Projet Design Pattern - Pacman

Olivier Goudet

October 4, 2018

1 Description du sujet

Un jeu de plateau tour par tour est composé d'un monde de taille limitée sur lequel évolue un certain nombre d'agents. Le déroulé général du jeu est le suivant :

- Au tour $t = 0$ le plateau jeu est initialisé suivant une configuration définie par l'utilisateur. Les agents sont créés et placés sur le plateau.
- A chaque tour t, chaque agent peut réaliser une action prédéfinie par un ensemble de règles et qui a un impact sur l'environnement. L'ensemble de ces actions vont conduire à un nouvelle état du monde au temps $t + 1$.
- Une fois que le nombre maximum de tour est atteint ou bien qu'une condition spécifique de fin de jeu est atteinte, le jeu s'arrête.

Nous allons créer un jeu de plateau de type Pacman qui comprend un ou plusieurs agents *pacman* ainsi que plusieurs agents *fantôme*. Voir la fiche Wikipedia pour plus d'information sur le jeu https://fr.wikipedia.org/wiki/Pac-Man.

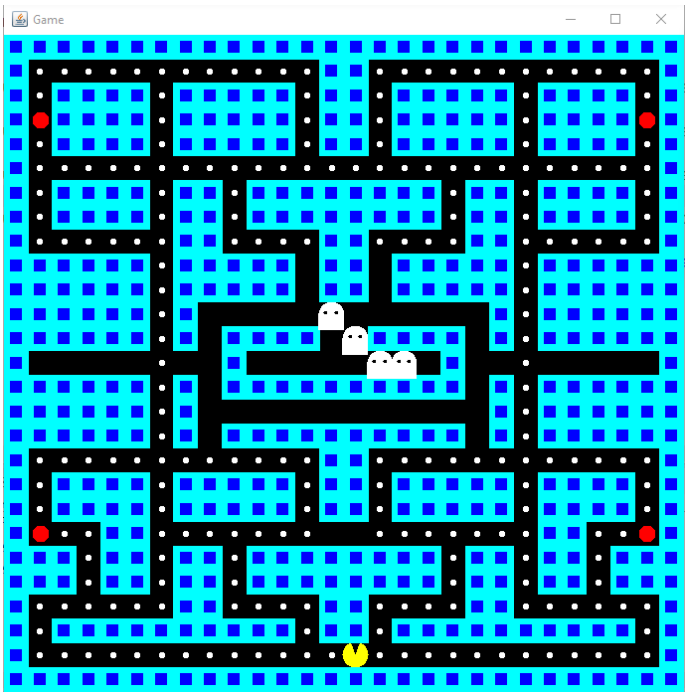

Figure 1: Plateau du jeu de pacman

1.1 Règles du jeu

Les règles du jeu de Pacman sont les suivantes :

- L'environnement est constitué d'un labyrinthe avec des cases libres et des cases murs. Sur chaque case libre peut se trouver des pac-gommes que les agents pacmans doivent manger pour gagner la partie. Une vue du plateau est présentée sur la figure 1.
- Chaque agent pacman ou fantôme se déplace d'une case à chaque tour dans la direction de son choix mais ne peut pas aller sur une case mur.
- Si un agent pacman se trouve sur la même case qu'un agent fantôme il est mangé et disparaît du plateau.
- Des capsules spéciales sont disposées sur le plateau. Une fois mangée par un pacman, elles rendent les fantômes vulnérables pendant 20 périodes au cours desquelles les Pacmans peuvent les manger.
- Le jeu s'arrête si les agents fantômes mangent l'ensemble des pacmans (victoire des fantômes), ou bien si les pacmans mangent l'ensemble des pac-gommes (victoire des pacmans)

1.2 Objectifs du projet

L'objectif de ce projet est d'implémenter un jeu de Pacman interactif avec une interface visuelle en utilisant les Design Patterns vus en cours. Une fois la base du jeu réalisée, il s'agira d'implémenter les comportements et les stratégies des agents pour remporter la partie.

Les premières séances de TP seront très guidées de façon à ce que chacun puisse réaliser une base du jeu. Une fois cette base réalisée, plus de liberté sera accordée pour le développement de l'interface, des commandes et des stratégies des agents.

L'avancement lors des séances de TP sera pris en compte pour la notation finale.

Les séances de TP vont s'articuler dans les grandes lignes de la façon suivante :

Séance 1 : réalisation d'un patron de conception général d'un jeu de plateau. Implémentation d'un prototype de jeu très simple. Test de l'architecture. Début de l'implémentation d'une architecture Modèle-Vue-Contrôleur (MVC) pour gérer l'affichage graphique et les commandes de l'utilisateur.

Séance 2 : Fin de l'implémentation d'une architecture Modèle-Vue-Contrôleur (MVC) pour gérer l'affichage graphique et les commandes de l'utilisateur. Initialisation de l'environnement de jeu Pacman. Création des agents sur la plateforme. Affichage du jeu.

Séance 3 : Implémentation des règles du jeu de Pacman, des états des agents ainsi que des comportements.

Séance 4 : Ajout de stratégies pour les agents. Ajout d'un mode interactif.

Séance 5 : Implémentation de stratégies avancées : stratégie de recherche dans des arbres, algorithme A*, coopération multi-agent...

2 Création de l'architecture du jeu

Il s'agit tout d'abord de créer une architecture générale d'un jeu tour par tour à l'aide du pattern *Patron de méthode*. Ce patron de méthode sera ensuite implémentée de façon concrète par une classe *SimpleGame*.

1. Créer une classe abstraite *Game* avec les éléments suivants :
   - un compteur du nombre de tours
   - un nombre de tours maximum prédéfini au moment de sa création

- une méthode concrète *init* initialise le jeu en remettant le compteur du nombre de tours à zéro et en appelant la méthode abstraite *initializeGame*
- une méthode concrète *step* : effectue un seul tour du jeu en appelant la méthode abstraite *takeTurn* si le jeu n'est pas terminé. Si le jeu est terminé, elle doit faire appel à la méthode abstraite *gameOver()* qui permettra d'afficher la fin du jeu
- une méthode concrète *run* lance le jeu jusqu'à la fin tant que trois conditions ne sont pas remplies : (i) le jeu n'est pas terminé, (ii) le jeu n'est pas mis en pause (testé par un flag booléen *isRunning* et (iii) le nombre maximum de tour n'est pas atteint.
- une méthode concrète *stop* met en pause le jeu en désactivant le flag booléen *isRunning*

2. Implémentez cette classe abstraite avec une classe concrète *SimpleGame*. A chaque fois que quelque chose se passe dans le jeu (dans les méthodes concrètes de *SimpleGame*), faites une sortie console. Cela servira à vérifier que la structure générale du jeu marche bien.

3. Implémentez une classe *Test* avec une méthode *main* qui permet de lancer *SimpleGame* et vérifiez que tout fonctionne correctement

4. Comme vous pouvez le constater le déroulé du jeu est très rapide jusqu'à la fin. Vous allez maintenant changer un peu la classe *Game* de façon à ce que la vitesse de déroulement du jeu soit contrôlable :

   - Demander à la classe *Game* d'implémenter la classe *Runnable* et ajouter une méthode *launch()* qui permet de lancer la simulation dans un objet de type Thread (qui sera un attribut de la classe *Game*).
   - Ajouter un temps d'arrêt paramétrable après chaque tour de jeu : *Thread.sleep((long) time)*.
   - Tester à nouveau avec *SimpleGame* que tout fonctionne correctement

3 Création du Modèle-Vue-Contrôleur

Relisez le cours de la séance 2 pour vous rappeler le principe de l'architecture MVC. Vous pouvez vous inspirer aussi de l'exemple donné dans le livre Design Pattern - tête la première - page 564 à 572.

3.1 Modèle

La classe abstraite *Game* est le modèle. Elle doit implémenter une interface sujet (ou observable). A chaque fois que le jeu change d'état, il faut notifier les observateur. Pour cela, inspirez vous de l'exemple vu en cours. Il s'agit d'effectuer des notifications aux observateurs quand il se passe quelque chose dans le jeu.

3.2 Création de la vue

Il s'agit de réaliser deux classes :

- une classe *ViewGame* qui permet l'affichage du jeu
- une *ViewCommande* qui affichage les principales commandes pour l'utilisateur (initialisation du jeu, lancement, mise en pause, etc...), ainsi qu'un compteur de tour du jeu

Pour l'implémentation, chacune de ces classe peut être composée d'une JFrame. Ces deux JFrame peuvent être disposées à deux endroits différents de l'écran.

3.2.1 Création de l'affichage des commandes du jeu

Un exemple d'affichage des commandes pour l'utilisateur est proposée sur la figure 2.

Pour réaliser cette affichage de commande il faut utiliser les classe *JPanel* et *JButton*. On peut les placer avec un layout de type *GridLayout*. Le slider peut être créé avec la classe *JSlider*.

Pour ajouter des icônes on peut utiliser la commande suivante :

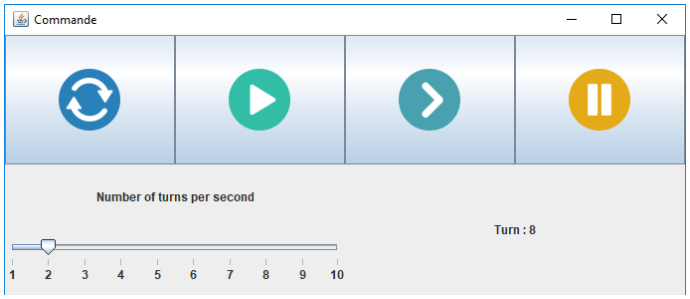

Figure 2: Interface de commande du jeu

```
Icon icon_restart = new ImageIcon("icon_restart.png");
choixInit = new JButton(icon_restart);
```

Les fichiers d'icône sont disponibles dans le dossier *Icones* de la section TP du projet sur Moodle.

Il s'agit ensuite d'ajouter des écouteurs d'actions sur les différents composant de l'interface. Voici le code par exemple pour réaliser une action quand l'utilisateur clique sur le JButton *choixInit*.

```
choixInit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evenement) {
                controleurGame.start();
        }
});
```

Lorsqu'une action est effectuée par l'utilisateur, on appellera une autre classe *controleurGame* qui sait comment gérer les actions des utilisateurs (cf. section 3.3 plus loin), c'est la stratégie pour la vue.

3.2.2 Création du panneau de jeu

Pour la partie affichage du jeu, créez simplement pour l'instant un panneau qui affiche sous format texte ce qui se passe dans le jeu (affichage du message qu'un nouveau tour de déroule, affichage d'un message quand le jeu est fini, ...).

3.3 Création du contrôleur pour l'interface graphique

La classe qui va faire le lien entre la vue et le jeu est une classe *ControleurGame*, qui est la stratégie pour la vue .

Ce contrôleur doit permettre de contrôler le jeu quand des actions ont été effectuées par l'utilisateur dans la Vue (par exemple lorsque l'utilisateur clique sur un bouton). Voir l'exemple du cours.

1. Définissez tout d'abord une interface de contrôleur.
2. **Contrôleur basique**. Implémenter un premier contrôleur basique qui exécute simplement les actions du jeu lorsqu'il est appelé. Testez l'architecture du jeu global maintenant avec ce contrôleur. Vérifier que tout fonctionne.
3. **Contrôleur plus avancé**. Implémenter un deuxième contrôleur plus évolué qui exécute les actions du jeu mais désactive aussi les boutons de la vue de façon à proposer des choix cohérents à chaque fois pour l'utilisateur. Par exemple, activez au début uniquement le bouton *init*, car c'est la première action à effectuer pour lancer le jeu. Ensuite quand l'utilisateur a cliqué sur ce bouton *init*, désactivez le (car il ne sert à rien de cliquer deux fois de suite dessus). Faites la même chose pour la gestion des autres boutons. Testez l'architecture du jeu global avec ce nouveau contrôleur. Vérifier que tout fonctionne comme attendu.

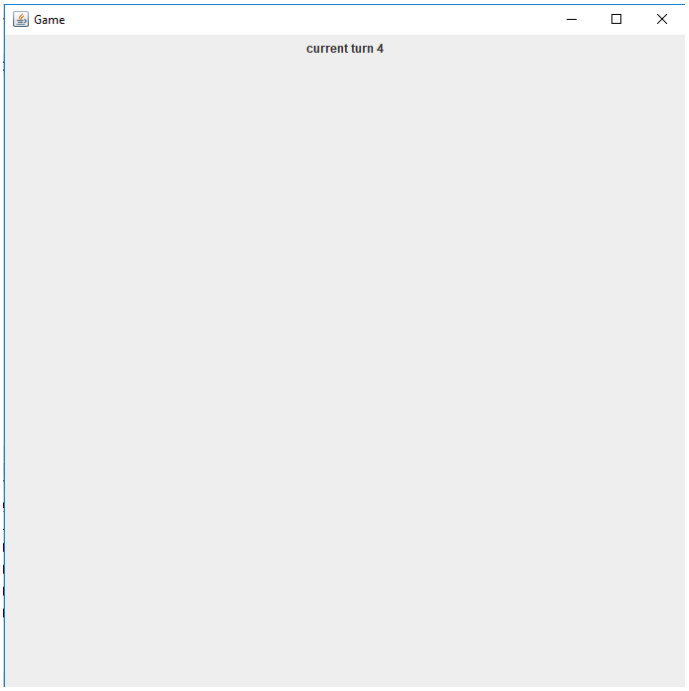

Figure 3: Affichage du jeu

4 Création du jeu Pacman

On va pouvoir maintenant facilement réutiliser l'architecture du jeu développée jusqu'ici pour créer le jeu Pacman. Certains éléments vous seront fournis pour ne pas perdre trop de temps, notamment pour la réalisation de l'interface graphique du jeu.

4.1 Matériel fourni

Dans la section TP pacman de Moodle se trouvent différents éléments qui vous seront utiles pour créer le jeu :

- Il y a tout d'abord un répertoire nommé *layouts* qui contient une base de labyrinthes au format texte
- Dans le répertoire *Sourcesjava* se trouvent trois fichiers :
  1. *Maze.java* vous fournit une classe qui charge l'ensemble du labyrinthe dans un objet *Maze*. Le constructeur de cette classe prend comme argument le nom du fichier texte du labyrinthe à charger.
  2. *PositionAgent.java* permet simplement de définir la position d'un agent et sa direction
  3. Le fichier *PanelPacmanGame.java* est créé en prenant en entrée un labyrinthe. C'est un JPanel qui permet l'affichage le jeu en fonction du labyrinthe mais aussi des positions courantes des agents. A noter que la méthode *paint* sera appelée automatique lorsque l'appel *setVisible(true)* sera effectué sur le JPanel. Une mise à jour du plateau peut être effectuée en appelant la méthode *repaint*.

4.2 Réalisations à effectuer pour obtenir une première simulation de jeu

A partir de maintenant les consignes sont volontairement moins précises, c'est à vous de concevoir et implémenter le jeu. Un fil directeur général est cependant proposé :

1. Créez une classe Agent qui permet de modéliser un agent, avec un type (soit Pacman, soit Fantôme) ainsi qu'une position courante

2. Créez une classe *AgentAction* qui permettra de définir l'action d'un agent dans le jeu. Combien de types d'action sont possibles pour un agent à chaque tour ?

3. Créez une classe *PacmanGame* qui étend la classe générique *Game* et qui contient un labyrinthe, une liste d'agents Pacman et une liste d'agents Fantôme.

4. Implémentez la méthode *initializeGame* pour initialiser le jeu

5. On peut ajouter un menu déroulant dans l'interface qui permettra de choisir le labyrinthe à charger, pour cela regardez la classe *JFileChooser* de Java

6. Implémentez les règles du jeu avec simplement des déplacements des agents et la possibilité de manger les pac-gommes (sans la gestion des capsules pour l'instant et sans la possibilité donné à un fantôme de manger un pacman). Pour cela il peut être utile de créer deux méthodes :

   - une méthode *isLegalMove* prend en entrée un agent et une action et renvoie vrai ou faux si l'action est possible dans le labyrinthe
   - une méthode *moveAgent* prend en entrée un agent et une action et met à jour la nouvelle position de l'agent

7. Implémentez la méthode *taketurn* qui effectue une action pour chaque agent puis met à jour le labyrinthe (par exemple si une pac-gomme a été mangée)

8. Faites en sorte que la méthode *update* de la Vue mette à jour le panneau du jeu en fonction des nouvelles positions des agents et de l'état du labyrinthe. Une fois les positions redéfinies dans l'objet *PanelPacmanGame*, il s'agira d'appeler la méthode *repaint* du JPanel pour effectuer la mise à jour graphique.

9. Implémentez un premier comportement d'agent sous la forme d'un déplacement aléatoire à chaque tour. Pour cela, implémentez une stratégie avec une méthode *getAction* qui prend en entrée un agent et le labyrinthe et renvoie une action valide à effectuer.

10. Implémentez un autre comportement d'agent simple. Quel Design Pattern peut être utile ?

11. Implémentez la mort d'un agent Pacman lorsqu'il rencontre un fantôme

12. Implémentez la règle avec les capsules, qui enclenche un *timer* de 20 périodes au cours desquelles les pacmans peuvent manger les fantômes.

13. Implémentez les conditions de fin du jeu

14. On veut maintenant créer des agents avec des stratégies qui dépendent du fait que le *timer* lié à la capsule soit enclenché ou non. Par exemple quand le *timer* est enclenché les fantômes vont avoir tendance à s'enfuir plutôt qu'à se rapprocher des Pacmans. Quel design pattern vu en cours peut être utile dans ce cas ?

15. Proposer différents types d'agents pacman et fantôme avec différents types de stratégies intégrées et qui varient en fonction du fait que le *timer* soit enclenché ou pas. Quel design pattern peut être utile pour créer ces différents types d'agent ?

16. Implémentez une stratégie qui est en mode interactif. C'est l'utilisateur qui choisit l'action (pour le pacman ou pour le fantôme). Ce choix d'action peut être fait avec le clavier ou bien avec la souris sur différents boutons.

4.3 Stratégies et conceptions avancées

Il est demandé de proposer des extensions au jeu. Le choix est libre mais entrera dans la note finale. Choisissez en binôme le thème qui vous intéresse puis discutez-en avec votre encadrant.

Plusieurs orientations sont possibles :

- Améliorer l'interface et la gestion du jeu (compte des points, du nombre de vies des pacmans), gestion des niveaux, etc...

- Ajouter un mode multijoueur (sur un seul ordinateur). Par exemple un joueur gère le pacman et un joueur gère le fantôme
- Ajouter de l'IA dans les agents (algorithme A* pour les pacmans et/ou les fantômes, coopération entre les fantômes pour ne pas prendre le même chemin et encercler le Pacman).
- Si on a le temps, un challenge pourra être mis en place sous la forme d'une compétition entre les stratégies des agents fantômes et pacmans proposées par les différents groupes