3iLINGENIEURS
</les sciences informatiques>

Dossier de conception – ProjetMars

Équipe Tatou composée de :

- *JOURET Clément*
- *CAGNIARD Guillaume*
- *BOUVIALA Théo*
- *ESPINOSA Paul*

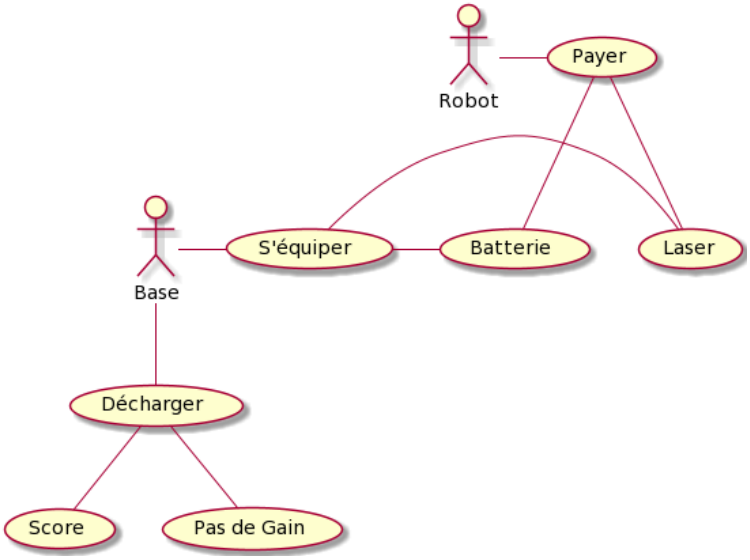
Introduction :

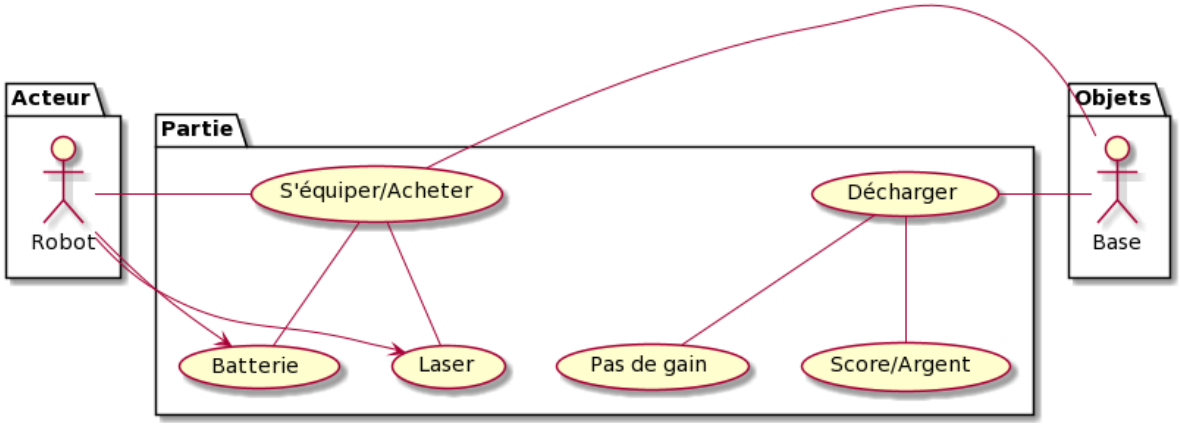
Ce document a pour but de faire comprendre la conception choisie, clarifier l'organisation de notre projet.

Il possède deux buts initiaux : le premier est de vous présenter les différents diagrammes UML que nous avons réalisé, puis le second est d'expliquer chacun de nos choix, pouvoir justifier nos décisions.

Pour commencer, nous présenterons le diagramme de cas d'utilisations, concernant l'analyse des besoins, en suivant nous retrouverons le diagramme de classe qui correspond plus à l'analyse du domaine et pour finir nous auront les diagrammes de séquence et d'état-transition qui eux, schématisent une analyse applicative.

Diagramme de cas d'utilisations d'une *Partie*

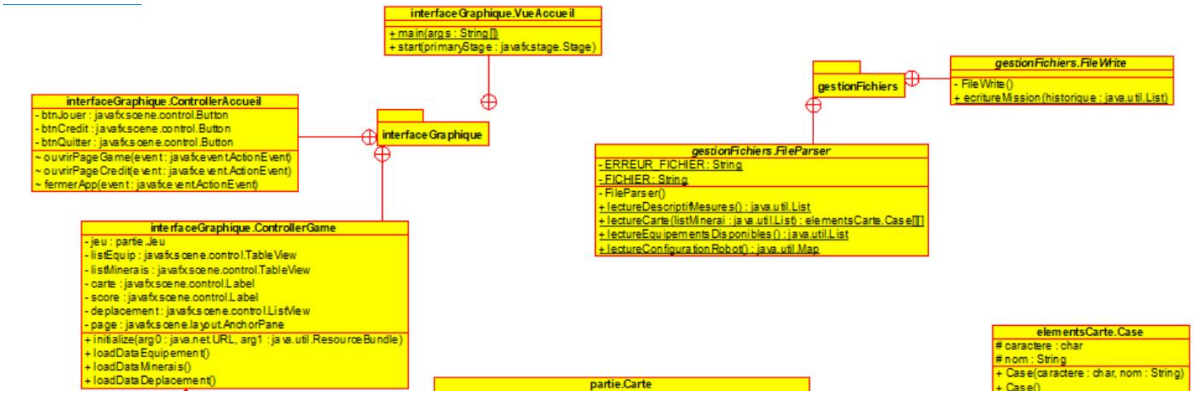




Au sein d'un diagramme de cas d'utilisations, comme son nom l'indique, nous allons retrouver les différentes « actions », « processus » qui vont pouvoir se dérouler tout au long d'un phénomène, dans notre cas : une partie.
Les acteurs et objets, comme le robot, la base, la carte, le laser, la batterie interagissent entre eux, nous pouvons voir quels acteurs sont concernés par quelles actions.

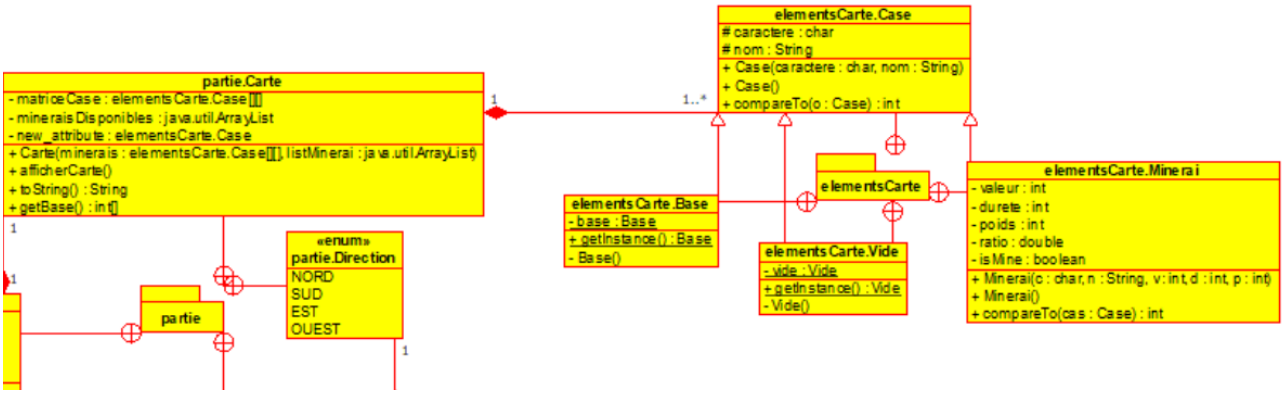
Diagramme de classe de notre *Projet*

Packages interfaceGraphique et gestionFichiers :



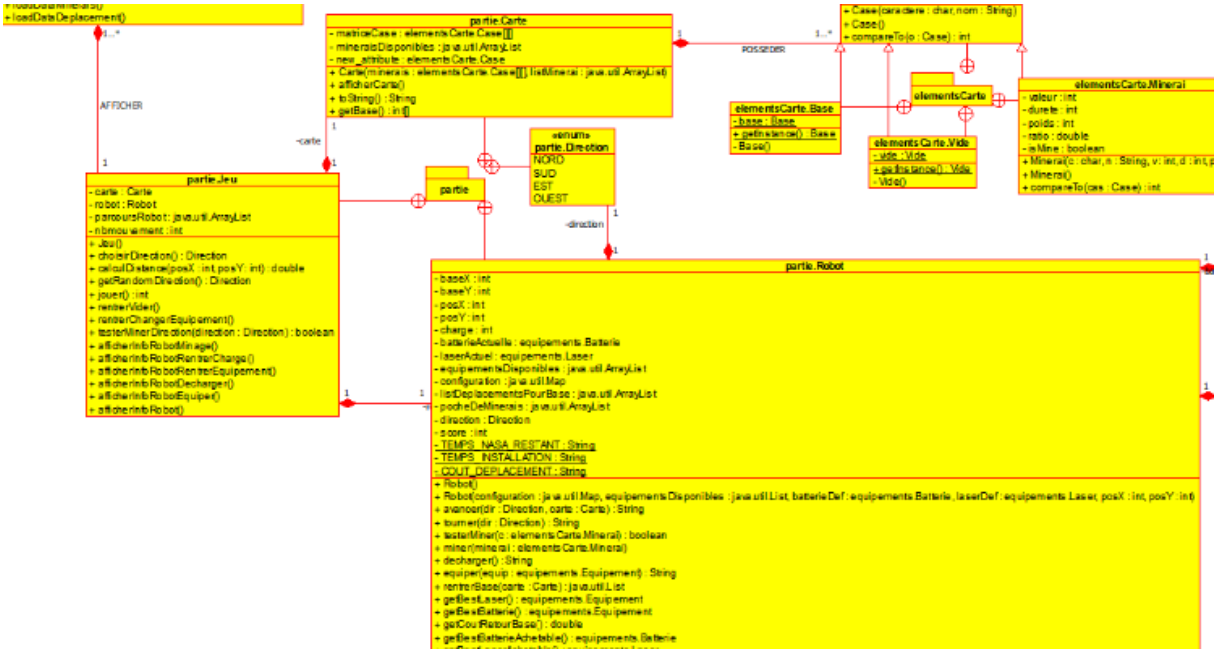
L'interface graphique a pour but de gérer l'affichage de notre jeu, grâce à JavaFX.
Celle-ci a donc besoin d'une vue, pour savoir quels éléments sont à afficher, mais ainsi que deux contrôleurs : Accueil et Game. Comme leurs noms l'indiquent, ils vont respectivement contrôler l'accueil de notre application, puis la partie « Jeu » au sein de l'application.
Plus à droite, on retrouve les classes FileParser et FileWriter qui vont nous servir à traiter les fichiers liés au jeu, notamment les fichiers .txt. L'objectif principal est de réussir à parcourir les fichiers afin de les lire, et de stocker les données lues pour les manipuler par la suite.

Package elementsCarte et début partie :

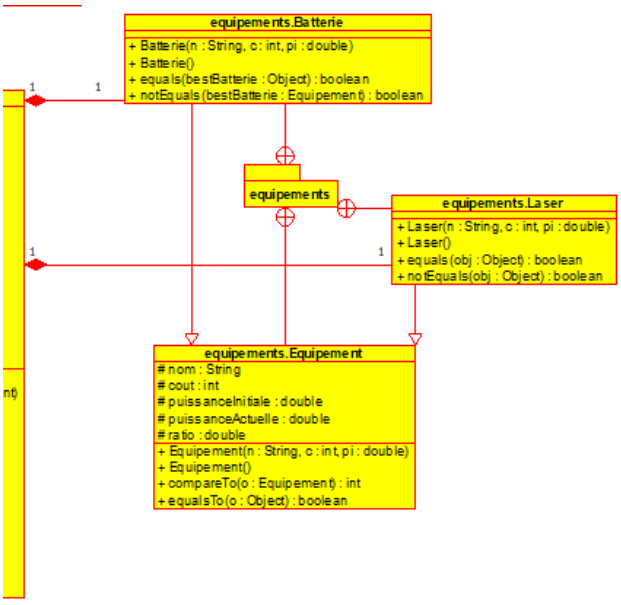


Ici, nous retrouvons les éléments constituants la carte : une carte est en réalité une matrice de cases. Une case elle-même peut être un minerai, la base du Robot, sinon la case est vide.
Ensuite, nous retrouvons le package partie, dont l'on observe uniquement le début pour le moment. Dans une partie, il y a un jeu qui est constitué d'une carte et d'un robot. Nous savons donc déjà comment est conceptualisée la carte.

Package partie et équipements :



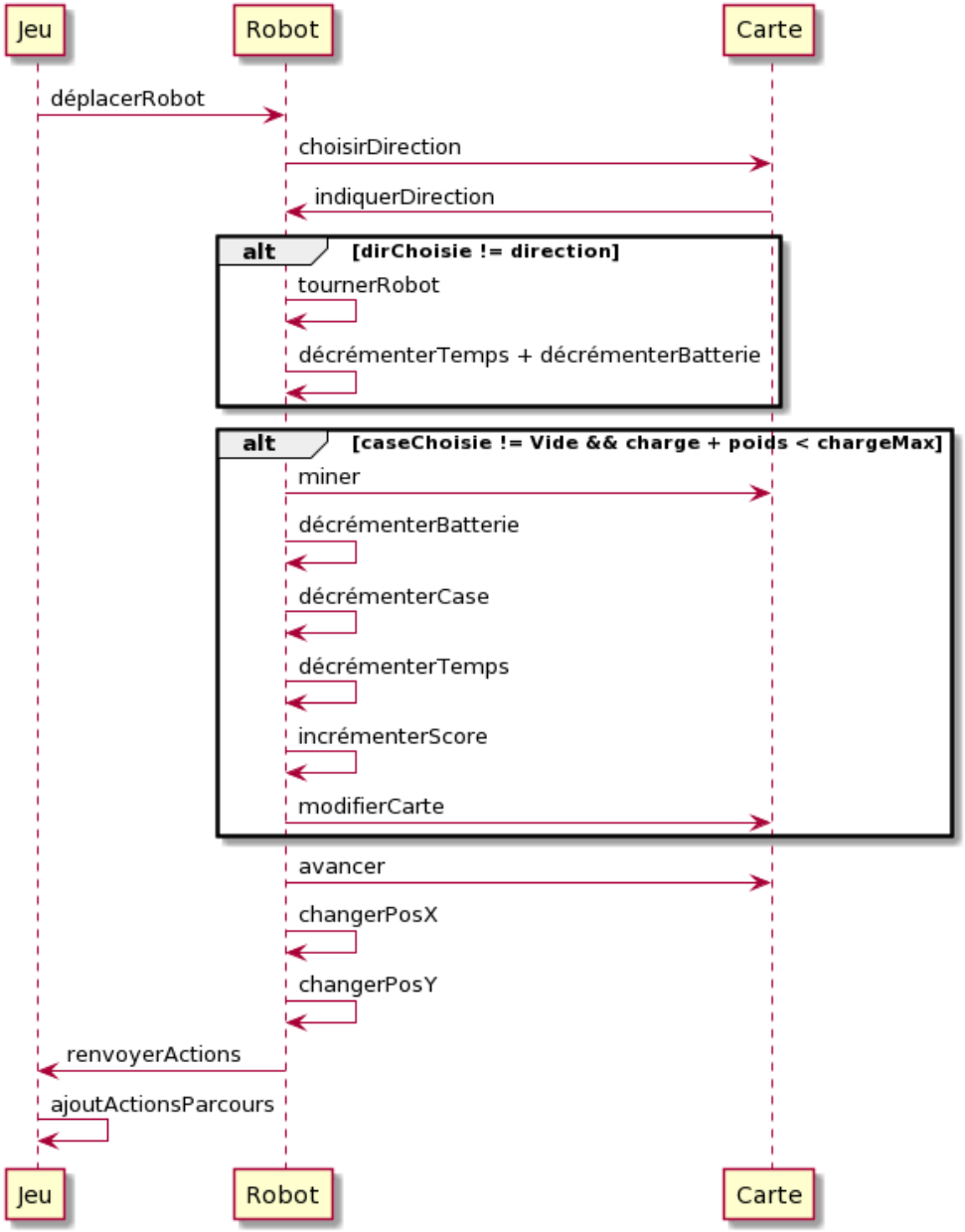
Voilà donc notre jeu et notre robot qui le compose. Dans partie, on retrouve des directions afin que le robot puisse se situer / s'orienter sur la carte pendant le jeu. La majorité des méthodes et attributs sont situés au sein de ces deux classes.



Pour finir, voilà les classes Laser et Batterie, qui sont en réalité des équipements (héritage). On voit ensuite que le robot est composé d'un Laser et d'une Batterie.
C'est la dernière partie sur laquelle nous effectuons un « zoom » pour la détailler. Pour les cardinalités, nous retrouvons souvent la relation « 1…1 » car nous sommes partis du principe où l'évènement se déroule à un instant t, et que par exemple un robot ou une carte ne peuvent interagir qu'avec un seul et même jeu à la fois.

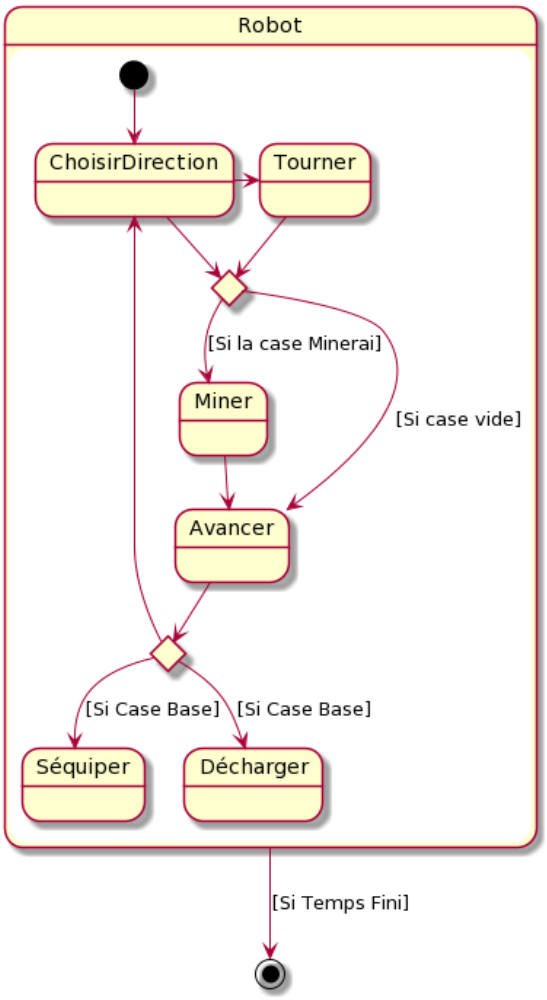
Alors qu'une carte pourrait pourtant appartenir à un ou plusieurs jeux tout au long de son existence. Voilà la justification de notre choix vis-à-vis des cardinalités.

Diagramme de séquence sur l'acteur *Robot*



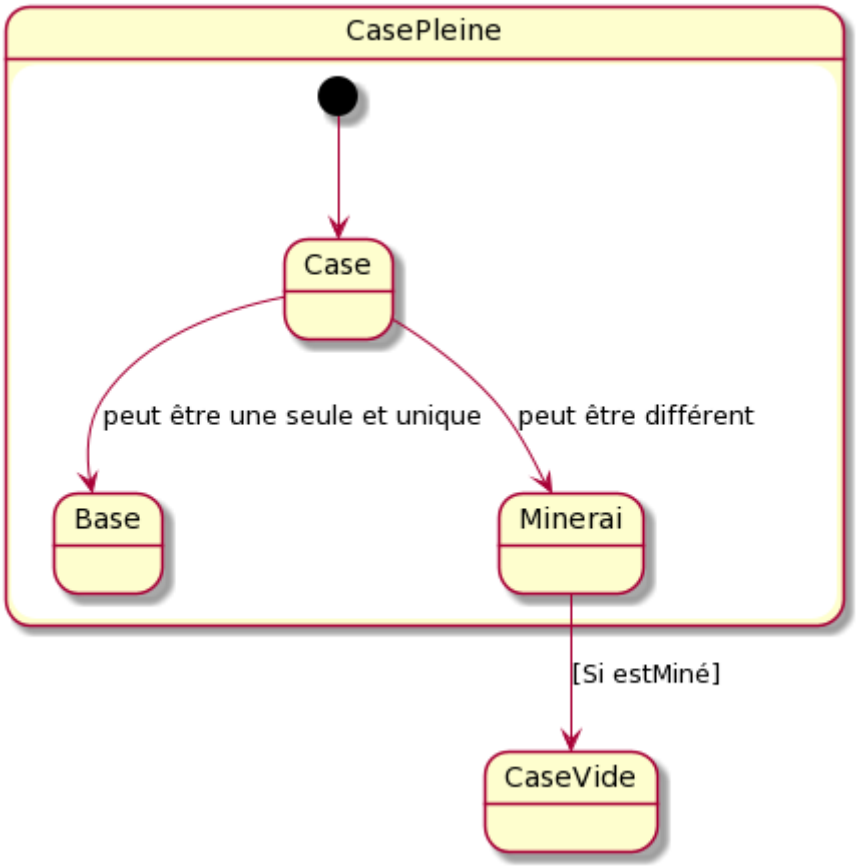
Dans ce diagramme de séquence, nous décrivons les interactions dans le temps durant une partie entre les acteurs et les objets principaux. On peut parler en premier du jeu, ayant besoin d'un robot et d'une carte pour avoir lieu.
C'est donc entre eux que les principales actions s'effectuent, le robot possédant lui-même une batterie et un laser afin de réaliser des actions comme miner, ou se déplacer.
Ces acteurs/objets, notamment le robot et la carte (étant composée de cases), vont s'analyser plus en détails, c'est-à-dire dans quels états peuvent-ils être en fonction de comment il se comportent, dans des diagrammes de transitions que nous allons voir juste après.

Diagramme d'état-transition sur l'acteur *Robot*



Pour ce diagramme, la stratégie a été de synthétiser le scénario du jeu. Notre robot peut se déplacer sur la carte, avec comme options se tourner et miner.
Ensuite, il peut rentrer à la base afin de décharger le butin récolté afin de le transformer en argent. Grâce à cet argent, il pourra s'acheter des nouveaux équipements au fur et à mesure de la partie.
Notre robot, passe alors chronologiquement par les états suivants à chaque « tour » :
ChoisirDirection est l'état lorsqu'il est en train de réfléchir dans quelle direction il va se diriger, ensuite si c'est nécessaire de s'orienter, le robot sera dans un état de rotation.
Après, plusieurs choix apparaissent, si la case dans laquelle il va se déplacer est un minerai, alors on mine puis on avance. Si la case est déjà minée, on avance juste.
Pour finir, après avoir avancé, on effectue un dernier test pour savoir si la case sur laquelle nous nous trouvons actuellement est la base.
Si c'est le cas, décharger et s'équiper, comme vu précédemment, sont deux options qui s'offrent à nous. Dans le cas contraire, on retourne au choix de la direction.

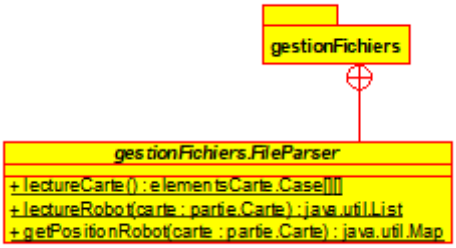
Diagramme d'état-transition sur l'acteur *Case*



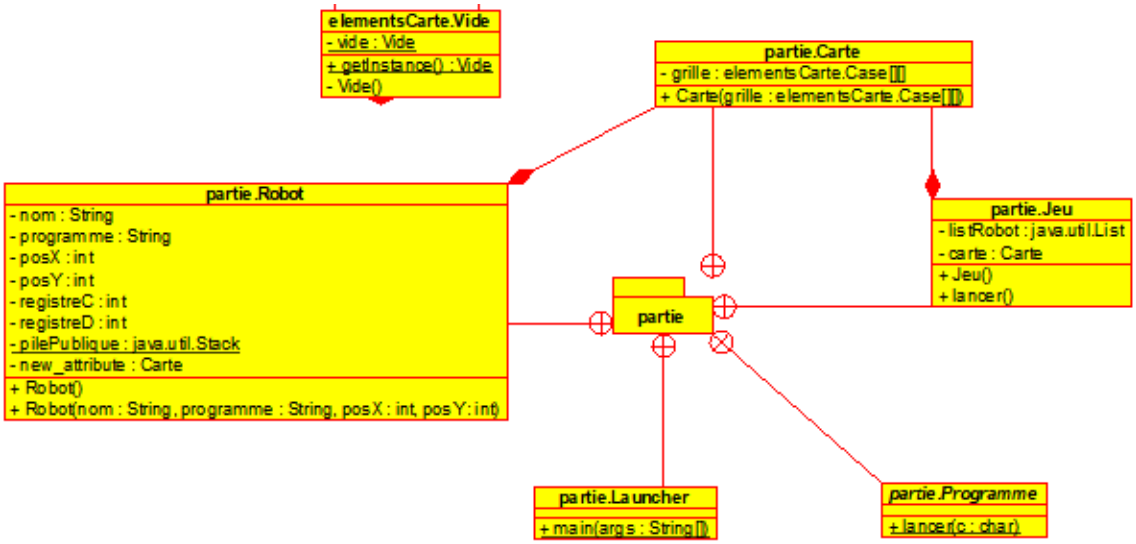
Sachant que sur le dernier diagramme d'état-transition, nous avons effectué beaucoup de tests vérifiant si une case était un minerai, vide ou la base, nous avons trouvé pertinent d'en effectuer un pour les états possibles d'une case.
Donc, lors de la lecture du fichier schématisant la carte, nous définissons l'ensemble des cases. Une case pleine peut être la base, mais il n'y en a qu'une et une seule d'après les règles du jeu, l'ensemble des cases pleines restantes peuvent être différents minerais.
Lorsqu'une case est un minerai, elle peut être minée et devient donc une case vide.
Ce sont tous les états par lesquels une case peut passer durant une partie.

Partie 2 : Combat

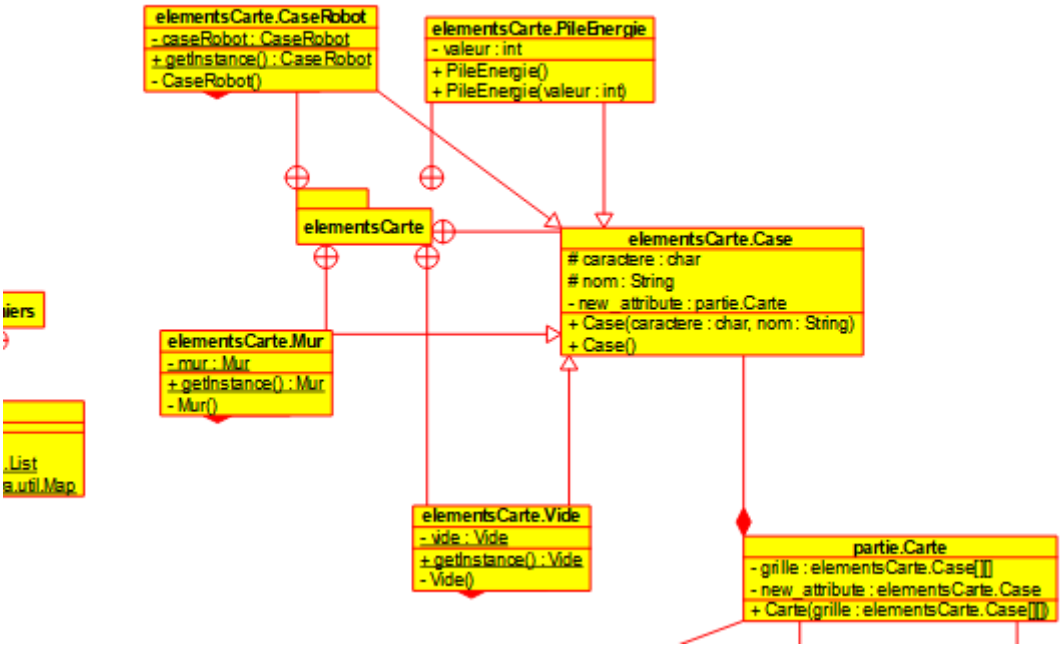
Diagramme de classe Partie 2



Dans le package gestionFichiers on retrouve la classe FileParser qui contient des méthodes statiques. Elles vont nous servir à traiter les fichiers qui sont utiles pour l'initialisation de la partie, notamment les fichiers .txt. L'objectif principal est de réussir à parcourir les fichiers pour ensuite crée une matrice de Case qui représentera notre carte. Depuis cette carte nous pourrons récupérer les robots et les ajouter dans une liste.



Le package partie comporte toutes les classes qui sont nécessaire au bon déroulement d'une partie de combat. La classe Jeu va gérer toutes les interactions entre les classes Carte et Robots. La classe Launcher contiendra notre main pour pouvoir lancer la partie.



Le package elementsCarte comporte toutes les classes qui composent la carte. Les classes CaseRobot, Vide, Mur sont des singletons car on aura besoin que d'une seule instance de ces objets car ils seront toujours identiques. La classe Case est la classe mère des classes Vide, Mur, CaseRobot, Pile Energie. Cet héritage permet de créer par la suite une matrice de Case et d'utiliser la notion de polymorphisme sur les différentes Classes filles.

Diagramme de classe (en entier) :