Projet Jeu Menhir – Livrable 1 : Analyse et diagrammes UML

BROCHOT Jason

PERROT Bryan

02/11/2015

Sommaire

Introduction

Ce projet porte sur l'étude, la conception et la réalisation d'un logiciel de jeu de carte nommé « Jeu du Menhir ». Entre 2 et 6 joueurs peuvent jouer à une partie. Un seul joueur est humain. L'objectif du jeu est de faire pousser plus de menhirs que les autres joueurs. Il existe deux modes de jeu : la partie rapide et la partie avancée.

La partie rapide se déroule sur une seule manche, composée de 4 tours (un tour représentant une saison de l'année). Pour cela, chaque joueur reçoit 2 graines et 4 cartes ingrédient, ces dernières ayant chacune une force qui dépend de la saison en cours et de l'action pour laquelle la carte est utilisée. Ces actions sont au nombre de 3 :

- Offrir la carte ingrédient au géant gardien de la montagne qui donnera au joueur un nombre de graines en échange (le nombre de graines données correspond à la force en vigueur de la carte)
- Utiliser la carte ingrédient pour fabriquer de l'engrais magique. Permettant ainsi au joueur de faire instantanément pousser un nombre de menhirs (le nombre de menhir qui poussent correspond à la force en vigueur de la carte)
- Donner la carte ingrédient aux farfadets chapardeurs en échange de quoi ils remettront au joueur un nombre de graines qu'ils voleront à un joueur cible choisi (le nombre de graines volées correspond à la force en vigueur de la carte)

La partie avancée est différente de la partie rapide en deux points : le nombre de tours et la possibilité de jouer des « cartes alliés » :

Une partie rapide constitue en fait une manche d'une partie avancée, laquelle comporte autant de manches qu'il y a de joueurs. La partie avancée est gagnée par le joueur qui a fait pousser le plus de menhirs sur l'ensemble des manches de la partie. Afin de conserver pour chaque joueur le nombre total de menhirs poussés d'une manche sur l'autre, on utilise une carte de comptage des points par joueur.

Au début de chaque manche, au lieu de recevoir 2 graines, un joueur peut choisir de recevoir une carte alliée, dont il existe deux variantes :

- les taupes géantes : Elles permettent de détruire un certain nombre de menhirs d'un joueur cible choisi (ce nombre est déterminé par la force de la carte qui dépend uniquement de la saison en cours)
- Les chiens de gardes. permet de se protéger des farfadets chapardeurs en diminuant la quantité de graines volées d'un certain nombre (ce nombre est déterminé par la force de la carte qui dépend uniquement de la saison en cours)

Dans ce premier livrable, nous inclurons sous forme de diagrammes UML (cas d'utilisation, classes et séquences) les résultats de notre étude du projet tel que nous le concevrons. Pour chaque diagramme, nous justifierons et détaillerons ses éléments afin de présenter au mieux notre future réalisation. Pour conclure nous décrirons quelles parties de la modélisation de l'application sont susceptibles de changer durant le développement ainsi que, à l'inverse les aspects de la modélisation qui sont fixés définitivement.

I. Diagramme de cas d'utilisation

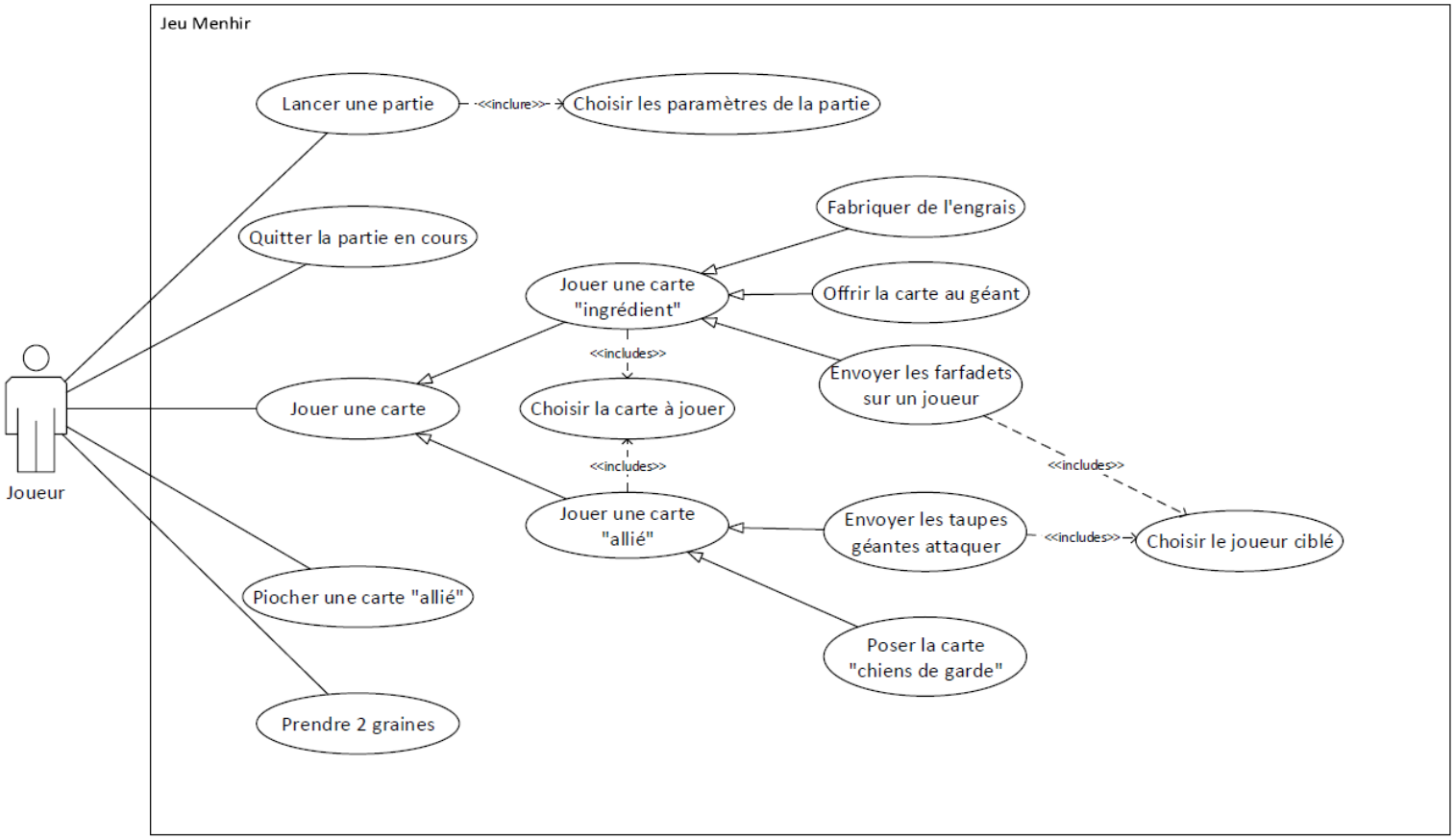

Figure 1 : Diagramme des cas d'utilisation

Dans le cadre de ce projet, un seul acteur interagit avec le système : il s'agit du joueur (l'utilisateur). Lors de l'analyse du sujet, nous avions initialement inclus un autre acteur, représentant le joueur virtuel. Choix sur lequel nous nous sommes ravisés, estimant que celui-ci faisait partie du système.

Ainsi, le premier cas d'utilisation associé au joueur est l'action de lancer une partie, qui inclue le paramétrage de cette dernière. En effet, afin de jouer, le joueur doit démarrer une partie depuis l'application en mentionnant le type de partie, le nombre de joueurs etc...

II. Diagramme des classes

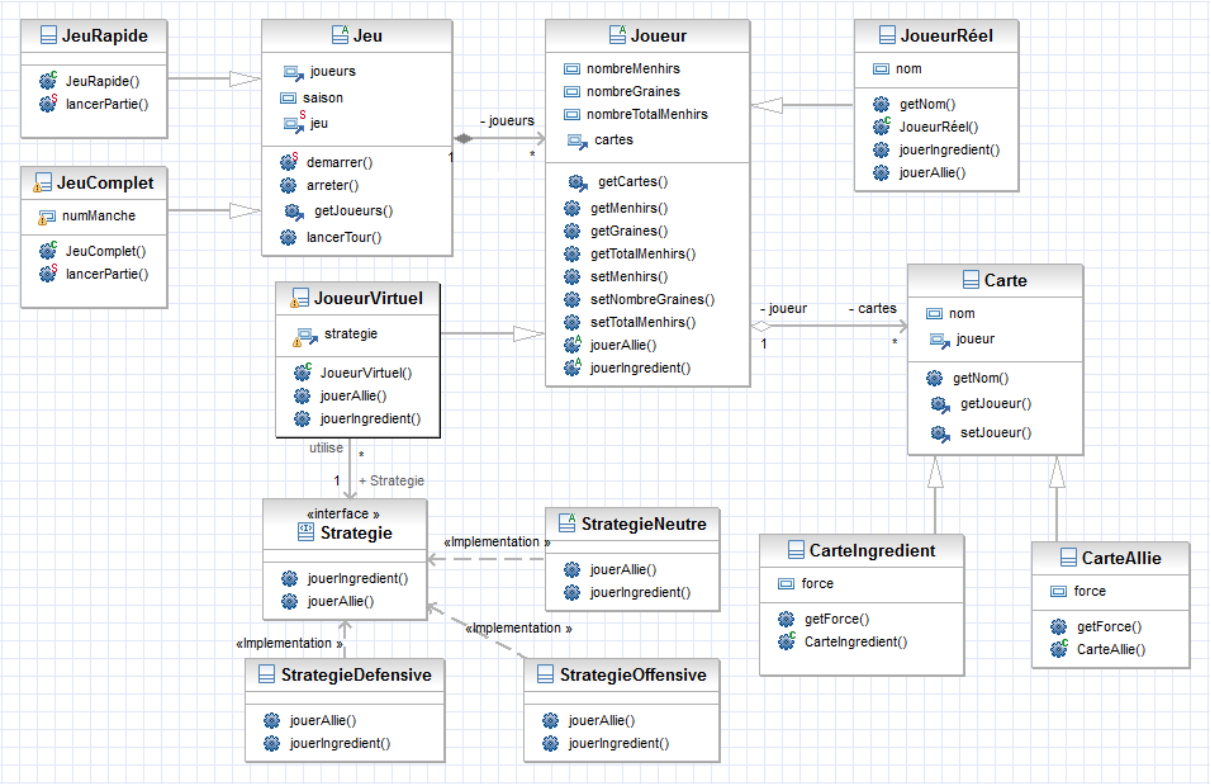

Figure 2 : Diagramme des classes

La classe Jeu représente la racine de l'application, et est l'objet du patron de conception du *Singleton*. En effet, pour lancer une partie, il faudra systématiquement passer par cette classe qui, par le biais de sa méthode "demarrer()" qui gère le *Singleton*, permettra d'instancier une et une seule partie simultanément. Cette classe est abstraite et se spécialise en deux autres classes, que sont la classe JeuRapide et la classe JeuComplet, qui ont chacune une méthode "lancerPartie()" différente. De plus, la classe Jeu possède un attribut "saison" représentant la saison en cours ainsi qu'une collection de joueurs, représentant tous les joueurs participants. Le cycle de vie d'un objet de type Joueur est lié à son agrégat, qui est l'objet de type jeu, et un objet Joueur doit être associé à l'objet de type Jeu, ce qui explique le lien de composition entre la classe Jeu et la classe Joueur. Il n'est nécessaire de pouvoir naviguer que de Jeu vers Joueur, puisqu'il n'est pas intéressant de faire l'inverse étant donné que le jeu est un *Singleton*.

La classe Joueur possède 4 attributs, respectivement le nombre de menhirs adultes du joueur, son nombre de graines, le nombre cumulé de menhirs adultes qu'il aura eu à chaque fin de partie et un attribut "cartes", qui est une collection de cartes dont il possède. Nous avons intégré à la classe des "getters", afin de récupérer les valeurs des attributs, et des "setters", afin de contrôler la modification des valeurs (pas de nombre de menhirs négatifs etc). La classe possède des méthodes "jouerAllie()" et "jouerIngredient()" abstraites qui déterminent le type d'action que le joueur doit effectuer (s'il s'agit de son tour, le joueur doit jouer une carte

ingredient, sinon il peut jouer une carte allié) et l'action à effectuer pour les cartes ingrédient. Enfin, elle a pour attribut une collection d'objets de type Carte, représentant les cartes en main du joueur. Le cycle de vie de ces cartes n'étant pas lié à celui du joueur (une carte peut changer de main), le lien qui nous a semblé le mieux représenter la réalité est l'agrégation. Lors de notre étude du sujet, il ne nous a pas semblé utile de permettre la navigabilité dans les deux sens : il nous suffit de pouvoir récupérer les cartes d'un joueur. La classe Joueur se spécialise en deux autres classes : JoueurRéel et JoueurVirtuel.

La classe JoueurRéel ne présente pas beaucoup de différences par rapport à la classe mère. Elle implémente les méthodes "jouerAllie()" et "jouerIngredient()".

La classe JoueurVirtuel quant à elle met en place le patron de conception *Strategy*. Celui-ci consiste en un attribut du type d'une interface qu'implémentent des classes. Nous n'avons ici besoin que d'une navigabilité de JoueurVirtuel vers Strategie, puisque l'inverse n'a aucun intérêt. Le patron de conception *Strategy* a pour intérêt d'attribuer de façon dynamique un algorithme à un objet au moment de l'exécution (liaison dynamique par opposition à la liaison statique).

Les 3 classes StrategieDefensive, StrategieOffensive et StrategieNeutre, qui implémentent l'interface Strategie, définiront une façon de jouer pour le joueur virtuel. Ainsi, "joueurVirtuel.jouerIngredient()" appellera "joueurVirtuel.strategie.jouerIngredient()" qui dépendra du type de "joueurVirtuel.strategie".

La classe Carte se spécialise en deux types : CarteIngredient et CarteAllie.

La classe CarteIngredient dispose d'un attribut "force", de type tableau de tableau d'entier, afin de caractériser la force de la carte à une saison donnée pour une action donnée.

La classe CarteAllie dispose elle aussi d'un attribut "force", mais celui-ci est d'un type différent. En effet, ces cartes n'ayant qu'une valeur par saison, elles se différencient des cartes ingrédient. Le fait que les deux classes filles aient un attribut de même nom mais de type différent empêche d'utiliser une seule et unique classe.

III. Diagramme de séquences

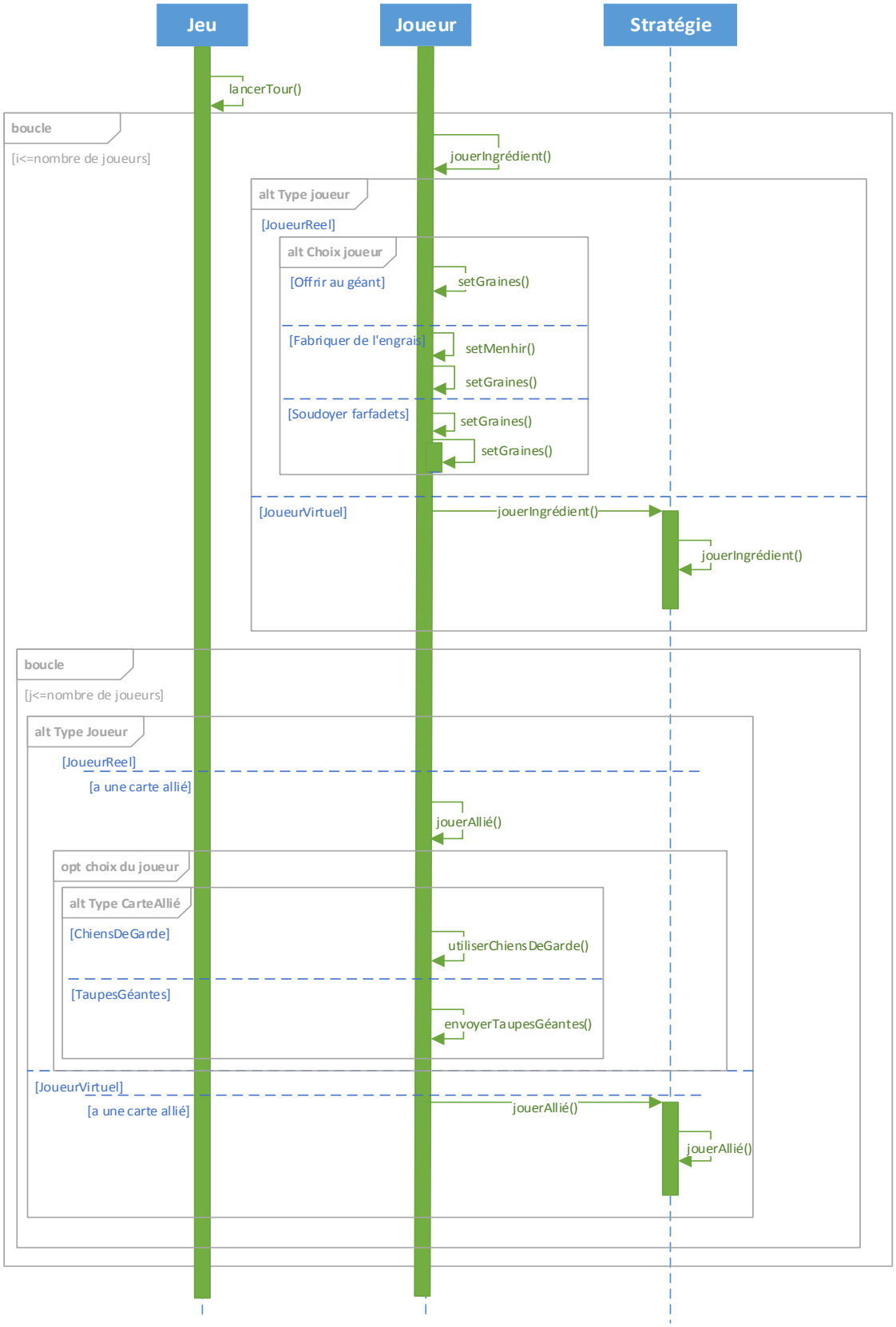

Figure 3 : Diagramme de séquence

Le diagramme de séquence reporté en Figure 3 représente le déroulement d'un tour du jeu.

Le déroulement d'un tour commence avec l'exécution de la méthode lancerTour() de la classe Jeu. Une boucle qui parcourra tous les joueurs est alors ouverte. Dans un premier temps le joueur dont c'est au tour de jouer jouera la carte ingrédient de son choix, puis chacun des joueurs qui possède une carte allié aura la possibilité de la jouer avant que le joueur suivant ne joue une carte ingrédient de son choix.

Pour le jeu de la carte ingrédient, il convient de distinguer 2 cas : le joueur qui joue est le joueur réel, et le joueur qui joue est un joueur virtuel.

le joueur réel a le choix entre offrir sa carte au géant, l'utiliser pour fabriquer de l'engrais magique ou la donner aux farfadets chapardeurs. S'il offre sa carte au géant, son nombre de graines sera mis à jour. S'il l'utilise pour fabriquer de l'engrais, son nombre de graines et son nombre de m menhirs sera mis à jour. S'il la donne aux farfadets chapardeurs son nombre de graines et celui de son joueur cible sont mis à jour.

Pour le cas d'un joueur virtuel, la manière dont il va utiliser sa carte ingrédient sera dictée par sa stratégie. Pour cela, la méthode jouerIngrédient() de la classe JoueurVirtuel est appelée. Cette méthode lance la méthode du même nom de la classe Stratégie (pour l'instance stratégie du joueur virtuel qui joue) et c'est cette dernière méthode qui utilisera la carte selon la stratégie fixée (mêmes manipulations sur les attributs du joueur que celles décrites pour le joueur réel)

Une fois que le joueur dont c'est au tour de jouer a utilisé sa carte ingrédient, une nouvelle boucle est ouverte. Elle parcoure tous les joueurs (y compris celui qui vient de jouer) et leur propose, s'ils ont une carte alliés, de la jouer.

De même que précédemment, il convient de distinguer les cas du joueur réel et du joueur virtuel : Selon que le joueur réel a une carte chiens de gardes ou taupes géantes, ce sont respectivement les méthodes utiliserChiensDeGarde() ou envoyerTaupesGéantes() de la classe joueur qui seront utilisées. Pour le cas d'un joueur artificiel qui a une carte alliée, le jeu ou non de sa carte alliée est déterminé par sa stratégie. De même que précédemment, la méthode jouerAllié() de la classe Joueur est appelée. Elle lance la méthode du même nom de la classe Stratégie qui joue ou non la carte alliée selon la stratégie fixée pour le joueur virtuel qui joue.

Une fois que le joueur dont c'est au tour de jouer a utilisé sa carte et que chacun des joueurs disposant d'une carte allié a eu la possibilité de la jouer, on passe au joueur suivant qui va utiliser une carte ingrédient (le compteur i est implémenté et on retourne à l'exécution de la première des boucles à avoir été ouverte).

Conclusion

Au cours de du développement, nous serons amenés à modifier quelques éléments de notre analyse présentée dans ce document.

Notamment, le cheminement de messages entre les classes, et plus particulièrement les messages relatifs aux actions de jouer une carte, nous semblent susceptibles d'être modifiés. De plus, pour des raisons de lisibilité du code, il est possible que nous ayons besoin de créer plus de méthodes afin de raccourcir la longueur du code des méthodes que nous avions prévues initialement. Enfin, le développement pourrait mettre en évidence des accesseurs superflus que nous éliminerions alors.

Toutefois il est probable que la structure des classes ne change que très peu. En effet, les patrons de conception que nous utilisons nous semblent peu enclins à être modifiés, et l'arborescence des classes paraît adaptée aux besoins de l'application.