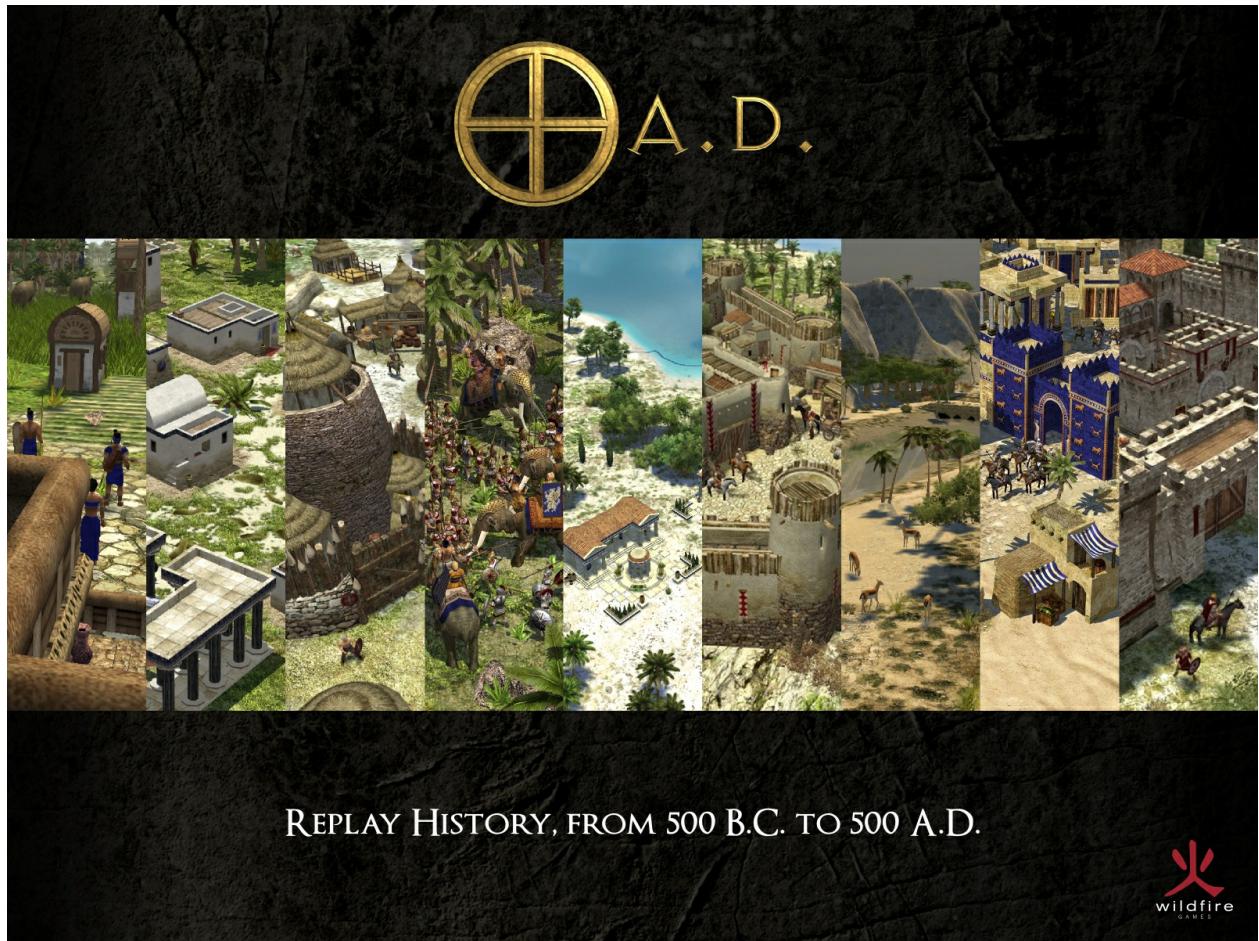


Analyse logicielle du jeu libre 0 A.D.



REPLAY HISTORY, FROM 500 B.C. TO 500 A.D.



Duraj Bastien et Carreteros Laetitia
UQAC, 01 novembre 2018

Plan

1. Introduction
2. Description du logiciel
3. Analyse architecturale
 - i. Les différents acteurs et scénarios possibles
 - ii. Development View
 - iii. Logical View
 - iv. Process View
 - v. Conclusion sur l'architecture
4. Analyse de la qualité du code
5. Design Patterns
6. Problème de conception
7. Conclusion

Introduction

Nous allons ici analyser un jeu vidéo libre : 0 A.D. . Il s'agit d'un jeu de stratégie en temps réel, en opposition au jeu de stratégie au tour par tour. Celui-ci propose de jouer avec une dizaine de civilisations ayant vécu de l'an 500 av. JC jusqu'à l'an 0. Le jeu peut être joué aussi bien tout seul qu'à plusieurs.

Né du jeu de stratégie en temps réel, Age of empires, lequel a été développé en partie par Relic Entertainment et édité par Microsoft Studios, le jeu était au départ un mod, c'est-à-dire qu'il utilisait les technologies et les ressources d'un jeu existant. Dans ce cas précis il s'agissait d'un mod du second volet de la saga: Age of empires II: The Age of Kings. Le but premier était de recréer Age of empires avec les technologies du second jeu mais le projet a fini par évoluer et devenir totalement autonome.

0 A.D. a, dans un premier temps, été en développement fermé avec la société Wildfire Games en 2002. Mais au cours de l'année 2009 l'entreprise décide d'ouvrir le développement et le code, permettant ainsi à qui le souhaite de participer.

Pour commencer nous allons expliquer ce qui nous a intéressé dans ce projet. Tout d'abord nous avons choisi un jeu vidéo car cela nous semblait intéressant de voir leur construction et leur architecture mais également leur évolution au cours du développement. Les jeux vidéos sont des logiciels de divertissement, ils ont donc des particularités qui leur sont propres notamment celle de faire collaborer des artistes et des designer avec des programmeurs, ou encore de devoir, sur un projet à long terme, changer de moteur graphique pour un meilleur rendu par exemple. Nous sommes donc intéressés pour voir comment ces contraintes sont gérées dans un développement ouvert.

Ensuite notre choix s'est porté sur 0 A.D. car il est encore assez actif et ceci depuis plusieurs années : la dernière mise à jour date du 17 mai 2018 ce qui est plutôt récent pour un tel projet. La documentation du projet est également très complète et aborde tous les sujets: comment participer, documentation technique, développer un patch, comment apporter une traduction, Le code et les ressources ainsi que l'exécutable sont facile d'accès et le forum très actif.

De plus depuis l'ouverture du projet celui-ci a beaucoup progressé et est devenue très complet : partie solo, partie multijoueur, didacticiel, éditeur de scénarios, support pour les mods, ect... , cherchant à atteindre un niveau de qualité proche des jeux commerciaux.

Il serait donc intéressant d'analyser ce logiciel et ceci sous plusieurs angles en commençant par une analyse architecturale en prenant en compte 4 vues : Logical view , Process view , Development view et Deployment view . Une fois cette analyse effectuée nous analyserons la qualité du code grâce à plusieurs métriques puis nous finirons avec l'analyse des Design Patterns présent dans le projet ainsi que leurs avantages et inconvénients.

Description du logiciel

Avant de nous plonger dans l'analyse formelle du logiciel nous allons décrire le jeu dans son fonctionnement et ses principales interfaces. Il s'agit d'un jeu de stratégie, le but est de battre l'adversaire ou les adversaires suivant un objectif précis. Pour cela on choisit une civilisation que l'on va diriger sur plusieurs aspects. Le jeu se présente avec une Caméra en vue de dessus assurant une vue globale sur la partie et permettant de mieux gérer ses unités et ses bâtiments. La figure 5 permet d'avoir un aperçu de ce concept. Passons maintenant à la présentation du jeu tel que les développeurs l'on conçut.

Au lancement du logiciel nous nous retrouvons donc avec une interface nous permettant soit d'apprendre à jouer, de jouer une partie en solo, de jouer en multijoueur, de régler les paramètres ainsi que d'autres fonctionnalités pouvant aider à l'amélioration du jeu. L'image ci-dessous permet de voir comment ce menu est présenté.



fig. 1 Menu principale du jeu

Si nous décidons de jouer une partie, ce qui est la fonctionnalité centrale de tout jeu vidéo, nous pouvons , comme montré à la figure 2 , 3 et 4, choisir la faction que nous voulons jouer, le type de carte (qui définit le nombre de joueurs), la difficulté des IA , et la possibilité de créer des équipes. Pour le choix du type de cartes il y a à disposition différents filtres : nouvelles cartes, cartes navales, carte de démonstration , carte à déclencheurs et toutes les cartes, ceci permet de ne montrer que les cartes qui nous intéressent.

Une fois la carte choisie nous pouvons décider si nous voulons avoir la carte révélée ou non ainsi que la présence de trésors en jeu.

Nom du joueur	Couleur	Position du joueur	Civilisation	Équipe
Joueur 1	▼ murph	▼	Aléatoire	▼ 1 ▼
Joueur 2	▼ IA : Petra Bot	▼	Aléatoire	▼ Aucune ▼
Joueur 3	▼ IA : Petra Bot	▼	Aléatoire	▼ Aucune ▼
Joueur 4	▼ IA : Petra Bot	▼	Aléatoire	▼ 1 ▼

Configuration

Bien que relativement faible pour un joueur expérimenté, le niveau de difficulté par défaut de l'IA représente un bon challenge pour permettre aux nouveaux joueurs de maîtriser les mécanismes de base du jeu. Pour les débutants, il est recommandé de commencer à jouer contre un niveau de difficulté inférieur (Sable à sable ou Très facile). Pour changer le niveau de difficulté de l'IA, cliquez sur l'icône d'engrenage à côté de chaque joueur IA sur le panneau de sélection des joueurs.

Montrer ce message ultérieurement

Paramètres joueurs

Plaines de Thessalie (4)

Type de carte: Escarmouche
Filtre de carte: Toutes les cartes
Choisir la carte: Plaines de Thessalie (4)
Désactiver les trésors:
Carte explorée:
Carte révélée:

Carte
Joueur
Type de jeu

Conquête Vaincre les adversaires en tuant toutes leurs unités et en détruisant toutes leurs structures.
Diplomatie Les joueurs peuvent établir des alliances et déclarer la guerre à leurs alliés.
Victoire alliée Si un joueur gagne, ses alliés gagnent aussi. S'il reste un groupe d'alliés, ceux-ci gagnent.
Cessez le feu Défendre son territoire.
Nom de la carte: Plaines de Thessalie (4)
Description de la carte La bouleuse plaine de Thessalie est traversée par des ruisseaux étroits, faciles à traverser. De larges espaces ouverts permettent l'expansion massive, et chaque joueur commence la partie à l'abri en haut d'une large acropole.
Type de carte Escarmouche
Filtre de carte Toutes les cartes
Centre-villes Les joueurs commencent avec un Centre-ville
Ressources de départ Faibles (300)
Limite de population 300

Retour Démarrer !

fig. 2 Configuration nouvelle partie / Paramètres carte

Une autre possibilité est de choisir la capacité limite de population dans une civilisation ainsi que le nombre de ressources de

départ pour personnaliser sa partie et ne pas vivre toujours la même expérience de jeu.

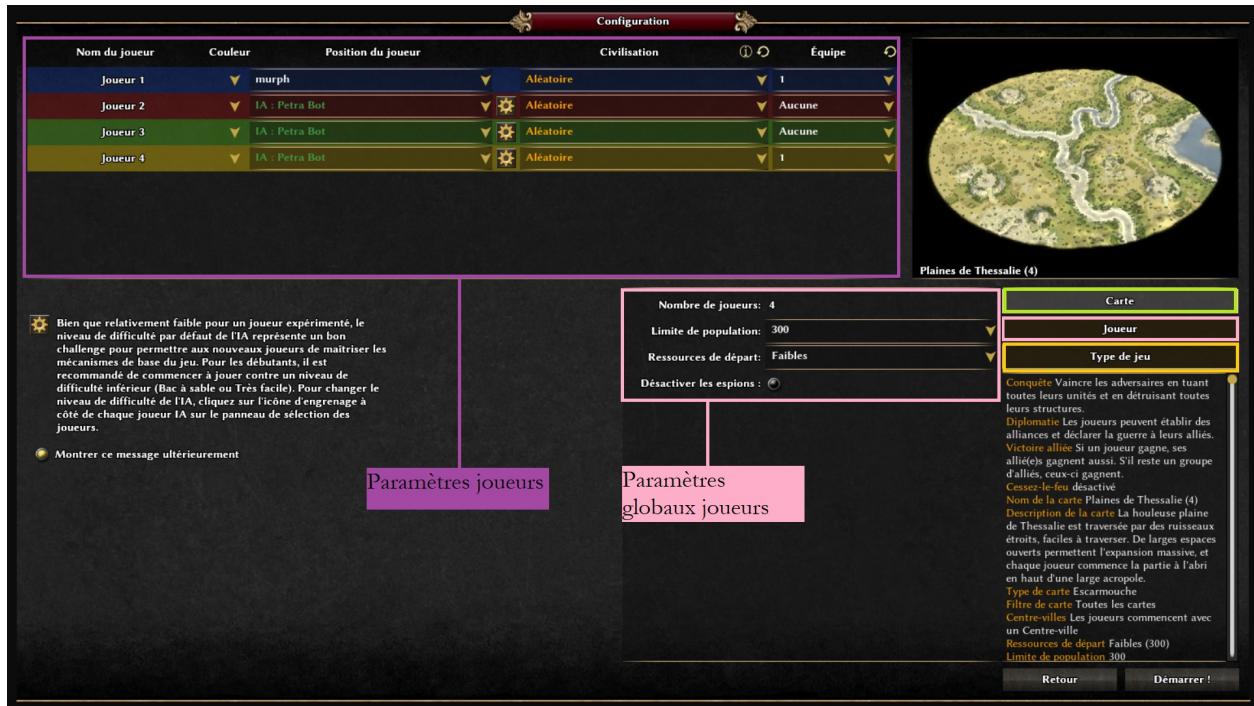


fig. 3 Configuration nouvelle partie / Paramètres global joueurs

La manière de terminer / gagner une partie peut également être déterminée par l'onglet "type de jeu". Nous pouvons aussi ajouter plusieurs conditions simultanées pour gagner ce qui complexifie grandement le jeu.



fig. 4 Configuration nouvelle partie / Paramètres fin de partie

La configuration d'une partie est donc très complète et au même niveau que des triple A tels que Starcraft ou encore Age of Empires.

Une fois la partie démarrée nous nous retrouvons avec notre civilisation et comme point de départ un centre ville entouré de quatre ouvrières, deux soldats de corps à corps (infanterie), un cavalier, et deux soldats de combat à distance (archerie). Ces unités de départ peuvent récupérer différentes ressources et chacune des unités a des bonus différents dans la récolte d'une

ressource ce qui permet de les différencier lorsque l'on doit récolter. Il existe plusieurs ressources récoltables qui sont la nourriture , les pierres , le métal et le bois. Grâce à cela les unités peuvent construire des bâtiments pour faire évoluer et développer notre civilisation. L'évolution se fait à travers 3 phases. La première de base est celle des villages , la seconde la phase des villes , et la dernière celle des cités. Le fait de changer de phase permet de débloquer de nouveaux bâtiments avec de nouvelles fonctionnalités. Mais pour débloquer ses phases avec les avantages qu'elles possèdent il faut remplir des conditions , par exemple un nombre de population , la présence de certains bâtiments de l'âge actuel, forçant le joueur à devoir se développer un peu avant de voir les meilleures unités débloquées.

Pour combattre les autres civilisations nous utilisons les unités à disposition qu'il faut entraîner dans des bâtiments spéciaux comme les casernes. Le jeu propose d'ailleurs une grande diversité suivant la civilisations choisi lors de la configuration du jeu. Le but est donc, dans le cas d'une victoire militaire, de détruire les bâtiments et unités adverses grâce à nos propres unités. Il existe plusieurs catégories : l'infanterie, l'archerie, la cavalerie et les armes de sièges. Chacune ayant un rôle précis et devant être utilisée correctement pour s'assurer la victoire.

Nous pouvons également mettre des unités dans certains bâtiments soit pour les protéger soit pour qu'ils attaquent automatiquement les ennemis. Mais le jeu ne se limite pas à la guerre on peut également faire du commerce avec des joueurs neutres ou alliés, ou utiliser la diplomatie pour gagner contre un joueur puissant.

Le jeu intègre également une notion de territoire qui limite la construction des bâtiment à notre territoire uniquement (sauf les postes avancées qui peuvent être en territoire neutres) . Pour étendre notre territoire nous devons bâtir nos bâtiments proches de nos frontières , cela va participer à son expansion au fil du temps et ainsi notre domination sur la carte. Il existe une façon plus rapide de s'étendre, la construction de nouveaux centre ville sur la carte. Ceux-ci peuvent se construire sur les territoires neutres. Cette mécanique n'est pas à prendre à la légère puisqu'elle permet de bloquer l'avancé d'autres civilisations en récupérant des ressources.



fig. 5 Interface de jeu lors d'une partie

Ce premier pas dans les nombreuses fonctionnalités du jeu permet dans un premier temps de comprendre le but recherché par les développeurs : créer un jeu de stratégie open source le plus complet possible avec un grand niveau de qualité. Puis dans un second temps de mieux comprendre l'analyse des fonctionnalités présente dans la prochaine partie.

Analyse architecturale

Les différents acteurs et scénarios possibles

Pour comprendre ce logiciel nous avons tout d'abord analysé l'architecture de celui-ci en commençant par identifier les différents acteurs qui peuvent intervenir dans le déroulement du jeu. Pour représenter ces acteurs et les fonctionnalités auxquels ils sont rattachés nous avons décidé de modéliser nos résultats avec, dans un premier temps, un diagramme regroupant les différentes parties du logiciel et les acteurs, puis dans un second temps un diagramme des cas d'utilisation pour la partie la plus intéressante et la plus complète : le déroulement d'une partie.

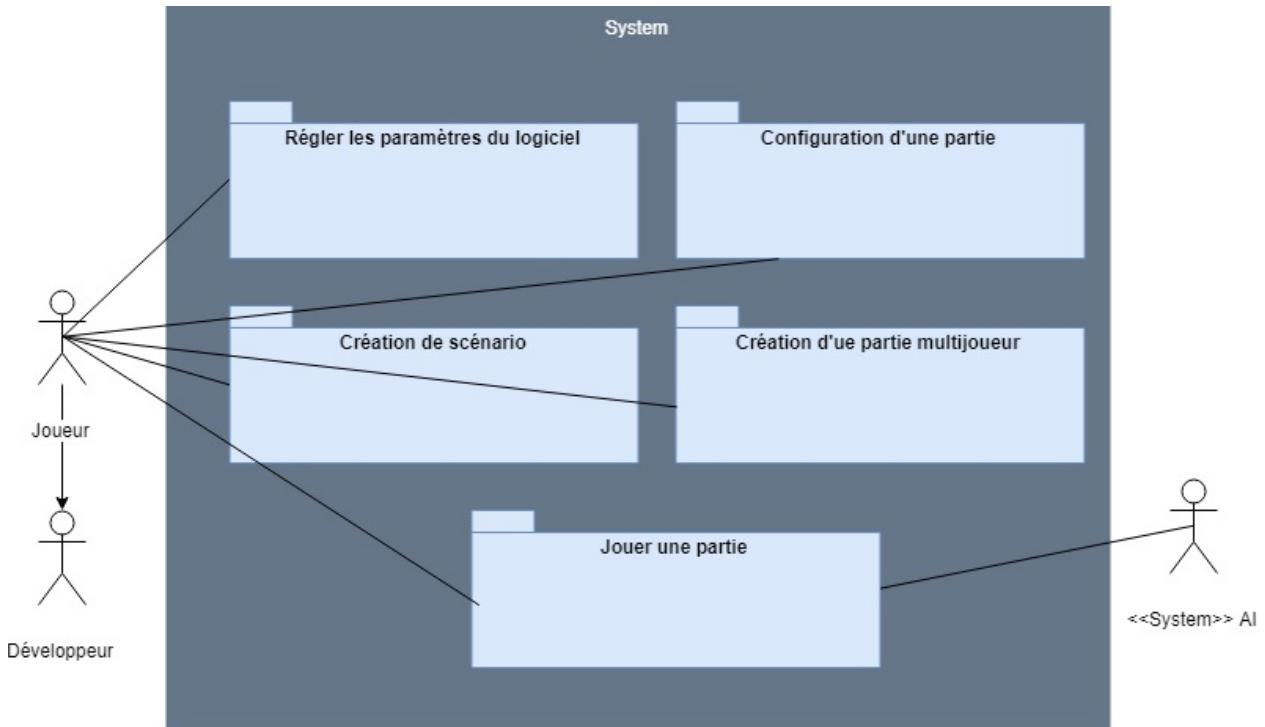


fig. 6 Diagramme représentant les grandes fonctionnalités du jeu

Nous avons donc divisé en 5 parties les différentes utilisations possibles du jeu avec 2 acteurs humains, les joueurs et les développeurs, et un acteur système représentant l'intelligence artificiel du jeu: Petra. Alors que le développeur peut faire tout ce que le joueur peut, en plus de la possibilité d'utiliser une console, l'IA ne peut que joueur une partie.

Nous avons ensuite détaillé chaque partie à l'aide d'un diagramme de cas d'utilisation pour nous permettre de mieux comprendre les objectifs et les besoins d'un tel logiciel.

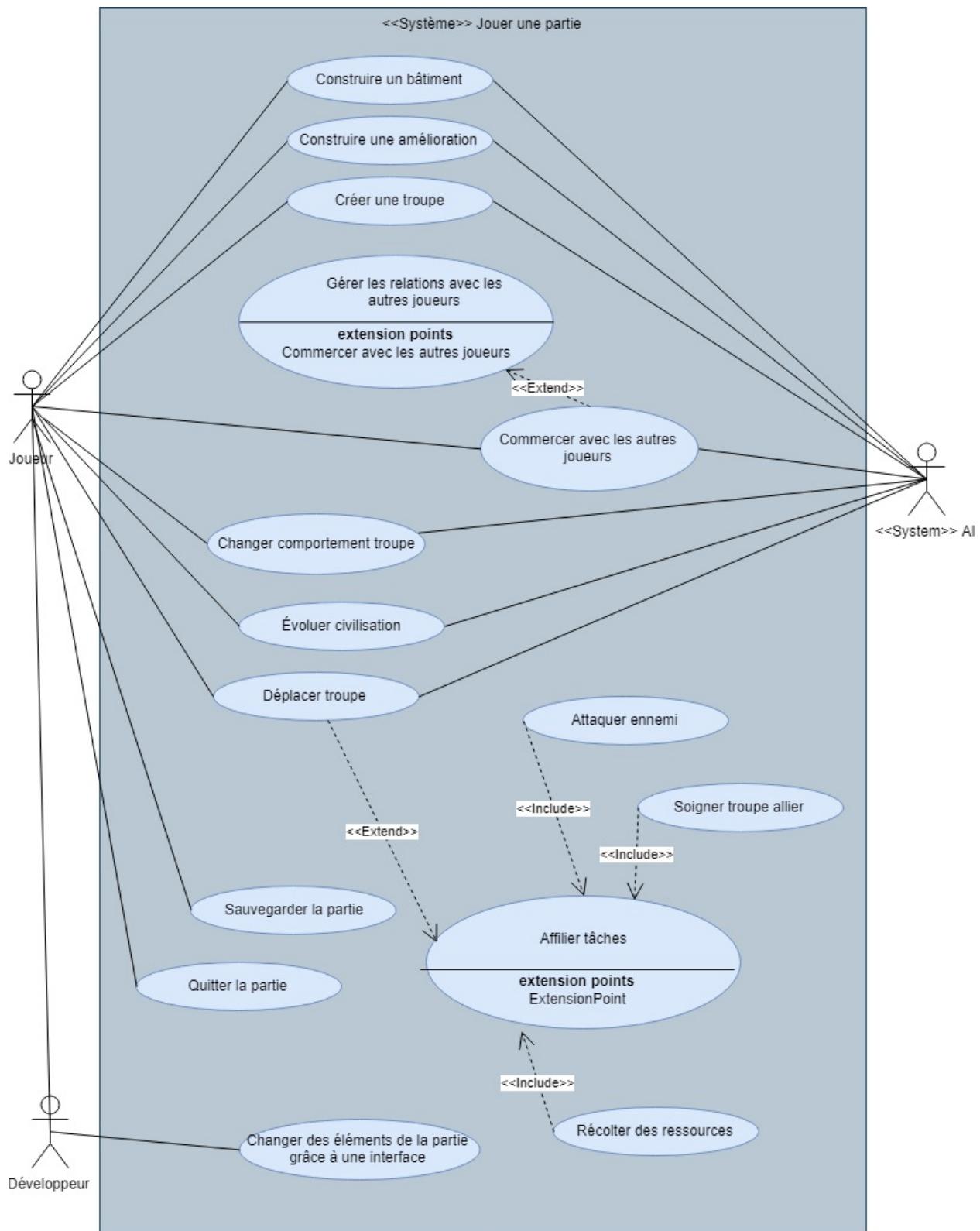


fig. 7 Diagramme des cas d'utilisations pour la fonctionnalité "Jouer partie"

Le diagramme qui nous intéresse le plus ici est celui qui concerne le déroulement d'une partie. Il est le centre du jeu et justifie l'existence même du moteur graphique, c'est aussi l'une des parties les plus complètes où chaque acteur à un rôle à jouer. Ce diagramme montre bien les nombreuses actions que le joueur ou l'IA peuvent faire pour remporter le jeu, qu'ils s'agissent des relations diplomatiques avec d'autres joueurs ou de la création de troupe. Il met également en avant un point important de tout jeu vidéo d'envergure, la possibilité de faire apparaître une console qui permet d'appeler directement des fonctions javascript pour par exemple tester de nouveaux éléments non implémentés.

Ce diagramme permet également de mieux appréhender les fonctionnalités mises en avant dans le jeu et de voir comment ces fonctionnalités ont été intégré dans l'architecture du projet grâce à la partie suivante portant sur les packages.

Diagramme des packages (Development View)

Notre analyse des packages a pour but de mieux comprendre comment le projet a été construit et comme premier point nous avons remarqué que les packages du projet représentent souvent la fonctionnalité "technique" qu'ils traitent comme le rendu, les graphismes, ou encore l'intégration des scripts dans le moteur. Ce qui donne le diagramme suivant:

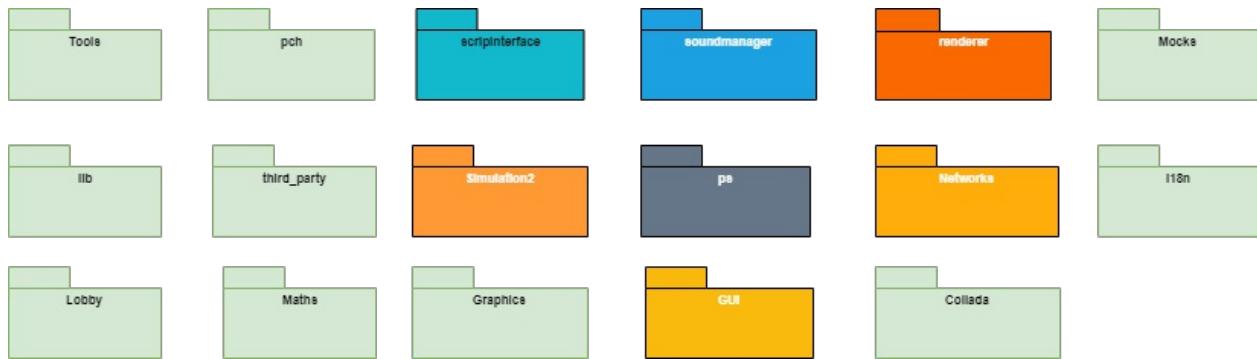


fig. 8 Diagramme des packages simplifié avec les dépendances principales

Le principal problème de cette architecture c'est qu'elle occassione beaucoup de dépendances entre les packages ce qui peut compliquer énormément le refactoring. Il aurait pu être intéressant de diviser le projet en services et fonctionnalités orientées utilisateurs à la place comme on peut le voir avec le package "Atlas" qui est l'éditeur de scénarios du jeu. Le diagramme des dépendances qui suit montre bien le problème. On remarque rapidement que les packages s'appellent beaucoup entre eux ce qui rend le tout assez illisible.

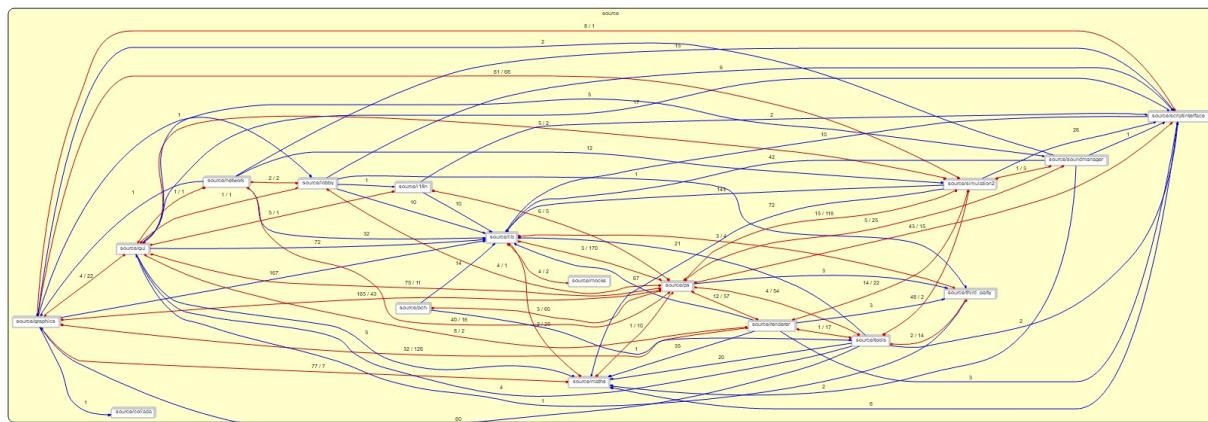


fig. 9 Diagramme des dépendances entre packages

Plus précisément le diagramme des packages est comme ceci:

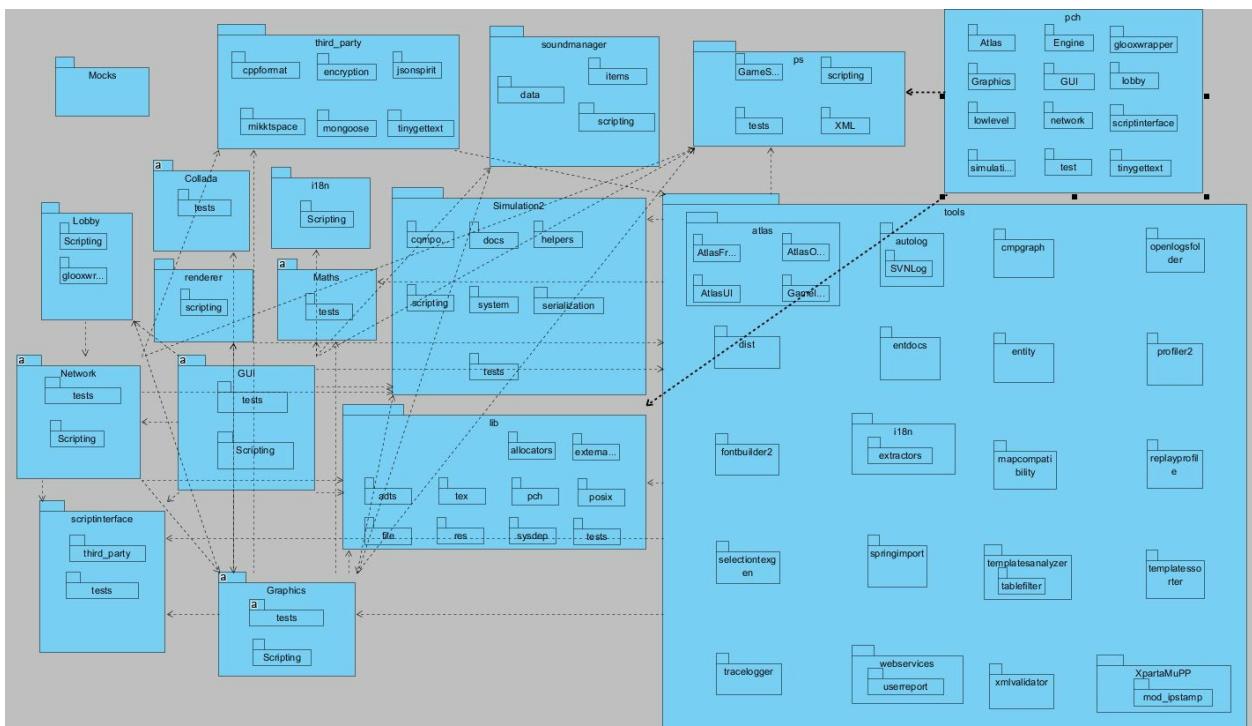


fig. 10 Diagramme des packages avec leurs dépendances

Celui-ci permet de remarquer que les différents packages ne sont pas tous bien organisés en dossiers plus petit, ce qui peut poser problème pour la compréhension du projet, par exemple dans le package "graphics" qui est très petit, on trouve 118 fichiers, 40 classes et 25 735 lignes de code ce qui rends l'exploration laborieuse. De plus le diagramme de package ne permet pas sous cette forme de reconnaître les éléments les plus importants de l'architecture comme la gestion des graphismes dans "graphics". En effet ce package possède un diagramme de classe très imposant comme montré ci-dessous à la figure 11.

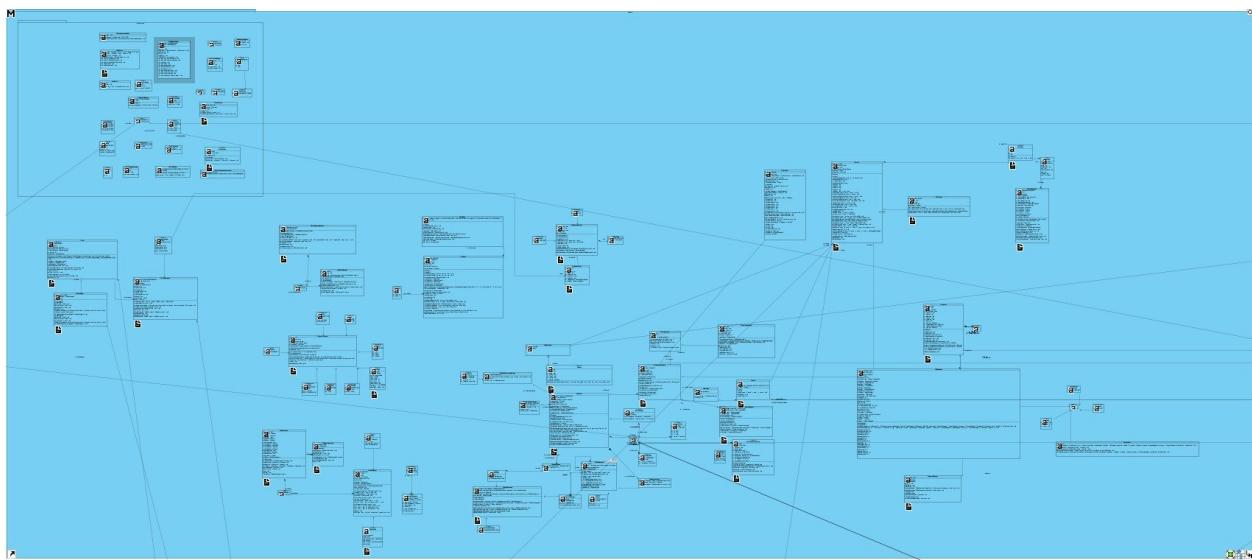


fig. 11 Diagramme des classes du package "graphics"

Mais celui-ci n'est pas divisé en sous-partie comme dans le package tools (voir figure 12).

!Diagramme de packages de tools et graphics]
<https://github.com/Bhastyen/AnalyseGenieLogiciel/blob/master/Images/PackageToolsDiagram.png?raw=true> "Diagramme de packages du package Tools et du package Graphics)

fig. 12 Diagramme de packages du package "Tools" et du package "Graphics"

Logical View

Le diagramme des classes étant assez grands et complexes, regroupant 655 classes dans 17 packages. Il comprend également 298 228 lignes de code, en voici un aperçu non exhaustif:

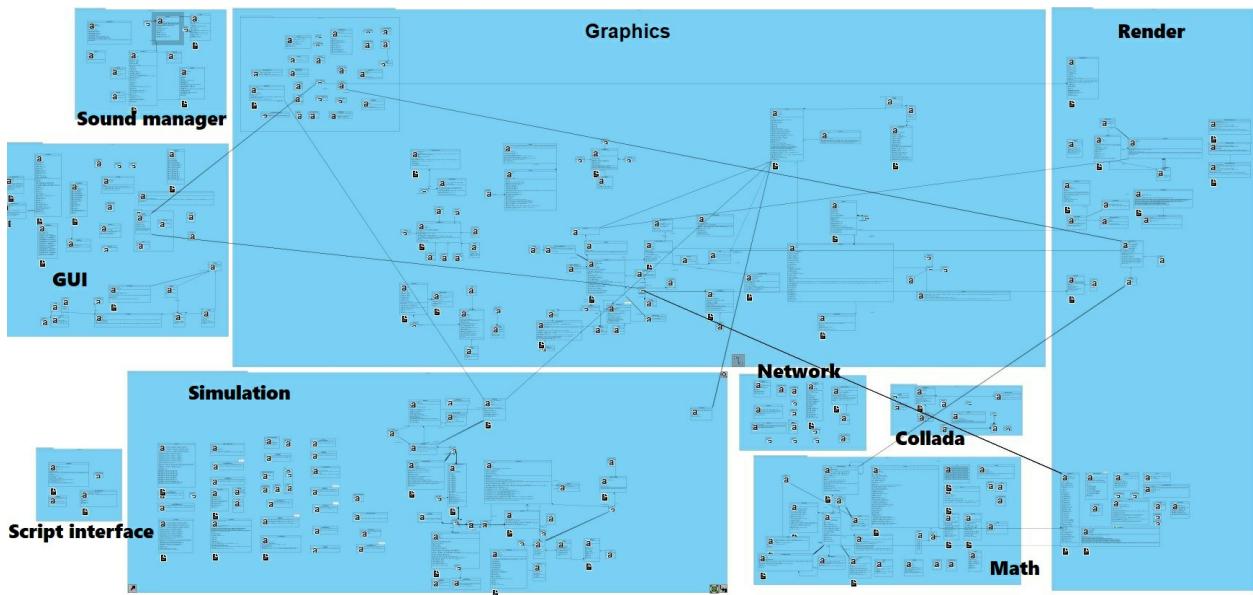


fig. 13 Partie du diagramme des classes concernant la gestion des unités

Nous avons donc décidé de réduire la présentation de cette partie à un élément unique du diagramme et une des fonctionnalités de base du jeu : la gestion des unités.

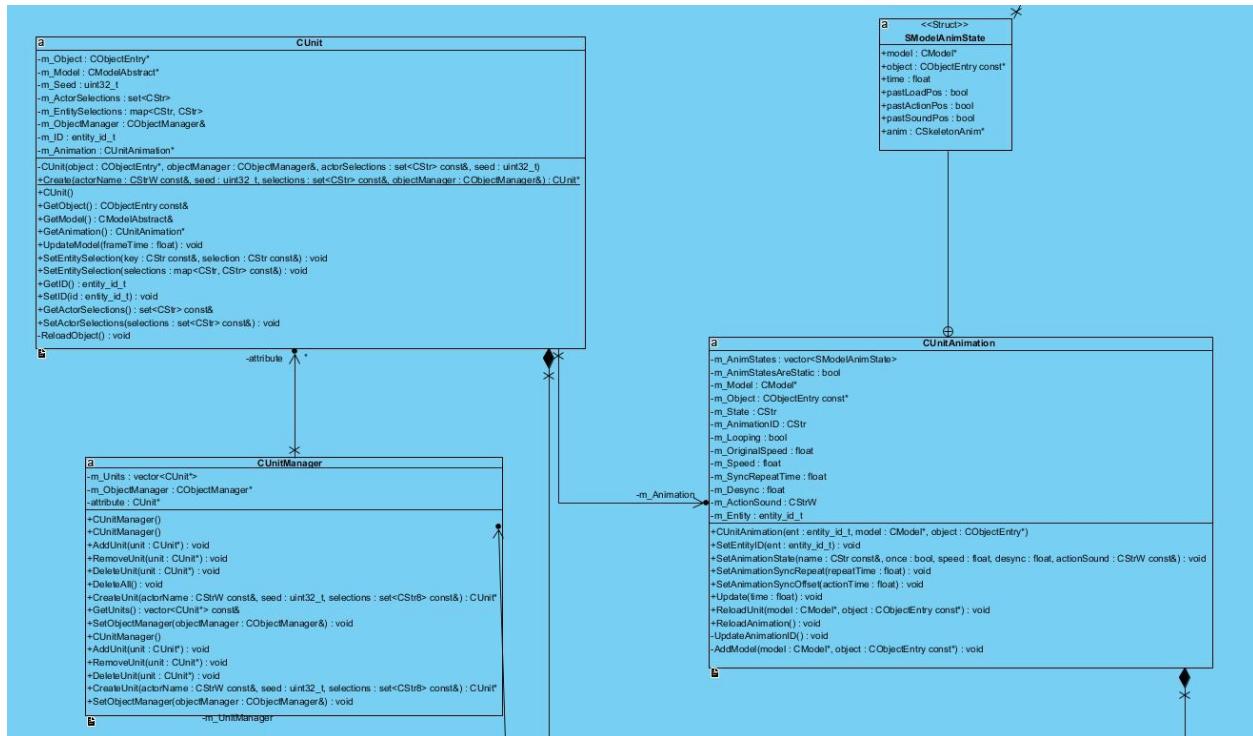


fig. 14 Partie du diagramme des classes concernant la gestion des unités

Cette partie du diagramme est assez représentative du reste du code. On trouve une classe de base gérant les unités appelées "CUnit" celle-ci comprend notamment un objet animation et un objet model pour la représentation graphique de l'unité. Ce qui est intéressant ici c'est qu'il existe une classe spécialisée appelée "CUnitManager" qui permet de gérer toutes les unités au même

endroit et de rapidement faire des opérations dessus. Ce schéma se retrouve souvent dans le logiciel sans doute pour pourvoir plus facilement optimiser l'usage de la mémoire, ce problème étant très courant dans les jeux vidéos. La classe "CUnit" utilise également un Abstract Design Pattern avec un constructeur privée et une méthode static "Create" se chargeant des allocations mémoires à faire, mais nous reviendrons sur les Design Patterns dans une autre partie. La classe est ensuite relié à une autre partie du code que voici.

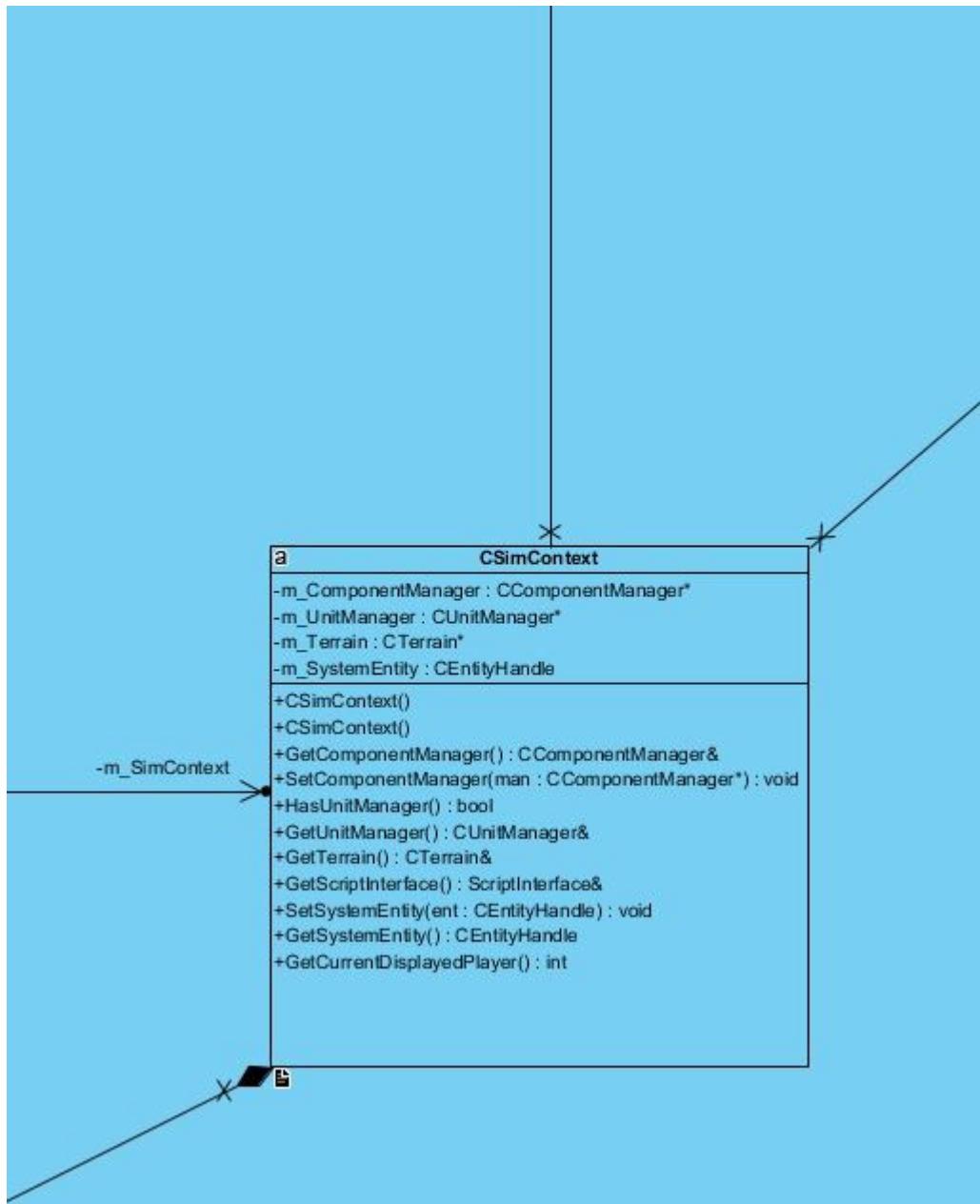


fig. 15 Diagramme UML de la classe "CSimContext" du package Simulation2

Il s'agit du contexte auquel va se référer le logiciel lorsqu'une partie sera lancée. Ce contexte possède donc un gestionnaire d'unités avec un terrain, un gestionnaire d'entité regroupant tous les types d'objet qui peuvent intervenir dans une partie et un gestionnaire de composants. Cet object regroupe donc un grand nombre d'informations essentielles au bon déroulement d'une partie.

Process View

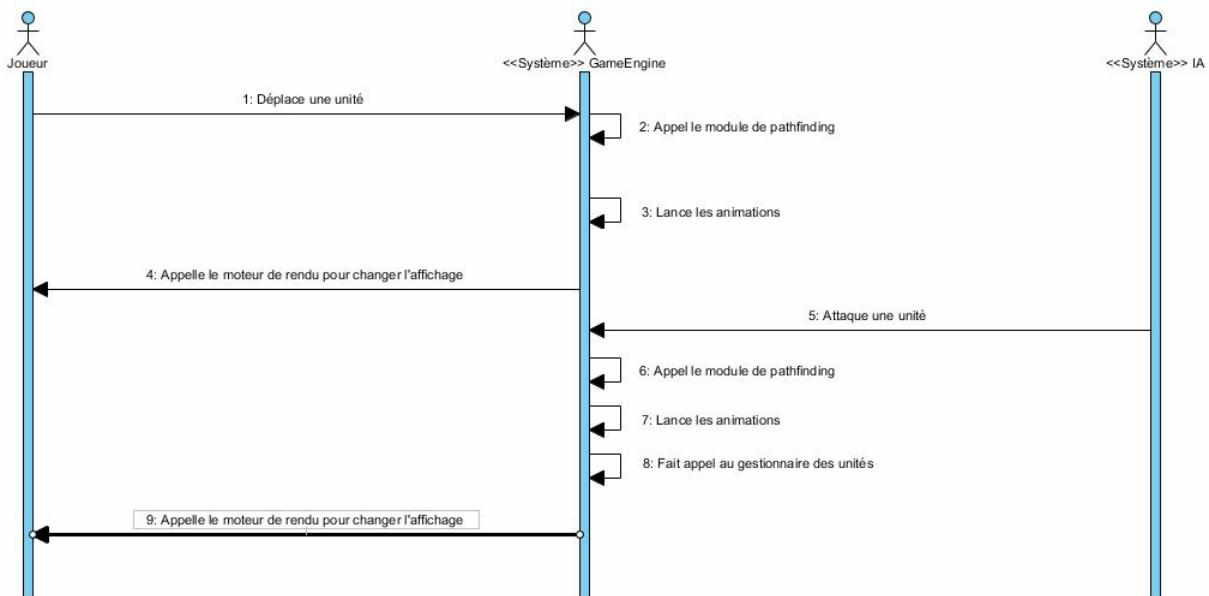


fig. 16 Diagramme de séquence dans le cas d'une partie jouée contre une IA

Ce diagramme de séquence nous apprends un peu plus de chose quand au fonctionnement du logiciel. Le moteur de jeu gère ici l'aspect controller, il reçoit des évènements puis appelle en conséquence les bons modules chacun gérant une partie du logiciel comme l'IA, le rendu, la caméra, le pathfinding, les animations, ect

Conclusion sur l'architecture

Les différents diagrammes que nous avons présentés nous ont permis d'en déduire une architecture pour le logiciel que voici:

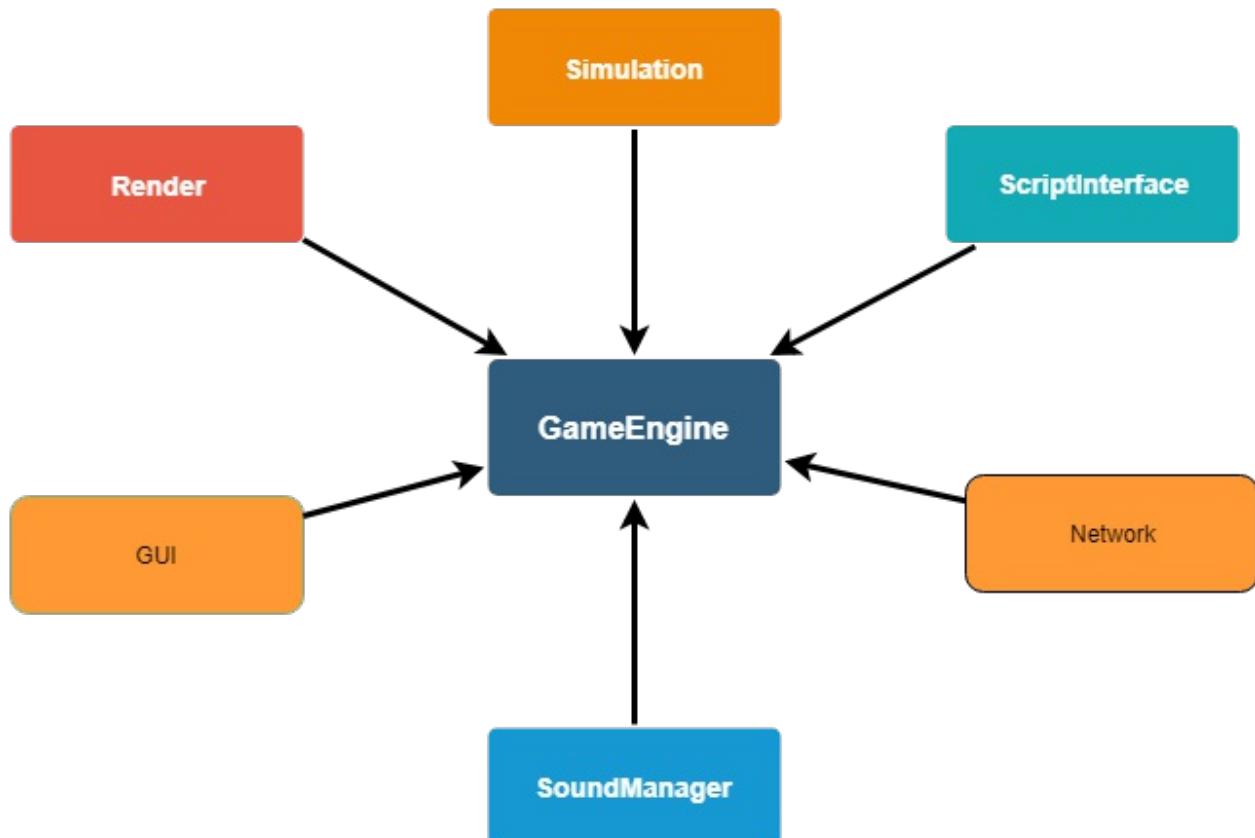


fig. 17 Architecture globale du logiciel

L'architecture est donc centré autour du moteur de jeu qui regroupe les différents composants essentiels au fonctionnement du jeu. Le moteur s'occupe de récupérer et d'interpréter les évènements transmis par le joueur (selectionner un élème, charger une partie, ...) et de synchroniser l'utilisation de tous ces composants pour éviter les problèmes de concurrences.