Projet Logiciel Transversal

HIVE

Table des matières

1 Objectif

1.1 – Présentation générale

Le but de ce projet est de réaliser le jeu de plateau et de stratégie “HIVE”.

1.2 – Règles du jeu

Hive est un jeu de stratégie sur plateau à cases hexagonales où 2 adversaires s’affrontent. Chaque joueur possède 13 pions qui ont leurs propres caractéristiques. Le but du jeu étant d’encercler la **Reine Abeille** adverse.
Il y a différents pions et chacun d’entre eux peut se mouvoir selon des règles spécifiques. Voici la liste de tous les pions et de leur caractéristique:

-Reine Abeille : elle se déplace d’un espace à la fois et doit être placée avant le cinquième tour (exclu).

-L’Araignée : elle se déplace de trois espaces.

-La Fourmi : elle se déplace d’autant d’espaces que le joueur le désire mais doit obligatoirement être voisine d’un autre pion.

-Grillon : il se déplace en sautant en ligne droite par-dessus d’une ou plusieurs autres pièces, jusqu’au premier espace libre. Elle se déplace dans la direction d’un de ses côtés, et non vers ses angles.

-Le Scarabée : il se déplace d’un espace à la fois, comme la reine, mais il a la capacité de « grimper » sur les autres pièces.

-Le Moustique : il prend le mode de déplacement de la ou des pièces adjacentes.

-La Coccinelle : elle se déplace de trois espaces à chaque tour, mais doit obligatoirement effectuer ses deux premiers déplacements en montant au-dessus des autres pièces (comme le scarabée) puis redescendre au sol lors de son troisième déplacement.

De plus, chaque pion doit obligatoirement être lié à au moins un autre : de sorte à ce qu’on ait un seul groupe d’insectes collés aux autres. On négligera la règle de liberté de mouvement.

1.3 – Conception Logiciel

Voici les textures de tous les pions pour chaque joueurs:

Figure 1 : Représentation des pions

2 – Description et conception des états

2.1 Description des états

Une partie est composée de 2 joueurs chacun ayant 11 pions. L'état du jeu dépend du nombre de tours et des pions placés sur un plateau.

Jeu :
Le jeu se déroule en plusieurs étapes. Les joueurs ont 4 tours pour placer leur pions "abeille" . Dès qu'un joueur a posé ce pion, il aura la possibilité,selon les règles, de déplacer ses pions déjà placés. Lorsqu'une des abeilles est encerclée la partie est terminée et le joueur ayant son abeille pas encerclée a gagné.

Joueur :
Nous avons 2 joueurs. Chaque joueur à une liste de pions placés et de pions non placés.
La couleur des pions désigne le joueur qui va commencer en premier.

Plateau :
Le plateau avec une grille hexagonale sera représenté par un tableau en 2 dimensions. Chaque case du plateau a donc une coordonnée et un état occupé ou non.
Lorsqu'un pion est placé sur une des cases, la case est alors occupée.

Insectes :
Nous avons donc au total 22 pions/insectes. Ils seront différenciés pour chaque joueur à l'aide de leur couleur. Un insecte a une position, c'est à dire ici une coordonnée sur le plateau s'il est placé ou bien "NULL" s'il ne l'est pas.

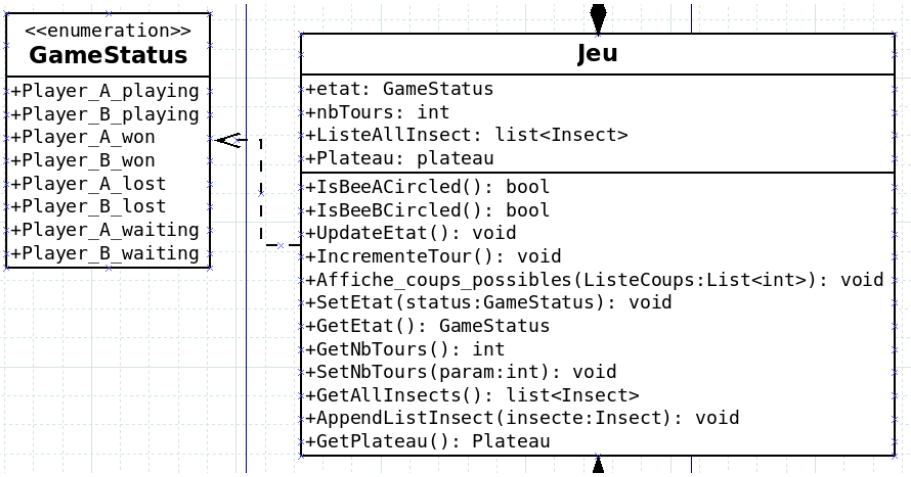
Les différents états du jeu seront :

- Player_X_playing : lorsque c'est le tour d'un des joueur
- Player_X_won : lorsque la partie est terminée, ce qui signifie qu'une des abeilles a été encerclée.

2.2. Conception logiciel

Après avoir décrit les différents états, nous allons aborder le diagramme de classe présenté en figure 2. On peut alors décrire plus précisément les classes que nous avons instauré pour le jeu et voir les attributs et méthodes qu'elles contiennent.

- Classe **Jeu :**

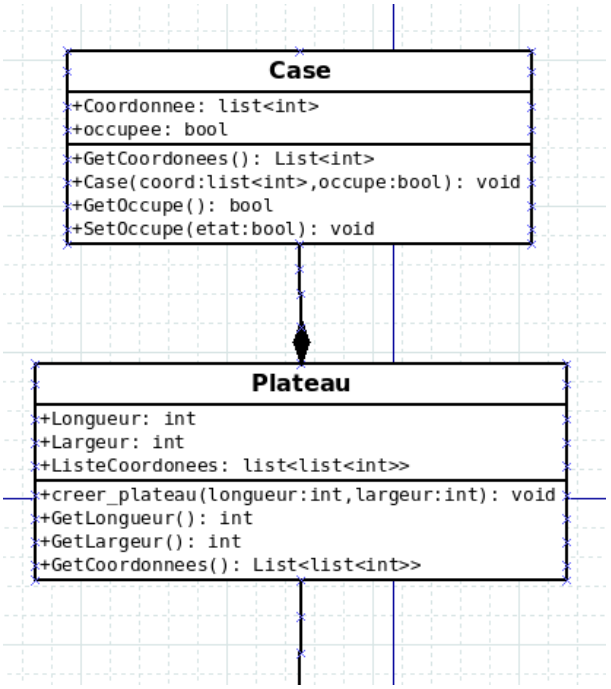


Cette classe permet de suivre l'état du jeu. Il est composé d'un attribut GameStatus qui renvoie l'état du jeu, d'un objet Plateau. On ajoute également le nombre de tours de type entier. On inclut la liste de tous les insectes dans le jeu avec ListAllInsect de type liste, cette liste nous permettra par la suite de récupérer la position des insectes sur le plateau.
C'est dans cette classe qu'on met à jour l'état du jeu, et qu'on vérifie si une des abeilles est encerclée ou non.

- Classe **Plateau :**

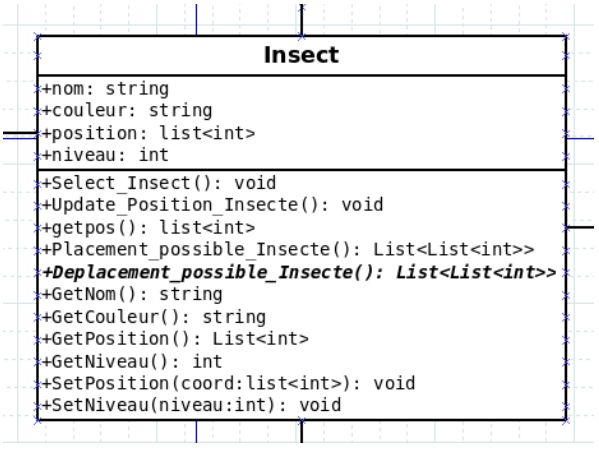
Dans cette classe on pourra générer un tableau avec les dimensions souhaitées. L'attribut ListeCoordonnees sera donc rempli à l'aide du constructeur. Chaque espace du tableau

aura un objet Case associé. Chaque objet Case aura une coordonnée et un attribut “occupe”.



- Classe **Insect :**

C’est une classe ayant une méthode abstraite, chaque insecte aura un nom, une couleur, une position et un niveau (étage pour le scarabée qui peut rester sur une pièce).
Les classes qui héritent de cette classe devront overrider la méthode permettant de déplacer le pion selon l'insecte.



- Classe **Joueur :**

C’est dans cette classe qu’on la couleur des pions du joueur et les différentes listes des pions. C’est à partir de ces listes que l’on va déplacer/placer les pions.

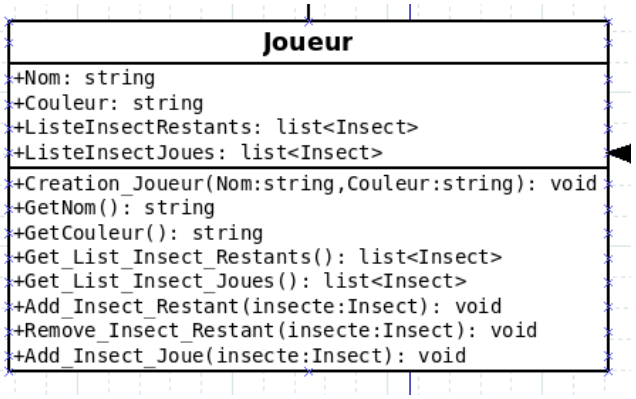
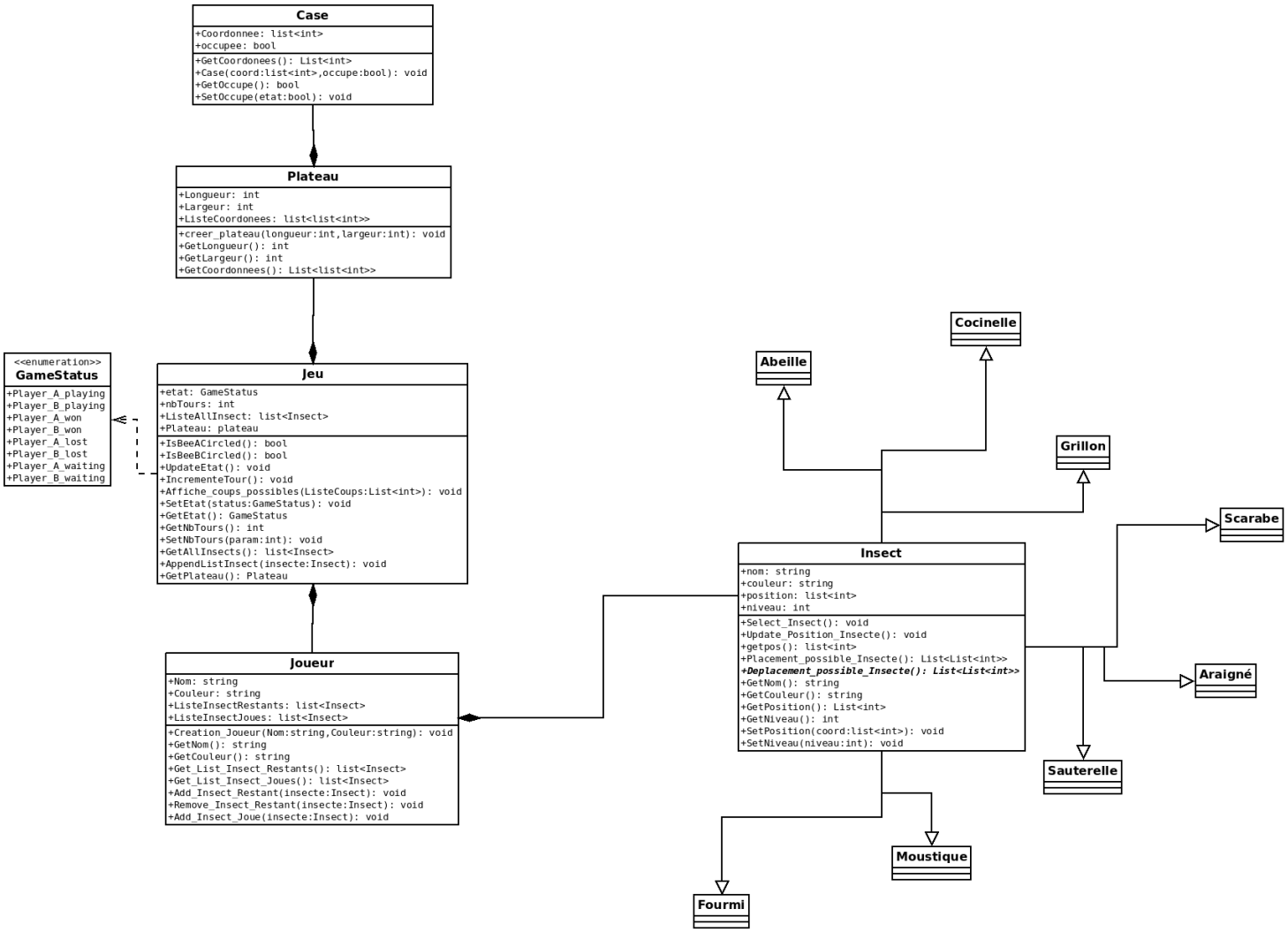


Diagramme des classes d'état :

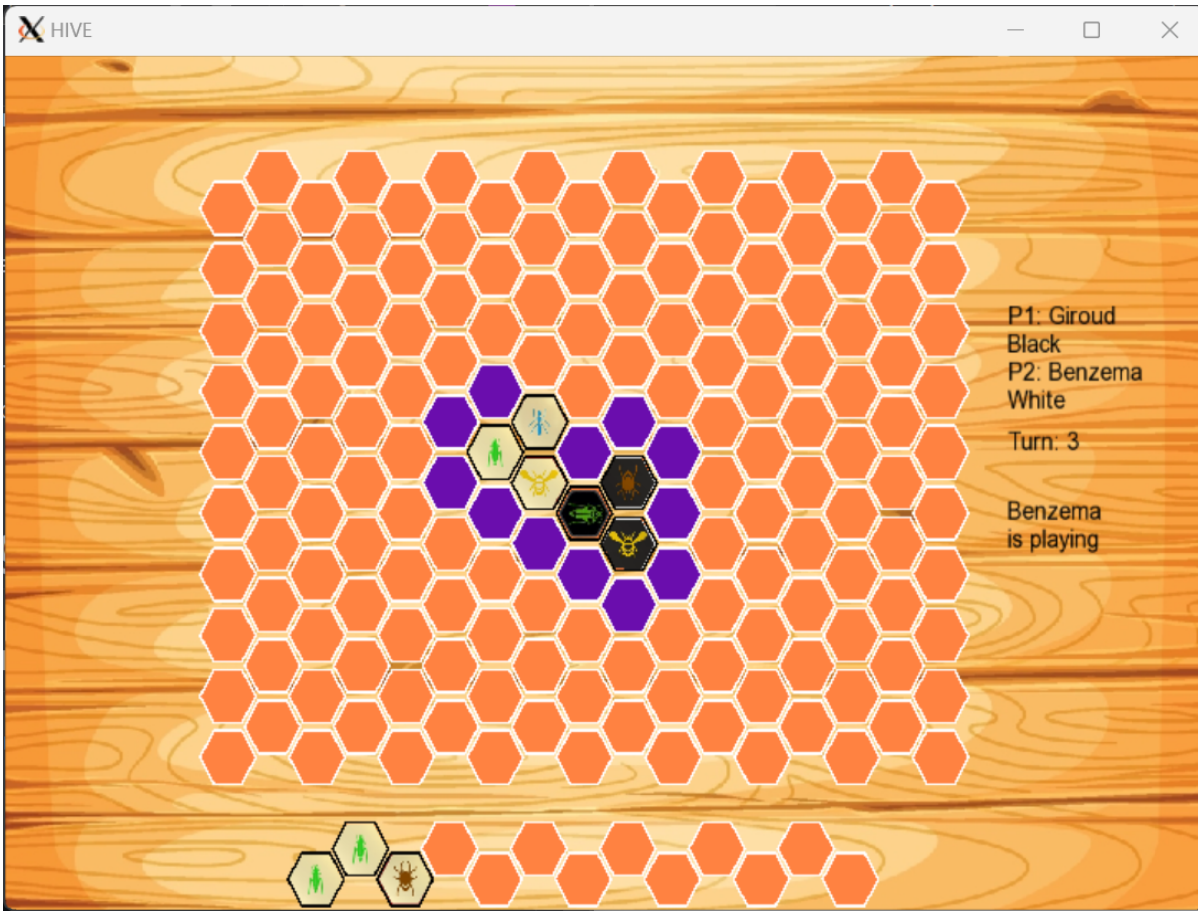


3 Rendu : Stratégie et Conception

3.1 Stratégie de rendu d'un état

Pour la réalisation du rendu d'un état, nous avons décidé de se focaliser sur l'affichage du jeu une fois lancé jusqu'à la fin d'une partie (pas de l'affichage d'accueil).

Voici à quoi ressemble la scène du jeu :

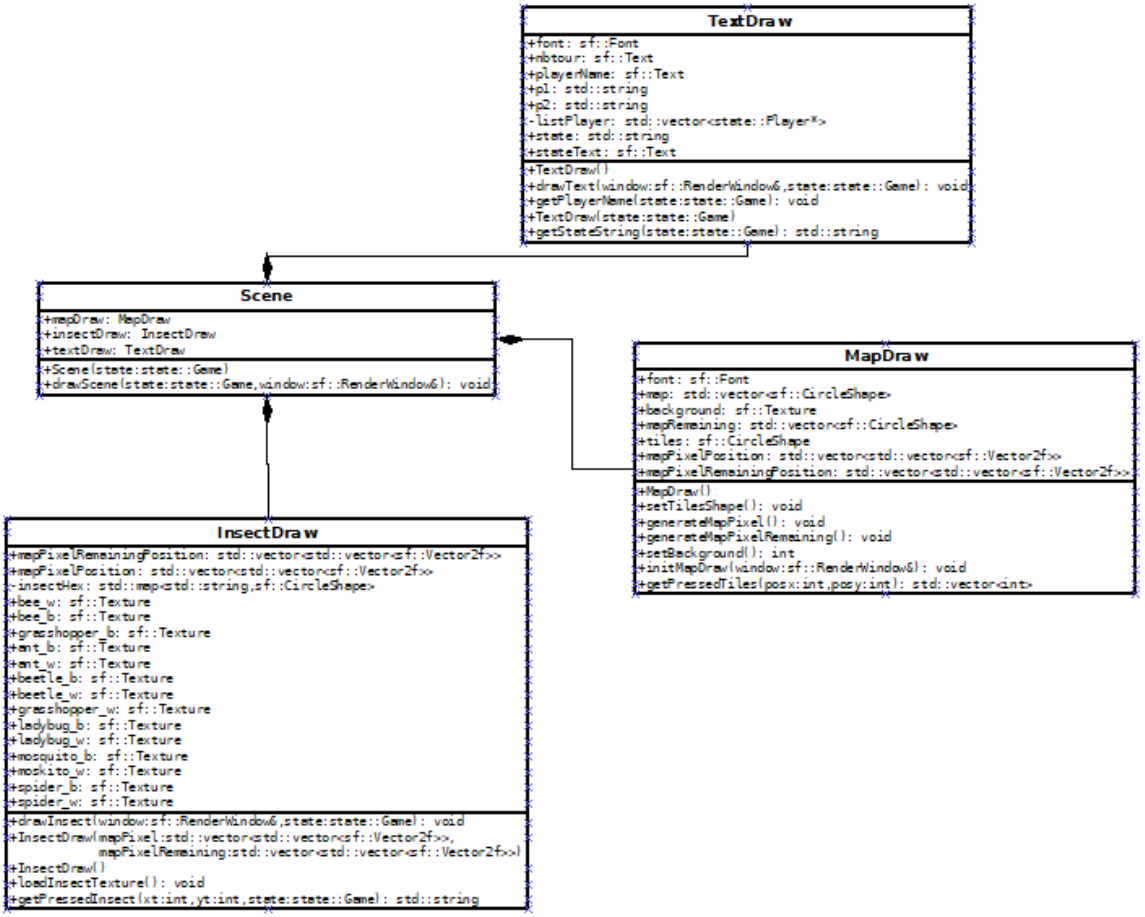


Elle est contient 4 éléments :

- La map du jeu ainsi que l'espace pour afficher les pions du joueurs
- Les pions
- Les informations nécessaires : noms des joueurs, le nombre de tour et l'état du jeu
- L'affichage des placements possibles

3.2 Conception logicielle

Nous avons donc pour la suite abouti à ce diagramme des classes :

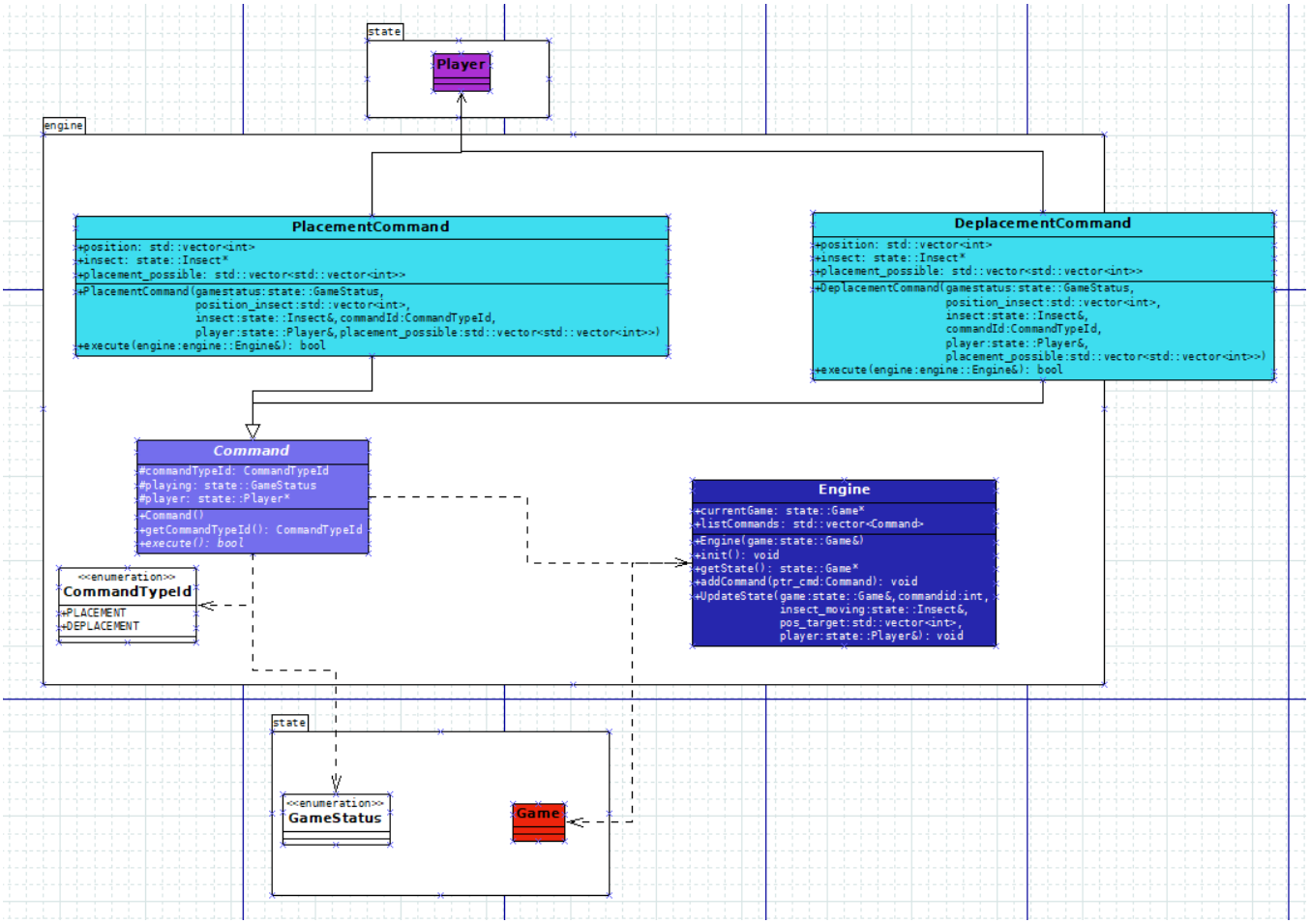


L'objectif est de avec seulement la classe state/game de pouvoir faire un rendu complet du jeu. Nous avons donc 4 classes :

- MapDraw, permet de dessiner la map ainsi que de stocker la position des différentes cases.
- TextDraw permet d'afficher du textes concernant les données du jeu
- InsectDraw permet d'afficher les différents pions, ceux du joueurs et ceux placé, cette classe nécessite d'avoir les positions stockées par MapDraw
- Scene est composé des classes précédentes permettant ainsi d'afficher ce qui est nécessaire selon l'état du jeu. Par exemple, afficher l'écran d'accueil, ensuite l'écran du jeu et un écran indiquant l'état final.

Comme nous pouvons voir, nous avons opté pour une map limitée dans notre jeu. Nous avons aussi décidé d'enlever certains pions pour arriver rapidement à un jeu jouable (début jusqu'à la fin).

4 Moteur de jeu



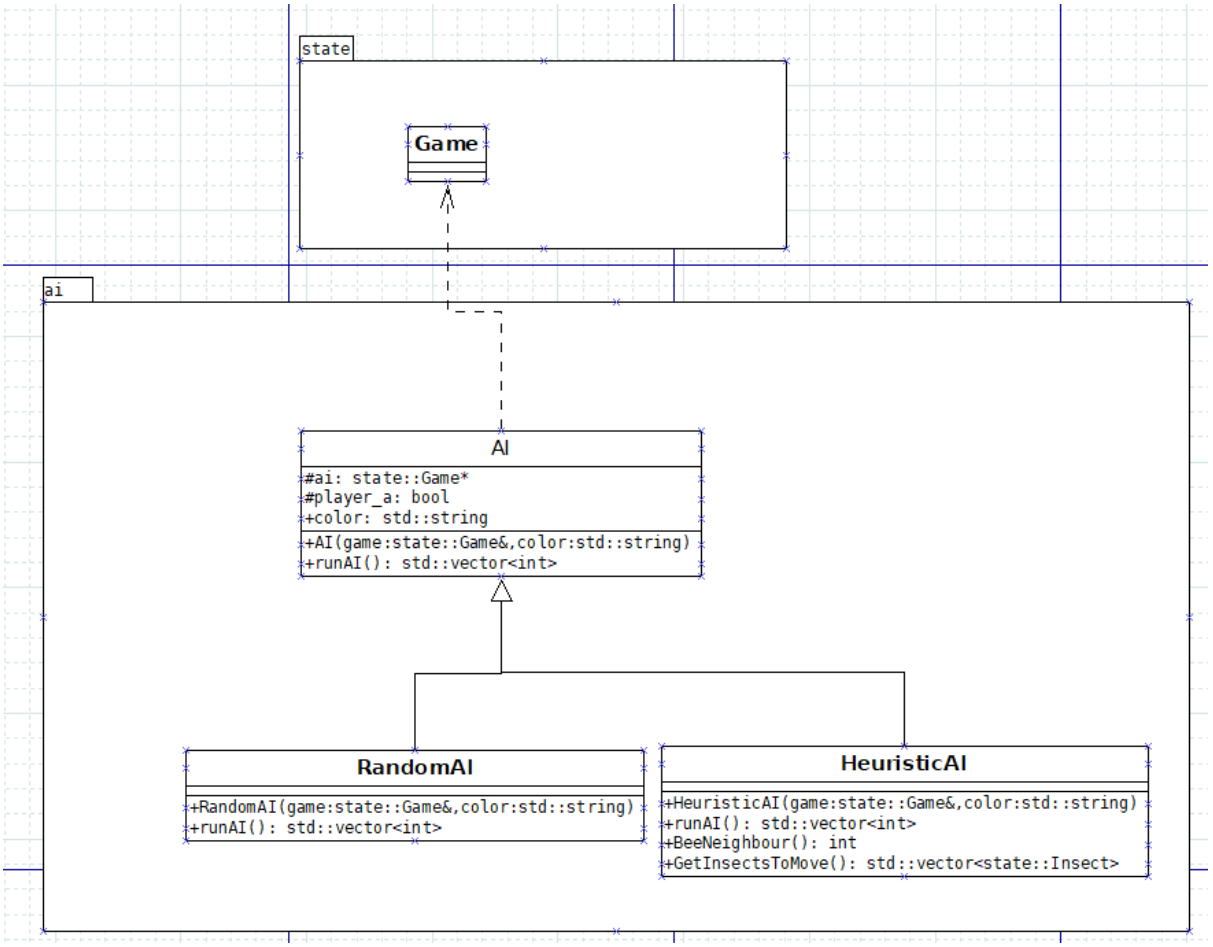
L'engine permet de vérifier les règles et faire les changements d'états du jeu. Nous avons donc 2 commandes permettant de lancer un changement d'état :

- La commande placement
- La commande déplacement

Lors de l'exécution de ses commandes, on vérifie donc certaines règles comme par exemple :

- tour du joueur
- placement interdit
- déplacement interdit
- action spéciale à réaliser selon les règles.

5 Intelligence Artificielle



Nous avons donc 2 IA, une aléatoire et une heuristique.
Concernant l'IA heuristique, comme nous pouvons en avoir plusieurs, nous avons choisi une IA qui choisit de se placer dès qu'il y a une possibilité de se coller à la reine.

RunAI(), est une méthode qui renvoie des coordonnées selon l'état du jeu.

6 Réalisation d’un serveur

6.1 Description

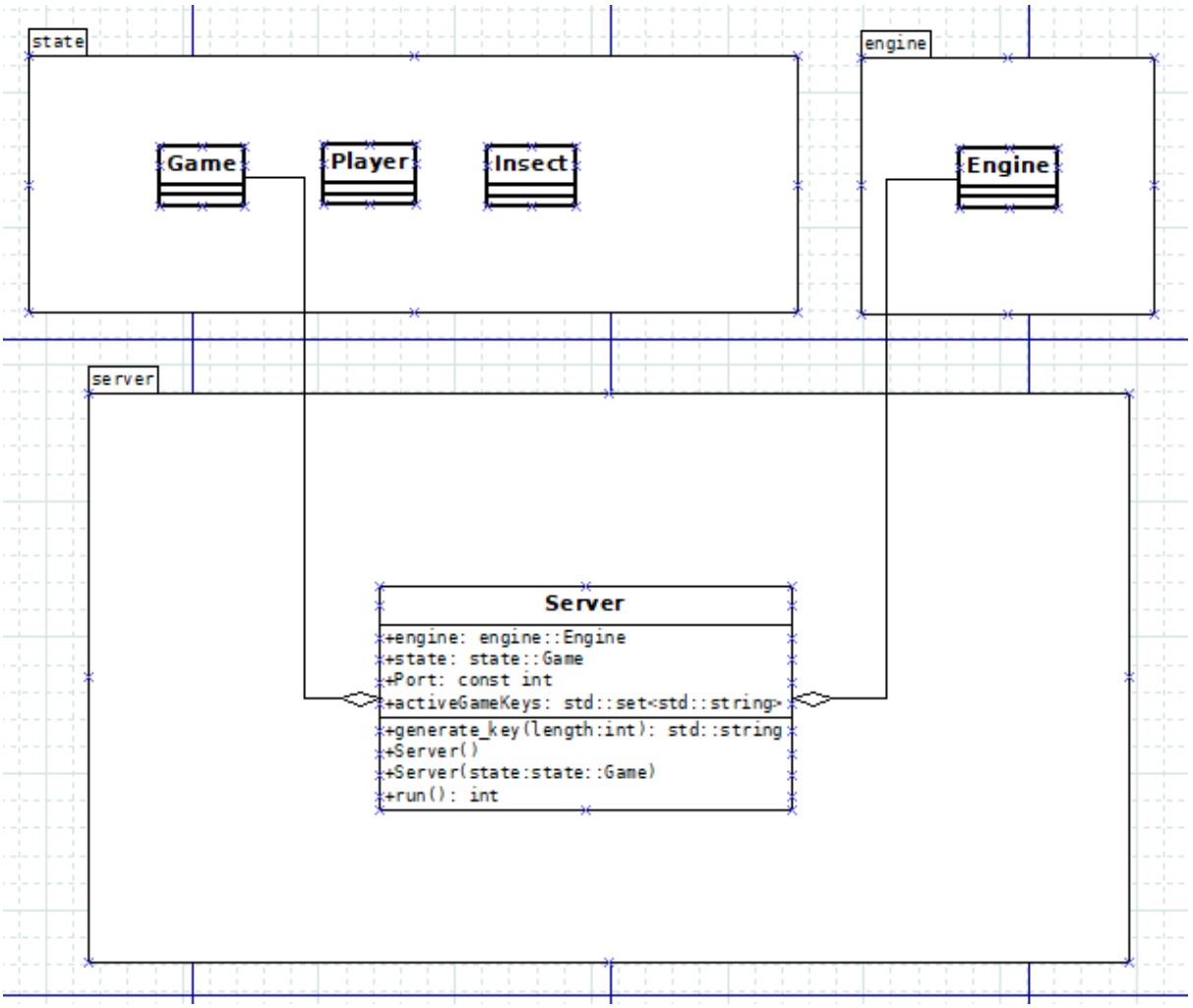
On souhaite développer une API Web REST à l’aide de la librairie microhttpd. Nous avons décidé de réaliser les routes suivantes :

GET
- /state pour récupérer l’état actuel
- /create pour créer une nouvelle partie et renvoyer une clé d’accès à cette partie

POST
- /join pour rejoindre la partie avec une clé
- /placement/<joueur> pour envoyer une commande de placement
- /deplacement/<joueur> pour envoyer une commande de déplacement

6.2 Conception logiciel

Voici le début du développement d'un diagramme de classe pour le serveur :



Nous utilisons les commandes curl et/ou postman pour tester notre serveur web REST. Dans un premier temps, nous avons implémenté la structure du serveur, permettant de créer une partie (un state), récupérer une clé permettant de jouer sur cette partie à 2 joueurs et récupérer l'état d'un jeu selon la clé.