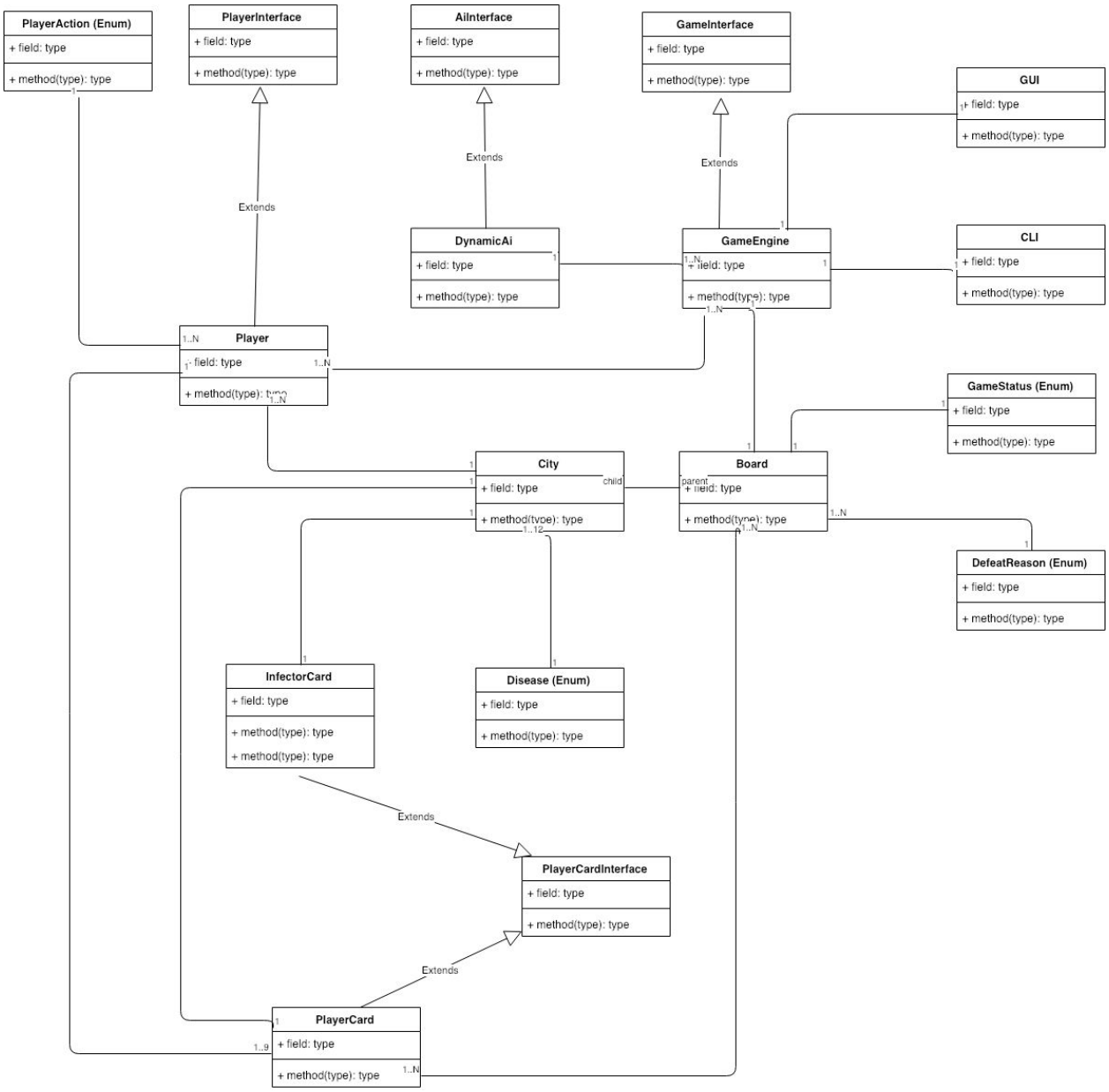
Document développeur

Introduction

La programmation de ce jeu fut pour nous un chemin difficile mais riche en apprentissage, difficile de part la richesse et la complétude de ce projet nécessitant une forte organisation et une importante phase de documentation. L'ampleur de celui-ci nous a pousser à définir une architecture générale, dans l'objectif d'une meilleure compréhension du projet, facilitant par la même voie une meilleure vision du rendu final.

Architecture



Méthode de travail

Moteur de jeu

Après plusieurs échanges informels autour du projet, chacun en ayant lu les consignes et règles de son côté, nous nous sommes réunis à deux reprises pour poser les base de notre future architecture. Ces réunions avec tableau ont été l'occasion de discuter de choix stratégiques concernant le moteur de jeu (utilisation d'une nouvelle classe Board instanciée par le GameEngine). Nous avons exploré différentes possibilités en nous basant sur le template MVC.

Nous avons tenté de généraliser au maximum notre future implémentation et considérant dès le départ qu'un GameEngine pourrait faire jouer plusieurs joueurs (humain ou AI), sur plusieurs parties différentes. En construisant notre diagramme de classes, nous nous laissions une part d'évolutivité au fur et à mesure que le projet serait développé (notamment sur les méthodes de chaque classe).

Ensuite, une fois les bases fixées et comprises par tous, chacun a pris la responsabilité d'une classe concourant au bon fonctionnement du moteur de jeu : Board, Player, City, PlayerCard & InfectorCard, GameEngine.

CLI / GUI

Une fois le moteur de jeu assemblé et fonctionnel, c'est-à-dire répondant à notre tests unitaires nous avons fait évoluer le squelette de nos controllers : d'abord le CLI puis le GUI. Une personne a travaillé à l'adaptation du CLI et une autre s'est chargée entièrement du GUI.

Afin de réaliser le CLI des méthodes ont été implémentées dans player, ces méthodes concerne principalement des méthodes d'affichage permettant au joueur de comprendre l'état du jeu.

Pour ce faire, une première base a été "dessinée" avec JavaFX afin de reproduire les villes, leur couleur, localisation et connexions. Ensuite, différents éléments dynamiques ont été ajoutés progressivement : la position du joueur, les invites d'actions, l'affichage des cartes, du taux d'infection ainsi que des outBreak notre objectif et d'offrir une interface riche en information permettant au joueur d'établir une réelle stratégie.

Intelligence Artificielle

Dès que notre CLI était fonctionnel, nous avions donc notre modèle MVC complet. Après avoir joué plusieurs parties en manuel , nous avons développé une première IA symbolique (NotSoStupidAI), très basique pour nous familiariser avec les méthodes disponible via l'interface de l'IA. Cette IA qui avait un taux de victoire très faible (entre 1 et 6) utilisait des actions prédéfinies dont la priorité a été définie selon nos préjugés et des avis de joueur récolter sur divers forum.

Dans un second temps nous avons réfléchi sur 2 stratégies à utiliser pour concevoir un IA plus robuste. La première qui n'a pas été retenue aurait été de lancer un grand nombre de partie avec notre première IA et d'analyser les actions qui rapportaient le plus de victoire. La seconde stratégie qui a été finalement choisie est de s'inspirer de l'algorithme min-max étudié en cours d'IA.
Ainsi nous avons implémenter dans l'ia une classe interne permettant de réaliser des simulations sur une représentation simplifiée du jeu que nous avons nommée FakeGame. Le principe de cette IA que nous avons nommée DynamicAI est de tester toutes les actions associées aux destinations possibles sur l'objet FakeGame et d'utiliser une fonction d'évaluation pour déterminer l'action qui a été la plus efficace.

Actuellement notre fonction d'évaluation se base sur 5 critères principaux du Board :

- Le nombre total de villes infectées (pondéré par le nombre d'infections de chaque ville, c'est-à-dire par la probabilité de déclencher une réaction en chaîne
- La position du joueur (qui détermine sa capacité à traiter des maladies lors de son action suivante)
- Le nombre de villes proches de déclencher une réaction en chaîne (pondéré par son nombre de voisins)
- Le nombre de cubes maladie restant (par maladie)
- Le nombre de player cards restantes (par maladie), influent directement sur la capacité de l'IA à trouver un vaccin dans le futur

Dans l'état notre IA possède deux principales limites que nous n'avons pas eu le temps de lever. La première est que nos simulations se déroule seulement pour un point d'action. A l'origine nous avions prévu une structure pour fakeFame qui permettrait de faire des simulations avec toutes les combinaisons de 4 actions possibles(ce qui explique l'attribut actions non utilisé dans fakeGame). Cette amélioration consommerait beaucoup plus de ressource et nécessite l'utilisation de thread mais elle serait probablement plus efficace que notre IA actuelle.

La seconde limite de notre IA est que les critères et les points attribués dans la méthode d'évaluation découle de test réalisés avec notre première IA. Une piste d'amélioration avait été d'utiliser notre première IA sur un très grand nombre de partie, de stocker les résultats dans un fichier texte puis essayer d'analyser les résultats avec des méthodes d'analyse de données

pour déterminer les actions qui apportent le plus souvent la victoire. Concernant cette piste nous l'avons abandonné car son implémentation nous semblait trop long et le résultat incertain.

La stratégie utilisée pour le discard a été la même pour les 2 IA, la stratégie consiste à conserver les carte dont la couleur est majoritaire (si la maladie n'a pas encore été soignée) jusqu'à avoir 5 cartes de même couleurs.

Fonctionnalités supplémentaires

- Mode multijoueur (CLI uniquemet)
- GUI Multijoueur
- Plusieurs AI (NotSoStupidAI et DynamicAI)

Difficultés rencontrées

- Implémentation du Board :Difficile de délimiter son rôle
- Implémentation du GameEngine : différentes loop, fonctionnement de discard, besoin de trouver les différences avec moteur du prof
- Implémentation de Player : playTurn, fonctionnement des actions, exceptions
- Amélioration de l'IA : difficultés avec threads, Exception coutent une action
- GUi, difficulté rencontré avec Javafx (Exception, apprentissage, code très riche)

Fonctionnalités

- Jeu à 1 joueur : humain ou IA, avec différentes tailles de main max, différents graphs,
- Jeu multijoueur (1 à 4 joueurs)
- CLI
- GUI