MATH EN INFO

TRAVAUX PRATIQUES N° 5

Actions

1 Objectifs

Durant le dernier TP, nous avons mis en place notre premier algorithme de calcul de distance. Il est temps d'apprendre à notre IA à l'utiliser correctement. Cependant, nous avons vu que dans la majorité des cas, notre algorithme ne va pas fournir qu'une seule action à notre algorithme mais plutôt une succession d'action. Nous allons donc réaliser une petite série de modifications pour faciliter la gestion des actions par notre IA. Une version "rapide" du serveur est disponible sur le commun. Attention, cette version peut provoquer des erreurs d'affichage dans la console de Netbeans.

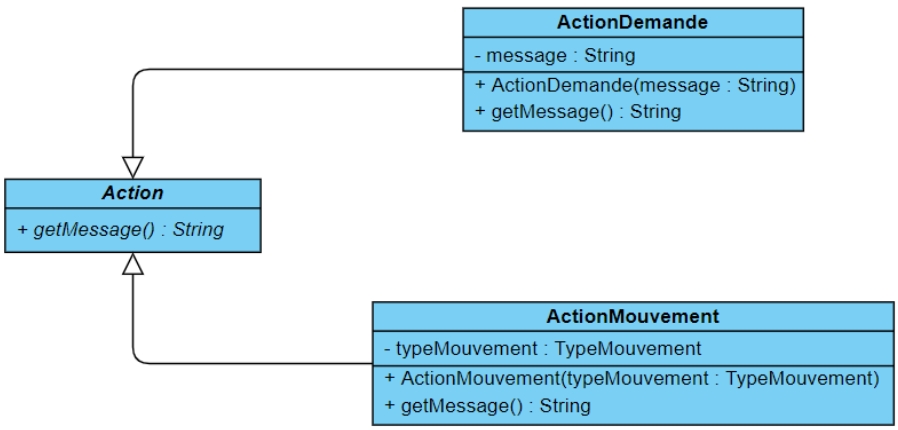
2 Gestion des actions

Nous allons mettre en place une structure permettant la gestion des différents types d'actions réalisées par notre IA. Actuellement, nous avons rencontré deux types d'actions : les demandes (MAP...) et les mouvements du personnage.

1. **A FAIRE : Créer un nouveau package "actions" dans le package "metier".**
2. **A FAIRE : Dans ce nouveau package, créer une classe abstraite "Action" disposant uniquement de la méthode suivante :**

   public abstract String getMessage() ;

3. **A FAIRE : Dans le package "actions", ajouter les classes "ActionDemande" et "ActionMouvement" en respectant l'UML suivant :**



- **La méthode "getMessage" de la classe "ActionDemande" renverra juste l'attribut "message".**
- **La méthode "getMessage" de la classe "ActionMouvement" renverra "MOVE|"+typeMouvement.**

Comme pour les cases et les objets, nous allons utiliser le design pattern de la fabrique pour gérer la création des différents types d'actions.

4. **A FAIRE : Créer une nouvelle énumération : "TypeDemande" dans le package "actions" composée d'un seul élément "CARTE" (d'autres viendront se rajouter plus tard).**
5. **A FAIRE : Créer une classe "FabriqueAction" dans le package "actions".**
6. **A FAIRE : Implémenter dans cette nouvelle classe la méthode suivante :**

   public static Action creerMouvement(TypeMouvement mouvement)

7. **A FAIRE : Implémenter dans la nouvelle classe la méthode suivante :**

   public static Action creerDemande(TypeDemande demande)

   Pour cette méthode, utilisez un switch malgré le fait qu'il n'y ait qu'une seule demande possible pour le moment afin de préparer le terrain pour les autres demandes qui apparaitront plus tard.

3 Utilisation des actions

Maintenant que nous disposons d'une notion d'action, nous allons pouvoir améliorer notre module de décision pour qu'il puisse gérer une liste d'actions.

8. **A FAIRE : Ajouter l'attribut suivant au module de décision en pensant bien à l'initialiser dans le constructeur :**

   private ArrayList<Action> listeDesActionsARealiser ;

9. **A FAIRE : Ajouter au module de décision, la méthode suivante (que l'on laissera vide pour le moment) :**

   private void determinerNouvellesActions()

10. **A FAIRE : Modifier le code de la méthode "determinerNouvelleAction" pour qu'il suive le pseudo-code suivant (attention, il s'agit bien d'une succession de if, et non de else if) :**

```
"messageReponse" <- "END"

//Gestion de la carte et du joueur
Si le module mémoire ne dispose pas de carte
        Faire créer par la fabrique une nouvelle actionDemande(CARTE)
        Ajouter cette action à la liste des actions à réaliser

//Détermine de nouvelles actions si besoin
Si la liste des actions à réaliser est vide
        appeler la méthode determinerNouvellesActions

//Réalisation de la première action de la liste
Si la liste des actions à réaliser est non vide
```

```
        messageReponse <— le message de l action en position 0 de la
            liste
        On enlève de la liste l action en position 0
Sinon
    messageReponse <— "SLEEP"

Renvoyer messageReponse ;
```

11. **A FAIRE : Testez votre IA. Celle-ci doit commencer par demander la carte au serveur, puis la position du joueur et enfin dormir sans fin.**

4 Une IA en vadrouille

A ce stade, notre IA est donc capable... de faire ce qu'elle était déjà capable de faire avant... mais la nouvelle structure que nous avons mise en place, va nous permettre maintenant de gérer facilement les mouvements. Nous allons modifier notre IA pour lui apprendre à choisir une destination aléatoirement et à s'y rendre (si possible). Pour permettre cela, notre module de décision devra pouvoir avoir accès à certaines informations auxquelles il n'a actuellement pas accès. Nous allons commencer par corriger ça.

12. **A FAIRE : Ajouter un getter pour l'attribut "carte" dans le module de mémoire.**
13. **A FAIRE : Ajouter dans le module mémoire, une méthode "getCaseJoueur" qui renvoie la case où se trouve actuellement le joueur.**
14. **A FAIRE : Ajouter un getter pour l'attribut "taille" dans la classe "Carte".**

Maintenant, nous allons pouvoir modifier notre IA :

15. **A FAIRE : Ajouter au module de décision, la méthode suivante (que l'on laissera vide pour le moment) :**

    private void seDeplacerEn(Coordonnee coordonnee)

16. **A FAIRE : Modifier le code de la méthode "determinerNouvellesActions" pour qu'il appelle la méthode "seDeplacerEn" en lui donnant un coordonnée aléatoire.**
17. **A FAIRE : Modifier le code de la méthode "seDeplacerEn" pour qu'il suive le pseudo-code suivant :**

```
Afficher dans la console : "——— Je veux aller en "+coordonnee+"
    ———"

On récupère la carte en la demandant au module mémoire

On crée un nouvel algorithme de parcours en largeur
On récupère la case où se trouve le joueur
On lance le calcul des distances à partir de cette case

On détermine la case de destination à partir des coordonnées données
    en entrée
On demande le chemin vers la case destination à notre algorithme
Pour chaque mouvement de ce chemin
        On demande à la fabrique de créer une nouvelle actionMouvement
        On ajoute cette action à la liste des actions à réaliser
```

18. **Testez votre IA.**

Si tout semble bien se passer au début, votre IA se met rapidement à faire n'importe quoi. Le module de décision n'avertit en effet pas le module mémoire du déplacement du joueur. Pour notre IA, le joueur reste donc immobile et les chemins calculés sont faits systématiquement à partir de la case de départ du joueur... Modifions cela !

19. **A FAIRE : Créer une enumération TypeAction dans le package "actions" contenant les éléments suivants :**

    DEMANDE,MOUVEMENT ;

20. **A FAIRE : Ajouter à la classe "Action" la méthode suivante :**

    public abstract TypeAction getType() ;

    et l'implémenter dans les classes filles.

21. **A FAIRE : Ajouter à la classe "Action" la méthode suivante :**

    public abstract TypeMouvement getDirection() ;

    et l'implémenter dans les classes filles (la méthode renverra null dans le cas d'une actionDemande).

22. **A FAIRE : Ajouter au module mémoire, la méthode suivante :**

    public void effectuerAction(Action action)

    et l'implémenter pour que si l'action donnée en paramètre est une action de type MOUVEMENT, le module de mémoire modifie correctement la position du joueur.

23. **A FAIRE : Modifier la méthode "determinerNouvelleAction" du module de décision pour qu'il fasse effectuer au module mémoire la première action de la liste juste avant de la supprimer de la liste.**

24. **A FAIRE : Vérifiez que votre IA n'a plus de problème de déplacement.**

Vous avez du remarquer qu'un dernier problème demeure, Abigail ne pouvant dormir que dans sa maison, nous devons la renvoyer chez elle avant de dormir. Cette successions d'actions (rentrer et dormir) arrivera fréquemment par la suite, nous allons donc implémenter une méthode spécifique pour cela.

25. **A FAIRE : Ajouter l'élément ACTIONSTATIQUE dans l'énumération TypeAction (ce nouveau type d'action représentera toutes les actions ne nécessitant pas de mouvement du joueur : dormir, planter...).**

26. **A FAIRE : Créer, dans le package "actions", une classe "ActionDormir" héritant de la classe "Action" et implémentez la (son type sera ACTIONSTATIQUE). Créer dans le module de décision une méthode.**

27. **A FAIRE : Créer, dans le package "actions", une nouvelle énumération TypeActionStatique contenant un unique élément : DORMIR ;**

28. **A FAIRE : Ajouter et implémenter dans la fabrique des actions la nouvelle méthode suivante :**

    public static Action creerActionStatique(TypeActionStatique type)

29. **A FAIRE : Créer dans le module de décision une méthode.**

    private void allerDormir()

30. **A FAIRE : Implémenter cette méthode pour que qu'elle face rentrer Abigail chez elle puis dormir.**
31. **A FAIRE : Modifier la méthode "determinerNouvelleAction" en remplaçant le messageReponse <- "SLEEP" par :**

```
allerDormir();
messageReponse <— le message de l action en position 0 de la
    liste
On fait réaliser l action par le module mémoire
On enlève de la liste l action en position 0
```

32. **A FAIRE : Tester l'IA.**