Rapport du projet Takenoko

Équipe Gigaboss

08/01/2021

Étudiants SI3

Enzo Manuel
Eric Naud
Loic Madern
Antoine Le Calloch

Remerciements

Tout d’abord, nous tenions à remercier M.Collet Philippe, responsable de projet, pour sa disponibilité et ses cours d’une aide précieuse.

Nous remercions aussi M.Cousté Mathias, M.Mortara Johann et M^{me}.Dery Anne-Marie pour leur aide et leur soutien tout au long du projet.

Sommaire

I Synthèse du projet 5
1.1 Synthèse 5
1.2 Implémentation 5
1.3 Stratégies des robots 5
1.3.1 Stratégies 5
1.3.2 Robots 6
1.4 Fin d'une partie 6
1.5 Confiance 6
II. Qualité du code 7
2.1 Architecture 7
2.1 Code 7
2.2 Test 7
II. Points forts du projet 8
2.1 Organisation 8
2.3 Découpage 8
III. Points faibles du projet 9
3.1 Stratégies des robots 9
3.2 *Milestone* 9
IV. Rétrospective 10
Annexe 11
Bibliographie 11

Introduction

Dans le cadre de notre première année de cycle ingénieur en informatique à Polytech Nice Sophia, nous avons l'opportunité de faire un projet informatique de deux mois.

Ce projet est la réalisation d'une version informatisée du jeu "Takenoko" d'Antoine Bauza. La représentation et le moteur du jeu ainsi que des robots jouant intelligemment ont dû être implémentés.

Le programme a été codé en java.

I Synthèse du projet

1.1 Synthèse

La simulation du jeu se fait par l'instanciation d'un objet Game qui possède une liste de robots en argument. Le nombre de joueurs peut varier de 1 à 4. La sortie standard affiche le pourcentage de victoire, de défaite et d'égalité ainsi que le score moyen pour chaque robot, pour un total de 1000 parties.

1.2 Implémentation

L'entièreté du jeu a été implémentée ainsi que toutes les règles.

1.3 Stratégies des robots

Chaque tour, un robot joue deux actions différentes parmi les actions suivantes : piocher une mission ou un canal, placer une parcelle et déplacer le panda ou le jardinier.

Au début de son tour, un robot joue l'action que lui procure la météo. Le robot choisit de piocher ou non une mission, il décide le type de mission qu'il veut piocher.

Enfin, pour chacune de ses missions, le robot calcule le nombre de coups minimum qu'il doit faire pour finir la mission. Ainsi, il fera l'action la plus optimisée pour finir la mission qu'il peut finir le plus rapidement possible.

S'il lui reste des actions à faire pendant le tour. Il réitère la méthode ci-dessus à partir du calcul des coups.

1.3.1 Stratégies

Pour calculer le nombre de coups pour finir une mission et savoir quelle est la meilleure action à jouer, le robot utilise plusieurs stratégies.

Il existe trois stratégies différentes : *MissionPandaStrat*, *MissionParcelStrat* et *MissionPeasantStrat*. Chaque stratégie est spécialisée dans la résolution d'un type de mission. *MissionPandaStrat* est spécialisée dans la résolution de mission panda, *MissionParcelStrat* dans la résolution de mission parcelle et *MissionPeasantStrat* dans la résolution de mission jardinier.

1.3.2 Robots

Nous avons implémenté cinq robots. Le premier type de robot que nous avons implémenté est le *RandomBot*. Contrairement aux autres robots, il ne calcule jamais les coups pour finir une mission. Il choisit et utilise ses actions aléatoirement.

Ensuite, nous avons implémenté trois robots : *PandaBot*, *ParcelBot* et *PeasantBot*, chacun est spécialisé dans un type de mission. Leur principale différence est le fait que ces robots piochent seulement le type de mission dont ils sont spécialisés. Certaines météos sont aussi jouées différemment.

Le dernier robot : *IntelligentBot* pioche différents types de missions en fonction de l'avancement de la partie et du nombre de joueurs présents dans la partie.

1.4 Fin d'une partie

Une partie s'arrête lorsqu'un joueur a fini 7,8 ou 9 missions si la partie est composée respectivement de 4,3 ou 2 joueurs ou si les ressources sont épuisées.

Le gagnant est le joueur ayant gagné le plus de points grâce à ses missions complétées. En cas d'égalité, le gagnant est celui ayant complété le plus de missions de type panda. S'il y a de nouveau égalité, il n'y a plus de moyen pour départager les joueurs.

1.5 Confiance

Nous avons confiance en notre réalisation. Le code ne retourne pas d'erreur. Grâce à nos exceptions, nous vérifions que les robots n'enfreignent pas les règles. De plus, nos nombreux tests permettent de vérifier que nos méthodes fonctionnent dans les cas limites. Les erreurs majeures ont donc été minimisées.

II. Qualité du code

2.1 Architecture

L'architecture a été le point le plus compliqué à gérer. Nous l'avons modifiée de nombreuses fois pour arriver à un bon résultat. En effet, nous avons mis du temps à obtenir un code ayant un couplage faible et une cohésion forte sur l'ensemble du projet.

Actuellement chaque classe a des responsabilités, les robots peuvent interagir avec le moteur du jeu uniquement grâce à la classe *GameInteraction* afin d'empêcher tout type de triche, et d'être sûrs des informations auxquelles ils ont accès .

L'architecture est faite de telle sorte que d'autres règles peuvent s'ajouter facilement à celles existantes notamment grâce aux nombreux types créés.

2.1 Code

Toutes les règles du Takenoko ont été implémentées. Les robots sont intelligents, ils jouent de manière optimisée. De plus, le code est commenté et documenté.

La dernière semaine, nous avons décidé de modifier la structure des robots pour les rendre plus performants. Cela s'est fait en dépit d'un code optimisé.

Dans l'ensemble, le code est relativement rapide puisque 2000 parties sont jouées en 2 minutes lorsque quatre joueurs sont présents.

2.2 Test

Les tests sont de bonnes qualités car ils couvrent la quasi-totalité du code implémenté ainsi que les cas limites. Des *mockitos* ont été réalisés pour gérer l'aléatoire et pour tester certaines classes comme celle du package mission. Par exemple pour faire jouer un robot sous certaines conditions.

Or, nous avons appris à utiliser *mockito* tardivement, par conséquent, certains tests faits au début du projet, n'utilisent pas *mockito*, même si ce procédé simplifiait ces tests.

II. Points forts du projet

2.1 Organisation

Durant ce projet, notre organisation a changé au cours des semaines. Tout d'abord, nous avons codé la première *Milestone* ensemble pour mettre en place facilement la structure du projet.

Pour chaque nouvelle *Milestone*, nous nous sommes divisé les tâches. Mais, nous avons alternés entre travailler seul, en duo, en trio ou tous ensemble. Cela a permis de garder une certaine rapidité de programmation tout en favorisant l'entraide. Dès qu'une personne avait un problème, nous le disions pour que les autres personnes du groupe viennent nous aider.

Chaque membre du groupe s'est spécialisé dans la réalisation des robots ou du moteur de jeu jusqu'à la fin décembre. Cependant, étant un groupe au niveau homogène, chacun a participé au projet en donnant des idées pour ajouter et modifier des fonctionnalités. Notamment grâce aux nombreux débats qui ont permis de garder les meilleures idées à chaque problème survenu.

Nous savions quand et pourquoi quelqu'un travaillait car nous communiquions beaucoup. Ainsi, nous avons évité un certain nombre de problèmes tels que des merges.

2.3 Découpage

Le découpage a été bon, nous avons amélioré les robots et ajoutés des fonctionnalités au jeu tout au long du projet. La façon dont nous avons découpé le projet nous a permis de rajouter les fonctionnalités du jeu facilement.

III. Points faibles du projet

3.1 Stratégies des robots

Malgré le fait que nos robots aient une stratégie qui permet de résoudre efficacement les missions, nous aurions pu améliorer leur intelligence en implémentant des fonctionnalités ci-dessous.

Les stratégies des robots sont focalisées sur eux-mêmes. Elles ne prennent pas en compte le jeu des adversaires. Une stratégie potentielle aurait été d'essayer de comprendre la stratégie de l'adversaire et de faire de l'antijeu.

Également, le classement des joueurs n'est pas considéré dans les stratégies. Nous aurions pu développer le fait que lorsqu'un joueur est premier, qu'il privilégie le nombre de missions faites à leur valeur. Inversement, un joueur dernier privilégie les points.

Enfin, les robots ne savent pas compléter les six missions les plus complexes : les trois missions parcelles bicolores, et les trois missions jardinier qui ont besoin de trois pousses de bambou sur plusieurs parcelles, ce qui aurait pu être corrigé.

3.2 *Milestone*

Jusqu'à la dernière semaine, nos *milestones* étaient fonctionnels, nous avions fermés des *milestones* un ou deux jours en retard pour finir des fonctionnalités. Mais, nous avons résolus ce défaut lors de la dernière semaine, en effet, nous avons créés et fermés une *milestone* par jour.

IV. Rétrospective

Le projet a démarré avec l'étude de la logique intrinsèque du jeu Takenoko. Cela nous a permis de comprendre le jeu et les stratégies pouvant être mis en place dans le futur. Cette démarche de compréhension du problème posé est très importante car elle a formé l'ossature de notre projet et a permis de rajouter les bonnes fonctionnalités au bon moment.

Cependant, nous avons mis un certain temps avant de comprendre comment créer une bonne architecture entre nos différentes classes. Ainsi, nous avons dû faire de nombreux ré usinages ce qui nous a ralentis. Un travail de réflexion en amont plus conséquent permettra dans nos futurs projets de ne pas répéter cette erreur…

L'organisation des tâches et le travail à plusieurs nous ont semblé être une bonne pratique pour travailler efficacement. La documentation ou encore la création de tests ayant des noms explicites sont de bonnes pratiques que nous garderons.

De la même manière, la communication et l'écoute ont été primordiales pour l'avancée de notre projet. Cela a permis de résoudre les problèmes rapidement lorsqu'ils sont survenus.

Enfin, ces deux mois de travail nous ont permis d'apprendre à utiliser correctement git pour gérer un projet ce qui sera utile pour nos prochains travaux.

Annexe

Bibliographie

Règle du jeu : https://www.trictrac.net/jeu-de-societe/ressource/clique/383f6c35aed2eaace1a85aec076074c1/takenoko