Projet Design Pattern
Pacman

Noé De Caestecker

2 décembre 2025

Table des matières

Résumé

L'objectif du projet est de créer un jeu de plateau séquentiel similaire au jeu Pacman composé d'un monde de taille limitée dans lequel évolue un certain nombre d'agents.

Les règles du jeu 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 éliminé et disparaît du plateau.
- Des capsules spéciales sont disposées sur le plateau. Une fois mangée par un pacman, elles rendent les fantômes vulnérables pendant un certain nombre de périodes au cours desquelles les agents 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 Design Patterns utilisés

L'ensemble des design patterns utilisés comprend :

Design Patterns	Utilisation	Explications
Patron de méthode	- Structure de l'algorithme de pathfinding - Structure du jeu	- le pathfinding est composé de plusieurs parties d'algorithmes pour plus de contrôle
Observateur / Observable	- Lien entre le modèle du jeu et les interfaces graphiques	
stratégie	- Comportements des agents	
composite	- Comportements complexes des agents	- certains comportements sont composés d'autres comportements et les utilisent en fonction du contexte
état	- Activation des boutons de l'interface de contrôle	
fabrique	- Création des comportements - Presets de pathfinding	- les comportements peuvent être créés en utilisant un *Enum* - l'algorithme de pathfinding demandant plusieurs classes en paramètre, la création est regroupée en presets pour plus de simplicité
singleton	- Enregistrement des EventListener	- pour les comportements au clavier, les comportements sont ajoutés dans une unique liste de listener elle-même ajoutée sur les interfaces pour écouter les events

2 Fonctionnalités ajoutées

Les fonctionnalités ajoutées les plus importantes sont :

- Possibilité d'ouvrir les murs extérieurs du layout pour créer des chemins qui font le tour du labyrinthe
- Temps d'attente dynamique entre chaque tour pour compenser les tours ayant pris trop de temps à s'exécuter
- Évaluation et affichage des tps (tours par seconde) afin de voir les ralentissements non liés à l'interface graphique
- Ajout de différents comportements :
  - un comportement aléatoire
  - un comportement d'arrêt (pas de mouvement)
  - un comportement pour jouer au clavier
  - un comportement changeant en fonction de si les fantômes sont effrayés
    (composé de 2 autres comportements)
  - un comportement changeant en fonction de la distance à un type d'entité choisi
    (composé de 2 autres comportements)
  - un comportement appliquant un autre comportement si le premier a renvoyé *null*
    (composé de 2 autres comportements)
  - un comportement de pathfinding paramétrable
- Refonte de l'affichage des murs pour une version plus jolie
- Ajout d'un espace entre les agents et les murs pour plus de visibilité
- Ajout d'une couleur propre à chaque fantôme
- Ajout d'une animation de bouche pour les pacmans
- Déplacement des yeux des fantômes en fonction de leur direction de mouvement
- Calcul et application de la taille optimale de la fenêtre
  (en gardant le rapport largeur / hauteur du labyrinthe)
- Ajout d'un bouton pour choisir le layout du labyrinthe
- Ajout d'une interface pour changer les comportements des agents

Rapport détaillé

Le reste du rapport est une description plus détaillée du projet, elle n'est pas utile et n'a pas besoin d'être lue.

3 Implémentation du Modèle

3.1 Structure globale

La structure globale du projet suit le pattern *Modèle-Vue-Contrôleur*.
La modélisation du jeu est composée de 2 classes principales :

- Une classe *Maze* qui stocke la structure du labyrinthe.
- Une classe *PacmanGame* qui gère l'architecture séquentielle du jeu et stocke les différents agents.

Maze

La classe *Maze*, fournie dans les fichiers source à utiliser pour le projet, a été modifiée pour pouvoir ajouter quelques fonctionnalités.

- Ajout d'un *Enum EntityType* pour donner un type d'entité en paramètre de méthodes.
- Ajout de méthodes pour récupérer la position de l'entité d'un type donné la plus proche d'une position donnée (utile pour les comportements des agents).
- Possibilité d'ouvrir les murs extérieurs du layout pour créer des chemins qui font le tour du labyrinthe (comme dans le pacman originel).

PacmanGame

La classe *PacmanGame* utilise le pattern *Patron de méthode* en héritant d'une classe abstraite *Game* qui gère la structure séquentielle du jeu. Parmi les changements d'implémentation de la classe *Game*, on trouve

- Temps d'attente dynamique entre chaque tour pour compenser les tours ayant pris trop de temps à s'exécuter.
- Évaluation des tps (tours par secondes) afin de voir les ralentissements non liés à l'interface graphique.

Les classes *Game* et *PacmanGame* implémentent aussi le pattern *Observateur / Observable* utilisé par les interfaces graphiques pour transmettre des informations comme le passage d'un tour, la fin d'une partie, les tps et l'initialisation du labyrinthe.

L'ensemble des agents est stocké dans la classe *PacmanGame* sous forme d'une liste d'agent, sans distinction entre pacmans et fantômes. Lors d'un tour, les actions de tous les agents sont évaluées et stockés puis chaque agent réalise son action. Un dernier passage est ensuite réalisé pour savoir si des agents ou des capsules / nourritures ont été mangés.

La vérification du fait qu'un agent en mange un autre est réalisée par l'agent qui a une méthode prenant en paramètre un *EntityType* et renvoyant si oui ou non il est capable de manger cette entité.

3.2 Agents

Les agents sont représentés pas deux classes *PacmanAgent* et *GhostAgent* héritant de la classe *Agent*. Ces agents utilisent le pattern *stratégie* pour définir leur comportement. En plus de leur position actuelle, ils stockent leur position précédente afin de pouvoir identifier les cas où un pacman et un fantôme échangent de position.

3.3 Comportements

Les comportements sont un ensemble de classes héritant toutes d'une classe *Behavior*. Cette classe est principalement composée d'une méthode qui renvoie le prochain mouvement à faire pour l'agent à qui elle est associée (ou *null* si aucun mouvement ne semble bon).

Certains comportements utilisent le pattern *composite* en étant composé de deux autres comportements afin de créer des comportements plus complexes dépendants du contexte.

Les comportements implémentés comprennent :

- un comportement aléatoire
- un comportement d'arrêt (pas de mouvement)
- un comportement pour jouer au clavier
- un comportement changeant en fonction de si les fantômes sont effrayés (composé d'un comportement à réaliser si les fantômes sont effrayés et d'un autre comportement à réaliser dans le cas contraire)
- un comportement changeant en fonction de la distance à un type d'entité choisi
- un comportement appliquant un autre comportement si le premier a renvoyé *null*
- un comportement de pathfinding paramétrable

3.3.1 Pathfinding

Le comportement de pathfinding utilise lui aussi le pattern *Patron de méthode* mais est composé de 4 classes décrivant les différentes parties de l'algorithme A* utilisé. Ces différentes parties sont :

- l'ordre des mouvements à tester pour propager chaque cellule (pour éviter ou non une préférence de direction arbitraire)
- le poids à utiliser (en plus de la taille du chemin) pour l'ordre de propagation des cellules (distance euclidienne à une cible, distance de Manhattan, distance négative pour fuir une cible, ...)
- la condition de fin de chemin (cible atteinte, distance minimum pour fuir, ...)
- les mouvements autorisés (pour empêcher ou non les fantômes de traverser d'autres fantômes ou toute autre entité choisie)

4 Implémentation des Vues et des Contrôleurs

Les classes responsables des interfaces graphiques implémentent le pattern *Observateur / Observable* pour récupérer les informations du modèle en cas de changement et les afficher.

Trois interfaces sont présentes : le jeu, le panneau de contrôle et le panneau de comportements.

4.1 Le jeu

L'interface du jeu utilise l'interface fournie dans les fichiers source à utiliser pour le projet, avec quelques modifications graphiques :

- réécriture de l'affichage des murs pour une version plus jolie
- ajout d'un facteur de réduction pour ajouter un espace entre les agents et les murs
- ajout d'une couleur propre à chaque fantôme
- ajout d'une animation de bouche pour les pacmans
- déplacement des yeux des fantômes en fonction de leur direction de mouvement
- calcul et application de la taille optimale de la fenêtre en gardant le rapport largeur / hauteur du labyrinthe

4.2 Le panneau de contrôle

L'interface du panneau de contrôle est similaire à celle à implémenter pour le projet avec les changements suivants :

- ajout des tps sur l'affichage
- ajout d'un bouton pour choisir le layout du labyrinthe

Cette interface utilise le pattern *état* pour désactiver certains boutons lorsque le jeu est lancé, arrêté ou a fait une seule itération.

4.3 Le panneau de comportements

L'interface du panneau de comportements est un nouveau panneau qui présente une interface pour changer le comportement de chaque agent. Il dispose de deux flèches pour sélectionner un agent, d'un bouton pour arrêter la sélection et d'un affichage en arbre du comportement de l'agent sélectionné pour le modifier.