BACLET Anne-Charline
BIAU-GUILHEMBET Marie
THAO KY Christophe

Projet de conception : Big Pirate

Informatique 2 - Année scolaire 2016-2017

Table des matières

Introduction **3**

I. Etude du système **3**

II. Itération n°1 - Jouer le tour du moussaillon **5**

1. Besoin 5
2. Analyse 5
3. Conception 6

III. Itération n°2 - Jouer le tour du pirate **9**

1. Besoin 9
2. Analyse 10
3. Conception 11

IV. Itération n°3 - Jouer le tour du fantôme **12**

1. Besoin 12
2. Analyse 13
3. Conception 14

V. Finalisation **15**

1. Implémentation détaillée 15
2. Contraintes et priorisation 16

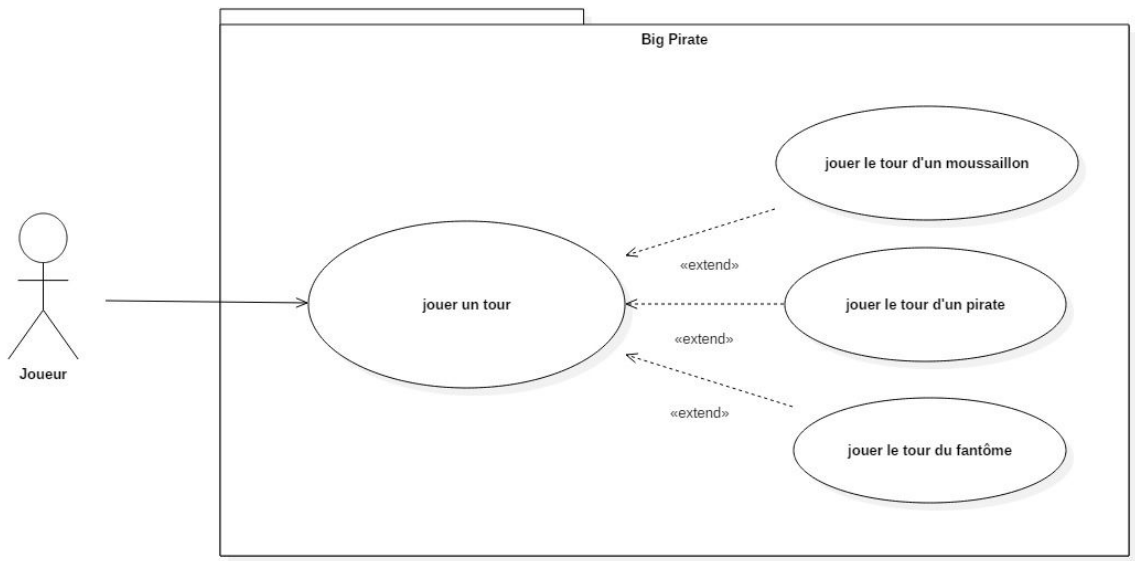
Conclusion **17**

Introduction

Le système à numériser est un jeu de stratégie intitulé Big Pirate. Ce projet est commun aux deux modules Modélisation / Méthode de conception et Java / Interfaces graphiques. Dans ce dossier, nous allons mettre en oeuvre les concepts d'itération et d'activités. Nous procéderons donc par découpe itérative du projet. Pour chaque itération, nous identifierons les besoins puis établirons l'analyse et la conception. Notre démarche, au travers de choix de conception, doit nous permettre d'effectuer une implémentation orientée objet aisée et évolutive du jeu.

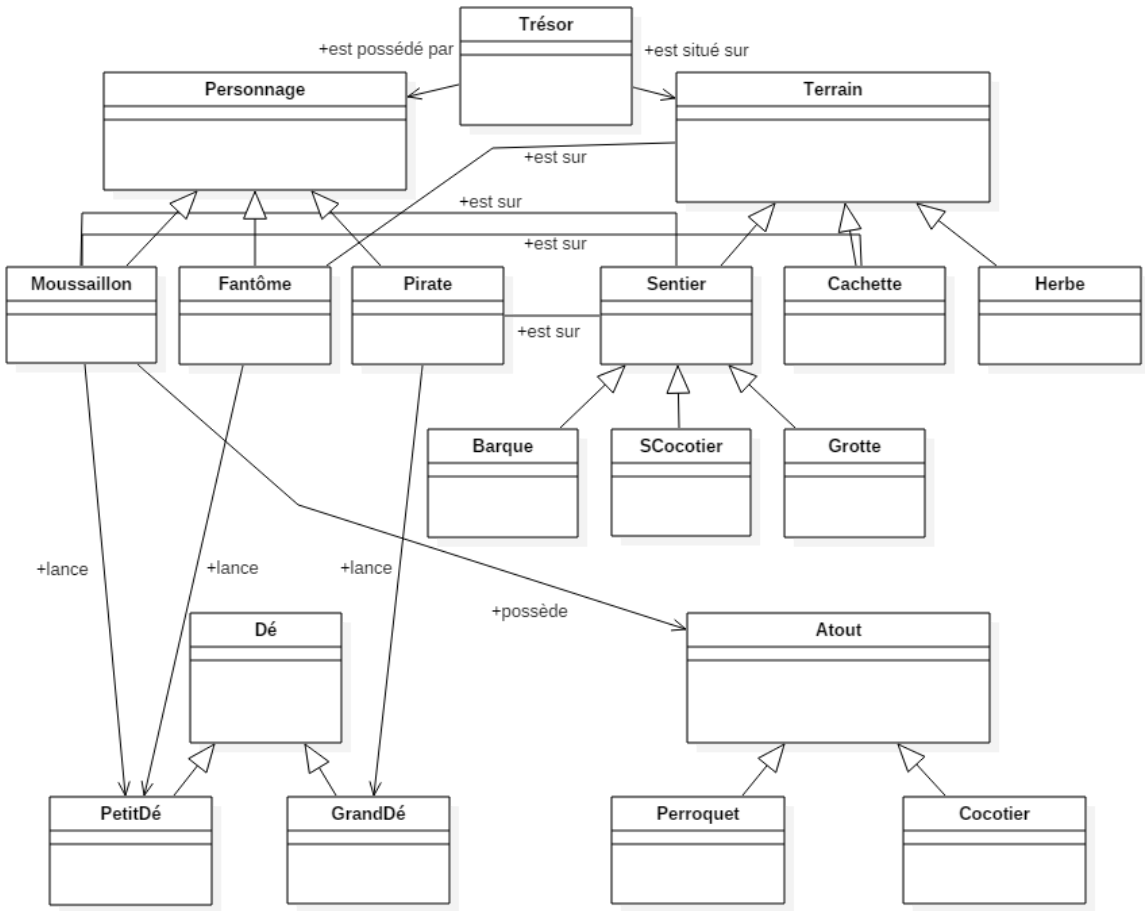
I. Etude du système

Le système que nous allons étudier tout au long de ce projet est le jeu intitulé Big Pirate. Afin de débuter notre étude du système, nous devons définir dans un premier temps ses limites. Les acteurs qui interagissent avec notre système sont les joueurs. Le système rend un type de service aux joueurs qui est de jouer un tour. En fonction du personnage incarné par le joueur, il s'agit soit du tour d'un moussaillon, soit du tour du pirate, soit du tour du fantôme. Cela nous permet d'établir le diagramme de cas d'utilisation suivant.



Le diagramme de cas d'utilisation nous permet de découper notre conception en trois itérations : jouer le tour d'un moussaillon, jouer le tour du pirate et jouer le tour du fantôme.
Nous réalisons également une ébauche du diagramme de classe sur laquelle nous nous appuierons pour établir les itérations et que nous améliorerons et détaillerons à la fin de ce dossier.

Dans ce diagramme, nous considérons 5 classes principales et dont les autres classes héritent. La classe Terrain regroupe toutes les cases du jeu. La classe Personnage contient les personnages du jeu qui sont situés sur une case du plateau. La classe Dé modélise les deux dés différents en fonction des personnages, l'un allant de 1 à 3 et l'autre de 1 à 6. La classe Atout modélise les deux types de cartes que peut avoir un moussaillon. Enfin, la classe Trésor est réalisée de telle sorte que tout trésor soit possédé par un personnage et situé sur une case du plateau.

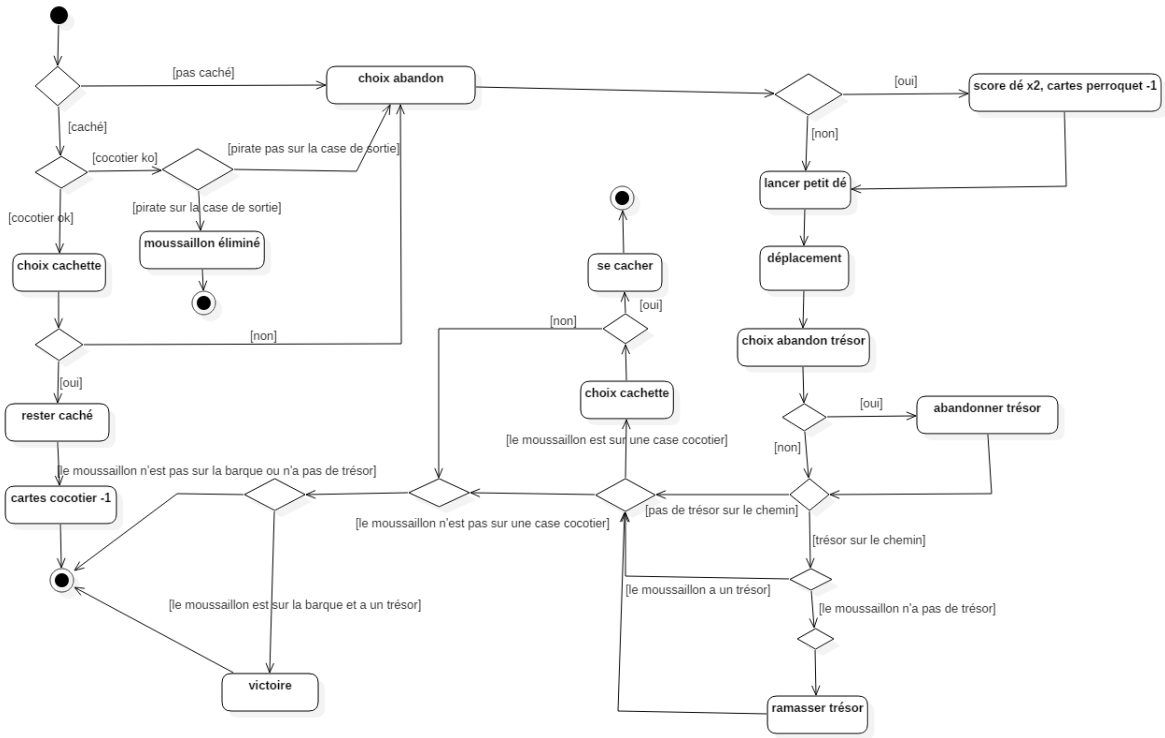


II. Itération n°1 - Jouer le tour du moussaillon

1. Besoin

Il y a de 1 à 3 moussaillons par partie.
Les moussaillons commencent la partie sur une case de la barque.
Un moussaillon doit pouvoir lancer un petit dé pour se déplacer, ramasser un trésor, abandonner son trésor, se cacher et rester caché dans une cachette grâce à des cartes cocotier, et doubler son score au dé grâce à une carte perroquet. Il doit également gagner la partie en revenant à la barque avec un trésor.

L'expression des besoins d'un moussaillon amène au diagramme d'activités suivant :



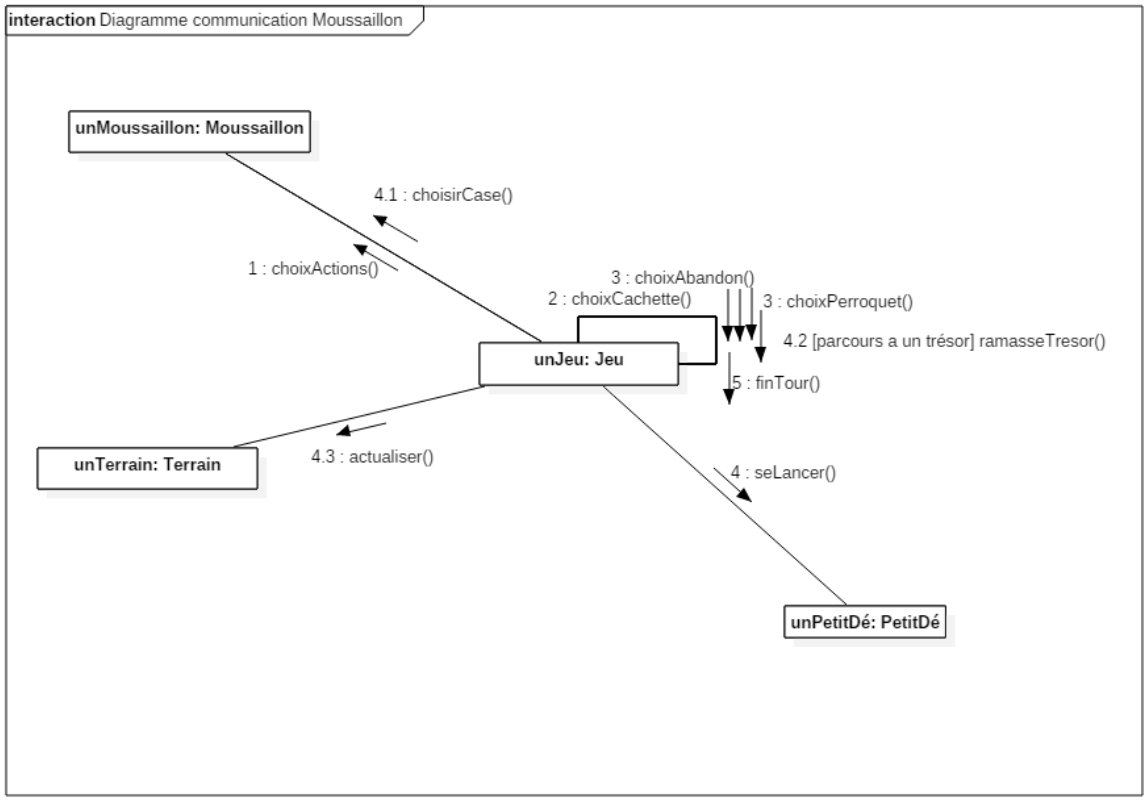
2. Analyse

Précondition : le tour du personnage précédent (fantôme ou autre moussaillon) est terminé
Postcondition: le moussaillon s'est déplacé ou est resté caché. S'il est retourné à la barque avec un trésor, il remporte la partie.

Le moussaillon a des comportements particuliers , qui sont les suivants :

- Au début du tour, il peut rester caché s'il l'est déjà et s'il lui reste une carte cocotier
- Utiliser une carte perroquet pour doubler son score au dé
- Abandonner son trésor pour retarder le pirate
- Se cacher s'il passe sur une case cocotier et qu'il possède une carte cocotier

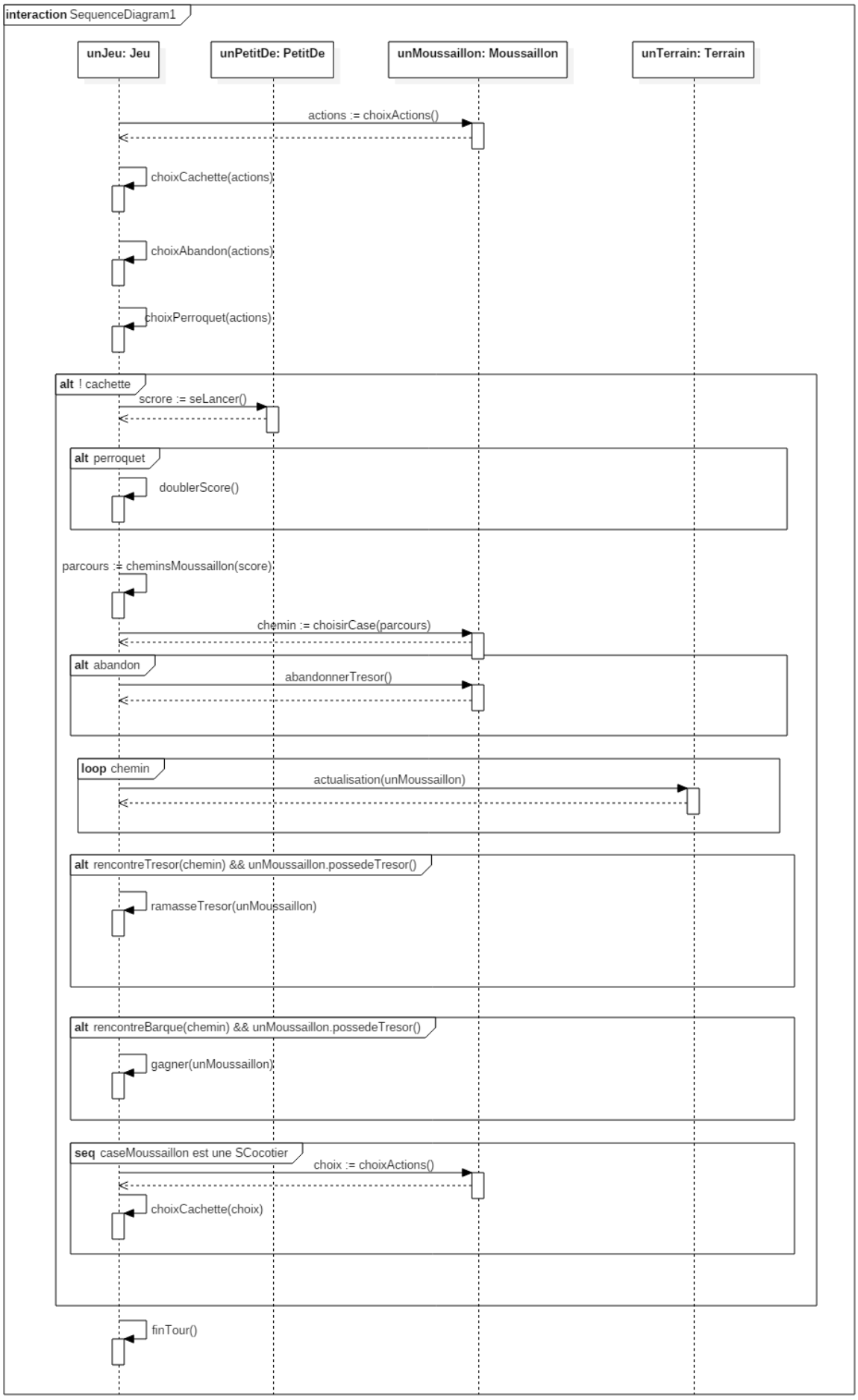
L'analyse du tour d'un moussaillon nous amène à réaliser le diagramme de communication suivant:



On suppose ici que le moussaillon était caché, qu'il décide au début du tour de sortir (le pirate ne se trouve pas sur la case de sortie) et qu'il décide également d'utiliser une carte perroquet et d'abandonner son trésor. On fait donc l'hypothèse qu'il n'y a pas de trésor sur le chemin, auquel cas le moussaillon le ramasserait s'il ne possédait pas déjà un trésor.

3. Conception

La solution envisagée pour répondre aux besoins d'un moussaillon se présente sous la forme d'un diagramme de séquence décrivant le tour d'un moussaillon :


interaction SequenceDiagram1
unJeu: Jeu
unPetitDe: PetitDe
unMoussaillon: Moussaillon
unTerrain: Terrain
actions := choixActions()
choixCachette(actions)
choixAbandon(actions)
choixPerroquet(actions)
alt ! cachette
scrore := seLancer()
alt perroquet
doublerScore()
parcours := cheminsMoussaillon(score)
chemin := choisirCase(parcours)
alt abandon
abandonnerTresor()
loop chemin
actualisation(unMoussaillon)
alt rencontreTresor(chemin) && unMoussaillon.possedeTresor()
ramasseTresor(unMoussaillon)
alt rencontreBarque(chemin) && unMoussaillon.possedeTresor()
gagner(unMoussaillon)
seq caseMoussaillon est une SCocotier
choix := choixActions()
choixCachette(choix)
finTour()

La méthode choixActions() permet à l'utilisateur d'indiquer quelles sont les actions (qui lui sont possibles) qu'il souhaite effectuer lors du tour. Cette méthode retourne une structure de données contenant des booléens correspondant aux choix de l'utilisateur, parmi :

- rester caché
- abandonner un trésor
- jouer un atout perroquet

La méthode choixCachette() va soit ne rien faire si le moussaillon n'était pas déjà caché, soit cacher le moussaillon s'il le souhaite, soit le ramener sur la case devant la cachette.
La méthode choixAbandon() abandonne le trésor du moussaillon sur sa case de départ lors du tour, si l'utilisateur a fait ce choix.
Enfin, la méthode choixPerroquet() indique, le cas échéant, qu'il faudra doubler le score du dé.

La méthode cheminsMoussaillon() donne les différents chemins possibles pour le moussaillon. Elle renvoie une structure de données contenant les différents chemins, parmi lesquels l'utilisateur choisit lors de la méthode choisirCase().

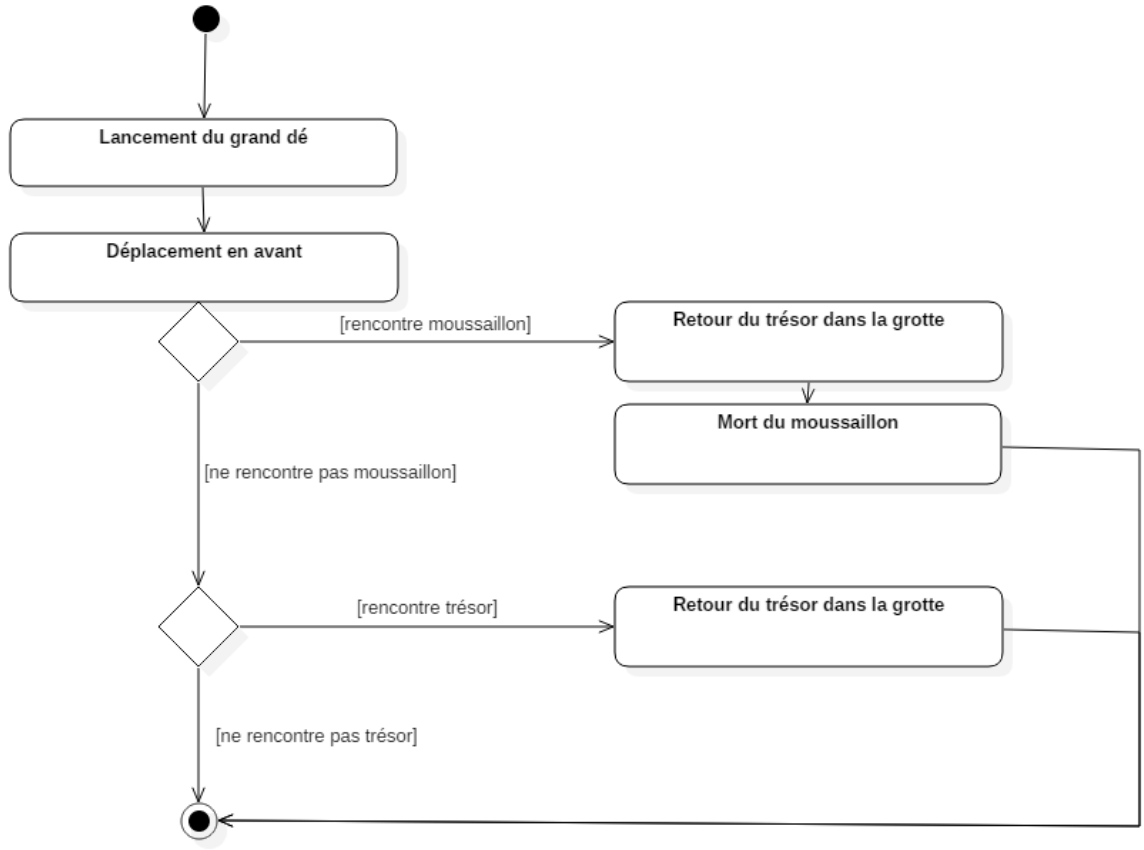
Une boucle actualise ensuite la position du moussaillon sur chaque case du chemin choisi pour réaliser le déplacement. La méthode ramasseTrésor() permet de ramasser un trésor si on peut, et la méthode gagner() met fin à la partie si le moussaillon retourne à la barque avec un trésor.

III. Itération n°2 - Jouer le tour du pirate

1. Besoin

Le pirate commence la partie sur la case grotte.
Le pirate doit pouvoir lancer un dé allant de 1 à 6 et il doit avoir la possibilité de choisir sa direction lorsqu'il atteint une intersection.

L'expression des besoins du pirate nous amène au diagramme d'activité suivant :



2. Analyse

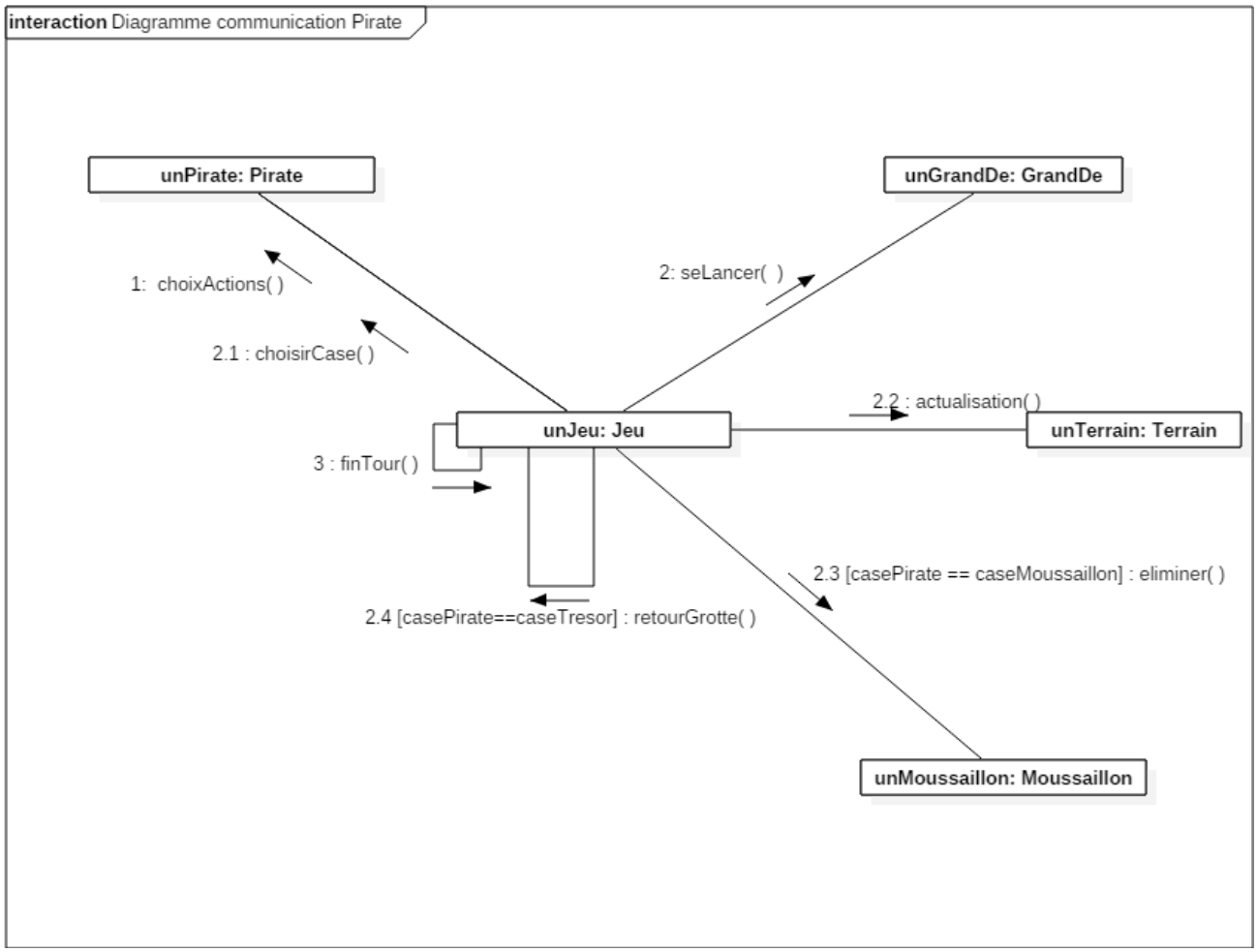
Précondition : le tour du moussaillon est terminé
Postcondition : le pirate s'est déplacé

Il a des comportements particuliers :

- Le pirate ne peut se déplacer qu'en avant
- Lorsqu'il rencontre un moussaillon, il le tue et si ce dernier est en possession d'un trésor, celui-ci est renvoyé à la grotte. Le pirate doit également s'arrêter sur la case du moussaillon.
- Lorsqu'il rencontre un trésor, il s'arrête à l'emplacement de celui-ci et renvoie le trésor à la grotte

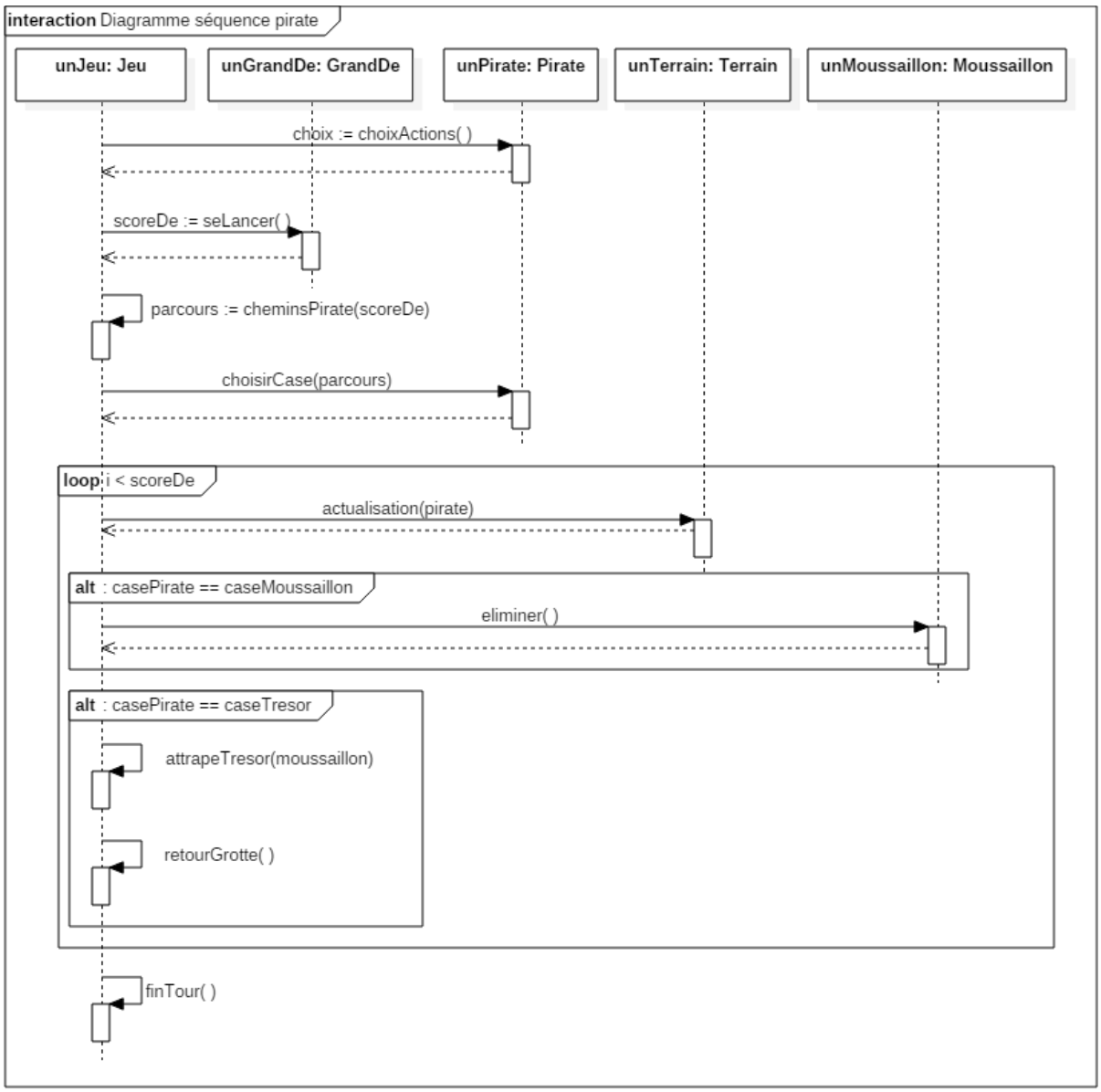
Une fois que le pirate s'est déplacé et que l'on ait vérifié l'ensemble des cases sur lesquelles celui-ci s'est déplacé, le tour se finit.

L'analyse du tour du pirate nous amène au diagramme de communication suivant :



3. Conception

Pour répondre à ce besoin, une solution a été trouvée sous la forme d'un diagramme de séquence décrivant les événements se produisant lors d'un tour du pirate :



La méthode choixActions() fait apparaître un pop-up pour l'utilisateur pour qu'il choisisse les actions à effectuer. Ici, le pirate ne peut que se déplacer donc cette méthode est purement inutile et ne sert que lorsque que l'on voudra améliorer le jeu en donnant au pirate d'autres choix. Cette méthode retourne une structure de donnée contenant des booléens modélisant les choix du pirate. Ces booléens seront utilisés pour déclencher les actions voulues par l'utilisateur.

La méthode cheminsPirate() calcule les chemins possibles du pirate en fonction du score du dé lancé. Elle retourne une structure de donnée contenant les chemins possibles du pirate.

La méthode choisirCase() utilise cette structure de données pour afficher les choix du pirate à l'utilisateur.

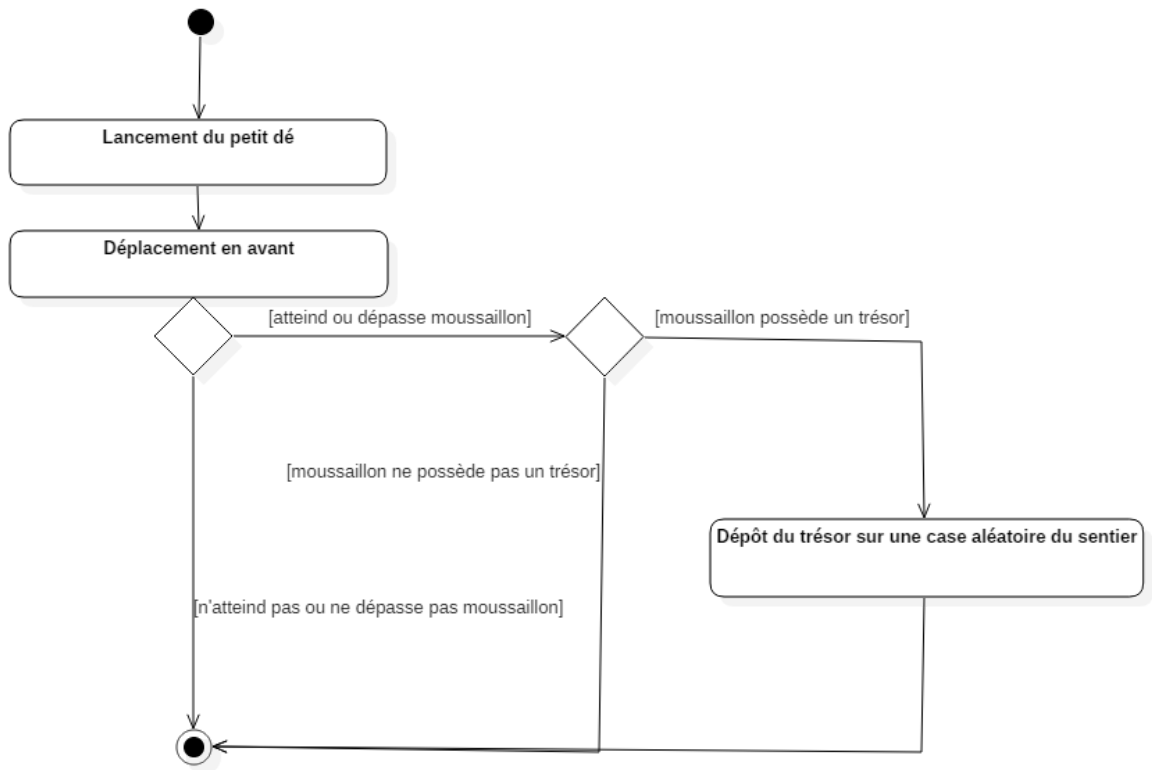
Enfin, une boucle actualise la position du pirate sur chaque case constituant le parcours choisi par l'utilisateur.

IV. Itération n°3 - Jouer le tour du fantôme

1. Besoin

Le fantôme démarre la partie sur la case F.
Aucun joueur ne déplace le fantôme.
Le fantôme doit pouvoir lancer un dé allant de 1 à 3 et il doit avoir la possibilité de traverser toutes les cases du plateau.

L'expression des besoins du fantôme nous amène au diagramme d'activité suivant :



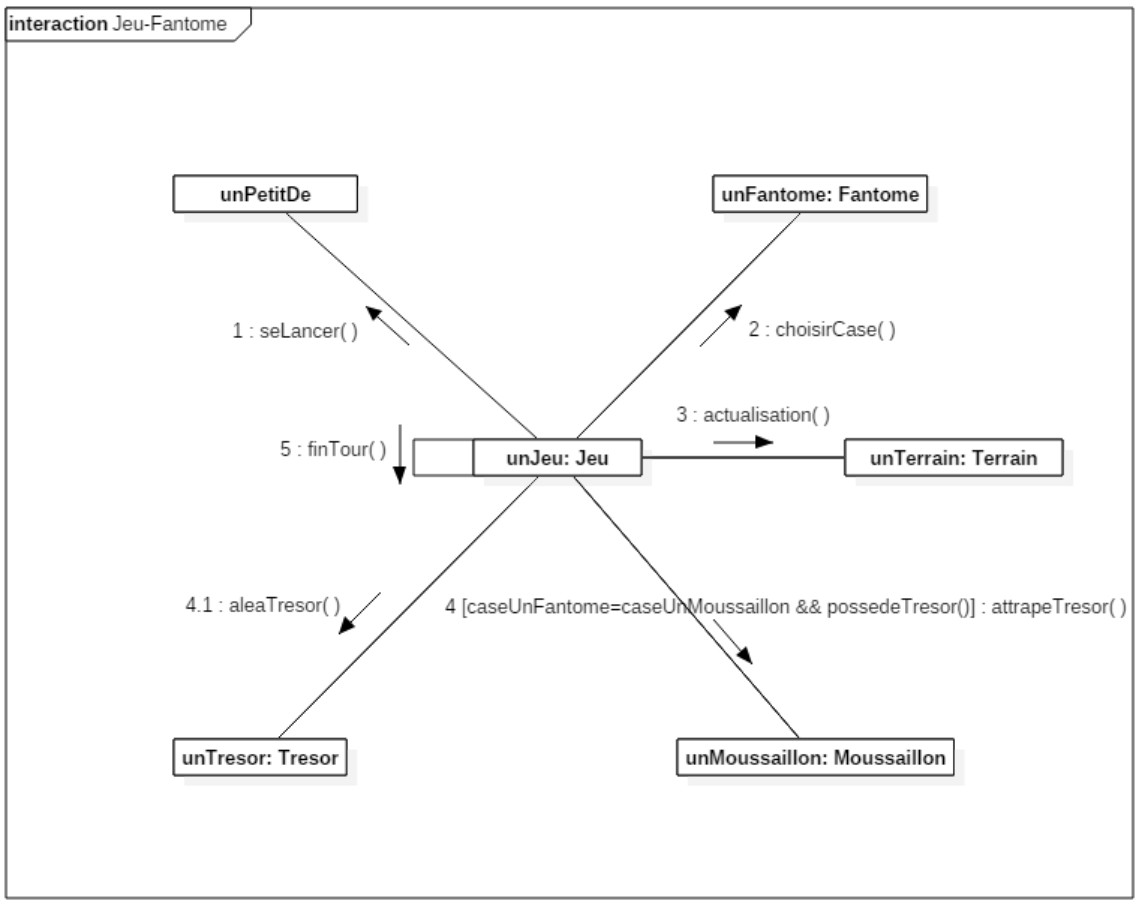
2. Analyse

Précondition : le tour du pirate est terminé.
Postcondition : le fantôme s'est déplacé.

Durant un tour, le fantôme peut accéder à certaines cases autour de lui. Si un moussaillon se trouve sur l'une de ces cases, alors le fantôme se déplace obligatoirement dans sa direction et s'arrête sur cette case. Sinon, s'il ne peut atteindre aucun case avec un moussaillon, alors il se déplace aléatoirement.

Le fantôme ne se déplace qu'en avant. Si, durant son déplacement, le fantôme rencontre un moussaillon possédant un trésor, alors le fantôme s'arrête sur cette case et s'empare du coffre, qui est déposé aléatoirement sur une case sentier du plateau. Cette case doit être vide (ni personnage, ni trésor).

L'analyse du tour du fantôme nous amène au diagramme de communication suivant (nous considérons le cas où un trésor est atteignable) :



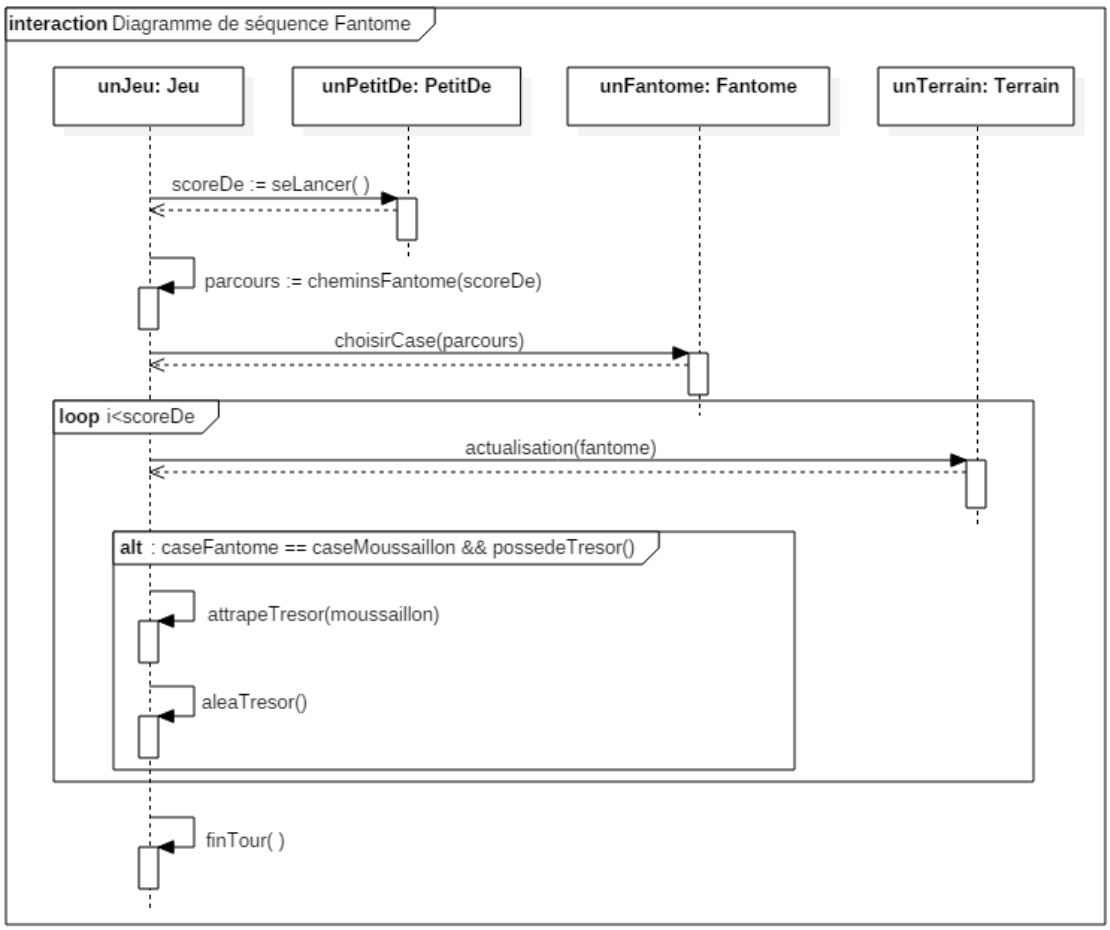
3. Conception

Afin de choisir le déplacement du fantôme (aléatoire ou vers un moussaillon avec trésor), nous choisissons d'implémenter la méthode cheminsFantome(scoreDe) qui est une recherche récursive de toutes les cases atteignables par le fantôme. S'il peut atteindre une case avec un moussaillon possédant un trésor, alors on envoie au fantôme le chemin menant à cette case dans la variable parcours. Dans le cas contraire, on lui envoie dans la variable parcours tous les chemins qu'il peut parcourir et le fantôme choisira aléatoirement parmi les chemins proposés.

La variable parcours est donc une collection de tableaux de 1 à 3 cases (en fonction du résultat du dé), chaque tableau représentant un chemin possible.

La méthode actualisation(fantome) met à jour la place du fantôme sur le plateau.
La méthode attrapeTresor(moussaillon) enlève au moussaillon son trésor.
La méthode aleaTresor() place un coffre aléatoirement sur une case vide du sentier.

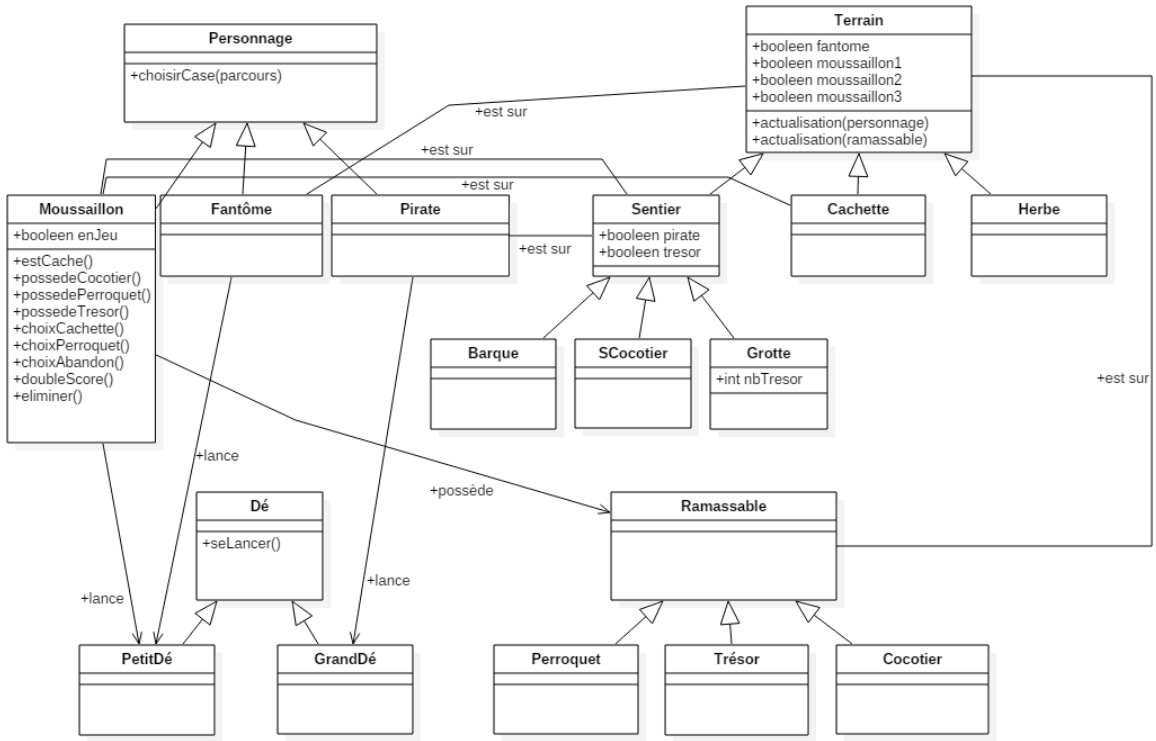
Pour répondre à ce besoin, une solution a été trouvée sous la forme d'un diagramme de séquence décrivant les événements se produisant lors d'un tour du fantôme :



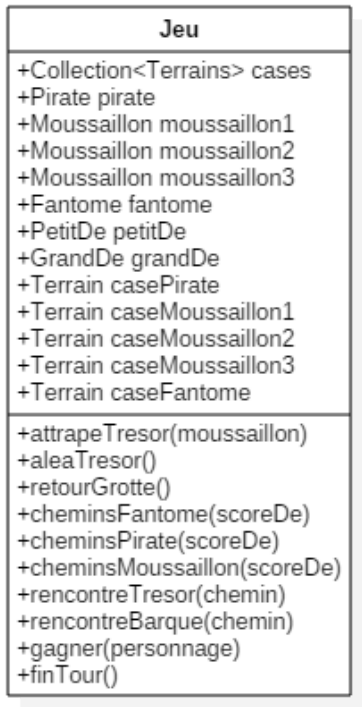
V. Finalisation

1. Implémentation détaillée

L'étude des itérations précédentes nous amène à établir un diagramme de classe détaillé :



Le design pattern que nous avons identifié est du type Modèle / Vue / Contrôleur. Le modèle correspond au diagramme de classe, la vue est l'interface graphique et le contrôleur correspond à la classe Jeu.



Le déroulement de la partie s'effectue de la manière suivante. On lance une partie en choisissant le nombre de moussaillons. La boucle de jeu (itérations 1 à 3) s'effectue tant qu'aucun moussaillon n'a atteint la barque et qu'il reste des moussaillons en jeu. Dès qu'une de ces conditions n'est plus remplie, le jeu se termine et on affiche le vainqueur.

2. Contraintes et priorisation

Les contraintes non fonctionnelles de notre jeu sont l'implémentation en Java à travers une approche orientée objet, la réalisation d'une interface graphique pour l'affichage et les interactions avec les joueurs, ainsi que l'obtention d'un jeu fluide et évolutif.

Le seul risque identifié, en supposant une bonne capacité à programmer en langage Java, est de ne pas finir dans le temps imposé. Pour pallier ce risque, il faut mettre en place un planning et faire preuve d'une bonne gestion du temps.

L'implémentation doit se faire suivant la priorisation suivante :
- classe Dé
- classes Personnage et Terrain
- classe Ramassable
- classe Jeu
- interface graphique

En effet, nous devons d'abord créer le modèle, puis le contrôleur, et enfin la vue.

Conclusion

La phase de conception du jeu informatisé Big Pirate nous a permis de dégager et d'étudier les trois grandes itérations du projet. Celles-ci consistent à jouer le tour du moussaillon, du pirate et du fantôme. En regroupant les itérations, nous avons élaboré un diagramme de classe complet ainsi que dégagé un design pattern de type Modèle / Vue / Contrôleur. Le risque majeur identifié est celui de la gestion du temps, c'est pourquoi il faudra planifier les étapes de l'implémentation. Notre jeu a été modélisé de façon évolutive et ergonomique afin de pouvoir implémenter des aménagements de la version de base.
Ce projet nous a permis de mettre en oeuvre la méthode itérative de modélisation Unified Modeling Language, nécessaire au bon développement d'un logiciel orienté objet. A présent, nous pouvons efficacement passer à l'implémentation en Java en s'appuyant sur la conception réalisée.