Polytech Nice-Sophia
Université Côte d'Azur

Emma Glesser
Thomas Prévost
Tom Niget
Emmeline Vouriot

Rapport de projet ET²

Sommaire

1. Introduction

Le Takenoko est un jeu de plateau multijoueur comportant des tuiles hexagonales (aussi appelées parcelles) constituant un plateau en forme de ruche, des concepts d'irrigation, de culture de bambou ainsi que des objectifs à remplir.

Le but du jeu est de remplir en premier un certain nombre d'objectifs et de remporter un maximum de points. Ces différents objectifs pourront être relatifs à la configuration actuelle des parcelles, la présence de bambous sur le terrain ou dans la main du joueur. Pour cela, le joueur sera aidé du jardinier et du panda qui respectivement cultive et mange du bambou, ainsi que des conditions climatiques qui auront un impact sur les actions réalisables durant le tour.

Le jeu doit se dérouler entre plusieurs robots, lancer 1000 parties et afficher les statistiques des parties. Un affichage plus précis des parties doit aussi être disponible.

Dans un premier temps, nous présenterons la conception de notre projet, de l'architecture aux tests. Puis nous parlerons du développement du jeu dans le temps et enfin, nous conclurons en faisant une rétrospective sur tout le travail effectué.

2. Projet

2.1. Problèmes et solutions retenues

2.1.1. Gestion du système de tuiles

Le plateau étant constitué de tuiles en forme d'hexagones, nous étions initialement tentés de nous reposer sur un repère constitué de 3 axes. Cependant, deux axes suffisent à naviguer partout sur le plateau, nous avons donc opté pour un repère 2D. On représente ces axes en choisissant deux paires de côtés opposés entre eux sur l'hexagone central puis en traçant un trait pour les relier ; la paire restante de côtés opposés peut être représentée par la somme des deux axes. Le résultat n'est pas intuitif mais évite la redondance de l'information. Les principes SOLID sont ici bien respectés, tout particulièrement en ce qui concerne les responsabilités. À noter que les tuiles contiennent également des références aux tuiles voisines, ce qui permet d'effectuer de manière plus performante les opérations travaillant sur le voisinage des parcelles que sur le plateau dans sa globalité.

2.1.2. Déroulement d'une partie

Une partie est gérée par la classe Game. Cette classe joue le rôle d'arbitre au sein d'une partie et effectue les communications avec les différentes autres classes. Une partie commence par mélanger les cartes au sein des decks qui contiennent les cartes objectifs ainsi que les tuiles. Elle lance ensuite des tours jusqu'à ce qu'un joueur gagne puis détermine le vainqueur. Un tour détermine la météo puis demande au robot l'action qu'il veut effectuer. Ce fonctionnement permet de faire interagir clairement les classes. La fonction centrale a été subdivisée en de nombreuses fonctions pour améliorer la lisibilité et la responsabilité de chaque fonction, mais la fonction principale qui appelle toutes ces sous fonctions est quasiment impossible à tester. Nous avons essayé de respecter au mieux les principes GRASP même si le résultat n'est pas parfait.

2.1.3. Interaction bot-jeu

La classe abstraite DecisionMaker définit des fonctions qui seront appelées par Game. En paramètre, il y a des listes parmi lesquelles DecisionMaker sélectionne les éléments à renvoyer. Seuls les choix

valides sont donnés à DecisionMaker puis Game vérifie par la suite que les valeurs renvoyées sont légales. De cette manière, les bots sont incapables de tricher. Chaque bot implémente les fonctions de DecisionMaker. Cette solution permet de respecter le principe SOLID, car très modulaire entre autres. L'utilisation des bots ayant systématiquement lieu via leur interface parente, le polymorphisme est largement utilisé. Ainsi, il est très facile d'ajouter de nouveaux bots. Il y a aussi un faible couplage entre DecisionMaker et Game.

Nous avons fait le choix de séparer la partie décisionnelle (DecisionMaker) et matérielle (Player) d'un joueur en suivant le principe suivant : un arbitre (Game) aurait tous ses amis au téléphone qui joueraient au Takenoko. Il leur exposerait les choix qu'ils peuvent prendre et les joueurs répondraient. L'arbitre modifierait lui-même le plateau et les plateaux individuels des joueurs. Ainsi, les responsabilités sont très claires et l'ajout d'un robot est d'autant plus simplifié.

2.1.4. Simulation et visualisation des parties

Une simulation peut être lancée avec une ligne de commande comportant tous les paramètres utiles (nombre de robots, types de robots, nombre de parties). Sont affichées ensuite les statistiques de la simulation. Pour afficher les états de parties, différents niveaux de logs sont disponibles. La simulation et visualisation de partie sont donc modulables. La simulation des parties agit comme contrôleur, ou interface entre le client et le jeu.

Afin de tirer profit des multiples cœurs des processeurs modernes, le lancement en parallèle des parties est possible. En pratique, le temps d'exécution est grandement réduit, et est environ divisé par le nombre de cœurs du processeur de l'ordinateur.

2.1.5. Des robots intelligents

Après avoir créé le premier bot qui suit une stratégie aléatoire, il fallait ajouter un bot avec une vraie stratégie. Pour éviter d'avoir un bot à stratégie fixe (qui devrait être entièrement refait à chaque ajout d'une fonctionnalité), nous avons choisi d'utiliser pour ce premier bot intelligent, et adaptable, la stratégie min-max. L'algorithme min-max fonctionne récursivement. Pour chaque coup possible, il clone l'état actuel du jeu, effectue le coup sur cette copie et regarde si les objectifs sont remplis. Si ce n'est pas le cas, il lance un appel récursif pour re-tester tous les coups suivants. Cela jusqu'à la profondeur qu'on lui a spécifiée (paramètre depth). Ainsi, le premier appel de la fonction renvoie le score que pourrait rapporter chaque coup (y compris adverse), en appliquant une pondération pour prioriser les coups rapportant des points dans le moins de tours possibles.

L'usage de l'algorithme min-max présente des avantages et des inconvénients. L'énorme avantage est qu'il ne s'agit pas d'une stratégie « fixe », mais adaptable. De plus, si le temps et la puissance de l'ordinateur permettent de paramétrer une profondeur suffisante, il est possible de prévoir tous les coups à l'avance et donc le bot ne peut théoriquement pas perdre. Par conséquent, c'est aussi un inconvénient majeur. En effet, le nombre de coups à simuler à un fort impact négatif sur la complexité *O((nombre de coups possibles par tour)^profondeur)*. Dans le Takenoko, beaucoup de coups sont possibles à chaque tour, et comme le temps d'exécution croît exponentiellement, il n'est pas possible en pratique de dépasser la profondeur 2 tout en gardant un temps de calcul raisonnable. De plus, il est assez complexe « d'annuler » un coup. On est donc obligés de simuler sur une copie du jeu, donc de cloner ce dernier à chaque fois, ce qui est très consommateur en ressources. Certaines décisions du bot comme le choix de la météo ou d'une pile où piocher un objectif ne sont pas adaptées à un algorithme min-max d'une si faible profondeur, la stratégie est donc statique et décidée empiriquement par le développeur. En outre, le fait d'avoir plusieurs objectifs possibles, et de ne pas

connaître ceux de l'adversaire, rend l'algorithme min-max moins pertinent. Un réseau neuronal aurait peut-être été plus adapté (il n'était cependant pas réalisable par manque de temps).

De plus, le besoin de cloner le jeu pour modifier une copie demande de modifier les modificateurs d'accès pour les rendre plus permissifs et donc violer les responsabilités de la classe Game. Il aurait peut-être fallu créer une sous-classe à Game pour qu'elle puisse être manipulée en tant que clone par les bots.

Un troisième bot a été ajouté une fois les fonctionnalités finies, plus simple cette fois-ci ; ce dernier suit une stratégie d'expansion sur le plateau afin de privilégier les objectifs parcelle en posant le plus de parcelles possibles sur le plateau. Beaucoup plus simple que le bot min-max, il atteint cependant des scores corrects (70%-30% contre l'aléatoire).

2.1.6. Bien effectuer les tests

Une partie très importe, mais souvent négligée dans les projets, est les tests ; afin de confirmer que le code est viable et conforme. Si certaines classes ou méthodes étaient assez évidentes à tester, ce n'était pas le cas de classes comme Game ou les bots. Mais de manière générale, pour réaliser de bons tests, il fallait simuler les cas les plus courants, limites et interdits. Pour cela il a souvent fallu s'aider de schémas du plateau pour créer les situations voulues ainsi que de schémas des scénarios attendus (pour la météo, le jardinier ou le panda). Les tests ont donc été faits efficacement, permettant d'avoir une grande couverture.

3. Déroulement du projet

3.1. Choix des milestones et issues

Chaque milestone avait du contenu vertical car chacune devait représenter une instance autonome du projet où chaque fonctionnalité était utilisée. Elle comportait un certain nombre de tâches (issues) ayant une valeur client. Chaque issue était attribuée à un membre de l'équipe.

Une issue était considérée comme finie lorsque tous ses tests étaient eux aussi finis. Chaque milestone voyait également une mise à jour des bots existants pour y ajouter les dernières fonctionnalités.

3.1.1. Milestone 1 – jeu basique

Afin de pouvoir faire des slices les plus verticales possibles, nous nous sommes assurés dès la première milestone d'avoir une partie, c'est-à-dire faire jouer des joueurs et déterminer un vainqueur. Pour qu'une partie puisse être gagnée, il faut remplir un certain nombre d'objectifs. Les objectifs les plus basiques reposent sur la présence d'une configuration spécifique de tuiles colorées sur le plateau. Pour la milestone 1, il nous fallait donc : un plateau avec des tuiles colorées, des objectifs de parcelles, des joueurs, une partie qui réunisse le tout ainsi qu'une manière d'interagir avec le jeu, de prendre les décisions. Cette première milestone étant bien remplie, pour la première interaction avec le jeu, nous avons fait des tests préliminaires avec une classe très simple de debug qui permettait de mettre le jeu en route en demandant de rentrer manuellement les valeurs désirées. Cette classe n'était pas très rigoureuse car elle n'était là que temporairement. L'équipe a également ajouté un robot aléatoire quand les autres fonctionnalités ont été finies.

Nous avons ensuite suivi des raisonnements semblables pour déterminer le contenu des milestones suivantes, gardant toujours la plus grande verticalité possible.

3.1.2. Milestone 2 – bambou, irrigation, jardinier

Les concepts les plus essentiels à ajouter ensuite sont l'irrigation et le bambou. La partie propose au bot de prendre et/ou poser des irrigations qui peuvent maintenant être stockées par le joueur. Les irrigations et le jardinier permettent de faire pousser du bambou. Les objectifs du jardinier sont ajoutés aux objectifs de parcelle afin que l'utilisation du bambou soit utile à la partie. Lorsque ces fonctionnalités ont été finies, un bot min-max a été ajouté.

3.1.3. Milestone 3 – jeu complet

Le panda, ses objectifs, le climat et les améliorations de parcelles ont été ajoutés. Les joueurs peuvent à présent déplacer le panda sur le plateau et stocker du bambou grâce à lui. Ce bambou permet de compléter des objectifs panda. Les conditions météo permettent aux joueurs de tirer des améliorations à ajouter aux tuiles, afin de les aider à faire pousser le bambou ou l'empêcher de se faire manger par le panda. Nous avons un peu surchargé cette milestone, n'ayant pas prévu le manque de temps dû à la préparation d'examens.

3.1.4. Milestone 4 – optimisation + bot

Étant donné la consigne de faire utiliser toute nouvelle fonctionnalité par un robot, l'équipe devait, chaque semaine, finir les fonctionnalités avant de mettre à jour les robots. Ainsi, certains bots n'avaient pas un fonctionnement optimal pour chaque release. Ils ont donc été totalement mis à jour lors de cette dernière milestone. L'équipe en a aussi profité pour ajouter un nouveau robot plotpattern ainsi que de s'attarder sur la qualité du code et ses performances. Des tests ont aussi été ajoutés pour garantir le bon fonctionnement global du jeu.

3.2. Avancement du projet

Toutes les fonctionnalités du jeu de plateau ont été implémentées et une grande partie du code est testée (86% de couverture), le tout est stable.

Les bots random et plotpattern sont fonctionnels.

Le bot min-max n'est pas totalement fini : certaines des décisions prises ne sont pas encore récursives, et donc limitées à une profondeur de 1.

Nous avons une grande confiance dans le moteur et la représentation du jeu mais nous sommes conscients du manque de rigueur (au niveau du clonage) du bot min-max, dans lequel nous avons donc beaucoup moins confiance.

4. Conclusion

4.1. Synthèse

Nous avons complété toutes les fonctionnalités motrices du jeu : plateau, joueurs, irrigations, bambou, améliorations, intempéries, jardinier, panda et objectifs.

Nous avons complété 2 robots simples (un robot aléatoire et un autre suivant une stratégie fixe). Un troisième robot est disponible mais pas tout à fait complété : ce robot est censé permettre un choix de complexité, mais celle-ci ne s'applique pas encore à toutes ses fonctionnalités.

Pour ce qui est des tests unitaires, nous nous sommes efforcés de les écrire au fur et à mesure, même si cela n'a pas toujours été facile.

4.2. Rétrospective sur le projet

Au début du projet, nous avons pris du temps pour définir comment organiser l'architecture ce qui nous a permis de respecter les patterns vus en cours ainsi qu'avoir une très grande lisibilité. Nous avons cependant omis de suivre cette démarche pour la conception des bots ce qui nous a forcé à modifier l'architecture de manière décevante *(cf. des robots intelligents)*, la prochaine fois, il nous faudra donc suivre cette démarche sur la totalité des éléments du projet. Nous avons cependant autorisé la gestion d'une possible mémoire des bots (dont nous ne nous sommes pas servis mais ne nous a bloqué en aucun point), nous savons donc que la démarche est fondamentalement bonne.

Chaque livraison du projet était une entité tout à fait fonctionnelle riche en nouvelles fonctionnalités *(cf. choix des milestones et issues)*. De plus, nous distribuions une issue par membre de l'équipe pour nous permettre de travailler en parallèle le plus possible, pour ne pas s'attendre les uns les autres avant de commencer à travailler. Nous conserverons la même méthode de répartition d'issues dans les milestones à l'avenir.

Nous avons également mal estimé le temps dont nous disposions pour travailler, notamment vers la fin du projet à l'approche des examens, et nous avons donc été trop gourmands sur la dernière milestone que nous avons trop remplie, comparé au temps dont nous disposions. Il nous faudra donc tenir compte des aléas lors de la prévision des milestones pour notre prochain projet.

Les tests unitaires ont bien été réalisés tout au long du projet *(cf. bien effectuer les tests)*, ce qui nous a été très utile lors des quelques régressions que nous avons eues, où nous avons pu cibler rapidement l'origine des bugs. Nous avons appris à nous servir de Mockito, qui s'est montré très utile dans les tests de classes compliquées, nous nous en servirons à l'avenir.

La communication constante (via Slack) nous a permis de nous entraider avec efficacité et de garder une grande cohésion.

4.3. Conclusion

Nous avons énormément appris sur le travail en équipe dans ce projet. L'équipe en elle-même était très agréable et nous avons beaucoup apprécié travailler ensemble. C'était en effet un gros projet qu'il était impossible de faire seul, et la charge de travail a été très bien répartie. Malgré quelques points faibles, nous avons un avis très positif du travail que nous avons réalisé.

5. Bibliographie

Règles du jeu : https://www.matagot.com/docs/Takenoko_rules_EN.pdf
Contenu matériel du jeu : http://jeuxstrategieter.free.fr/Takenoko_complet.php