Institut National des Sciences Appliquées de Rennes
Département Informatique
4ème année

Projet Canon Noir

Rapport de conception

Maxime HAVEZ
Gareth THIVEUX
4INFO - G2.2

2010-2011

Sommaire

I. Introduction

Ce rapport a pour objectif de détailler notre analyse conceptuelle pour notre projet de réalisation d'un logiciel. Le logiciel en question correspond à l'adaptation d'un jeu de société : le jeu du Canon Noir.

Ce jeu de société, constitué d'un plateau, d'îles, de trésors, d'un canon (noir) et de bateaux, peut accueillir de deux à quatre joueurs. Le vainqueur est le joueur qui ramènera le premier trois trésors à son port. Bien entendu, les trésors doivent être ramassés sur des cases spécifiques, et pour pimenter le jeu, il existe tout un mécanisme de tirs et duels au canon afin de tenter de couler ou ralentir la progression de son/ses adversaire(s).

La réalisation et la réussite de ce projet passe avant tout par la mise en œuvre des connaissances acquises en cours de Programmation et Modélisation Orientées Objet et de Design Patterns.

Nous allons dans un premier temps, et notamment par le biais de ce rapport de conception, aborder et réutiliser les outils UML qui nous ont été présentés lors de notre cursus. Ceux-ci nous ont permis de retranscrire les fonctionnalités de notre application ainsi que sa mise en œuvre et son fonctionnement interne. Ce modèle conceptuel est donc une représentation abstraite de notre système, et il permettra par la suite de faciliter l'étude ainsi que notre communication.

Nous détaillerons notre démarche, qui évidemment part d'une analyse globale du logiciel, pour qu'ensuite nous puissions approfondir les différentes étapes. Nous commencerons donc par vous présenter les fonctionnalités du jeu à l'aide de cas d'utilisation, nous représenterons ensuite la modélisation globale du jeu et ses différents modules grâce à des diagrammes de classe, et mettrons en valeur les Design Patterns associés. Dans un troisième temps nous modéliserons le fonctionnement des différents modules par le biais de diagrammes d'activités et d'états-transitions. Enfin, nous exposerons le détail de différents scénarios choisis avec des diagrammes d'interactions.

II. Fonctionnalités du jeu

Plutôt que de lister les règles du jeu Canon Noir, qui sont somme toutes nombreuses et relativement complexes, nous allons plus simplement illustrer les fonctionnalités du jeu à l'aide de cas d'utilisation.

Les diagrammes de cas d'utilisation font partis de la catégorie des diagrammes comportementaux. Simples et efficaces, les cas d'utilisation doivent nous permettre de retranscrire les principales fonctionnalités (pour l'utilisateur) de notre logiciel, mais également de montrer ses limites.

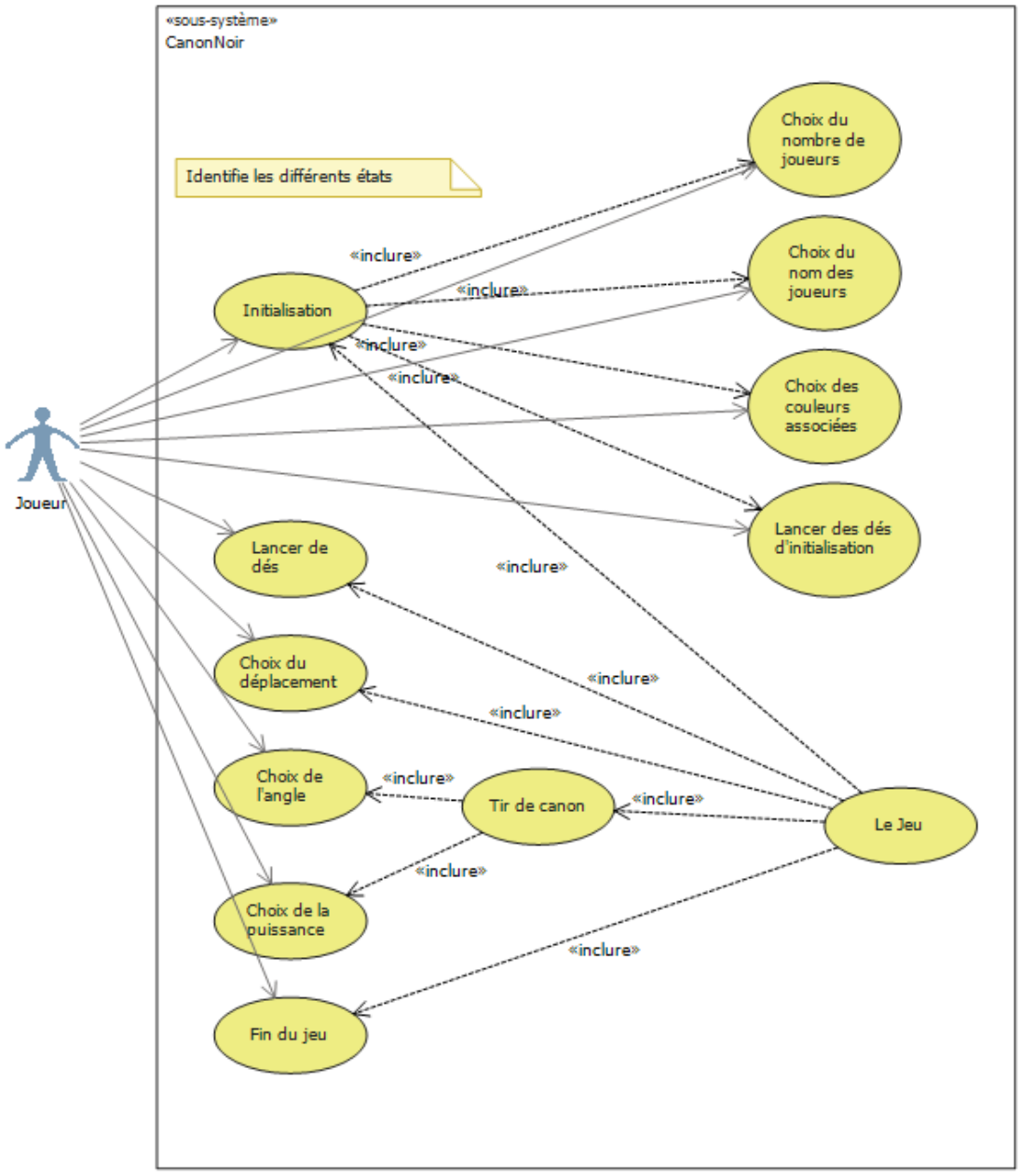


Figure 1 - Diagramme de cas d'utilisation de l'application

III. Modélisation du jeu

Nous réalisons la modélisation du jeu par le biais des diagrammes de classe. Ce sont les diagrammes les plus connus et les plus couramment utilisés, et font partis des diagrammes structurels. Ils retranscrivent par conséquent la structure statique de notre application.

A. La modélisation globale du jeu

Notre application comporte deux parties distinctes qui sont :

- *Le moteur*
- *L'interface graphique*

Le moteur, comme son nom l'indique, permet de gérer les évènements du jeu, alors que l'interface permet quant à elle de gérer l'affichage de notre application ainsi que d'assurer l'interface avec l'utilisateur (prise en compte des actions de l'utilisateur).

Remarque : cette séparation, couramment utilisée, laisse la possibilité à un développeur d'utiliser notre moteur avec un affichage différent.

Afin de pouvoir utiliser notre moteur écrit en C++ par le biais d'une interface de type WPF, il a fallu mettre en place un adaptateur. C'est un *wrapper* qui va assurer la compatibilité des langages.

D'autre part, en se penchant sur les règles du Canon Noir, nous avons remarqué la nécessité de gérer différents états. Il semblait donc judicieux d'utiliser le design pattern adéquat. Notre moteur possède donc un état courant qui évolue en fonction des événements.

Le schéma ci-dessous illustre la vision globale de notre application :

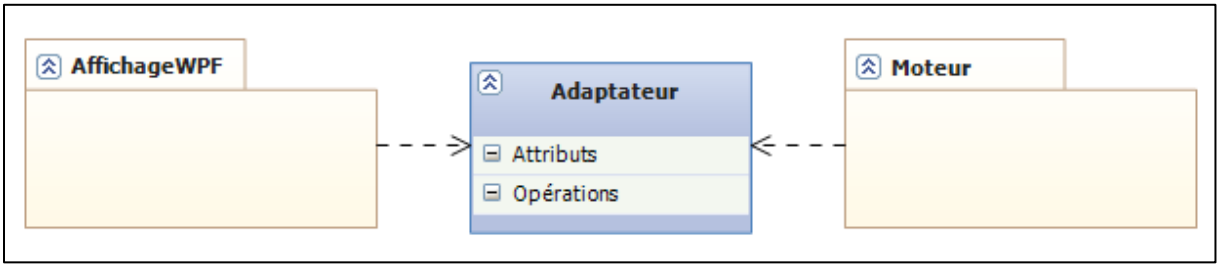


Figure 2 - Vision globale de l'application

B. Les différents modules du jeu

1. Moteur

Le moteur se divise en plusieurs parties distinctes que nous allons vous présenter :

- La Façade
- Le MoteurJeu qui possède :
  - Un état courant
  - Des joueurs
  - Un plateau (constitué de cases)
  - Deux dés

Le diagramme de classes correspondant à la figure 3 est celui de la structure de notre moteur. Comme vous pouvez le constater, c'est lui qui gère les éléments du jeu tels que le plateau, les joueurs et les dés.

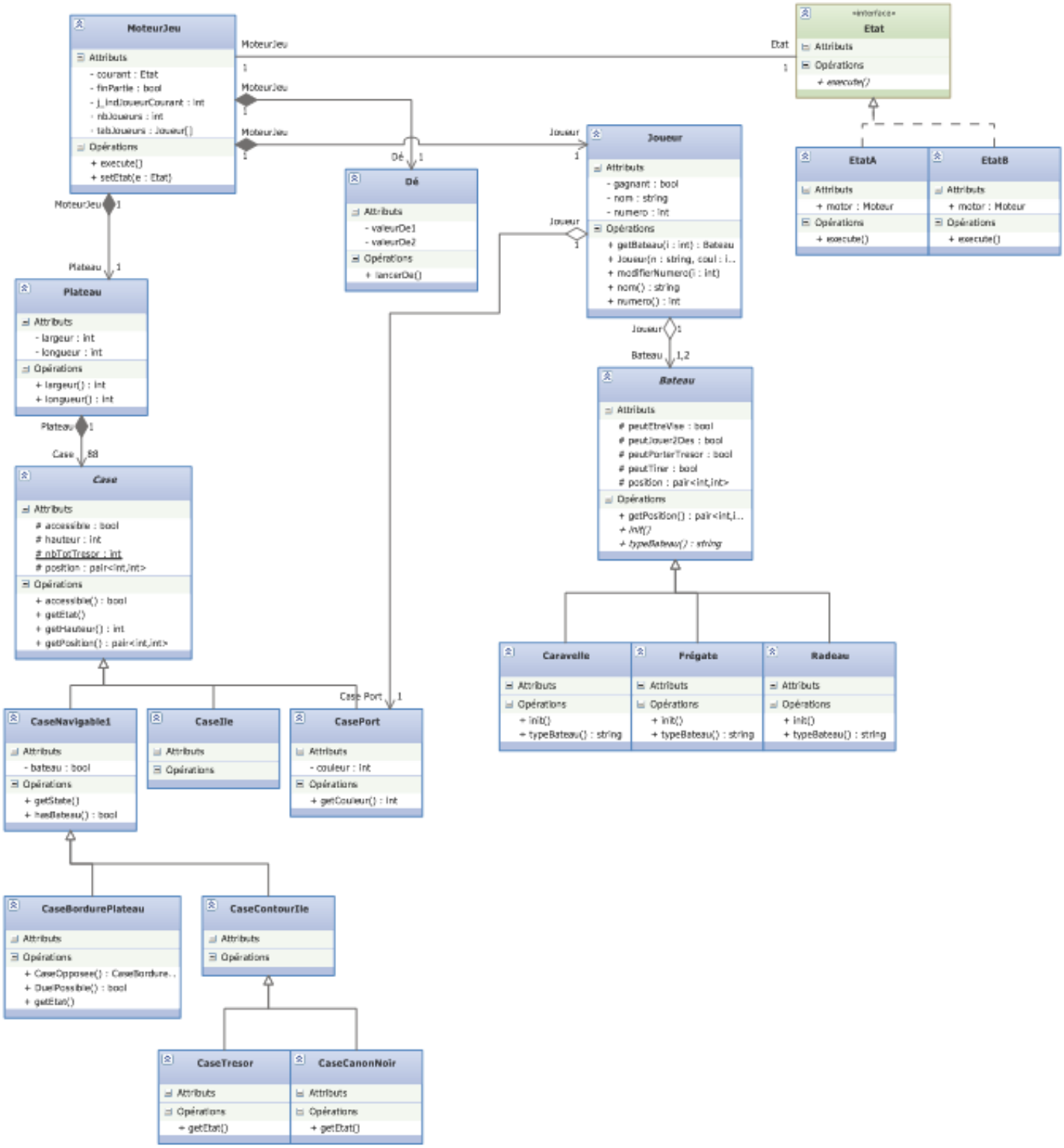


Figure 3 - Diagramme de classes concernant le Moteur

Nous allons maintenant détailler le design pattern État que nous utilisons. Il permet, comme son nom l'indique, de gérer les réactions du moteur selon son état. Il comporte ainsi 14 états différents qui correspondent évidemment au déroulement du jeu. La figure 4 présente l'ensemble des états possibles.

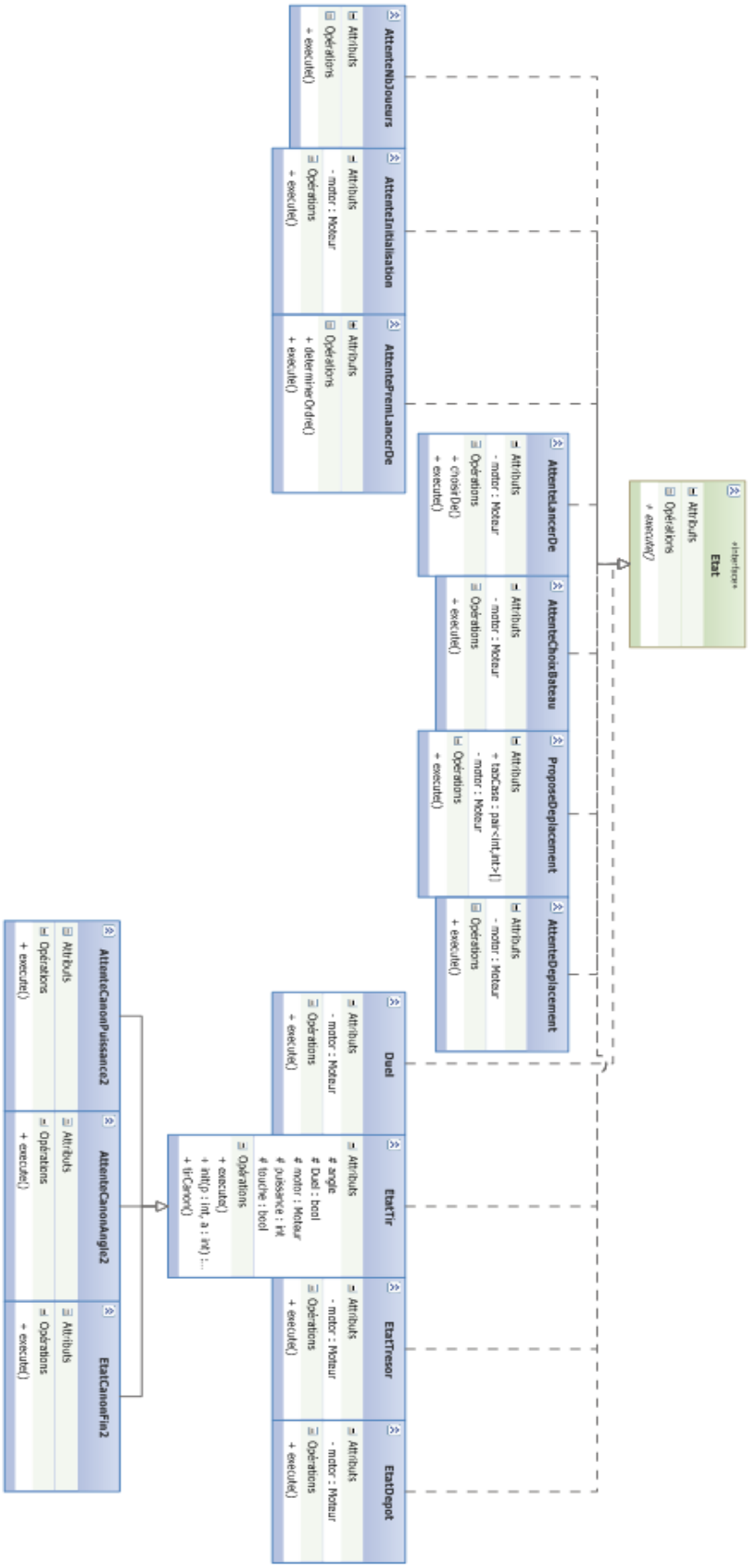


Figure 4 - Diagramme de classes des états du moteur

Pour ce qui est de la gestion des états, comme vous pouvez le remarquer sur le schéma, il en existe différents « types ». En effet, nous avons choisi de distinguer les états activés en se positionnant sur une case particulière (trésor, canon, duel, …) car pour ces états, ce sont les cases elles-mêmes qui doivent changer l'état courant. D'autre part, nous distinguons également les états qui concernent l'initialisation des autres états, puisqu'ils sont gérer dans une autre fenêtre.

Nous souhaitons vous donner succinctement une description des différents états, dont vous pourrez apprécier le rôle par le biais des diagrammes suivants.

Etats d'initialisation :

- **AttenteNbJoueurs :** attend que l'utilisateur clique sur la case correspondante au nombre de joueur qu'il souhaite
- **AttenteInitialisation :** connaissant le nombre de joueurs l'application attend que l'utilisateur rentre les noms et les couleurs des joueurs
- **AttentePremLancerDe :** permet de gérer les lancements successifs des dés qui doivent déterminer l'ordre des joueurs dans la partie

Etats courants :

- **AttenteLancerDe :** attend un clic sur le bouton de lancer de dés
- **AttenteChoixBateau :** attend (dans le cas où il y a deux joueurs) que l'utilisateur choisisse un bateau
- **ProposeDeplacement :** affiche les déplacements possibles
- **AttenteDeplacement :** attend un clic sur un des déplacements proposés

Etats activés par les cases :

- **Duel :** si le joueur clic sur une case bordure du jeu et qu'il y a un adversaire en face lance l'état Tir
- **EtatTir :**
  - **AttenteCanonPuissance :** attend que l'utilisateur indique la puissance de son tir
  - **AttenteCanonAngle :** attend que l'utilisateur indique l'angle qu'il souhaite
  - **EtatCanonFin :** une fois que les paramètres entrés, on attend que l'utilisateur clique sur le bouton OK
- **EtatTresor :** si le joueur clique sur une case trésor
- **EtatDepot :** lorsque le joueur clique sur une case port

2. Gestion de l'affichage

Afin que notre « moteur » puisse être applicable à différents types d'affichage, nous avons mis en place une façade, basée sur le design pattern du même nom. Cette classe agrège une instanciation de MoteurJeu et offre à l'interface plusieurs fonctions. Cette classe comporte donc un nombre important d'attributs et de méthodes qui pourront éventuellement évoluer lors du développement. C'est cette classe, représentée en figure 5, qui regroupe les attributs sur lesquels l'affichage va se baser pour se rafraîchir.

Figure 5 – Zoom sur la classe Façade

Comme nous avons pu vous l'expliquer précédemment, notre code étant en C++, nous utilisons un *wrapper* afin de pouvoir utiliser notre moteur par le biais d'une interface de type WPF. Notre affichage se compose ainsi de trois classes qui représentent la fenêtre principale et les deux fenêtres auxiliaires utilisées lors de l'initialisation de l'application et lorsque le moteur entre dans un état de Tir. Le schéma de la figure 6 représente ce diagramme.

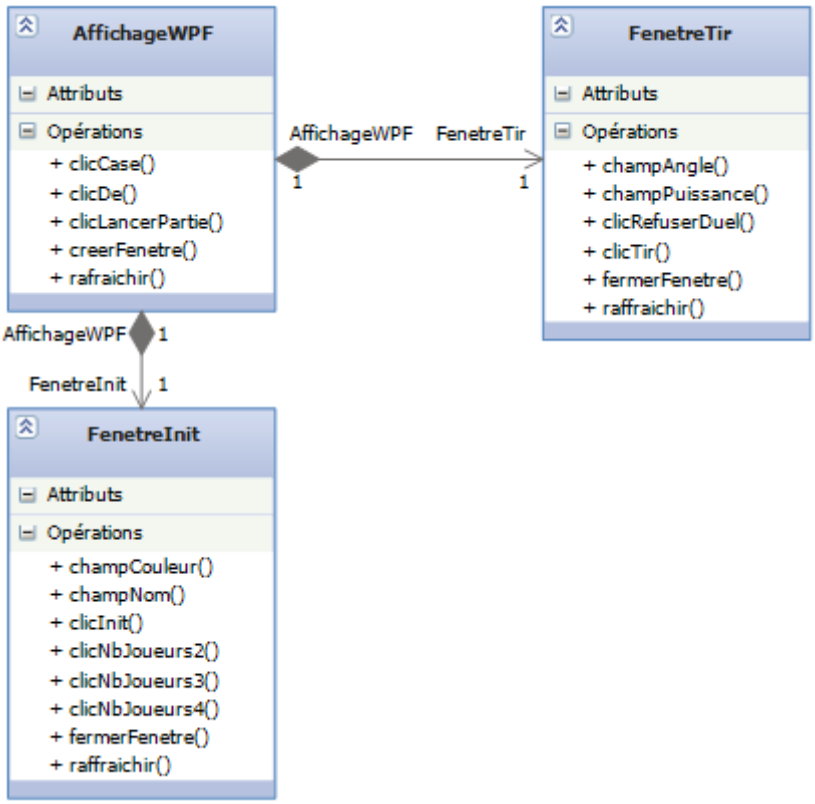


Figure 6 – Zoom sur les classes liées à l'affichage

3. Gestion de la carte

La carte du jeu sera constituée d'un canvas avec une simple image en fond, celle du plateau du jeu Canon Noir.

Chaque clic sur la carte sera capté par l'affichage comme un clic associées à des coordonnées (abscisse, ordonnée) pixels, et donc représentées par une paire d'entiers. Chaque case du jeu englobera donc plusieurs coordonnées pixels. Grâce à cette association coordonnées/case, le moteur pourra alors interpréter un clic, non pas comme des coordonnées pixels, mais à un clic sur une case du jeu. Et selon le type de la case cliquée, il pourra alors effectuer le traitement approprié.

C'est donc vis-à-vis de ce traitement qui doit pouvoir s'adapter à la case considérée qu'il a fallu différencier les types de cases. On distingue alors cinq types de cases : les cases Trésor, les cases Canon Noir, les cases en bordure de plateau, les cases Île et les cases Port.

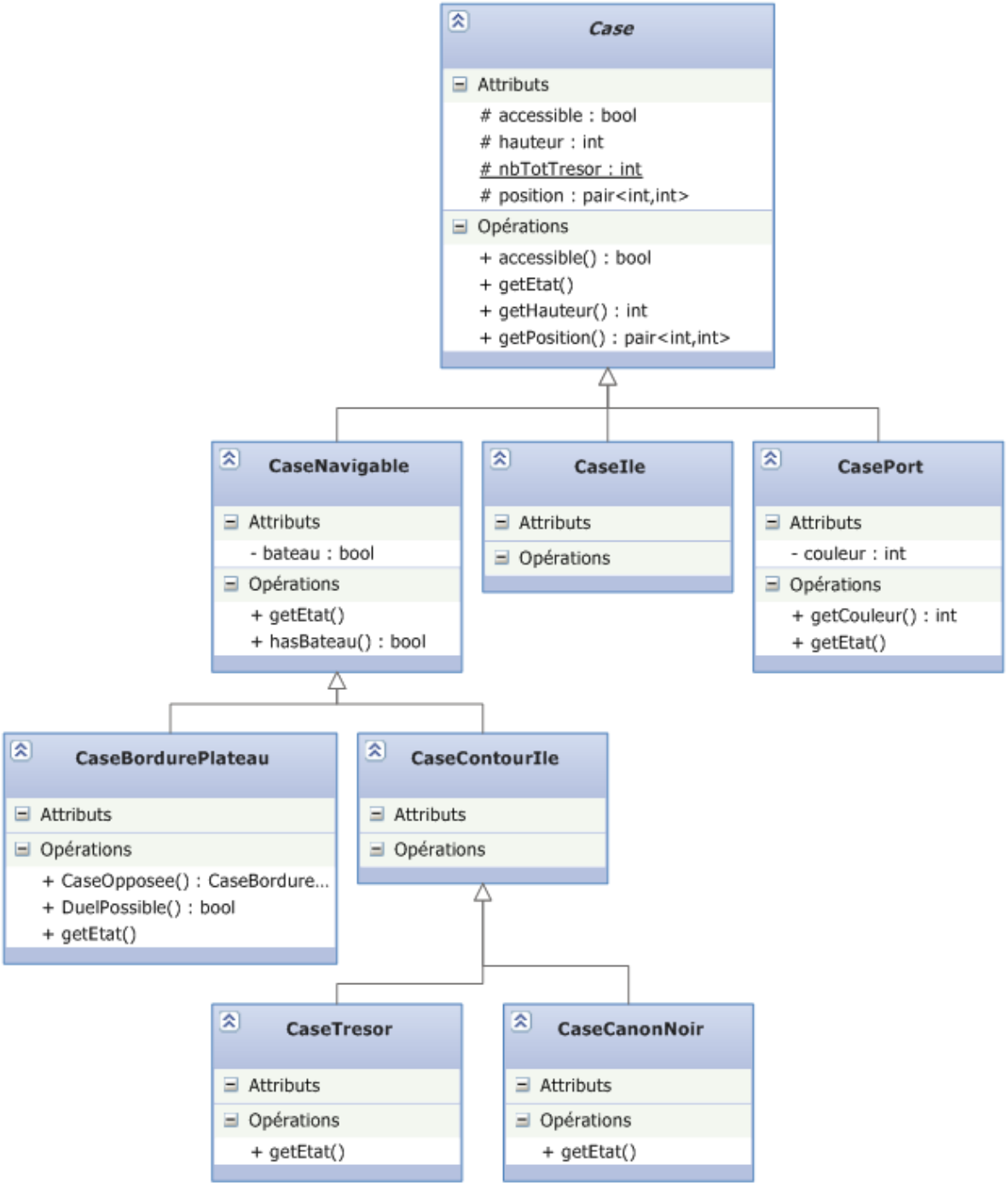


Figure 7 - Zoom sur les types de cases

Les classes CaseTresor, CaseCanonNoir héritent elles-mêmes d'une classe CaseContourIle. En effet, au-delà du fait d'avoir remarqué que ces cases ne pouvaient se situer que sur les bordures des îles, cette hiérarchie permet d'envisager d'implémenter un positionnement de ces cases d'une partie sur l'autre, sous la contrainte de rester sur le contour de l'île, d'où l'intérêt de ce découpage.

Les classes CaseContourIle et CaseBordurePlateau héritent elles-mêmes d'une classe CaseNavigable, en ce sens que ce sont des cases sur lesquels les bateaux peuvent se déplacer, aussi bien les traverser que s'y arrêter. Contrairement aux cases Canon Noir, Trésor et Port pour lesquelles le traitement particulier semble évident (respectivement, pouvoir tirer au canon, pouvoir récupérer un trésor, et enfin pouvoir déposer un trésor et/ou changer de bateau), on peut s'interroger sur l'intérêt de distinguer les cases en bordure de plateau du reste des autres cases « maritimes ». La classe CaseBordurePlateau permet donc de distinguer les cases desquelles il est possible d'engager un duel, pour peu que la condition d'un alignement de la case avec celle d'un bateau adverse soit respectée.

IV. Fonctionnement des différents modules

Afin d'illustrer au mieux le fonctionnement des différents modules du jeu, ainsi que leurs interactions, est de s'appuyer sur des diagrammes dits comportementaux. Après en avoir vu une première forme, à travers un diagramme de cas d'utilisation, nous allons étudier le jeu au travers de diagrammes d'activités et d'un diagramme d'états-transitions.

A. Diagramme d'activités

Les diagrammes d'activités, comme leur nom l'indique, se concentrent sur l'activité d'un processus.

Le diagramme d'activités suivant insiste donc sur les interactions lors de la face d'initialisation du jeu (ouverture de la fenêtre d'initialisation, choix du nombre de joueurs, choix de leur noms et de leur couleurs, et enfin premier lancer de dés afin de déterminer l'ordre de jeu des joueurs).

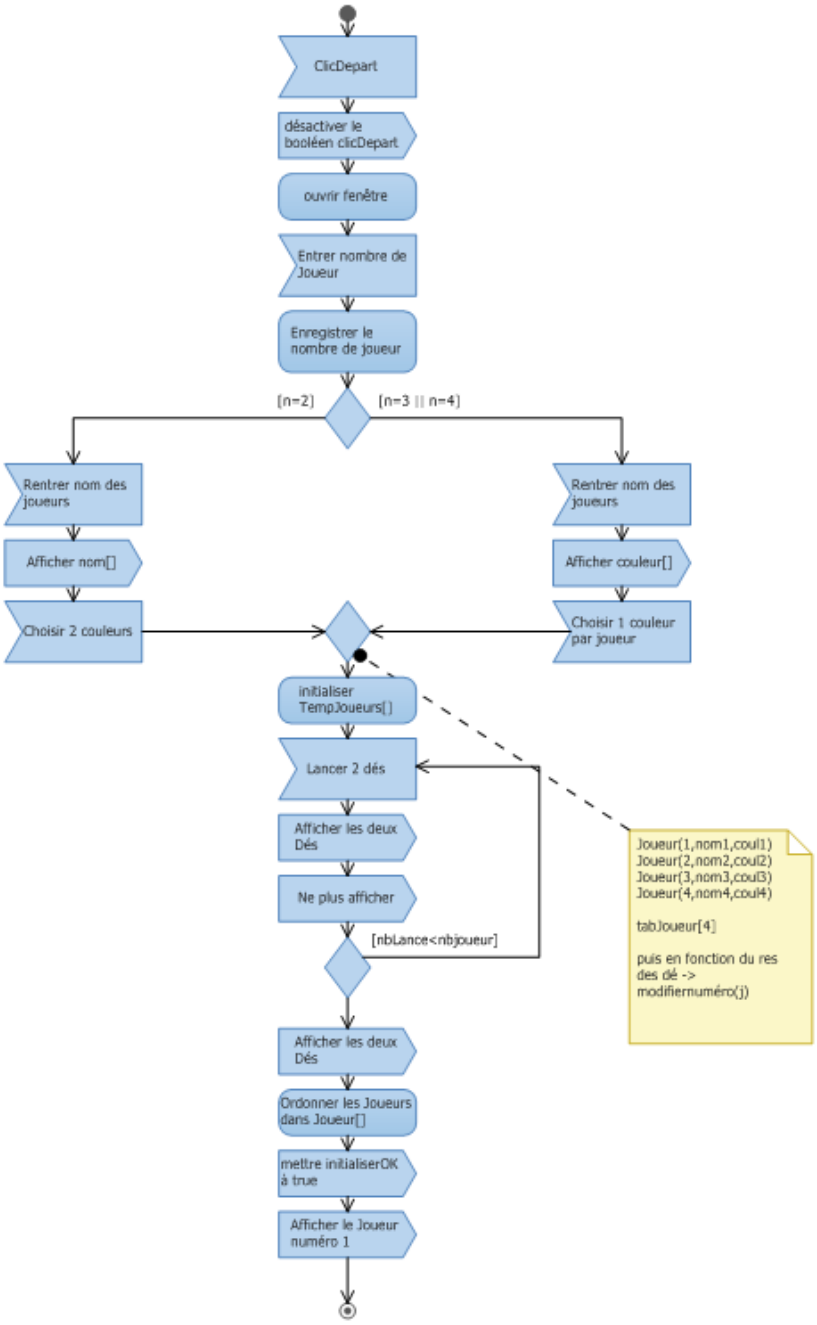


Figure 8 - Diagramme d'activité de l'initialisation

Ce deuxième diagramme d'activités représente une phase de jeu de déplacement, quoi de plus classique, avec les possibilités qui découle de la case d'arrivée du bateau.

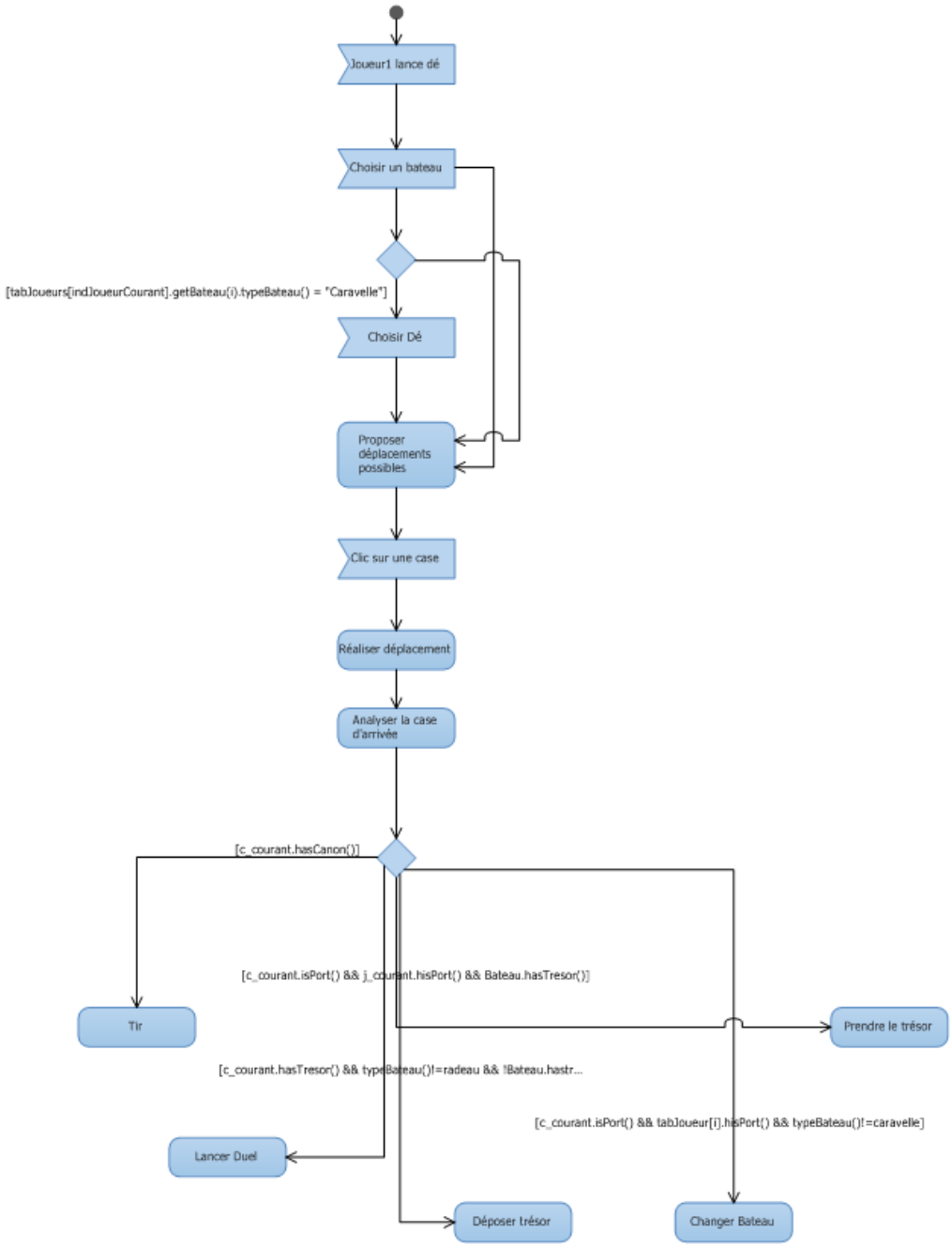


Figure 9 - Diagramme d'activité d'une phase de déplacement

B. Diagramme d'états-transitions

Les objets interagissent pour implémenter des comportements qui peuvent être décrit par des diagrammes d'états-transitions, qui sont pour leur part d'avantage centrés sur l'objet.

Nous allons donc exposer ci-après le diagramme d'états-transitions des Etats du moteur, justement, ainsi que les transitions associées au passage d'un état à un autre.

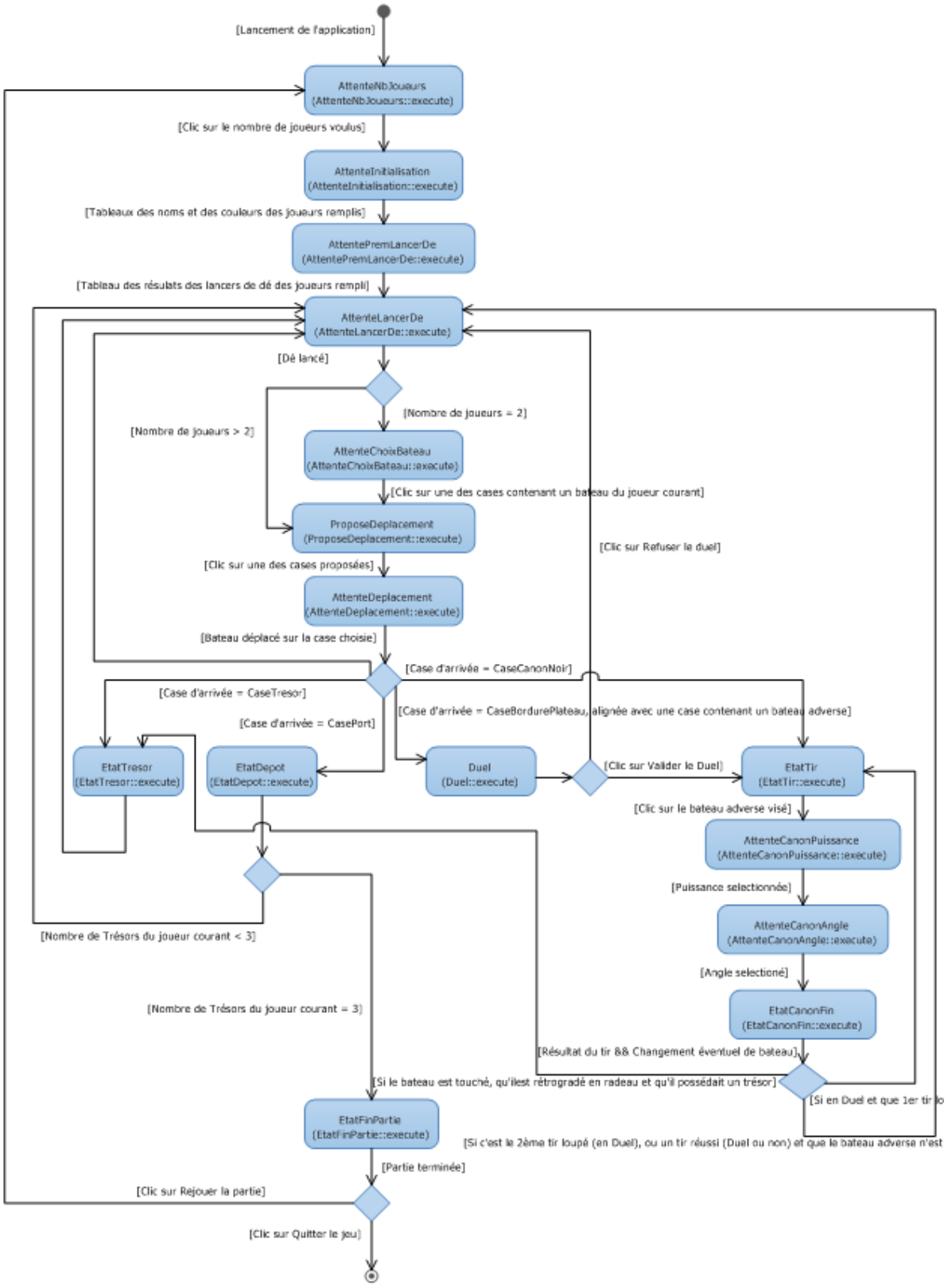


Figure 10 - Diagramme d'état-transitions

V. Différents scénarios de jeu

Nous avons choisis de vous présenter deux scénarios, un premier qui correspond à l'ouverture du programme (initialisation du jeu) et le second à l'étape du lancement des dés jusqu'au choix d'une case par le premier joueur. Dans les deux cas, nous considérons que nous sommes dans le cas où nous avons deux joueurs (cas particulier plus difficile à gérer).

Nous représentons ces deux scénarios à partir de diagramme d'interactions. En effet, ils illustrent les réalisations des cas d'utilisation, en décrivant un exemple de scénario. Bien qu'il existe plusieurs types de diagramme d'interactions, nous ne représenterons ici que des diagrammes de séquences. Ils correspondent à l'échange de messages entre les différents objets de notre diagramme de classe.

Remarque : Pour le scénario correspondant à l'utilisation, nous avons également étudié la possibilité d'un scénario alternatif et celle d'un scénario d'erreur.

A. Scénarios nominaux

Le scénario nominal de l'initialisation, qui correspond au démarrage du jeu, et à tous les paramétrages qui en découlent, à savoir le choix du nombre de joueurs, leur noms et couleurs associés, et la définition de l'ordre de passage, est représenté par la figure 11 (comme précisé plus tôt, le scénario ne s'arrête qu'après le premier lancer de dé, en vue d'un déplacement, du joueur défini comme premier.

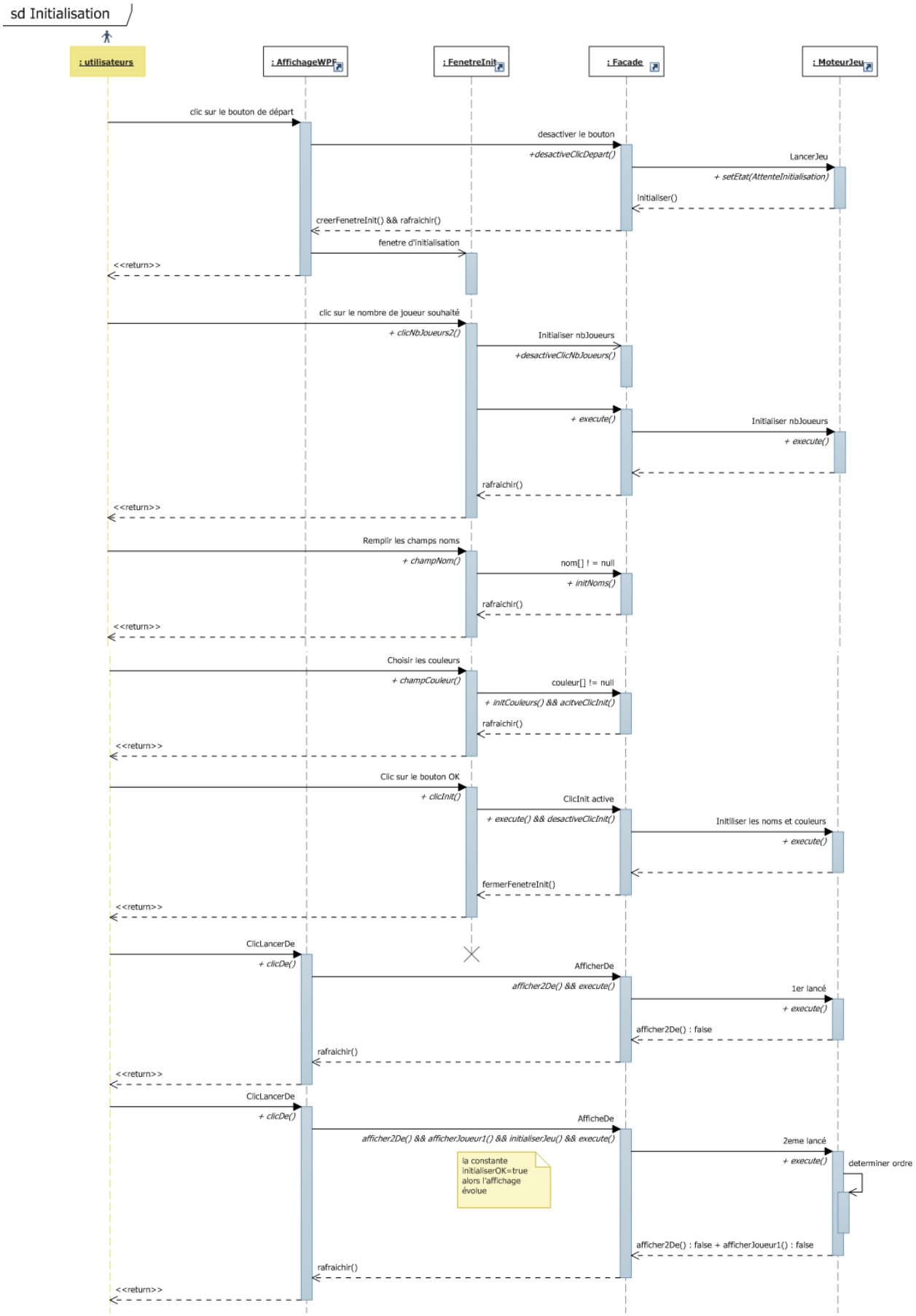


Figure 11 - Diagramme de séquences correspondant à l'initialisation

Nous avons également opté pour la présentation un scénario correspondant à une étape classique dans le jeu : le joueur lance le dé puis choisit où il veut aller (On rappelle qu'on est toujours dans le scénario avec deux joueurs). La figure 12 représente ce scénario.

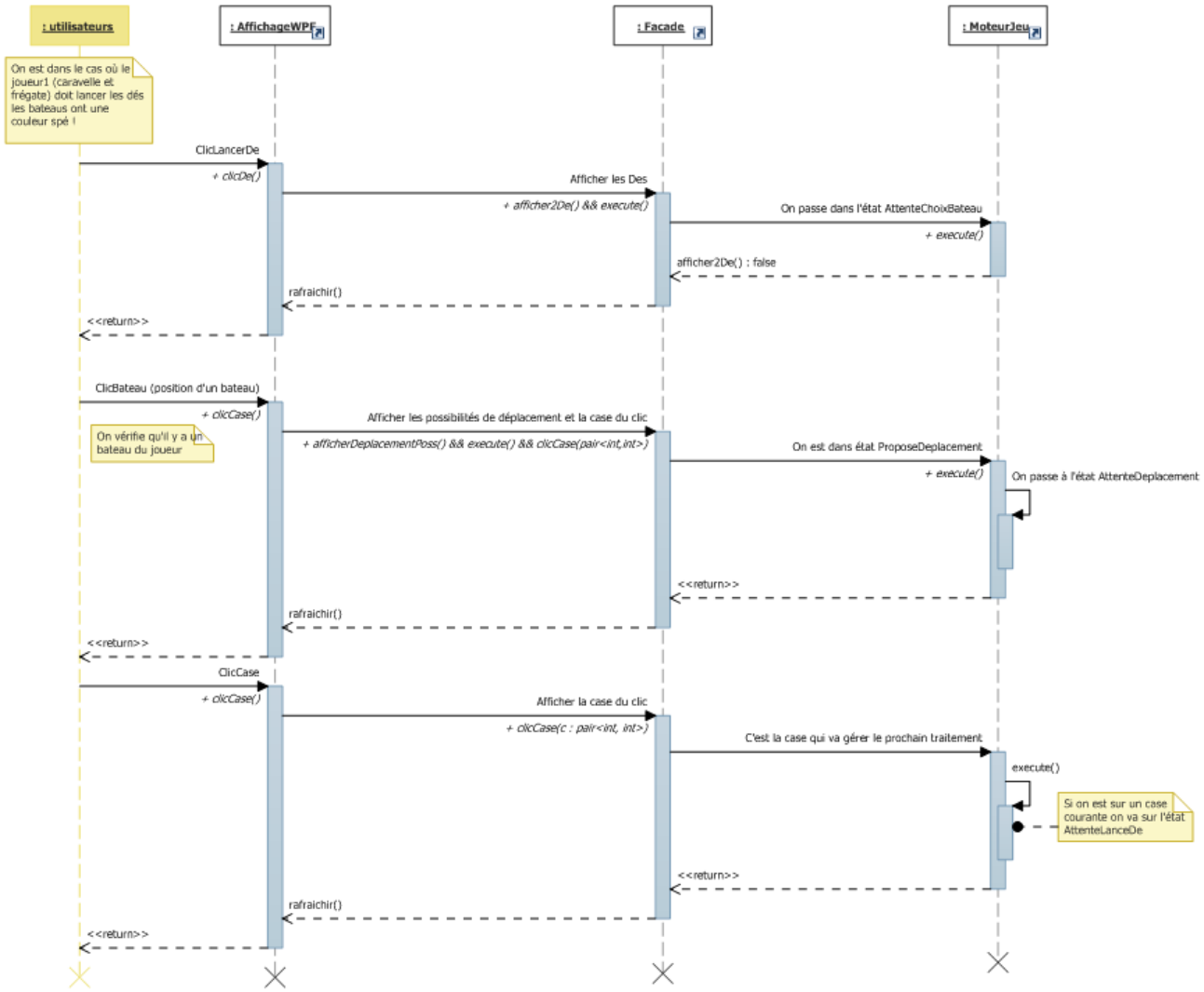


Figure 12 - Diagramme de séquences correspondant au lancement des dés

B. Scénario alternatif

Ci-dessous un scénario alternatif lors de l'initialisation. En effet, alors qu'un clic pour lancer les dés était attendu, lors de la phase d'attribution de l'ordre de passage, c'est un clic sur une case qui est effectué.

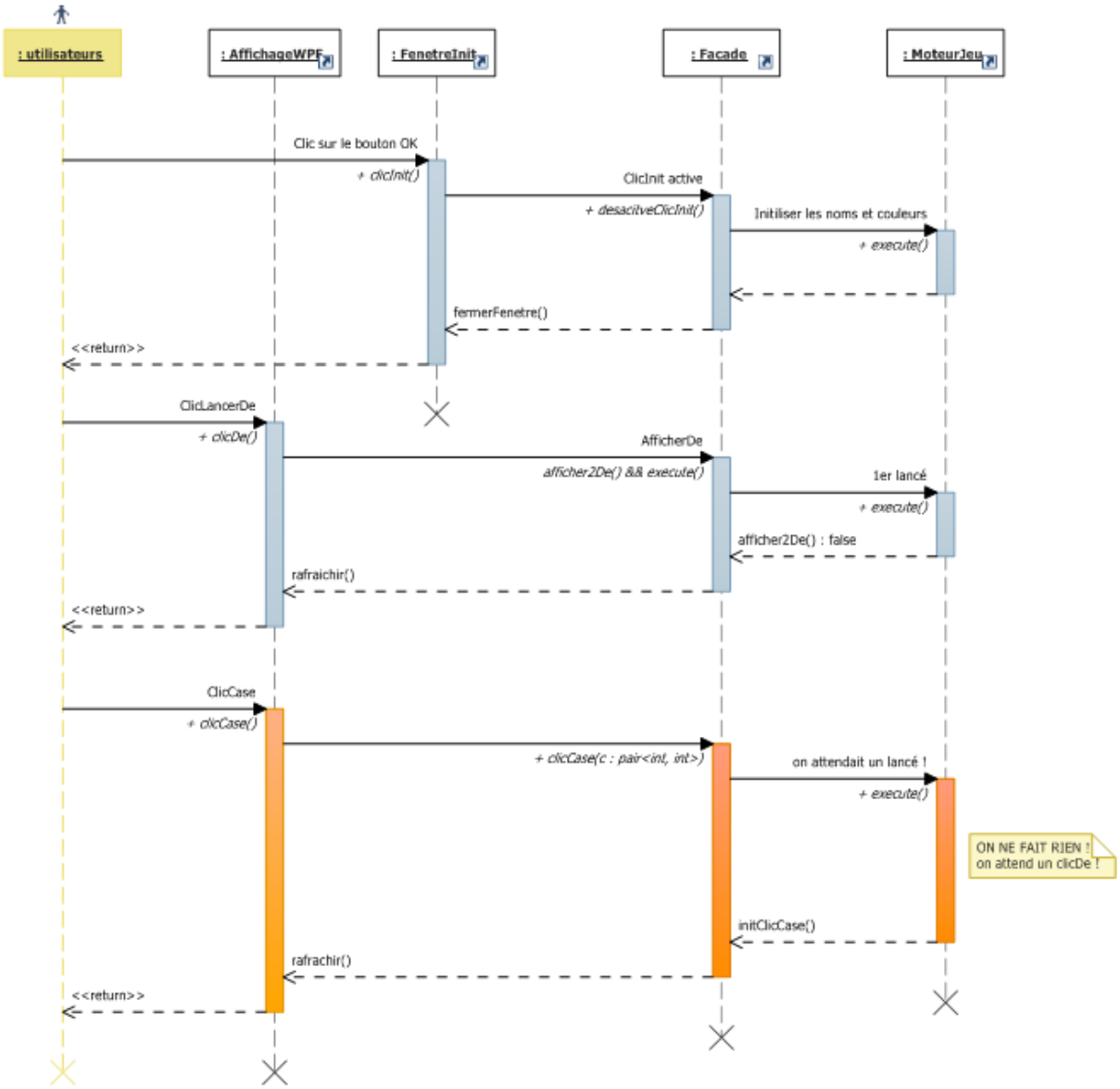


Figure 13 – Diagramme de séquences représentant un scénario alternatif lors de l'initialisation

Un second scénario alternatif qu'il nous aurait été possible de traiter est le cas où deux joueurs obtiennent le même résultat pour les premiers lancers, définissant l'ordre de passage. Dans ce cas nous les faisons relancer chacun leur tour une nouvelle fois.

C. Scénario avec des erreurs

Ci-dessous un scénario avec erreur lors de l'initialisation, à savoir le nombre de champs Nom remplis est inférieur au nombre de joueurs choisis.

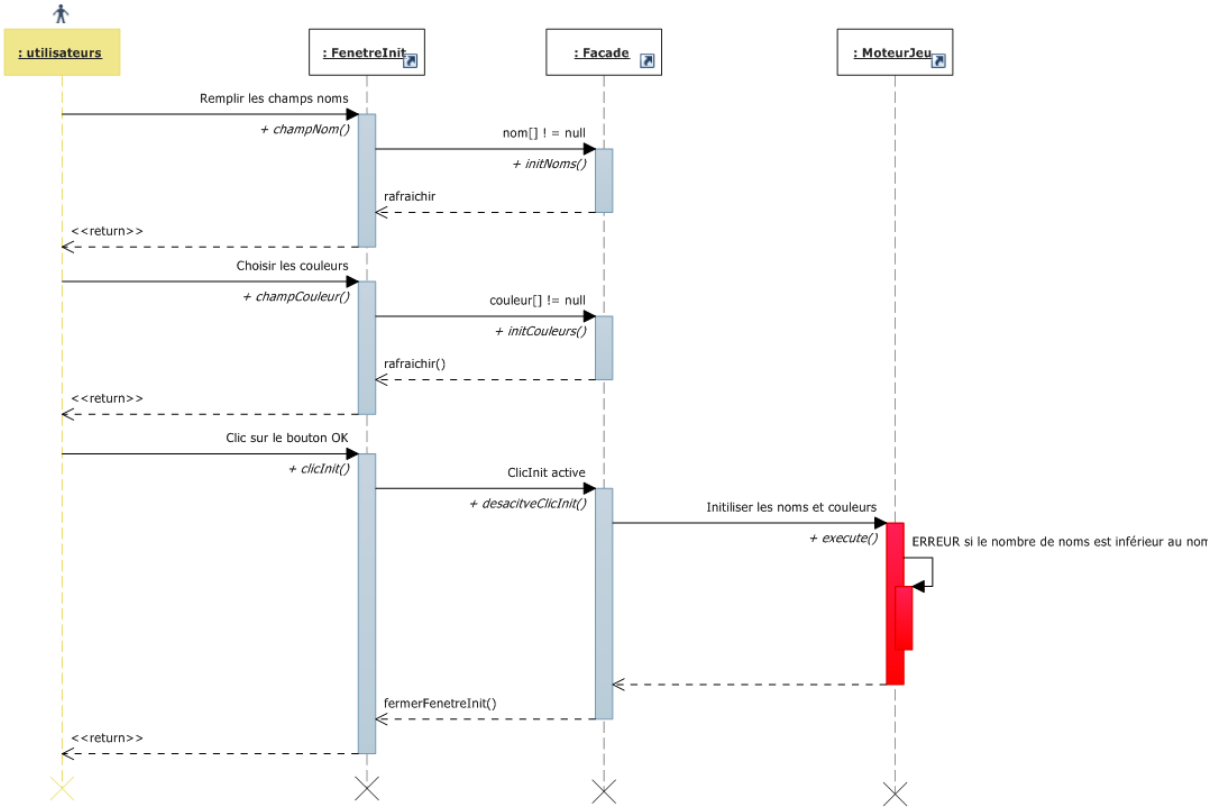
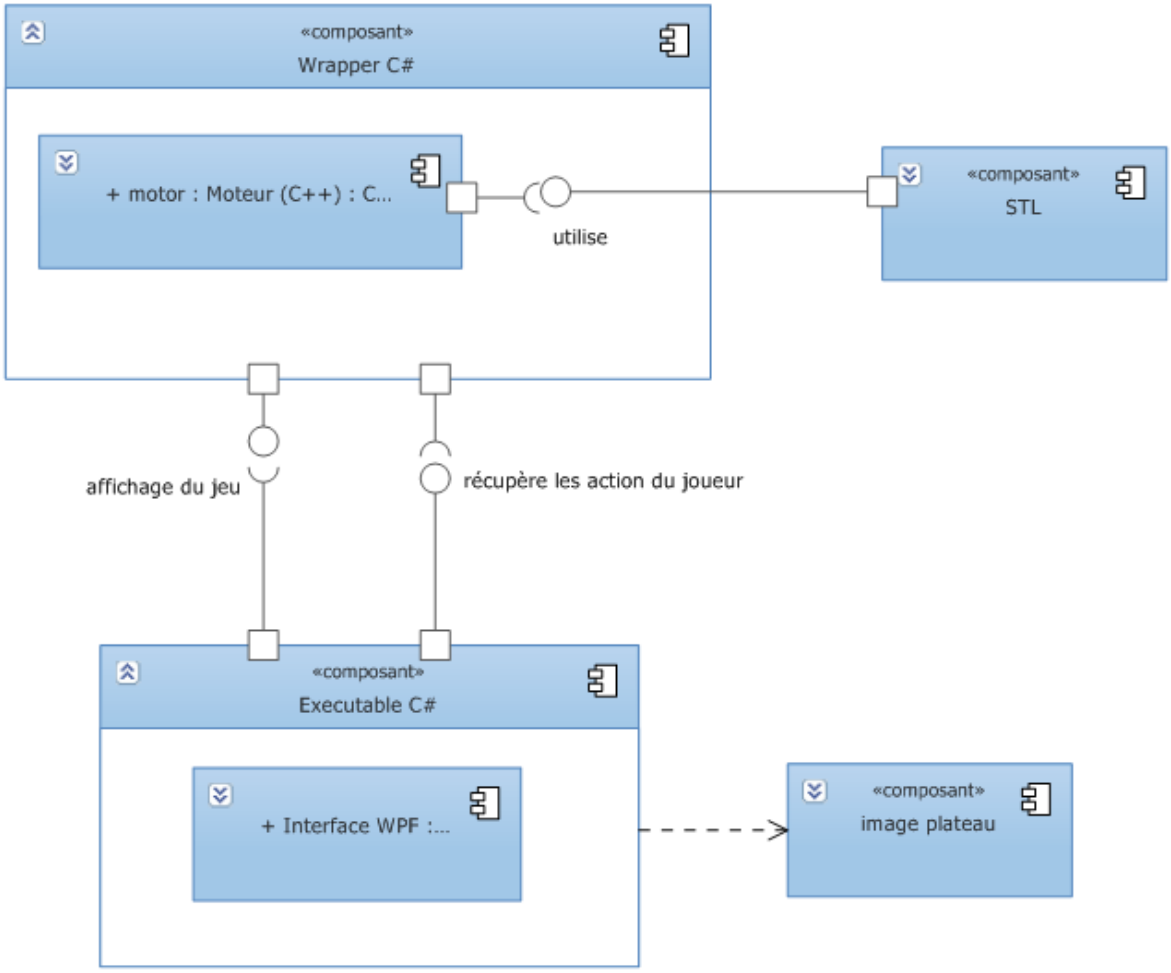


Figure 14 - Diagramme de séquences représentant un scénario avec erreur

VI. Composants et déploiement

Ci-après, le diagramme de composant de l'application :



VII. Interface Graphique

Voici une première ébauche d'interface graphique pour notre logiciel :

VIII. Conclusion

Ce rapport restitue et résume notre phase d'analyse et de conception du projet d'adaptation, sous forme logicielle, du jeu de société Canon Noir.

Il regroupe donc la modélisation du projet sous différentes formes de diagrammes. Ceux-ci nous permettent de spécifier au mieux l'architecture logicielle, et de faire ressortir ou de mettre en place des Design Patterns.

Plus cette conception est détaillée et précise, plus il sera facile de générer le code de l'application par la suite. Cette phase est donc primordiale et indispensable au succès du projet.

Bien que cette modélisation aurait du être exhaustive, nous serons très probablement amenés à devoir modifier certaines des classes, fonctions ou attributs exposés, notamment en ce qui concerne la façade, qui est au cœur de toutes les interactions, et dont le début du codage permettra de mettre en exergue des erreurs.