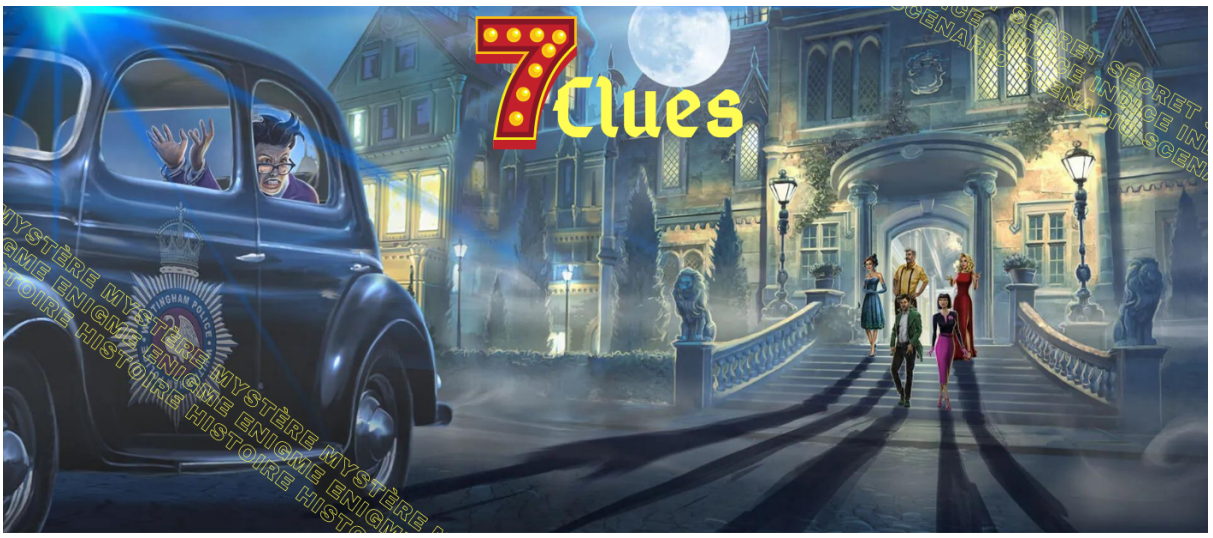
Projet long - Jeu de société "7 Clues"
Rapport général - Groupe IJ-01

Agathe Perrin, Antonin Litschgy, Mickaël Song, Maëlis Marchand,
Thierry Xu, Tom Bonetto, Antoine Dalle-Fratte



Département Sciences du Numérique - Première année
2021 - 2022

Table des matières

Introduction

L'objectif de notre projet était de réaliser un jeu de société type Cluedo qui se joue sur ordinateur. Les principales fonctionnalités du jeu de base y sont retrouvées, avec notamment : le déplacement d'un pion sur le plateau de jeu, la formulation d'hypothèses et d'accusations, l'accès à une grille pour noter ses indices au fil de la partie, etc.

Ce rapport présente ce que nous avons réalisé pour notre projet.

Principales fonctionnalités et leur état d'avancement

Fonctionnalité	Degré d'avancement	Itérations concernées (1, 2 et/ou 3)
Choisir la stratégie du bot	25%	1, 2 et 3
Choisir son personnage	100%	2
Tirer aléatoirement les 3 cartes de l'enquête	100%	3
Distribuer les cartes aux joueurs	100%	3
Installer les pions	0%	
Lancer les dés et afficher le résultat	100%	1
Choisir un déplacement possible avec son score aux dés	0%	
Déplacer son pion	0%	
Prendre un raccourci	0%	
Énoncer une hypothèse	100%	3
Chacun leur tour les joueurs montrent une carte de	0%	

leur jeu si la carte est dans les hypothèses		
Si l'accusation est vraie : le joueur gagne et la partie est finie	0%	
Si l'accusation est fausse : le joueur est éliminé et la partie continue	0%	
Ajouter des notes	100%	1
Effacer des notes	100%	1
Retour au menu	0%	
Jouer en français	100%	1, 2 et 3
Jouer en anglais	0%	
Couper le son / la musique	100%	3
Afficher les règles	100%	3
Quitter l'affichage des règles	100%	3
Revenir sur le menu principal	100%	3
Changer le thème	0%	
Quitter le jeu	100%	1 et 3

Découpage de l’application en sous-systèmes

Pour réaliser notre application, nous avons découpé celle-ci en différents sous-systèmes.

Le premier sous-système est relatif au **jeu et sa mise en place** et comprend notamment les classes Arbitre, Joueur, Jeu, Carte, TirageCrime.

Un autre sous-système concerne le **plateau** de jeu avec les classes Plateau, Salles, Salle, Couloir, Porte, Case, Position. Il permet de gérer les positions et les déplacements des pions sur le plateau.

Le 3ème sous-système porte sur le **joueur**, les classes relatives à ce dernier sont : Carnet, Hypothèse, Personnage, Carte, Bot.

Le dernier sous-système est celui de **l’affichage graphique** avec comme classes : regleJeu, ChoixPersonnage, GUI, MenuPrincipal. Ce sous-système permet au joueur d'interagir avec l’environnement graphique.

Principaux choix de conception et réalisation

Dans cette partie, nous allons présenter les principaux choix de conception et de réalisation que nous avons effectués.

Tout d'abord, nous avons réalisé une classe **Carte**. Une carte possède trois attributs :

- un attribut "type" de type **TypeCarte** qui indique si la carte est de type SALLE, PERSONNAGE ou ARME ;
- un attribut "nom" de type **NomCarte**, par exemple : "REVOLVER" ;
- un attribut "image" de type ImageIcon qui correspond au visuel de la carte, dont voici un exemple :

Les classes **TypeCarte** et **NomCarte** sont des énumérations qui listent respectivement tous les types et noms de cartes.

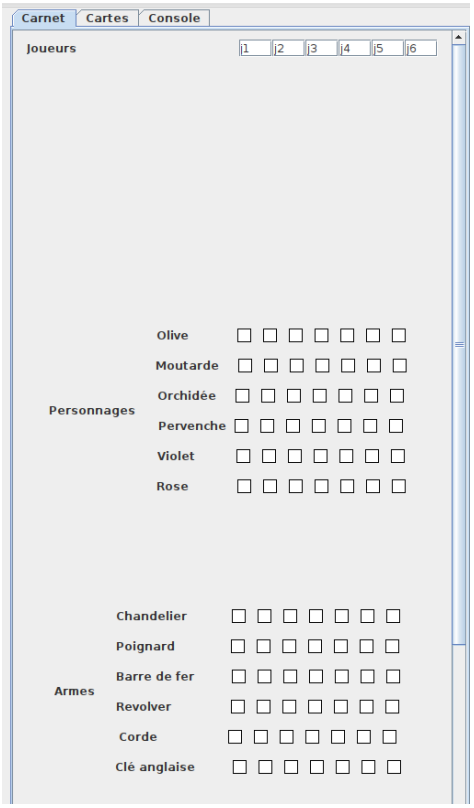
Toutes les cartes du jeu sont créées et répertoriées dans la classe **Jeu**. Ainsi, un jeu possède un paquet de cartes de type ArrayList<Carte> qui contient les 21 cartes. Dans la classe Jeu, nous avons aussi créé trois autres paquets plus petits qui contiennent respectivement les cartes Arme, Salle et Personnage, que nous utilisons notamment pour la mise en place de la partie (tirage des cartes du crime, distribution des cartes aux joueurs).

Ensuite, nous avons créé la classe **Joueur** qui représente un joueur (humain) qui joue au 7 Clues. Un joueur possède plusieurs attributs :

- un nom de type String ;
- une position sur le plateau de type Position ;
- un personnage du 7 Clues, de type Personnage ;
- un carnet de notes de type Carnet - à noter que ce carnet n'est pas directement utilisé par le joueur humain (cf. suite) ;
- un numéro de type int ;
- l'ancienne salle du plateau qu'il a visitée, de type Salle ;

Le joueur humain n'utilise pas directement son attribut "carnet" car il dispose d'une grille dans l'interface graphique pour cocher ses déductions au fil de la partie. Par exemple, s'il

apprend lors d'un tour que le joueur 2 possède la carte "REVOLVER", alors il coche la case qui se situe sur la ligne "Revolver" et la colonne "j2".



Le carnet de type Carnet est utile pour les joueurs ordinateurs (bots) qui héritent de la classe Joueur : ainsi, nous avons créé les classes **BotNaif** et **BotExpert**. Nous avons cependant concentré nos efforts sur la classe BotNaif. Dans cette classe, nous avons défini de nombreuses méthodes au travers desquelles le bot met en œuvre sa stratégie. Par exemple, le bot naïf formule des hypothèses en choisissant trois cartes au hasard. En ce qui concerne la stratégie de déplacement, les ordinateurs choisissent de se déplacer vers la salle la plus proche et qui n'a pas été visitée dernièrement. Une fois que l'ordinateur a choisi sa destination, il va emprunter le chemin le plus court pour s'y rendre (c'est-à -dire le chemin qui comporte le moins de case à emprunter). Pour réaliser ce déplacement, nous avons implémenter et adapter à notre plateau de jeu une version simple de l'algorithme du plus court chemin sur plateau (A* sans heuristique). Le plateau étant de petite taille, les temps de calculs n'avaient pas à être optimisés.

Le déroulement d'une partie est géré par la classe **Arbitre**. Même si le jeu de base du Cluedo ne possède pas d'arbitre, cette classe nous a semblé nécessaire pour assurer le passage d'un joueur à l'autre, la communication des hypothèses d'un joueur aux autres, le dévoilement des cartes, etc.

Pour ce qui est de l'implémentation du plateau, nous avons créé la classe **Plateau** qui définit une matrice de **Case.** En réalité, le plateau est composé de salles, de couloirs, de murs et de portes, tous sont des cases mais avec des fonctions qui diffèrent. Nous avons donc créé les classes **Salle, Couloir** et **Porte** qui héritent de la classe Case. La classe **Case** est abstraite car elle possède une méthode abstraite : *ajouterJoueur,* en effet la méthode utilisée

pour affecter un joueur à une case de type porte ou couloir n'est pas la même que pour une case de type salle. Nous avons fait le choix de rendre les salles accessibles pour le joueur uniquement via les portes, puisqu'il faut de toute façon passer par une porte et qu'une fois dans une salle, il n'y aucun déplacement à faire, ni positionnement particulier à choisir (une salle se comporte comme une case de grande taille sur laquelle plusieurs joueurs peuvent être présents). Vu qu'il existe différentes salles et que chacune se compose de plusieurs cases, il était logique de choisir une structure de type HashMap où la clé correspond à la salle (de type énumération) et la donnée est une liste de cases (ArrayList<Case>). Les murs sont non empruntables, regroupés en bloc et sont considérés comme des salles dont la clé correspond à NULL.

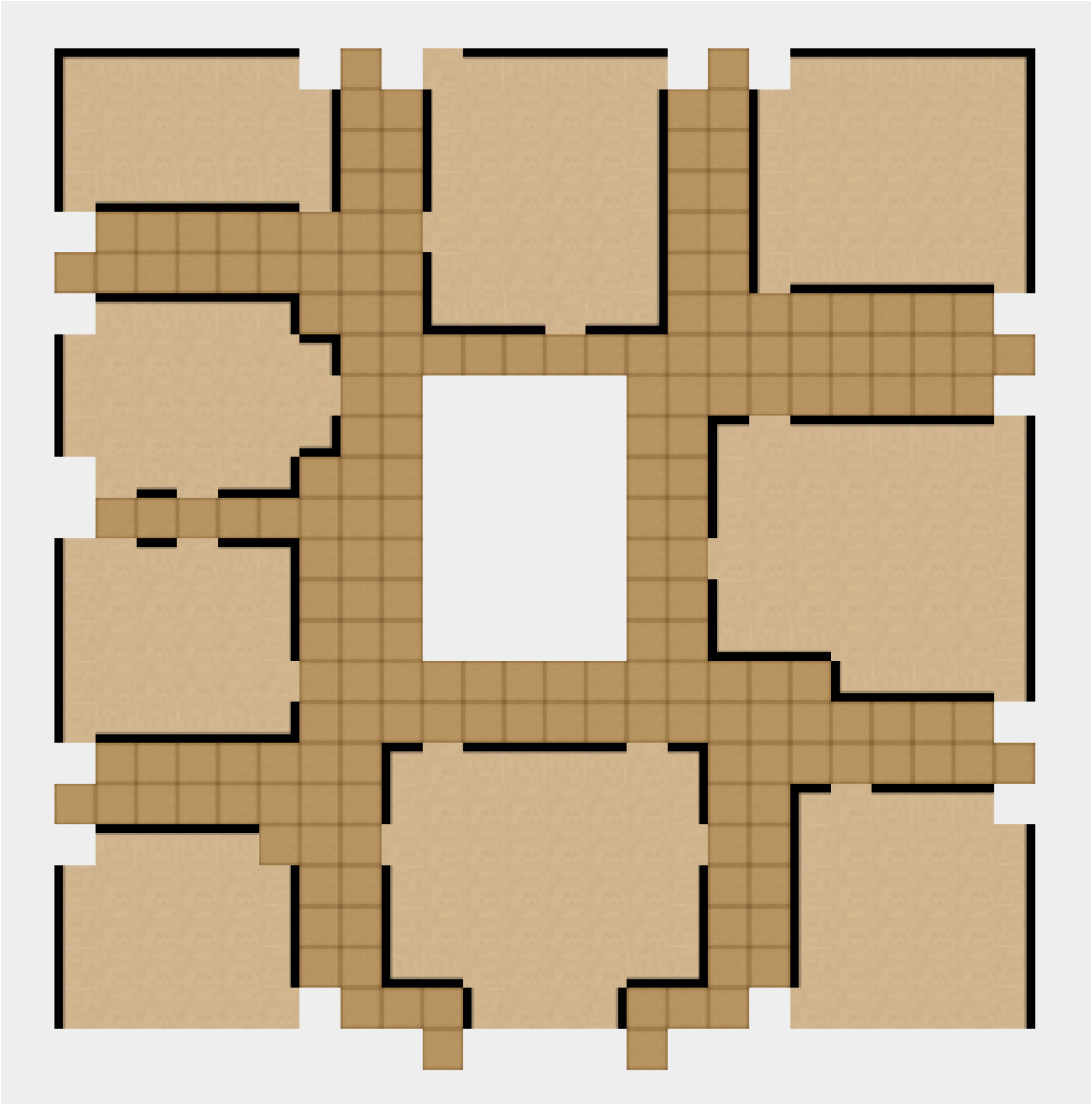
Pour finir, nous avons réalisé toutes nos interfaces graphiques avec **Java Swing**. En particulier, nos fenêtres sont de type **JFrame**. Le **GUI** est une fenêtre de type **JFrame** qui est composé d'un BorderLayout qui est lui même en trois parties:

- au centre, le plateau de jeu (**BoardPanel** qui hérite de **JPanel**),
- à gauche, la fenêtre d'action avec les dés, les boutons hypothèse et accusation (**JButton**)
- à droite, une fenêtre avec différents onglets: le carnet, les cartes et la console.
  Pour pouvoir naviguer entre les onglets, nous avons choisi le type **JTabbedPane** pour la fenêtre (voir figure plus haut). Ensuite, étant donné que le carnet du cluedo est relativement grand, il a fallu pouvoir scroller dans cette même fenêtre d'où le choix du **JScrollPane**. Ce dernier contiendra le carnet de type **GrilleCluedo.** Quant aux onglets Cartes et Console, ils sont de type **JTextArea**.

L'affichage du plateau est basé sur un fichier texte qui contient les 600 cases du plateau et leur caractéristique: un mur, une salle, un couloir etc. Son interface graphique est en réalité une matrice de **BufferedImage** de taille 25x24. Chaque case de la matrice de type **BufferedImage** représente une case du plateau. La méthode qui crée la matrice est contenue dans la classe **Plateau**. Cette matrice est parcourue en parallèle du fichier texte, ligne par ligne, pour faire correspondre la case fichier et celle de la matrice. Lors du parcours de la matrice, le choix de l'image se fait selon la lettre du fichier texte, les cases autour de la case actuelle mais également de la présence ou non d'un joueur. En effet, les cases correspondant aux murs délimitant une salle ne sont pas différenciées des cases au centre de la salle. L'orientation des murs se fait alors en fonction des cases aux alentours (couloir, murs, etc). Une variable de type StringBuilder permet de construire, au fur et à mesure, le chemin d'accès vers l'image selon les caractéristiques de la case actuelle. Etant donné que les conditions ne permettaient pas de placer correctement tous les murs, certains ont dû être placés manuellement.

De plus, chaque **BufferedImage** de la matrice représentant chaque case du plateau peut se voir superposer l'image d'un pion du joueur. Cela est géré par une méthode nommée superposerImage() qui prend en arguments les deux images et renvoie une image qui, comme le nom l'indique, correspondra à la superposition de l'image du pion sur la case.

La méthode creerPlateau() est utilisée dans **BoardPanel** pour instancier le plateau et son affichage. **BoardPanel** est par la suite lui-même instancié dans le **GUI**.



Problèmes rencontrés et solutions apportées

Tout d'abord, nous avons eu du mal à mettre en commun tous les codes réalisés par les différents membres du groupe. Nous avons dû parfois modifier les codes de certaines classes par incompatibilité.

Un problème important est que le travail au sein de l'équipe a été mal réparti : l'investissement et la motivation d'un membre à l'autre étaient très variés. Cela a quelque peu freiné l'avancement du projet qui aurait sûrement pu être plus abouti si chacun s'était investi suffisamment.

De plus, certaines notions de Java n'ont pas été suffisamment voire pas du tout abordées en cours : c'est le cas du traitement des fichiers audios et de certains éléments de Java Swing (les éléments de type ImageIcon par exemple).

Enfin, nous avons eu quelques conflits au niveau de svn et d'Eclipse et travailler sur machine virtuelle n'était pas idéal, donc nous avons installé TortoiseSVN et VScode sur nos ordinateurs personnels pour y remédier.

Diagrammes de classes

Diagramme de classes concernant l'implémentation du plateau :

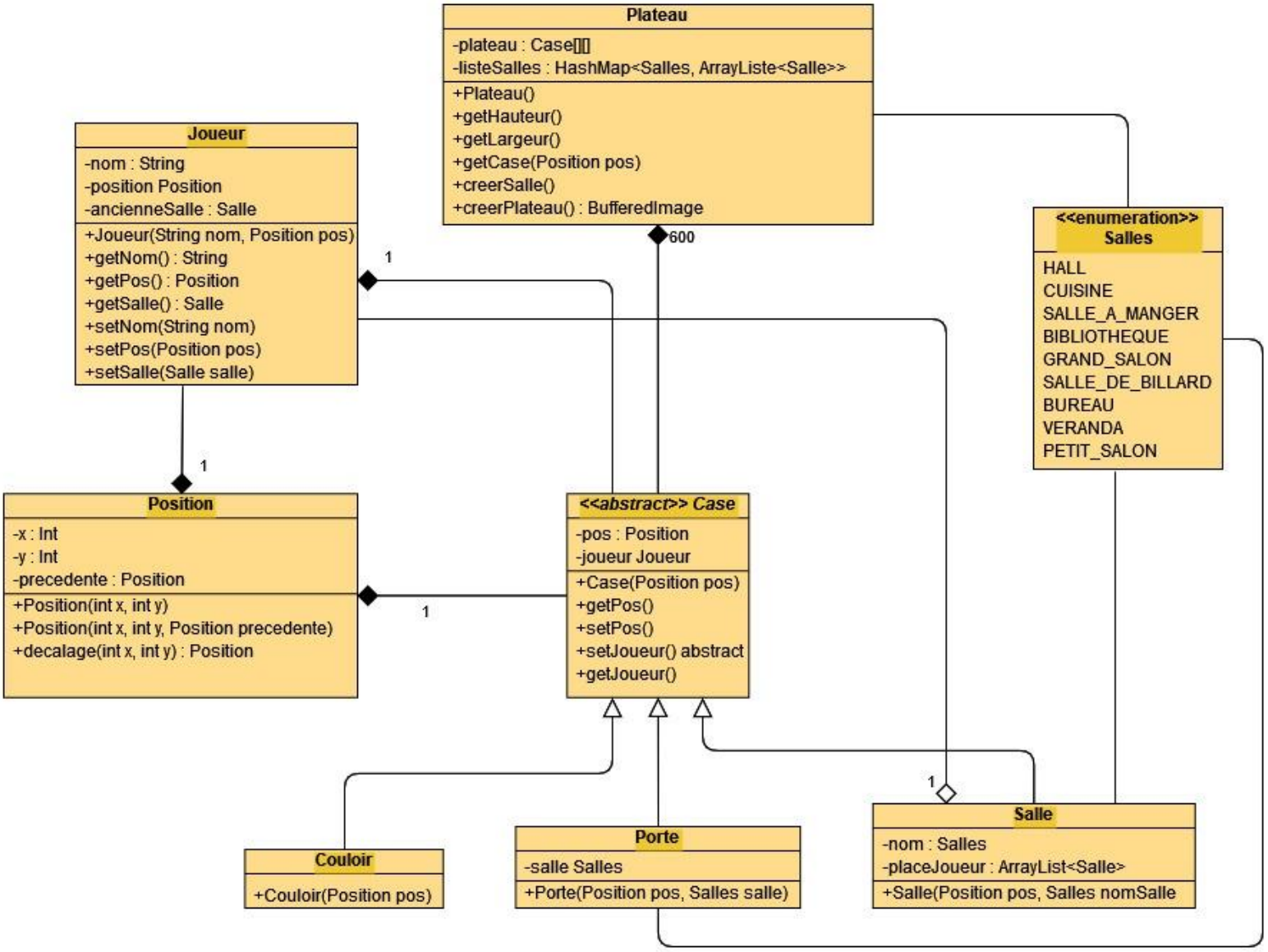


Diagramme de classes concernant la classe Joueur :

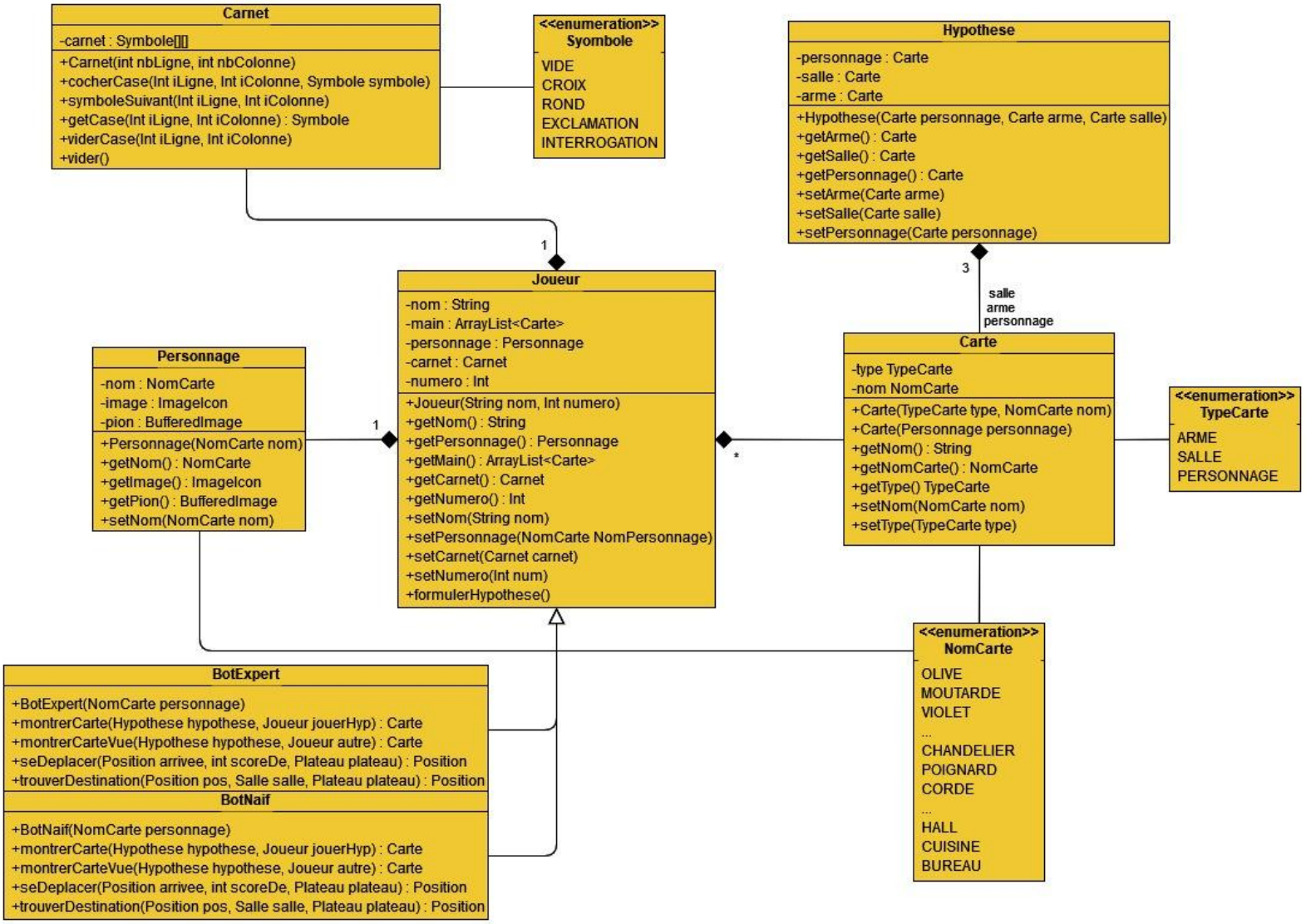


Diagramme de classes concernant l'affichage graphique :

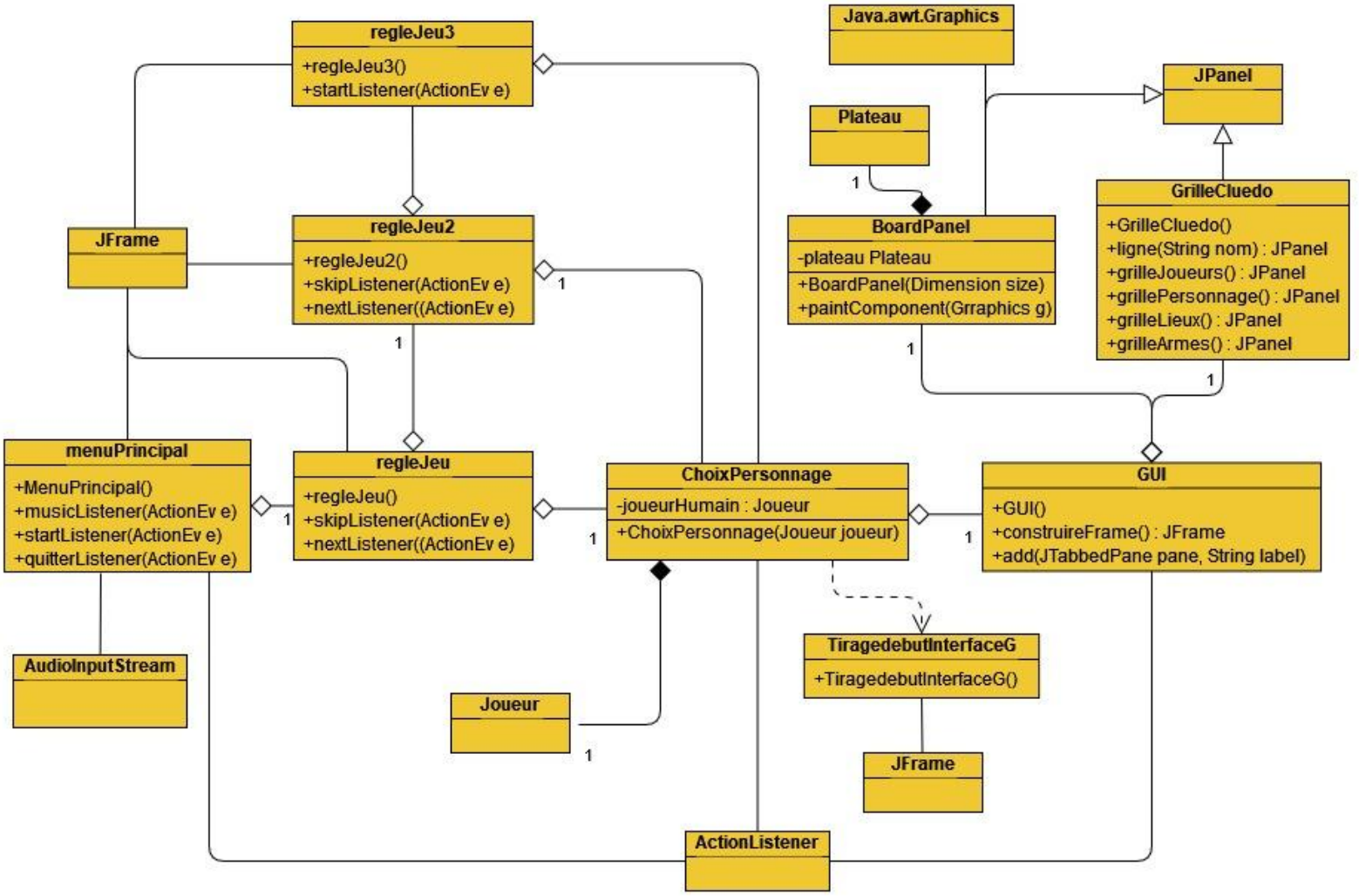
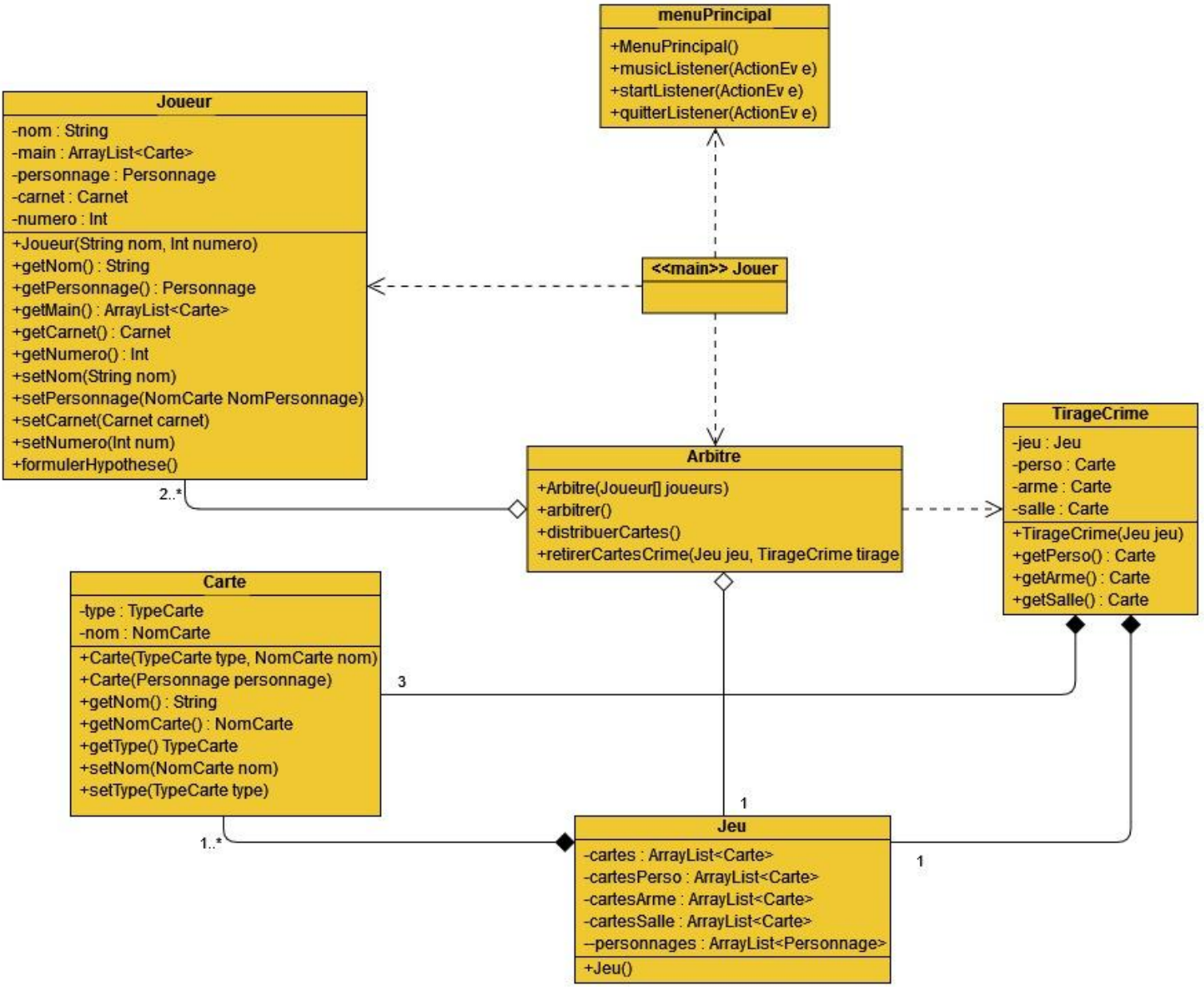


Diagramme de classes pour la mise en place du jeu :



Organisation de l’équipe

Nous nous sommes réparti les tâches très tôt dans le projet, lors de la première réunion, à partir du backlog nous avons choisi ensemble l'architecture globale du code. Puis nous nous sommes répartis les tâches correspondant à des cases du backlog. Nous nous sommes peut-être répartis les tâches trop tôt et n’étions pas vraiment au courant des avancés des uns des autres cependant, nous faisions des réunions une fois par semaine afin de tenter d’avancer en groupe.

Mise en oeuvre des méthodes agiles

Charte produit

Dans un premier temps, nous avons créer la charte produit.

Vision produit

Pour les personnes à partir de 8 ans **qui veulent** se divertir. **Notre produit est** un programme de jeu d'enquête **qui** permet de jouer à un jeu de type Cluedo. **À la différence des** autres jeux de type Cluedo, celui-ci **permet de** jouer à plusieurs ou seul avec des ordinateurs proposant différentes difficultés/stratégies.

Critères de succès (métriques)

- Jeu fonctionnel
- Utilisateurs satisfaits

Clients / Utilisateurs

- Cible marketing : personnes de plus de 8 ans
- Catégorie professionnelle : toutes

Risques

- Produit non terminé ou non fonctionnel (bogué)

Valeur métier

1. Choisir la stratégie de l'ordinateur
2. Jouer seul
3. Avoir un historique de ses parties
4. Thème du plateau de jeu en DLC

Performances

- Répondre en un temps raisonnable

Jalons : dates ou fonctionnalités

- 9/04 : rendu de l'itération 1
- 23/04 : rendu de l'itération 2
- 21/05 : rendu de l'itération 3
- 31/05 : oral de projet

<< Tradeoff matrix>>

	Fixé	Ferme	Flexible
Périmètre		X	
Délai	X		
Coût			X
Qualité		X	

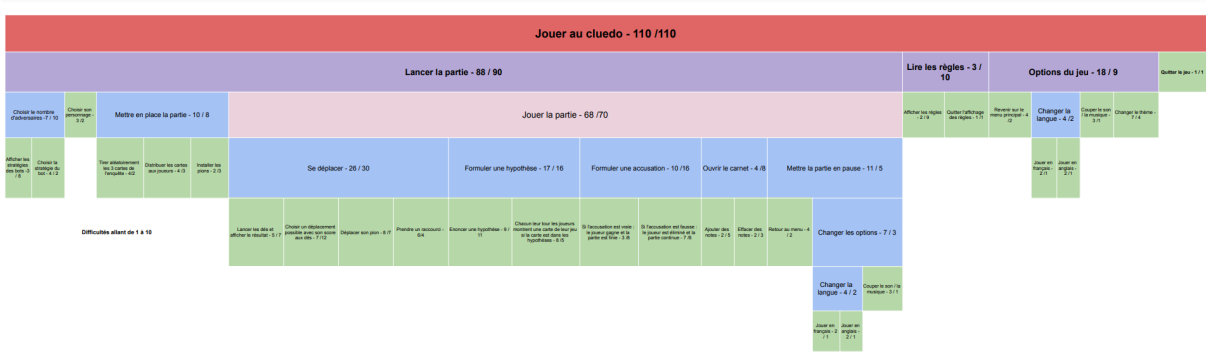
Fonctionnalités

Puis nous nous sommes mis d'accord sur les fonctionnalités de l'application.

- **Lancer une partie**
- **Options**
- **Lire les règles (sur le menu et pendant le jeu)**
- **Couper le son/musique de fond**
- **Choisir son personnage**
- Distribuer les cartes du début
- Accès à une grille de jeu
- Lancer le dé
- Se déplacer sur le plateau en fonction du nombre sur le dé
- **Mettre en place le jeu avec les 3 cartes du début**
- Formuler une hypothèse
- Formuler une accusation pendant son tour

- **Choisir la stratégie des bots**
- Montrer une carte, le joueur a le choix s'il en possède plusieurs qui sont présentes dans l'hypothèse formulée par un autre joueur
- **Choisir le thème du jeu (exemple thème Pirate, chats)**
- Remplir la grille de jeu (différents styles)
- Effacer la grille
- **Sélectionner le nombre d'ordinateurs**
- **Choix du déplacement**
- **Changer la langue**
- **Possibilité de prendre un raccourci**
- **Mettre la partie en pause**
- Reprendre la partie après une pause (trop dur ?)
- Quitter la partie
- **Quitter le jeu**

Nous avons ensuite créer le backlog de l'application



Backlog également disponible sous le nom backlog.pdf dans livrables

Nous avions 110 points à répartir pour finir le projet. La vélocité sur les 6 semaines d'itération devait être d'environ 20 points par semaine ou environ 3 points par semaine et par personne. Cette vélocité a été très hétérogène, certaines personnes ont produit beaucoup plus de points que d'autres.