INSA de Rennes
Rapport de conception

Projet Modélisation et Programmation Orientée Objet

Frank CHASSING et Amandine FOUILLET

Rennes, le 11 novembre 2014

Table des matières

Introduction

Dans le cadre des cours de Programmation Orientée Objets et de Modélisation et Conception de Logiciels, nous sommes amenés à réaliser un jeu pour ordinateur semblable au jeu Small World. Ce projet se déroule en deux temps : dans la première partie nous avons réalisé la modélisation du problème grâce à divers diagrammes UML puis, dans la seconde partie, nous réaliserons l'implémentation du jeu.

Le jeu à réaliser est un jeu tour à tour dans lequel chaque joueur dirige un peuple qui contient plusieurs unités. Le but du jeu est de gérer les unités sur une carte du monde pour obtenir le plus de points possible à la fin d'un certain nombre de tours. Pour gagner il faut contrôler le plus de cases possible en attaquant les unités adverses, en défendant son territoire et en se déplaçant sur les cases vides. Dans notre implémentation du jeu, deux joueurs s'opposeront, ils pourront choisir trois peuples différents : les Elfs, les Orcs et les Nains et se déplacer sur trois types de cases différents : la forêt, le désert, la montagne et la plaine.

Ce rapport présente le travail réalisé lors de la phase de modélisation au cours de laquelle plusieurs diagrammes ont été réalisés afin de décrire les différents aspects du jeu. Ainsi, dans un premier temps nous étudierons la phase de création d'une partie grâce aux diagrammes de cas d'utilisation, d'activité et de séquence. Puis nous étudierons le déroulement d'une partie, d'un tour de jeu et d'un combat là encore avec des diagrammes d'interaction, d'activité et de cas d'utilisation. Pour résumer et conclure sur la modélisation du jeu, nous finirons par détailler le diagramme de classes réalisé ainsi que les différents patrons de conception utilisés.

1 Création de la partie

Commençons par détailler ce qui se passe lors de l'ouverture du jeu. Lors de la création d'une nouvelle partie, l'utilisateur commence par choisir une carte parmi les trois différents types (la démo, la petite et la normale). C'est ce choix de carte qui détermine le nombre de cases et le nombre de tours, elle est créée de manière aléatoire. Pour pouvoir jouer, il est nécessaire que les joueurs soient au nombre de deux, ils sont créés après le choix de la carte. Les deux joueurs doivent choisir un pseudo et un peuple différent (Elf, Nain ou Orc). Une fois ces tâches réalisées, la partie peut commencer. Si une partie précédente avait été abandonnée avant la fin du jeu, lors de l'ouverture les joueurs peuvent choisir de reprendre cette ancienne partie ou en commencer une nouvelle. De même, de façon évidente, l'utilisateur peut abandonner la création de la partie à tout moment en quittant le jeu.

1.1 Diagramme de cas d'utilisation

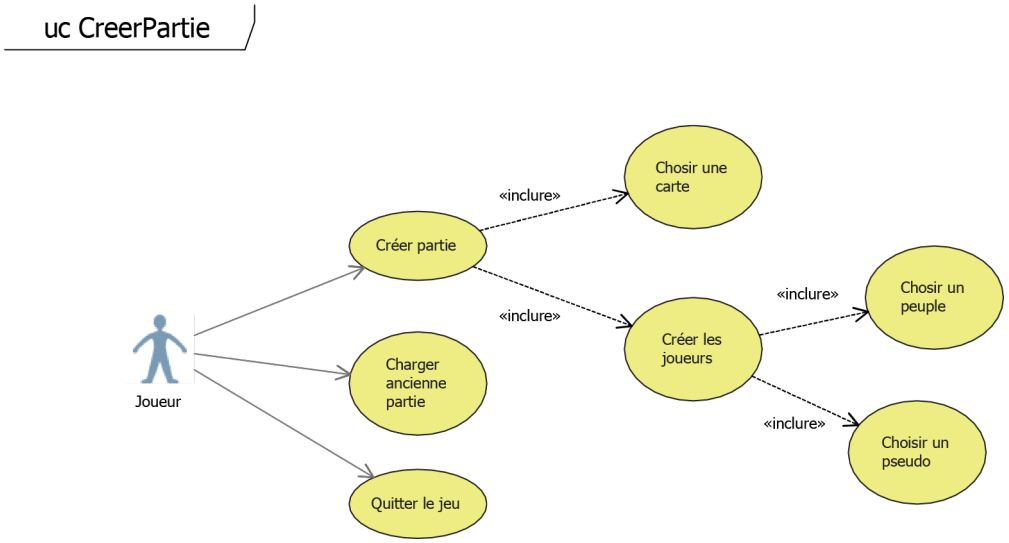

FIGURE 1 – Diagramme de cas d'utilisation - Créer une partie

Le diagramme de cas d'utilisation ci-dessus (FIGURE 1) illustre la création d'une partie du point de vue utilisateur. En arrivant sur l'interface d'accueil du jeu le joueur a trois possibilités : créer une nouvelle partie, charger une ancienne partie ou quitter l'application. Si l'utilisateur décide de créer une nouvelle partie, il commence par choisir une carte puis il crée les deux joueurs en donnant à chacun un pseudo et un peuple.

1.2 Diagramme d'activité

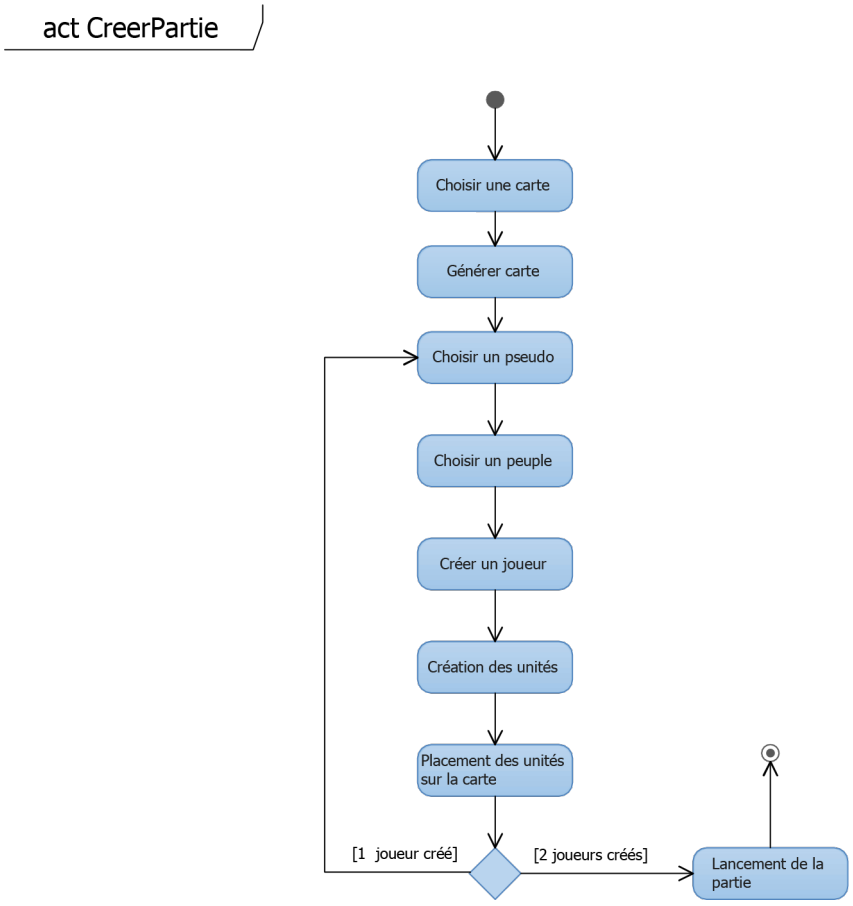

FIGURE 2 – Diagramme d'activité - Créer une partie

Le diagramme d'activité ci-dessus (FIGURE 2) illustre le processus de création d'une partie. Lorsque le système est en état de création d'une partie, l'évènement permettant à l'utilisateur de choisir une carte est le premier à se déclencher. Ensuite, le système rentre dans un processus de création d'un joueur : choix d'un pseudo, choix d'un peuple, création du joueur puis placement des unités sur la carte. À la fin du processus de création d'un joueur, le système vérifie le nombre de joueurs déjà créés : s'il n'y a qu'un joueur créé on retourne au début du processus de création d'un joueur, s'ils sont deux on lance la partie.

1.3 Diagramme de séquence

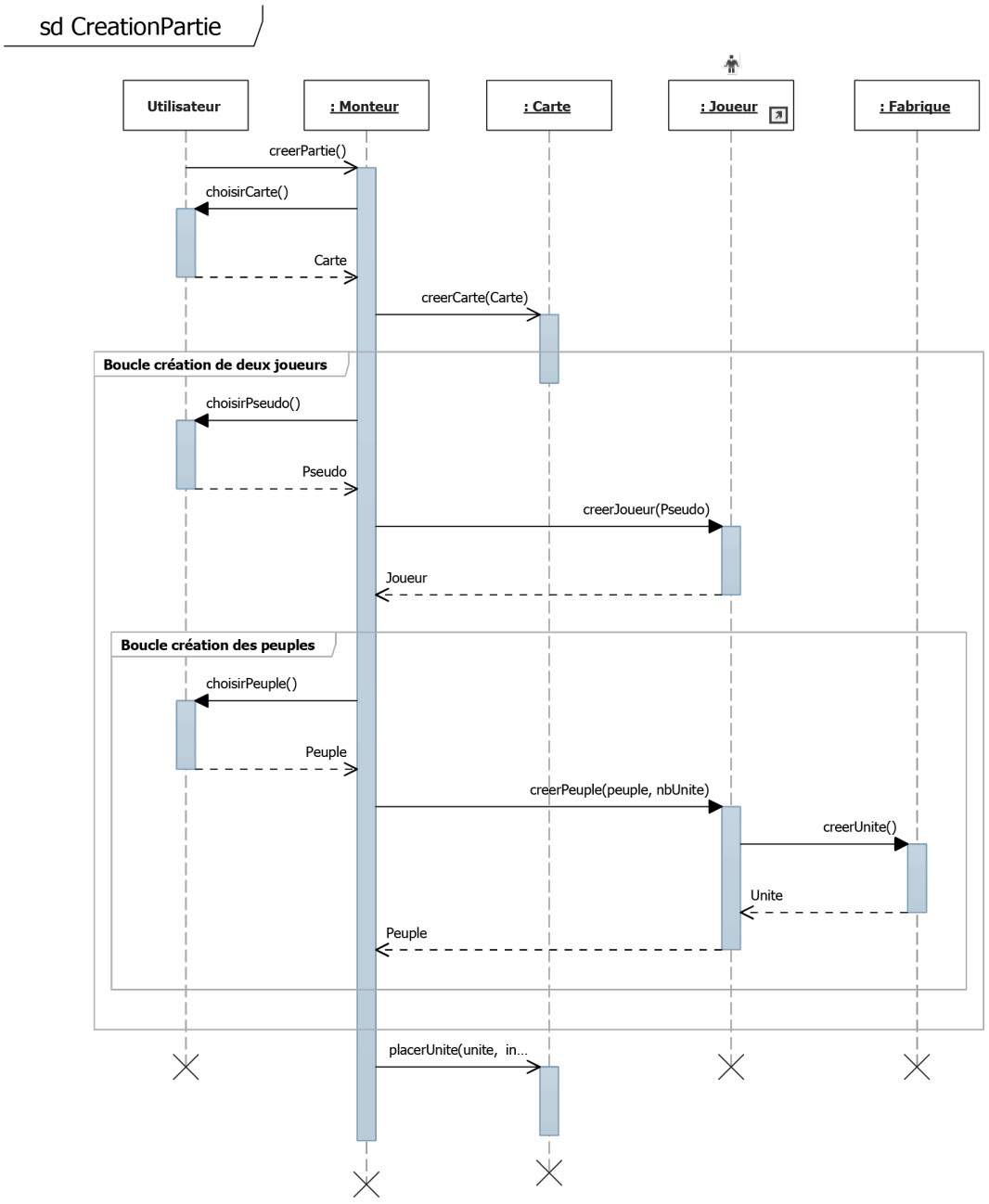

FIGURE 3 – Diagramme de séquence - Créer une partie

Le diagramme de séquence ci-dessus (FIGURE 3) illustre les interactions entre objets lors de la création d'une partie. On observe les mêmes événements qu'avec le diagramme d'activité mais, en étudiant la dimension temporelle, on met en évidence des boucles de création. En effet, le monteur crée deux joueurs à qui il attribue deux peuples différents. Chaque peuple crée ensuite ses unités que le monteur place sur la carte.

2 Déroulement d'une partie

Une fois que la partie est lancée, l'ordre de jeu est choisi aléatoirement et un des deux joueurs commence à jouer. Le déroulement d'un tour de jeu est décrit dans la suite de ce rapport. Un des joueurs peut perdre ses unités durant le tour, la partie s'arrête alors et l'autre joueur gagne. À la fin du nombre de tours, si aucun joueur n'a perdu avant, on calcule le nombre de points de chaque joueur pour déterminer le vainqueur. Le diagramme d'activité ci-dessous (FIGURE 4) illustre le processus de déroulement d'une partie. L'état "Lancement de la partie" a été détaillé dans le diagramme d'activité de création d'une partie tandis que l'état "Tour du joueur" sera détaille par la suite.

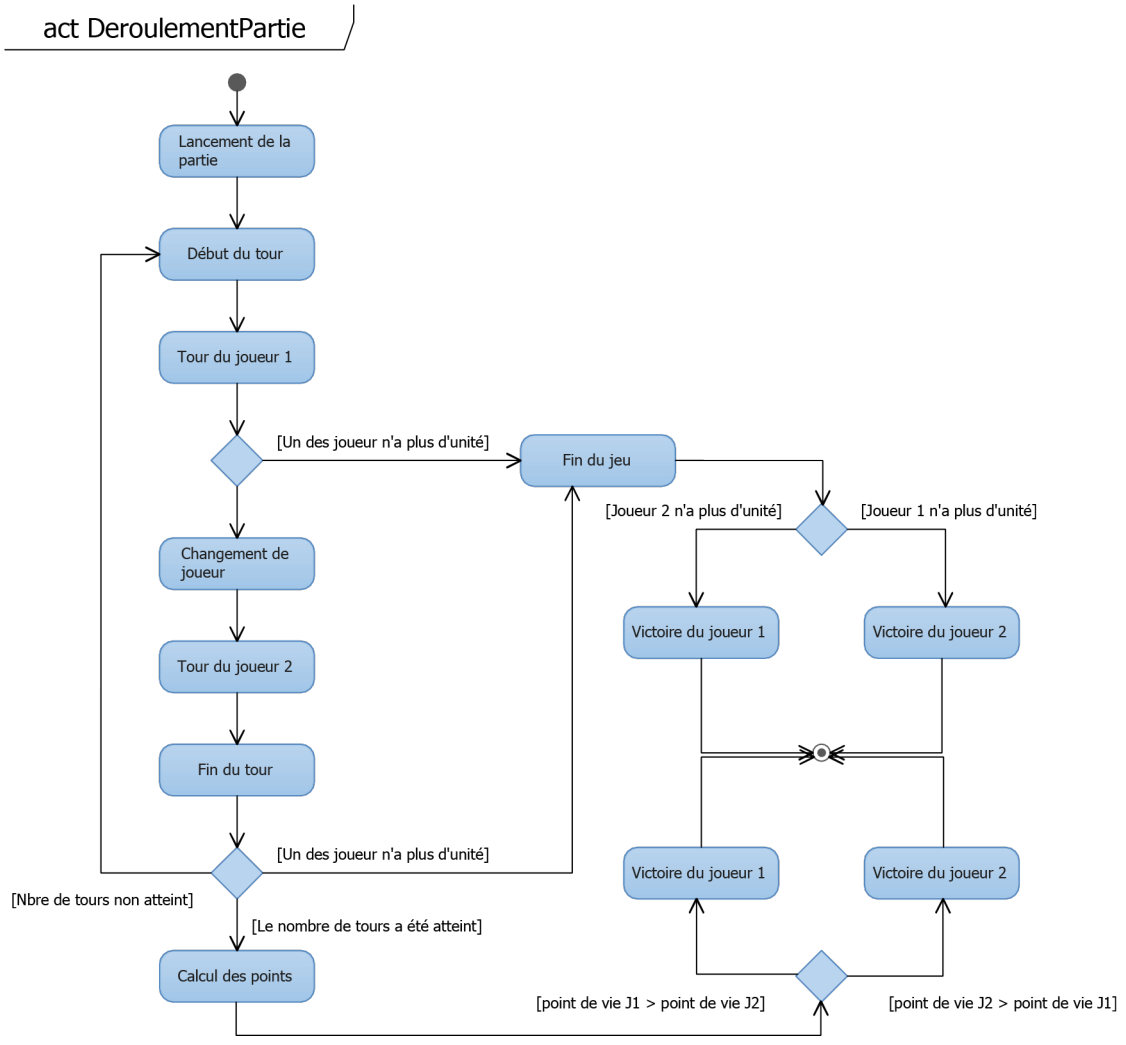

FIGURE 4 – Diagramme d'activité - Déroulement d'une partie

2.1 Déroulement d'un tour de jeu

Lorsqu'un joueur peut jouer, il peut déplacer toutes chacune des unités suivant son nombre de points de mouvement ou choisir de passer son tour. Une unité combattante peut engager un combat si elle se déplace sur une case ennemie. Lorsqu'un joueur à finit son tour, il clique sur le bouton "Fin du tour". Le tour peut cependant être arrêté prématurément si un des joueurs perd un combat et n'a plus d'unité, le jeu se termine alors.

2.1.1 Diagramme de cas d'utilisation

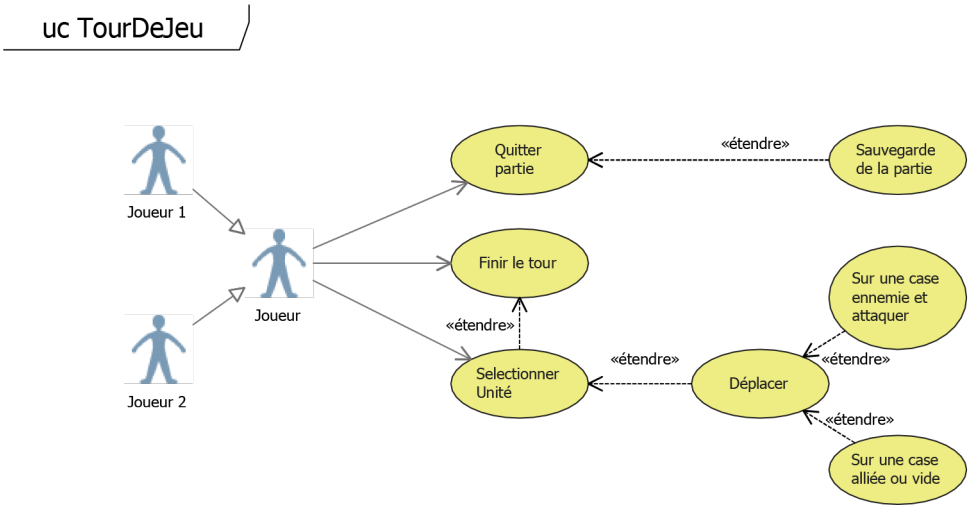

FIGURE 5 – Diagramme de cas d'utilisation - Déroulement d'un tour de jeu

Le diagramme de cas d'utilisation ci-dessus (FIGURE 5) illustre le déroulement d'un tour de jeu du point de vue utilisateur. Lorsque c'est à son tour de jouer, l'utilisateur doit choisir une action pour chacune de ses unités. Il a la possibilité de ne rien faire, déplacer l'unité dans une case vide ou d'attaquer le peuple ennemi. À tout moment, le joueur peut décider de finir le tour ou de quitter la partie.

2.1.2 Diagramme d'activités

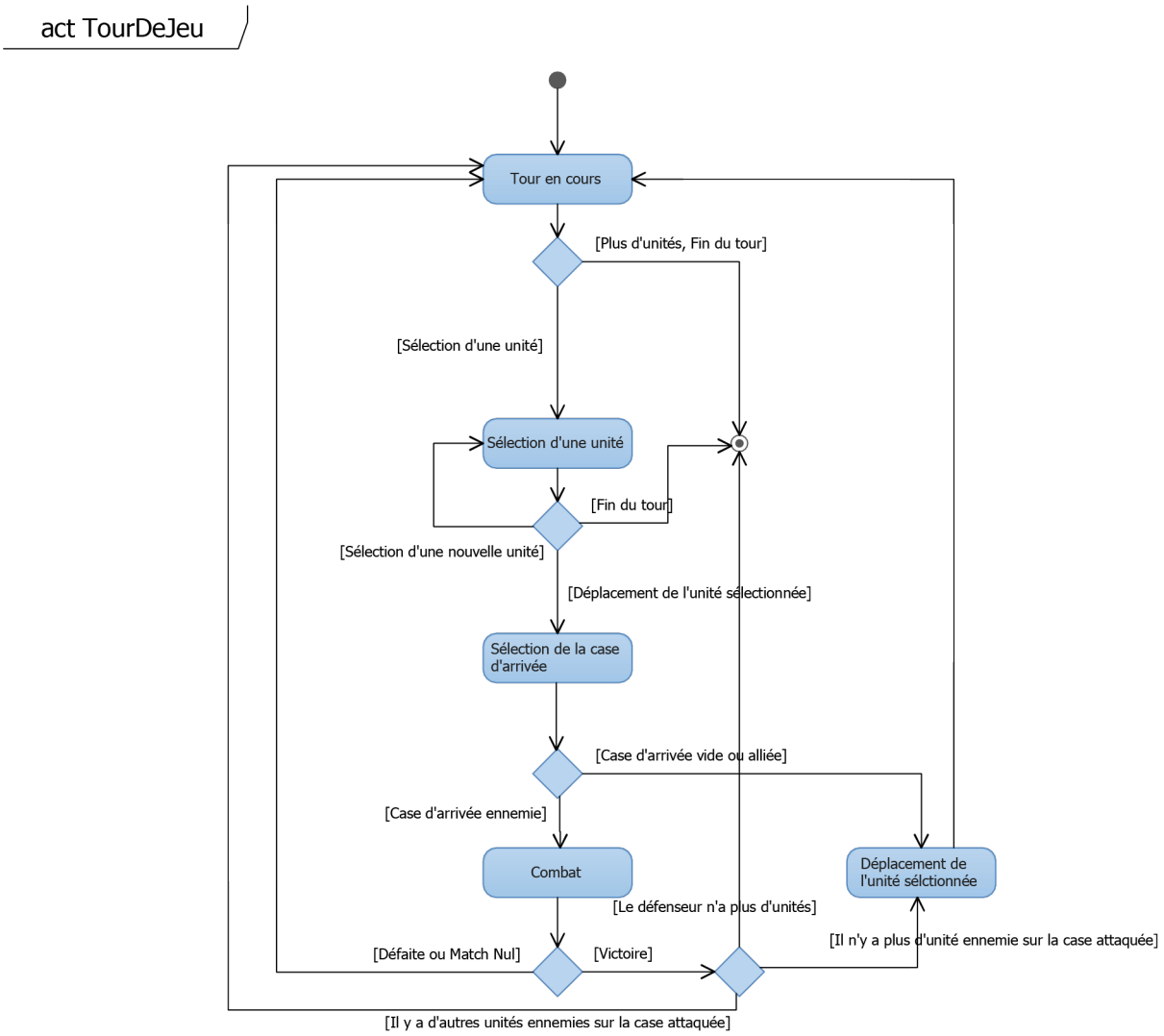

FIGURE 6 – Diagramme d'activité - Déroulement d'un tour de jeu

Le diagramme d'activité ci-dessus (FIGURE 6) illustre le processus du déroulement d'un tour de jeu. Un tour est en cours, si le joueur n'appuie pas sur le bouton "Fin du tour", il doit sélectionner une unité. Il y a alors deux possibilités : soit l'unité passe son tour et on en sélectionne une nouvelle, soit l'unité se déplace. Si l'unité se déplace, on doit sélectionner la case d'arrivée. Si cette case d'arrivée est vide ou alliée, on déplace l'unité sélectionnée, sinon, un combat se déroule. En cas de défaite ou de match nul, on revient au début du processus tour de jeu si le joueur n'a plus d'unité pour continuer le jeu ou s'il décide d'arrêter, le tour est fini, sinon, on retourne dans le processus de sélection d'une unité. D'un autre côté, si l'unité gagne et s'il n'y a plus d'unité dans la case attaquée elle est déplacée sur cette case.

2.1.3 Diagramme de séquence

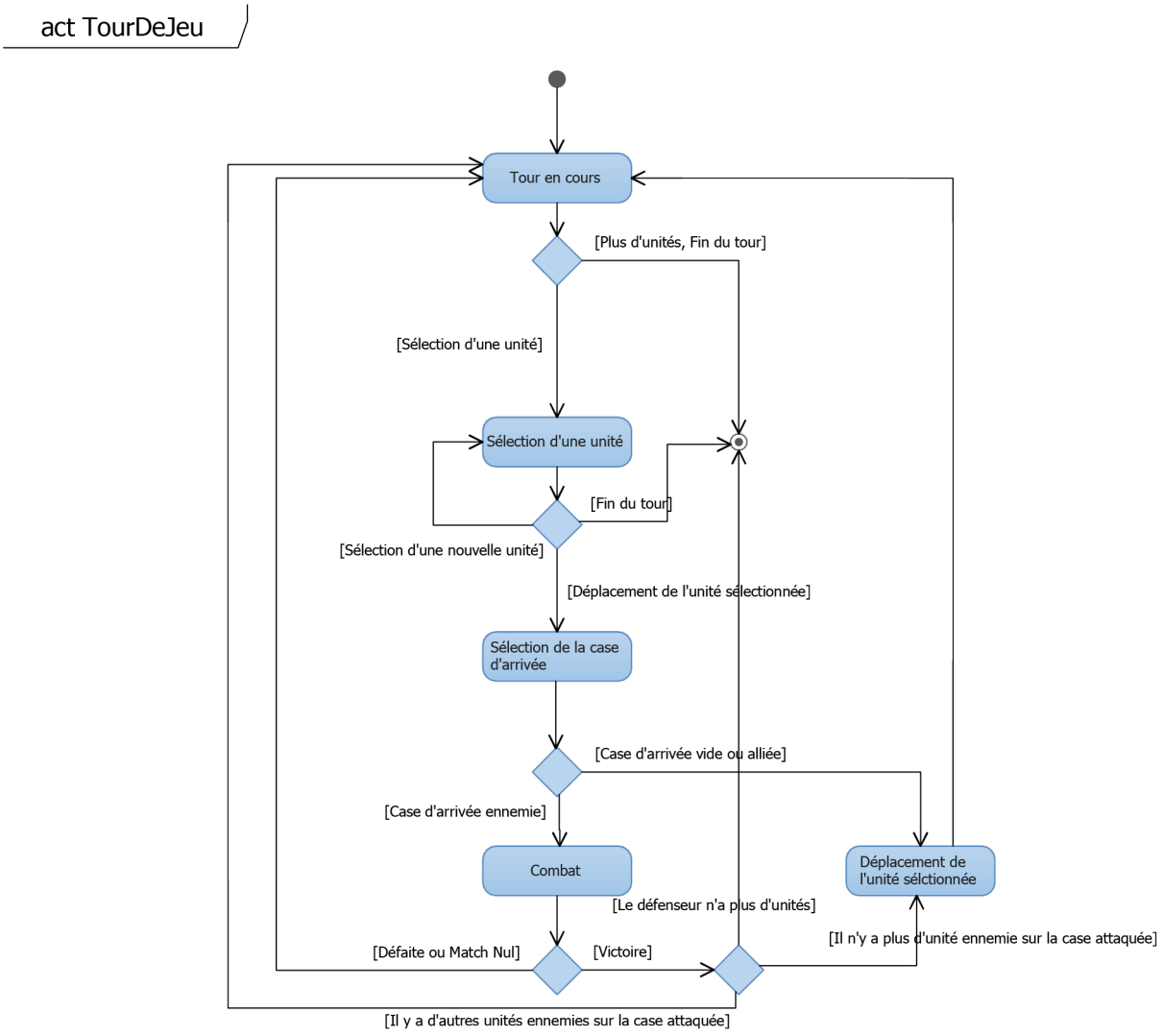

FIGURE 7 – Diagramme de séquence - Déroulement d'un tour de jeu

Le diagramme de séquence ci-dessus (FIGURE 7) illustre les interactions entre objets lors du déroulement d'une partie.

2.1.4 Diagramme d'état transition

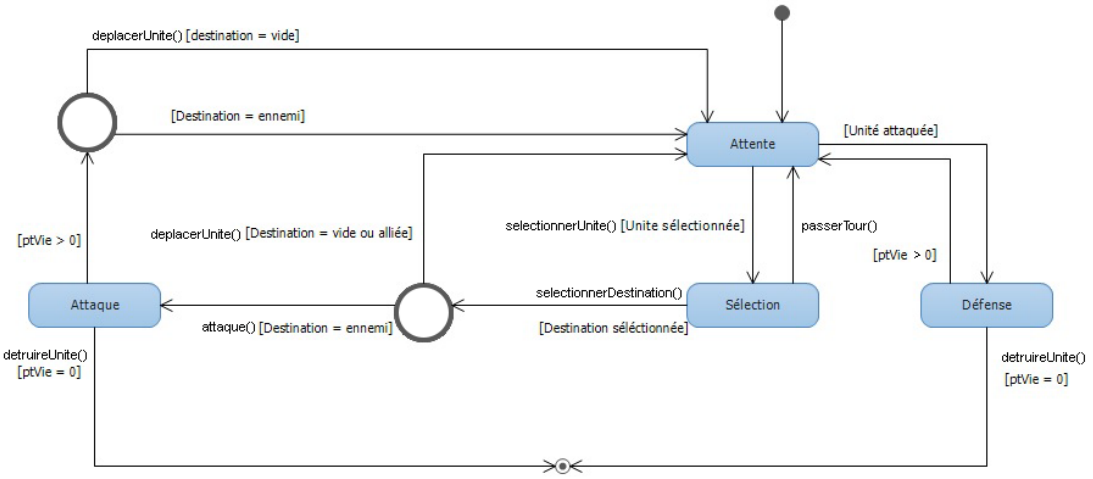

FIGURE 8 – Diagramme d'état transition- Déroulement d'un tour de jeu

Le diagramme de séquence ci-dessus (FIGURE 8) illustre les interactions entre objets lors du déroulement d'une partie.

2.2 Déroulement d'un combat

2.2.1 Diagramme d'activité

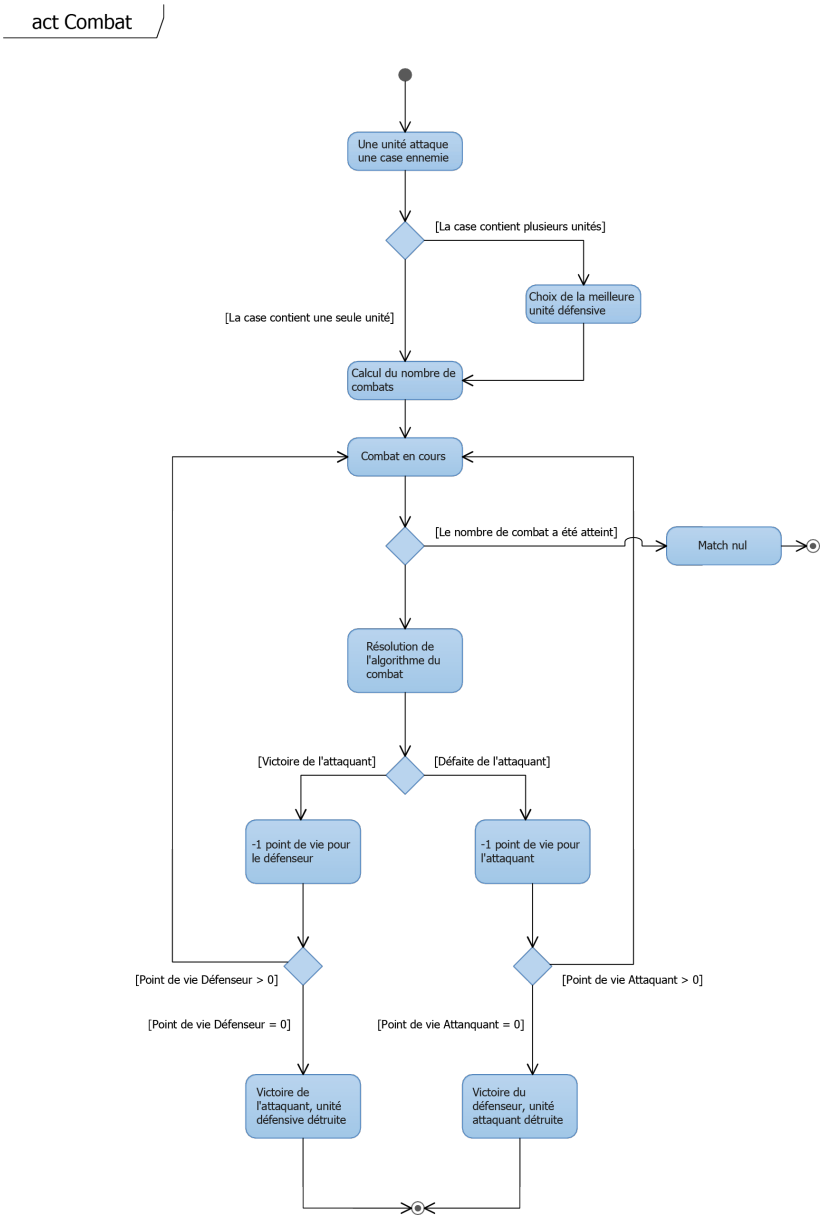

FIGURE 9 – Diagramme d'activité - Déroulement d'un combat

Le diagramme d'activité ci-dessus (FIGURE 9) illustre le processus du déroulement d'un combat. Une unité attaque une case ennemie, si cette case contient plusieurs unités, on choisit la meilleure unité défensive puis on calcule aléatoirement le nombre de combat. On rentre ensuite dans le processus du combat. Tant que le nombre de combat déterminé précédement n'a pas été atteint et que les deux unités ont encore des points de vie, on résout l'algorithme du combat puis l'attaquant ou le défenseur perd un point de vie.

2.2.2 Diagramme de séquence

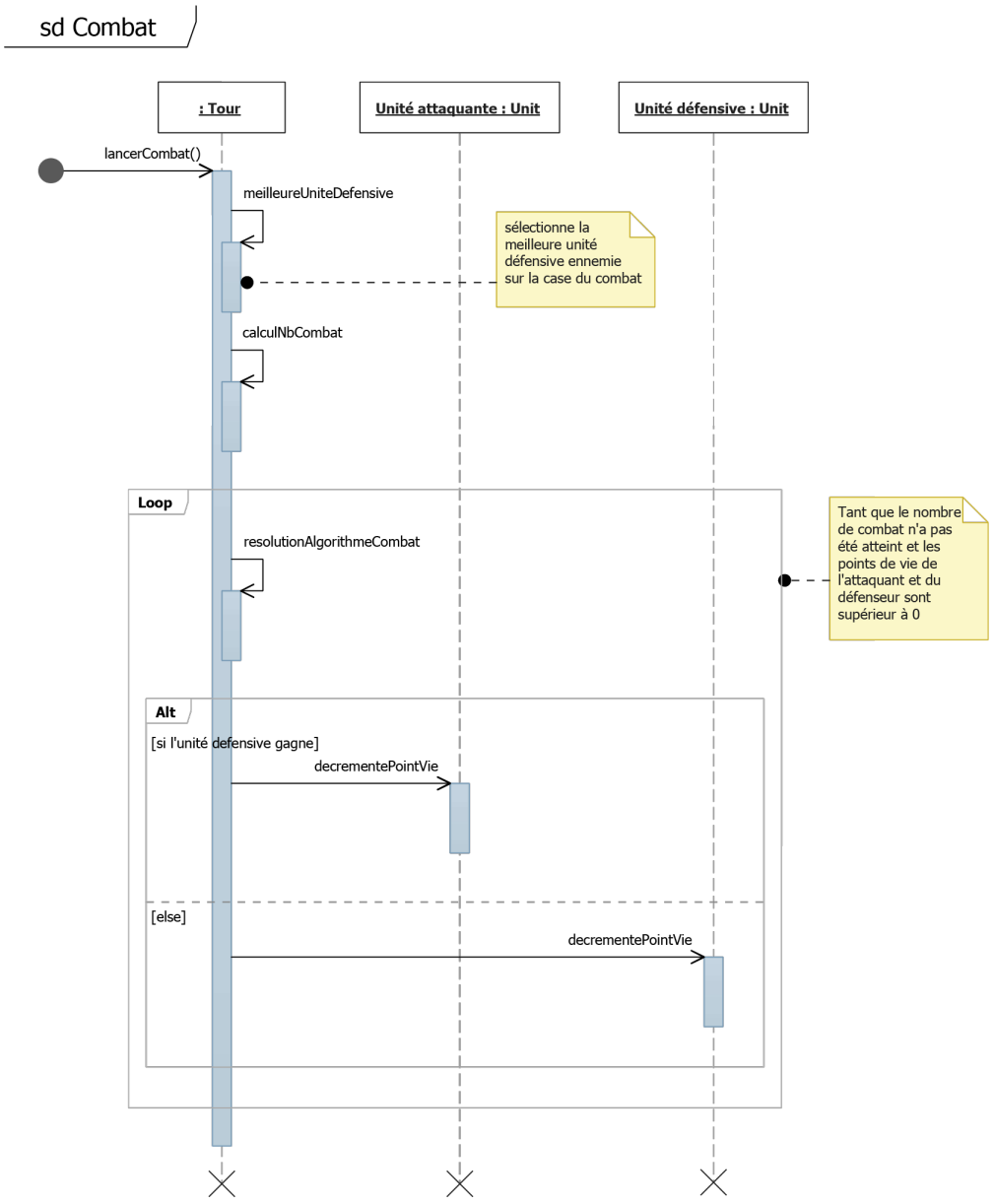

FIGURE 10 – Diagramme de séquence - Déroulement d'un combat

Le diagramme d'activité ci-dessus (FIGURE 10) illustre le processus du déroulement d'un combat. Une unité attaque une case ennemie, si cette case contient plusieurs unités, on choisit la meilleure unité défensive puis on calcule aléatoirement le nombre de combat. On rentre ensuite dans le processus du combat. Tant que le nombre de combat déterminé précédement n'a pas été atteint et que les deux unités ont encore des points de vie, on résout l'algorithme du combat puis l'attaquant ou le défenseur perd un point de vie.

3 Diagramme de classe

Le diagramme de classe ci-dessous illustre l'ensemble de nos classes ainsi que l'ensemble des relations entre celles-ci. Ce diagramme fait abstraction des aspects temporels et dynamiques. Chaque classe décrit les responsabilités, le comportement et le type d'un ensemble d'objets. Au sein même de ce diagramme nous observons différentes parties significatives de notre jeu : • Une partie consacrée à la création d'une partie et à la modélisation de la carte sous forme d'un patron de conception : un monteur. On retrouve également une Stratégie permettant le choix du type de carte. • Une seconde partie représente l'ensemble des classes liées à la Carte. Elles sont visibles sous la forme d'un poids-mouche. Nous observons plusieurs types de cases (Plaine, Désert, Montagne et Forêt) héritant de la classe CaseImpl et créé à partir de la classe FabriquecaseImpl.La classe CarteImpl est composée de plusieurs objets de la classe CaseImpl. • Nous retrouvons une partie dédié aux objets appartenant aux joueurs. Un joueur possède un peuple. Il existe trois types de peuples (Orc, Nain ou Elfe). Un joueur possède également des unités. Ces unités peuvent être de trois types différents en fonction du peuple. • Une dernière partie rassemble les classes se rapportant au jeu. La classe JeuImpl possède l'ensemble des méthodes permettant l'accès aux joueurs, au gagnant, au calcul des points, etc. La classe TourImpl possède l'ensemble des méthodes permettant de gérer le tour de jeu d'un joueur (combattre, déplacer ou sélectionner les unités). Dans la suite, nous détaillerons l'ensemble des patrons de conception utilisés.

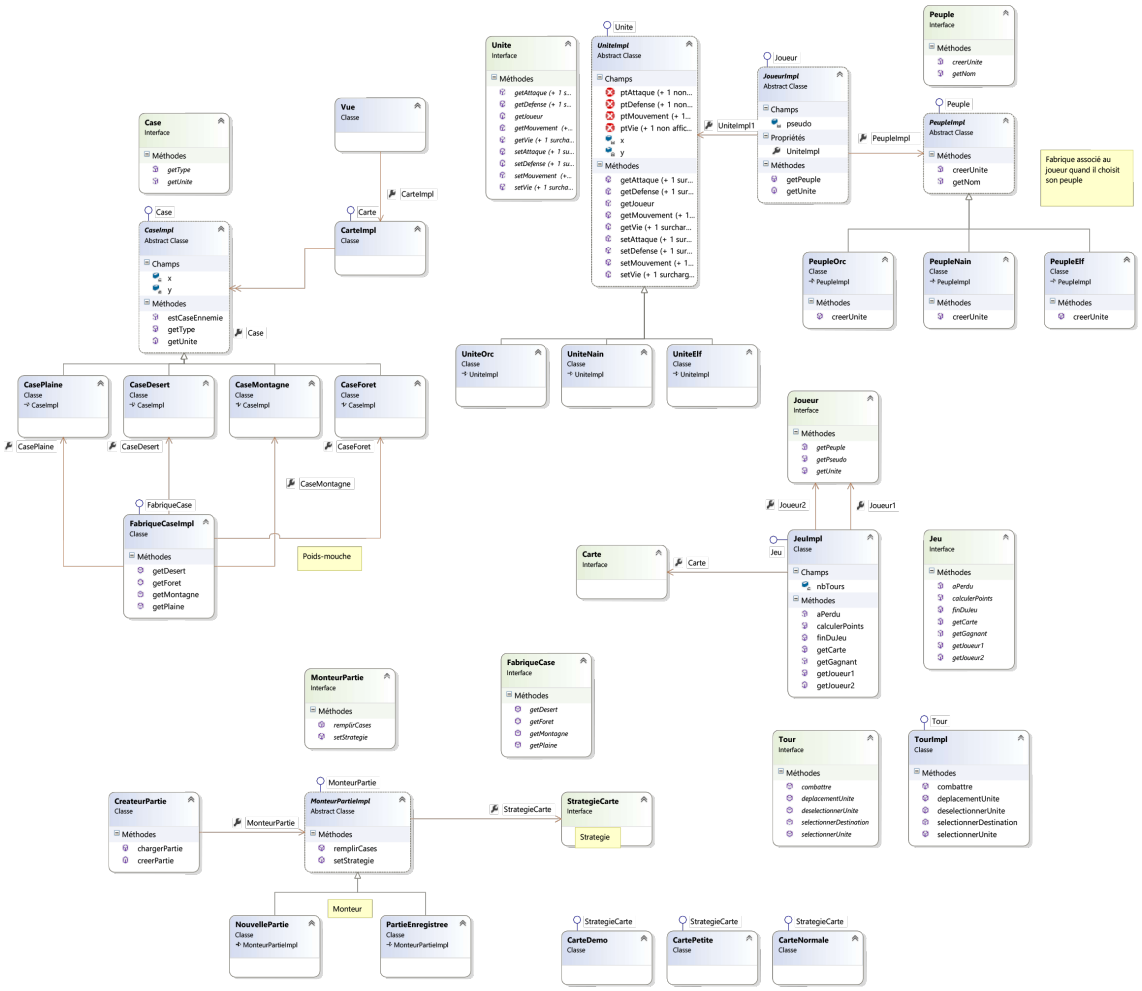

FIGURE 11 – Diagramme de classe - Modélisation globale

3.1 Fabrique

Afin de gérer la création des peuples au sein de notre jeu, nous avons mis en place dans le diagramme de classe une fabrique consacrée à la manipulation des peuples. Le but de notre fabrique est de définir un objet (ici un peuple) dédié à la création d'autres (nain, orc ou elfe). Nous pouvons ainsi au sein même de notre fabrique fournir des opérations de constructions au nom explicite.

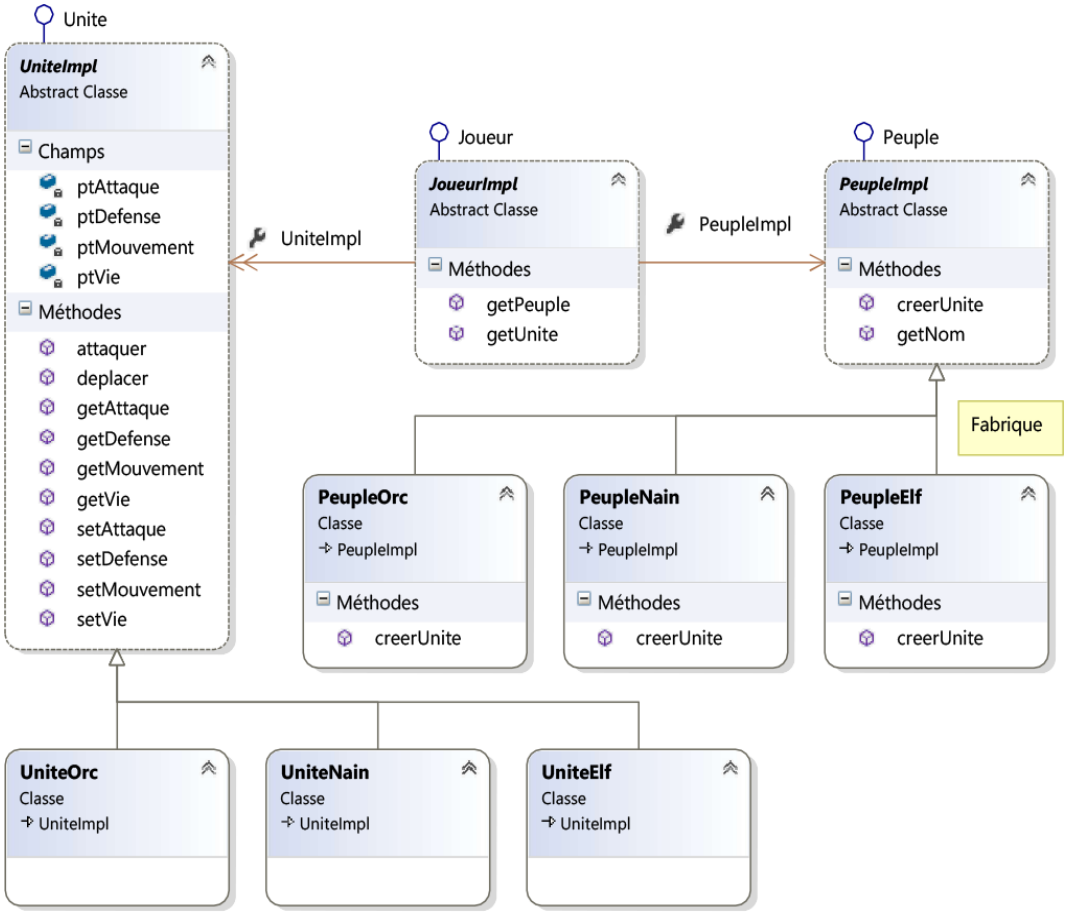

FIGURE 12 – Diagramme de classe - Fabrique

3.2 Monteur

Nous avons choisi d'utiliser un monteur pour faciliter l'implémentation de la création d'une partie ou de la création d'une partie enregistrée. La classe CreateurPartie définit le processus de création. Quant aux deux classes NouvellePartie et PartieEnregistree, elles héritent de la classe MonteurPartieImpl et constituent différentes implémentations de création. En sachant le type de partie que l'utilisateur veut créer, la mise en place de celle-ci se fera différemment. En effet, le remplissage des cases se fera de façon aléatoire lors d'une création d'une nouvelle partie alors qu'il sera défini dans le cas d'un chargement d'une partie enregistrée.

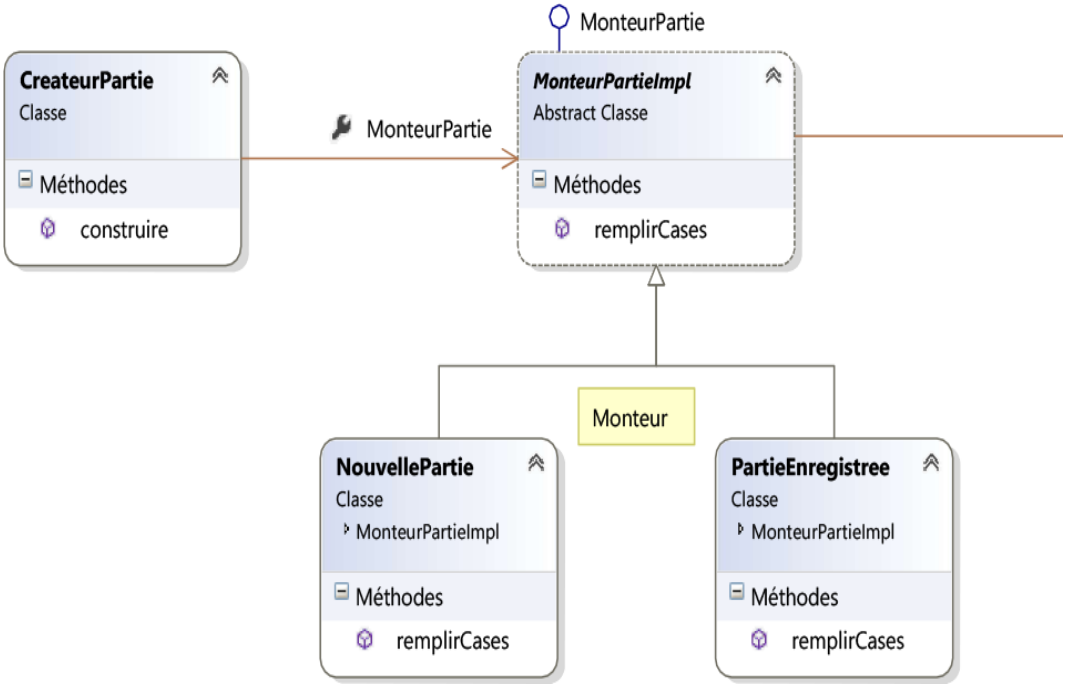

FIGURE 13 – Diagramme de classe - Monteur

3.3 Poids-mouche

Pour la modélisation de la carte, nous avons mis en place un poids-mouche. En effet, dans notre cas nous devons instancier un grand nombre d'objets (cases) ce qui s'avère coûteux en mémoire. Grâce au poids-mouche, il est possible de réduire le nombre d'objets à instancier si tous les objets sont semblables et se différencient seulement sur quelques paramètres. Ici, notre carte se compose de plusieurs cases de types différents. La mise en place du poids-mouche va nous permettre d'obtenir un coût en mémoire plus faible.

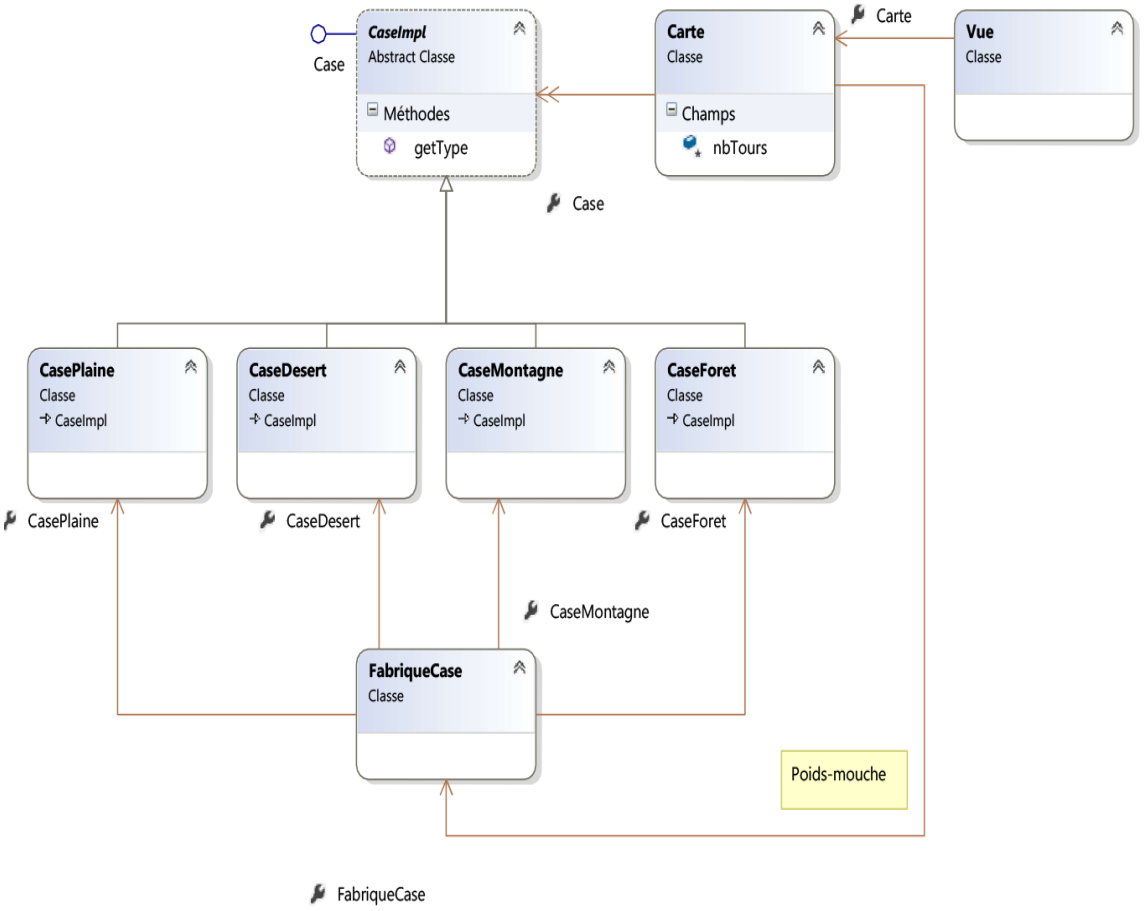

FIGURE 14 – Diagramme de classe - Poids-mouche

3.4 Stratégie

Stratégie : Nous voulons que e comportement de notre carte évolue en fonction du contexte choisi par le joueur. En effet, en fonction que le joueur choisisse une carte démo, une petite carte ou une carte normale, on ne créera pas la carte de la même façon. C'est pourquoi nous avons mis en place une Stratégie.

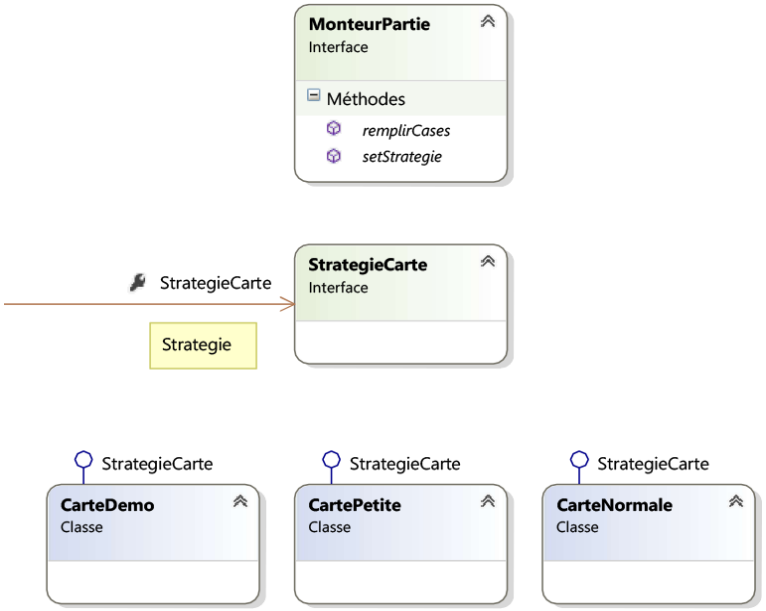

FIGURE 15 – Diagramme de classe - Stratégie

Conclusion

La modélisation du jeu par le biais de différents diagrammes UML nous a permis de mieux appréhender la réalisation de l'implémentation du jeu. En effet, grâce à ces diagrammes nous avons pu séparer les parties importantes de notre jeu en différentes classes. Nous avons pu réfléchir à comment le jeu va fonctionner, et comment nous allons implémenter chaque classe du jeu. Le fait d'utiliser des diagrammes différents a pu mettre en évidence une vision globale du comportement fonctionnel du système, les interactions entre les acteurs et le système selon un ordre chronologique et le déclenchement des différents évènements en fonction des états du système. L'utilisation de patrons de conception aura pour effet de simplifier la partie implémentation du jeu et améliorera par conséquent cette implémentation. A partir de ce travail sur la modélisation du jeu, nous pouvons d'ors et déjà générer le code lié à nos diagrammes. Nous aurons dans ce cas une base sur lequel nous allons nous appuyer pour débuter la partie programmation de notre jeu.

Table des figures