Recherche arborescente de Monte-Carlo sur un jeu de dés inspiré d'Orlog [1]

Le but de ce projet est d'implémenter une méthode de recherche de Monte-Carlo pour jouer à une version inspirée et simplifiée du jeu de dés Orlog d'assassin's creed Valhalla.

1 Présentation du problème

L'objectif de ce jeu à deux joueurs est d'être le premier à faire perdre les 15 jetons de vie de son adversaire. Pour cela, il faut attaquer et se défendre des attaques de son adversaire.

Chaque joueur dispose 6 dés. Chaque dé possède 2 faces avec une hache, 2 faces avec une flèche, une face avec un casque et une face avec un bouclier. Un casque annule une attaque de hache et un bouclier annule une attaque de flèche.

Au début du jeu, déterminer aléatoirement le premier joueur.
Chaque manche est constituée au plus de six lancers (trois pour chaque joueur). Chaque joueur lance à tour de rôle ses dés. Après chacun de vos lancers, vous pouvez choisir quels résultats vous désirez garder et quels dés vous voulez relancer après le prochain lancer du joueur adverse. Vous pouvez voir les dés conservés par votre adversaire après chaque lancer. Vous pouvez effectuer jusqu'à trois lancers.

Le nombre de jetons supprimés à l'adversaire correspond aux nombres d'attaques non contrées.
Les manches s'enchaînent jusqu'à ce que la santé d'un des joueurs soit réduite à zéro.

2 Travail demandé

Vous devez implémenter efficacement la rechercher arborescente de Monte-Carlo sur le jeu de dés présenté ci-dessus.

2.1 Modélisation du jeu

Choisissez une représentation de l'état du jeu.

Une fonction d'évaluation est généralement définie pour appliquer un algorithme de type alpha-bêta. Lorsque cette fonction est difficile à définir et/ou que le temps

1. `https://www.youtube.com/watch?v=0tyqH6mLPZ0`

de recherche de l'algorithme alpha-beta est important, l'évaluation des états atteignables en un coup, laquelle permet de choisir le prochain coup à jouer, peut être faite à l'aide de la recherche de Monte-Carlo. Cette approche génère un enchaînement aléatoire de coups jusqu'à obtenir une fin de partie et retient le résultat (par exemple, Gain=1, Nul=0, Perte=-1). L'évaluation de l'état correspond à la moyenne des résultats. En appliquant cette évaluation à chaque état atteignable par un choix de conservation au regard de celui de votre adversaire, il devient possible de sélectionner le prochain coup, lequel correspond à l'état ayant l'évaluation la plus forte.

Ici, il est uniquement nécessite de sauvegarder l'état courant généré par la recherche et de tirer au hasard le coup suivant.

Un nombre d'évaluations ou une limite de temps par action possible peut être fixé au départ .

2.2 Réalisation

Le choix du langage de programmation est libre.

Paramètre modifiable dans l'interface :
— Le nombre d'enchaînements de coups à réaliser pour évaluer l'intérêt d'un état.

Informations à afficher :
— la profondeur moyenne trouvée pour une fin de partie ;
— la valeur de l'évaluation des coups immédiatement possibles ;
— toute autre information jugée utile.

2.3 Présentation

Les soutenances auront lieu les 11 et 12 mai 2023 en fonction de vos disponibilités. Une présentation d'environ cinq minutes suivie d'une discussion d'une dizaine de minutes vous permettront d'une part d'exposer vos résultats en précisant leurs points forts, leurs points faibles, et les améliorations possibles et d'autre part d'effectuer une démonstration de l'implantation de cette stratégie. Le support de la présentation et les sources doivent être déposés sur le site arche dédié au module Intelligence Artificielle au plus tard le mercredi 10 mai 2023. Merci de nommer votre archive avec le nom du projet et les vôtres (ex. des-Nom1-Nom2-Nom3.zip).

Pour aller plus loin

L'évolution des états du jeu peut être modélisé à l'aide d'un graphe. Chaque nœud correspond à un état du jeu et chaque arc à une action possible.

Vous pouvez vous inspirer des éléments suivants (d'après le livre "Artificial Intelligence for Games", Ian Millington, John Funge, 2nd Edition, Morgan Kaufmann, 2009) :

```
class Board:
  def getMoves()
  def makeMove(move)
  def evaluate()
  def currentPlayer()
  def isGameOver()
```