Rapport-diceforge-teamC

- **Synthèse du projet :**

Le jeu est complet, il présente toute les fonctionnalités et possibilités du vrai jeu de plateau.
Les cartes basique et avancée sont disponibles, pour chaque carte de chaque île, une des deux versions est tirée au hasard de façon équiprobable pour chaque partie. Le programme effectue bien 1000 parties entre notre meilleur bot et notre deuxième meilleur bot, puis un match miroir du même nombre de partie avec notre meilleur bot. Chaque bot commence premier exactement 500 fois par match pour éviter les problèmes liés à l'avantage de la position. Les bots disponibles sont , dans l'ordre d'efficacité : RandomBot, ZizaBot, EasyBot, NidoBot, NidoBotV2, AubotV2.

RandomBot :
A chaque décision, il regarde parmi toutes les décisions possibles et en choisis une de façon aléatoire et équiprobable (ce qui inclus une chance de 1/3 de passer son tour).

ZizaBot :
Même stratégie que EasyBot mais avec des choix de paramètre différents et des modifications dans certains utilisation de carte.

EasyBot :
Forge pendant les premiers 5 premiers tours en faisant attention à ce qu'il ai suffisamment d'or pour que ce soit une action rentable. Priorise l'achat de face or en début de opartie, sinon achète les faces les plus chères dans ses moyens. Il achète la carte la plus chère en ressource qui est dans ses moyens quand il ne forge pas. Rejoue s'il possède plus d'un soleil ou plus d'une lune.

NidoBot :
Même schéma que EasyBot mais avec des tours de forge choisis précisément. A une liste de carte prioritaire qu'il suit lors d'un achat.

NidoBotV2 :
Même chose que NidoBot mais avec de meilleurs choix de paramètres, et une meilleure gestion des pouvoirs des cartes.

AubronBotV2 :
Bot qui s'améliore grâce a un algorithme génétique, une grosse partie de ses méthodes sont codées en dur (tout les choix qui paraissent évident pour un humain), comme la gestion des effets des cartes et des face. Mais les choix moins intuitifs comme le type d'action à effectuer, la face ou la carte à acheter en priorité est décidé par la partie génétique.Voir le commentaire de la classe pour en savoir plus.

Malgré un arsenal de bot fourni et un jeu complet, le projet n'est pas tout à fait parfait. La fiabilité n'est pas certaine a 100 % vis à vis de certaines interactions rare et complexe telles qu'une combinaison de face X3 et voile celeste (par exemple), elles semblent fonctionner mais la structure du code ne nous permet pas de faire des test convaincants pour les tester. Cependant les interactions et effet simple (ce qui constitue la base du jeu) sont testés et fiables.

- **Points fort du projet :**

Notre plus gros atout est que notre version du Dice Forge a été aboutie rapidement et cela nous a donc permis de tester extensivement notre code par le biais de longue heure de développement de bot.

Nous avons ainsi aussi pu développer des bots très performants dont le meilleur atteint 126 points de gloire en moyenne, utilisant un algorithme partiellement génétique.
Nous avons aussi pris soin de d'avoir une version fonctionnelle avec des bots jouant sur tout le jeu disponible à chaque démo, nous ne nous somme pas engagé avec trop d'empressement sur le développement du jeu : nous avons toujours cherché à le laisser ouvert à l'extension en pensant aux implémentations futures.

Nous avons pris le temps de penser au code, comme par exemple pour créer l'interface des bots à travers une classe abstraite,qui permet au développeur de celui si d'avoir accès à un maximum d'information tout en aillant un minimum de code à écrire.

- **Points faibles du projet :**

Il est vrai que sur certains points on peut dire que notre projet est effectivement mauvais. Par exemple, nous avons assez mal utilisé les outils mis à notre disposition avec le gitHub, pour ce qui est de la gestion de projets telles que les créations d'issues mais surtout lier les issues avec nos commit, chose que nous avons faite assez tard dans la réalisation du projet.

Cela pose surtout problème lorsqu'il s'agit d'organiser ou de se repérer dans l'organisation du projet. En effet, on ne sait pas vraiment ce qui nous attend dans le futur ce qui pourrait être dramatique dans un projet de plus grande envergure.

Sinon nous avons tendance à oublier de tester nos méthodes directement après les avoir créées ce qui devrait être un réflexe à avoir. Et pour finir nous avons eu un problème de répartition du travail très évident. Nous n'avons pas défini de tâches à chacun c'est donc les plus motivés qui on fait la majorité du travail. Cependant nous avons essayé de corriger le tir sur la fin du projet en aidant les plus en retard et travaillant plus en commun.

- **Points à améliorer dans le futur :**

S'obliger à écrire des tests pour chaque fonctions non trivialles immédiatement après l'avoir faite, ce qui implique :

- Designer les fonctions pour qu'elles soient plus "test-friendly" ce qui souvent implique faire du code plus orienté objet

- Designer les fonctions intelligemment, pour éviter d'avoir à changer les paramètres plus tard et de devoir refaire les tests, ce qui implique faire du code plus maintenable et plus propre.

- Éventuellement utiliser des logiciels de modélisation UML afin de planifier l'ensemble du projet en amont.

- Affecter par écrit des tâches à chaque membre du groupe, que ce soit par le biais d'issues github ou autres moyens (tickets, etc.).

- Nécessité de bien partager les tâches entre les différents membres du groupe et d'équilibrer la charge de travail sur chacun de ses membres.

- Obligation de lier chaque commit à une issue.

Et surtout mieux gérer, planifier et entretenir l'outil de gestion des versions (github) pour que chaque membre du groupe, mais aussi les personnes extérieurs au groupe puisse connaître à tout instant l'avancement du projet, mais aussi pour pouvoir faire des prévisions quand au déroulement futur du projet.