

Projet Java

« Jeu fantastique »

12/18/2014

KALWANT Steven
skalwant@enssat.fr

LATNI Sofia
slatni@enssat.fr

MOUGAMADOU Javid
jmougama@enssat.fr

LSI2

Encadrés par Dr Gwénolé Lecorvé

Résumé

Ce document permet de comprendre les choix d'implémentations que nous avons mis en place pour développer un le jeu « Bataille Fantastique » dont la conception a été faite dans la partie UML. Ce projet représente une traduction et une implémentation des diagrammes de classes et de séquences ainsi que tous les autres diagrammes utilisés dans la phase de conception.

En effet, nous avons traité en premier abord la description du projet, puis sa planification à l'aide d'un diagramme de Gantt, ensuite les aspects techniques mis en œuvres (nous avons expliqué les aspects techniques itération par itération), pour expliquer par la suite le déroulement de notre partie de test et un bilan général sur notre projet. Pour finir avec une conclusion résumant le document et une perspective.

Sommaire

Introduction

I.	Présentation du projet.....	5
II.	L'organisation du projet	6
III.	Présentations fonctionnelle et technique de l'application	6
A.	Aspects fonctionnels.....	6
B.	Positionnement au regard de la phase de conception	7
C.	Aspects techniques	7
1.	Itération 1 : Paramétrer le jeu	8
2.	Itération 2 : Composition de l'équipe	9
3.	Itération 3 : Jouer la partie	10
4.	Itération 4 : Consulter les informations du jeu	12
5.	Itération 5 : Sauvegarder la partie.....	13
D.	Recette	14
E.	Bilan	15

Conclusion

Introduction

Le projet jeu « Bataille Fantastique » consiste à mettre en œuvre toutes les étapes nécessaires pour que 2 joueurs puissent jouer une partie du jeu avec un enchainement des tours jusqu'à ce que la partie finisse. En effet, la première partie du projet avait pour but de mettre en place les concepts d'itérations et d'activités (analyse, conception, ébauche d'implémentation). En conséquence, en partant du diagramme de classes général et du diagramme de séquence général établis durant la première partie de conception, nous pouvons implémenter et tester le système étudié et modélisé en première partie puis implémenter des interfaces graphiques pour l'affichage du déroulement du jeu.

Ce rapport va mettre en lumière les différentes étapes d'implémentation en commençant par l'implémentation des différentes classes puis la création des IHM. Le plan de notre rapport se base sur cinq axes principaux qui traitent de la description du projet, la planification du projet, la présentation fonctionnelle et technique de notre application, la phase du test du projet, et le bilan du projet.

I. Présentation du projet

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, choisit un personnage attaquant et le genre d'attaque souhaité.

Récapitulatif de la modélisation UML :

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

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

Nous avons décidé de distinguer 5 itérations présentées comme ceci :

1^{ère} itération : Paramétrer le jeu

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

2^{ème} itération : Constituer son équipe

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

3^{ème} itération : Jouer la partie

- Tirer au sort le 1^{er} 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

4^{ème} itération : Sauvegarder le jeu

- Garder la progression de la partie en quittant le jeu

5^{ème} itération : Consulter les informations du jeu

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

Phase de construction	Priorité	Ordre
Paramétrer le jeu	Primordiale	1
Constituer son équipe	Primordiale	2
Jouer la partie	Primordiale	3
Sauvegarder le jeu	Secondaire	5
Consulter les informations du jeu	Essentielle	4

Figure n°1: Ordres et priorités des itérations dans la phase conception

II. L'organisation du projet

Pour organiser le déroulement du projet nous avons mis en place un diagramme de Gantt afin de rendre le projet réalisable dans les temps et évoluer d'une manière organisée. Les tâches mentionnées dans le diagramme de Gantt ci-dessous représentent les itérations identifiées lors de la phase de conception.

Notons que l'ordre d'exécution des tâches a connu pendant la phase d'implémentation un léger changement. En effet, nous avons commencé par développer le code de l'itération n°1 « paramétrage du jeu », puis le développement de l'itération n°2 de « Constitution des équipes ». Et en parallèle nous avons décidé de créer les interfaces graphiques qui nous permettront de tester ce qui a été implémenté sur Eclipse.

Il était important pour nous de créer ces interfaces avant de développer les autres itérations restantes, car nous voulions être sûrs que le travail déjà réalisé avait le rendu souhaité.

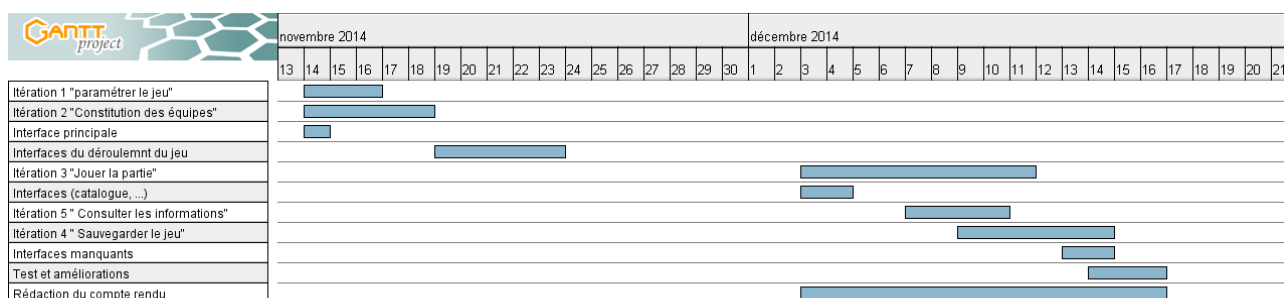


Diagramme de Gantt du déroulement du projet

III. Présentations fonctionnelle et technique de l'application

A. Aspects fonctionnels

Le système étudié implémente plusieurs fonctionnalités mises à disposition des deux joueurs.

L'application doit permettre aux deux joueurs de pouvoir choisir la taille du tableau ainsi que le nombre de personnages avec lesquels ils pourront jouer une partie. Elle doit permettre aussi à chacun des deux joueurs de consulter le catalogue pour choisir les personnages voulus (en respectant des conditions : pas plus de 2 personnages du même type et le nombre total des personnages est exactement égale au nombre choisi au début). Pour faire ce choix, nous avons mis en place un catalogue qui contient les différents personnages, chaque personnage ayant une carte d'identité qui précise les attaques qu'il peut faire ainsi que ses points de vie. Le catalogue contenant les différents personnages nous permet de réaliser les actions suivantes :

- Ajouter un personnage à la liste des personnages déjà choisis (on coche juste une case ou les deux).
- Supprimer un personnage que nous avons déjà ajouté à la liste.

Le système doit aussi permettre aux deux joueurs de jouer une partie (qui est une succession de tours). Chaque joueur devra au début de la partie placer ses personnages sur le plateau. Quand un joueur veut lancer une attaque l'application lui affiche la portée de son attaque et ce n'est qu'après qu'il confirme l'attaque ou il choisit une autre attaque. Lorsqu'une attaque est effectuée, le système se charge de mettre à jour les points de vie et enlève du plateau les personnages qui ont pour point de vie 0.

A la fin du jeu, le programme affiche le nom du gagnant et demande au joueur s'ils veulent commencer une nouvelle partie.

B. Positionnement au regard de la phase de conception

La figure n°1 montre notre choix d'ordre de priorité des itérations lors de la phase de conception.

En effet, à chaque itération nous affectons un ordre de priorité. Pendant la phase d'implémentation, l'ordre d'exécution des itérations était le même, car c'est l'enchaînement dans le jeu qui nous oblige à garder cet ordre-là pour que les tests se fassent à fur et à mesure.

Par contre, nous avons effectué quelques modifications dans les différentes itérations. Ces modifications seront détaillées par la suite dans chacune des itérations.

Nous avons utilisé plusieurs interfaces graphiques qui permettent d'interagir avec les joueurs pour avancer dans la partie. Les différentes interfaces vont être détaillées par la suite dans la partie des aspects techniques.

C. Aspects techniques

Nous avons mis en œuvre le design pattern de type M-V-C (modèle-vue-contrôleur) car l'organisation d'une interface graphique est délicate. L'idée est de bien séparer les données, la présentation et les traitements. Ce choix de design pattern est dû au fait qu'il apporte de la clarté par l'architecture qu'il impose. Cela simplifie la tâche du développeur qui tenterait d'effectuer une maintenance ou une amélioration sur le projet.

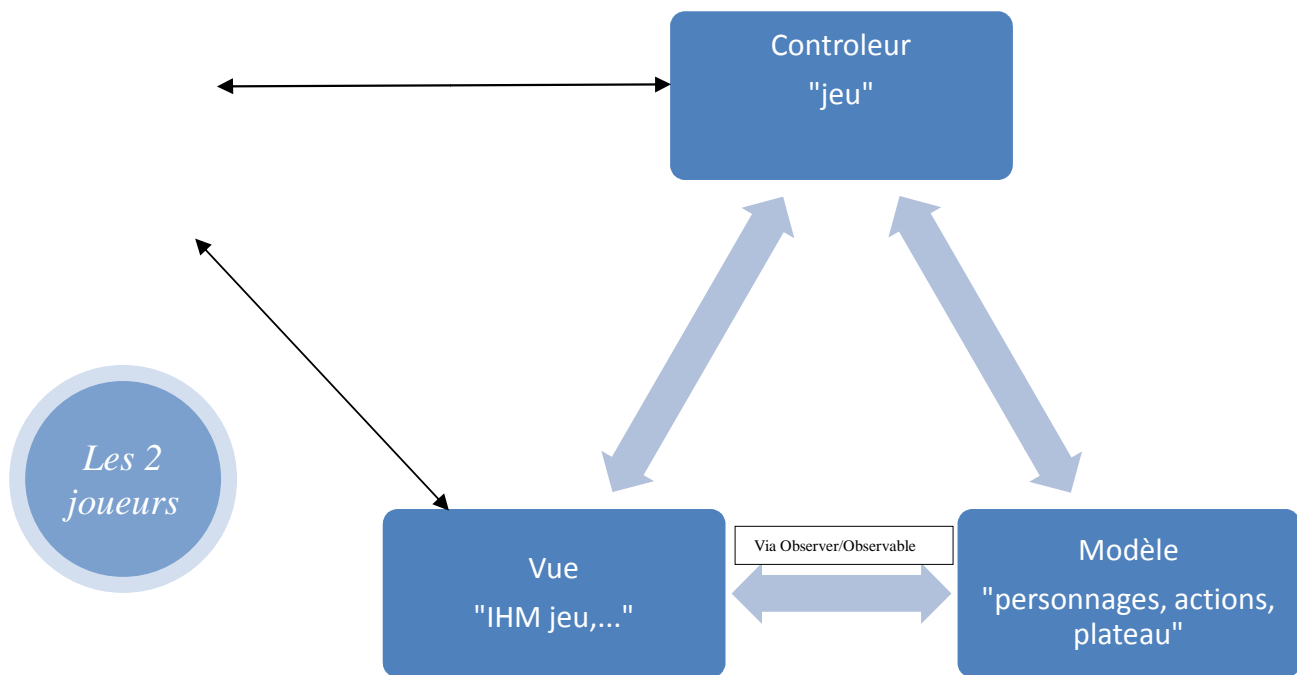


Figure n°2 : Le design pattern MVC

La figure n°2 regroupe les fonctions nécessaires en 3 parties :

- Le modèle : contient les différentes classes utilisées pour modéliser l'application
- La vue : les différentes interfaces graphiques utilisées pour l'interaction avec les joueurs.
- Le contrôleur (ici les contrôleurs : C'est la classe principale « **jeu** » qui gère le tout et est en communication avec toutes les autres classe.

1. Itération 1 : Paramétrer le jeu

Le déroulement :

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 10. Si non il demande de recommencer.
6. Un message s'affiche à la fin pour dire le nombre de personnages par équipe et la taille du plateau.

Par rapport à la partie UML, la taille maximale du plateau a été modifiée. Pour ne pas avoir un tableau de très grande taille, celle-ci a été limitée à 150 cases au maximum (15 ligne et 10 colonnes).

Pour paramétrer le jeu, c'est la classe contrôleur « jeu » qui teste si les valeurs entrées par les joueurs sont valides sinon le jeu envoie un message d'erreur et demande de rentrer de nouvelles valeurs. Pour tester si les valeurs sont correctes nous avons implémentés 2 classes d'exceptions « TaillePlateauIncorrect » et « NbPersonnageIncorrect » qui détectent le non respect des conditions imposées (méthode jeu.verifierTaille(int X, int Y) et jeu.verifierNbPersonnagesEquipe(int n)).

Pour garantir cette interaction entre le jeu et les joueurs nous avons mis en place plusieurs interfaces graphiques qui héritent de la classe « JFrame » qui existe déjà sur « Eclipse ». Ces interfaces graphiques sont les suivantes :

- **FenetrePrincipale** : C'est la fenêtre de démarrage du jeu et qui contient deux boutons qui permettent de passer à d'autres interfaces.
- **FenetreSelectionNb** : C'est une fenêtre qui demande aux joueurs de saisir le nombre de personnages souhaité et qui renvoie une erreur dans le cas du non respect des limitations.
- **FenetreSelectionTaille** : C'est une fenêtre qui demande aux joueurs de saisir la taille du tableau et renvoie une erreur dans le cas du non respect des limitations.
- **FenetreRecapitulatif** : elle affiche après la fin de la saisie les valeurs choisies par les joueurs avant de passer à la prochaine étape.

2. Itération 2 : Composition de l'équipe

Le déroulement :

- 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.

Par rapport à la partie UML, nous avons rajouté une fenêtre où chacun des deux joueurs peut rentrer son nom ou pseudo qu'il souhaite utiliser durant la partie.

Pour constituer les équipes, nous avons utilisé les deux classes modèles « Catalogue » et « CarteIdentite ». Ces deux classes sont des JFrames utilisées par le contrôleur « jeu ». Dans ces deux classes nous avons aussi fait appel à plusieurs méthodes de la classe « Jeu » et nous avons utilisé les classes du package « Personnage » dans nos deux classes « Catalogue » et « CarteIdentite ».

En effet, ces deux classes qui héritent de JFrame sont deux interfaces graphiques :

- **Catalogue** : c'est la fenêtre de sélection des personnages qui vont constituer une équipe, cette interface graphique utilise des JCheckBox. Le bouton suivant (pour passer à la prochaine étape) n'est valide que lorsque le nombre de JCheckBox coché est égale au nombre de personnages choisi auparavant dans l'itération 1 (si on est en dessous de ce nombre ou quand on le dépasse on ne peut pas appuyer sur *suivant* pour passer à l'étape d'après). C'est l'interface qui permet d'éviter la création d'une exception si le joueur choisi un nombre de personnage différent du Nb personnages par équipe. Nous avons choisi comme contrainte d'avoir au maximum deux personnages du même types dans une même équipe (2 Magiciens dans une même équipe : oui mais 3 Magiciens dans une même équipe : non).
- **CarteIdentite** : durant la phase de sélection des personnages qui vont constituer une équipe, les joueurs ont la possibilité de cliquer sur un personnage pour consulter sa carte d'identité qui contient des informations relatives à ses points de vie, le genre d'attaque qu'il peut lancer... Cette interface graphique utilise des méthodes des différentes classes qui se trouvent dans le package « Personnage ». Nous avons utilisé des variables 'static' pour les Cartes d'identité.

- **FenetreListePersonnage** : Nous avons ajouté cette interface graphique qui permet au joueur, après création de l'équipe, de donner des surnoms à ses personnages s'il le souhaite.

3. Itération 3 : Jouer la partie

Le déroulement

- 1) Le système fait un tirage au sort puis affiche le joueur qui démarre la partie. Affiche ensuite le bouton « démarrer le jeu ».
- 2) L'un des joueurs appuie sur le bouton.
- 3) Le système affiche pour chaque joueur la liste des personnages choisis.
- 4) Le système affiche le plateau du jeu.
- 5) Chaque joueur place ses personnages sur le plateau.
- 6) Le système affiche le plateau avec les personnages dessus.
- 7) Le 1^{er} joueur joue avec un personnage de son choix en précisant l'attaque voulue.
- 8) Le système exécute l'attaque.
- 9) Le système affiche les changements après l'attaque et vérifie si la partie est finie avant le début d'un prochain tour.
- 10) Déclare la partie finie quand un personnage perd tous ses joueurs.
- 11) Réinitialise le jeu et ses paramètres.

Par rapport à la partie UML, nous avons rajouté quelques méthodes et variables statiques dans l'itération 3. Cet ajout s'est imposé à fur et à mesure de notre implémentation.

C'est la partie la plus conséquente du projet en termes de temps d'implémentation et de contenu. C'est dans cette partie que nous avons implémenté la classe contrôle « Jeu » ainsi que d'autres classes modèles et vue.

- **La classe « contrôle » :**

- **Jeu** : C'est la classe principale qui est en lien avec toutes les autres classes et qui contient le corps de l'application. Elle contient séquentiellement les différentes étapes de notre jeu, et jouer une partie consiste à instancier cette classe en créant un objet de type « Jeu ».

- **Les classes « vue » : Les différentes IHM**

- **IHMJeu** : C'est l'interface graphique principale de notre Jeu Bataille Fantastique. elle est composée de l'IMHPlateau, IHMBandeauJoueur et des IHMCommande (Vue1, Vue2 et Vue3), une explication plus détaillée sur son fonctionnement est donnée après le listing des différentes classes utilisées.
- **IHMPlateau**: C'est l'interface graphique affichant le plateau. Il est constitué d'un tableau de IHMCase

- **IHMCase:** C'est l'interface graphique affichant une case du plateau. Chaque IHMCase est reliée à une Case (par le biais de l'implémentation du Design Pattern Observer/Observable)
 - **IHMBandeauJoueur:** Il s'agit des détails et des caractéristiques concernant un personnage de l'équipe du joueur. Celui ci est composé d'un IHMPersonnageInfo et d'un IHMListeAttaque et d'un IHMListePersonnages.
 - **IHMPersonnageInfo:** Lorsqu'une case contenant un personnage est cliquée, elle affiche les caractéristiques du personnage.
 - **IHMListeAttaque :** Il affiche les attaques possibles d'un personnage qui est sélectionné dans le IHMPersonnageInfo
 - **IHMCommandeVue1:** Il s'agit d'une interface graphique permettant de sélectionner des personnages et de les placer sur le plateau
 - **IHMCommandeVue2 :** Il s'agit d'une interface graphique où le joueur peut sélectionner l'une des commandes suivantes :
 - Déplacer
 - Action
 - Fin du Tour
 - **IHMCommandeVue3 :** Il s'agit d'une interface graphique qui est activé uniquement si le joueur sélectionne une action, celui ci peut choisir quelles actions parmi celles disponibles pour le personnage et lancer cette action sur une case.
- **Les classes « Modèle » (ou Dur) :**
 - **Tour :** classe contenant les informations du tour actuelles (joueurCourant, actions, déplacement, etc...)
 - **Joueur (ou Equipe) :** lorsqu'on va sélectionner les personnages pour effectuer une commande
 - **Les Classes Personnages :** les différentes classes qui correspondent aux personnages que nous avons.
 - **Les Classes Actions :** Les différentes classes des actions que les personnages peuvent effectuer.

Pour notre interface graphique du jeu (IHMJeu), nous nous sommes basés sur l'interface graphique donnée dans l'énoncé du projet et nous l'avons modifié et customisé de manière à avoir une interface de jeu plus attirante, dynamique et surtout très simple à utiliser avec la plupart des commandes et des informations sur cette interface sans passé par une autre fenêtre. Cette interface graphique est constituée de :

- **Un terminal :** Il se situe en haut du plateau (sert à orienter les joueurs) et possède la plupart des informations concernant les phases principales du Jeu (sélection des personnages / déplacement, etc..) et également la plupart des messages d'erreur.

- **Deux Bandeaux Joueur:** (à gauche et à droite du plateau) chaque bandeau possède la plupart des informations concernant les personnages de l'équipe du joueur qui se trouve du même côté que la bande.
- **Des Commandes :** qui se situent en bas du plateau, le joueur peut sélectionner la commande qu'il veut et peut suivre les aides qui s'affichent sur le terminal en haut de la page pour l'orienter.

Remarques importantes sur l'itération 3 :

- ❖ Lorsqu'un joueur sélectionne un personnage pour effectuer un déplacement ou une attaque, il a la possibilité d'annuler le choix de ce personnage et de choisir un autre personnage tant qu'il ne l'a ni déplacer ni effectuer une attaque avec. Pour ce faire, il doit juste cliquer sur le même bouton.
- ❖ Lorsqu'un joueur choisit un personnage pour le déplacer, il voit afficher sur le plateau en vert les cases où il peut le placer, cette portée de déplacement dépend des points de mouvement du personnage choisi. Si le joueur appuie sur une case verte, alors son déplacement est valide et est effectué, sinon aucun déplacement n'est effectué et un message lui indiquant qu'il a choisi une mauvaise case lui est affiché en haut de la fenêtre.
- ❖ Lorsque le joueur choisit un personnage puis l'une de ses actions, il voit affiché en rouge sur le plateau les cases où l'attaque choisie du personnage choisi peut arriver, ceci dépend de la portée de l'attaque. Pour valider son attaque le joueur doit sélectionner l'une des cases rouges, dans le cas contraire aucune attaque n'est faite et un message indiquant une erreur est affiché en haut de la fenêtre.
- ❖ Pour mettre en œuvre ces deux dernières remarques, nous avons ajouté des exceptions (ceci sera précisé dans une autre partie qui explique les exceptions choisies) et de plus l'affichage du message d'erreur se fait avec un PopUp ou sur le terminal (en haut de la fenêtre).
- ❖ Le « Coup de Jarnac » (*jeu.hasMultipleAction(this.nbMultipleAction)*) est une attaque spéciale car elle peut être utilisée deux fois dans le même tour.
- ❖ Un personnage peut être au ralenti ou protégé grâce à l'une de ses actions (comme Instinct d'esquive), pour modéliser ceci nous avons rajouté des flags booléens pour savoir si les personnages sont protégés ou ralentis mais aussi le numéro du tour où l'effet sera perdu. À chaque fin de tour, une fonction vérifie les flags et les met à jour en fonction du numéro du tour actuelle (*jeu.majEtatPersonnage*).
- ❖ Lorsqu'un personnage se déplace ou est retiré du plateau, nous utilisons le Design Pattern Observer/Observable que nous avons implémenté pour la classe Case avec la classe IHM-Case.

4. Itération 4 : Consulter les informations du jeu

Le déroulement :

1. Arrivé à son tour de jeu, le joueur appuie sur la fenêtre des personnages

2. Une fenêtre s'affiche avec la liste des personnages présents sur le plateau par équipe.
3. Il appuie ensuite sur la fiche du personnage sélectionné
4. Une fenêtre s'ouvre et affiche son état : point de vie, de mouvement et altérations possibles
5. il appuie ensuite sur la fiche des actions du personnage
6. Une fenêtre s'ouvre et affiche la liste des attaques disponible pour le personnage et leurs effets

Par rapport à la partie UML, nous avons modifié le fonctionnement de l'itération. En effet au lieu de cliquer sur un bouton pour voir les informations d'un personnage, celle-ci sont directement affichées dans une fenêtre intégré à la fenêtre de jeu.

Pour consulter les informations du jeu, nous avons utilisé les trois classes modèles « IHMListeAttaque », « IHMListePersonnages » et « IHMPersonnageInfo ». Ces trois classes sont des JFrames utilisées par le contrôleur « Jeu ». Dans ces trois classes nous avons fait appel à plusieurs méthodes de la classe « Personnage » et aussi à des méthodes des classes « Joueur » et « Equipe ».

Ces deux classes sont des interfaces graphiques :

IHMListeAttaque : c'est la fenêtre qui affiche les attaques du personnage sur lequel on a cliqué en dernier. Elle affiche le nom de toutes les attaques que le personnage peut effectuer avec leur descriptif. Cette interface utilise la méthode « getAction » de la classe « Personnage » qui retourne les actions du personnage et la méthode « getDescription » présente dans les différentes classes du package « Action » et qui retourne la description de l'action.

IHMListePersonnages : c'est la fenêtre qui affiche les personnages présents dans l'équipe. On peut voir une image représentant la classe des personnages ainsi que leur classe, leur nom, leurs points de vie, de mouvement, leur état, leur type (personnage terrestre ou aérien). On peut aussi voir si le personnage est vivant ou KO. Cette interface utilise des méthodes des classes « Joueur », « Equipe » et « Personnage ».

IHMPersonnageInfo : cette interface permet de voir les informations du personnage sur lequel on a cliqué en dernier. On peut ainsi voir les informations d'un personnage même si ce n'est pas celui qu'on a sélectionné. On peut voir une image relative à la classe du personnage, sa classe, son nom, ses points de vie, de mouvement, son état. Cette fenêtre inclus un JButton « Attaque du personnage » qui permet d'ouvrir une fenêtre « IHMListeAttaque » qui permet de voir les attaque du personnage. Cette interface utilise des méthodes de la classe « Personnage ».

5. Itération 5 : Sauvegarder la partie

Le déroulement :

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

Pour l'implémentation de la sauvegarde, nous avons repris le même concept qu'en UML, nous avons utilisé :

- **L'interface Serializable** : qui nous permet de sérialiser/deserialiser un objet vers un flot de données binaires

- **La classe FileInputStream/FileOutputStream** : qui nous permet de prendre en entrée/sortie un fichier externe étant un flot de données binaires
- **La classe ObjectInputStream/ ObjectOutputStream** : qui nous permet de récupérer un objet flot de données binaires en entrée/sortie

Pour cela, nous sérialisons toutes les classes dont nous voulons sauvegarder le contenu :

- Classe « Jeu »
- Classe « Equipe »
- Classe « Joueur »
- Classe « Plateau »
- Classe « Case »
- Classe « Tour »

Nous sérialisons de même les classes concernant l'interface graphique du jeu (IHMJeu, IHMBandeau, etc...) de manière à sauvegarder au détail près toutes les actions que le joueur était en train d'effectuer avant la sauvegarde. En clair nous sauvegardons la fenêtre dans l'état actuelle du tour. C'est pourquoi nous avons dû ajouter deux classes pour réussir la sauvegarde complète de l'IHM:

- **ActionListenerSerializable** : c'est une classe qui implémente un « ActionListener » et « Serializable ».
- **WindowAdapterSerializable** : c'est une classe héritant de « WindowAdapter » et qui implémente « Serializable ».

Remarques :

- ❖ On peut sauvegarder uniquement lorsqu'on souhaite quitter le jeu via le PopUp Quitter.
- ❖ On ne peut effectuer qu'une seule sauvegarde (la nouvelle sauvegarde efface l'ancienne)
- ❖ On aurait pu utiliser un JChooseFile pour permettre d'une part de pouvoir sauvegarder dans l'emplacement que l'on veut et avec le nom de la sauvegarde désiré et d'avoir un nombre infini de sauvegarde.

D. Recette

Nous avons développé le jeu « Bataille Fantastique » qui couvre toutes les fonctionnalités attendues par le cahier des charges.

Notre application permet à deux joueurs de jouer une ou plusieurs parties de notre jeu « Bataille Fantastique ». Elle assure le bon enchainement des étapes depuis le paramétrage du jeu et la constitution des équipes, jusqu'à la succession des tours et la fin de la partie, en passant par l'étape de la consultation des cartes d'identités des joueurs qui n'est pas obligatoire. De plus, nous avons ajouté la sauvegarde d'une partie (dans un fichier) et le chargement de celle-ci au prochain lancement du jeu (ou commencer une autre nouvelle partie si on ne souhaite pas charger la sauvegarde). Cette fonctionnalité n'était pas demandé, nous l'avons mise dans notre partie conception UML comme étant optionnelle, mais comme nous avons fini le projet avant le délai, nous l'avons implémenté avec succès.

Nous avons testé le bon fonctionnement de notre jeu, Lors de la phase du test nous avons aussi testé les scénarios anormaux qui pouvaient se passer (comme une saisie incorrecte de la taille du plateau et le nombre des personnages par équipe, ou aussi sélectionner une case occupée ou dans la zone adverse pour se déplacer, ou bien infligé une attaque à un personnage qui ne se trouve pas dans le champ de cette attaque...). Donc tous les cas normaux et anormaux ont été testés, nous

avons géré les scénarios anormaux à l'aide des exceptions, et à chaque fois que le joueur essayait de faire une action interdite, on lui affichait un message d'erreur dû à une exception bien déterminée.

A chaque itération, nous avons implémenté un TestAvancé, qui nous permettait de tester directement l'itération en cours sans passer par les itérations précédentes. (Par exemple, on préconfigure les équipes des personnages pour l'Itération 3 comme cela on teste directement la phase du jeu sans avoir à configurer à chaque fois les paramètres du jeu et à choisir les personnages car nous avons déjà testé le bon fonctionnement de ceci auparavant). Cela permet un gain de temps énorme durant nos tests.

Après l'implémentation de chaque itération, nous testions si les fonctionnalités offertes correspondaient bien à celles attendues dans le cahier des charges. Dans le cas où ceci n'était pas respecté nous rajoutions des Exceptions pour les cas d'erreurs ou les différents cas possibles considérés comme anormaux. (Ex: Le joueur tente de poser un personnage sur une case non Vide). Lors des captures de ces Exceptions (clause try/catch) on génère un message d'erreur (PopUp).

E. Bilan

Le planning initial a subi un léger changement surtout au niveau des créneaux consacrés pour le développement des IHM puisque nous avons passé beaucoup plus de temps sur l'aspect interface graphique que sur la partie Code / Implémentation en Dur. Pour finir le projet dans les temps, nous avons dû travailler sur le projet durant des créneaux où nous avions pas prévu de travailler dessus dans le diagramme de Gantt décrit dans la figure n°2. Mais ce travail en plus a été très rentable puisque nous avons fini le projet avant le délai prévu et il était entièrement prêt (avec la possibilité de faire une sauvegarde) pour la démonstration faite devant la classe.

Les difficultés que nous avons rencontrées étaient généralement orientées création des IHM. En effet, comme cela a été évoqué au dessus, nous avons mis un peu de temps au début pour se familiariser avec l'implémentation de la partie IHM et à prendre connaissance des différents outils que nous pouvions utiliser pour obtenir le design souhaité. Mais avec un peu de persévérance nous avons pris en main les techniques de réalisations des interfaces graphiques. Nous avons essayé de réduire le nombre des variables globales utilisées (static), mais il y a encore quelques unes dans les classes Actions.

Grâce à notre modélisation de notre projet en UML, nous avons pu faire une implémentation rapide et efficace. Toutefois, comme évoqué précédemment la mise en œuvre de la partie graphique est plus longue de manière à avoir un système complet, validant le cahier des charges et d'avoir un thème ou un style d'affichage qui conviendra à notre groupe.

Nous avons effectué quelques changements par rapport à la conception faite en UML :

- Nous avons rajouté beaucoup de méthodes graphiques pour les IHM
- Nous avons déplacé des méthodes d'une classe à une autre (ex : la méthode de la taille du plateau a été déplacée de la classe « Plateau » vers la classe « Jeu »)
- Nous avons renommé des méthodes voire même supprimer quelques une (la méthode *case_est_attaquable* a été supprimée mais une autre méthode similaire *voir-CaseDeplacement* a été ajoutée...)

Conclusion

Le jeu de « Bataille fantastique » est une succession d'interfaces qui permettent à deux joueurs se trouvant devant la même machine de jouer une ou plusieurs parties jusqu'à ce qu'il n'y ait qu'un seul vainqueur à la fin, chacun des deux joueurs doit composer une équipe de personnages fantastiques, ces derniers sont 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, choisit un personnage et puis choisit de le déplacer puis d'attaquer ou de faire l'inverse, sinon il peut ne pas jouer et passer au tour suivant. Les joueurs ont aussi la possibilité de charger une partie déjà commencée et sauvegardée avant.

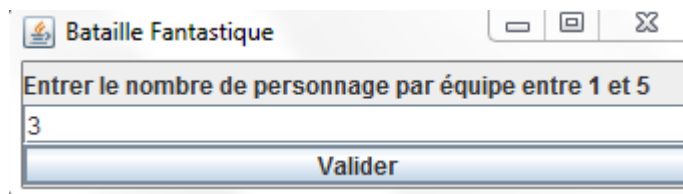
Ce projet nous a permis de mettre en pratique les différentes techniques de Java vues en cours et en TP, ainsi que de bien maîtriser ce nouveau langage. Au cours du projet, nous avons été conscients de l'importance de la phase de conception qui nous a permis d'implémenter plus efficacement et rapidement notre jeu. Nous avons ainsi limité les risques grâce à la partie UML, chose qui nous a aidée à bien répartir les tâches et à bien développer les itérations séparément. Cette application nous a permis de cerner les difficultés de la création des IHM et aussi de prendre connaissances de différentes techniques qui leur sont liées.

Bien que nous sommes contents du rendu de notre projet, il est toujours possible de l'améliorer, si nous avions plus de temps, nous aurions pu mettre des animations dynamiques pendant les attaques et optimiser le nombre des fenêtres JFrame. De plus, nous aurions pu modifier le Système (A-B-C-D).

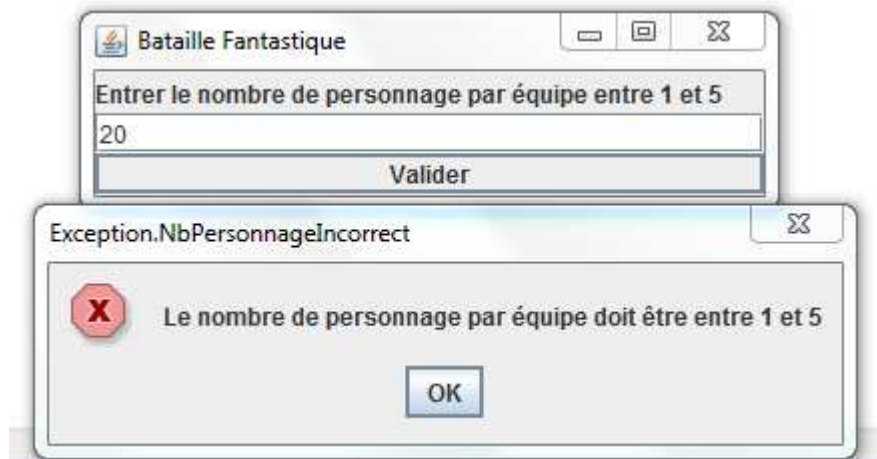
Annexes



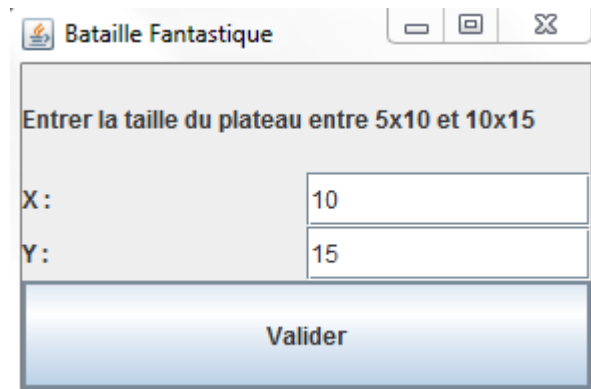
Annexe 1 : Interface d'accueil



Annexe 2 : Interface de choix du nombre de personnages



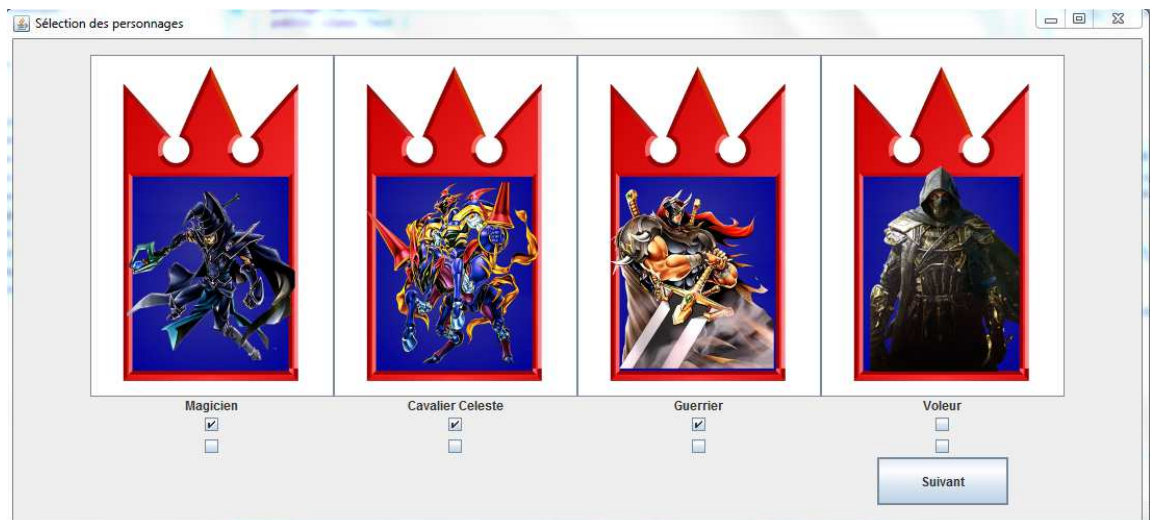
Annexe 3 : Interface cas d'erreur nombre de personnages saisi incorrect



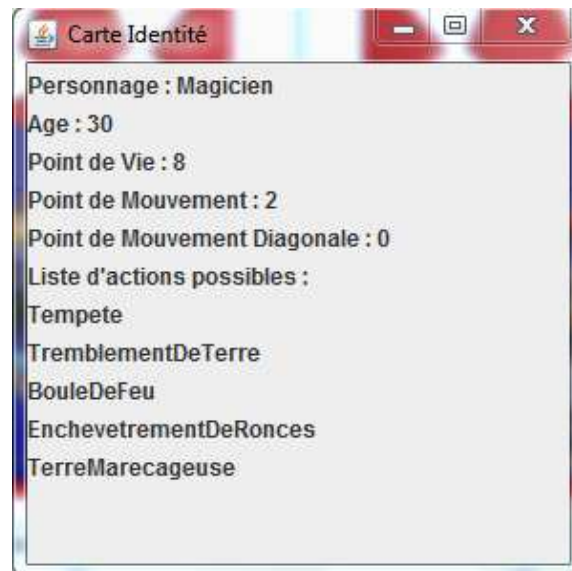
Annexe 4 : Interface de choix des dimensions du plateau



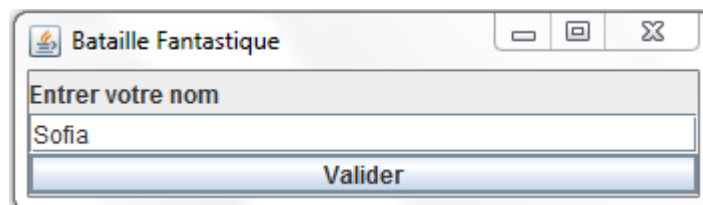
Annexe 5 : Récapitulatif des choix effectués



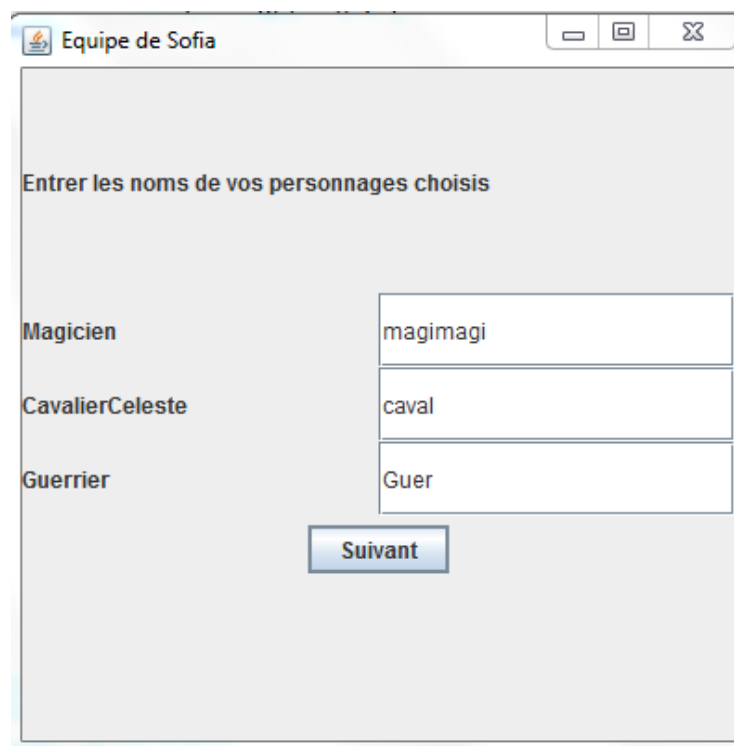
Annexe 6 : Catalogue pour le choix des personnages



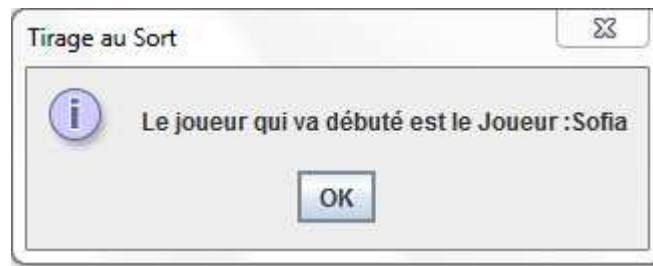
Annexe 7 : Exemple d'une carte d'identité



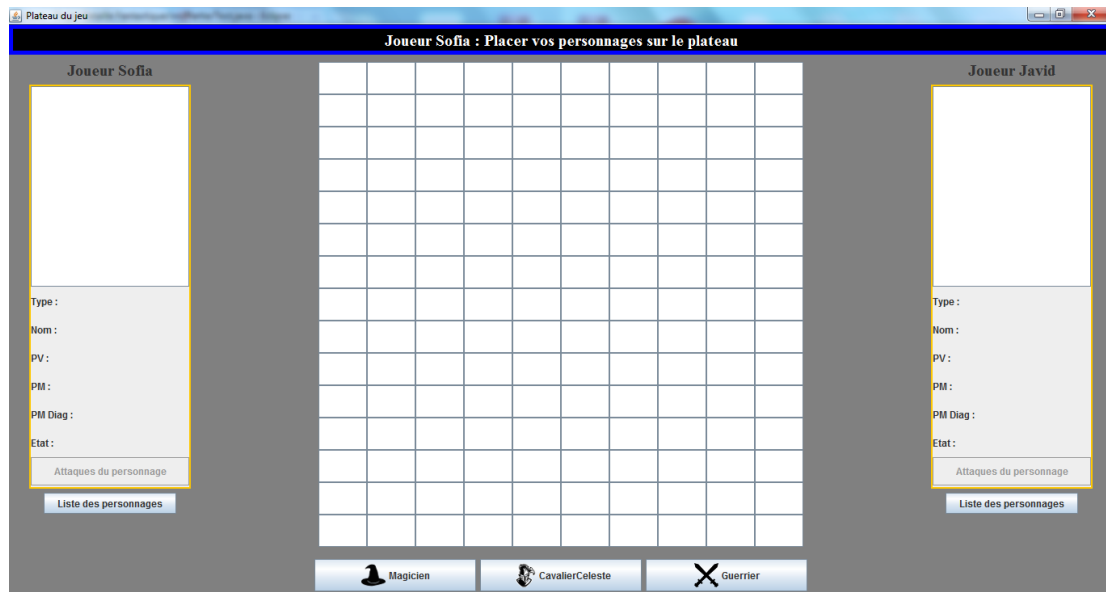
Annexe 8 : Fenêtre de saisie facultative du nom du joueur



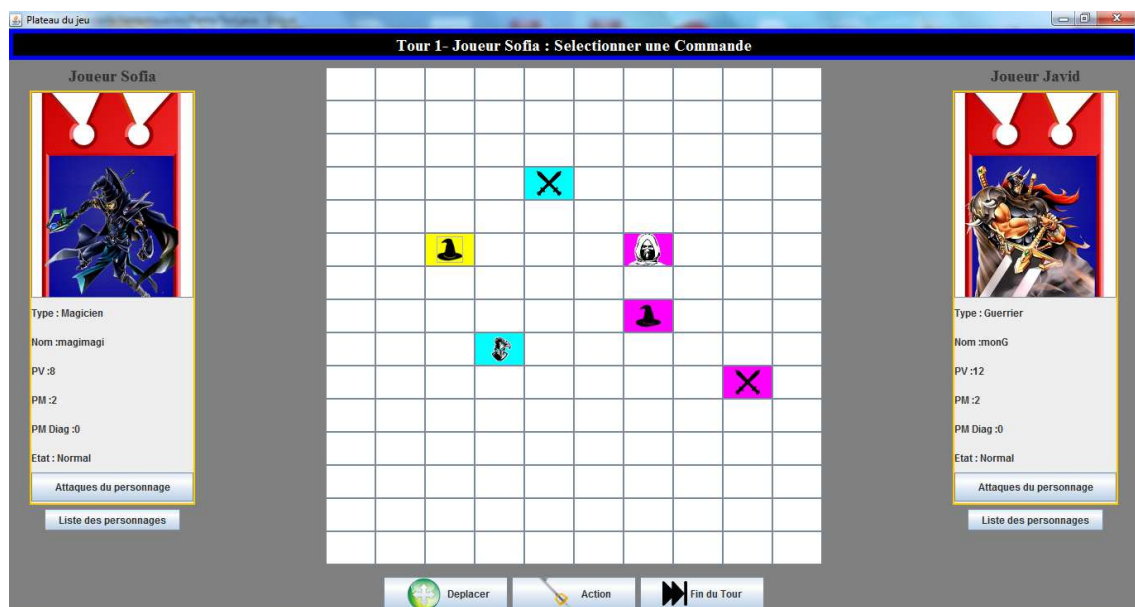
Annexe 9 : Fenêtre de saisie facultative des pseudos pour les personnages



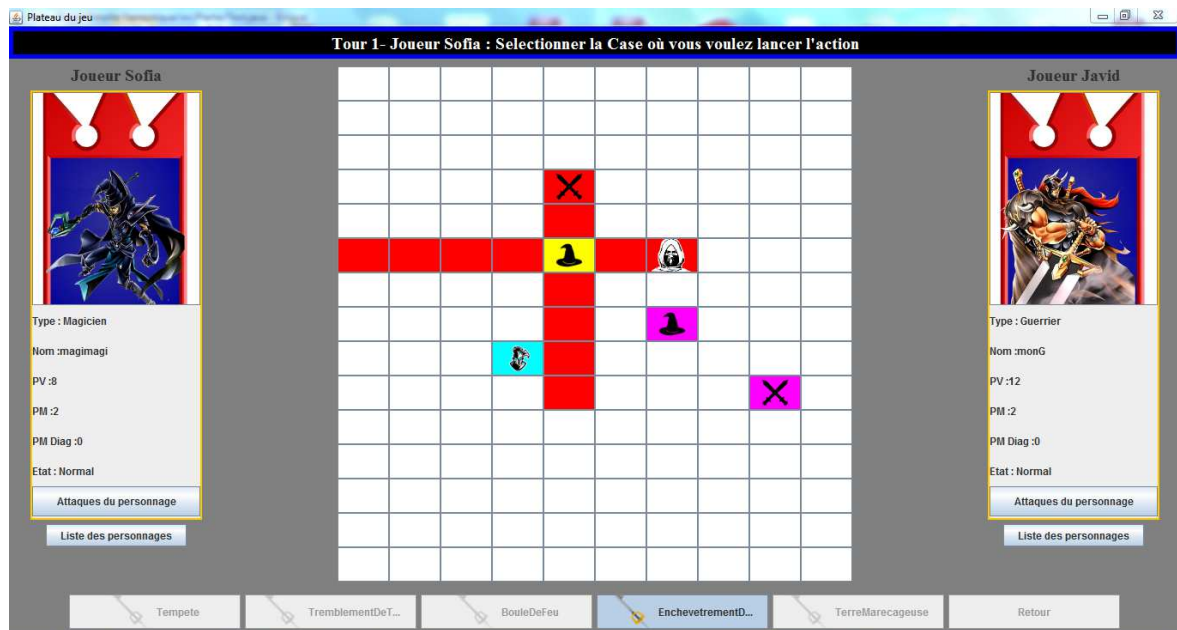
Annexe 10 : tirage au sort indiquant le nom du joueur qui commence la partie



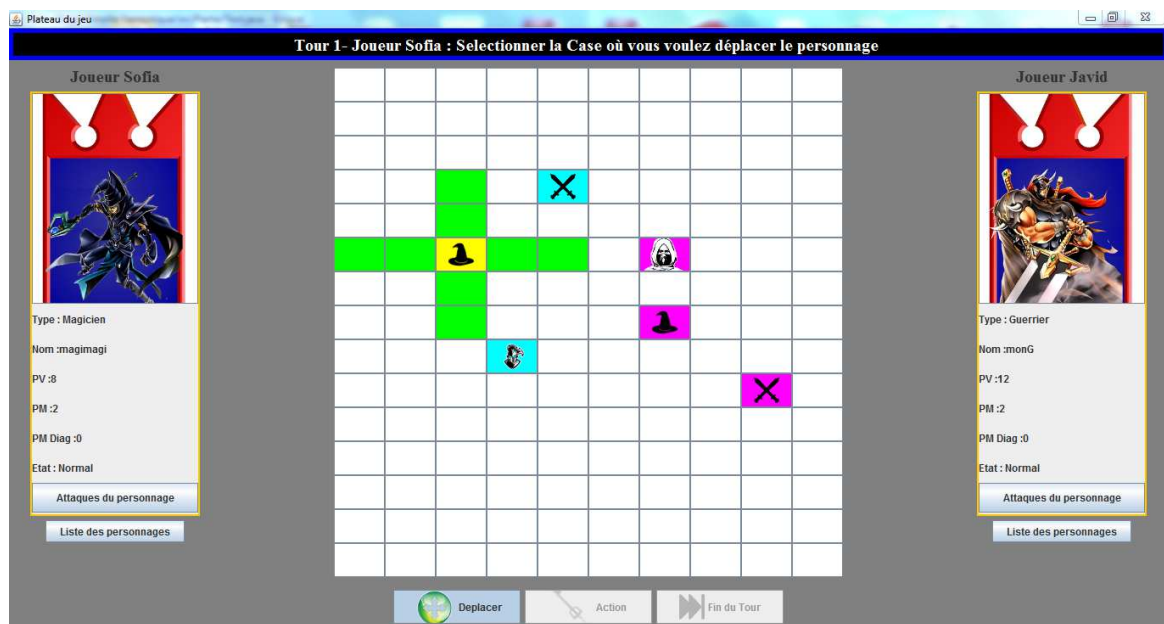
Annexe 11 : Interface de début d'une partie



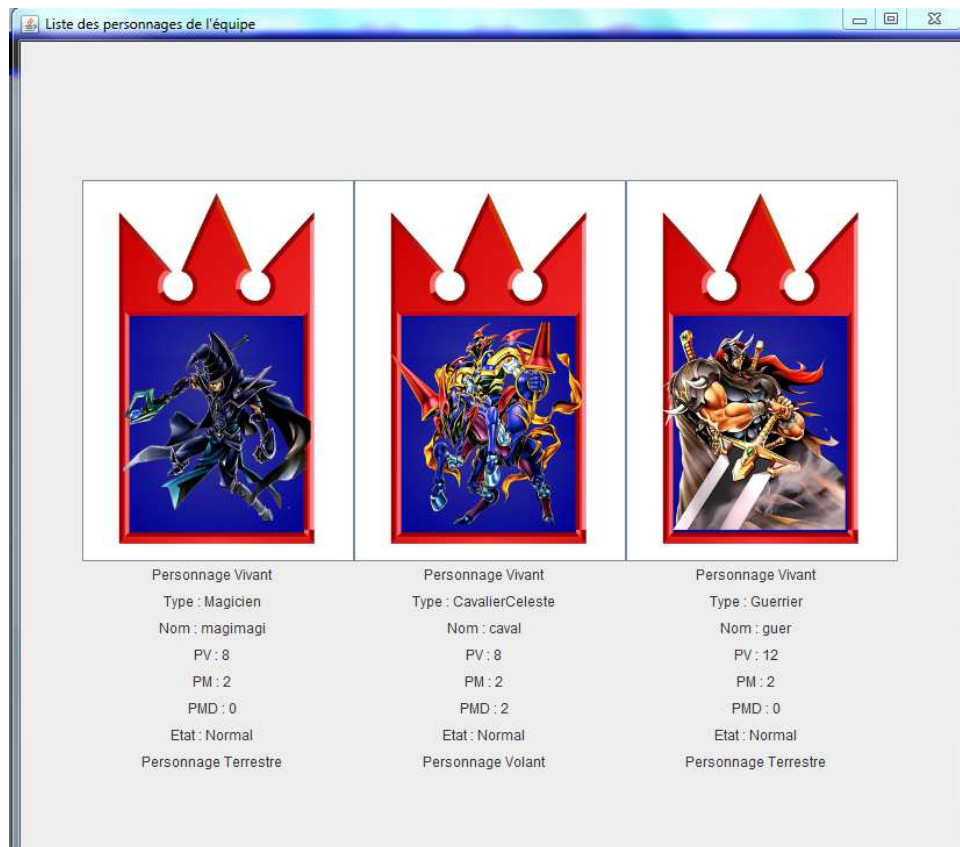
Annexe 12 : Interface avec plateau et bandeaux après placement des personnages



Annexe 13 : Interface lors de la réalisation d'une attaque



Annexe 14 : Interface lors de la réalisation d'un déplacement



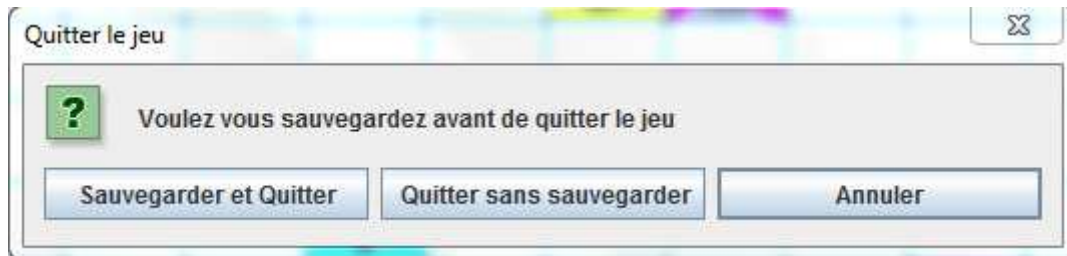
Annexe 15 : Liste des personnages d'une équipe avec leurs caractéristiques à jour durant la partie



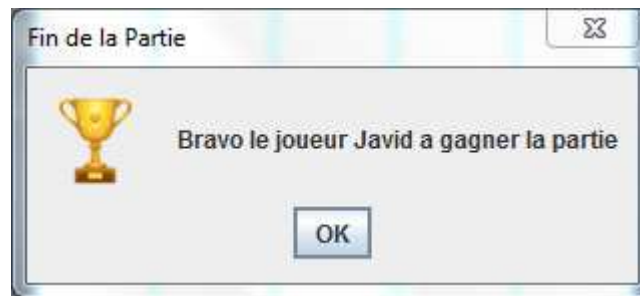
Annexe 16 : le bandeau du joueur Sofia



Annexe 17 : les commandes du jeu



Annexe 18 : Fenêtre lorsqu'on quitte une partie en cours



Annexe 19 : Fenêtre de fin lorsque le joueur Javid gagne



Annexe 20 : Fenêtre après la fin d'une partie suite à la déclaration d'un vainqueur