PAQUETTE, NICOLAS

RAPPORT DE PROJET
Chess GamByte

Travail présenté à
Jean-Christophe Demers

Dans le cadre du cours
420-C61-IN

Techniques de l'informatique
Cégep du Vieux Montréal
25-05-2021

Présentation Générale

Chess GamByte est une plateforme de jeu d'échec développée en python. Il est possible de jouer contre un autre humain ou contre l'intelligence artificielle du programme. Il est possible de choisir la couleur de ses pièces ou de choisir de manière aléatoire. Tous les coups légaux et spéciaux sont intégrés dans le jeu. Tous les coups joués durant la partie s'ajouteront à la liste de coups à droite de l'échiquier. Il est possible de visualiser toutes les positions atteintes au cours de la partie en naviguant parmi cette liste. Une fois la partie terminée, les informations de partie sont sauvegardées localement dans une base de données. Toutes les positions des parties enregistrées peuvent être visionnées en parcourant l'historique des parties jouées, tout en ayant accès aux informations tels que la date, la couleur du gagnant et le nombre de coups total.

Résumé du développement

Le développement du programme s'est relativement bien déroulé. La partie la plus cruciale du projet était tout ce qui était en rapport avec le sprint 1. C'est donc dans cette période du travail que les plus grandes embuches sont survenues. C'est le sprint 1 qui a donc pris la plus grande partie du temps de développement si l'on compare aux 2 autres sprints suivant. Il y a eu beaucoup d'essais-erreurs : le développement d'un jeu d'échec semble simple dans un contexte où l'entièreté du jeu se déroule dans un échiquier de 8 par 8. Cependant, il y a beaucoup de vérifications nécessaires et d'interactions entre les différentes pièces du jeu. Un autre problème est survenu lors de l'élaboration de l'algorithme de type minimax pour la recherche d'un coup optimal. Étant un algorithme récursif s'exécutant des centaines ou milliers de fois par tour, il est difficile de cerner où se trouve une erreur spécifique lorsqu'un problème survient. Cela m'a causé des ennuis pendant quelques jours lorsqu'une des pièces disparaissait sans raison de l'échiquier lors de l'évaluation de toutes les possibilités de jeu. Outre des ennuis techniques, la session s'est bien déroulé. Chaque semaine apportait des fonctionnalités supplémentaires et le sentiment de progression d'un projet de plus grande envergure est très satisfaisant.

Fonctionnalités

Presque tout ce qui a été envisagé au début de la session a été réalisé au fil de la session. Tout ce qui a été intégré au projet est entièrement fonctionnel et réalisé dans son entièreté. Commençons par les 2 éléments qui n'ont pas été réalisés :

- Bar de visualisation du score : la valeur du score est déjà affichée et l'écran est déjà assez saturé d'information.
- Tables de transposition : préférence à passer plus de temps à améliorer l'évaluation d'une position plutôt que la vitesse totale d'exécution.

Au niveau des fonctionnalités complétées :

- Tous les coups légaux et coups spéciaux (en passant, *castling*, promotion des pions)
- Possibilité de jouer humain contre humain ou humain contre ordinateur
- Implémentation de l'algorithme minimax
- Évaluation des positions de jeu : échanges favorables, structure des pions, sécurité du roi, vérification des échecs et mat
- Visualisation des coups effectués durant la partie
- Sauvegarde des informations de jeu et visualisation des anciennes parties
- Possibilité pour le AI de choisir aléatoirement une ouverture de jeu connue

Améliorations possibles

Malgré toutes les fonctionnalités implémentées qui rendent l'intelligence artificielle de faire des décisions éclairées selon les situations présentées, il faut pouvoir atteindre un certain niveau de profondeur de recherche dans l'algorithme minimax afin que tout ce travail porte fruit adéquatement. Présentement, pour avoir un temps d'exécution par tour entre 2 et 10 secondes, le niveau de profondeur de recherche doit être de 2. On ne regarde donc que 2 coups dans le futur, rendant l'algorithme aveugle dans une panoplie de contexte où l'humain peut facilement contextualiser une tactique plus performante. Même si l'évaluation d'une position est excellente, si on ne peut pas voir les ramifications dans un futur moyennement rapproché, les coups choisis perdent extrêmement de pertinence. Si cela était à refaire, je choisirais un langage compilé, comme

le C++, permettant d'utiliser efficacement la puissance computationnelle de mon ordinateur pour calculer le plus de positions possibles en un laps de temps raisonnable. De plus, l'utilisation d'objets lourds et complexe (un objet échiquier contient 64 objets carrés, qui contiennent à leur tour potentiellement un objet pièce) permet de représenter instinctivement le problème au sens large mais l'utilisation constante de structures de ce genre n'aide pas à réduire le temps d'exécution des vérifications de positions, qui est le problème flagrant de l'algorithme.

Auto-évaluation

Je suis satisfait du travail accompli au cours de la session. Je crois que ce projet est une combinaison d'un grand nombre de concepts que nous avons appris au fil de la technique. Beaucoup d'éléments sont interconnectées les uns aux autres formant un tout complexe ayant des éléments techniques intéressant. La grande utilisation de l'orienté-objet, les *designs patterns* qui aident à la généricité de certaines composantes et l'élaboration d'algorithmes de calculs plus approfondis participent tous à bien représenter l'ensemble des acquis au cours du programme. En laissant de côté la vitesse d'exécution qui peut être attribué en partie au langage Python, je crois que mon projet représente bien mes capacités actuelles et je suis agréablement surpris d'avoir réussi en si peut de temps, que ce soit le fait d'effectuer ce travail en quelques mois ou le fait d'avoir commencé la programmation il y a moins de 1 ans et demi, un travail qui réalise des tâches complexes et tangibles. Une note de 90% d'auto-évaluation semble représenter l'étendue des réalisations lors de ce travail.