aléatoirement (qui sert de position cible pour l'Alien)

- Méthode **removeAgent**:

Elle prend en paramètre un Agent et permet de le supprimer à la fois du Panel et de la liste des agents.

- Méthode **addAgent**:

Elle prend en paramètre un Agent et permet de l'ajouter à la fois au Panel et à la liste des agents.

- Méthode **initializeAgents**:

Cette fonction permet d'initialiser les hydrogènes et les héliums dans le Panel avec des positions aléatoires, une masse que nous avons définie, une accélération et une vitesse nulles pour le commencement.

- Méthode **agentPosition**:

Elle prend en paramètres un Agent et un type double, qui correspond à l'intervalle de temps deltaT nécessaire aux équations de la vitesse et de la position. Elle met donc à jour la vitesse et la position de l'Agent en paramètre grâce à son accélération calculée dans la classe Acceleration, qui se retrouve dans les équations de la vitesse et de la position.

- Méthode **updateAgentsPositions**:

Elle prend en paramètre le même deltaT que pour la méthode agentPosition, et appelle cette méthode pour tous les hydrogènes, héliums, proto-étoiles et étoiles.

- Méthode **isOutOfBounds**:

Elle retourne un type booléen et prend en argument une Position. Si la Position est en dehors des dimensions du panneau d'affichage, alors celle-ci retourne true, sinon false.

- Méthode **massToSize**:

Elle retourne un type int et prend en argument un type double, ici la masse de l'étoile. Grâce à une formule d'interpolation linéaire, elle calcule la taille de l'étoile en fonction de sa masse et la retourne.

- Méthode **startSimulation**:

Cette méthode gère la simulation en mettant régulièrement à jour les positions des agents et en rafraîchissant le panneau graphique grâce à un timer et des threads.

Dans cette méthode il y a 3 mécanisme principaux:

1. Un Timer pour les mises à jour régulières des agents et du panneau
2. Un thread pour actualiser les mouvement de l'alien
3. Un thread pour l'actualisation des autres agents

1. Le timer est configuré pour exécuter un bloc de code toutes les 10 millisecondes. A chaque délai de ce temps, de nouveaux atomes d'Hydrogène sont ajoutés à la simulation. Puis, tous les quatre Hydrogènes, un Hélium est ajouté.
C'est également ici que la méthode **triggerFlash** est appelée pour gérer l'animation du flash lumineux, mais aussi l'appel de la méthode **uptadeEvolution** de la classe Transformations pour l'évolution de nos agents. Pour finir, dans ce timer il y a aussi la suppression des agents hors du panel avec **isOutOfBounds** et un repaint().

2. Premièrement, un thread est démarré pour gérer le mouvement de l'alien vers des cibles aléatoires générées par la méthode generateRandom. La méthode **getNextStep** de la classe Pathfinding est utilisée pour calculer la prochaine position de l'alien en fonction de sa position actuelle et de la cible. Si l'alien est bloqué et ne peut pas progresser vers la cible, une nouvelle cible est générée.

La position de l'alien est mise à jour et une pause (Thread.sleep(18)) est introduite pour simuler le déplacement.

3. Cet autre thread est présent pour mettre à jour les positions des autres agents en utilisant la méthode **updateAgentsPositions**. Cette méthode est appelée en boucle avec une temporisation de 10 millisecondes pour assurer des mises à jour rapides mais contrôlées

- Méthode **paintComponent**:

Cette méthode montre l'utilisation de la bibliothèque Swing pour créer une interface graphique 2D. Elle illustre la modélisation des agents avec des propriétés et comportements spécifiques. C'est grâce à elle qu'on soigne l'esthétisme du projet que l'on comprend facilement la dynamique du système.

Chaque agent possède une position déterminée dans l'espace, et sa représentation graphique est effectuée avec la méthode fillOval. Cela permet de dessiner des cercles.

L'hydrogène est définie par la couleur cyan et est fixé à 10*10 pixels. Pour l'Hélium, c'est identique à défaut de la couleur bleue. Ces deux types d'agents sont traités de manière similaire, mais différenciés visuellement par leur couleur.
Pour les proto-étoiles, la couleur est déterminée par protoEtoile.getColor(). La méthode **updateColor** dans la classe ProtoEtoile permet de faire changer la couleur en fonction de la température. Elles sont une taille de 23*23 pixels.
Puis, les étoiles sont répresenté en jaune mais elles sont plus complexes que les autres agents grêce aux méthode, **isFlashing** et **massToSize**.

- **massToSize** fait varier la taille de l'étoile en fonction de sa masse.
- **isFlashing** en lien avec **triggerFlash**, fait que une fois arrivé a sa masse critique l'étoile crée un "flash"(Le dégradé passe du blanc opaque (centre) au blanc transparent (bord)), puis disparait du panel.

L'alien est représenté en vert avec une taille de 20*20 pixels.

1.4. Transformations

Transformations:

Cette classe permet premièrement, de regrouper nos agents dans des régions, qui nous servent à définir une masse pour chaque région pour la transformation des régions denses mais aussi "l'explosion" des étoiles trop massives. Elle contient également la formule pour la température, et bien sûr les méthodes pour créer les proto-étoiles, les étoiles, et pour enlever de la simulation les étoiles trop proches.

- Méthode **updateEvolution**:

Elle prend en arguments la liste des hydrogènes, des héliums, des proto-étoiles et des étoiles. On utilise une collection contenant des types Point correspondant au centre des régions, régions dans lesquelles sont présents les agents. En effet, tous nos agents sont associés à une région pour ensuite pouvoir toutes les parcourir et définir nos formules pour les bons agents.

On utilise un type double pour incrémenter progressivement la masse des agents concernés. Si la région ne contient pas d'étoile ou de proto-étoile, et qu'elle atteint une masse critique définie, on crée une proto-étoile grâce à la méthode **createProtoEtoile**.

Pour les proto-étoiles, on calcule et on met à jour la température associée à leur région. Si la région contient bien une proto-étoile et ne contient pas d'étoile, et qu'elle atteint une température critique, alors on appelle la méthode **createEtoile**.

Et pour les étoiles, si la masse de leur région atteint une température critique, on appelle la méthode **triggerFlash** dans la classe TriggerFlash. Pour mettre à jour la température des étoiles, on utilise une formule d'interpolation linéaire pour que l'étoile ne dépasse pas une masse maximale que nous avons nous-mêmes définie pour un meilleur affichage.
Pour finir, on appelle également la méthode **removeCloseStars** pour supprimer les étoiles trop proches.

- Méthode **removeCloseStars**:

Elle prend en paramètres une liste d'Etoile. On y crée une liste d'étoiles à supprimer et en parcourant toutes les étoiles de la liste, si la distance entre deux étoiles est plus petite qu'une certaine distance définie, alors elles sont ajoutées à la liste d'étoiles à supprimer et sont donc supprimées de la liste d'agents et du Panel.

- Méthode **createProtoEtoile**:

Elle prend en paramètres une liste d'agents, ici ce sont les agents d'une même région, puis la liste des hydrogènes, la liste des héliums et celle des proto-étoiles. Elle s'occupe d'abord de retirer les atomes d'Hydrogène et d'Hélium de la région de la liste des agents et du Panel, elle calcule ensuite la position moyenne de la région pour attribuer celle-ci à la proto-étoile qui est finalement créée.

- Méthode **createEtoile**:

Elle prend en paramètres une liste d'agents, ici ce sont de nouveau les agents d'une même région, puis la liste des hydrogènes, la liste des héliums, la liste des proto-étoiles et des étoiles. Elle est très similaire à la méthode précédente : tous les objets présents dans la région sont supprimés puis une étoile est créée de la même manière.

1.5. Le pathfinding (recherche de chemin)

PathFinding:

On choisit de prendre un alien qui se déplace dans un environnement en deux dimensions, le panel. L'objectif est de trouver un chemin optimal entre un point de départ et un point d'arrivée tout en évitant des obstacles (étoiles et proto-étoiles). Le programme utilise l'algorithme A* pour cette recherche de chemin.

Méthodes principales du pathfinding:

- Méthode **run():**

Implémente l'interface Runnable, permettant d'exécuter cette classe dans un thread. Cette méthode sert à faire continuer le pathfinding tant que la target n'est pas atteinte. Il calcule également le prochain pas en appelant la méthode **getNextStep**, et met cette nouvelle position à jour de manière synchrone (sécurisé pour les threads). Il y a également une pause de 100 ms pour ralentir les mises à jour et éviter une surcharge.

- Méthode **getNextStep():**

Cette méthode utilise l'algorithme A* pour déterminer la prochaine position de l'alien sur le chemin vers la cible. Elle prend en compte les collisions potentielles et ajuste la direction en conséquence. Elle comporte plusieurs étapes, premièrement la vérification d'une potentielle collision. Cela est calculé grâce à la méthode **InSecurityRadius**, qui va détecter si une position future est dans le rayon de sécurité (45). Si l'alien se trouve dans le rayon de sécurité d'un obstacle, la méthode calcule une nouvelle position grâce à la méthode **moveAwayFrom** en déplaçant l'alien dans la direction opposée à l'obstacle pour éviter une collision.
Une fois la vérification des collisions effectuée, l'algorithme A* est initialisé.
Le chemin est calculé en tenant compte de la position de départ, de la position cible, et des obstacles. Il fonctionne en explorant les voisins d'un nœud donné et en choisissant le chemin avec le coût total (g + h) le plus faible. L'algorithme continue jusqu'à ce qu'il trouve un chemin vers la cible ou qu'il n'y ait plus de nœuds à explorer.
Composantes principales de A* :

Node : Représente chaque position explorée lors de la recherche du chemin, en conservant des informations telles que le coût du chemin (g), l'estimation vers la cible (h) et la somme de ces deux valeurs (f).
g(n) : Coût pour aller du nœud de départ jusqu'au nœud n.
h(n) : Heuristique, estimation du coût pour aller de n jusqu'à la cible.
f(n) : Somme des deux valeurs g(n) + h(n), qui guide l'algorithme dans son exploration.

Si la liste des nœuds ouverts n'est pas vide, la méthode retourne la position du nœud avec le coût total estimé le plus bas.

Cette méthode est donc un mixte entre les techniques de pathfinding et des vérifications de sécurité pour garantir que l'alien se dirige efficacement vers sa cible tout en évitant les collisions

- Méthode **getHeuristic():**

Elle aide l'algorithme A* à choisir les nœuds à explorer pour atteindre la cible plus efficacement. Elle utilise une méthode Manhattan (somme des différences absolues en X et Y)

- Méthode **willCollide():**

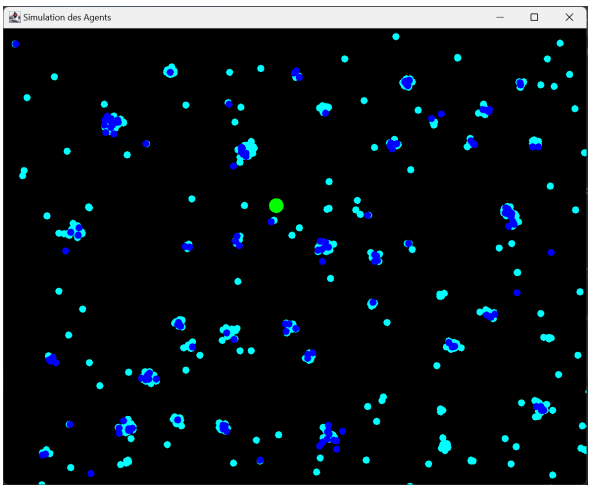
Cette méthode vérifie si une position donnée entraînerait une collision avec une étoile ou une proto-étoile en retournant un type booléen.
Comme pour **InSecurityRadius**, elle utilise un rayon de sécurité. La fonction parcourt toutes les étoiles et proto-étoiles. Pour chaque obstacle, elle calcule la distance entre l'obstacle et la position potentielle de l'alien en utilisant la formule de la distance euclidienne.

$$\text{distance} = \sqrt{(\text{etoileX} - x)^2 + (\text{etoileY} - y)^2}$$

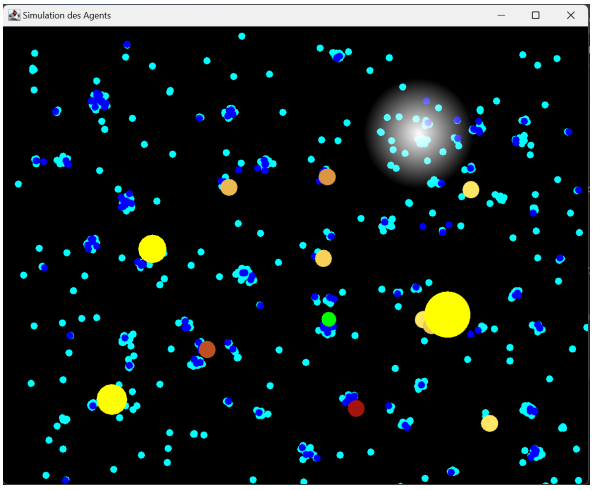
Si la distance calculée est inférieure au rayon de sécurité, la fonction retourne "true" pour indiquer une collision. Sinon, elle continue à vérifier les autres obstacles.
Si aucune collision n'est détectée après avoir vérifié toutes les étoiles et proto-étoiles, la fonction retourne false.

Ce sera dans le Panel qu'il y aura l'actualisation du mouvement de l'alien.

2. Aperçu de la simulation, détails associés et modélisation utilisée



Formation de clusters



Différentes phases

Tout d'abord, on peut voir que nous avons décidé de mettre le fond du panneau en noir pour imiter l'espace.

Comme défini dans la méthode **paintComponent** de la classe Panel détaillée précédemment, nos atomes sont de même taille, de couleur cyan pour les Hydrogènes, puis bleue pour les Héliums. Au début de la simulation (sur la première image), on peut voir clairement la création de clusters entre ces atomes, et cela continue bien sûr tout au long de la simulation.

Ensuite, les proto-étoiles sont, comme on peut le voir sur la seconde image, un peu plus grosses que les atomes, et on peut apercevoir qu'elles ne sont pas toutes de la même couleur. C'est effectivement dû à la méthode **updateColor** de la classe ProtoEtoile commentée précédemment. Les proto-étoiles entourées de peu d'atomes sont jaunes orangées (leur température est basse) tandis que celles qui ont beaucoup d'atomes autour d'elles sont très rouges (leur température est élevée).

Puis, les étoiles présentes sur la seconde image sont toutes de couleur jaune, elles sont plus ou moins grosses. C'est la méthode **massToSize** de la classe Panel qui est responsable de ce changement. Les étoiles entourées de peu d'atomes et de proto-étoiles sont assez petites tandis que celles qui, au contraire, en ont beaucoup autour d'elles, sont très grosses et finissent par exploser grâce à la méthode **triggerFlash** de la classe TriggerFlash. Ce flash est modélisé dans la méthode **paintComponent** de la classe Panel par un rond de couleur blanche au centre dégradée vers le transparent sur les bords.

Enfin, on peut voir sur nos deux images un rond de couleur verte, qui représente l'Alien qui se déplace au fur et à mesure de la simulation grâce à l'algorithme de pathfinding.

Dans notre projet, plusieurs types de modélisation ont été employés pour représenter et gérer les différentes entités et interactions du système. Ces modélisations nous ont permis de structurer et de simuler efficacement un environnement dynamique.

- Nous avons tout d'abord utilisé la modélisation orientée objet. Elle nous permet de représenter les différentes entités du système sous forme de classes et d'objets. En effet, chacun de nos agents est défini dans une classe avec des attributs et des méthodes associées.
- Ensuite, notre projet s'appuie sur la modélisation multi-agent, qui reflète donc directement le sujet de notre projet. Cette modélisation est essentielle pour simuler les comportements locaux des agents (comme la fusion d'atomes) et leurs conséquences globales (formation d'étoiles).

- Pour finir, nous pouvons voir que nous avons choisi une modélisation 2D dans un panneau (Swing). Les déplacements de nos agents se font selon les axes X et Y, et leur mouvement est visible en temps réel, ce qui rend la simulation visuellement intuitive.

IV. Difficultés rencontrées

Premièrement, nous avons rencontré des difficultés lors notamment de la création du PathFinding. L'algorithme de pathfinding A* est en effet assez complexe en raison de la recherche du chemin optimal et de la gestion des nœuds explorés. Nous avons dû modifier et ajouter de nouvelles méthodes à de nombreuses reprises, tout d'abord parce qu'au début de la création de la classe, l'Alien ne bougeait pas, bien que la méthode principale de la classe s'exécutait. En effet, le calcul du chemin optimal prenait énormément de temps à être calculé car la méthode principale retournait un chemin entier à suivre. De plus, cette méthode là ne fonctionnait que lorsque les agents n'étaient pas en mouvement, car le chemin n'était donc calculé qu'une seule fois et n'était jamais mis à jour. Nous avons donc changé le type de retour de cette méthode, qui retourne maintenant un type Position, qui correspond donc à la position suivante que l'Alien doit suivre. Et puis, nous avons par la suite dû prendre en compte le fait que lorsque des proto-étoiles et des étoiles s'approchent trop près de l'Alien, celui-ci doive reculer pour éviter une potentielle collision en raison du mouvement assez imprévisible de celles-ci.

Ensuite, nous avons rencontré quelques difficultés quant à la compréhension des nouvelles notions telles que les threads. L'action des threads n'étant pas clairement visible dans le Panel, il nous a été difficile de visualiser et comprendre réellement leur utilité et comment les faire fonctionner.

De plus, il nous a été difficile de programmer un Timer dans un JPanel, de la bibliothèque Swing. Ces difficultés ont surtout été liées à la compréhension de ce qu'était un ActionListener et la manière de l'utiliser.

Enfin, il a été difficile de trouver un équilibre dans l'évolution des agents : quel est le moment où la région dense se transforme en proto-étoile, quel est celui où la proto-étoile se transforme en étoile, quand est-ce que l'étoile meurt etc. Nous avons dû tester la simulation de nombreuses fois avec des masses différentes pour les atomes, des masses critiques et des températures critiques différentes. En effet, notre objectif était de rendre visible tous les phénomènes importants de la création des étoiles tout en maintenant les mouvements et les évolutions dynamiques.

V. Interprétation

Dans notre simulation, nous avons programmé des interactions locales entre agents en utilisant l'accélération gravitationnelle et les équations de mouvement. Ces règles locales ne codent pas directement la formation de clusters, mais celle-ci émerge comme de façon globale dans notre simulation, et cela est bien en accord avec nos attentes initiales. Cela illustre bien un phénomène d'émergence où un comportement collectif apparaît spontanément à partir de simples interactions individuelles.

Les transformations des régions denses en proto-étoiles, puis celles-ci en étoiles, et la mort des étoiles sont des phénomènes déterminés par des règles explicites et donc directement codés dans la simulation. Nous avons observé l'importance de la précision d'attribution des masses, et l'influence

qu’elles ont sur leur propre comportement mais aussi sur le comportement des autres agents sur lesquels agit la force gravitationnelle. Nous avons pu observer que différents paramétrages modifient nettement les interactions en rendant le processus plus lent ou bien plus rapide.

Le fait que nous ayons attribué nous-mêmes ces masses ne reflète donc pas la complexité réelle du processus. En effet, les proto-étoiles et les étoiles semblent se créer très rapidement, et ces dernières semblent mourir très rapidement aussi, ce qui n’est bien évidemment pas le cas dans la réalité. Les étoiles et proto-étoiles sont de ce fait très proches les unes des autres dans notre panneau d’affichage, et une bonne proportion de celles-ci finissent par se rencontrer, ce qui ne reflète pas non plus la réalité.

Pour finir, nous pouvons dire que cette simulation constitue une approximation simplifiée du processus de formation des étoiles. Avec un simple panneau d'affichage, nous avons modélisé dans les grandes lignes les étapes clés du phénomène, représentant ainsi certains mécanismes fondamentaux de la dynamique stellaire.

Répartition du projet:

Camille:

2ème semaine:
- choix du projet

3ème semaine:
- recherche sur la création des étoiles
- commencement du code en créant des classes pour chaque agent
- diagramme de Gantt

4ème semaine:
- recherche des formules mathématiques

5ème semaine:
- commencement de la création du Panel avec swing
- agents représentés graphiquement

6ème semaine:
- premières interactions entre les agents (force gravitationnelle dans le code)

7ème semaine:
- 1er compte rendu

8ème semaine:
- transformation des atomes en proto-étoile

9ème semaine:
- transformation des proto-étoiles en étoiles
- enlever les étoiles trop proches
- générer des atomes tout au long de la simulation

10ème semaine:
- supprimer les agents hors Panel
- appliquer un dégradé de couleur pour les proto-étoiles en fonction de leur température
- définir un flash pour les étoiles trop massives
- finalisation : attribution des bonnes masses et des bonnes températures (mises en commun)

11ème semaine:
- préparation du compte rendu
- préparation oral

Melissa:

2ème semaine:

- choix du projet

3ème semaine:

- agent et descriptif bien complet
- compréhension de la création d'étoile

4ème semaine:

- commencement du code des classes
- recherche et utilisations des threads

En Java, un thread est une unité d'exécution qui permet d'exécuter du code de manière concurrente. En d'autres termes, les threads permettent d'exécuter plusieurs tâches en parallèle dans un programme. Ils sont essentiels pour améliorer les performances dans certaines situations, notamment lorsque des tâches lourdes ou indépendantes doivent être exécutées simultanément.

5ème semaine:

- répondre à la question de pourquoi les threads s'exécute en "random" :

L'ordre d'exécution des threads dans un programme Java peut sembler aléatoire (random) à cause de la façon dont le système d'exploitation et la JVM gèrent l'ordonnancement des threads. Les threads peuvent être exécutés en fonction de leur état, de la charge du système et de l'algorithme d'ordonnancement utilisé. Cela permet une exécution concurrente, où plusieurs tâches peuvent progresser en parallèle, mais sans garantie d'ordre d'exécution.

- commencement du pathfinding

6ème semaine:

- pathfinding

7ème semaine:

- 1er compte rendu

8ème semaine:

- pathfinding
- approfondissement des threads dans le code
- finition diagramme de Gantt pour le 1er rendu

9ème semaine:

- thread dans le panel
- pathfinding

10ème semaine:

- finalisation du pathfinding
- finalisation thread
- grossissement de l'étoile proportionnel à la masse
- finalisation : attribution des bonnes masses et des bonnes températures (mises en commun)

11ème semaine:

- préparation du compte rendu
- préparation oral
- finalisation diagramme de Gantt pour le 2nd rendu

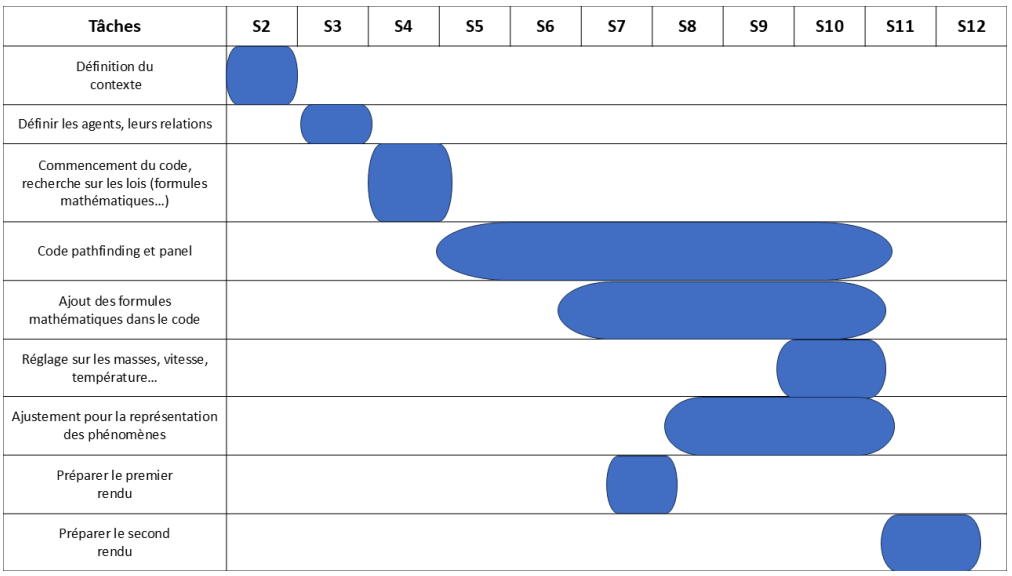


Diagramme de Gantt