Projet UML

“Jeu fantastique”

11/6/2014

KALWANT Steven **LATNI Sofia** **MOUGAMADOU Javid**

LSI2

Sommaire

Introduction

Durant cette première phase du projet UML/Java, il a fallut que nous réalisons la conception et l'étude de la mise en place du jeu « Bataille fantastique », Dans cette partie, il faut en particulier mettre en œuvre les concepts d'itération et d'activités en modélisant les différentes itérations par les diagrammes d'états et de comportement, Cette première partie est très importante car c'est la base du projet et l'implémentation en Java se basera sur cette conception fondamentale.

L'application que nous souhaitons réaliser se présente comme ceci : il s'agit d'une application qui est le jeu « Bataille fantastique » qui permet à deux joueurs se trouvant devant la même interface de jouer plusieurs tours jusqu'à ce qu'il y ait un vainqueur, Chacun des deux joueurs doit composer une équipe de personnages fantastiques, ces derniers seront choisis parmi plusieurs personnages présentés dans un catalogue. Le jeu se déroule sur un plateau qui contient plusieurs cases, Chaque joueur à tour de rôle choisi un personnage attaquant et le genre d'attaque souhaité, Des descriptions plus détaillées sur jeu seront données au fur et à mesure dans l'avancement dans l'explication de nos choix.

Phase d'inception/élaboration :

Comme le système étudié est relativement restreint, on fusionne les phases d'inception et d'élaboration.

Phase de démarrage :

On cherche d'abord à identifier le système à informatiser et à comprendre ce qu'il fait.

Dans ce projet, le système que l'on va modéliser est le jeu « Bataille Fantastique ». Ce système sera utilisé par deux joueurs humains, l'interface cliente sera partagée par les deux joueurs. Ce système devra fournir les services suivants :

- Paramétrer le jeu
- Constituer une équipe
- Jouer la partie
- Consulter les informations du jeu
- Sauvegarder le jeu

Étant donné que le système étudié ici est relativement restreint, les cours d'UML et de Java que nous avons eue devraient nous permettre de réussir ce projet.

Le système sera implémenté en utilisant le langage Java. L'architecture envisagée pour l'implémentation est du type MVC.

Les concepts saillants du système sont au nombre de quatre et sont :

- un joueur : c'est un utilisateur du système.
- un personnage : c'est un élément du système contrôlé par le joueur qui possède des caractéristiques et des actions.
- Une action : c'est le comportement que peut avoir un personnage sur un autre.
- un tour : c'est une séquence de jeu durant laquelle un joueur peut effectuer des actions, comme sélectionné un personnage et attaquer.

On cherche maintenant à étudier les cas raffinés. Comme vu précédemment, le système doit fournir au moins cinq services. L'étude détaillée de ces services nous donne :

1er cas : Paramétrer le jeu

- Démarrer le jeu
- Définir le nombre de personnage par équipe
- Définir la taille du plateau

2ème cas : Constituer son équipe

- Choisir ses personnages parmi les différentes classes possibles
- Placer ses personnages sur les différentes cases du plateau

3ème cas : Jouer la partie

- Tirer au sort le 1er joueur
- Jouer à tour de rôle
- Lors de son tour, choisir le personnage avec lequel jouer
- Choisir de déplacer et/ou d'attaquer avec son personnage

4ème cas : Sauvegarder le jeu

- Garder la progression de la partie en quittant le jeu

5ème cas : Consulter les informations du jeu0

- Consulter la liste des personnages encore sur le plateau
- Afficher l'état du personnage
- Afficher les attaques du personnage

A partir de cette étude des cas raffinés on peut planifier l'ordre des différents cas. On obtient ainsi le planning suivant :

- Itération 1 : paramétrer le jeudi
- Itération 2 : constituer son équipe
- Itération 3 : jouer la partie
- Itération 4 : consulter les informations du jeu
- Itération 5 : sauvegarder le jeu

I. Itération 1 : Paramétrer le jeu

A. Capture des besoins : description détaillée du cas

Résumé : Les 2 joueurs doivent définir le nombre de personnages par équipe et la taille du plateau de jeu.

Pré condition : Le jeu a été lancé, les 2 joueurs sont sur la page de paramétrage du jeu.
Post condition : Le nombre de personnages par équipe et la taille du plateau de jeu ont été fixé.

Scénario principal :

1. Après avoir lancé le jeu, une fenêtre s'affiche.
2. Dans la fenêtre un message demande aux joueurs de définir le nombre de personnages par équipe.
3. Le jeu prend en compte le nombre fixé si celui-ci est compris entre 1 et 5. Sinon il demande de recommencer.
4. Un 2ème message s'affiche ensuite et demande aux joueurs de définir la taille du plateau
5. Le jeu prend en compte la taille fixé si celle-ci est comprise entre 5 x 10 et 15 x 30. Sinon il demande de recommencer.
6. Un message s'affiche a la fin pour dire le nombre de personnages par équipe et la taille du plateau.

Alternatives :

- **Le nombre de personnages par équipe est incorrect (étape 2)**

Si les joueurs entre un nombre qui n'est pas compris entre 1 et 5 pour le nombre de personnages par équipe, celui-ci ne sera pas pris en compte. Un message s'affiche pour demander aux joueurs de rentrer un nouveau nombre.

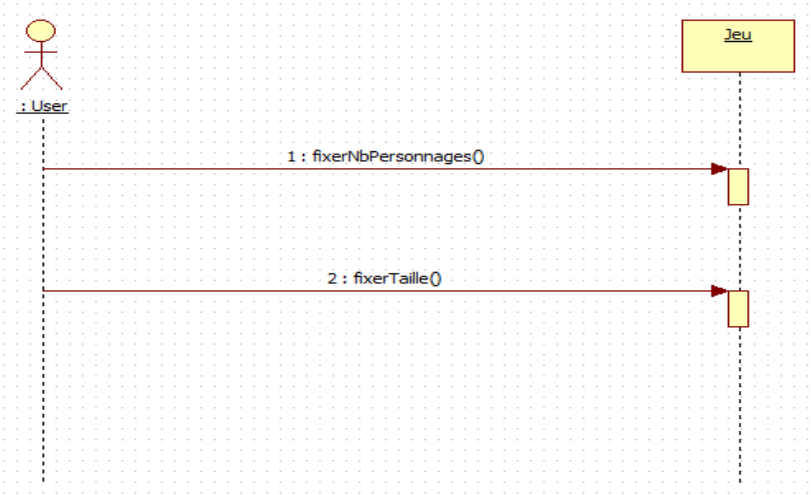
- **La taille du plateau est incorrect (étape 4)**

Si les joueurs définissent une taille pour le plateau qui n'est pas comprise entre 5x10 et 15x30, celle-ci n'est pas prise en compte. Un message s'affiche pour demander aux joueurs de rentrer une nouvelle taille.

B. Analyse

On cherche les instances qui participent activement au cas.

Analyse en boîte noire :



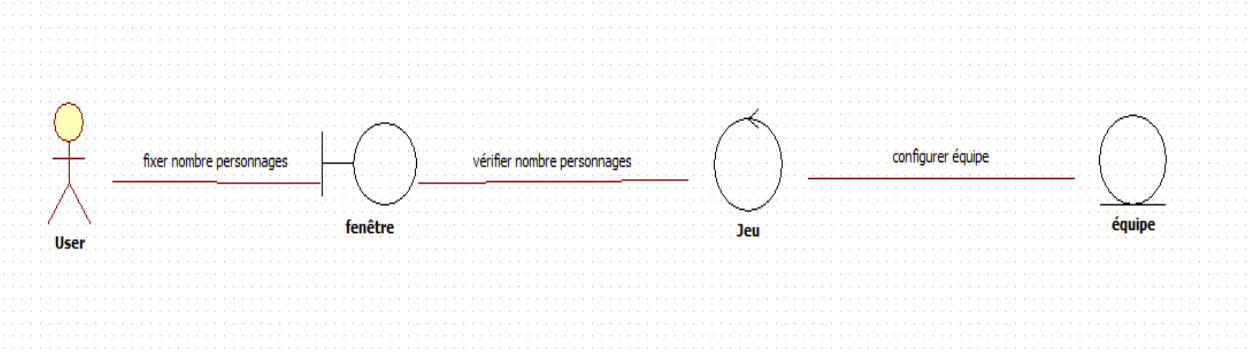
On effectue une première approche du cas de manière peu détaillée. L'utilisateur effectue 2 actions sur le système, ici l'application : il fixe le nombre de personnages et la taille du plateau.

Analyse en boîte blanche :

On effectue une analyse plus détaillée du cas en détaillant chaque action avec un diagramme de communication.

Diagramme de communication : fixerNbPersonnages( ) :

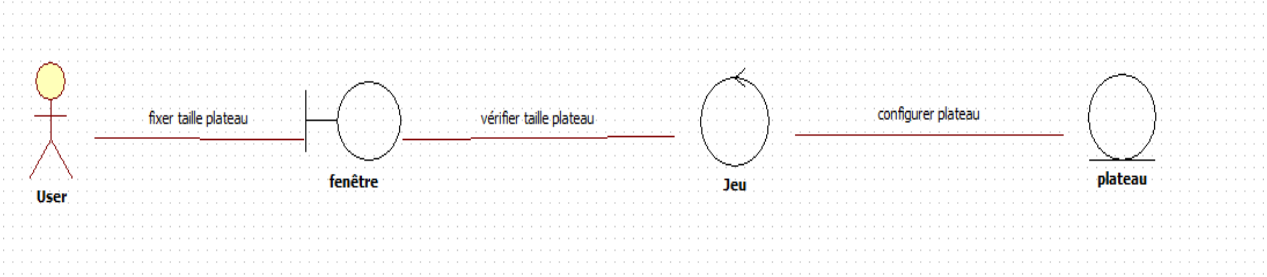
Pour la 1ère action, fixer le nombre de personnages, on a le diagramme suivant :



Les instances qui participent à cette action sont : le contrôleur jeu, l'entité équipe et une interface fenêtre. Le joueur interagit avec la fenêtre et rentre un nombre de personnages. La fenêtre va alors transférer ce nombre au contrôleur jeu, qui vérifie que le nombre entré est correcte. Celui-ci transfère ensuite le nombre à l'entité équipe pour qu'elle configure le nombre de personnages.

Diagramme de communication : fixerTaille( ) :

Pour la 2ème action, fixer la taille du plateau, on a le diagramme suivant :



Les instances qui participent à cette action sont : le contrôleur jeu, l'entité plateau et l'interface fenêtre. Le joueur interagit avec la fenêtre et rentre une taille de plateau. La fenêtre va alors transférer cette taille au contrôleur jeu, qui vérifie que la taille entrée est correcte. Celui-ci transfère ensuite la taille à l'entité plateau pour qu'elle configure la taille du plateau de jeu.

On en déduit le diagramme de classes d'analyse suivant :

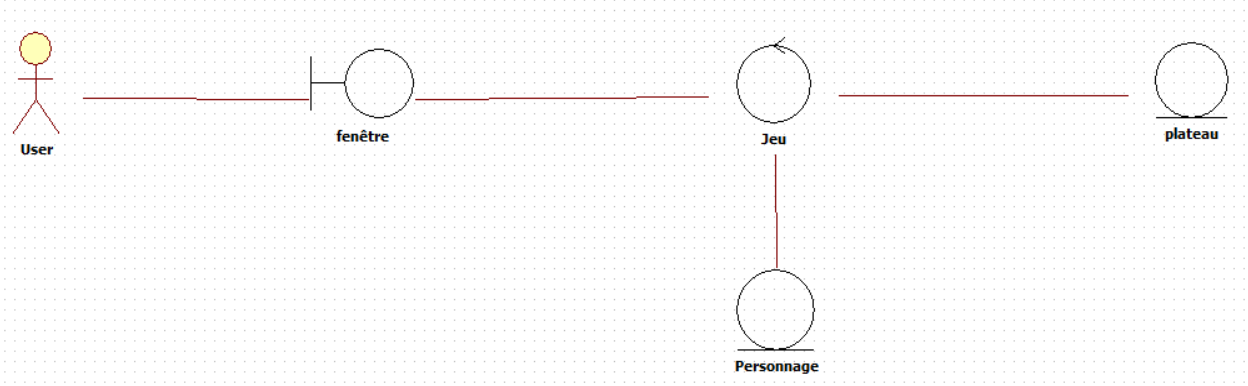


Diagramme de classe d'analyse de l'itération 1

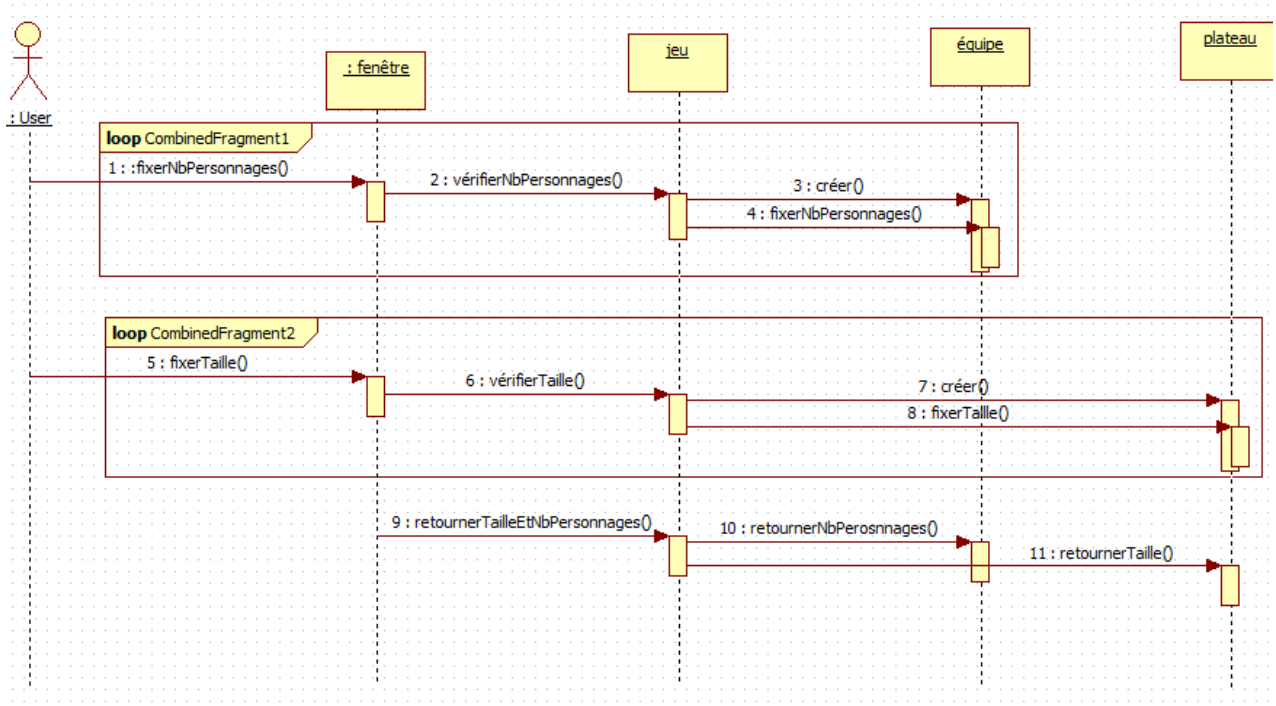
L'utilisateur interagit avec une interface fenêtre pour rentrer un nombre de personnages ou une taille de plateau. Cette interface se charge ensuite de transférer les données entrées au contrôleur jeu. C'est le jeu qui vérifie que les données entrées sont correcte et qui les envoie aux entités plateau et équipe pour les configurer.

C. Conception

Dans cette partie nous allons donner une ébauche de solution, avec des modèles proche de l'implémentation.

Diagramme de séquence :

A partir des diagrammes précédents, on en déduit le diagramme de séquence suivant :



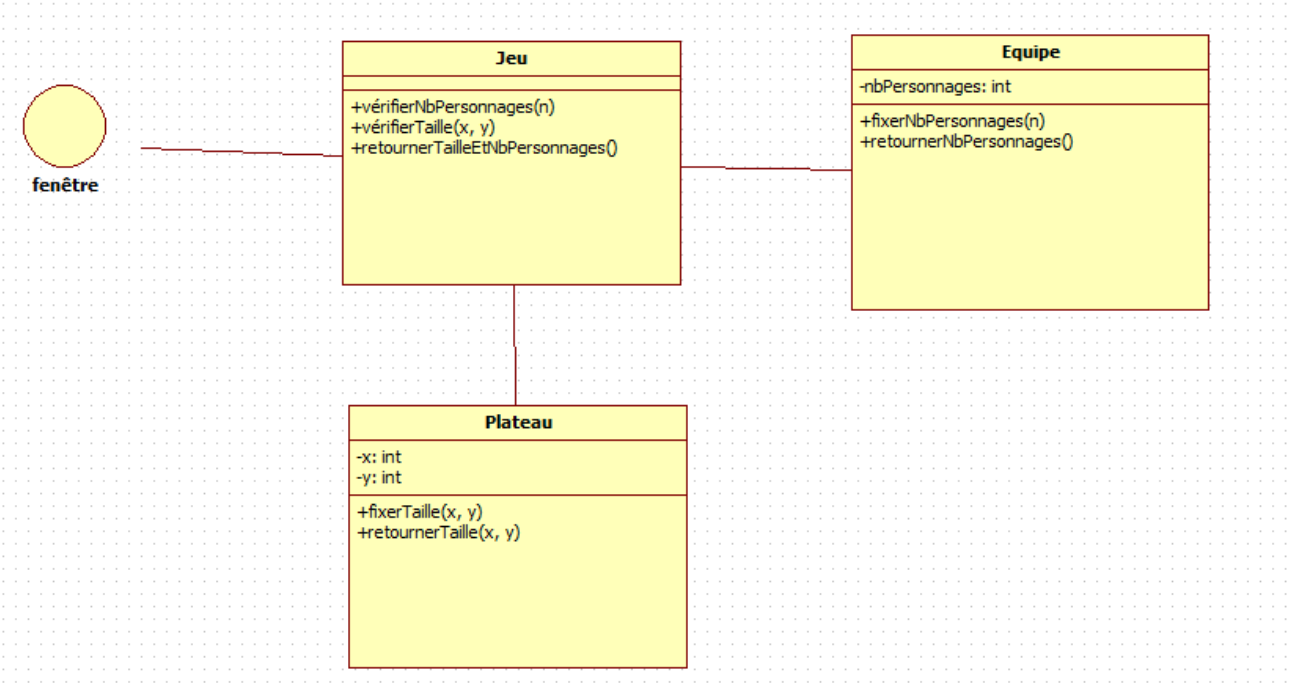
Pour la 1ère opération, le joueur entre un nombre dans l'interface fenêtre pour fixer un nombre de personnages. La fenêtre transmet ce nombre à l'instance jeu, qui vérifie que le nombre entré est bien compris entre 1 et 5. Si c'est le cas, il appelle d'abord le constructeur de la classe équipe pour créer deux instances équipe puis il appelle la méthode fixerNbPersonnages() de la classe équipe pour fixer le nombre de personnages des équipes avec le nombre entré. Sinon il renvoie un message d'erreur à la fenêtre, qui demande au joueur de rentrer un nouveau nombre.

Pour la 2ème opération, le joueur entre une taille dans l'interface fenêtre pour fixer une taille de plateau. La fenêtre transmet cette taille à l'instance jeu, qui vérifie que la taille entré est bien comprise entre 5x10 et 15x30. Si c'est le cas, il appelle d'abord le constructeur de la classe plateau pour créer une instance plateau puis il appelle la méthode fixerTaille() de la classe plateau pour fixer la taille du plateau avec la taille entrée. Sinon il renvoie un message d'erreur à la fenêtre, qui demande au joueur de rentrer une nouvelle taille de plateau.

A la fin, la fenêtre appelle la méthode retournerTailleEtNbPersonnages() de la classe jeu, qui appelle les méthodes retournerNbPersonnages() et retournerTaille() des classes équipe et plateau, qui retournent le nombre de personnages par équipe et la taille du plateau. Cela permet d'afficher le nombre de personnages par équipe et la taille du plateau au joueur.

Diagramme de classes :

On obtient alors le diagramme de classes suivant :


fenêtre
Jeu
+vérifierNbPersonnages(n)
+vérifierTaille(x, y)
+retournerTailleEtNbPersonnages()
Equipe
-nbPersonnages: int
+fixerNbPersonnages(n)
+retournerNbPersonnages()
Plateau
-x: int
-y: int
+fixerTaille(x, y)
+retournerTaille(x, y)

II. Itération 2 : Composition de l'équipe

A. Capture des besoins : description détaillée du cas

Résumé : Les 2 joueurs sélectionnent des personnages pour composer leur équipe par le biais du catalogue.

Pré condition : Le nombre de personnages par équipe et la taille du plateau de jeu ont été fixé.
Post condition : Les équipes sont définies.

Scénario principal :

1) Le système affiche le catalogue des personnages disponibles.
2) Le joueur sélectionne les personnages qui constitueront son équipe. Le système crée une nouvelle instance pour chaque personnage choisis.
3) Il affiche la liste des personnages choisis.

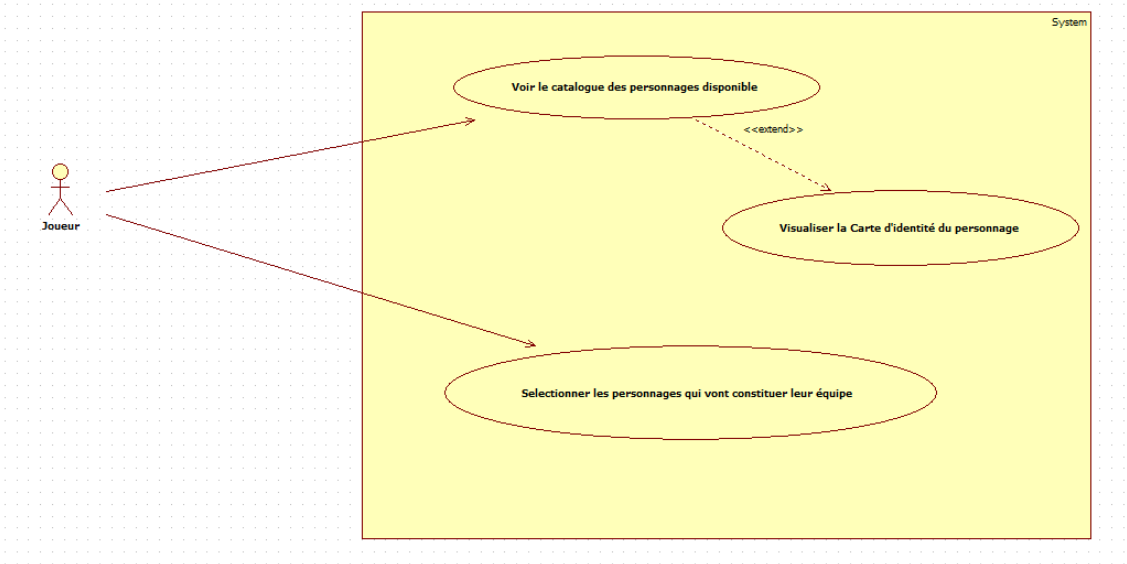
Alternatives :

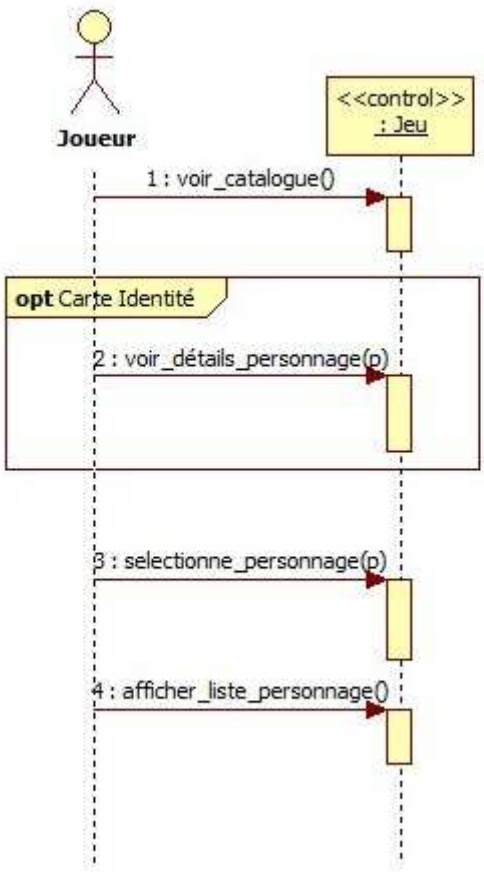
- **Le joueur peut en cliquant sur le personnage visualiser sa carte d'identité (étape 1a)**
  Une fois que le catalogue affiché, le joueur peut voir en cliquant sur le personnage sa carte d'identité, i.e. voir ses caractéristiques principales (PV, attaques, etc…).
- **Sélection des équipes (étape 2)**
  Le joueur ne peut sélectionner que 2 personnages maximum de même type dans son équipe.

B. Analyse

On cherche les instances qui participent activement au cas.

Analyse en boîte noire :





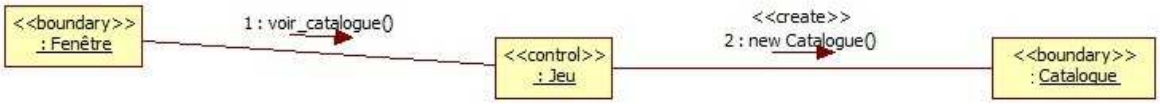
Nous avons détaillé les principaux services de cette itération que le système offre aux joueurs par le biais d'un message :

1. **voir_catalogue()** : Affiche une nouvelle fenêtre « Catalogue » regroupant les différents personnages disponibles.
2. **voir_détails_personnage(p)** : Service Optionnel prenant en entrée une sous-classe de Personnage « p » (par ex : Magicien) et permettant d'afficher une nouvelle fenêtre « carte d'identité » de la sous-classe pour voir ses attributs et ses actions.
3. **selectionne_personnage(p)** : service permettant aux joueurs de sélectionner le personnage qui doit constituer son équipe en passant « p » le type de personnage (sous classe de Personnage)
4. **afficher_liste_personnage()** : affiche la liste des personnages choisis dans l'équipe dans une nouvelle fenêtre .

Analyse en boîte blanche :

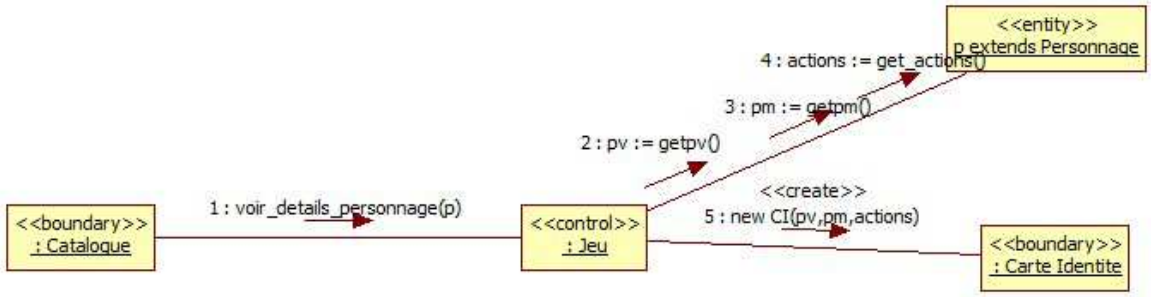
On détaille chaque action avec un diagramme de communication.

Diagramme de communication : Voir_catalogue() :



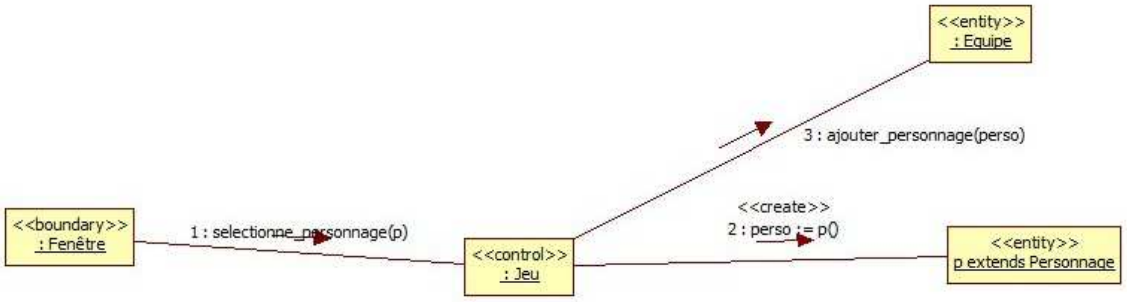
On crée une instance de la classe Catalogue.

Diagramme de communication : Voir_details_personnage(p) :



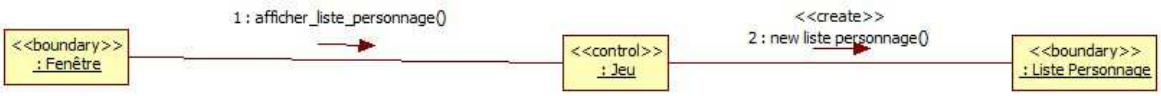
On crée une nouvelle instance de la classe Carte Identité en passant en entrée dans son constructeur les variables pv,pm,actions correspondant aux points de vie, points de mouvements et attaques possible par le personnage.

Diagramme de communication : Selectionne_personnage(p) :



On crée une nouvelle instance perso de la classe p qui est une sous classe de Personnage (par exemeple p = Magicien) et on ajoute perso dans l'équipe du joueur.

Diagramme de communication : Afficher_liste_personnage() :



On crée une nouvelle instance de la classe Liste Personnage.

On en déduit le diagramme de classes d'analyse suivant :

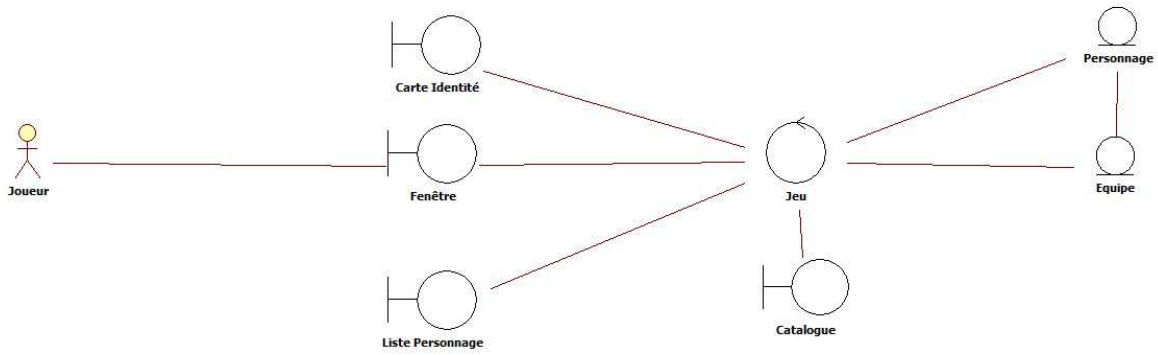


Diagramme de classe d'analyse itération 2

C. Conception

Dans cette partie nous allons donner une ébauche de solution, avec des modèles proche de l'implémentation.

Diagramme de séquence :

A partir des diagrammes précédents, on en déduit le diagramme de séquence suivant :

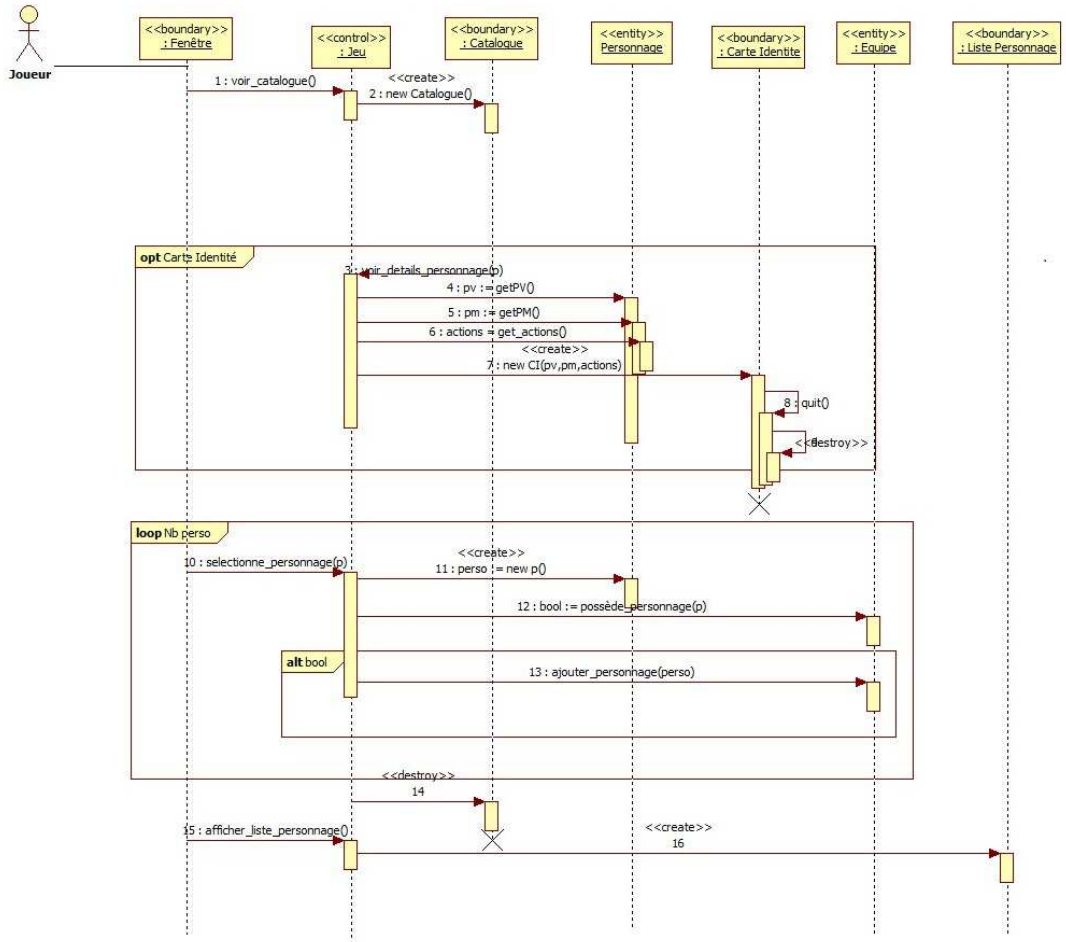
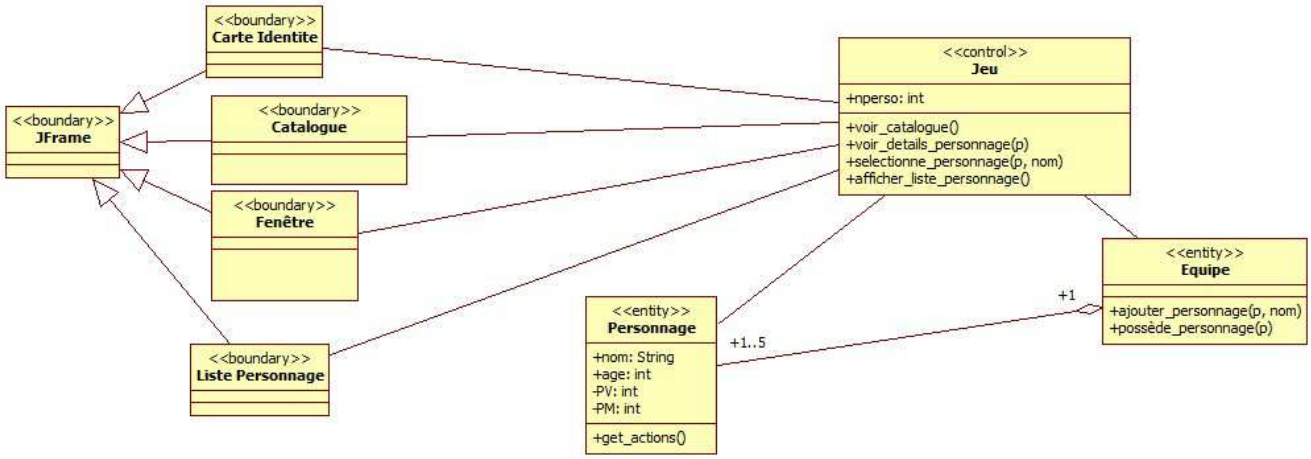


Diagramme de classes :

On obtient alors le diagramme de classes suivant :


<<boundary>>
Carte Identite
<<boundary>>
JFrame
<<boundary>>
Catalogue
<<boundary>>
Fenêtre
<<boundary>>
Liste Personnage
<<control>>
Jeu
+nperso: int
+voir_catalogue()
+voir_details_personnage(p)
+selectionne_personnage(p, nom)
+afficher_liste_personnage()
<<entity>>
Personnage
+nom: String
+age: int
-PV: int
-PM: int
+get_actions()
+1..5
+1
<<entity>>
Equipe
+ajouter_personnage(p, nom)
+possède_personnage(p)

III. Itération 3 : Jouer la partie

A. Capture des besoins : description détaillée du cas

L'ordre de passage des joueurs est donné par l'application, chacun des joueurs va placer ses joueurs sur le plateau et la partie va se dérouler en lançant un tour après un autre jusqu'à la fin de la partie.

Le scénario suivant résume le déroulement de la partie :

Nom	Jouer la partie
Auteur(s)	Joueurs
Description succincte	Le déroulement d'un tour joué par l'un des deux utilisateurs
Auteur	trinôme
Date(s)	03/10/2014 (première rédaction)
Pré-conditions	Le jeu doit être paramétré et les équipes déjà composées.
Post-conditions	La partie est terminée et un joueur a gagné
Démarrage	Les utilisateurs demandent de commencer la partie (appuient sur démarrer)

Le dialogue : le scénario nominal

Étape du scénario	Utilisateur	Système
1.		Le système fait un tirage au sort et affiche le joueur qui débute le jeu
3.		Affiche le bouton « démarrer le jeu »
4.	L'un des joueurs appuie sur le bouton.	
5.		Il affiche pour chaque joueur la liste des personnages choisis.
6.		Il affiche le plateau du jeu.
7.	Chaque joueur place ses personnages sur le plateau.	
8.		Il affiche le plateau avec les personnages dessus.
9.	Le 1er joueur joue avec un personnage de son choix en précisant l'attaque voulue.	
10.		Il exécute l'attaque Affiche les changements après l'attaque.
11.		Avant le début du nouveau tour vérifie si la partie est finie.
12.		Déclare la partie finie quand un personnage perd tous ses joueurs.
13		Réinitialise le jeu et ses paramètres

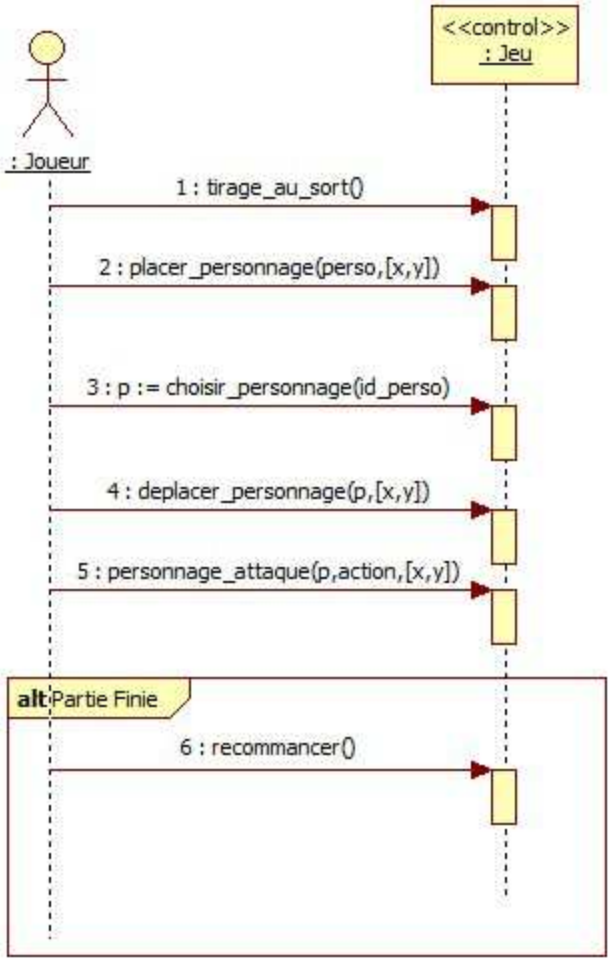
Le dialogue : les scénarios alternatifs

Étape du scénario	Utilisateur	Système
5.a	Il peut décider de consulter les propriétés de chaque personnage.	
8.a	Il peut décider de changer la disposition des personnages avant le début du 1er tour.	
10.a	Il peut décider de consulter l'état de ses personnages et ceux de son adversaire.	
Fin	Scénario nominal : à n'importe quel moment du jeu sur simple demande de l'utilisateur ou à l'étape 12.	

B. Analyse

La boite noire :

Dans un premier temps, nous avons essayé d'identifier la manière générale dont se déroule une partie, pour ce faire nous avons isolé les principaux messages envoyés par l'utilisateur de l'application vers l'application même. Ces principales interactions entre l'utilisateur et le système nous permettent de définir ce que nous appelons une « boite noire » qui se présente sous la forme suivante :



La boite noire de l'itération 3

Explications :

Nous distinguons six grandes actions demandées par l'utilisateur à l'application :

1-**tirage_au_sort()** : C'est pour demander au système d'effectuer un tirage au sort avant le début du premier tour, le but est de savoir quel joueur commence la partie.

2-**placer_personnage(perso,[x,y])** : C'est une demande de la part de l'utilisateur envoyée au système pour qu'il place le personnage « perso » rentré en paramètre à la case qui a pour coordonnées x en abscisses et y en ordonnées.

3-**p := chosir_personnages(id_perso)** : on stocke dans p le personnage choisi et qui a pour identifiant « id_perso », le fait de stocker le personnage dans p nous facilitera la manipulation de ses déplacements et ses attaques par la suite.

4-**deplacer_personnage(p,[x,y])** : C'est une demande de la part de l'utilisateur envoyée au système pour qu'il déplace le personnage « p » (récupéré avant) rentré en paramètre à la case qui a pour coordonnées x en abscisses et y en ordonnées.

5-**personnage_attaque(p,action,[x,y])** : une méthode réalisant l'attaque du personnage « p » qui se trouve dans la case de cordonnées x et y et on choisit d'exécuter l'attaque « action ».

6- **recommencer()** : c'est pour commencer le prochain tour qui aura exactement les mêmes étapes que

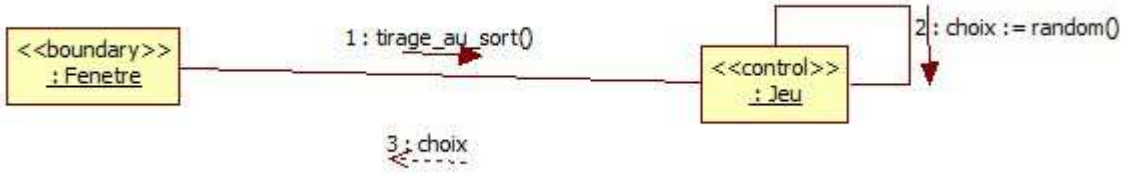
le précédent sauf que c'est le second joueur qui va commencer ce tour là.

Analyse en boîte blanche :

Pour chaque élément (action) de la boite noire, nous devons faire une boîte blanche sous forme d'un diagramme de communication. Donc au total nous aurons 6 diagrammes de communication.

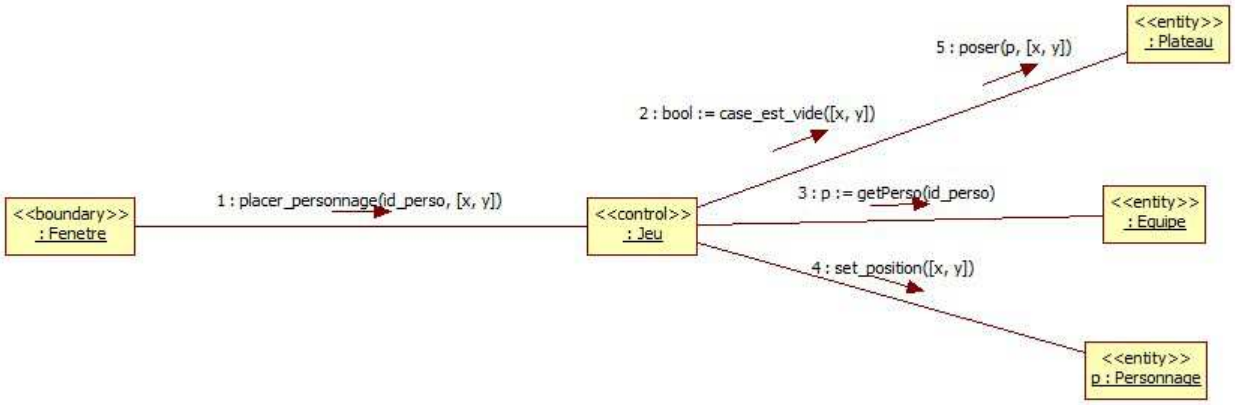
Nous détaillons chaque action prédéfinie ci-dessus dans la boite noire du système avec un diagramme de communication.

Diagramme de communication : Tirage_au_sort() :



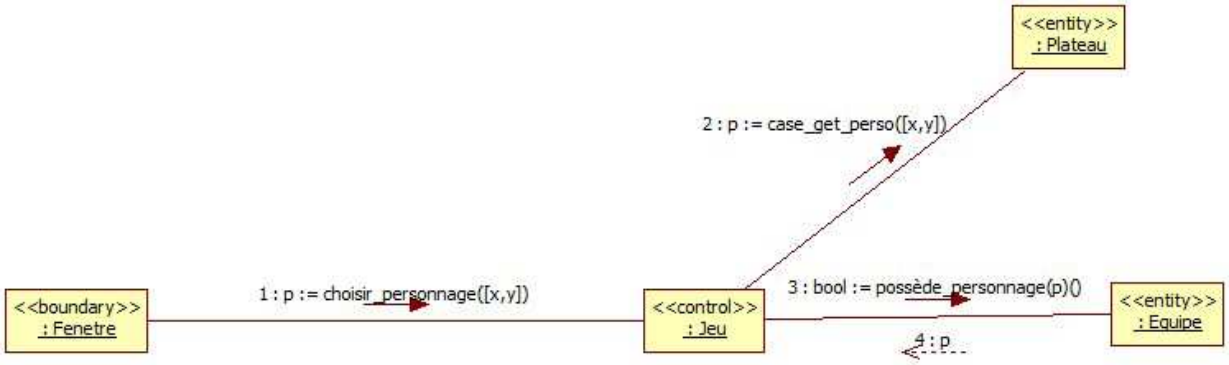
On effectue un tirage au sort par le biais de fonction random() qui nous renverra soit 1 ou 2 pour que l'un des deux joueurs puissent commencer la partie.

Diagramme de communication : Placer_personnage(id_perso,[x,y]) :



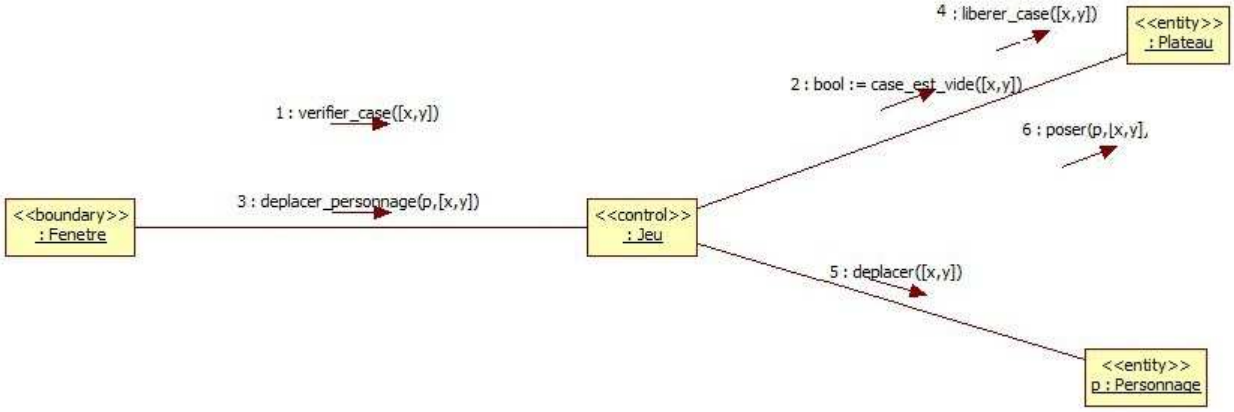
On sélectionne la case [x,y] et si elle est vide alors on pose un personnage sur cette case.

Diagramme de communication : Choisir_personnage([x,y]) :



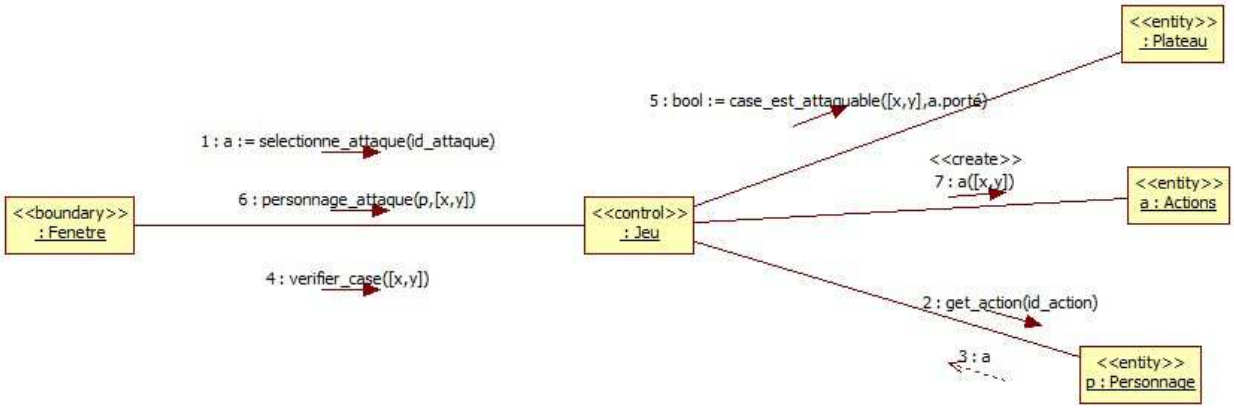
On choisit un personnage situé sur une case. On s'assure que la case n'est pas vide et que le personnage n'appartient pas à l'équipe adverse.

Diagramme de communication : Deplacer_personnage(p,[x,y])



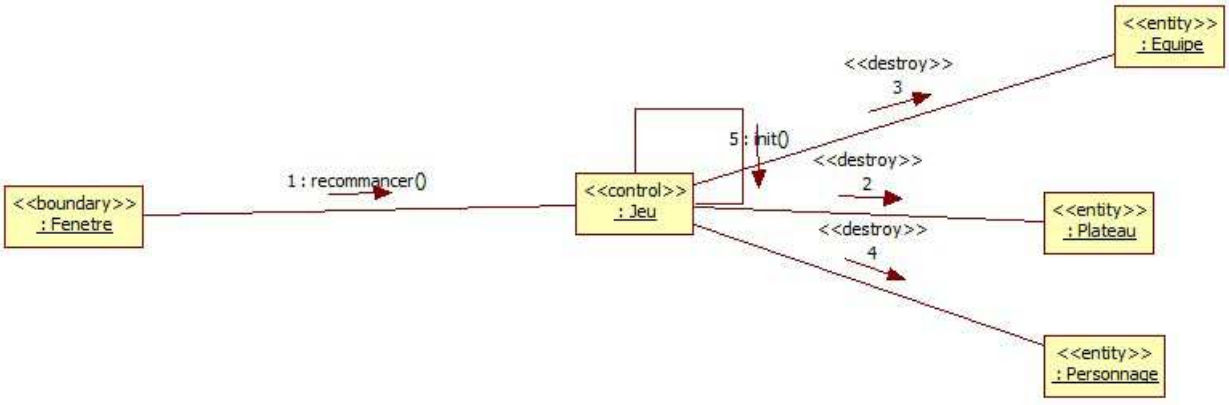
On déplace le personnage, choisi précédemment, situé sur une case du plateau à la case [x,y]. On s'assure que la case sélectionnée est vide.

Diagramme de communication : Personnage_attaque(p,[x,y]) :



Le personnage p lance une attaque sélectionnée sur une case [x,y]. Nous vérifions que la case sélectionnée est attaquable, c'est-à-dire que la portée de l'attaque permet l'attaque sur cette case-ci.

Diagramme de communication : Recommencer() :



Si la partie est terminé, alors le joueur peut effectuer une nouvelle partie. Pour cela, on supprime toutes les instances que l'on a créé et on lance la fonction init().

On en déduit le diagramme de classes d'analyse suivant :

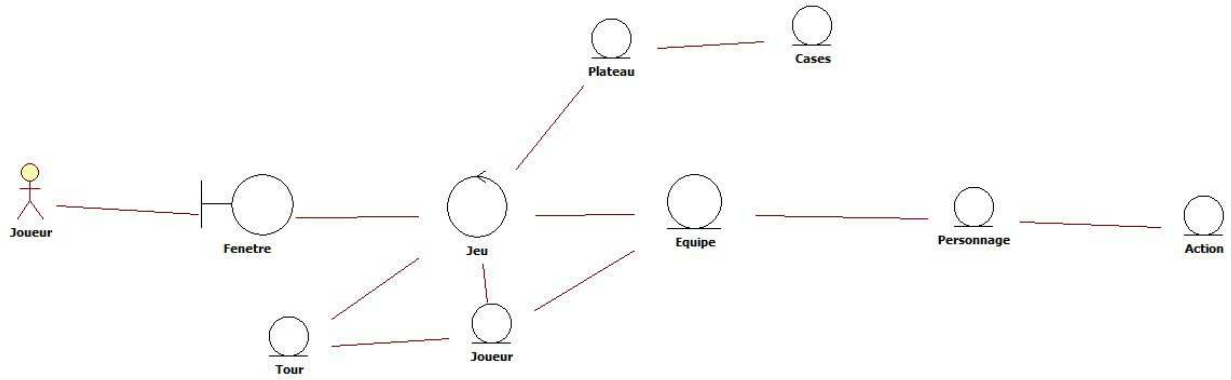


Diagramme de classe d'analyse de l'itération 3

La fenêtre sert d'interface entre le joueur t l'application. Le joueur est présenté comme étant l'utilisateur du système. Et le jeu (ou l'application) est le contrôleur qui gère toutes les autres classes métier (Equipe, Plateau, Cases, Personnage, Action, Joueur, Action et Tour).

C. Conception

Dans cette partie nous allons donner une ébauche de solution, avec des modèles proche de l'implémentation. Nous présenterons un diagramme de séquence contenant toutes les actions de l'itération 3 et un diagramme de classe contenant toutes les méthodes citées auparavant.

Diagramme de séquence :

A partir des diagrammes de communication précédents et de la boite noire, on déduit le diagramme de séquence suivant qui contient le déta

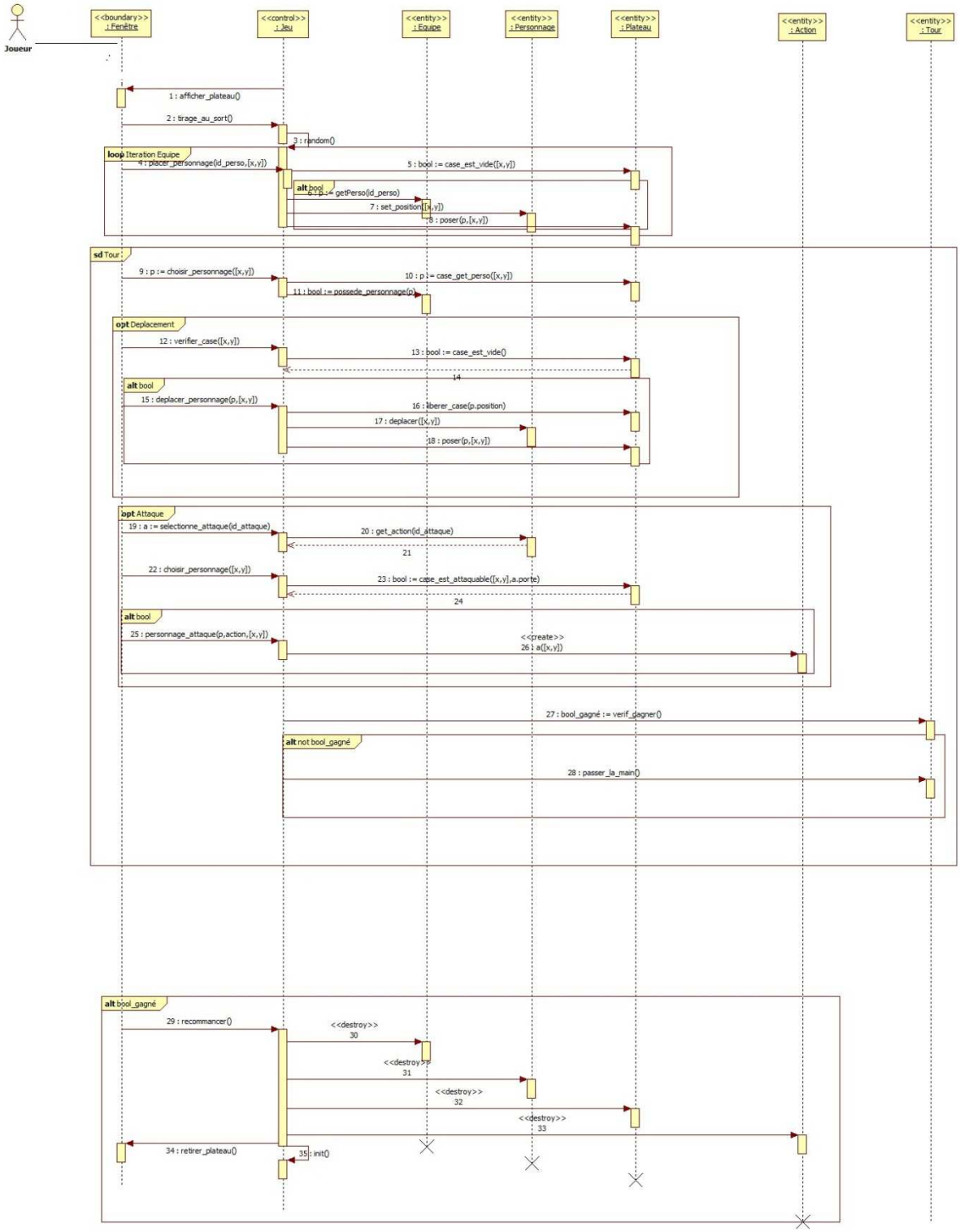
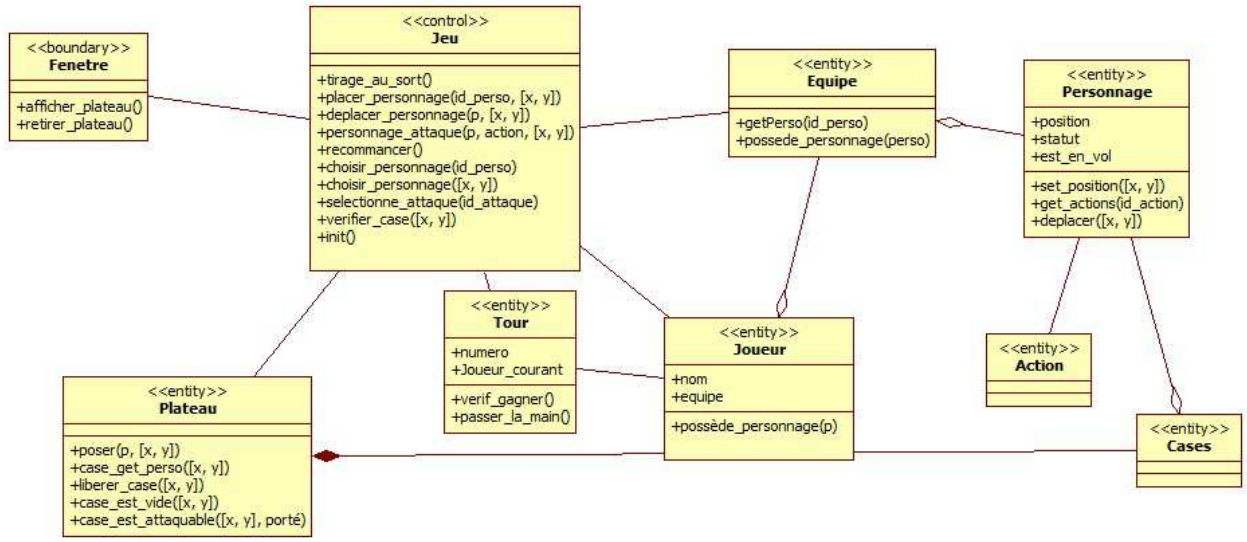


Diagramme de classes :

On obtient alors le diagramme de classes suivant qui résume l'ensemble des classes et méthodes nécessaires pour l'itération 3.



IV. Itération 4 : Consulter les informations du jeu

A. Capture des besoins : descriptions détaillée du cas

Résumé : Le joueur qui joue veut consulter les informations de son personnage, comme son état et ses attaques et veut aussi voir la liste des personnages présents sur le plateau de jeu.

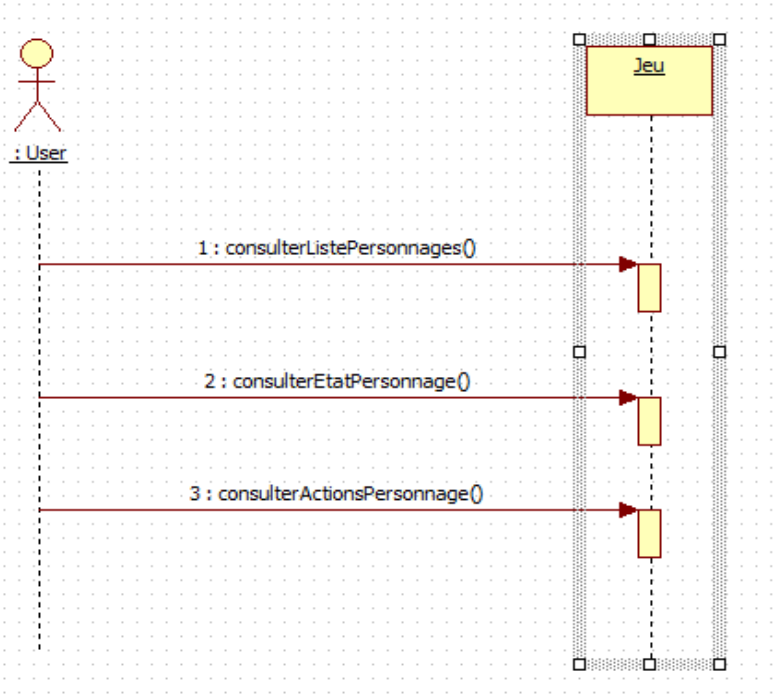
Pré condition : Une partie a été commencée.
Post condition : Retour à la partie.

Scénario principal :
7. Arrivé à son tour de jeu, le joueur appuie sur la fenêtre des personnages
8. Une fenêtre s'affiche avec la liste des personnages présents sur le plateau par équipe.
9. Il appuie ensuite sur la fiche du personnage sélectionné
10. Une fenêtre s'ouvre et affiche son état : point de vie, de mouvement et altérations possibles
11. il appuie ensuite sur la fiche des actions du personnage
12. Une fenêtre s'ouvre et affiche la liste des attaques disponible pour le personnage et leurs effets

B. Analyse

On cherche les instances qui participent activement au cas.

Analyse en boîte noire :



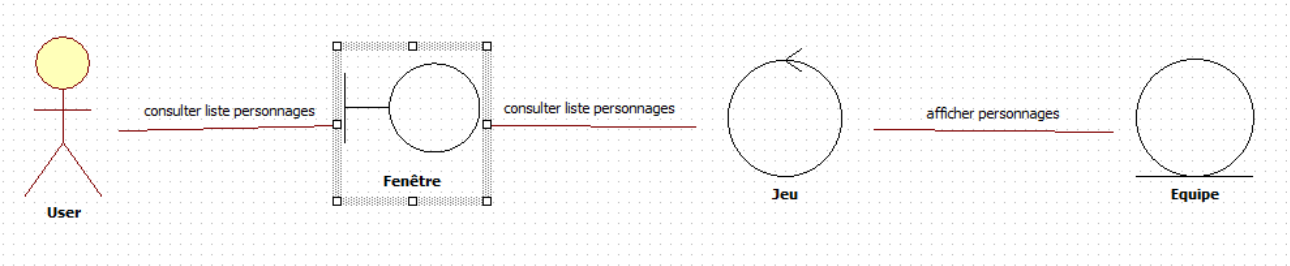
On effectue une première approche du cas de manière peu détaillée. L'utilisateur effectue 3 actions sur le système, ici l'application : il consulte la liste des personnages, l'état du personnage sélectionné et les actions du personnage sélectionné.

Analyse en boîte blanche :

On effectue une analyse plus détaillée du cas en détaillant chaque action avec un diagramme de communication.

Diagramme de communication : consulterListePersonnages( ) :

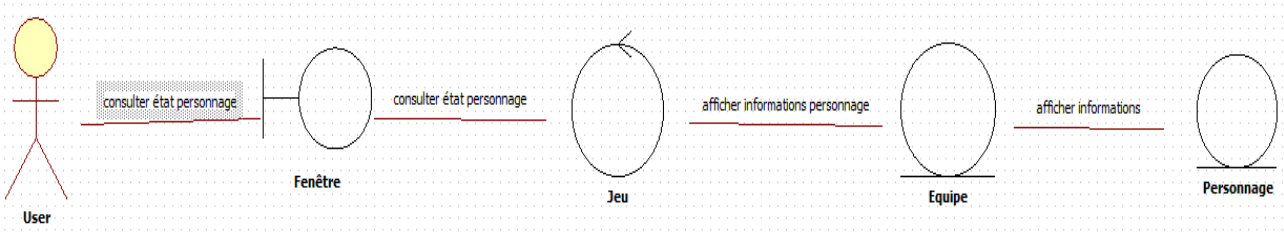
Pour la 1ère action, consulter la liste des personnages, on a le diagramme suivant :



Les instances qui participent à cette action sont : le contrôleur Jeu, l'entité Equipe et l'interface Fenêtre. Le joueur interagit avec la fenêtre et demande à consulter la liste des personnages. La fenêtre transfère cette demande au contrôleur jeu. Celui-ci demande ensuite aux instances de la classe équipe crées d'afficher les personnages qu'elles contiennent.

Diagramme de communication : consulterEtatPersonnage( ) :

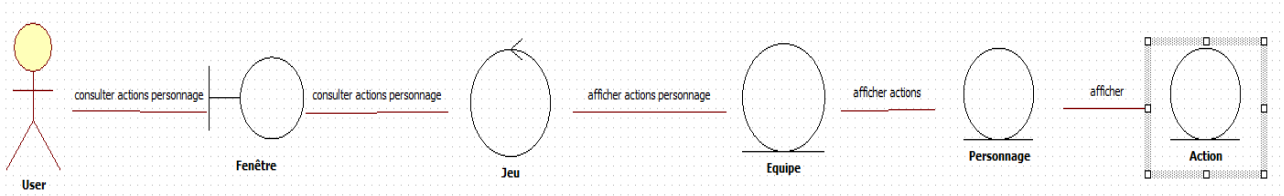
Pour la 2ème action, consulter l'état du personnage, on a le diagramme suivant :



Les instances qui participent à cette action sont : le contrôleur Jeu, l'entité Equipe, l'entité Personnage et l'interface Fenêtre. Le joueur interagit avec la fenêtre et demande à consulter l'état du personnage sélectionné. La fenêtre transfère cette demande au contrôleur Jeu. Celui-ci demande à l'instance de la classe Equipe qui contient ce personnage d'afficher ses informations. Cette instance va alors demander à l'instance de la classe Personnage qui correspond au personnage sélectionné d'afficher ses informations.

Diagramme de communication : consulterActionsPersonnage( ) :

Pour la 3ème action, consulter les actions du personnage, on a le diagramme suivant :



Les instances qui participent à cette action sont : le contrôleur Jeu, l'entité Equipe, l'entité Personnage, l'entité Action et l'interface Fenêtre. Le joueur interagit avec la fenêtre et demande à consulter les actions du personnage sélectionné. La fenêtre transfère cette demande au contrôleur Jeu. Celui-ci demande à l'instance de la classe Equipe qui contient ce personnage d'afficher ses actions. Cette instance va alors demander à l'instance de la classe Personnage qui correspond au personnage sélectionné d'afficher ses actions. Ce dernier va alors demander aux instances de la classe Action qu'il contient d'afficher leurs caractéristiques.

On en déduit le diagramme de classes d'analyse suivant :

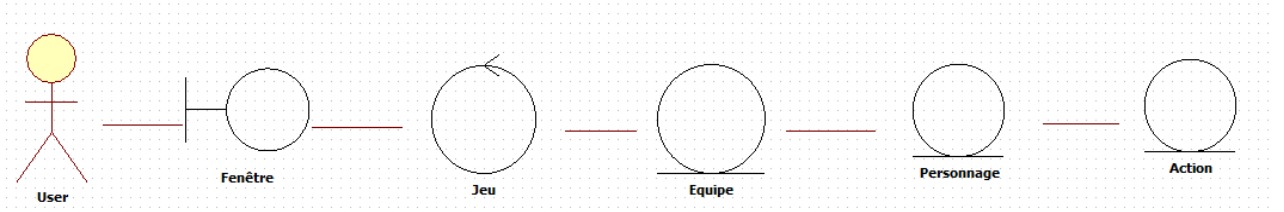


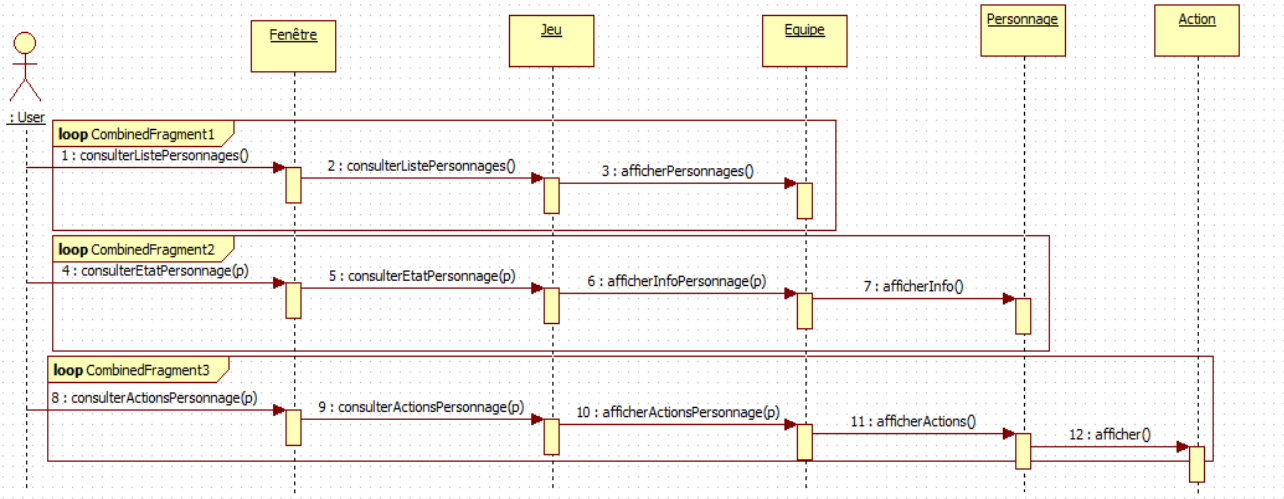
Diagramme de classe d'analyse itération 4

L'utilisateur interagit avec une interface fenêtre pour consulter la liste des personnages, l'état ou les actions d'un personnage sélectionné. Cette interface se charge ensuite de transférer ces demandes au contrôleur Jeu. Celui-ci envoie ensuite ces demandes soient aux instances Equipe crées, soit à l'instance Equipe qui contient le personnage sélectionné. Ce dernier peut alors transmettre la demande aux instances de la classe Action concernée.

C. Conception

Dans cette partie nous allons donner une ébauche de solution, avec des modèles proche de l'implémentation.

Diagramme de séquence :



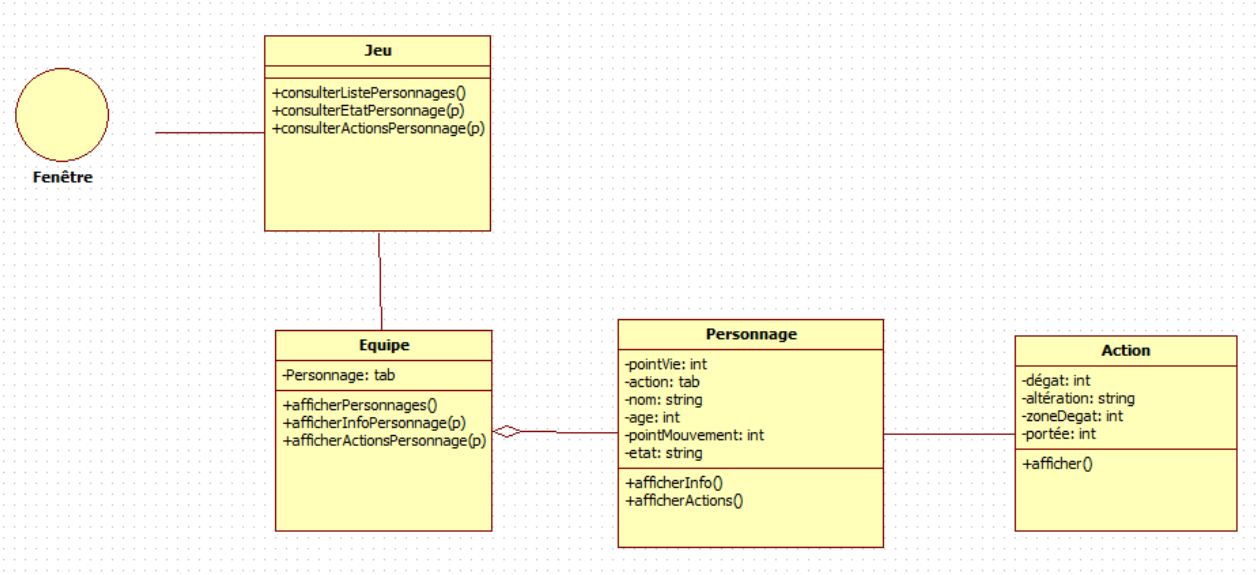
Pour la 1ère action, le joueur qui joue le tour peut demander à consulter la liste des personnages présents sur le plateau en appuyant sur un bouton de l'interface. L'interface transmet alors la demande à l'instance Jeu. Celle-ci demande alors aux deux instances Equipe d'afficher les personnages qu'elles contiennent en faisant appel à la méthode afficherPersonnages() de la classe Equipe.

Pour la 2ème action, le joueur qui joue le tour sélectionne d'abord un de ses personnages encore vivant présent sur le plateau puis demande à consulter son état en cliquant sur le bouton de l'interface. L'interface transmet la demande à l'instance Jeu. Celle-ci transmet la demande à l'instance Equipe du joueur qui contient le personnage sélectionné en faisant appel à la méthode afficherInfoPersonnage(p) qui prend en paramètre le personnage sélectionné. L'instance Equipe fait alors appel à la méthode afficherInfo() de la classe Personnage pour que le personnage sélectionné affiche ses informations.

Pour la 3ème action, le joueur qui joue le tour sélectionne d'abord un de ses personnages encore vivant présent sur le plateau puis demande à consulter ses actions en cliquant sur le bouton de l'interface. L'interface transmet la demande à l'instance Jeu. Celle-ci transmet la demande à l'instance Equipe du joueur qui contient le personnage sélectionné en faisant appel à la méthode afficherActionsPersonnage(p) qui prend en paramètre le personnage sélectionné. L'instance Equipe fait alors appel à la méthode afficherActions() de la classe Personnage pour que le personnage sélectionné affiche ses actions possibles. Celui-ci fait alors appel à la méthode afficher() de la classe Action pour que les actions qu'il possède affiche leurs caractéristiques.

Diagramme de classes :

On obtient alors le diagramme de classes suivant :



V. Itération 5 : Sauvegarder la partie

A. Capture des besoins : description détaillée du cas

Résumé : Les 2 joueurs peuvent sauvegarder la partie à tout moment du jeu et charger une partie au démarrage.

Pré condition : Le nombre de personnages par équipe et la taille du plateau de jeu ont été fixé.
Post condition : La partie est sauvegardée ou chargée.

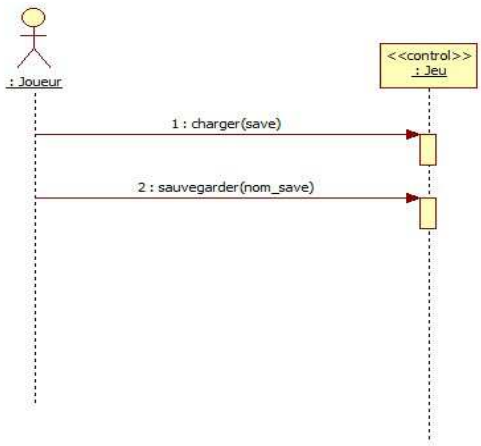
Scénario principal :

1) Le joueur peut charger une partie pré enregistrée.
2) Le joueur peut sauvegarder sa partie.

B. Analyse

On cherche les instances qui participent activement au cas.

Analyse en boîte noire :



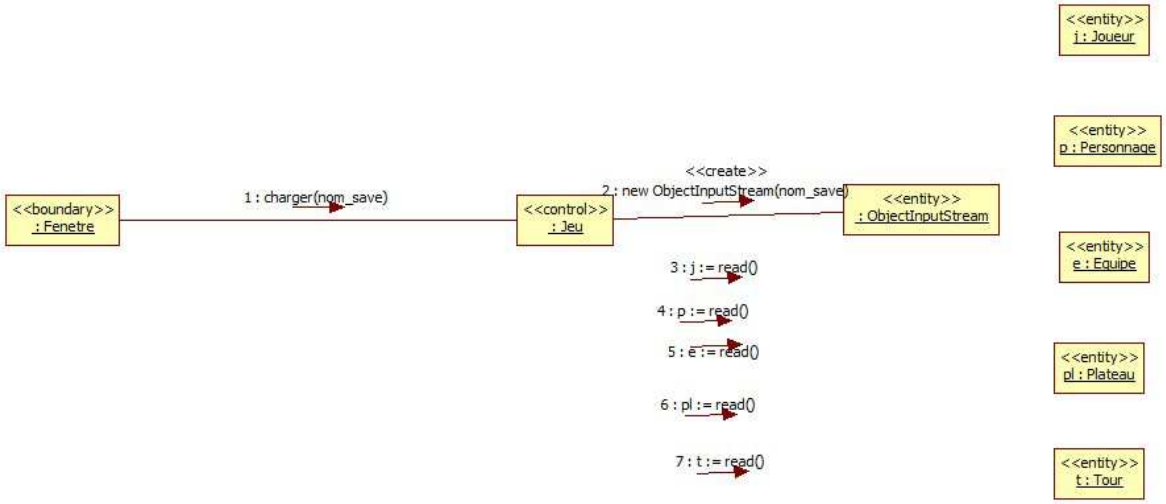
Nous avons détaillé les principaux services de cette itération que le système offre aux joueurs par le biais d'un message :

1. **charger(save)** : Charge la partie à partir du fichier « save ».
2. **sauvegarder(nom_save)** : Sauvegarde la partie en cours dans un fichier nommé « nom_save ».

Analyse en boîte blanche :

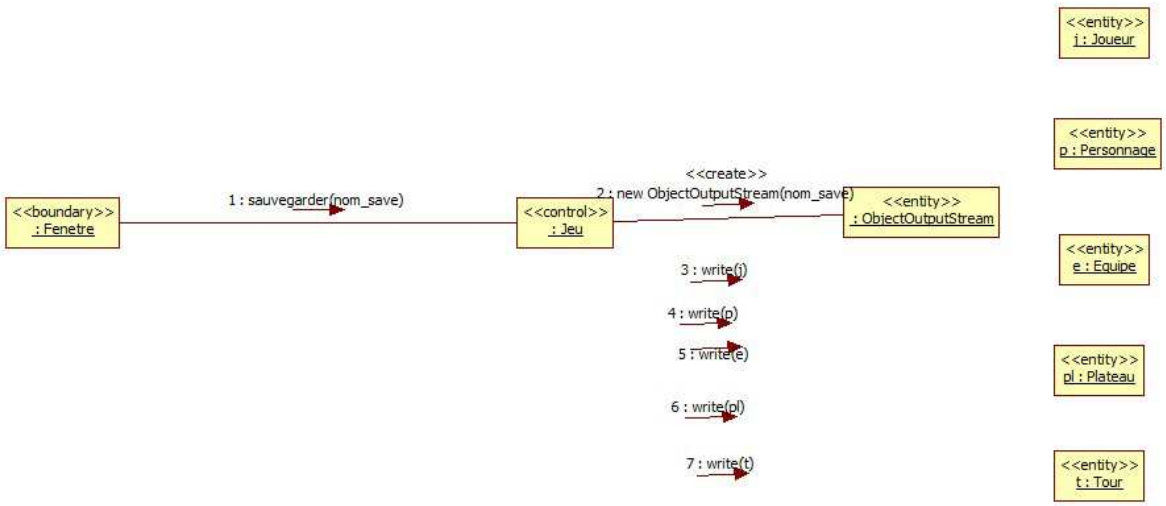
On détaille chaque action avec un diagramme de communication.

Diagramme de communication : charger(save) :



On crée une instance de la classe ObjectInputStream en passant comme paramètre « nom_save » dans son constructeur puis on appelle la méthode read() qui permet de lire séquentiellement les instances sauvegardées dans ce fichier et on les enregistre dans les instances correspondant.

Diagramme de communication : sauvegarder(nom_save)) :



On crée une instance de la classe ObjectOutputStream en passant comme paramètre « nsave » dans son constructeur puis on appelle la méthode write() qui permet d'écrire séquentiellement les instances à sauvegarder dans ce fichier.

On en déduit le diagramme de classes d'analyse suivant :

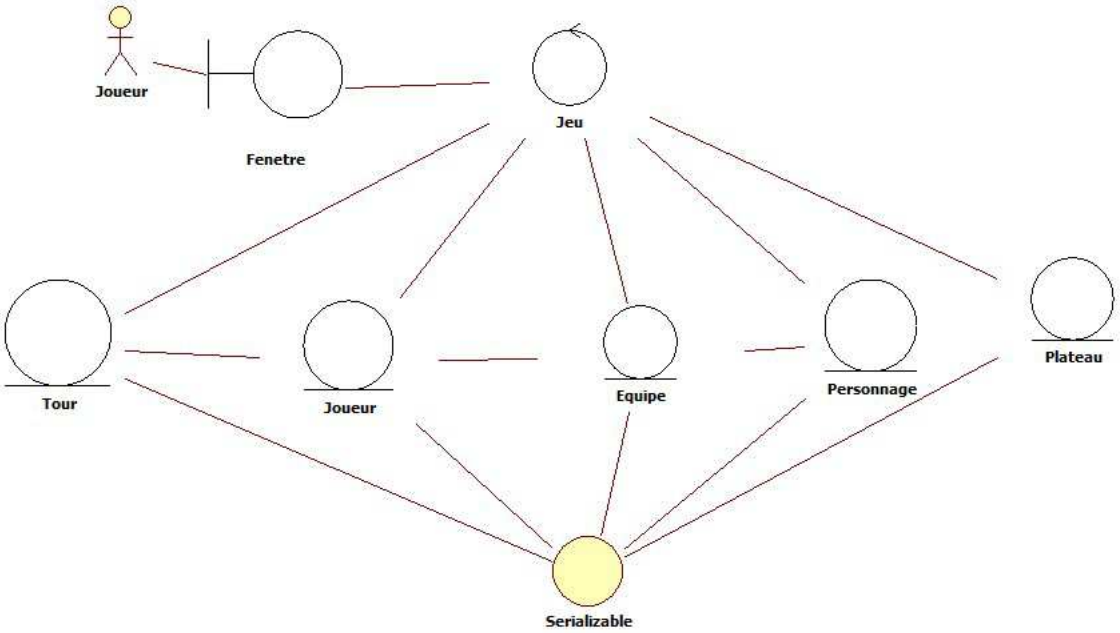


Diagramme de classe d'analyse de l'itération 5

Pour sauvegarder ou charger des instances, il faut que les classes métiers implémentent la classe Serializable.

Le diagramme de classe de l'itération 5 ressemble à celui de l'itération 2. Il suffit juste de changer le nom de quelques méthodes.

Modèle analytique : Diagramme de classe d'analyse général du « jeu fantastique »

A partir des différents diagrammes de classes d'analyse pour les différentes itérations, on obtient le diagramme de classe d'analyse suivant :

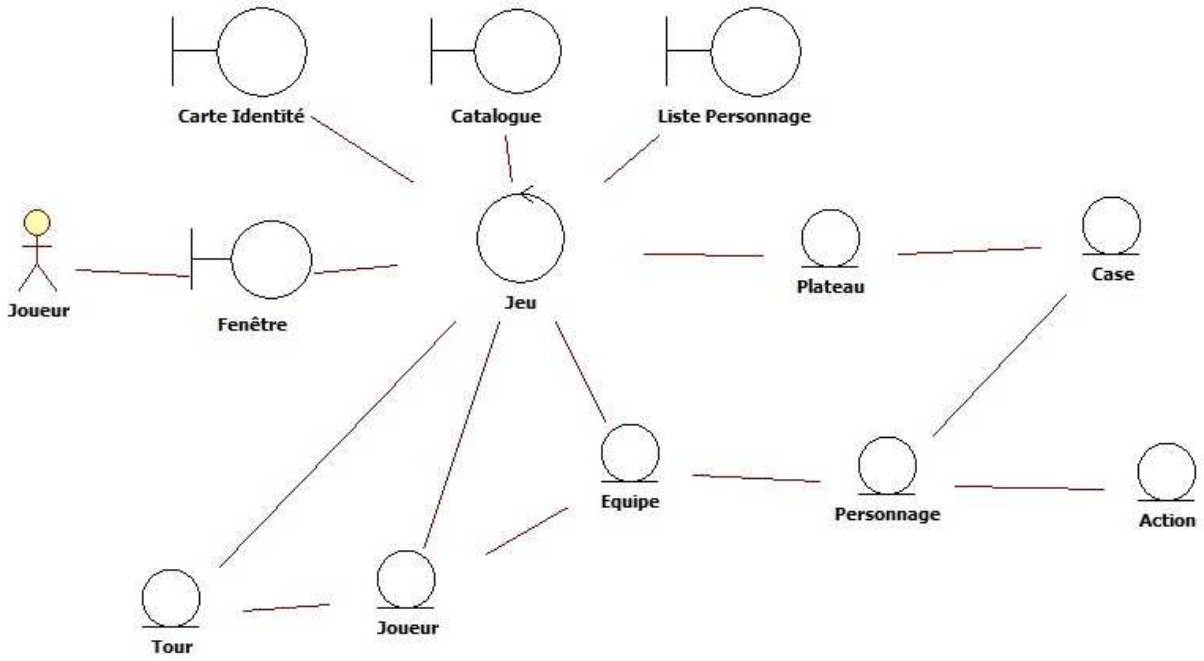


Diagramme de classe d'analyse général du système

Modèle de conception : Diagramme de classe général du « jeu fantastique »

A partir des différents diagrammes de classe obtenue à la fin de l'analyse détaillée de chaque itération, on peut construire un diagramme de classe global du système, aussi appelé modèle de conception. Le modèle de conception du système étudié est le suivant :

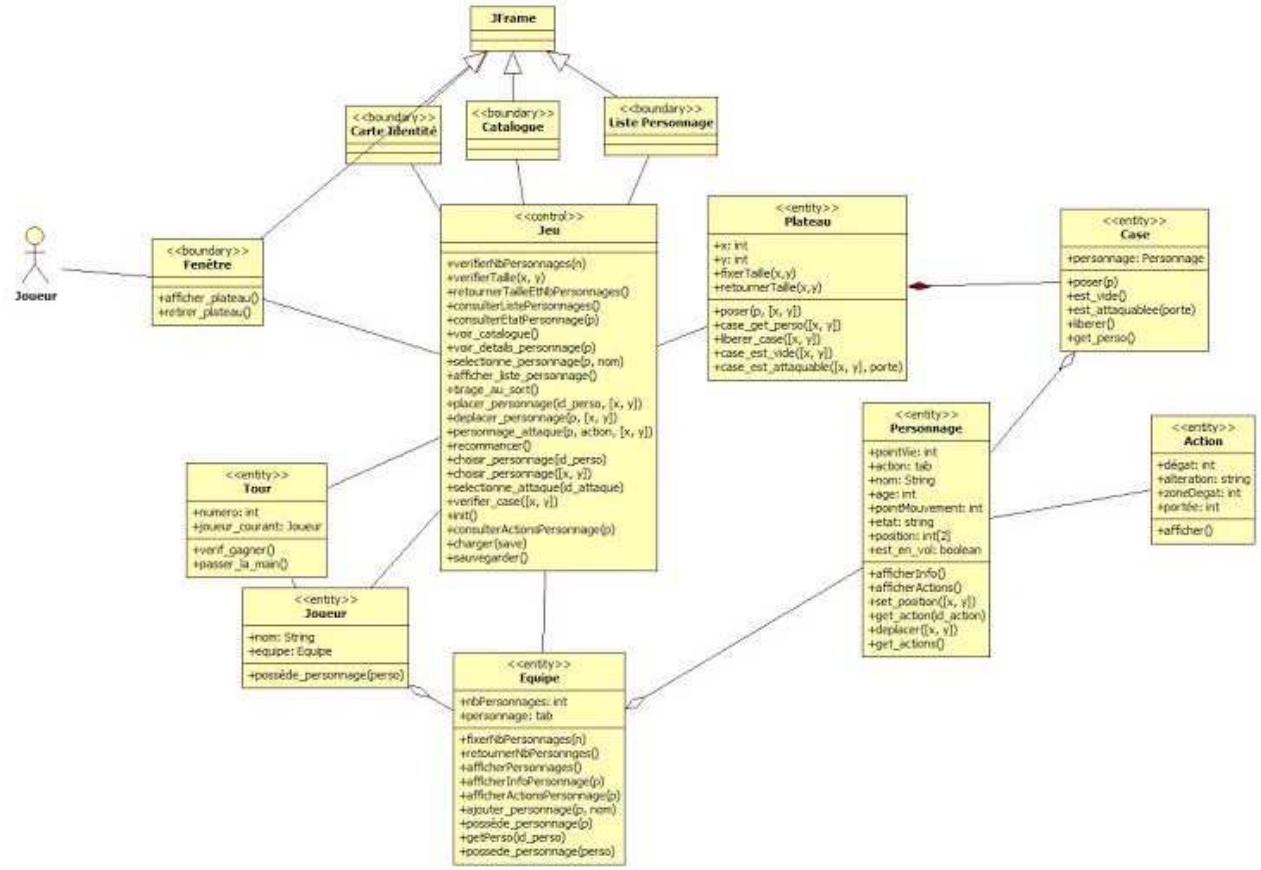


Diagramme de classe général de l'application

Ce modèle de conception nous sera très utile lors de l'implémentation du système en Java. Par ailleurs, pour l'implémentation du système, nous envisageons d'utiliser certains design patterns. Par exemple nous envisageons d'utiliser le design pattern Singleton pour l'instance plateau car il ne doit y avoir qu'un seul plateau de jeu. Nous envisageons aussi d'utiliser le design pattern Observer/Observable pour mettre à jour les données des objets personnages lors d'une attaque par exemple.

Conclusion

Ce projet nous a permis de mettre en pratique les différentes étapes de modélisation vu en cours, ainsi que l'utilisation des différents diagrammes statiques et comportementaux qui nous permettent de faire l'analyse et la conception du système. Vu la simplicité de l'outil « StartUML », nous n'avons pas eu de problème à le manipuler et à mettre en place nos diagramme, la familiarisation avec le logiciel a été faite très rapidement. Nous avons même eu le temps de faire l'analyse et la conception de l'itération 5 (sauvegarder la partie), que nous avons considéré comme étant facultative puisqu'elle n'empêche pas la bonne organisation du jeu, mais elle rend le rendu de l'application meilleur.

En étudiant d'abord le système de manière globale, afin de dégager les différentes itérations, puis en étudiant en détail chacune de ces itérations, nous sommes parvenus à produire un modèle de conception pour le système. Cela va nous permettre de mieux appréhender l'implémentation du système en Java car nous savons comment doit se comporter le système, à la fois de manière globale et de manière spécifique. Nous avons ainsi une vue d'ensemble du système et une vue détaillée de chacun des cas que comporte le système. Nous avons ainsi une base pour l'implémentation en Java.