*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

[Sommaire 2](#_Toc278923934)

[I. Introduction 3](#_Toc278923935)

[II. Fonctionnalités du jeu 4](#_Toc278923936)

[A. Les cas d’utilisation 4](#_Toc278923937)

[III. Modélisation du jeu 6](#_Toc278923938)

[A. Les diagrammes de classes 6](#_Toc278923939)

[B. La modélisation globale du jeu 6](#_Toc278923940)

[C. Les différents modules du jeu 6](#_Toc278923941)

[1. Moteur 6](#_Toc278923942)

[2. Gestion de l’affichage 6](#_Toc278923943)

[3. Gestion de la carte 6](#_Toc278923944)

[IV. Fonctionnement des différents modules 7](#_Toc278923945)

[A. Diagrammes d’activités 7](#_Toc278923946)

[B. Diagrammes d’états-transitions 7](#_Toc278923947)

[V. Différents scénarios de jeu 8](#_Toc278923948)

[A. Diagrammes d’interaction 8](#_Toc278923949)

[B. Scénario nominal 8](#_Toc278923950)

[C. Scénario alternatif 8](#_Toc278923951)

[D. Scénario avec des erreurs 8](#_Toc278923952)

[VI. Différents scénarios de jeu 9](#_Toc278923953)

[A. Diagrammes de composants et déploiement (diagrammes structurels) 9](#_Toc278923954)

[VII. Conclusion 10](#_Toc278923955)

1. 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.

1. 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.

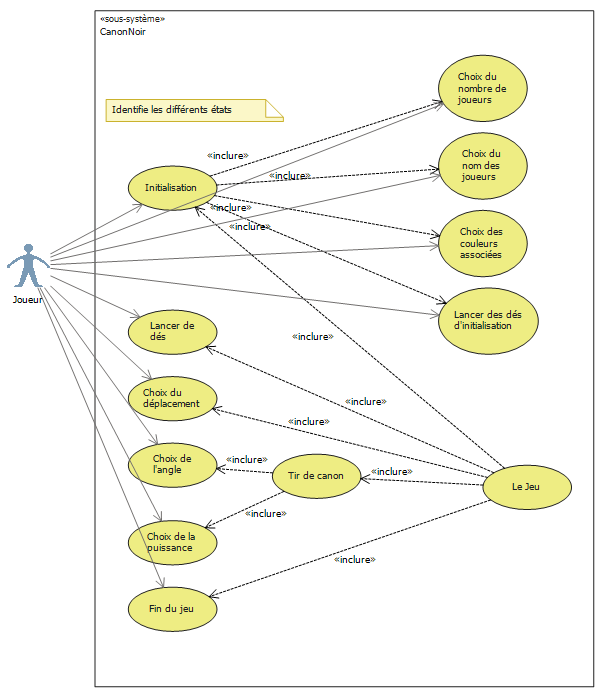


Figure 1 - Diagramme de cas d'utilisation de l'application

1. 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.

## 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

## Les différents modules du jeu

### 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.

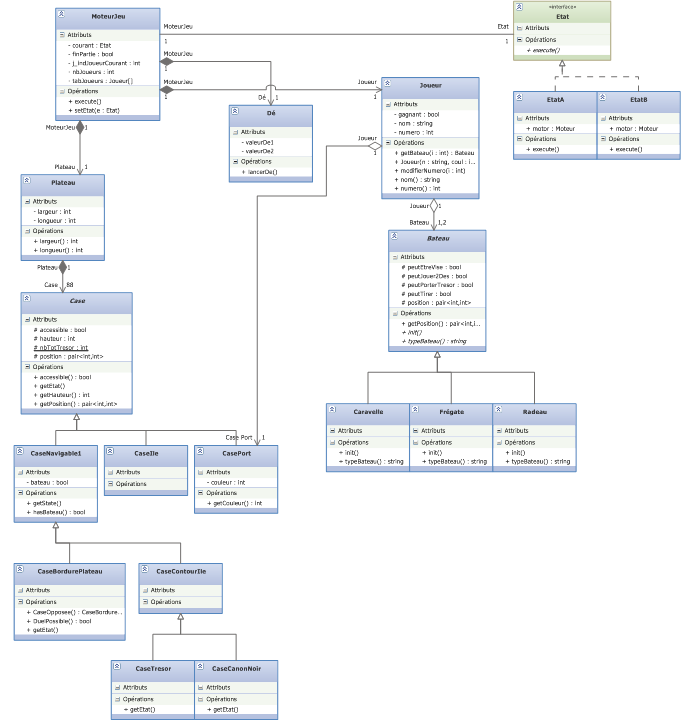


Figure 3 - Diagramme de classes global

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.

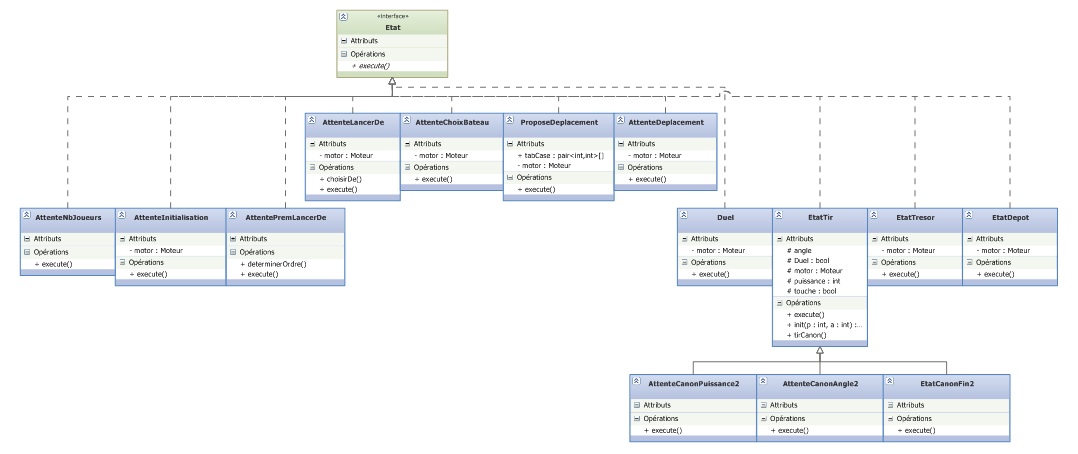


Figure 4 - Diagramme de classes des états du moteur

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

### 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.

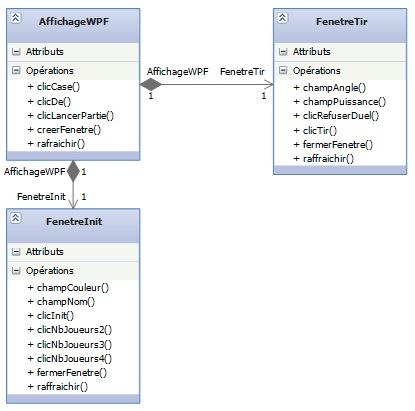
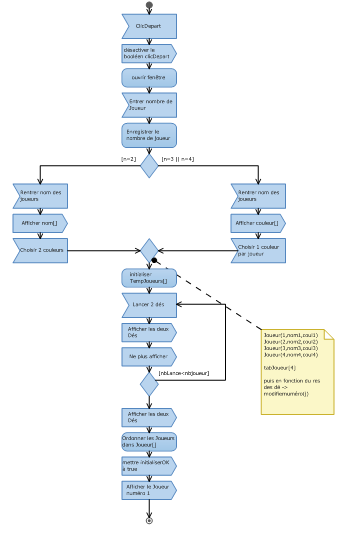


Figure 6 – Zoom sur les classes liées à l'affichage

1. Fonctionnement des différents modules

## Diagrammes d’activités

Les diagrammes d’activités, comme leur nom l’indique, se concentrent sur l’activité d’un processus. A l’instar des cas d’utilisation, ils font partis des diagrammes comportementaux.



## Diagrammes d’états-transitions

Encore des diagrammes comportementaux ! Les objets interagissent pour implémenter des comportements qui peuvent être décrit par des diagrammes d’interactions ou des diagrammes d’états-transitions qui sont pour leur part d’avantage centrés sur l’objet. Le diagramme se concentre donc sur un unique objet. (représenter les états possibles pour les objets de la classe)

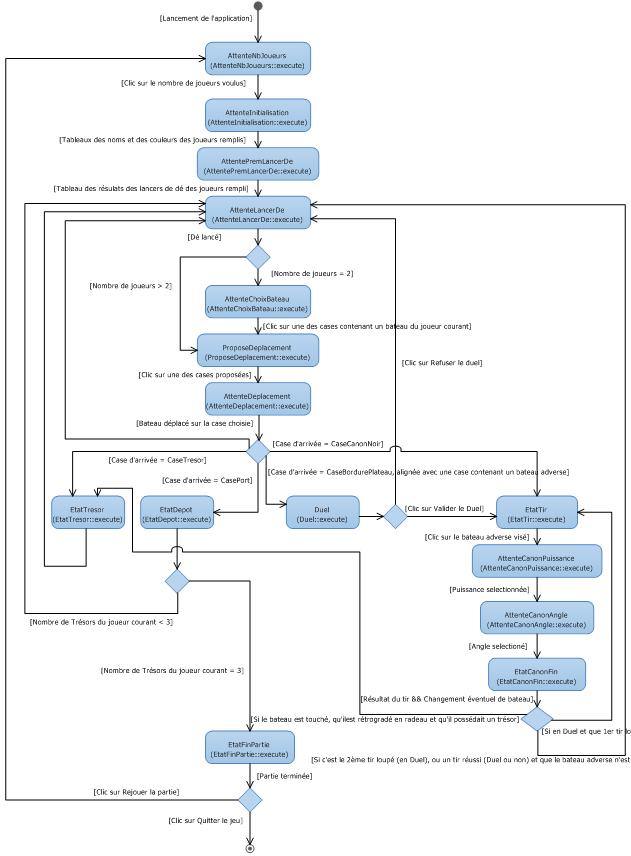


Figure 7 - Diagramme d'état-transitions

1. Différents scénarios de jeu

## Nous avons choisis de vous présenter deux scénarios qui correspondent à l’ouverture du programme et à l’étape du lancement des dés jusqu’au choix d’une case par le 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’utilisations, 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 : Nous n’avons étudié un scénario d’erreur et alternatif que dans le cas du scénario correspondant à l’initialisation.*

## Scénario nominal

Le scénario nominal de l’initialisation correspond aux figures 8.1 et 8.2

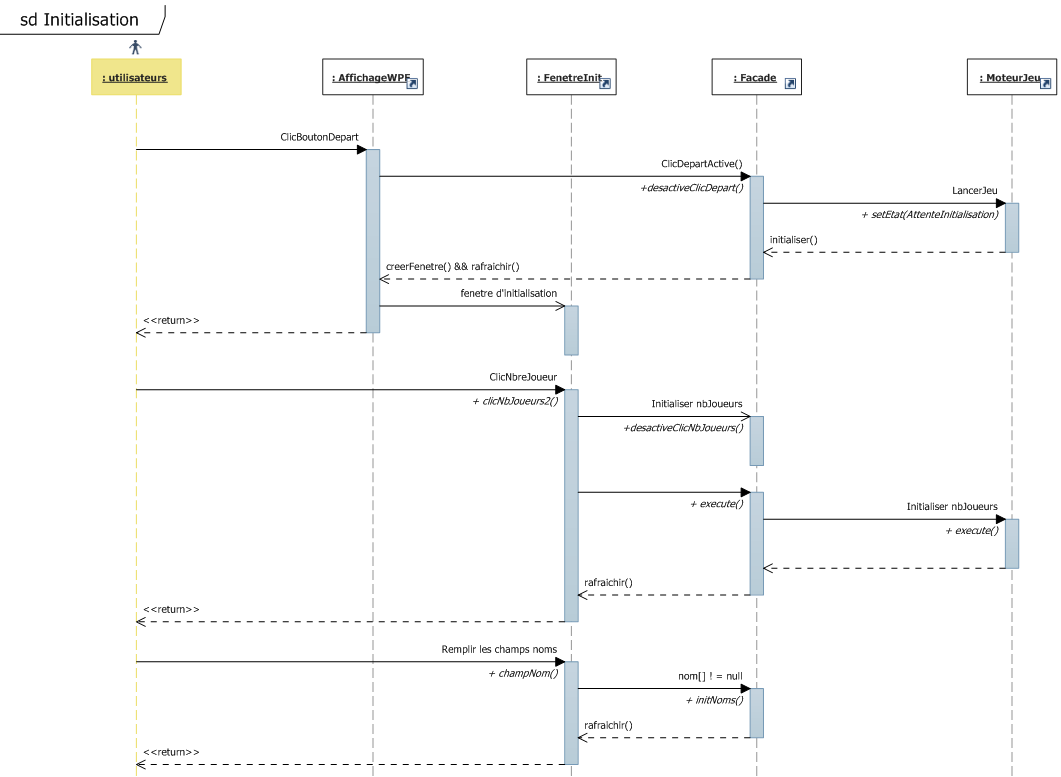


Figure 8.1 - Diagramme de séquences correspondant à l'initialisation (1/2)

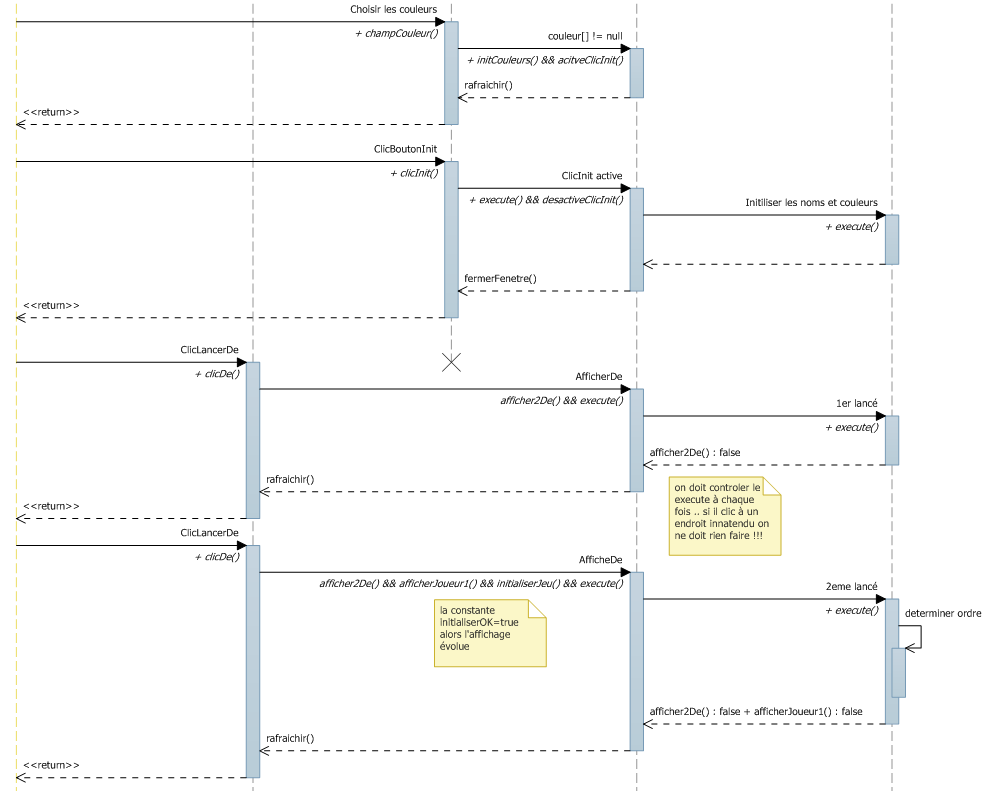


Figure 7.2 - Diagramme de séquences correspondant à l'initialisation (2/2)

Nous avons également opté pour vous présenter un diagramme 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 .. représente ce diagramme.

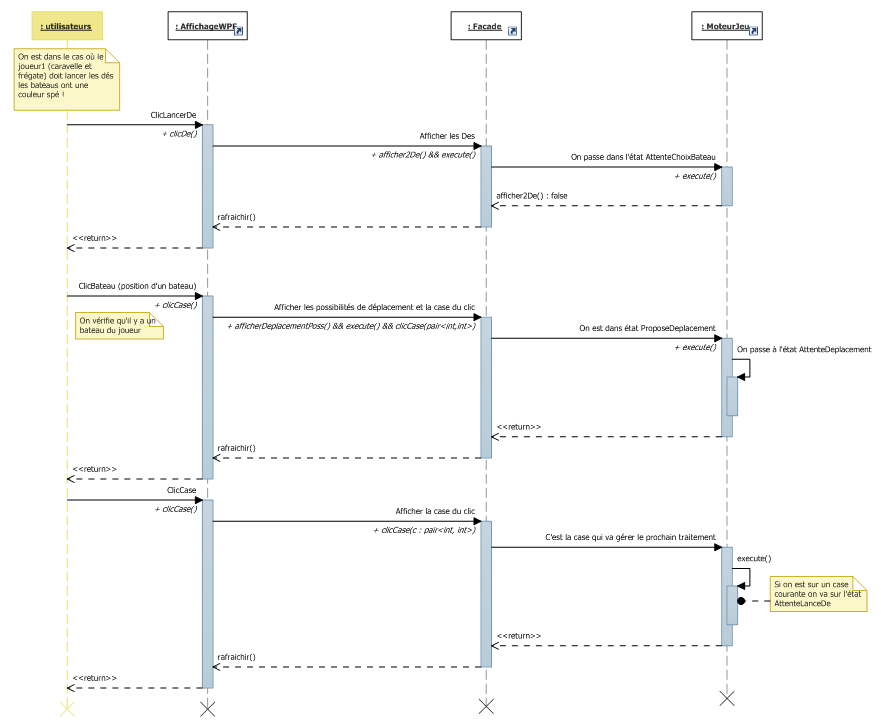


Figure 8 - Diagramme de séquences correspondant au lancement des dés

## Scénario alternatif

Ci-dessous un scénario alternatif lors de l’initialisation.

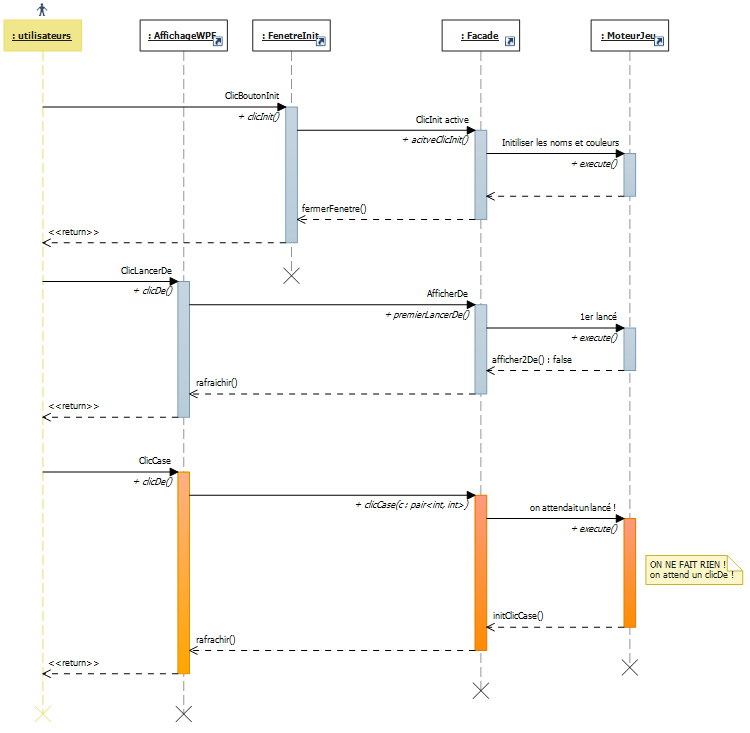


Figure 9 – Diagramme de séquences représentant un scénario alternatif lors de l'initialisation

## Scénario avec des erreurs

Ci-dessous un scénario alternatif lors de l’initialisation.

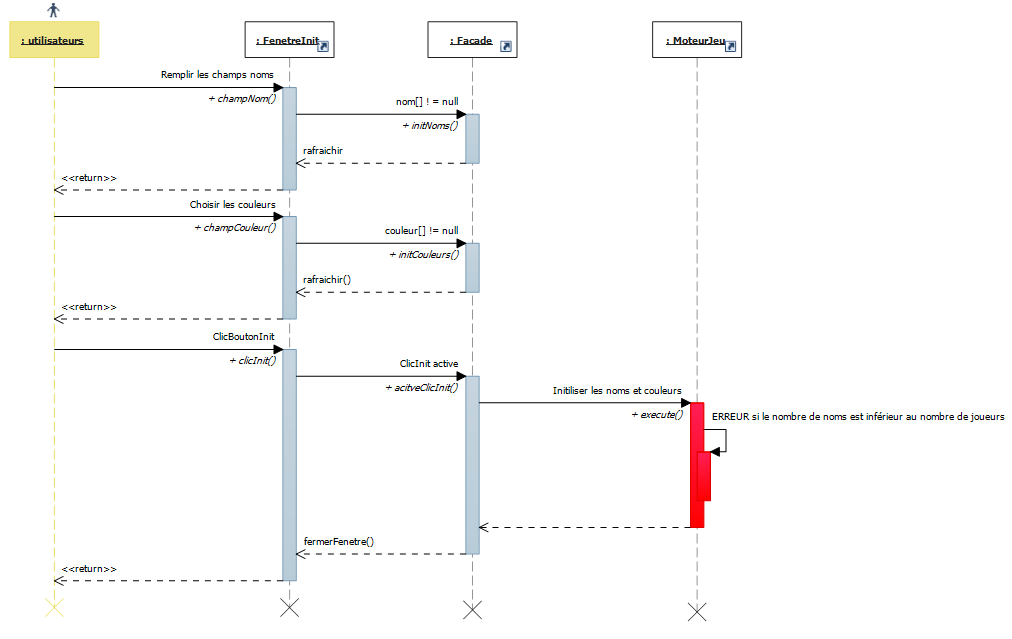


Figure 10 - Diagramme de séquences représentant un scénario avec erreur

1. Différents scénarios de jeu

## Diagrammes de composants et déploiement (diagrammes structurels)

1. Conclusion