UNIVERSITÉ
CAEN
NORMANDIE

Développement d'un Jeu de Blackjack

Méthodes de Conception

THOMAS Matthieu : 22304534

SIAGHI Massinissa : 22312276

TELLIER Basile : 22104032

5 décembre 2025

Table des matières

1 Introduction

1.1 Présentation générale du projet

Le projet s'inscrit dans le cadre de l'UE *Méthodes de conception*. Il consiste à développer un **jeu complet de Blackjack**, en plusieurs étapes :

- Développer un **jeu de cartes générique** : cartes, paquet, mélange, tirage...
- Développer un **jeu de Blackjack** reposant sur ce modèle de cartes.
- Implémenter les règles complètes : distribution, actions, gains, blackjack, bust, split, etc.
- Proposer une **version évoluée** du jeu adaptée au modèle MVC.

Le projet a donc suivi une progression logique structurée en deux grandes étapes :

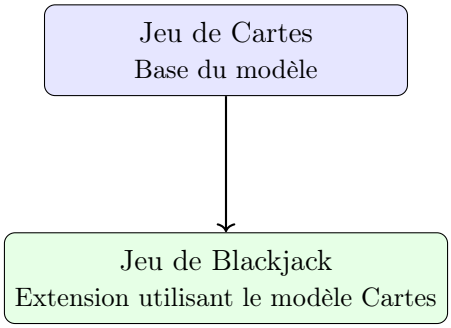

FIGURE 1 – Présentation des deux grandes étapes du projet de Blackjack

1.2 Problématique et points clés

Le développement de ce jeu de Blackjack a posé plusieurs défis techniques et de conception :

- **Modélisation d'un jeu de cartes générique** : Créer une base solide et réutilisable pour représenter les cartes, les paquets, les mélanges et les tirages, tout en garantissant l'intégrité des données et en évitant les incohérences.
- **Implémentation des règles complexes du Blackjack** : Le Blackjack comporte de nombreuses règles spécifiques (blackjack naturel, assurance, split, double, etc.) qui doivent être implémentées de manière précise et cohérente. La gestion des états des joueurs (bust, blackjack, stand) et du croupier a nécessité une attention particulière.
- **Architecture MVC pour les deux modules** : Adopter le modèle MVC à la fois pour le jeu de cartes générique et pour le jeu de Blackjack a demandé une réflexion sur la séparation des responsabilités. Il fallait garantir que le modèle soit indépendant de la vue, et que le contrôleur orchestre correctement les interactions.
- **Gestion des joueurs et des stratégies** : Le système devait supporter différents types de joueurs (humains, IA) avec des stratégies de jeu variées. L'implémentation de l'IA, en particulier, a nécessité la définition de comportements basés sur les règles classiques du Blackjack.
- **Interface graphique interactive** : La vue devait présenter de manière claire l'état du jeu (mains des joueurs, cartes du croupier, scores, soldes) et permettre aux joueurs d'effectuer leurs actions (tirer, rester, doubler, split, etc.). La synchronisation entre le modèle et la vue via le pattern Observer a été cruciale.
- **Tests exhaustifs** : Assurer la fiabilité du modèle à travers des tests unitaires couvrant les cas normaux et les cas limites, notamment pour les règles complexes comme le split et l'assurance.

— **Maintenir la modularité et l'évolutivité** : Le code devait être structuré de manière à faciliter l'ajout de nouvelles fonctionnalités (nouvelles règles, nouveaux types de joueurs, nouvelles interfaces) sans compromettre la stabilité du système existant.

1.3 Présentation du plan du rapport

Ce rapport a pour objectif de retracer de manière claire et progressive le développement du projet **Blackjack**. Il est structuré en sept sections, chacune correspondant à une phase essentielle du projet :

— **Section 1 – Introduction** : Pose le contexte du projet, ses objectifs pédagogiques et les motivations ayant guidé notre équipe.

— **Section 2 – Structuration du projet** : Décrit les besoins fonctionnels et techniques, les fonctionnalités envisagées, ainsi que l'organisation du travail collaboratif.

— **Section 3 – Éléments techniques** : Détaille les algorithmes, structures de données et patterns de conception utilisés, notamment le moteur de jeu, la gestion des paquets de cartes, et les stratégies de jeu.

— **Section 4 – Architecture du projet** : Propose une vue d'ensemble de l'architecture logicielle, articulée autour du patron MVC pour les deux modules (Cartes et Blackjack). Cette partie s'appuie sur des diagrammes UML pour illustrer les relations entre les différentes classes et modules.

— **Section 5 – Tests du modèle** : Démontre les différents tests réalisés pour éprouver la robustesse des méthodes des modèles de Cartes et de Blackjack.

— **Section 6 – Expérimentations et usages** : Illustre l'utilisation concrète du jeu à travers des cas d'usage commentés et des captures d'écran, en soulignant les résultats observés.

— **Section 7 – Conclusion** : Dresse un bilan général du projet, récapitule les fonctionnalités implémentées et propose plusieurs axes d'amélioration pour la suite.

2 Structuration du projet

2.1 Analyse des besoins

Les besoins fonctionnels identifiés pour le projet Blackjack sont les suivants :

- **Jeu de cartes générique** :
  - Représentation des cartes (valeur, couleur)
  - Gestion d'un paquet de cartes (création, mélange, tirage)
  - Support pour différentes tailles de jeux (52 cartes, 32 cartes)
- **Jeu de Blackjack** :
  - Distribution initiale des cartes
  - Tour de jeu pour chaque joueur (tirer, rester, doubler, split, assurance)
  - Tour du croupier selon les règles standard
  - Calcul des gains et des pertes
  - Gestion des soldes des joueurs
- **Interface utilisateur** :
  - Interface graphique pour le Blackjack (table, mains, scores, actions)
  - Support pour plusieurs joueurs (humains et IA)
- **Architecture logicielle** :
  - Respect du modèle MVC pour une séparation claire des responsabilités
  - Code modulaire et extensible
  - Tests unitaires complets

2.2 Fonctionnalités implémentées

Les fonctionnalités suivantes ont été implémentées avec succès :

- **Module Cartes** :
  - Classes `Carte`, `Paquet` avec toutes les opérations de base
  - Factory pour créer des jeux de cartes standards (52, 32 cartes)
  - Mélange, coupe, tirage aléatoire
  - Modèle MVC complet avec vue graphique interactive
- **Module Blackjack** :
  - Implémentation complète des règles du Blackjack
  - Gestion des joueurs (humains et IA) avec différentes stratégies
  - Actions spéciales : split, double, assurance
  - Calcul automatique des scores (avec gestion de l'As flexible)
  - Détermination des gagnants selon les règles professionnelles
  - Modèle MVC avec vue graphique complète
- **Intégration** :
  - Le module Blackjack utilise le module Cartes comme base
  - Interface graphique unifiée permettant de jouer au Blackjack
  - Tests unitaires couvrant les deux modules

2.3 Organisation du travail

Le travail a été réparti entre les trois membres du groupe de manière à valoriser les compétences de chacun :

Membre	Responsabilités principales
Matthieu	Implémentation du **modèle Blackjack** : — Conception de la logique modèle du Blackjack — Tests pour le modèle de Blackjack — Aide à la conception des tests Cartes — UML des classes de Blackjack — Rédaction du rapport
Massinissa	Développement de la **vue** et du **contrôleur** : — Interfaces graphiques (Cartes, Blackjack, etc.) — Mise en place du MVC pour Cartes & Blackjack — Mise en place du Contrôleur Cartes — Mise en place du Contrôleur Blackjack — Mise en place du Modèle & Vue Blackjack — UML Blackjack — Rédaction du rapport
Basile	Conception du **modèle Cartes** : — Conception du modèle de Cartes — Tests de Cartes — UML de Cartes — UML de Blackjack — Rédaction du rapport — Aide à la conception des tests Blackjack

TABLE 1 – Répartition des rôles au sein de l'équipe

Une collaboration régulière a été assurée via un dépôt Git, avec un suivi des commits et une validation croisée du code. Des séances hebdomadaires de synchronisation ont permis de maintenir une vision partagée de l'avancement.

3 Éléments techniques

3.1 Structures de données utilisées

Les principales structures de données utilisées dans le projet sont :
- **Classe Carte** : Représente une carte à jouer avec une valeur et une couleur.
- **Classe Paquet** : Représente un ensemble de cartes. Utilise une `List<Carte>` pour stocker les cartes et fournit des méthodes pour mélanger, tirer, couper, etc.
- **Classe Main** : Représente la main d'un joueur au Blackjack. Gère la liste des cartes, calcule le score (avec gestion de l'As flexible) et détermine l'état (blackjack, bust, etc.).
- **Classe Joueur** : Classe de base pour les joueurs. Contient le solde, la mise, la main, et l'état du joueur.
- **Classe Croupier** : Représente le croupier avec des règles spécifiques pour le tirage des cartes.
- **Classe Action** : Représente une action parmi les actions que le joueur peut exécuter et qui agit sur sa main.

3.2 Patterns de conception

Plusieurs patterns de conception ont été utilisés pour structurer le code :

3.2.1 Observer Pattern

Utilisé pour mettre en œuvre la communication entre le modèle et la vue. Lorsque l'état du modèle change (par exemple, une carte est tirée), la vue est automatiquement mise à jour.

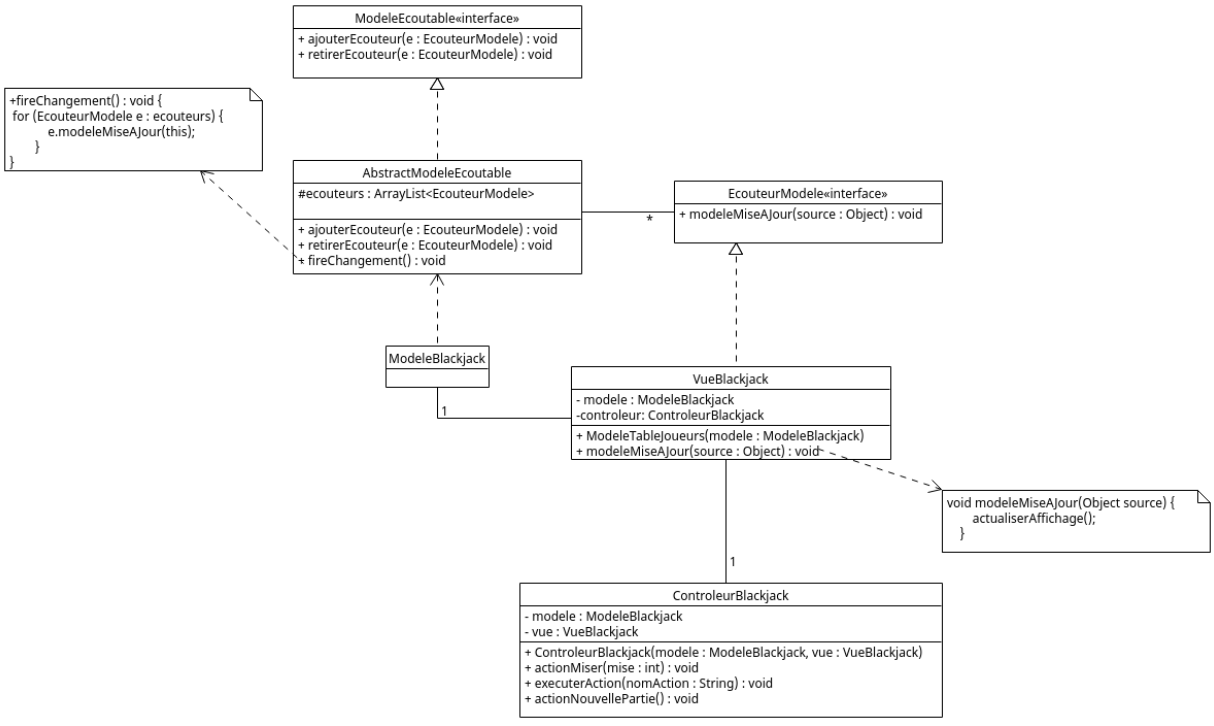

FIGURE 2 – Diagramme UML du pattern Observer

3.2.2 Strategy Pattern

Utilisé pour implémenter les différentes stratégies de jeu des joueurs IA. Chaque stratégie (basique, prudente, agressive) implémente une interface commune `StrategieJeu`.

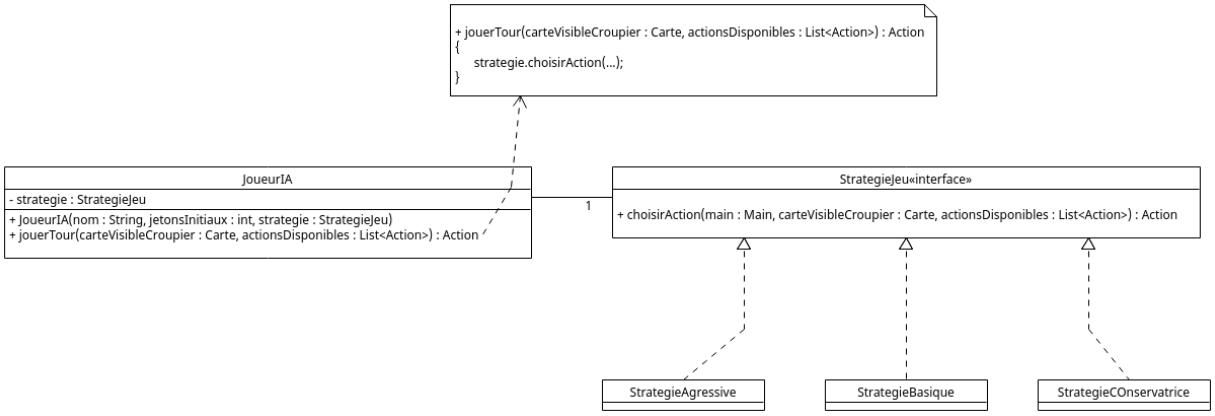

FIGURE 3 – Diagramme UML du pattern Strategy

3.2.3 Factory Method

Utilisé dans la classe `Paquet` pour créer des jeux de cartes standards (52 ou 32 cartes) via les méthodes `creerPaquet52()` et `creerPaquet32()`. Voici l'implémentation de ces méthodes :

```
private final static String[] TABLE_VAL_32 = {
    "As", "7", "8", "9", "10", "Valet", "Dame", "Roi"
};
public static Paquet creerJeu32() {
    return creationPaquet(TABLE_VAL_32);
}
public static Paquet creationPaquet(String[] hauteurs){
    String[] couleurs = {"Trefle", "Carreau", "Coeur", "Pique"};
    Paquet p = new Paquet();
    for (String couleur : couleurs) {
        for (String hauteur : hauteurs) {
            p.ajouter(new Carte(hauteur, couleur));
        }
    }
    return p;
}
```

Ces méthodes utilisent la méthode `creationPaquet(String[] hauteurs)` pour générer les paquets avec les hauteurs et couleurs appropriées.

3.2.4 Adapter Pattern

Utilisé pour adapter l'affichage des données des joueurs dans une table dynamique `JTable`, notamment pour l'affichage des scores, gains, pertes, etc.

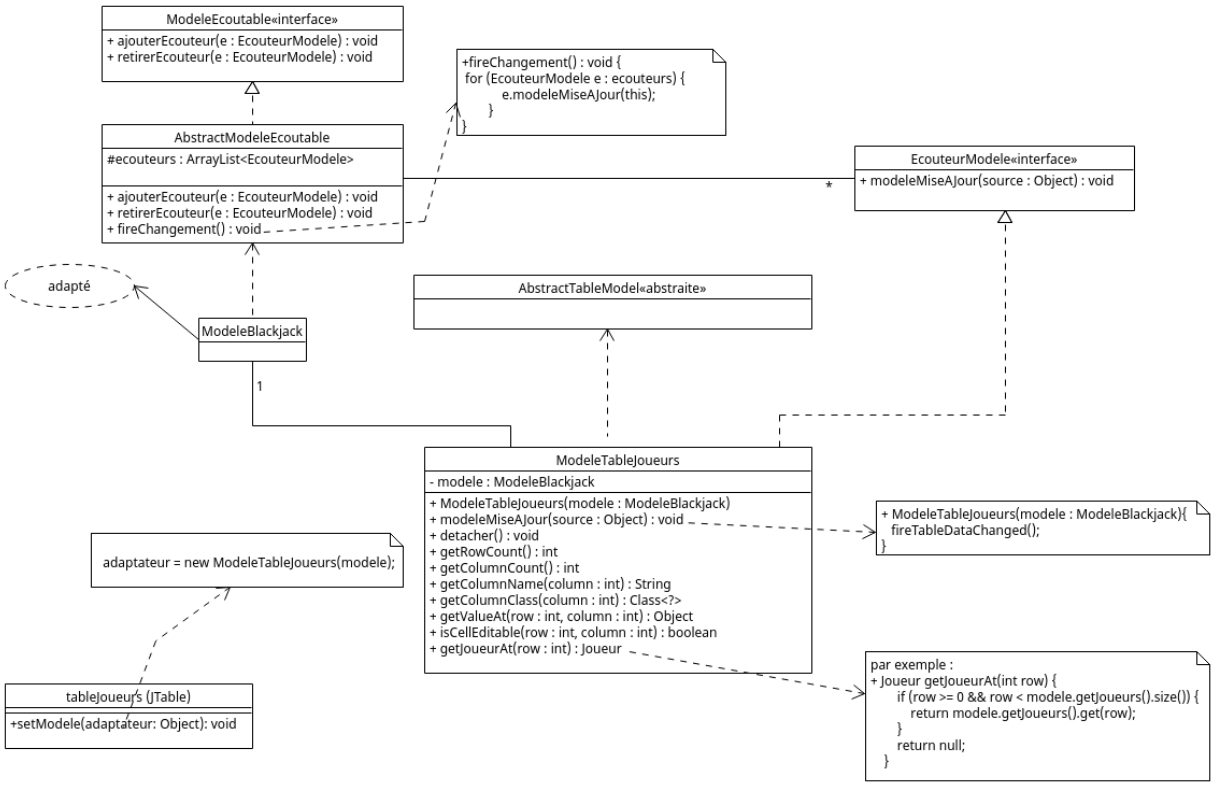

FIGURE 4 – Diagramme UML du pattern Adapter combiné avec Observer

3.2.5 Command Pattern

Utilisé pour rendre le jeu extensible et pouvoir ajouter de nouvelles actions simplement.

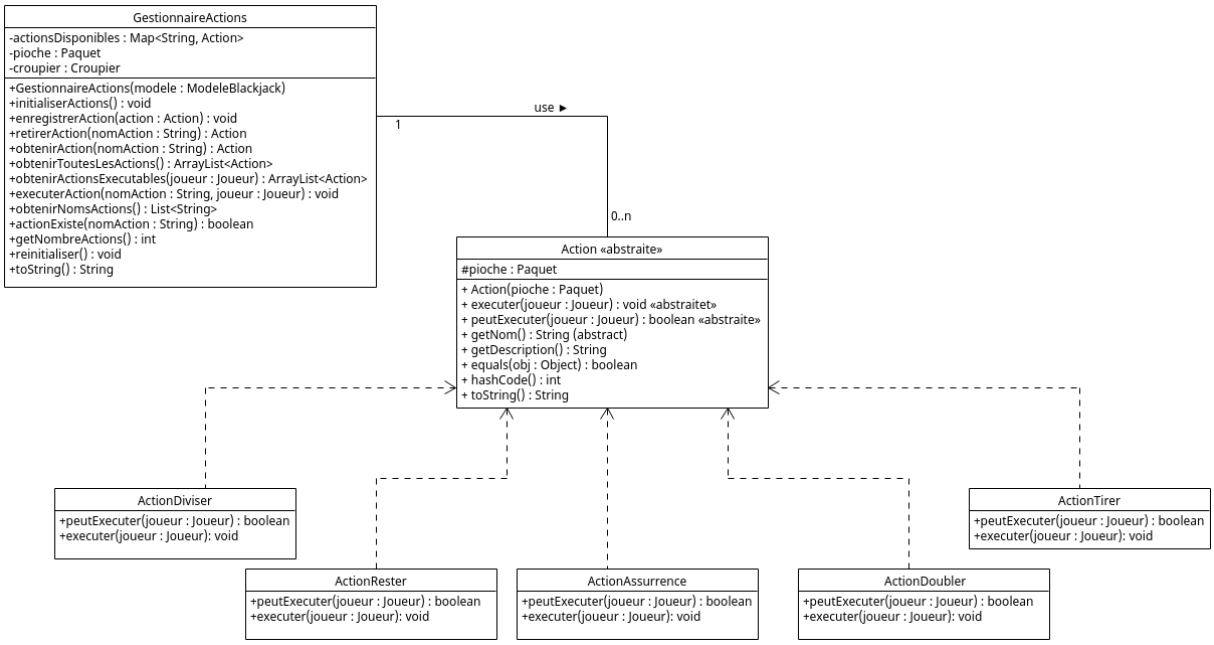

FIGURE 5 – Diagramme UML du pattern Command

3.3 Tests Mock

Nous avons mis en place des mocks dans les tests pour la classe `JoueurIA` car celle-ci utilise des actions provenant de `StrategieJeu`, un répertoire différent du modèle. Ceci nous a permis de tester les différentes méthodes propres au `JoueurIA` sans dépendre du package de stratégie de jeu. Pour les mêmes raisons, nous avons utilisé un mock de `Strategie` lors des tests de `ModeleBlackjack`.

4 Architecture du projet

Ce projet de jeu de Blackjack a été réparti en deux architectures distinctes tout en gardant une dépendance entre les modules Cartes et Blackjack.

4.1 Architecture de Cartes

4.1.1 Contrôleur : orchestration de la simulation

Le contrôleur du module Cartes gère les interactions entre l'utilisateur et le modèle. Il interprète les actions de l'utilisateur (mélanger, tirer une carte, etc.) et met à jour le modèle en conséquence.

4.1.2 Vue : interface graphique réactive

La vue du module Cartes affiche l'état actuel du paquet de cartes (cartes restantes, cartes tirées) et permet à l'utilisateur d'interagir avec le paquet via des boutons (mélanger, tirer, etc.).

4.1.3 Communication modèle ↔ vue

La communication entre le modèle et la vue se fait via le pattern Observer. Le modèle notifie la vue des changements d'état, et la vue se met à jour en conséquence.

4.1.4 Synthèse : une architecture M-V-C

L'architecture MVC du module Cartes assure une séparation claire des responsabilités : le modèle gère la logique métier, la vue présente les données, et le contrôleur gère les interactions.

4.2 Architecture de Blackjack

4.2.1 Contrôleur : orchestration de la simulation

Le contrôleur du module Blackjack gère le déroulement d'une partie : distribution des cartes, tour des joueurs, tour du croupier, calcul des gains. Il traduit les actions des joueurs (humains via l'interface, IA via les stratégies) en modifications du modèle.

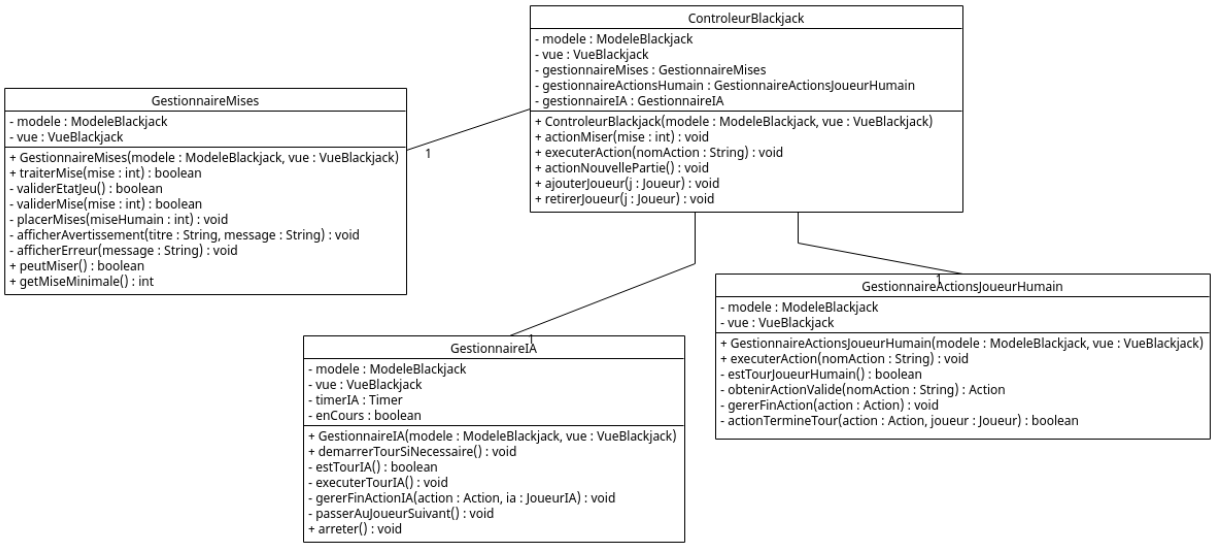

FIGURE 6 – Diagramme UML des classes du contrôleur

4.2.2 Vue : interface graphique réactive

La vue du module Blackjack affiche la table de jeu : les mains des joueurs, les cartes du croupier, les scores, les soldes, et les boutons d'action (tirer, rester, doubler, etc.). Elle est mise à jour en temps réel à chaque changement d'état du modèle grâce au pattern Observer.

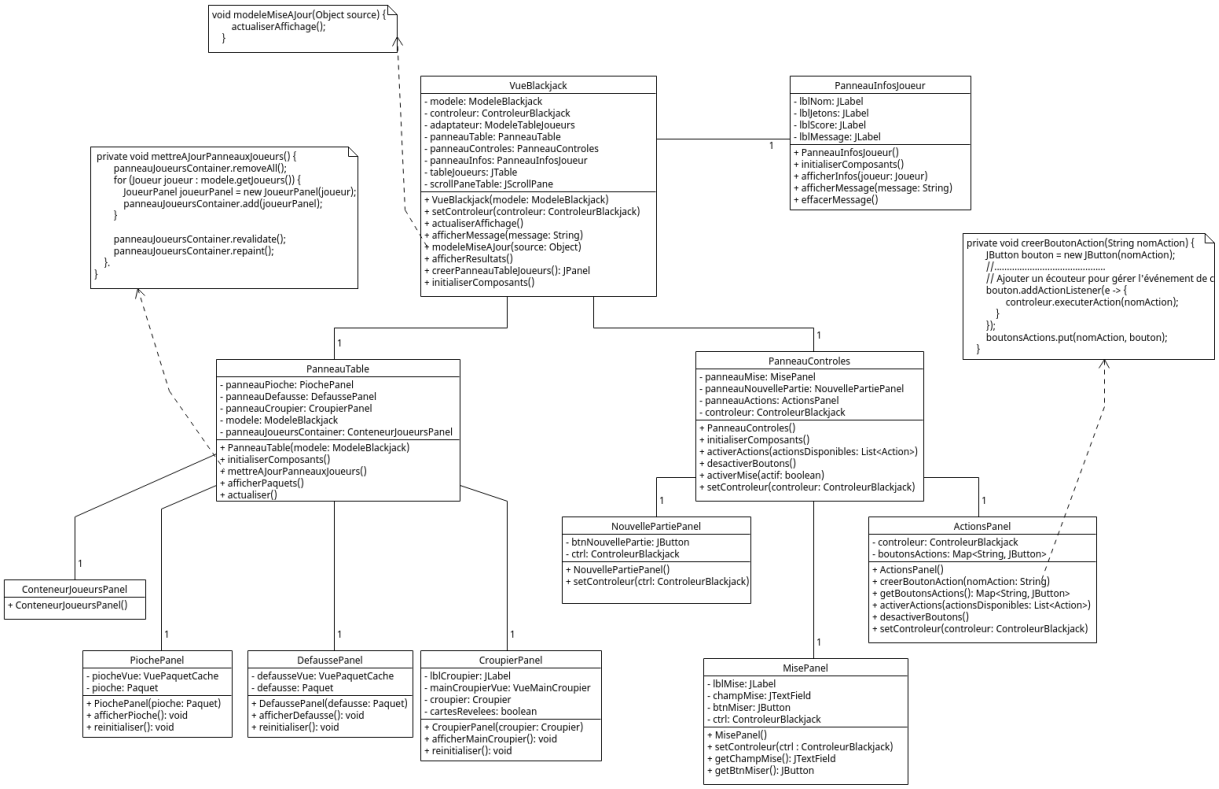

FIGURE 7 – Diagramme UML de la vue

4.2.3 Modèle : le cœur du Blackjack

Le modèle du module Blackjack contient toute la logique métier du jeu : gestion des joueurs, du croupier, des paquets de cartes, des actions, et du déroulement de la partie.

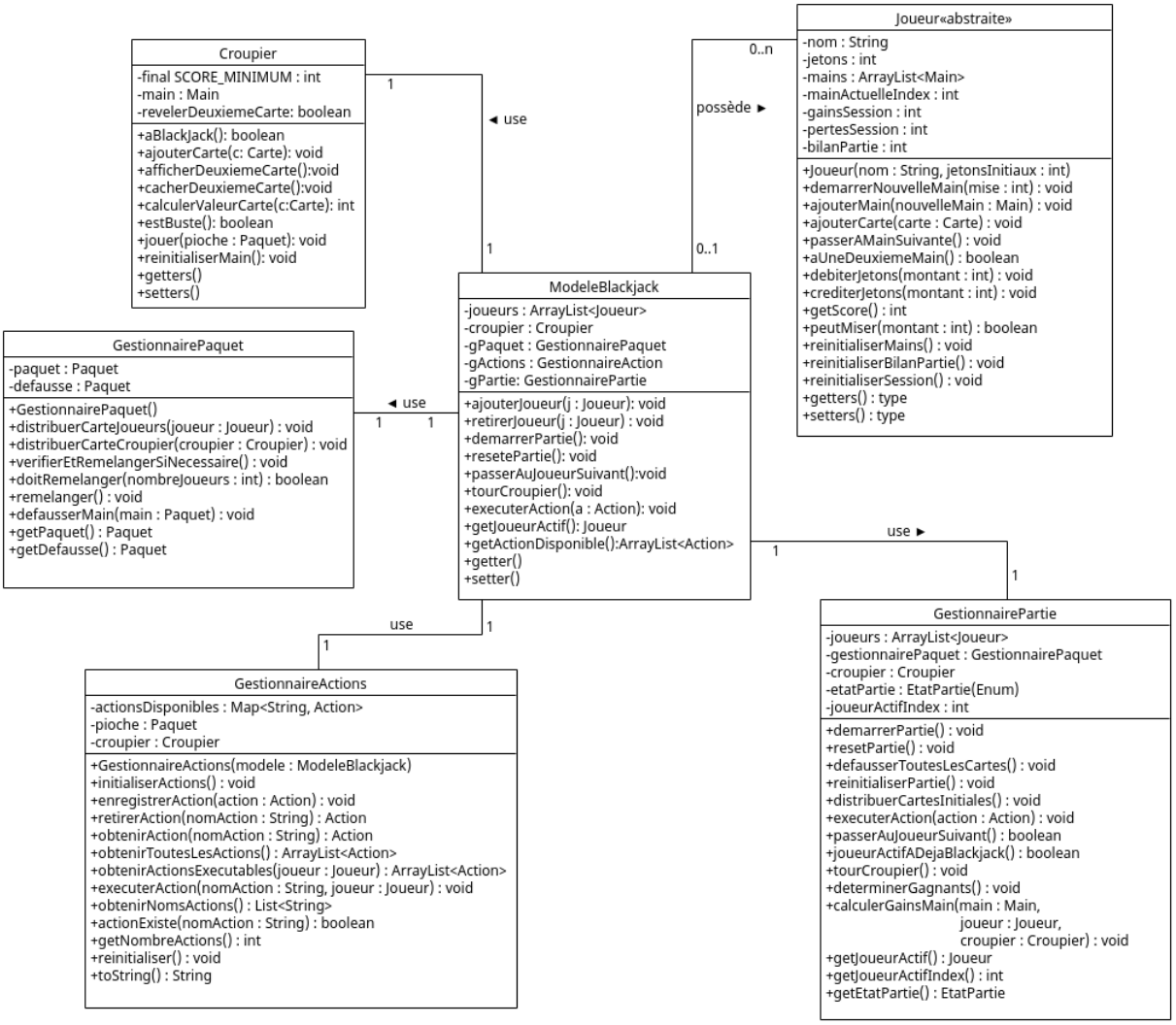

FIGURE 8 – Diagramme UML du modèle

4.2.4 Communication modèle ↔ vue

Comme pour le module Cartes, la communication entre le modèle et la vue se fait via le pattern Observer. Le modèle notifie la vue à chaque changement (nouvelle carte tirée, changement d'état d'un joueur, etc.).

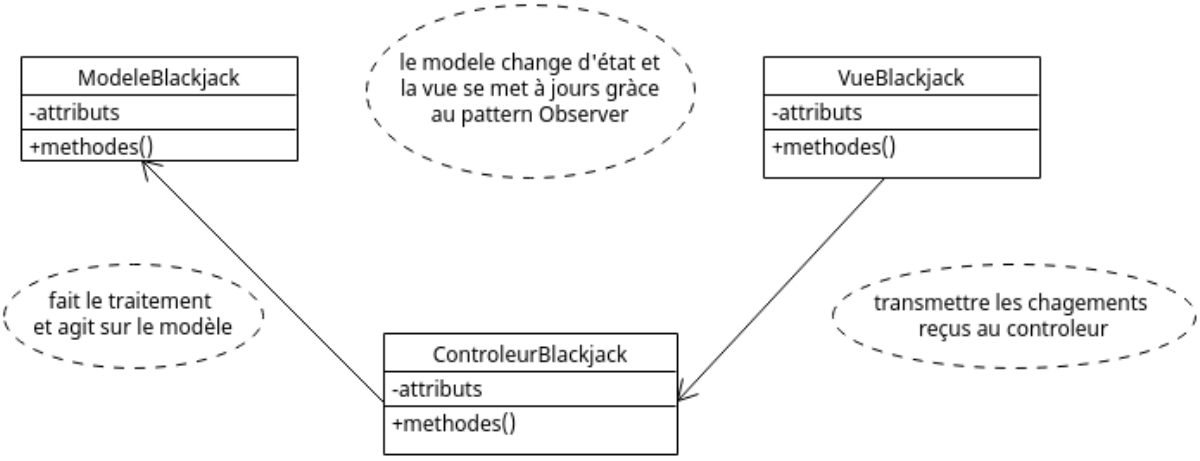

FIGURE 9 – Schéma explicatif de la communication entre la vue et le modèle

4.2.5 Synthèse : une architecture M-V-C

L'architecture MVC du module Blackjack permet une gestion claire et modulaire du jeu. Le modèle contient toute la logique métier, la vue est responsable de l'affichage, et le contrôleur gère le flux du jeu et les interactions.

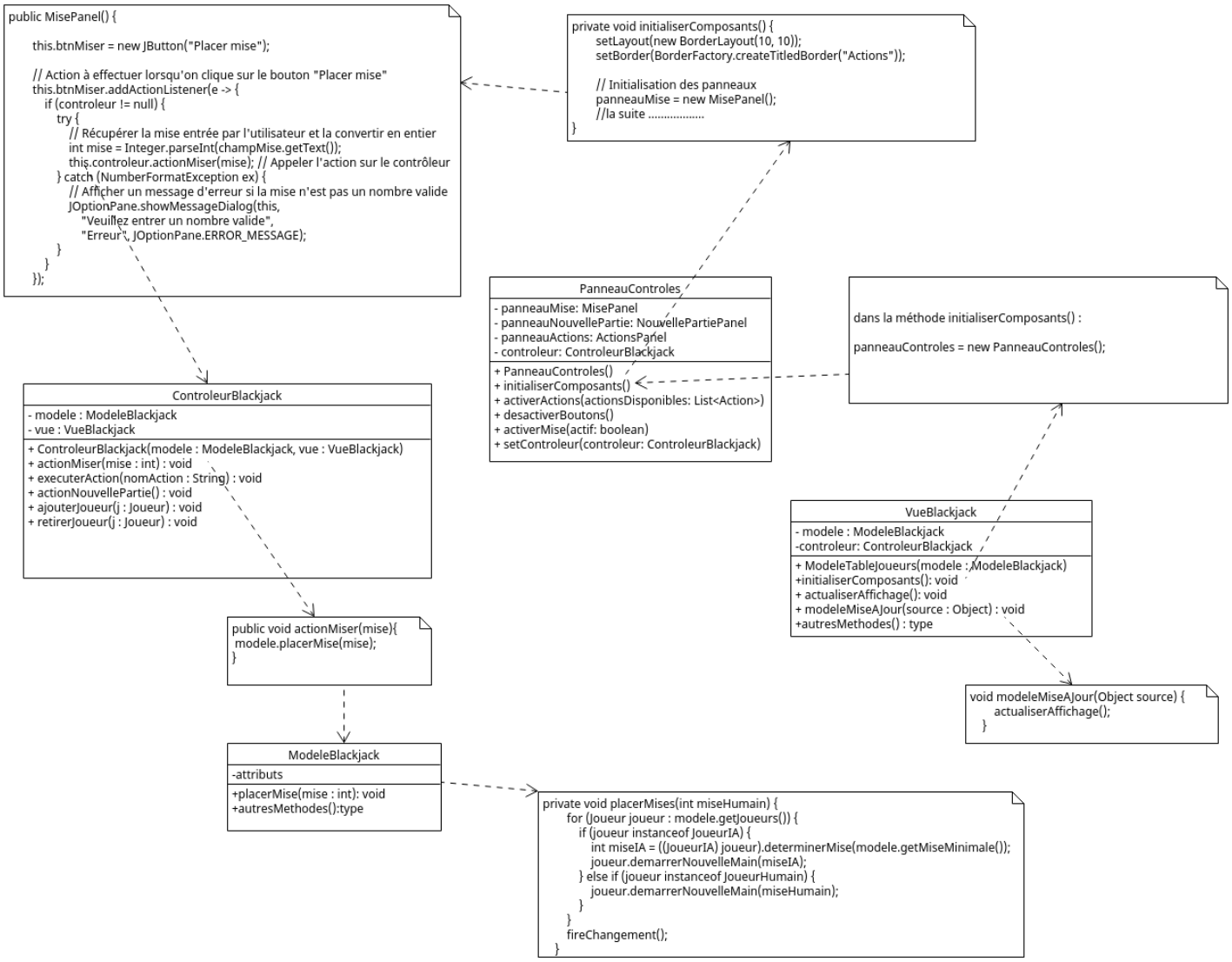

FIGURE 10 – Exemple de fonctionnement du MVC

5 Tests du modèle

Afin de valider le bon fonctionnement des différentes méthodes du modèle, nous avons conçu des classes de test couvrant les principaux cas rencontrés lors d'une partie de Blackjack. Ces tests permettent d'assurer la fiabilité des méthodes du jeu indépendamment de l'interface graphique.

5.1 Tests du modèle de Cartes

Les tests des classes liées aux cartes sont répartis en quatre classes de test, chacune ciblant un ensemble précis de fonctionnalités.

Les tests menés sur la classe `Paquet` avaient pour objectifs de valider les éléments suivants :

- **Gestion du contenu du paquet**
  - Vérification qu'un paquet nouvellement créé est vide.
  - Ajout et retrait de cartes, y compris les cas limites (doublons, retrait jusqu'à épuisement du paquet).
  - Vérification de la présence d'une carte via `contient()`.
- **Fonctionnalités avancées**
  - Sélection d'une carte aléatoire sans modifier le paquet.
  - Mélange du paquet sans perte de cartes et en conservant sa taille.
  - Test de la coupe du paquet en garantissant l'intégrité du contenu.
- **Factory method**
  - Création correcte d'un jeu de 52 cartes :
    - présence des 13 valeurs pour chacune des 4 couleurs ;
    - intégrité du contenu.
  - Vérification de la construction d'un paquet de 32 cartes standard.
- **Méthodes utilitaires**
  - Tests du `toString()` sur un paquet vide et non vide.
  - Vérification de la cohérence entre `getPaquet()` et la taille réelle du paquet.

5.2 Tests du modèle Blackjack

Les tests des classes liées au Blackjack sont répartis en trois parties propres à un dossier du modèle. Certains de ces tests peuvent avoir un comportement probabiliste, suivant certaines conditions les tests peuvent échouer.

5.2.1 Tests du modèle des actions

Les tests des actions vérifient que chaque opération (tirer, rester, doubler, etc.) modifie correctement l'état du jeu conformément aux règles du Blackjack. Ils valident également que les actions interdites (par exemple, doubler après plusieurs cartes) sont correctement refusées. Certaines actions impliquant un tirage aléatoire, des scénarios contrôlés ont été utilisés pour garantir la reproductibilité des résultats.

5.2.2 Tests du modèle des joueurs

Les tests des joueurs évaluent la gestion des mains, du solde, ainsi que des états particuliers comme le Blackjack ou le bust. Pour le `JoueurIA`, les tests vérifient la cohérence des décisions pour assurer un comportement conforme aux règles.

5.2.3 Tests du modèle de jeu

Les tests du jeu vérifient le déroulement complet d'une manche : distribution initiale, actions successives des joueurs, comportement du croupier et phase finale de résolution. Ils assurent également

la mise à jour correcte des gains et pertes, ainsi qu'une réinitialisation cohérente du jeu pour la manche suivante.

5.3 Conclusion des tests

L'ensemble des tests réalisés, couvrant les cartes, les actions, les joueurs et le modèle de jeu, a permis de valider la robustesse et la cohérence du modèle. Ces tests ont mis en évidence plusieurs dysfonctionnements au cours du développement et ont assuré une fiabilité globale du système, indépendamment de la vue ou du contrôleur. Ils constituent ainsi une base solide pour garantir un comportement conforme aux règles du Blackjack.

6 Expérimentations et usages

6.1 Cas d'utilisation : Partie basique

Une partie simple avec un joueur humain et le croupier. Le joueur mise, reçoit deux cartes, décide de tirer ou rester, puis le croupier joue selon les règles. Le système calcule automatiquement le gagnant et met à jour le solde.

6.2 Cas d'utilisation : Ajouter un joueur

L'utilisateur peut ajouter un joueur (humain ou IA) à la table. Le nouveau joueur est inclus dans la manche suivante avec un solde initial.

6.3 Cas d'utilisation : Retirer un joueur

Un joueur peut quitter la table. S'il quitte en cours de manche, sa main est terminée et il perd sa mise.

6.4 Cas d'utilisation : Split

Lorsqu'un joueur a deux cartes de même valeur, il peut choisir de splitter. Le système crée deux mains distinctes, chacune recevant une carte supplémentaire, et le joueur joue chaque main séparément.

6.5 Cas d'utilisation : Bust

Lorsqu'un joueur dépasse 21, il fait bust. Le système marque automatiquement le joueur comme perdant et retire sa mise.

6.6 Cas d'utilisation : Blackjack naturel

Si un joueur obtient un blackjack naturel (As + carte valant 10) dès la distribution, il est payé 3 :2 (sauf si le croupier a aussi un blackjack).

6.7 Cas d'utilisation : Assurance

Lorsque le croupier montre un As, les joueurs peuvent prendre une assurance. Le système gère les mises d'assurance et les rembourse si le croupier a un blackjack.

6.8 Cas d'utilisation : Double

Un joueur peut doubler sa mise après la distribution initiale. Il reçoit alors une seule carte supplémentaire et ne peut plus tirer.

7 Conclusion

Récapitulatif des fonctionnalités principales

Le projet **Blackjack** que nous avons développé propose une plateforme complète et interactive pour jouer à une table de blackjack. Voici les fonctionnalités majeures que nous avons intégrées :

- **Jeu de cartes générique** : Un module complet de gestion de cartes avec mélange, tirage, coupe, et factory pour créer des jeux standards.
- **Blackjack complet** : Implémentation de toutes les règles du Blackjack : blackjack naturel, assurance, split, double, etc.
- **Interface graphique** : Deux interfaces graphiques (une pour le module Cartes, une pour le Blackjack) interactives et intuitives.
- **Architecture MVC** : Une architecture bien séparée pour les deux modules, facilitant la maintenance et l'évolution.
- **Joueurs IA** : Des joueurs artificiels avec différentes stratégies de jeu.
- **Tests** : Une suite de tests unitaires complète pour valider le bon fonctionnement des modèles.

Propositions d'améliorations

Plusieurs axes d'amélioration peuvent être envisagés pour enrichir encore le projet :

- **Des tests plus poussés** : Notamment des tests statistiques pour vérifier que le mélange est bien uniforme, ou des tests sur un grand nombre de parties pour valider les stratégies des joueurs IA.
- **Plus de stratégies IA** : Implémenter des stratégies plus avancées (comptage de cartes, apprentissage par renforcement) pour rendre les IA plus réalistes.
- **Multijoueur en réseau** : Permettre à plusieurs joueurs humains de se connecter à une même table via le réseau.
- **Animations graphiques** : Ajouter des animations pour le tirage des cartes, le mélange, etc., pour améliorer l'expérience utilisateur.
- **Gestion du bankroll** : Ajouter des fonctionnalités de gestion de bankroll pour les joueurs (limites de mise, historiques des parties, etc.).
- **Variantes du Blackjack** : Implémenter d'autres variantes du jeu (Blackjack espagnol, Blackjack avec jokers, etc.).
- **Utilisation du pattern Composite pour l'UI** : Permettre le traitement uniforme d'objets (JPanel) et de collections d'objets.

En conclusion, ce projet nous a permis de mettre en pratique les concepts vus en cours de méthodes de conception, en particulier le modèle MVC, les patterns de conception, et les tests unitaires. Le résultat est un jeu de Blackjack fonctionnel, modulaire et extensible.