Note : Les fichiers de configuration des conteneurs sont dans le dossier `resources`. Ils sont récupérés via la méthode usuelle `getResource()`, il faut donc les ajouter dans le projet. Pour lancer la console puis la résolution, il faut d'abord lancer `JadeConsoleBoot` puis `SudokuBoot`. Le `DF` est utilisé pour contacter les agents. Les constantes utilisées sont dans la classe `utils.Contants`. Les cinq grilles sont définies dans cette dernière, et le choix de la grille à résoudre se fait dans la classe `EnvironmentAgent`. La Javadoc est dans le dossier `doc`.

1 Rôle des agents

Le solveur de Sudoku est implémenté via trois types agents : agent de **simulation**, agent d'**environnement** et agent d'**analyse** – dont plusieurs instances sont exécutées.

1.1 Agent de simulation

L'agent de simulation est en charge de la coordination de la résolution des grilles de Sudoku. Il connaît les agents qui travaillent concrètement sur la résolution ainsi que l'agent qui est en charge des opérations sur les grilles. Il demande de lancer une itération de l'algorithme de résolution à intervalles réguliers. Il est à noter que cet agent est générique, dans la mesure où il n'a pas directement connaissance de l'objet concerné par la simulation.

1.2 Agent d'environnement

L'agent d'environnement est en charge du stockage, de la mise et à jour et du maintien de l'état de la grille de Sudoku. Il ne travaille pas à la résolution, mais se place comme intermédiaire entre les phases de simulation et d'analyse. C'est lui qui est capable de modifier la grille pour s'adapter aux changements induits par l'analyse. Il est aussi en charge d'informer la simulation lorsque la grille est résolue.

1.3 Agent(s) d'analyse

Les agents d'analyse implémentent les quatre algorithmes de résolution du Sudoku. Un agent travaille sur neuf cellules, qui représentent indifféremment une colonne, une ligne ou un carré de la grille. Tous les agents s'enregistrent auprès de la simulation puis travaillent sous l'impulsion de l'environnement. Par convention, un agent est créé pour chaque entité de la grille (ligne, colonne, carré), soit 27 agents d'analyse au total.

2 Tâches des agents

2.1 Behaviours de la simulation

L'agent de simulation implémente trois behaviours simples encapsulés dans des behaviours composites. Le behaviour de plus haut niveau est de type `SequentialBehaviour`, implémenté par `SimulationBehaviour`. Il est composé de :

1. Un behaviour simple, `WaitRegistrationBehaviour`, qui attend l'enregistrement de tous les agents d'analyse et les stocke dans une `Map`. Ce behaviour termine lorsque tous les agents se sont manifestés.

2. Un behaviour **parallèle**, `PerformSimulationWrapper`, qui gère la simulation effective, composé de deux sous-behaviours [1] :
   — Un behaviour *à horloge*, `PerformSimulationBehaviour`, qui envoie à intervalles réguliers (défini par une constante) une demande à l'agent d'environnement pour chaque agent d'analyse, en précisant son indice et son nom ;
   — Un behaviour simple, `WaitEnvironmentBehaviour`, qui attend simplement une notification de l'agent d'environnement indiquant que la simulation est terminée (*i.e.* la grille est résolue). Il peut alors stopper le behaviour à horloge.

2.2 Behaviours de l'environnement

L'agent d'environnement implémente deux behaviours simples encapsulés dans un behaviour séquentiel (`EnvironmentBehaviour`). Ce dernier exécute, dans l'ordre :

1. Un behaviour simple, `AdvanceSimulationBehaviour`, qui reçoit les demandes de l'agent de simulation puis communique avec les agents d'analyse. Il appelle les méthodes métiers de la grille pour récupérer les données ou les mettre à jour. Il termine lorsque la grille est résolue (*i.e.* quand toutes les cellules ont une valeur).
2. Un `OneShotBehaviour`, `SendEndNotificationBehaviour`, qui envoie une notification de fin à l'agent de simulation.

2.3 Behaviours de l'analyse

L'agent d'analyse implémente deux behaviours simples encapsulés dans un behaviour séquentiel (`AnalyseBehaviour`). Ce dernier exécute, dans l'ordre :

1. Un `OneShotBehaviour`, `RegisterBehaviour`, qui envoie un message à l'agent de simulation pour signaler sa présence.
2. Un behaviour cyclique, `ResolutionBehaviour`, qui reçoit les demandes de l'environnement, effectue **zéro ou une modification** [2], puis répond avec les données mises à jour.

3 Types de messages

Tous les contenus des messages envoyés sont implémentés sous forme d'objets simples (champs et accesseurs) héritant de la classe abstraite `Model`, qui fournit les méthodes de sérialisation et de désérialisation JSON.

Les agents d'analyse s'enregistrent auprès de l'agent de simulation avec des messages de type `Subscribe` contenant leur nom (`RegisterModel`). L'agent de simulation envoie des demandes à l'agent d'environnement avec des messages de type `Request` contenant l'indice [3] et le nom de l'agent d'analyse (`SimulationModel`). L'agent d'environnement et les agents d'analyses communiquent par des séquences `Request` – `Inform` et avec le modèle `AnalyseModel`, contenant les 9 cellules sur lesquelles appliquer les algorithmes. Dans ce cadre, l'indice sert d'*identifiant de conversation*. Enfin, l'agent d'environnement notifie la simulation de la fin de la résolution avec un message vide de type `Inform`.

1. Notons qu'il aurait été possible et équivalent d'encapsuler le `TickerBehaviour` dans un `OneShotBehavioour` pour déclencher le dernier behaviour d'attente, mais cela induit une complexité supplémentaire concernant la gestion des références pour arrêter le behaviour à horloge une fois la simulation terminée.
2. L'idée est de rendre l'action la plus courte possible, plutôt que de boucler sur les méthodes de résolution chaque fois qu'il y a une modification, ce qui peut prendre du temps.
3. Par convention, les 9 premiers nombres représentent les lignes, les 9 suivants les colonnes, les 9 suivants les carrés.

4 Indépendance des agents d'analyse

Même si les agents d'analyse étaient situés sur 27 stations différentes, la résolution serait encore possible. En effet, chaque agent d'analyse travaille sur une liste de neuf chiffres, **sans contexte**, et sans environnement. Les algorithmes s'appliquent indifféremment sur des lignes, des colonnes, ou des carrés, sans besoin de connaître le contenu de la grille ou d'autres entités. Il est donc possible de distribuer ces agents.