Rapport de Conception
Jeu de Mémoire

Rachmaine Imane , Taghi Chaimaa

15 avril 2025

Table des matières

1 Introduction et Architecture

Ce document présente la conception et l'implémentation d'un jeu de mémoire interactif développé en Java avec Swing, permettant à un ou deux joueurs de tester leur mémoire visuelle en reproduisant des formes géométriques.

1.1 Architecture générale

L'architecture du projet repose sur le modèle MVC (Modèle-Vue-Contrôleur) avec les packages suivants :

- `model` : Classes représentant les données du jeu
- `view` : Classes gérant l'interface utilisateur
- `controller` : Classes gérant la logique du jeu
- `command` : Implémentation du pattern Command
- `strategy` : Implémentation du pattern Strategy pour la génération de formes
- `evaluation` : Implémentation du pattern Strategy pour l'évaluation des scores
- `state` : Implémentation du pattern State

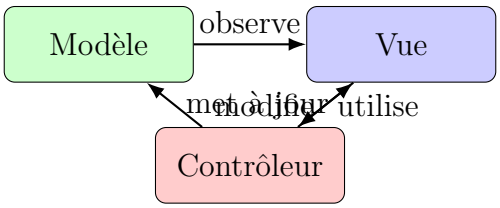

FIGURE 1 – Pattern MVC

2 Design Patterns implémentés

2.1 Pattern État (State)

Le pattern État permet de modifier le comportement d'un objet lorsque son état interne change. Dans notre application, ce pattern gère les différents états d'interaction de l'utilisateur avec le jeu.

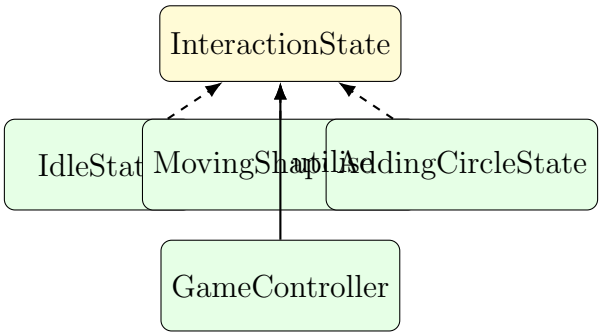

FIGURE 2 – Pattern État

Les principaux états sont :

- `IdleState` : État par défaut, en attente d'interaction
- `MovingShapeState` : État lors du déplacement d'une forme
- `AddingCircleState` : État lors de l'ajout d'un cercle
- `AddingRectangleState` : État lors de l'ajout d'un rectangle

2.2 Pattern Commande (Command)

Le pattern Command encapsule une requête sous forme d'objet, permettant les opérations réversibles.

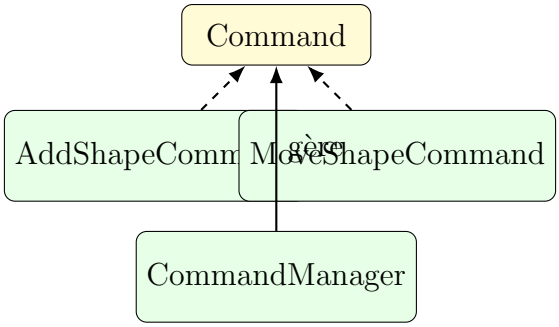

FIGURE 3 – Pattern Commande

Ce pattern permet :

— L'annulation (undo) et la répétition (redo) des actions
— La séparation entre l'action et son exécution

2.3 Pattern Stratégie et Fabrique

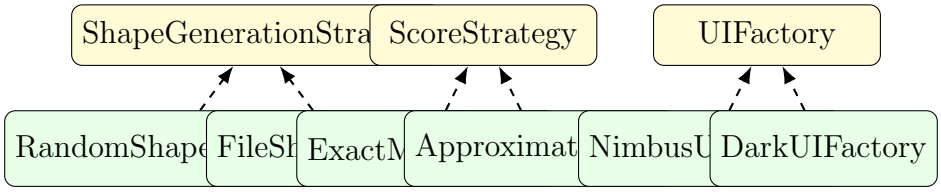

FIGURE 4 – Patterns Stratégie et Fabrique

Pattern Stratégie : Utilisé pour la génération de formes (`ShapeGenerationStrategy`) et l'évaluation des scores (`ScoreStrategy`).

Pattern Fabrique : La fabrique `UIFactory` permet de changer facilement l'apparence de l'application avec différentes implémentations (Nimbus, Dark, Metal).

2.4 Pattern Observateur (Observer)

Le pattern Observateur définit une relation un-à-plusieurs permettant la notification automatique lors des changements d'état.

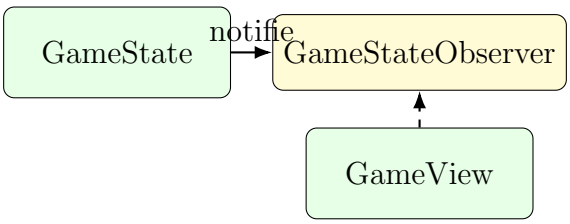

FIGURE 5 – Pattern Observateur

Ce pattern est utilisé pour maintenir la synchronisation entre le modèle et les vues, en notifiant automatiquement les changements d'état du jeu.

3 Algorithmes et Fonctionnalités

3.1 Algorithmes principaux

Génération de formes aléatoires :

1. Génération d'un nombre aléatoire de formes
2. Pour chaque forme, sélection aléatoire du type, de la position, de la couleur et des dimensions
3. Vérification de non-superposition avec les formes existantes

Évaluation du score :

1. Pour chaque forme du modèle, recherche de la forme la plus similaire dessinée par le joueur
2. Évaluation basée sur la position, la taille, la couleur et le type
3. Calcul du score final en pourcentage

3.2 Fonctionnalités

Mode un joueur / deux joueurs :

— Mode un joueur : formes générées automatiquement à mémoriser
— Mode deux joueurs : le premier joueur dessine, le second reproduit

Sauvegarde et interface adaptable :

— Sauvegarde des ensembles de formes pour réutilisation
— Choix entre plusieurs thèmes visuels (Nimbus, Metal, Dark)
— Actions réversibles (annulation et répétition)

4 Conclusion

Cette application a été conçue en suivant les principes de la conception orientée objet avec plusieurs design patterns pour assurer une architecture modulaire et extensible.

Perspectives d'amélioration :

— Ajout de nouveaux types de formes
— Niveaux de difficulté progressifs
— Mode multijoueur en réseau

A Captures d'écran

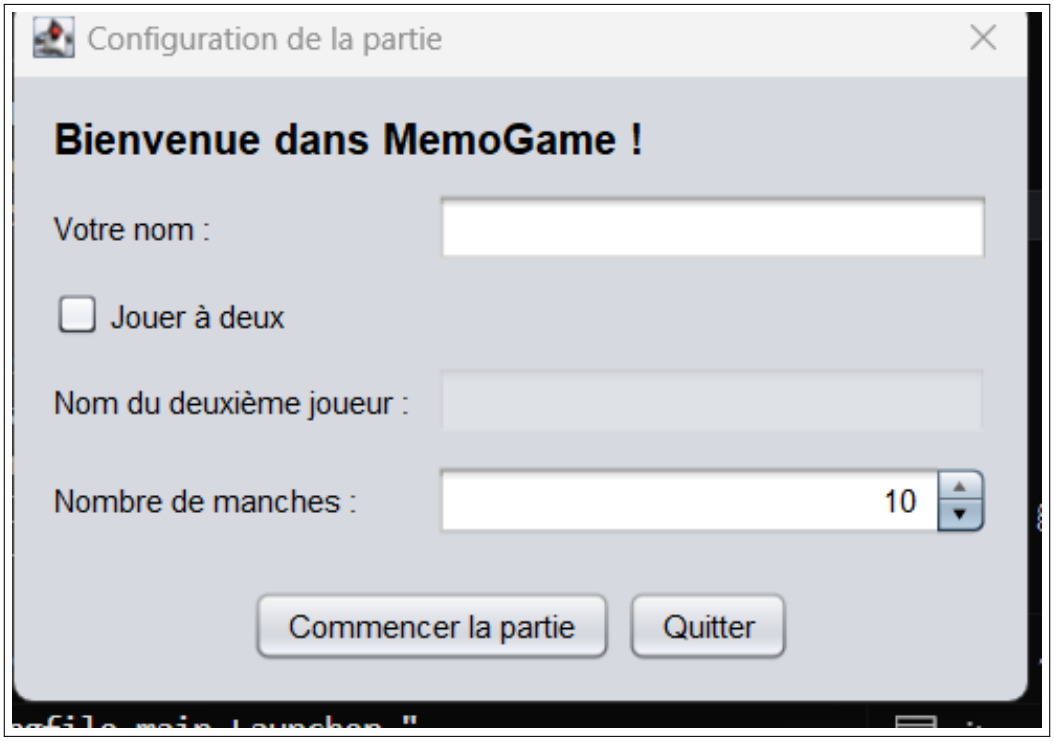

FIGURE 6 – Configuration Session

FIGURE 7 – Interface principale du jeu

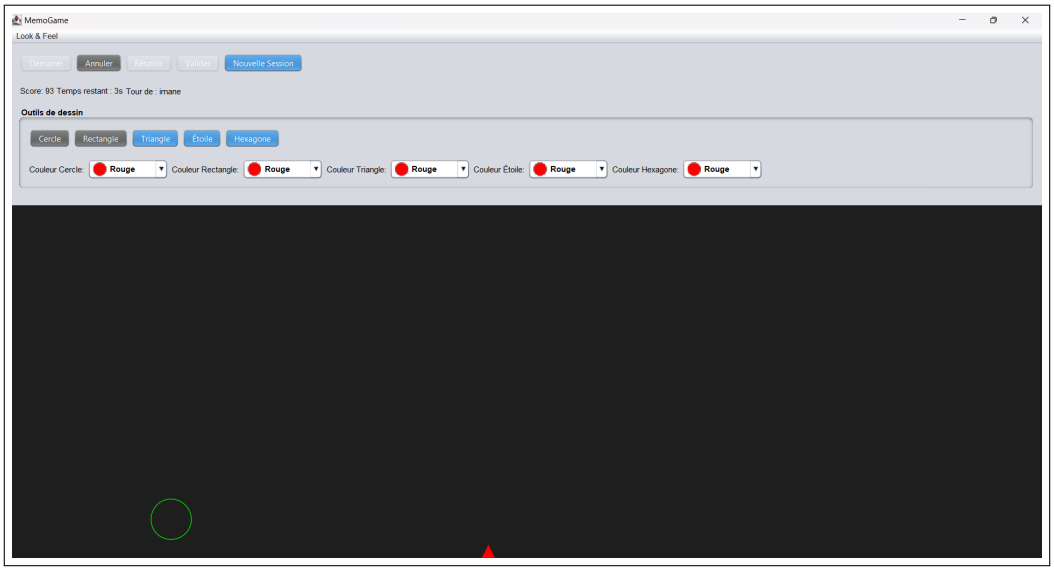

FIGURE 8 – Application avec le thème sombre