UNIVERSITÉ D'AVIGNON
ET DES PAYS DE VAUCLUSE

C E N T R E
D'ENSEIGNEMENT
ET DE RECHERCHE
EN INFORMATIQUE

L3 Infortmatique
Spécialité Ingénierie Logiciel
UE u06-0082

>>> # Le jeu du 1 000 Go

ZAIM Hamza
AIT IHYA Hajar
BRICH Oussama

CERI - LIA
339 chemin des Meinajariès
BP 1228
84911 AVIGNON Cedex 9
France

Tél. +33 (0)4 90 84 35 00
Fax +33 (0)4 90 84 35 01

http://ceri.univ-avignon.fr

Encadrement
R. Dufour

Sommaire

1 Introduction

Ce rapport se propose de faire la synthèse de travail qui concerne la réalisation d'un logiciel informatique de Jeu Carte. le premier chapitre est consacré à l'étude et la présentation de projet. le deuxième chapitre aborde la conception et la modélisation de jeu en réalisant plusieurs diagrammes UML à savoir le diagramme de cas d'utilisation, le diagramme de classes, le diagramme d'états-transitions, et le diagramme de séquence. le troisième chapitre présente la réalisation de jeu et les outils utilisés.

Pour consulter le code latex de ce rapport veillez accéder le lien suivant :
https ://www.overleaf.com/2788478938twhcdzgxwvbp

2 Conception

2.1 Planning de projet

Afin de mener à bien notre projet, il nous a fallu déterminer les différentes tâches, les découper et les répartir équilibrement entre nous. Le diagramme de Gantt permet de visualiser dans le temps les divers tâches liées au projet et de représenter graphiquement leur avancement.
Nous avons commencé à établir un diagramme de Gantt prévisionnel, nous avons tenu également un diagramme de Gantt afin de voir ce sur quoi nous avion pris de retard ou les tâches dont nous avion surestimé la durée et la difficultés.
La répartition et l'organisation ont abouti à la réalisation de ce premier diagramme Gantt.

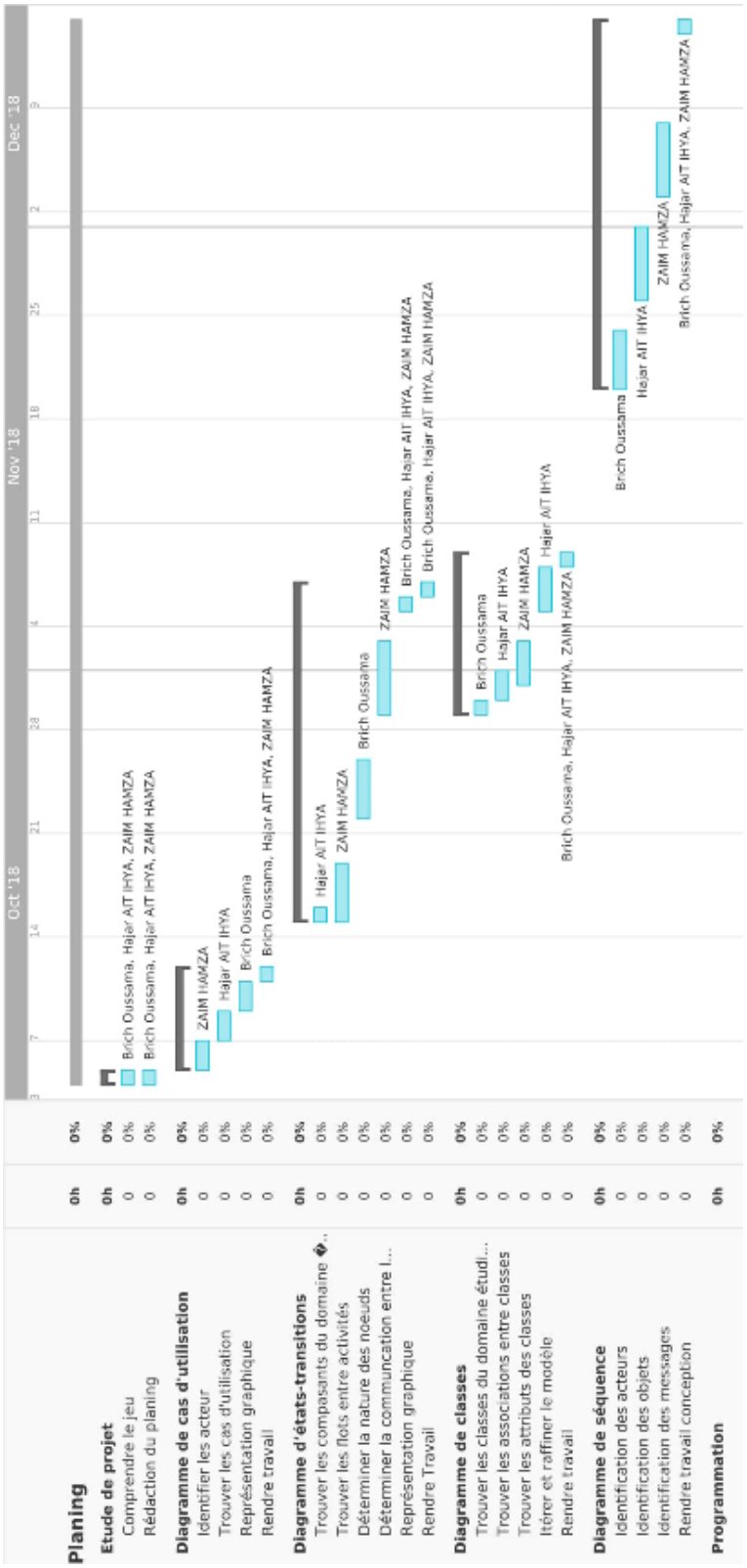

Figure 1. planning de notre projet

2.2 Diagramme de cas d'utilisation.

2.2.1 Identifier les acteurs

Dans le jeu, on trouve qu'un acteur qui est le joueur, et qui peut tre un joueur en local ou un joueur en réseau, alors c'est le seul acteur qui réagit au système.
Notre système est le moteur de jeu, est c'est l'ensemble des objets de la Partie, des joueurs, de la pioche, des cartes etc...

2.2.2 Trouver les cas d'utilisation

Les différents cas d'utilisation du jeu sont :

1. Piocher une carte : c'est un événement direct de l'utilisateur.
2. Configurer une partie :
   - Réseau en entrant que l'adresse IP de la partie à rejoindre.
   - Locale en choisissant le nombre de joueurs et en entrant le niveau de difficultés des bots.
3. Jouer une carte : c'est le cas d'utilisation primaire du Jeu qui peut être pour :
   - Attaquer : empêcher les autres joueurs de transférer des données.
   - Défendre : éviter les attaques .
   - Transférer les données : Transférer des données selon la valeur de la carte (Nombre de Gb).
   - Immunite

2.2.3 Représentation de diagramme

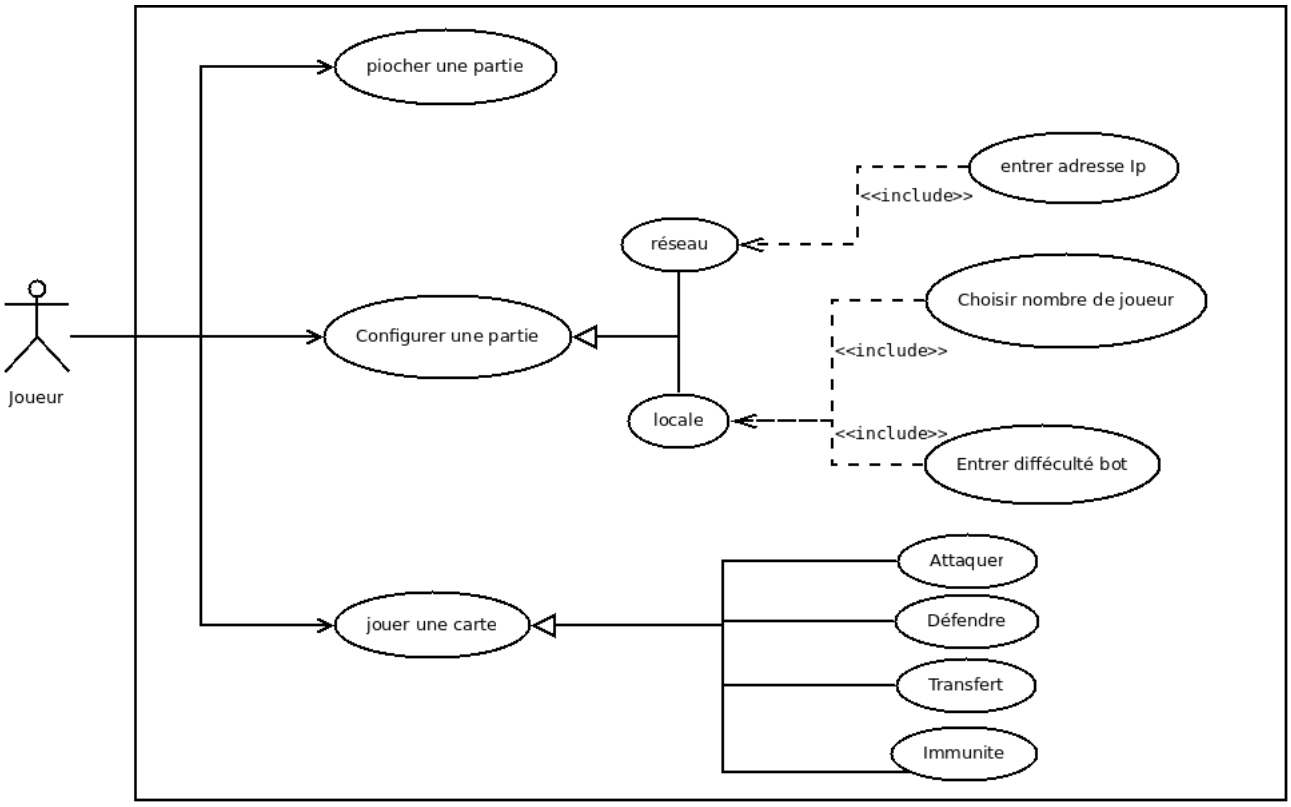

Figure 2. Diagramme de cas d'utilisation

2.3 Diagramme d'activité

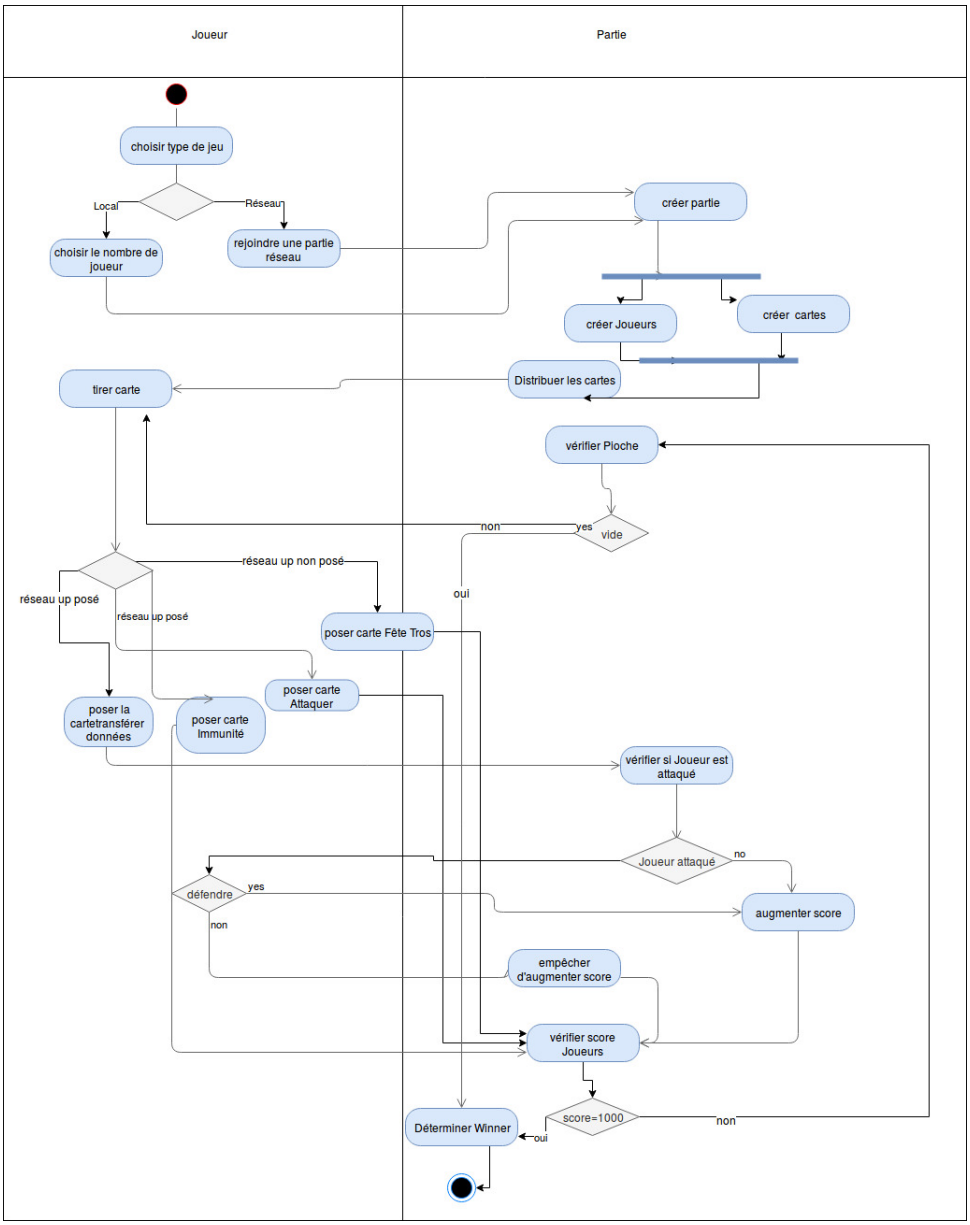

Figure 3. Diagramme d'activité

Dans le diagramme d'activité ci-dessus, il est possible de voir qu'on choisi tout d'abord le mode de jeu qui peu être en local ou en réseau,si on est on le joueur choisi la partie en local alors il doit enter le nombre de joueur à condition de ne pas dépasser six joueurs et après la partie se charge en créant les joueurs et les cartes. la même chose lorsqu'il choisi de jouer en réseau, mais il doit configurer la partie et saisir son adresse ip.
Après, la partie distribue les cartes au joueurs pour que le joueur pourra commencer à à jouer. Tout d'abord il tire une carte, selon une condition si il a une carte Réseau Up ou pas il pose une carte qui peut être : carte de transfert des données, carte immunité, carte d'attaque ou carte de défendre. en puis la partie teste le nombre des GB qu'il a le joueur : s'il arrive ou il dépasse 1000 la partie détermine le gagnant et la partie sera terminé,sinon la partie vérifie la pioche, si elle est vide on détermine le gagnant et on sorte dans la partie, si elle n'est pas vide le joueur dans son tour recommencer à tirer une carte.

2.4 Diagramme de classes

Le diagramme de classe exprime la structure statique du système en termes de classes et de relations entre ces classes.
Le diagramme de classe permet de représenter l'ensemble des informations finalisées qui sont gérées par le système. Ces informations sont structurées, c'est-à-dire que elles ont regroupées dans des classes. Le diagramme met en évidence d'éventuelles relations entre ces classes.

2.4.1 les classes du domaine étudié

Pour réaliser notre diagramme de classe, on a étudié la préparation et le déroulement du jeu : Tout d'abord avant de commencer le jeu, il faut mélanger et distribuer les cartes pour avoir 1000 Go pour chaque joueur, alors on a créé la classe Pioche.
La classe Pioche est définie avec un ID de partie, liste et nombre de carte disponible, et elle a la méthode mélanger() qui mélange d'une manière aléatoire les cartes, et la fonction distribuer() pour les distribuer.
Pour le déroulement de jeu on a créé la classe abstraite Partie de laquelle on va hériter les classes PartieLocale et PartieRéseau. les deux classe vont hériter le idPartie le nbJoueur. Pour rejoindre ou quitter une partie quelconque, on a créé la fonction RejoindrePartie(Joueur) et quitterPartie(Joueur) qui prennent un argument Joueur.
La classe PartieRéseau contient de plus une adresse Ip sous forme d'une chaîne de caractères et une fonction pour la récupérer.
Pour définir les joueurs, on a créé une classe abstraite joueur de laquelle on va hériter deux autre classe Humain et Bot. Les deux classes vont hériter un identifiant Joueur idJoueur, le nombre des géga transférés autant qu'un score nbreGbTransfere, et la main de chaque joueur autant qu'une liste de carte Carte.
Également, les deux classes vont hériter les fonctions suivantes :

- getIdJoeur() : Pour récupérer les identifiants des joueurs
- poserCarte() : Pour jouer une carte et la supprimer de la main de joueur, la liste Carte.
- piocherCarte() : Pour ajouter une carte à la main du joueur, en ajoutant la carte à la liste de carte Carte et la supprimer de la pioche.
- calculerGbTransfere pour rendre le nombre de géga transféré.

La classe Bot contient un attribut de difficulté et une fonction getDeficulte() pour l'avoir.
Pour la création des objets de cartes, on a créé une carte abstraite Carte qui contient :

- un attribut de valeur et qui va nous permettre de faire la différence entre les différents objets de carte
- une fonction getValue() pour récupérer la valeur de chaque carte pour la tester après en définissant les effets de chaque carte
- une fonction setValue() pour modifier la valeur de chaque carte si on voulais ajouter un autre type de carte après et pour garantir l'aspect évolutif de notre jeu.

C'est de la classe abstraite Carte qu'on a hérité les classes suivante :

- Reseau : cette classe permet de créer l'objet carte réseau et qui est définie par une Valeur "Value" et une fonction getValue() pour la récupérer.
- Cartetransfert : cette classe nous permet de créer la carte de transfert réseau et qui a les même attributs et fonction de la classe parent.
- CarteDefendre : cette classe nous permet de créer l'objet de carte de défendre dans le jeu.
- CarteAttack : cette classe nous permet de créer l'objet de carte d'attaque dans le jeu
- CarteImmunite : cette classe nous permet de créer l'objet de carte immunité dans le jeu

2.4.2 Relation entre les classes

Les différentes relations dans le diagramme de classe étaient des relations :

1. Héritage : on a choisi de créer des classes abstraites et les hériter pour définir les niveau hiérarchique des objets instanciés et c'est le cas des classes :
   - Carte en héritant les classes de carte pour créer les objets carte d'attack, de défendre, de Transfer et d'immunité.
   - Joueur en héritant les classes BOT et Humain pour créer les objets de joueurs humain et Bot.
   - Partie en héritant les classes PartieLocale et PartieRéseau pour créer les objets de partie locale et réseaux.
2. Associations : on a associé :
   - la classe Joueur avec la classe Carte sous le nom Avoir et on a définit un cardinale de 1 pour Joueur et 1..6 pour carte , alors qu'un joueur peut avoir d'une jusqu'à 6 cartes.
   - la classe Joueur avec la classe Partie sous le nom Jouer et on a définit un cardinale de 2..6 pour joueur et 1 pour partie ; alors 2 jusqu'à 6 joueur peuvent jouer dans une seule Partie.
   - la classe Partie avec la classe Pioche sous le nome utiliser et un cardinale de 1 pour Partie et 1 pour Pioche, alors dans une partie on peut avoir une seule pioche.
   - la classe Carte avec la classe pioche sous le nom avoir et un cardinale de 1..106 pour carte et 1 pour Pioche, alors une pioche peut avoir d'une jusqu'à 106 carte .

2.4.3 Représentation du diagramme

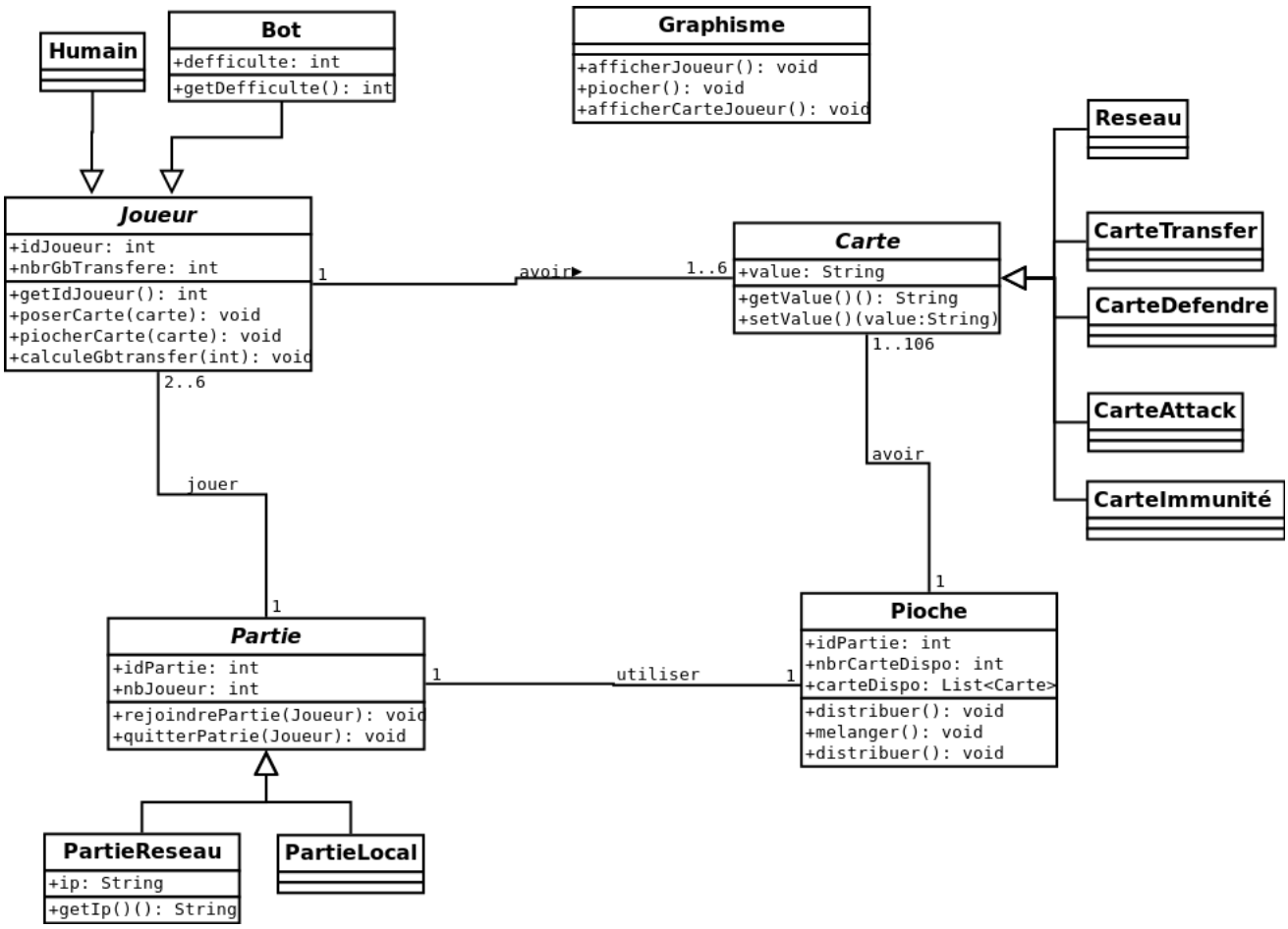

Figure 4. Diagramme de classe

2.5 Diagramme de séquence

Pour représenter les collaborations entre objets selon un point de vue temporel on a proposé le diagramme de séquence ci-des, ce diagramme contient 4 objets (Joueur,Partie,Pioche et Carte)

les messages entre les objets qu'on a identifiés :

- le message synchrone **configurer** (une seule fois pour chaque partie).
- le message synchrone **MelangerCarte** partie pour mélanger les carte en manière aléatoire dans la pioche.
- le message synchrone **Distribuer** 6 carte par joueur (une seule fois dans le début de la partie).

  dans une boucle (boucle par indice de joueur)
- le message synchrone **DéterminerJoueur** pour déterminer le joueur à chaque tour.

  dans une condition (avoir le carte de transférer) dans une condition (réseau up)
- le message synchrone **Transférer** données.

  dans une condition (avoir le carte d'attaque)
- le message synchrone **AttaquerJoueur**

  dans une condition (avoir le carte d'immunité)
- le message synchrone **Immunité** : après l'immunité on a le message TirerCarte qui est relié à ce dernier.

  dans une condition (avoir le carte de défendre)
- le message synchrone **DefendreContrAttaque**.

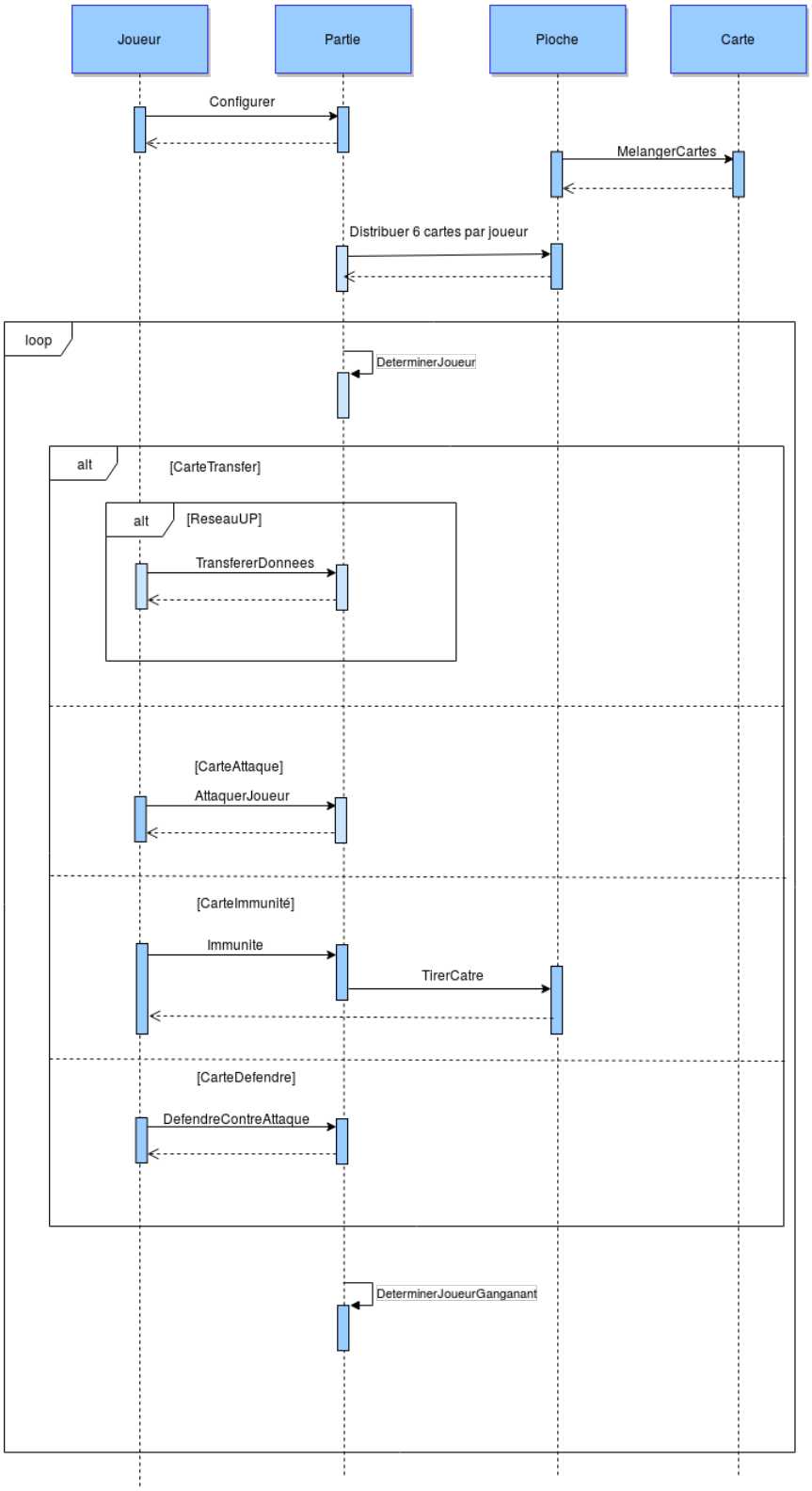

Figure 5. Diagramme de séquence

3 Implémentation

Dans la partie implémentation on a essayé d'appliquer la modélisation dont on a déjà donné, ceci est représenté par la programmation du jeu en utilisant la programmation orienté objet en Java sur l'environnement de développement Eclipse. Aussi on a trouvé que le faite D'avoir des diagrammes UML logiques donne des idées claires et exacts pour les étapes et le déroulement de développement.

malheureusement nous n'avons pas fini cette partie pas à cause de sa difficulté mais à cause du manque de temps.

3.1 Fonctionnalités Réalisées

- configuration d'un jeu (nombre de joueurs)
- création d'une pioche avec des cartes de position aléatoire
- l'ajout des joueurs dans le jeu
- distribution des cartes aux joueurs dans le début du jeu
- tirer une carte a chaque tour
- poser une carte
- mettre le réseau UP au joueur
- passer le jeu au joueur suivant

3.2 Fonctionnalité manquantes et leur implémentation

- l'ajoute les effets pour les cartes.
  - Carte attaque : pour implémenter l'effet des carte d'attack on a voulu créer une fonction pour démunier le nombre de gb transférées du joueur attaqué.
  - Carte défendre : L'effet de la carte défebdre était défini dans une fonction qui arrete l'effet de la carte attaque quand elle est posé dans un round de joueur en vérifiant les conditions.
  - Carte Immunité : On a pensé à créer une fonction qui arrete l'effect d'être attaqué pendant tout le round du joueur.
- déterminer le joueur gagnant : on a pensé de créer une fonction determinerGagnat (elle a comme type de retour un Objet Joueur) dans la classe Partie qui permet :

  - e tester à chaque transfert de donnée si le joueur a ou il a dépasser 1000 GB, si c'est le cas elle le retourne comme gagnant.
  - de tester le joueur qui a le nombre élevé des GB si la pioche est vide.

vous trouverez ci-joint les classes qu'on a implémente (l'ajoute des commentaires sur la dernière version qu'on a déposé)

4 conclusion

Ce projet a été sous plusieurs aspects riches d'enseignements aussi ce dernier est une opportunité pour améliorer les connaissances en modélisation UML et être capable de donner une représentation simplifiée d'un problème informatique.
La réalisation d'un tel projet, nous a permis d'apprendre et de toucher du doigt une partie de divers aspects du métier de développeur.

4.1 Mise à jour du Planning

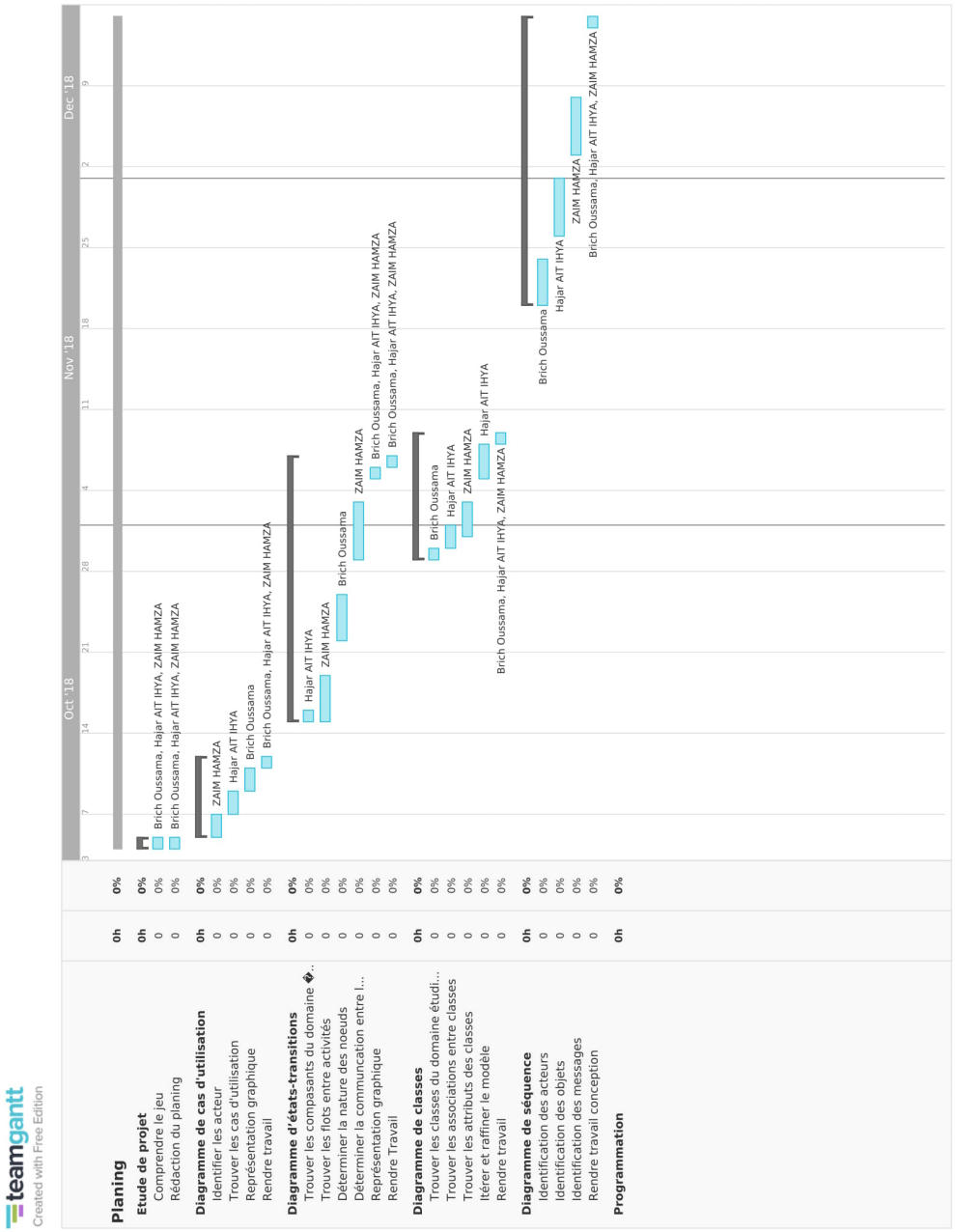

Figure 6. Mise à jour du planning du projet

4.2 Bilan d'activité

1. ZAIM Hamza

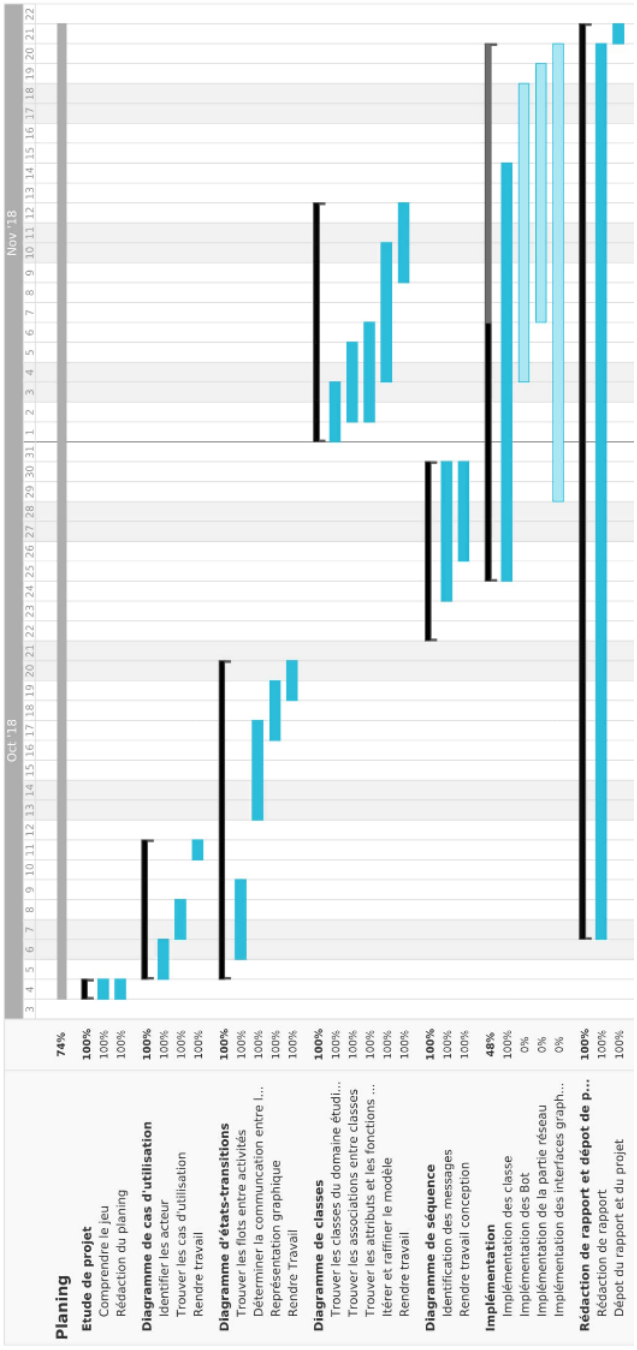

Figure 7. Tâches réalisées par ZAIM Hamza durant le projet

2. AIT IHYA Hajar

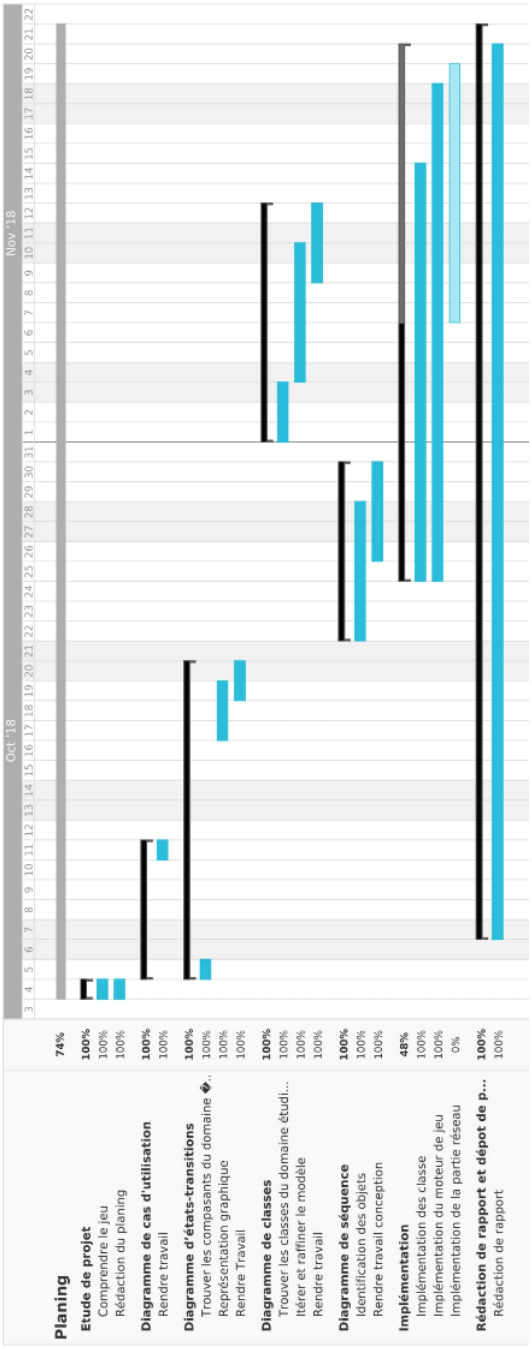

Figure 8. Tâches réalisées par Ait Ihya Hajar durant le projet

3. BRICH Oussama

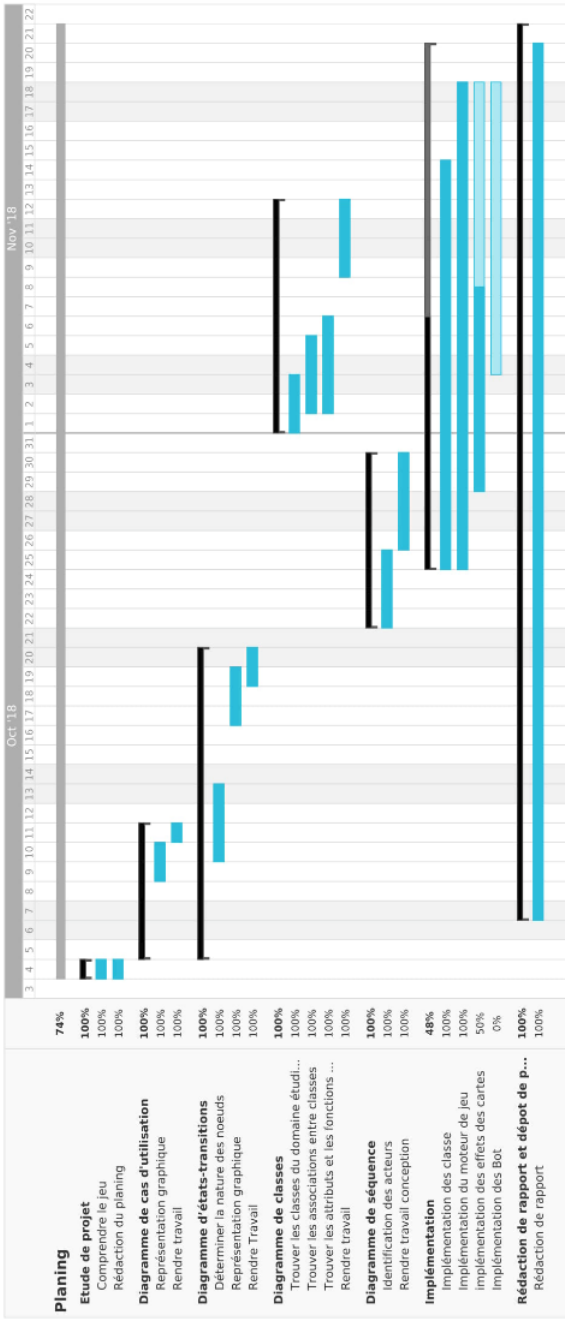

Figure 9. Tâches réalisées par Brich Oussama durant le projet

4.3 Problème rencontrés

Un des problème qu'on a rencontré c'était le problème du temps, et qui était insuffisant pour implémenter le jeu dans une semaine, également on a d'autres rendus de projet et de TD qui ont également entravé l'avancée du projet.