2016

Rapport du projet d'IA02

IMPLEMENTATION PROLOG DU JEU DU KHAN

MAXIME CHURIN, ROBIN BRIA

Table des matières

Introduction

Au cours de l'unité de valeur IA02, il nous a été donné de d'implémenter entièrement un jeu de stratégie appelé jeu du Khan. Ce jeu oppose deux joueurs composés de 5 pions classiques + 1 pion Kalista chacun sur un plateau carrée 6x6. Le but étant de prendre en respectant un ensemble de règles la Kalista adverse. Ce projet a pour objectif de mettre à l'épreuve nos connaissances sur la logique et prolog pour développer les bases du jeu mais aussi l'IA en cas de partie contre une machine.

Descriptions des structures de données

Avant-propos :

Avant de nous lancer dans le code du projet, nous avons longuement discuter sur la façon dont nous souhaitions l'aborder. En effet, plusieurs choix s'offraient à nous : implémenter le jeu comme une matrice ou chaque case est elle-même une liste contenant la valeur de la case et éventuellement un pion, une matrice pour le plateau et deux listes contenant les pions de chaque joueur et enfin deux matrices une pour le plateau de jeu et une pour le plateau de pion. Après avoir réfléchi, pour chaque architecture, l'ensemble des règles, notre choix s'est porté sur une architecture avec deux matrices. Une de nos principales motivations a été pensé que travailler sur deux objets identiques dans une logique d'accès séquentielle ou itératif factoriserait les prédicats. Nous ne regrettons pas notre choix car nous n'avons, au cours du projet, jamais été bloqué par notre structure de donnée. Nous avons aussi beaucoup utilisé les assert et retract pour nos variables globes comme les plateaux, les kalistas, le khan…

Explication :

Nous avons donc codé 4 matrices symbolisant le plateau pour les différentes orientations, puis lors de la saisie des pions des joueurs nous remplissons la matrice de pion. Celle-ci sera donc modifier à chaque déplacement. Le joueur rouge possède des pions codés avec la valeur 1 et le joueur ocre avec la valeur 2. Chaque kalista étant sauvegardé dynamiquement nous n'avons pas besoin de créer une valeur spécifique. Le khan, quant à lui, varie entre 0 et 3, 0 pour l'initialisation mais aussi lorsque qu'aucun pion ne repose sur la valeur du khan courant. En ce sens nous avons écrit deux prédicats pour modifier la valeur du khan et des kalistas (modificationKhalista et modificationKhan). L'ensemble de notre jeu est disponible par le prédicat main qui initialise les variables globales, place les pions et lance la boucle de déplacement. De plus ample détail sur nos prédicats sont données dans la partie suivante.

Présentation des prédicats

Prédicat de base :

Pour aborder le problème, les étapes listées dans le sujet nous ont bien aidé à progresser efficacement. Nous avons donc commencé par écrire des prédicats pour afficher notre matrice (afficheMatrice), puis nos deux matrices conjointement (afficheBoard) et enfin des prédicats permettant l'accès (valeurCase) ou l'insertion (modificationMatrice) à une case spécifique de la matrice.

Placement des pions :

Notre modèle de base étant posé nous sommes passés sur le placement des pions d'un joueur (lireJoueur). L'utilisateur saisit au fur et à mesure la ligne puis la colonne où il désire placer son pion. Nous avons ensuite implémenté un système de retour utilisateur très complet qui lui permet d'avoir un message d'erreur approprié en fonction d'une case déjà occupée, un numéro de ligne ou de colonne inapproprié ... Notre convention de placement des joueurs impose qu'un utilisateur entre en dernier sa kalista.

Liste des mouvements possibles :

Cette étape a été la première grosse épreuve de notre projet. Pour des raisons de transparence avec le joueur mais aussi pour lui faciliter un visuel en temps réel des coups possible nous avons décidé de casser ce prédicat en deux parties. Nous générons tous d'abord une liste contenant l'ensemble des pions déplaçables et le joueur sélectionne son numéro dans la dite liste (affichageChoix et nbElement). On renvoie alors la liste des cases accessibles par ce pion, encore une fois sous forme de liste avec saisie du numéro par le joueur. Nous évitons ainsi à nos utilisateurs de devoir saisir la ligne et la colonne de départ et d'arrivée, qui s'avère être une opération redondante et ennuyante. Ce choix a notamment été motivé par la gestion future de l'IA qui récupère ainsi deux listes contenant l'ensemble des possibilités.

Notre code va donc dans un premier temps parcourir le plateau de pion pour lister les pions déplaçables selon la valeur du khan, les bords du plateau et la possibilité d'être bloqué par d'autre pion (possibleMoves). Si une quelconque case d'arrivée est possible pour le pion il est alors insérer dans une variable globale que nous afficherons plus tard (checkDeplacable). Une fois cette étape validé nous récupérons la ligne et la colonne de ce pion et nous cherchons la liste complète de ces cases d'arrivées possibles (accessibleMoves). Pour cela, en fonction de la valeur du khan nous appelons un prédicat qui va tester un mouvement droite, gauche, haut et bas en sauvegardant un historique du chemin parcouru et appeler ce même prédicat en décrémentant le khan qui s'apparente au nombre de case parcouru(checkAccessible). Nous vérifions aussi que la case suivante n'appartient pas au chemin déjà emprunté (retour en arrière interdit : checkMoveInMatrice). Une fois arrivée à une valeur de khan égale à 1 et les conditions respectées nous enregistrons la case comme case d'arrivée possible.

Nous avons ainsi réglé l'ensemble des déplacements possibles. De nouveau un système complet de retour utilisateur est présent pour accompagner le joueur et l'avertir d'une mauvaise saisie.

Boucle de tour de jeu :

Pour cette nouvelle étape il a seulement fallut rassembler nos précédents prédicats, créer des dialogues pertinents pour guider le joueur et vérifier la condition de fin de jeu avant de passer la main à son adversaire et répéter la boucle (deplacement et deplacementJoueur). Nous avons aussi ajouté, pour respecter les règles, le cas où le joueur à le choix de remettre un pion en jeu ou de déplacer celui qu'il souhaite (autreMove et ajoutPion). Pour la condition de fin de jeu nous vérifions que la coordonnée de la kalista d'un joueur comprend bien un pion de de son type grâce à notre variable globale (finDeJeu).

Description de l'IA

Nous avons conçu une IA avec un comportement plutôt agressive. En effet, l'objectif premier de notre IA est de manger la kalista adverse, dans cette optique le meilleur déplacement est donc de se rapprocher le plus proche possible de la kalista adverse. Pour ce faire, nous avons mis en place un système de score pour chaque mouvement. Ce score est donné selon la formule suivante :

Tmp =|LigneKalista-LigneNouvellePosition|+|LigneKalista-LigneDeLaNouvellePosition|

Score = Tmp + Malus.

Le malus est égal à 3 si la valeur de la case d'arrivée est différente Tmp, sinon le malus est égal à 0. Le calcul de Tmp favorise le déplacement qui rapproche un pion de la kalista adverse. Le malus, quant à lui, permet de pondérer ce déplacement en fonction de la valeur du Khan. Il est plus intéressant d'arrivée sur une case qui permettra de manger la kalista au tour suivant. En effet, si Tmp vaut trois et la valeur de la case d'arrivée aussi, au prochain tour du jour, si le khan vaut trois ou zéro (en cas de blocage), le prédicat (mangeableKhalista) permettra au pion de prendre la kalista et ainsi de terminer la partie. Notre stratégie ne visant qu'à manger la khalista adverse, il nous a semblé intéressant de ne pas remettre en jeu de pions, afin de permettre un blocage plus fréquent des pièces de l'IA. Ceci dans le but de permettre un déplacement plus libre des pions restant afin de manger la kalista adverse.

Cependant la Kalista étant considéré comme un pion elle possède le même comportement aggressif ce qui conduit parfois à son suicide, et plus particulièrement en cas de blocage. Il aurait fallu pondérer ce calcul en rajoutant un malus si la kalista est prenable par l'adversaire au tour suivant.

Difficultés rencontrées et amélioration possibles

Nous avons longuement hésité sur le choix de notre structure de données sans vraiment trouver un argument qui permettait d'en détacher une des autres. Notre structure est donc possiblement améliorable même si nous n'y avons pas trouvé de solution

La génération des mouvements possibles nous a aussi demandé beaucoup de réflexion. Au départ nous étions partis naïvement sur le test de toutes les combinaisons de déplacements possible que nous avons peu à peu améliorer, notamment par une matrice qui sauvegarder le chemin parcouru.

Enfin pour la gestion de l'IA nous avons commencé par listé les meilleurs coups possibles :

- 1er : Prendre la Kalista adverse
- 2eme : Sauver sa Kalista menacer (en deplacant la Kalista sur une case non menacée ou sinon en prenant le pion nous menancant)
- 3eme : Bouger sur une case avec une valeur de Khan présent parmi les pions déplaçables de l'adversaire
- 4eme : Se rapprocher le plus possible de la Kalista adverse

Mais au moment d'implémenter cette IA nous nous nous sommes confrontés à de lourde condition et un manque de temps. Cette partie nous intéressait tout particulièrement mais nous n'en avons réalisé qu'une petite partie (à asavoir la 1ere et la 4eme option). L'IA est donc la principale source d'amélioration dans notre projet.

Conclusion

Ce fut un projet très intéressant qui nous a permis de découvrir un jeu qui nous était inconnu et de mettre à profit nos connaissances pour le développer. Nous avons au final obtenu un jeu avec une interface agréable et des dialogues efficaces pour gérer la partie. C'est toujours très valorisant de produire un contenu utilisable et fonctionnelle et c'est le cas de ce projet. Notre principal regret est que malheureusement ce n'était pas notre seul projet de ce semestre, et nous n'avons donc pas été aussi loin que nous le souhaitions notamment dans la gestion de l'IA et l'optimisation du code