Ant Colony Simulation - Design Pattern Mediator

authors: Rachel Tranchida, Quentin Surdez, Eva Ray

Instructions de Compilation

Versions utilisées pour le développement du projet :

- Java 21
- Maven compiler plugin: 3.8.1
- Maven jar plugin: 3.2.0

Au niveau de la racine du projet, exécutez la commande suivante pour compiler le projet :

```
mvn clean package
```

Ensuite, dirigez-vous dans le dossier `target` avec la commande `cd target` et exécutez la commande suivante pour lancer le projet :

```
java -jar Mighty_Ants-14.5.2.jar
```

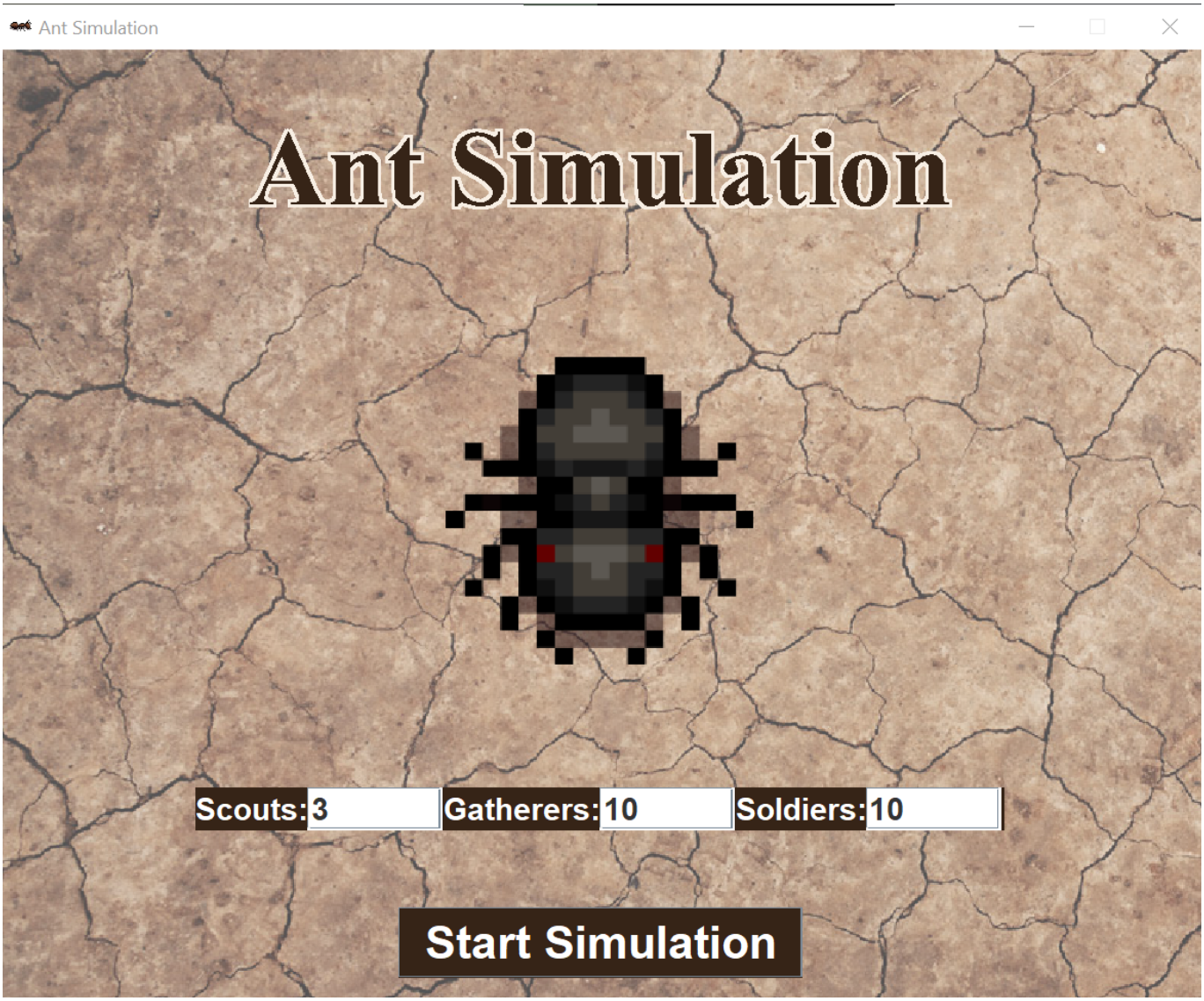
Description du Projet

Afin de mettre en oeuvre le design pattern Mediator, nous avons choisi de simuler le comportement de fourmis dans une colonie. Dans la nature, les fourmis communiquent entre elles pour trouver de la nourriture, construire des nids ou encore se défendre contre des prédateurs. Nous avons été inspiré par ces multiples communications et avons décidé d'utiliser le design pattern Mediator pour les simuler. Cependant, les communications que nous avons imaginées s'éloignent un peu de la réalité, afin de rendre le projet plus ludique.

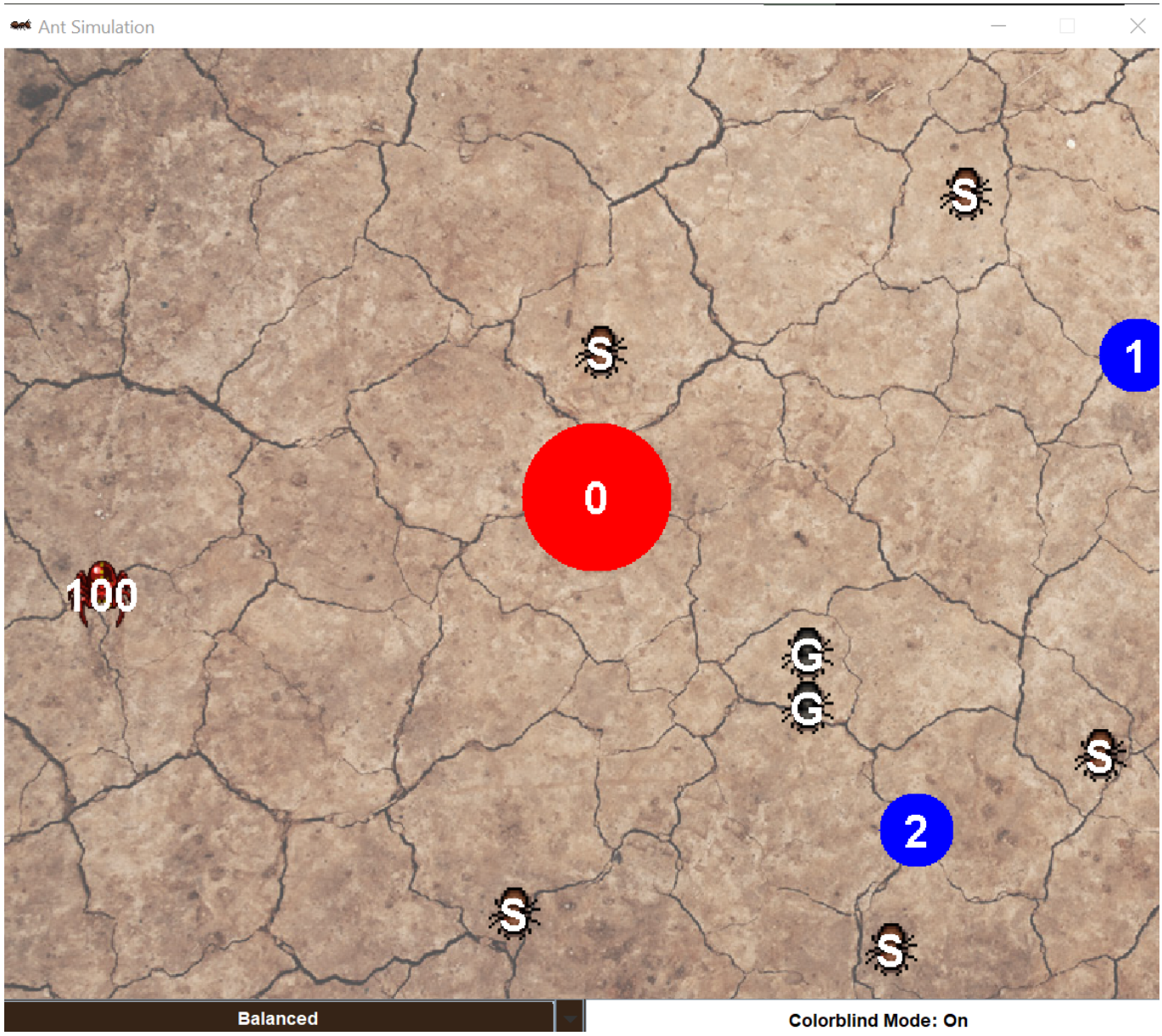
La simulation se déroule sur une carte de taille fixe. Il y a trois types de fourmis qui évoluent dans la simulation : les scouts, les gatherers et les soldiers. Chaque type de fourmi a des aptitudes différentes, qui seront expliquées en plus amples détails dans la section Collègues. Il y a aussi des prédateurs qui se déplacent sur la carte et qui peuvent attaquer les fourmis. Les fourmis peuvent explorer leur entourage pour découvrir des ressources et des prédateurs. En fonction des découvertes, les fourmis peuvent se voir assigner différentes tâches : aller chercher des ressources, attaquer des prédateurs ou fuir. C'est le médiator qui coordonne les interactions entre les entités et leurs réactions aux événements.

Fonctionnalités

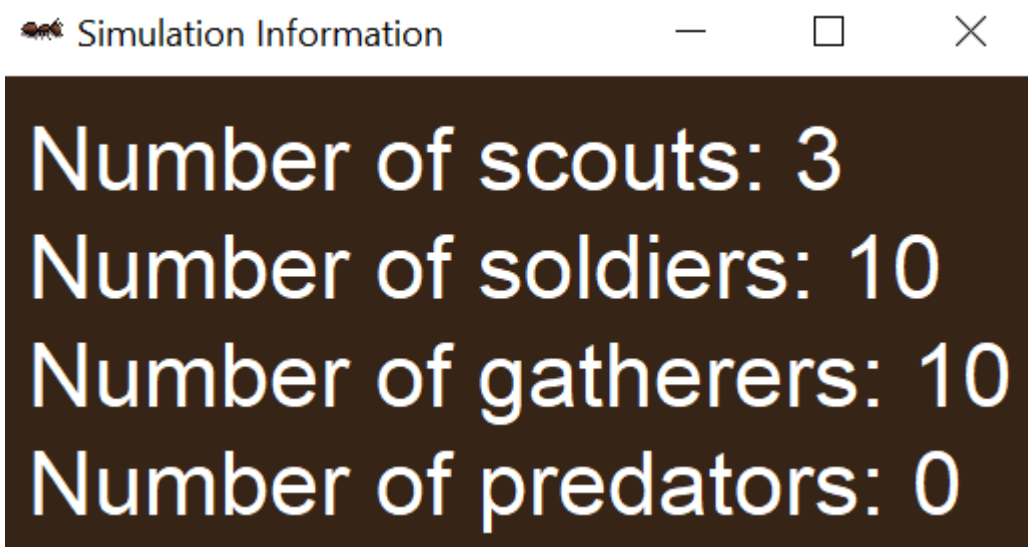
Une fois le programme lancé, vous verrez en premier l'écran titre. Celui-ci contient trois champs de texte qui permettent de choisir le nombre de fourmis scouts, gatherers et soldiers, respectivement. Il y a aussi un bouton "Start" qui permet de lancer la simulation avec les paramètres choisis.



Une fois que vous avez cliqué sur le bouton "Start", la simulation démarre. L'écran titre disparaît et est remplacé par l'écran de simulation. Celui-ci affiche une carte sur laquelle se déplacent les fourmis et les prédateurs, selon les comportements définis par le médiateur. En bas de la fenêtre se trouve un panneau de contrôle qui contient une liste déroulante qui permet de changer le médiateur en cours d'utilisation. Vous pouvez choisir entre les médiateurs `Balanced`, `Aggressive` et `Explore`. Les différents comportements induits par ces médiateurs sont définis dans les sections du rapport les concernant. Par défaut, le médiateur `Balanced` est utilisé. Sur le panneau de contrôle se trouve aussi un bouton `Colorblind Mode` qui permet d'afficher un code lexical pour les différents types de fourmis s'il est enclenché. Les fourmis scouts sont représentées par la lettre `S`, les gatherers par la lettre `G` et les soldiers par la lettre `F` (pour fighter). Si la fenêtre de simulation est fermée, la simulation s'arrête.



Enfin, il y a une deuxième fenêtre qui s'affiche lorsque vous cliquez sur le bouton "Start". C'est la fenêtre d'informations. Elle affiche le nombre de fourmis scouts, gatherers, soldiers et de prédateurs présents dans la simulation. Ces nombres sont mis à jour toutes les 1500 millisecondes pour éviter de surcharger la simulation. Cette fenêtre peut être fermée sans arrêter la simulation.



La simulation s'arrête lorsque vous fermez la ftre de simulation ou s'il n'y a plus de fourmis dans la simulation.

Vous trouverez ci-dessous une légende pour les différents objets de la simulation :

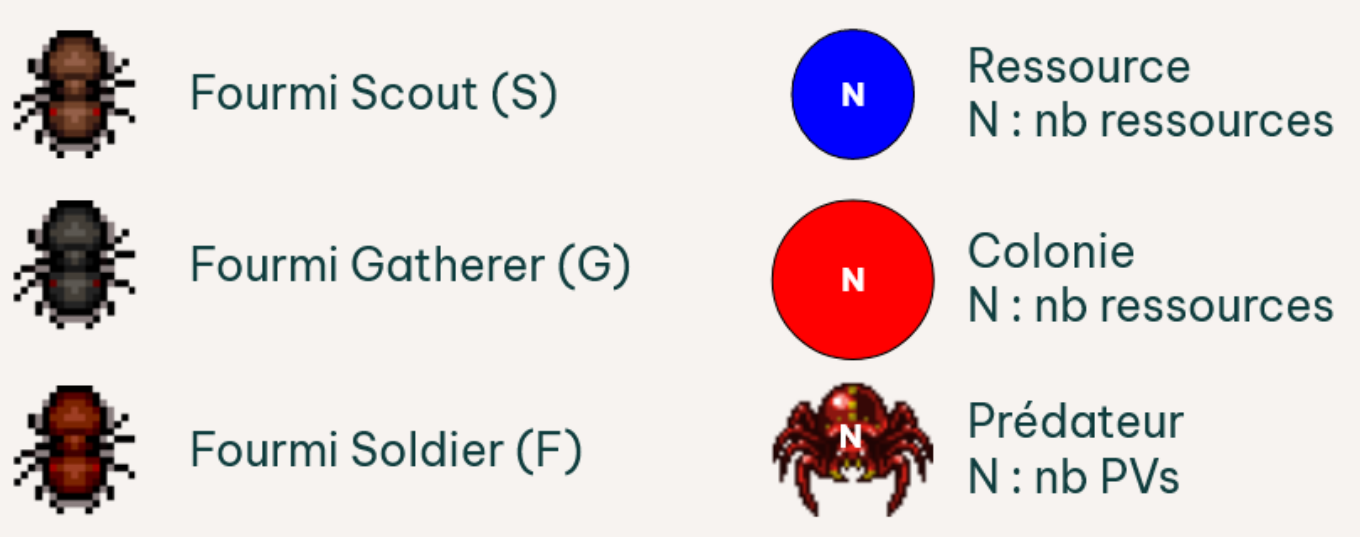
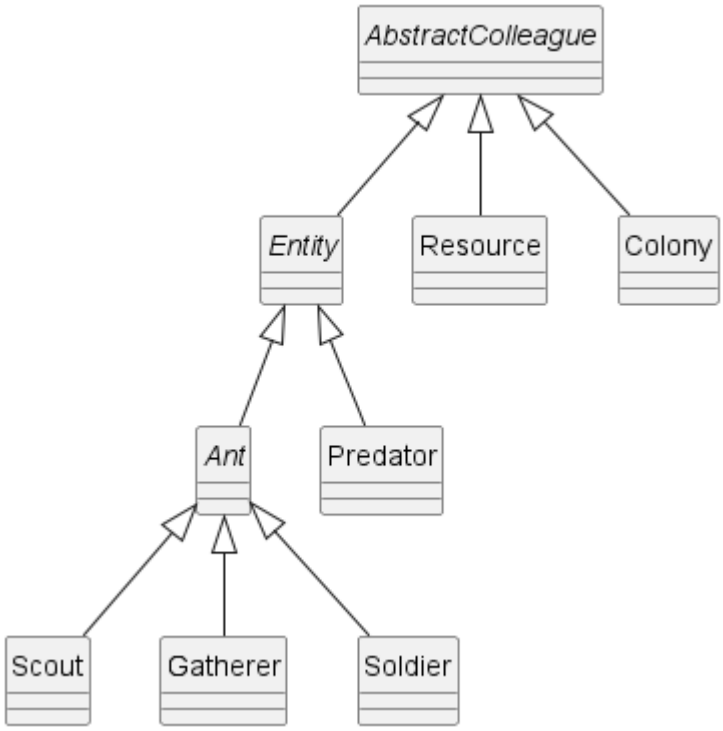


Diagramme de Classes

Le diagramme de classes étant assez conséquent et complexe, nous avons décidé de le fournir en annexe plutôt que l'intégrer directement au rapport. Le diagramme de classe est divisé en deux parties : la GUI et la partie contenant le médiateur, les collègues et la logique. Ces digrammes de classe se trouvent dans les fichiers suivants :

- GUI: GUI_uml.png
- Médiateur et collègues : MCR_Projet_uml.png

Collègues

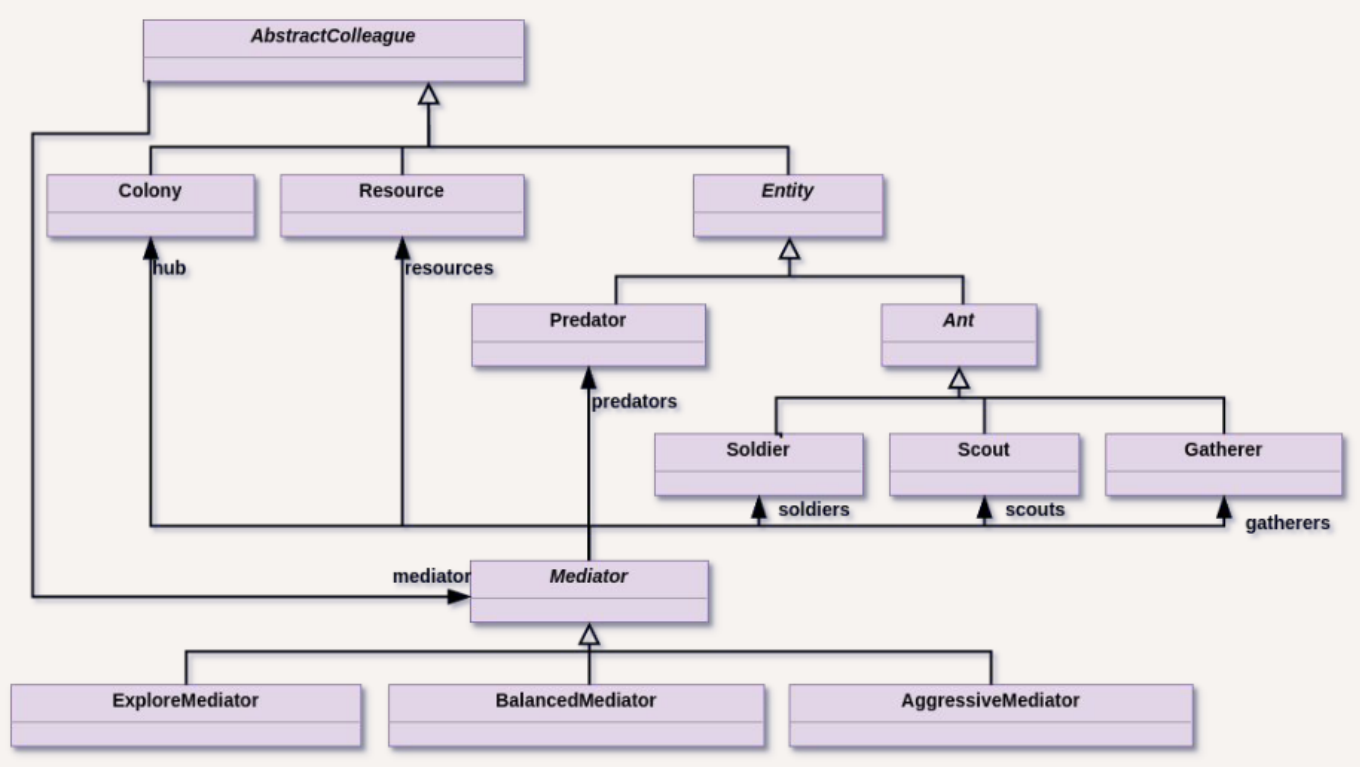


Les collègues concrets sont les objets qui interagissent entre eux via le médiateur. Ils ont tous des aptitudes et des utilités différentes. Voici une description de chaque collègue concret :

1. **Fourmis** : Les fourmis sont des entités qui peuvent interagir avec les ressources et les prédateurs via le médiateur. Elles sont divisées en trois types :
    - **Scouts** : Les Scouts sont des fourmis qui ont un grand champ de vision. Cela leur permet de repérer les ressources et les prédateurs à une plus grande distance. Elles sont donc bien adaptées pour explorer la carte.
    - **Gatherers** : Les Gatherers sont des fourmis qui ont peu de force et peu de points de vie (PVs). Leur rôle principal est de collecter des ressources et de les ramener à la colonie. Elles sont donc peu adaptées pour se défendre contre les prédateurs.
    - **Soldiers** : Les Soldiers sont des fourmis qui ont beaucoup de force et de PVs. Ils sont plus résistants et peuvent se défendre contre les prédateurs. Elles sont donc bien adaptées pour protéger la colonie.
2. **Predator** : Le Predator est une entité qui a beaucoup de force et énormément de PVs. Il peut attaquer les fourmis.
3. **Resource** : La Resource est une entité qui a un nombre de ressources disponibles. Les fourmis peuvent collecter ces ressources pour les ramener à la colonie.
4. **Colony** : La Colony stocke les ressources rapportées par les fourmis. C'est aussi là que les fourmis naissent, dans le cas où le médiateur est `Explore`.

Chaque collègue interagit avec les autres via le médiateur, ce qui permet de décentraliser la logique de communication et de rendre le code plus modulaire et plus facile à maintenir.

Mediator



La classe `Mediator` est le cœur de notre implémentation du design pattern du même nom. C'est elle qui gère et contrôle toutes les communications et interactions entre les différents collègues concrets de la simulation (fourmis, prédateurs, ressources, etc.).

Toute la logique de l'application se trouve centralisée dans cette classe. C'est elle qui décide de ce que chaque collègue doit faire à chaque step de la simulation. Elle possède des listes et des maps qui lui permettent de stocker les différentes entités et de les manipuler. Elle possède aussi des méthodes qui permettent de gérer les différentes actions des collègues.

Plusieurs spécialisations du `Mediator` ont été implémentées. Ces dernières peuvent être changées dynamiquement durant le runtime, sans modifier le code et sans relancer le programme. Le `Mediator` de chaque `Collegue` est mis jour en cours d'utilisation par celui qui a été choisi dynamiquement.

Nous avons décidé d'avoir une classe abstraite qui possède tous les attributs communs aux spécialisations de `Mediator`. Les méthodes communes aux différentes spécialisations y sont aussi définies. La plupart des attributs sont en `protected` afin que les spécialisations puissent y accéder sans définir des getters et des setters.

Communications

Le coeur du design pattern Mediator est la communication entre les collègues concrets. C'est le médiateur qui est l'équivalent d'un chef d'orchestre et qui décide de qui doit faire quelles actions en fonction des actions des autres entités. Comme on peut le voir dans le diagramme de classe, aucun collègue ne possède de liaison avec un autre collègue. Les seuls couples qui sont formés sont avec le `Mediator`. Pour gérer ces différentes communications et la connaissance des entités par le `Mediator`, nous avons utilisé des listes et des maps.

Listes et Map

Tous nos collègues concrets possèdent des `LinkedList` qui leur sont dédiées. Cela permet au `Mediator` de connaître chaque collègue et de pouvoir les manipuler. L'exemple concret d'utilisation dans le `Mediator` est de pouvoir itérer sur toutes les ressources et définir si oui ou non la fourmi est dans son champ de vision.

Nous avons aussi des `HashMap` qui permettent de stocker quels collègues sont assignés à quels autres collègues. En effet, nos fourmis peuvent avoir un objectif qui est une ressource ou la colonie. Cet objectif sera stocké et le médiateur aura l'information, à tout instant, de qui est assigné à qui. Cela permet de centraliser la logique dans le `Mediator` et de ne pas avoir de couplage entre les collègues.

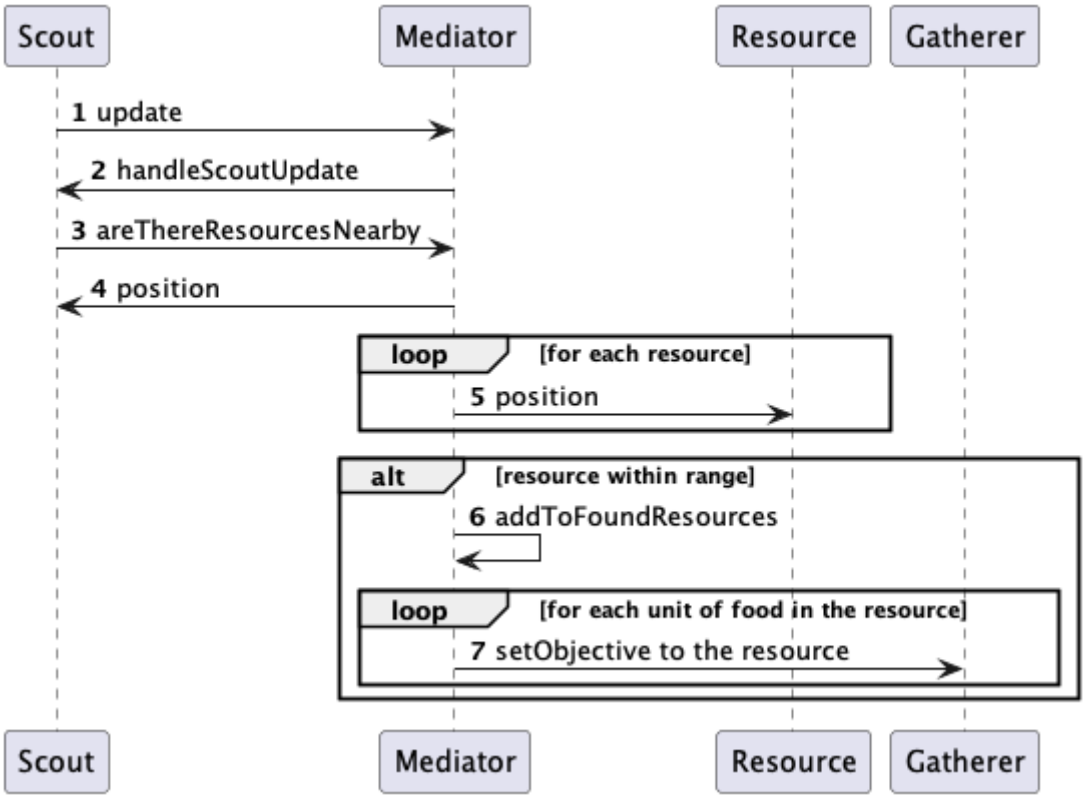
Voici les listes et maps utilisées dans le `Mediator` :

- `HashMap<Entity, Entity> targets` : Stocke les targets que les entités ont. La clef est l'entité attaquante et la valeur est l'entité attaquée.
- `HashMap<Entity, AbstractColleague> objectives` : Stocke les objectifs des entités. La clef est l'entité et la valeur est le collègue à atteindre.
- `LinkedList<Gatherer> gatherers` : Stocke les fourmis de type `Gatherer`.
- `LinkedList<Scout> scouts` : Stocke les fourmis de type `Scout`.
- `LinkedList<Soldier> soldiers` : Stocke les fourmis de type `Soldier`.
- `LinkedList<Resource> resources` : Stocke les ressources.
- `LinkedList<Predator> predators` : Stocke les prédateurs.
- `Colony colony` : Référence sur l'unique colonie de la simulation.

- `LinkedList<Resource> resourcesFound` : Stocke les ressources trouvées par les fourmis.
- `LinkedList<Predator> predatorsFound` : Stocke les prédateurs trouvés par les fourmis. Uniquement utilisées dans le médiateur `Aggressive`.

Flux de communication

Vous trouverez ci-dessous un diagramme de séquence illustrant le flux de communication entre le `Mediator` et un `Scout` durant un step de la simulation. Le `Scout` est en train de se déplacer sur la carte et le `Mediator` doit décider si ce dernier voit ou non une ressource. Si c'est le cas, le `Mediator` doit assigner n `Gatherer` pour aller chercher la ressource. n étant le nombre d'unités de nourriture que la ressource possède. Le `Mediator` ajoute cette ressource dans la liste des ressources trouvées.



Des communications du même type sont effectuées pour à la découverte de prédateurs, pour notifier les `Soldier` d'aller attaquer les prédateurs, ou d'autres types de fourmis en fonction du médiateur utilisé.

Stratégies

Au cours de notre implémentation du `Mediator`, nous avons rencontré un problème au niveau des comportements que nous souhaitions attribuer aux différentes fourmis. Nous avions des fonctions comportementales au sein des fourmis. Ce fait nous dérangeait, comme le principe derrière le design pattern Mediator est de centraliser la logique dans le `Mediator`. En effet, ce dernier est un pattern de type comportemental et c'est bien lui qui doit décider quel comportement doit être appliqué à quel moment à quelle entité.

Nous avons décidé de créer une abstraction des mouvements que nous avons nommée `MovementStrategy`. C'est une interface fonctionnelle fournissant la fonction `execute()` qui sera redéfini dans les différentes implémentations. Ce choix nous permet de mieux respecter le principe de l'OCP, comme chaque implémentation a sa responsabilité et ne sera pas changée. Qui plus est, s'il y a un souhait d'ajouter une

stratégie quelconque, cela est possible en implémentant une nouvelle fois l'interface fonctionnelle dans une nouvelle classe.

Le `Mediator` se charge de définir quelle stratégie est assignée à quelle fourmi. Ainsi, les stratégies sont des sortes de sous-traitants de `Mediator` en permettant une encapsulation du code dans un composant approprié. Une `HashMap` permet de stocker quelle fourmi possède quelle stratégie. Les clefs sont créées lors de la première insertion, mais ne sont jamais changées par la suite. Uniquement les valeurs ont un intérêt à être changées pour mettre à jour quelle est la stratégie applicable à l'entité qui est la clef.

Balanced Mediator

Ce médiateur représente une colonie de fourmis équilibrée. C'est une simulation qui se veut "réaliste" dans le sens où les fourmis ont des comportements qui se rapprochent de ceux que l'on peut observer dans la nature. Les comportements propres à ce médiateur sont les suivants :

- Les fourmis scouts se déplacent aléatoirement sur la carte à la recherche de ressources et de prédateurs. Lorsqu'elles trouvent un prédateur, elles fuient jusqu'à ce qu'elles ne le voient plus.
- Les fourmis gatherers vont chercher des ressources, si elles ont été assignées à le faire, et les ramènent à la colonie. Si elles rencontrent un prédateur, elles fuient jusqu'à ce qu'elles ne le voient plus.
- Les fourmis soldiers vont attaquer les prédateurs si elles ont été assignées à le faire. Sinon, elles restent à la colonie.
- Lorsqu'une ressource est découverte, le bon nombre de gatherers est assigné pour aller la chercher (s'il y en a assez qui n'ont pas déjà un objectif). Les resources sont ramenées à la colonie dans leur ordre de découverte.
- Lorsqu'un prédateur est découvert par une fourmi scout, toutes les fourmis soldiers qui n'ont pas de cible vont se voir assigner ce prédateur comme cible.
- Lorsqu'un prédateur est tué, les fourmis qui l'attaquaient retournent à la colonie.
- Les ennemis apparaissent un peu moins fréquemment que les ressources, mais assez pour que la situation soit considérée comme équilibrée.

Aggressive Mediator

Ce médiateur représente une colonie de fourmis agressives. Les comportements propres à ce médiateur sont les suivants :

- Les fourmis scouts se déplacent aléatoirement sur la carte à la recherche de prédateurs (et de ressources).
- Les fourmis gatherers vont attaquer un prédateur si elles ont été assignées à le faire. Sinon, elles restent à la colonie.
- Les fourmis soldiers vont attaquer un prédateur si elles ont été assignées à le faire. Sinon, elles restent à la colonie.
- Seules les fourmis scouts fuient.
- Lorsqu'un prédateur est découvert par une fourmi scout, toutes les fourmis qui n'ont pas de cible vont se voir assigner ce prédateur comme cible.
- Lorsqu'un prédateur est tué, les fourmis se voient assigner un nouveau prédateur à attaquer s'il en reste qui ont été découverts. Sinon, elles retournent à la colonie.
- Lorsqu'une ressource est découvert, rien ne se passe. Aucune fourmi ne va chercher de ressources, ici, le but est de

- survivre.
- Les ennemis apparaissent très fréquemment et les ressources très rarement.

Explore Mediator

Ce médiateur représente une colonie de fourmis exploratrices dans un monde où les ressources sont abondantes. Les comportements propres à ce médiateur sont les suivants :

- Les fourmis scouts se déplacent aléatoirement sur la carte à la recherche de ressources et de prédateurs. Lorsqu'elles trouvent un prédateur, elles fuient jusqu'à ce qu'elles soient en sécurité.
- Les fourmis gatherers se se déplacent aléatoirement sur la carte à la recherche de ressources et de prédateurs. Lorsqu'elles trouvent un prédateur, elles fuient jusqu'à ce qu'elles soient en sécurité. Elles vont chercher des ressources et les ramènent à la colonie si elles sont assignées à le faire.
- Les fourmis soldiers se déplacent aléatoirement sur la carte à la recherche de ressources et de prédateurs. Elles attaquent un prédateur si elles ont été assignées à le faire.
- Les ressources apparaissent très fréquemment et les ennemis très rarement.
- Lorsqu'une ressource est découverte, le bon nombre de gatherers est assigné pour aller la chercher (si il y en a assez qui n'ont pas déjà un objectif). Les resources sont ramenées à la colonie dans leur ordre de découverte.
- Lorsqu'un prédateur est découvert par une fourmi scout, toutes les fourmis soldiers qui n'ont pas de cible vont se voir assigner ce prédateur comme cible.
- Lorsque 6 ressources ont été ramenées à la colonie, une nouvelle fourmi naît. S'il n'y a aucun prédateur sur la carte, c'est une fourmi gatherer qui naît. Sinon, c'est une fourmi soldier.
- Lorsqu'un prédateur est tué, les fourmis qui l'attaquaient retournent à la colonie.

GUI (Graphical User Interface)

L'interface graphique a été réalisée en utilisant Swing. Elle permet de visualiser la simulation en temps réel.

L'interface graphique fonctionne de manière assez simple. Il y a une classe `SimulationDisplayer` qui implémente l'interface `Displayer` et possède le JFrame principal. La première chose affichée est l'écran titre, représenté par la classe `TitleScreen` qui est un JPanel. C'est donc d'abord uniquement ce panel qui est ajouté au JFrame. Lorsque l'utilisateur clique sur le bouton "Start", le panel `TitleScreen` est retiré et les panels `SimulationPanel` et `ControlPanel` sont ajoutés. Le premier affiche la simulation et le second permet de contrôler la simulation.

L'interface graphique possède aussi une deuxième fenêtre, représentée par la classe `InformationFrame`. Cette fenêtre s'affiche lorsque l'utilisateur lance la simulation en appuyant sur le bouton "Start". Elle affiche le nombre de fourmis scouts, gatherers, soldiers et de prédateurs. Celà permet à l'utilisateur de suivre l'évolution de la simulation.

En ce qui concerne l'affichage des collègues dans la simulation, elle est gérée par différent Renderer. Pour chaque classe collègue concrète, il y a une classe Renderer associée qui implémente l'interface `Renderer`. Ce sont les collègues eux-mêmes qui possèdent l'information de quel Renderer utiliser pour les afficher. Cette information est récupérable grâce à la méthode `getRenderer()`, qui est redéfinie dans chaque classe de collègue concret. Cette manière de procéder se rapproche de l'utilisation du design pattern fabrication et nous a été inspirée par le second laboratoire de MCR.

Chaque élément affichable de la simulation implémente l'interface `Displayable`. Cette interface possède une méthode `draw()` qui est implémentée par la classe `AbstractColeague` et qui appelle donc la méthode `getRenderer()`. C'est ensuite la classe `Mediator` qui s'occupe d'afficher les collègues concrets dans sa méthode `display()`, en appelant la méthode `draw()`.

Collisions et déplacements

Pour simplifier les mouvements et les calculs, les positions et les vecteurs de directions sont des entiers. Les entités se déplacent pixel par pixel dans huit directions différentes, en diagonale ou en ligne droite. Pour ce faire, nous avons implémenté une classe utilitaire `Vector2` qui permet de représenter un vecteur 2D de position ou de direction et qui donne accès à des méthodes pour effectuer des opérations sur ces vecteurs. Cette classe possède également une enumération `POSSIBLE_DIRECTIONS` pour représenter les huit directions existantes.

Pur rendre la simulation plus réaliste, nous avons décidé d'implémenter une gestion des collisions entre les entités. Les collisions ont été implémentées d'une manière assez simple. Lorsqu'une entité se déplace, elle demande au médiateur de valider son déplacement. Le médiateur vérifie si la nouvelle position demandée est en collision avec une entité et si c'est le cas, vérifie que la distance avec l'entité actuellement en collision est plus faible que précédemment. Une gestion similaire est utilisée pour la collision avec les bords de la map. Dans le cas d'une collision, l'entité va essayer d'éviter d'entrer en collision avec l'entité. Elle essaiera en priorité de se diriger orthogonalement à la direction de la collision. Si cela n'est pas possible, elle essaiera de se diriger dans toutes les directions possibles jusqu'à ce qu'une soit valide ou, le cas contraire, se stoppera. Cette gestion permet de mieux visualiser les déplacements des différentes entités sans qu'elles ne se superposent tout en leur permettant de s'éviter. Il peut arriver que les entités mettent un certain temps à se débloquer si elles se collisionnent mutuellement, cela n'est pas un bug, mais pourrait être une piste d'amélioration future.

Simulation

C'est la classe `Simulation` qui s'occupe de gérer la simulation. C'est elle qui possède le `Mediator` en cours d'utilisation. Elle possède une méthode `run()` qui définit la boucle principale de la simulation. C'est dans cette méthode que sont appelées les méthodes `update()` et `display()` du `Mediator`. Cette boucle est créée à l'aide d'un timer et les méthodes citées précédemment sont appelées à chaque tick du timer. La méthode `run` n'est pas appelée directement dans le main mais lorsque l'utilisateur clique sur le bouton "Start" de l'interface graphique.

Cette classe fournit aussi différents getters qui permettent de récupérer le nombre de collègues de chaque type, ce qui est utile pour que la classe `InformationFrame` puisse récupérer et afficher ces informations.

La classe `Simulation` possède aussi des méthodes qui permet de changer le `Mediator` en cours d'utilisation. Cela permet de changer la stratégie de communication des collègues en cours de simulation, de manière dynamique. C'est grâce à cela que le médiateur en cours d'utilisation change lorsqu'on en sélectionne un autre dans la liste déroulante de l'interface graphique.

Possibilités d'Amélioration

Une piste d'amélioration possible serait d'ajouter des mediators de types différents qui communiquent entre eux. Par exemple un médiator pour les comportements et un autre qui gèrerait un contexte plus global de la simulation, comme la météo. Une autre idée aurait pu être d'avoir un médiator qui gère les fourmis et un autre qui gère les prédateurs ou encore un mediator par type de fourmi qui représenterait le "chef" de chaque

type de fourmi. Cela permettrait de varier plus facilement les comportements des différents types de fourmis de manière plus indépendante et rendrait la simulation plus riche.

Une seconde amélioration aurait été d'avoir une plus grande diversité de `Colleague`. Cela permettrait d'avoir de plus amples communications entre les différents collègues et de rendre la simulation peut-être plus réaliste ou tout du moins plus complexe et intéressante.

Enfin, une idée intéressante serait d'améliorer la GUI en ayant une indication visuelle de lorsque la fourmi porte une ressource. On pourrait aussi imaginer une petite animation lorsqu'une entité subit ou fait une action. Par exemple, on pourrait imaginer que quand une fourmi scout trouve une ressource, un signal visuel, comme un petit haut-parleur avec un texte, apparaisse afin d'imager la découverte et la communication entre les fourmis. Ces idées auraient permis une meilleure compréhension de la simulation.

Conclusion

Ce projet de simulation de colonie de fourmis illustre de manière concrète et interactive l'application du design pattern Mediator. En centralisant la communication entre les différentes entités de la simulation au sein du médiateur, nous avons pu créer un système flexible et évolutif. Bien que la mise en place du pattern `Mediator` dans notre projet n'a vraiment pas été simple. Néanmoins, une fois réalisée, il a été très facile de rajouter de nouveaux médiateurs qui définissent de nouveaux comportements. Les différents médiateurs implémentés (`Balanced`, `Aggressive`, `Explore`) démontrent la capacité du pattern à gérer des comportements variés et complexes, tout en restant modulaire et facile à maintenir. Il serait d'ailleurs très simple d'ajouter encore plus de nouveaux `Mediator` pour diversifier les comportements. Le pattern nous a également permis d'éviter un couplage trop fort entre les `Colleague` de la simulation, en les rendant indépendants les uns des autres et en nous permettant de modifier leur comportement de manière dynamique sans devoir modifier les différents `Colleague`.

Dans notre projet, si le design pattern mediator n'était pas utilisé, chaque `Colleague` devrait probablement posséder une quantité significative de références sur les `Colleague` composant la simulation afin de pouvoir connaître leurs emplacements. Cela lui serait nécessaire pour décider de son comportement en fonction de son environnement. Au-delà de son comportement, chaque `Colleague` communique avec les autres `Collegue`, ce qui implique que chaque `Colleague` devrait avoir des fonctions dédiées à ce but. Ces deux points auraient rendu le code plus complexe, moins modulable et le comportement des `Colleagues` bien plus difficiles à modifier.

L'interface graphique Swing, bien que simple, offre une visualisation intuitive de la simulation, permettant à l'utilisateur d'observer les interactions entre les fourmis, les prédateurs et les ressources. De plus, la possibilité de changer de médiateur en temps réel met en évidence la flexibilité du système et l'impact direct des différentes stratégies de communication sur le comportement global de la colonie.