# Fonctionnalités

Le programme que nous proposons permet à une IA de jouer au jeu Pandémie mais avec des règles du jeu simplifié.

Seul un joueur peut jouer. En outre les différents rôles des joueurs, les cartes joueurs évènements spécial et les stations de recherche ne sont pas disponibles.

Aussi, le Partage d’information entre joueurs n’est pas possible.

Deux IAs sont proposées. Une IA aléatoire qui choisit ses actions au hasard. Une IA qui implémente une version modifiée de l’algorithme H-Minimax.

Un affichage sur console montrant le déroulé du jeu est proposé. Une Interface graphique basique est également disponible. Dans la version actuelle du programme, elle ne fait qu’installer le joueur à Atlanta et crée la pile des cartes joueurs.

# Architecture

## Diagramme de classe

Le programme s’articule autour des 3 classes essentielles suivantes GameEngineForSimulation, SimulationAI, Player.

La GameEngineForSimulation contient tous les éléments du jeu : le plateau représenté par la classe Board, les joueurs représentés par un champs List<Player>.

En appelant la méthode createPlayersAndOtherStuff() elle charge et crée tous les éléments du jeu.

Puis en appelant la méthode setup(), elle installe les éléments du jeu, par exemple, donner les mains pour chaque joueur, ou construire la pile des cartes joueur.

La classe SimulationAI est la classe de base de toutes les classes qui gère l’intelligence articielle du jeu.

La classe Player represente un joueur.

## Le jeu

Les éléments du jeu, joueurs, pions, cartes, sont repartis entre la classe Board et la GameEngine.

En règle général, tous les éléments qui sont dans le monde physique associés au tableau, comme les pions ou les cartes, sont situés dans la classe Board. Et tous les éléments en rapport avec le jeu, comme les joueurs sont installés dans la classe GameEngine.

Puisque la majorité des éléments (les cartes par exemple) sont piochés d’un tas dans le jeu physique, nous avons choisi d’implémenter ces éléments par une Stack. Dans ce cas, le retrait de élément en haut de la pile se fait par une instruction très simple stack.pop(), ce qui est largement plus lisible et plus convenable que la même opération lorsqu’elle effectuée sur une list (list.get(list.size()-1).

La collection des cartes d’infections est accédée à partir de ses deux extrémités. En effet, lors de la mise en place, lorsque nous devons d’abord infecter 9 villes, les cartes sont tirées du haut du paquet.

Par contre, à l’étape d’infection de la ville (la deuxième d’un tour), la carte mentionnant la ville à infectée est tirée du bas de la pile. Pour efficacement adresser cette caractéristique, nous avons choisi d’implémenter cette pile par une Deque.

## Déroulement du jeu

Pour réaliser le jeu, une communication principalement entre la classe GameEngine et ThreadAI est établie. Ces deux classes sont des Thread.

La GameEngine via sa méthode loop() fait exécuter le jeu.

Pour faire communiquer le thread de l’IA avec celui de la GameEngine, nous utilisons le mécanisme suivant.

Au départ, dès que l’IA est instanciée, elle fait un sleep.

Puis un processus représenté par le pseudo code suivant est exécuté par la GameEngine dans la methode loop().

Tant que le jeu n’est pas fini

Passer au prochain joueur (Dans notre cas il y a un seul joueur)

Exécuter la méthode ThreadAI.playTurn dont le but est de réveiller le thread de l’IA

Et juste après le thread de la GameEngine s’endort pour une durée égale à turnDuration secondes

Si la GameEngine n’a pas été reveillé par le thread de l’IA, la GameEngine sort de son sleep d’elle même, et pose le flag interrupt de la l’IA.

Sachant que ThreadAi a deux champs p, g, qui sont équivalents aux paramètres de playTurn, Lorsque la gameEngine appelle ThreadAI.playTurn avec les paramètres p, g, voilà ce qui est exécuté :

go = 1;

Poser les paramètres dans les champs correspondants.

Réveiller le Thread de l’IA

Le thread de l’IA quand à lui effectue sa tache représentée par le pseudo code suivant :

Faire tant que

Faire tant que go = null

Dormir

Executer la méthode playTurnInternal

Reveiller le thread de la GameEngine

Mettre go = null

En résumé, pour réaliser AiInterface.playTurn, le thread de la gameEngine donne une valeur à go et réveille le thread de l’IA. Puisque l’IA s’endort de nouveau seulement lorsque go est null, et comme ce dernier a une valeur, l’IA saute l’instruction qui la fait dormir et exécute playTurnInternal . playTurnInternal effectue exactement le travail qui doit etre fait par playTurn selon l’API.

# IA

## Architecture l’IA

Il y a essentiellement deux types d’IA : une IA qui est utilisé pour le choix des actions, que nous appeleront l’IA du jeu.

Et une autre qui sert à la simulation, SimulationAI.

Comme expliqué dans la section précédente, L’IA du jeu est un Thread, contrairement à l’autre.

L’IA du jeu est celle qui implémente AiInterface. C’est avec elle que communique le Thread de la GameEngine.

La différence entre les deux se situe au niveau de la methode playTurn.

Dans le cas de l’IA du jeu elle implémente playTurn de AiInterface. Elle a donc naturellement les paramètres GameInterface, et PlayerInterface.

Dans le cas de l’IA de simulation en plus de ces paramètres, elle inclue l’action candidate et les coordonnées qui localisent l’endroit ou cette action doit être effectuée dans l’execution.

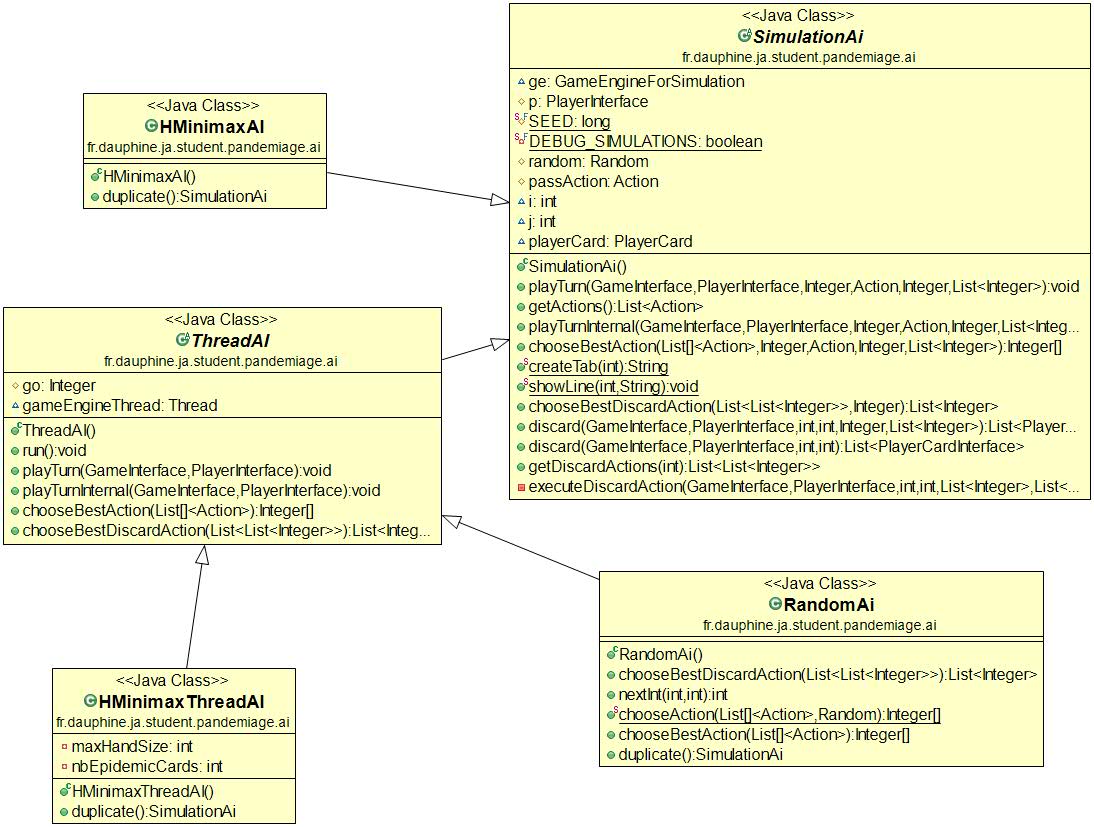


Figure 1: Diagramme de classe de tous les éléments en charge de réaliser l'IA du jeu

## Implémentation

Nous proposons 2 IA de jeu : RandomAI et HMinimaxThreadAI.

RandomAI choisi ses actions de manière aléatoire.

HMinimaxThreadAI implémente l’algorithme minimax à une différence près, il n y a pas le nœud min.









Elle utilise une heuristique pour ne pas exécuter le jeu jusqu’aux nœud terminaux.

Pour évaluer une action, on fait appel à la méthode loop() de GameEngineForSimulation. Dans cette fonction, les méthodes de la classe SimulationAi sont majoritairement appelés. SimulationAi permet de faire des simulations, c’est-à-dire dérouler le jeu jusqu’à une certaine profondeur si une action donnée est executée.

On donne à loop l’action à évaluer ainsi que la coordonnée de cette action dans le graph des états-actions du jeu. Grace à cette coordonnée, les méthodes de SimulationAi savent sauter toutes les instructions en amont et ne commence l’exécution qu’au bon endroit. A partir de cet endroit une copie de l’environnement est faite, puis une simulation est effectuée jusqu’à la fin du jeu (victoire, défaite) ou jusqu’à ce que la limite de profondeur est atteinte.

L’heuristique utilise plusieurs caractéristiques de l’état du jeu pour l’évaluer. Parmi ceux-ci on peut citer,

* Le nombre de foyer d’infections
* La taille de la pile des cartes joueurs (on espère ainsi éloigner la progression du jeu de l’état fatal pour lequel la pile vide)
* Le nombre de carte d’une couleur (on espère ainsi pousser le jeu vers un état ou il y a suffisamment de cartes pour pouvoir découvrir un remede)

La methode GameEngineForSimulation.computeUtility effectue ce calcul.