MATH EN INFO

TRAVAUX PRATIQUES N° 2

Gestion de la carte

1 Objectifs

Lors de la précédente séance, nous avons pu mettre en place une discussion entre notre future IA et le serveur de jeu. Cependant, notre première IA n'ayant pas accès à la carte, il ne nous était pas possible de diriger notre personnage correctement. Durant cette séance, nous commencerons par structurer raisonnablement le code de notre IA pour que nous puissions l'améliorer par la suite puis nous apprendrons à notre IA à récupérer et à modéliser la carte du jeu.

2 Organisation du code

Lors de la séance précédente, nous avions construit notre première IA à partir du projet test avec lequel nous avions appris à communiquer avec le serveur. Cette structure de ce projet est trop simpliste pour être facilement améliorable. Nous allons donc repartir d'une structure plus propre.

1. **A FAIRE : Récupérez le projet "IA - Base" présent sur le commun, lancez le sur Netbeans 8.x et explorez-en le code source afin de comprendre comment l'IA va être structurée et le rôle de chacune des classes et méthodes de ce projet.**
2. **A FAIRE : En inspirant ce que vous aviez fait à la précédente séance, modifiez le projet pour que l'IA se comporte de la même façon que celle de la séance précédente.**

3 Gestion de la carte

Nous allons maintenant apprendre à notre IA à récupérer et à gérer la carte du jeu. Pour que notre IA soit en mesure de gérer cette carte, notre IA doit disposer des classes adéquates pour la stocker.

3.1 Création des bases de la structure

Afin de regrouper les différentes classes dont nous allons avoir besoin pour modéliser la carte, nous allons commencer par créer différents packages.

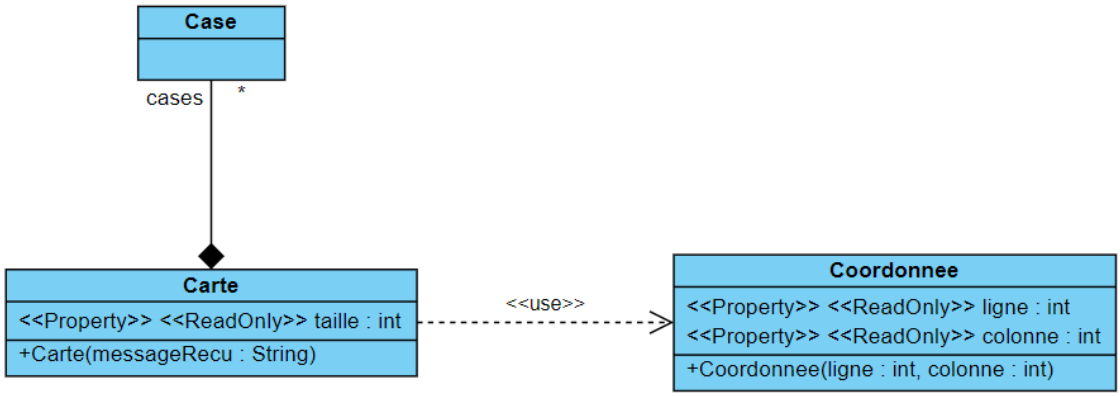
3. **A FAIRE : Créez un nouveau package "metier" dans le package "base". Créez un autre package : "carte" dans le package "metier". Et enfin, à l'intérieur du package "carte" créez deux autres packages nommés respectivement : "cases" et "objets".**

Le package "metier" servira à stocker toutes les classes représentant des éléments du jeu (cases, objets...) que notre IA aura besoin de connaître. Le package "carte" servira à stocker toutes les classes utiles pour la gestion de la carte, le package "cases" servira à stocker les différentes classes qui représenteront les différents types de cases et enfin le package "objets" servira à stocker celles qui représenteront les différentes objets.

4. **A FAIRE : Créez une classe "Carte" et une classe "Coordonnee" dans le package "carte". Créez une classe "Case" dans le package "cases" et une classe "Objet" dans le package "objets".**

3.2 Classes "Carte" et "Coordonnee"

Nous allons commencer par implémenter les classe Carte et "Coordonnee" dont voici l'UML :



La structure de données utilisée ici pour stocker les cases de la carte sera une HashMap<Coordonnee,Case>.

5. **A FAIRE : Implémentez les classes "Carte" et "Coordonnée". Dans le constructeur de la classe "Carte", on ne fera pour le moment qu'initialiser la HashMap "cases" et la taille à 50. N'oubliez pas d'implémenter les getters de ligne et colonne dans la classe "Coordonnee".**

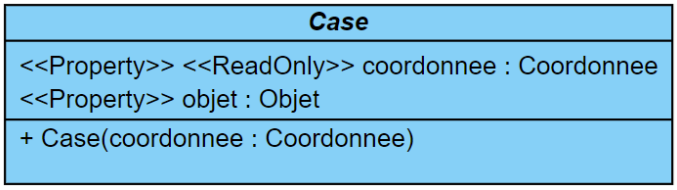
Comme expliqué en cours, notre classe "Coordonnee" étant utilisée comme clé d'une HashMap, nous avons besoin de rajouter deux méthodes à celle-ci : equals et hashCode. Netbeans peut générer de lui-même ces deux fonctions, pour cela, placez-vous dans la classe "Coordonnee" à l'endroit où vous souhaitez ajouter le code de ces deux méthodes puis appuyez sur alt+inser et sélectionnez "equals() et hashCode()". Dans la fenêtre qui apparaît alors, pensez à bien sélectionner tous les attributs dans les colonnes.

6. **A FAIRE : Faites générer à Netbeans les méthodes equals et hashCode pour la classe "Coordonnee".**

3.3 Gestion des cases

Notre carte pouvant avoir différents types de cases, nous allons tout naturellement faire de la classe "Case" une classe abstraite et utiliser une structure d'héritage.

7. **A FAIRE : Implémentez la classe "Case" en respectant l'UML suivant :**



N'oubliez pas d'implémenter les différents getters et setters.

8. **A FAIRE : Implémentez 3 classes héritant de la classe abstraite "Case" que nous appellerons : "CaseHerbe", "CaseTerre" et "CaseEau".**

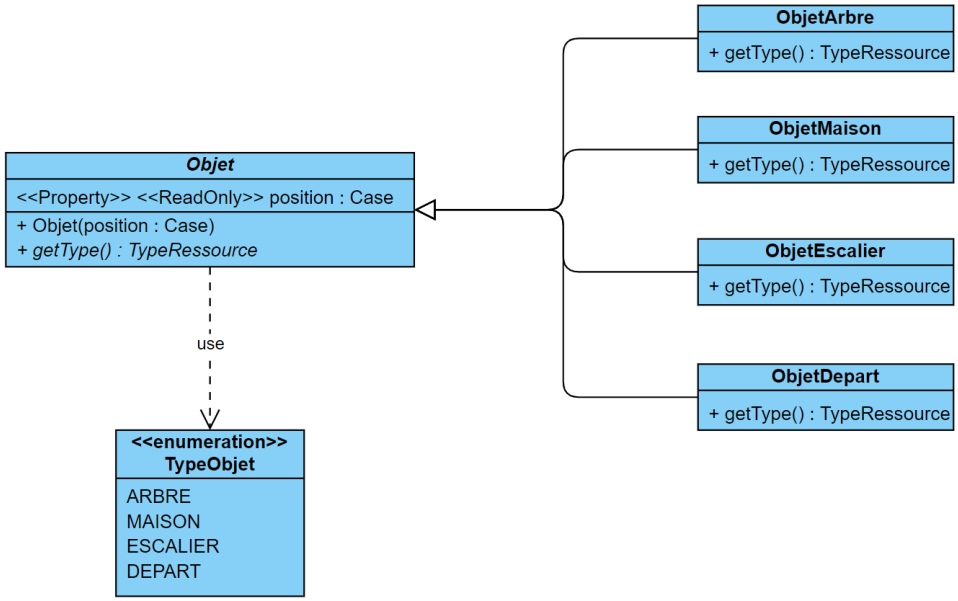
Notez qu'actuellement, lorsque l'on dispose d'une case, il n'est pas possible de savoir de quel type est cette case. Pour y remédier nous allons procéder en deux temps :

9. **A FAIRE : Créez dans le package "cases" une énumération appelée "TypeCase" et dont les valeurs sont : HERBE, TERRE et EAU.**
10. **A FAIRE : Ajoutez à la classe "Case" une méthode abstraite "getType() : TypeCase" et implémentez la dans les différentes classes filles.**

3.4 Gestion des objets

Pour la gestion des objets, nous allons procéder de la même manière que pour les cases.

11. **Implémentez les différentes classes et méthodes nécessaires en respectant l'UML suivant :**



3.5 Création des cases et des ressources

D'après le protocole réseau, le serveur nous décrira la carte sous forme d'une longue chaine de caractères où chaque caractère représente une case, la valeur du caractère indiquant le type de la case et la présence éventuelle d'un objet. Nous souhaitons donc être capable de fabriquer le bon type de case et éventuellement le bon type d'objet en fonction du caractère en question. Pour cela, la bonne solution est d'utiliser le design pattern de la Fabrique (que vous avez rencontré ou que vous allez rencontrer au TP8 du module de POO).

Succinctement, un design pattern est juste un terme désignant une "solution type" pour un "problème classique" de programmation. En d'autres termes, le problème que nous rencontrons est si classique qu'il existe une méthode type résolvant efficacement ce problème.

Commençons par nous occuper des objets.

12. **A FAIRE : Créez dans le package "objet" une classe (vide) FabriqueObjet.**
13. **A FAIRE : Ajoutez à cette classe une méthode statique dont l'entête sera :**

    public static Objet creer(Case position, Character lettre)

14. **A Faire : En utilisant un bloc "switch", implémentez cette méthode pour que, en fonction de la lettre donnée en entrée, elle construise et renvoie un objet du bon type (la méthode renverra null si la lettre donnée en entrée correspond à une case ne contenant pas d'objet).**

Passons maintenant à la fabrication des cases.

15. **A FAIRE : Créez dans le package "Case" une classe (vide) FabriqueCase.**
16. **A FAIRE : Ajoutez à cette classe une méthode statique dont l'entête sera :**

    public static Case creer(Coordonnee coordonnee, Character lettre)

17. **A FAIRE : Implémentez cette méthode en commençant par utiliser un bloc "switch" pour construire le bon type de case (en fonction de la lettre reçue en entrée) puis utiliser la fabrique des objets pour ajouter le bon objet à la case. On considèrera que la maison et le point de départ sont des cases d'herbes.**
18. **A FAIRE : Faites vérifier vos deux fabriques par votre enseignant.**

Maintenant que nous disposons de ces fabriques, nous allons pouvoir modifier la classe "Carte" pour pouvoir générer la carte à partir du message reçu.

19. **A FAIRE : Ajoutez à la classe Carte, une méthode privée dont l'entête sera :**

    private void ajouterCase(Coordonnee coordonnee, Character lettre)

    et qui utilisera la fabrique de cases pour construire une nouvelle case et l'insérera dans la HashMap.
20. **A FAIRE : Modifiez le constructeur de Carte de la façon suivante :**

```
public Carte(String messageRecu) {
    this.cases = new HashMap<>();
    this.taille = (int) Math.sqrt(messageRecu.length());
    for(int i=0;i<this.taille;i++) {
        for(int j=0;j<this.taille;j++) {
            this.ajouterCase(new Coordonnee(i,j), messageRecu.charAt
                (j+this.taille*i));
        }
    }
}
```

Attention, ne faites pas de copier-coller des codes fournis dans les sujets au risque d'insérer des caractères "non valides" dans votre code (lié à l'encodage des caractères dans le format pdf).
Vous pouvez reconnaitre une formule vue en TD permettant de trouver les coordonnées d'une case d'une grille en fonction de sa numérotation absolue.

3.6 Demander la carte au serveur

Nous savons maintenant modéliser la carte du jeu en fonction de la réponse du serveur. Il nous faut cependant encore apprendre à notre IA à demander cette information et à lui apprendre à lancer la création de la carte après avoir reçu la réponse du serveur.

21. **A FAIRE : Ajoutez et implémentez dans le ModuleMemoire :**
    - **un attribut "carte" de type Carte,**
    - **une méthode "void genererCarte(String messageRecu)" qui génère la carte à partir du messageRecu.**
    - **une méthode "boolean hasCarte()" qui renvoie si la carte a été créée ou non (en testant si this.carte != null).**
22. **A FAIRE : Modifiez la méthode determinerNouvelleAction du ModuleDecision de telle sorte que :**
    - **si le module mémoire (à demander à l'IA) ne dispose pas de carte la méthode renvoie "MAP",**
    - **si le module mémoire dispose déjà d'une carte, la méthode renvoie "END" et demande l'arrêt de la discussion à l'IA.**
23. **A FAIRE : Modifiez le ModuleReaction de la façon suivante :**

```
/**
 * Méthode principale de réaction à un message reçu
 * @param messageEnvoye dernier message envoyé par l'IA
 * @param messageRecu dermier message reçu par l'IA
 */
public void reagirAuMessageRecu(String messageEnvoye, String
    messageRecu) {
    switch(messageEnvoye) {
        case "MAP" : reactionCarte(messageRecu); break;
    }
}

/**
 * Réaction à la réception de la carte
 * @param messageRecu la chaine de caractère représentant la
     carte
 */
private void reactionCarte(String messageRecu) {
    this.getIA().getModuleMemoire().genererCarte(messageRecu);
}
```

3.7 Tester tout ça

Tout est maintenant en place pour que notre IA demande la carte au serveur et la modélise. Afin de s'assurer que tout ce passe bien, nous allons ajouter une méthode d'affichage pour pouvoir tester la génération de la carte.

24. **A FAIRE : Ajoutez la méthode suivante à la classe "Carte" :**

```
public void afficheConsole() {
        for(int i=0;i<this.taille;i++) {
            for(int j=0;j<this.taille;j++) {
                String affichage = "E";
                Case caseEnCours = this.cases.get(new Coordonnee(i,j));
                if(caseEnCours.getType() == HERBE) {
                    if(caseEnCours.getObjet() == null) {
                        affichage = "H";
                    }
                    else {
                        switch(caseEnCours.getObjet().getType()) {
                            case ARBRE : affichage = "A"; break;
                            case MAISON : affichage = "M"; break;
                            case ESCALIER : affichage = "S"; break;
                            case DEPART : affichage = "D"; break;

                        }
                    }
                }
                else if(caseEnCours.getType() == TERRE) {
                    affichage = "T";
                }
                System.out.print(affichage);
            }
            System.out.println("");
        }
    }
}
```

25. **A FAIRE : Faites afficher la carte dans la console après sa création et vérifiez avec l'enseignant le résultat obtenu.**

Vous constaterez que pour l'instant, l'escalier n'apparaît pas. C'est normal, le serveur ne nous indique pas explicitement sa position. Cependant, le manuel nous indique où les escaliers sont placés par rapport à la case de départ.

26. **BONUS : Modifiez la fabrique des objets et le constructeur de carte pour qu'après avoir généré la carte à parti du message du serveur, on rajoute les escaliers au bon endroit.**