ENSICAEN
6, bd maréchal Juin
F-14050 Caen cedex 4

Spécialité Informatique
2e année

ENSICAEN

Rapport de projet

Munchkin en réseau

DELETTRE Jérome
LAIR Frédéric
LEPERS Florian
SABINE Thomas

Suivi :
CARRE Daniel

Année 2006-2007

Table des matières

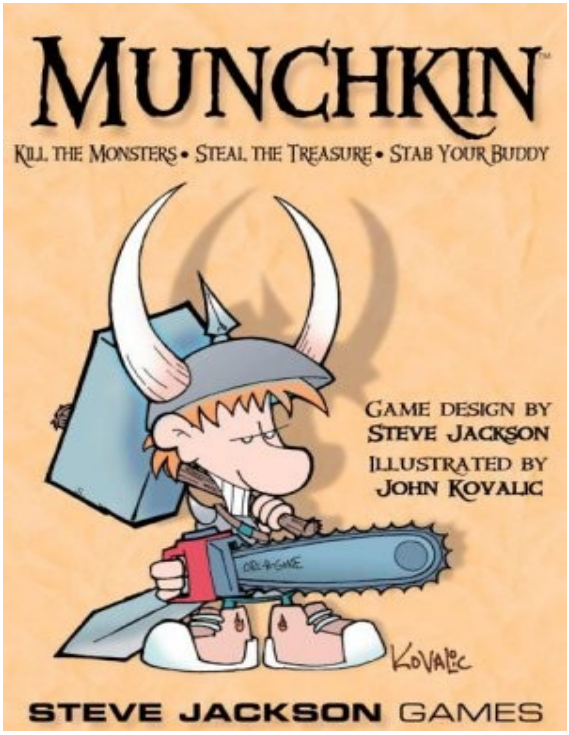
Introduction

Ce projet nous a été proposé par Claire Draperi dans le but d'adapter le jeu de cartes Munchkin a une version informatique. Ce jeu, créé par Steve Jackson et édité en France par Ubik, est relativement répandu mais n'a jamais fait l'objet d'un portage sur ordinateur.

Afin de recréer au mieux les conditions d'une partie réelle, cette version devra donc être jouable en réseau, et intégrer un chat, pour permettre la communication indispensable aux trahisons et collaborations entre les joueurs tout au long de la partie. L'interface graphique sera donc constitué d'un tableau blanc comprenant une fenêtre de chat ainsi que differents aperçus de la partie et des cartes des joueurs.

De plus, le jeu de cartes originel possède de nombreuses extensions, l'implementation de celles-ci devra donc être possible de façon simple, sans avoir à changer la structure du logiciel en lui-même.

Ce rapport a pour but de présenter ce jeu de cartes de façon succinte, et d'expliquer notre approche du problème ainsi que les choix de developpement (langage de programmation, modélisation, ...) et l'avancement actuel du projet.



1 Description du projet

1.1 Qu'est-ce que Munchkin ?

Munchkin est un jeu de cartes proposant une simulation simplifiée d'un jeu de rôles dans un donjon ; il propose une partie pour 2 à 5 joueurs dont le but est d'atteindre le premier le niveau 10 en traversant les salles du donjon (ie en tirant des cartes de type « Donjon ») et en s'aidant des trésors récupérés lors de sa progression (ie en utilisant des cartes de type « Trésor »). Différents choix s'offrent aux joueurs tout au long de la partie : tuer des monstres, les eviter, aider ou trahir ses « amis », piller des trésors ou les vendre...

Une partie se déroule au tour par tour, les joueurs se retrouvant confronter à des situations auquelles ils doivent faire face les uns après les autres. La durée moyenne d'une partie de ce jeu de cartes est d'environ une heure, mais peut varier grandement suivant les participants et le tirage des cartes.

1.2 Objectifs du projet

L'objectif du projet est de recréer sur ordinateur le déroulement et l'ambiance d'une partie de Munchkin. Il faudra donc tout d'abord développer une version simplifiée sur console, programmée en Java, et n'utilisant d'un éventail réduit des cartes disponibles, inseré dans une base de données, elles-même ne présentant qu'un panel succint des effets possibles.

Cette première version devra gérer correctement et être adaptée aux nombreuses situations et cas particuliers induits par le jeu (gestion des classes, races ainsi que les pouvoirs spécifiques associés à celles-ci). Seules des cartes du jeu de base seront introduites dans un premier temps, la gestion des add-ons (extension) sera implentée plus tard.

Ensuite, il faudra développer tout l'aspect jouabilité en réseau et communication entre joueurs, chaque joueur ayant à sa disposition des cartes visibles par tous les autres joueurs, et des cartes dans sa main qui ne doivent être visibles que de lui seul.

L'interface graphique devra donc remplir ses fonctionnalités, d'où la necessité d'utiliser un tableau blanc constitué d'un chat pour permettre la bonne communication entre joueurs, et proposant une vue globale de l'ensemble de la partie ainsi qu'une vue présentant les cartes de la main du joueur.

1.3 Fonctionnalités attendues

Comme il a été dit précédemment, ce programme devra, pour être fonctionnel et jouable facilement, inclure différentes fonctionnalités, telles qu'un tableau blanc et un support réseau.

1.3.1 Jouabilité en réseau

Afin de créer une partie en réseau accessible par plusieurs joueurs simultanément, il est nécessaire de créer une fenêtre permettant soit de créer, soit de rejoindre une partie en attente de joueurs.

Cette fenêtre sera donc graphique et comportera deux onglets :

- le premier onglet offre la possibilité à un joueur de créer et d'initialiser une partie, en la nommant et en lui indiquant un nombre maximum de joueurs. Cette partie sera alors en attente de joueurs. Cet onglet permet aussi au joueur de mettre ses paramètres personnels, tels que son nom, par exemple.
- le deuxième onglet permet de rejoindre une partie en attente en indiquant l'IP de l'ordinateur hôte de la partie. Comme précédemment, le joueur pourra remplir ici ses paramètres personnels.

1.3.2 Tableau blanc

Une fois la partie lancée, les joueurs auront la possibilité d'interagir avec un tableau blanc. Tout d'abord, ils pourront communiquer les uns avec les autres à l'aide d'un chat situé dans la partie droite du tableau blanc.

Dans la partie centrale se trouvera la situation actuelle de la partie en cours . Cette partie sera commune et identique à tous les joueurs, et représentera le combat ou l'effet en cours à ce moment de la partie, ainsi que les pioche et défausse (celle-ci consultable par tous les joueurs).

Le partie superieure sera séparée en deux sections : tout d'abord un récapitulatif des scores, puis un accès aux cartes des différents joueurs.

1.3.3 Interface graphique

L'interface graphique, pour pouvoir être lisible et compréhensible, devra permettre l'affichage des cartes en jeu, qu'elles soient jouées par l'utilisateur ou les autres joueurs, de même que les cartes dans sa main. Pour faciliter ceci, la partie du tableau blanc donnant accès aux cartes des joueurs sera constituée de boutons ouvrant de nouvelles fenêtres.

Les boutons présents dans cette barre permettront de choisir différentes vues, parmi celles-ci les cartes de la main du joueur, les cartes qu'il a en jeu au moment présent (que ce soit sa race, sa classe, son équipement ou les diverses malédictions dont il est affecté), ou celles que le joueur de sont choix a en jeu en ce moment.

Les fenêtres crées par ces boutons contiendront les differentes cartes vues de façon plus complète que dans la partie centrale de la fenêtre principale, la place disponible dans celles-ci étant plus grande que dans la vue originale.

2 Méthode de travail

2.1 Choix du langage de programmation

Le projet contenant beaucoup d'interface graphique à programmer, nous avions le choix entre la programmation en Java ou la programmation en C++ (utilisation de Qt pour l'aspect graphique).

Nous avons finalement choisi le langage Java pour sa simplicité. En effet, Qt n'étant passé libre que depuis peu, il est difficile de trouver de la documentation et des aides dans ce langage. De plus, les interfaces programmées en Java sont plus belles et offrent plus de fonctionnalités pour l'instant.

2.2 Organisation du travail

Notre groupe de travail étant constitué de 4 étudiants, il était indispensable d'adopter des méthodes de travail efficaces, et d'optimiser notre temps de travail. En effet, il nous était difficile de travailler à 4 en même temps sur un même aspect du développement.

Nous avons donc dans un premier temps travaillé tous ensemble sur la modélisation afin de nous mettre d'accord sur les fonctionnalités devant être assurées par chaque classe ainsi que leurs intéractions, puis nous avons travaillé en binôme sur la programmation des classes, comme le permet les langages orientés objet.

Un planning a donc été réalisé en début de projet, suivant les conseils de M. Daniel Carré, que nous avons essayé de suivre au maximum, nous permettant d'avoir un temps de travail optimal pour chacun.

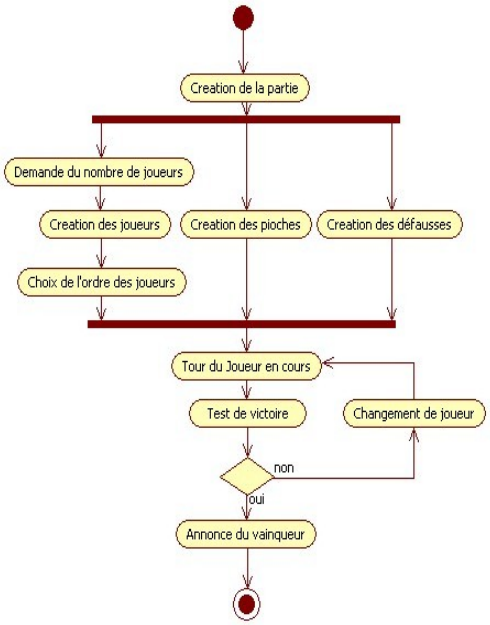
2.3 Planning

	Delettre Jerome	Lair Frederic	Lepers Florian	Sabine Thomas
UML	x	x	x	x
implémentation de Carte	x		x	
implémentation de Pioche et de Défausse	x			x
implémentation de Joueur		x	x	
implementation de Partie		x		x
Base de donnée		x		x
Interface (tableau blanc)	?	?	?	?
Résaux	?	?	?	?
Chat	?	?	?	?
Facultatif :				
Sauvegarde	?	?	?	?

3 Développement et programmation

Une fois la vue d'ensemble du projet déterminée, il a fallu s'attaquer à la modélisation du logiciel, ainsi que réaliser différents diagramme UML. Voici à présent quelques détails sur le déroulement d'une partie ainsi que la structure du programme.

3.1 Déroulement d'une partie



Le déroulement d'une partie est relativement simple dans sa structure. La partie est tout d'abord créée.

Pour ce faire, le logiciel demande au joueur créateur de la partie le nombre de joueurs souhaité, et une fois ceux-ci connectés, ayant rejoints la partie et remplis leur description, les instances les représentant sont créées, et leur ordre de passage est déterminé.

Dans le même temps, le logiciel charge les cartes constituant les deux pioches depuis la base de données, et créé les instances de celles-ci, ainsi que celles des défausses, vides au début de la partie

Une fois tout ceci fait, la partie peut réellement commencer. Le programme procède alors aux tours de tous les joueurs de la partie, et vérifie à la fin de chacun de ceux-ci si un joueur à rempli les conditions de victoire (ie atteint le niveau 10). Si oui, le joueur est désigné gagnant, et la partie s'arrête. Dans le cas contraire, la partie continue et c'est au tour du joueur suivant de jouer.

3.2 Modélisation

Une fois le déroulement d'une partie définie, nous pouvons nous interesser aux classes que nous allons utiliser pour simuler la partie de Munchkin. Le schéma page suivante résume les dépendances qu'il y a entres elles.

La première est celle de la partie. Ce sera celle-ci qui gèrera le tour des joueurs, la vérification des conditions de victoire, la création des pioches, etc... C'est le centre névralgique du logiciel, et la classe dite « controleur » du jeu.

Ensuite, la classe joueur. Celle-ci est là pour modéliser (on s'en doute) les joueurs prenant part à la partie. Elle a comme attibuts le niveau du joueur, sa puissance (le niveau auquel s'ajoute les éventuels bonus accordés par les objets du joueur), ainsi que sa main et les cartes qu'il a joué et possède devant lui. Elle contient aussi toutes les fonctions nécessaire pour pouvoir jouer ses cartes, en piocher d'autres ou s'en défausser.

Viennent ensuite les deux classes de pioche et de défausse. Elles sont relativement similaire, et contiennent un vecteur de cartes, ainsi que les fonctions nécessaires à leur ajout ou suppression.

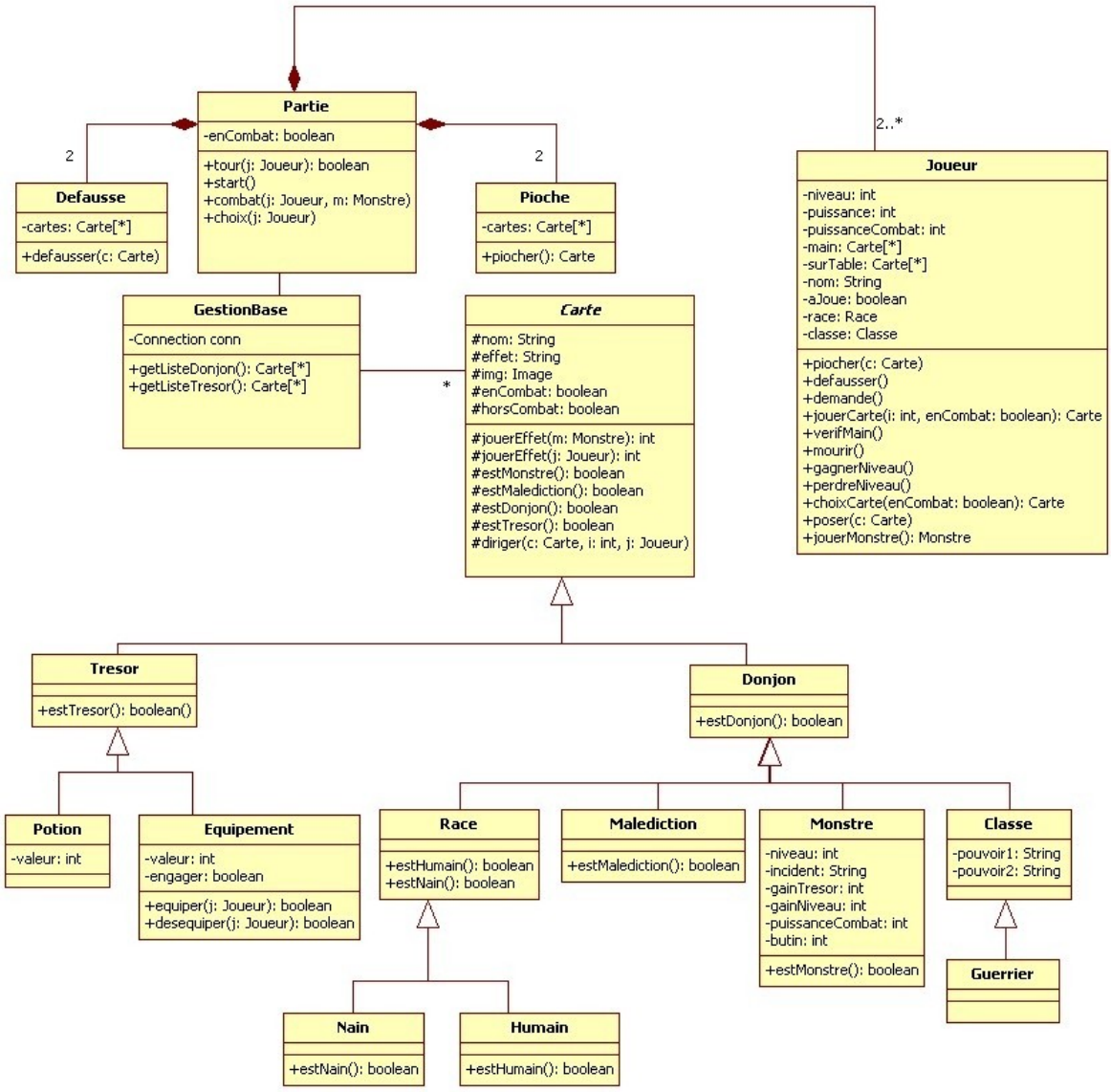
Enfin, les classes correspondants aux cartes du jeu. Celles ci sont réparties en deux catégories : les cartes « Donjon », et les cartes « Trésor ».

Les cartes « Donjon » correspondent aux monstres et aux malédictions qui devront affronter les joueurs au cours de la partie, ainsi que les cartes des races et des classes qu'ils pourront incarner au fil de la partie.

Les cartes « Trésor » correspondent quant à elles aux objets magiques et aux potions que les joueurs pourront utiliser pour améliorer leur puissance, ainsi qu'aux autres cartes qu'ils peuvent obtenir en triomphant de monstres, telles que des niveaux supplémentaires.

Tous ces types de cartes ont des classes spécifiques, héritant de la classe « donjon » (respectivement « Trésor ») principale, du fait de leurs attributs différents, et pour permettre de gérer facilement dans le programme leur pose et leur jouabilité. Ces deux classes héritent elles même de la classe générique « Carte », dont les attributs et les méthodes regroupent ceux communs a toutes les cartes du jeu.

Et pour importer toutes les cartes dans le jeu existe la classe GestionBase.



3.3 Difficultés d'adaptation du jeu à la programmation

Lorsque nous avons commencés à réfléchir à la programmation du jeu, plusieurs problèmes se sont posés rapidement à nous.

Le premier a été le stockage des cartes et des informations les concernant. Comment, en effet, faire pour créer toutes les instances des cartes au début de chaque partie sans alourdir le code de façon trop grande, et sans utiliser une quantité énorme de mémoire ? La meilleure solution à nos yeux a été de stocker les informations des cartes dans une base de données créée à cet effet, et ainsi de pouvoir initialiser les instances de celles-ci au début de chaque partie sans augmenter la taille du code de façon démeusurée.

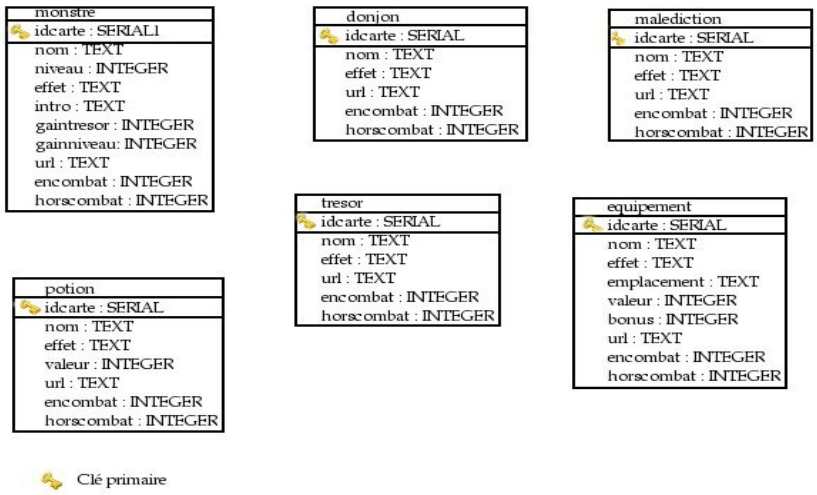
L'autre problème majeur, et sans doute le plus grand, a été la gestion des différents effets des cartes. Les cartes ont quasiment toutes des effets différents, ou alors avec des paramètres différents. Dès lors, comment gérer les effets de chaque carte ? En créant autant de fonctions que d'effets ? Et comment les stocker dans la base de données ? Ce problème a été résolu en créant un champ effet dans les tables des cartes, ce champs étant rempli par une string composé des différents effets de la cartes suivi de son argument s'il en demande un. Par exemple : « potion:2 » indique que la carte a comme effet d'ajouter 2 à la puissance du joueur ou du monstre ciblé en combat. La reconnaissance de cet effet est fait par une fonction comportant un tokeniser, pour découper la chaîne de caractères, et ainsi faire l'effet attendu.

3.4 Base de données

3.4.1 Choix du SGBD

Nous avons choisi pour la partie base de données de ce projet d'utiliser postgres. Ce choix a été fait à cause de plusieurs paramètres, mais surtout par facilité. En effet, nous avons eu au cours de l'année des cours sur ce langage de requête et de gestion de base de données. De plus, nous avions à notre disposition une base de données grâce au fait que nous avions eu des tp sur cette matière. Ce choix s'est donc logiquement imposé à nous.

3.4.2 Modèle conceptuel final



4 Perspective pour l'avenir

Le cahier des charges du projet étant bien rempli (Programmation complète du jeu avec ses spécificités complexes, de l'interface de connection à une partie, Gestion du réseau avec le tableau blanc, ses interactivités et son chat etc ..), il est normal que nous n'ayons pas pu arriver au bout de celui ci. Voici donc les informations sur les points abordés et remplis, et sur ce qu'il reste à faire ou les fonctionnalités pouvant être ajoutées.

4.1 Avancement du projet

Malgré tout, le plus gros du projet a été réalisé, c'est à dire la programmation du jeu en Java. Plusieurs joueurs peuvent faire une partie basique (environ une vingtaine de cartes typiques servent de panel représentatif pour le jeu), et en suivre le déroulement à l'identique d'une partie du jeu de carte normal.

Le programme gère la partie au tour par tour, en proposant des combats contre des monstres aux joueurs, et permet à chacun d'y intervenir en demandant à chaque joueur s'y veut jouer une carte de sa main pour modifier l'issue du combat.

Les effets particuliers des cartes a été programmé tels qu'ils existent dans le jeu original, par exemple concernant la modification des niveaux des combattants, la modification des conditions de fuite du combat, ou l'utilisation d'objets. Si une modification est faite par un joueur, la partie demande à nouveau à chaque joueur s'il désire jouer une carte.
Lorsque chaque joueur passe (ie décide de ne pas jouer de carte), la partie analyse le vainqueur du combat : en cas de défaite du joueur, elle effectue un random entre 1 et 6 (s'il n'y a pas de modificateur de fuite) et annonce le résultat en effectuant les conséquences correspondantes (gain/perte de niveaux ou objets).

Néanmoins, le combat a plusieurs n'a pas été géré tel qu'il est prévu par le jeu de carte, un joueur peux aider un adversaire contre un monstre pour des raisons stratégiques, mais n'y gagnera pas de trésors, alors qu'il le devrait.

Une interface graphique simpliste a été ajoutée, indiquant au joueur les cartes qu'il a en main lorsque celui-ci a la possibilité d'intervenir, aux moments importants. Les cartes sont ensuite jouées par simple clique dessus.

Enfin, quelques fonctionnalités particulières du jeu n'ont pas été programmées entièrement, comme les classes et races (bien que les classes Java soient écrites), ou encore les effets qui requièrent de lancer un dé.

4.2 Développement futur

4.2.1 Développement de l'aspect réseau

Il reste donc du travail à effectuer, pourquoi pas pour un futur projet. En effet, il reste à développer tout l'aspect réseau d'une partie de Munchkin, aussi bien l'interface invitant les joueurs à créér et rejoindre une partie, que le tableau blanc avec les fonctionnalités qui l'accompagnent.

Même si une ébauche de tableau blanc nous a été fournis par M. Régis Clouard, il est nécessaire de l'adapter au cahier des charges initial que nous nous sommes fixés, et d'y ajouter les éléments graphiques indispensables, comme les boutons ouvrants une fenêtre décrivant les cartes de sa main, les cartes jouées etc ..

4.2.2 Ajout des extensions

Tel que cela avait été prévu par le cahier des charges, l'implémentation de nouvelles cartes (celles des extensions par exemple) peut se faire de manière très simple, puisqu'il suffit de les ajouter à la base de données, en respectant les champs décrits précédement ainsi que quelques syntaxes pour les effets de la carte (par exemple 'potion:5' pour l'effet d'une potion augmentant les niveaux d'un camp de 5).

Néanmoins, comme dit au paragraphe précédent, les cartes qui nécessitent un dé pour un effet autre que la fuite ne peuvent être ajoutées sans avoir programmé cet aspect dans la classe Partie au préalable.

Conclusion

Comme nous nous y attendions après avoir réalisé la soutenance du projet, où nous avons vu que la modélisation se révélait plus compliquée que prévue puisqu'elle avait alors encore quelques défauts, nous n'avons pu arriver au bout du travail en quelques mois seulement vu le peu de temps libre disponible.

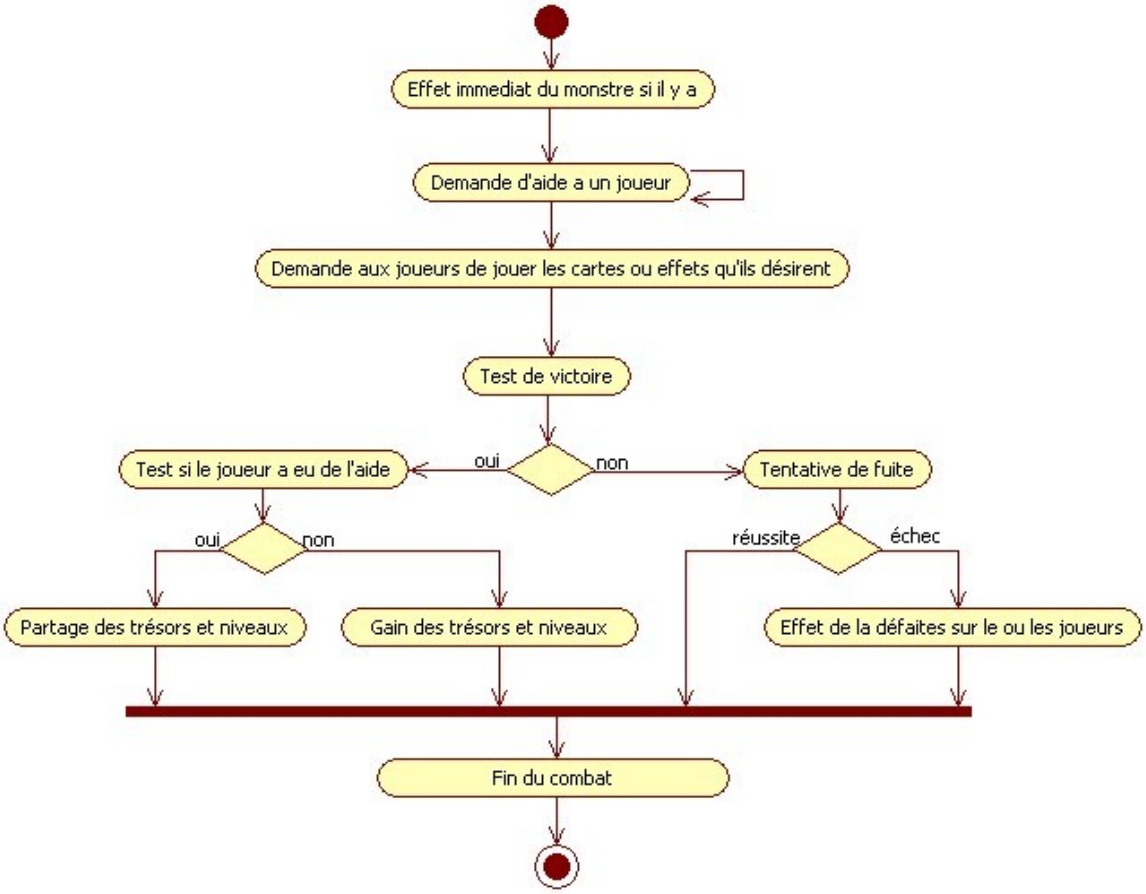
Pourtant, une bonne partie a été réalisé, donnant jour à une première version de programme permettant concrètement de tester le jeu informatique sur console, avec une interface graphique basique. Même si toutes les fonctionnalités complexes et variées du jeu de carte ne sont pas encore assurées dans cette version, la plupart ont été programmée.
Ainsi une partie complète peut être simulée, avec combats, intervention de tout les joueurs etc..

Les commentaires, exemples et explications apportées à notre projet permettront de facilement ajouter des cartes à celles introduites, ainsi que de reprendre les méthodes écrites afin d'y ajouter des fonctionnalités du jeu, ou bien de réaliser l'aspect réseau.

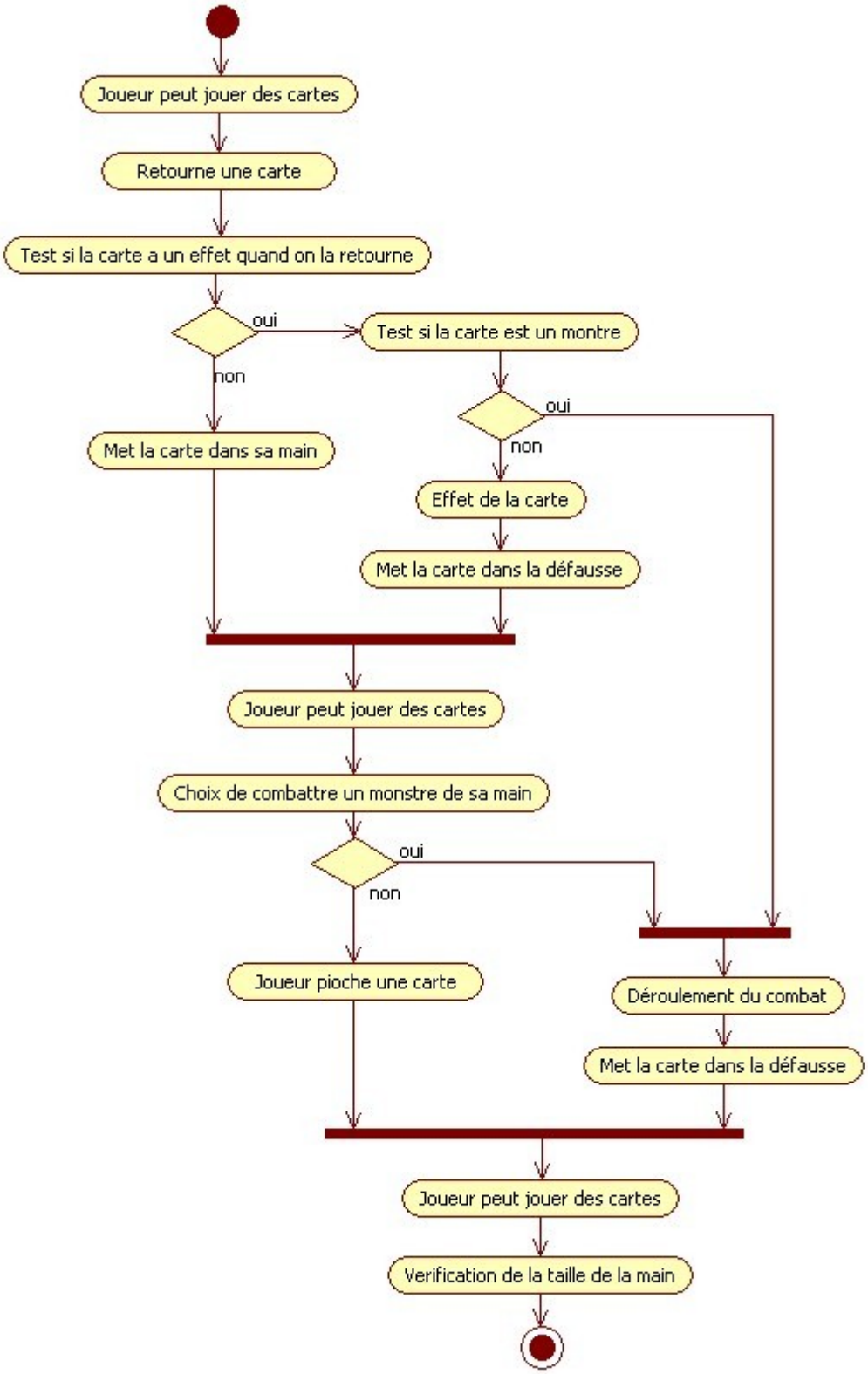
Au final, Munchkin s'est averé être un jeu de carte très intéressant à programmer, et très enrichissant vu sa complexité à adapter à l'informatique, malgré son apparente simplicité. Le travail à 4 est particulièrement difficile et requiert une grande organisation de notre part, chacun d'entre nous devant se tenir informé de l'actualité de la programmation, des nombreuses modifications ayant été apportées tout au long du développement et savoir précisement son rôle à tenir et ce qu'il a à faire afin que personne ne se retrouve délaissé.

Annexes

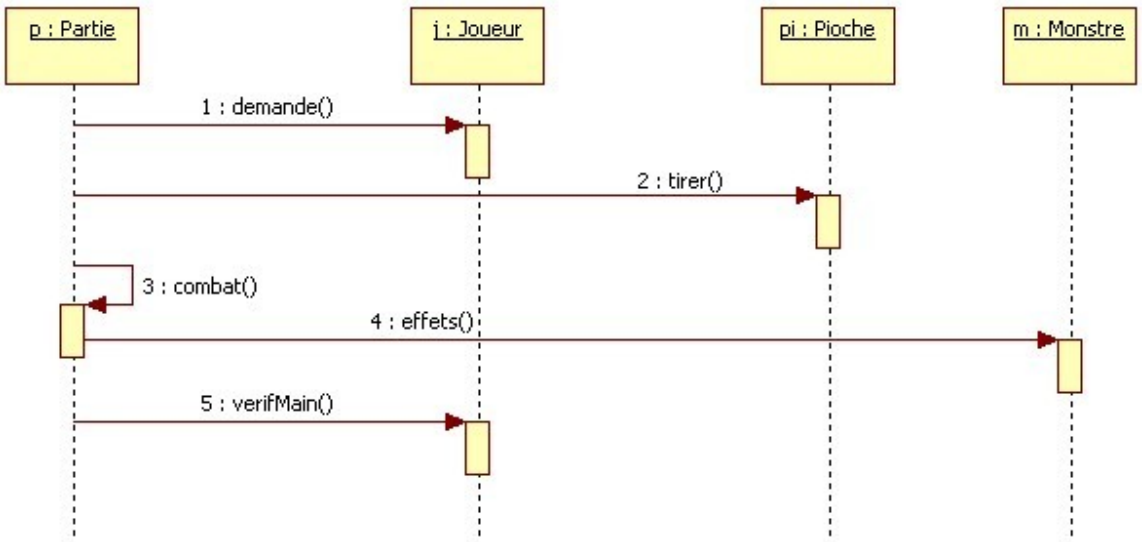
[1] Déroulement d'un combat



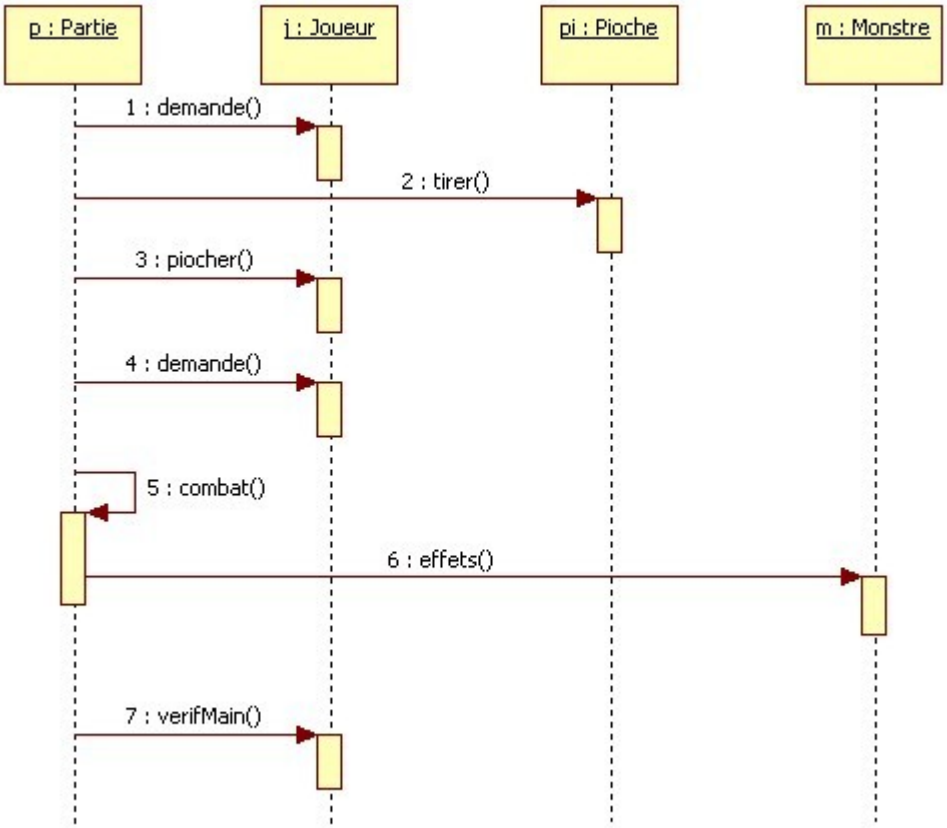
[2] Déroulement d'un tour



[3] Déroulement d'un tour avec combat, cas 1



[4] Déroulement d'un tour avec combat, cas 2



[5] Exemple de tableau blanc