ITI 1521. Introduction à l'informatique II
Hiver 2020

Devoir 4
(Dernière modification le 10 mars 2020)

Échéance: 5 avril, 2020, 23 h 30

Objectifs d'apprentissage

- **Employer** l'héritage et les classes abstraites
- **Expérimenter** avec l'apprentissage machine

Introduction

Nous avons créé la base de notre implémentation de MENACE, un système automatisé qui apprend à jouer au Tic-Tac-Toe. Dans ce devoir, nous utiliserons notre solution du devoir 3, ainsi, vous devez donc d'abord la compléter. Ensuite nous l'utiliserons pour créer notre propre implémentation de la machine proposée dans l'article de Donald Michie de 1961.

1 MENACE

Nous sommes maintenant prêts à implémenter **MENACE**. Si vous ne l'avez pas encore fait, vous devriez vraiment lire l'article publié par Donald Michie en 1961 dans *Science Survey*, intitulé *Trial and error*. Cet article a été réimprimé dans le livre *On Machine Intelligence* et se trouve à la page 11 sur ce site : https://www.gwern.net/docs/ai/1986-michie-onmachineintelligence.pdf. Vous pouvez également regarder https://www.youtube.com/watch?v=R9c-_neaxeU.

Dans ce qui suit, nous ne traiterons que des jeux 3×3, car c'est ce qui a été défini dans le document. Vous n'avez pas à vous soucier des autres tailles de jeux, seulement 3×3.

Nous vous avons donné tout ce qui est nécessaire pour cette partie, vous n'avez donc qu'à vous concentrer sur la réalisation de **ComputerMenacePlayer** et **MenaceTicTacToeGame**. Vous devez cependant utiliser votre solution fonctionnelle du devoir 3.

Dans notre solution, nous avons notre joueur MENACE, qui joue contre une série de joueurs possibles : un humain, un joueur aléatoire, un joueur parfait ou un autre joueur MENACE.

Le joueur humain et le joueur aléatoire ont déjà été implémentés dans les devoirs précédents. Nous fournissons une implémentation d'un joueur parfait. Vous n'avez pas à comprendre comment il fonctionne précisément (mais bien sûr, vous le pouvez !) mais vous devez absolument jeter un coup d'œil au code car cela vous aidera beaucoup pour l'implémentation de MENACE.

Nous avons apporté quelques modifications à la conception : tout d'abord, nous souhaitons que nos joueurs partagent des méthodes supplémentaires. Auparavant, il suffisait qu'un joueur donne une implémentation concrète de la méthode **play**. Maintenant, nous voulons pouvoir informer le joueur qu'un nouveau jeu commence et qu'un jeu est terminé. Nous voulons que le joueur conserve quelques statistiques sur ses performances : combien de fois il a gagné et perdu au total, ainsi qu'au cours des 50 dernières parties (pour suivre ses progrès). La mise en œuvre de certaines de ces méthodes est commune à tous les joueurs, c'est pourquoi nous souhaitons ajouter le code directement dans **Player**. Cependant, **Player** est une interface nous empêchant de le faire. Nous l'avons donc transformé en une classe à part entière. Nous ne pouvons toujours pas fournir une implémentation par défaut pour la méthode **play**, de sorte que cette méthode est toujours **abstract**, donc **Player** est maintenant une **classe abstraite**. Nous avons fourni l'implémentation pour toutes les autres méthodes de la classe **Player**.

Deuxièmement, certains de nos joueurs auront besoin d'une version spécialisée de **TicTacToeGame**. Par exemple, notre joueur idéal doit spécialiser les instances de **TicTacToeGame** pour enregistrer les coups possibles qui sont gagnants et ceux qui sont perdants. Nous avons donc créé la classe **PerfectTacToeGame**, une sous-classe de **TicTacToeGame** que **ComputerPerfectPlayer** utilise pour enregistrer les informations supplémentaires. Le fonctionnement de la classe est le suivant : lorsqu'une instance de **ComputerPerfectPlayer** est créée, elle crée d'abord la liste de tous les jeux possibles, comme nous l'avons fait pour le devoir 3. La liste est cependant constituée d'instances de **PerfectTacToeGame** au lieu d'instances de **TicTacToeGame** comme nous l'avions fait pour le devoir 3. Une fois la liste créée, l'instance **ComputerPerfectPlayer** précalcule tous les coups de toutes les parties possibles pour déterminer ceux qui doivent être joués et ceux qui doivent être évités. Elle est alors prête à jouer les parties. Lorsque sa méthode **play** est appelée, elle reçoit une instance de **TicTacToeGame** comme paramètre, qui est l'état actuel de la partie. Il consulte sa liste de toutes les parties précalculées pour trouver celle qui correspond à l'état actuel de la partie (jusqu'à la symétrie), puis sélectionne à partir de là l'un des meilleurs coups possibles, tel que précalculé lors de l'initialisation. **ComputerMenacePlayer** fonctionnera de la même manière.

Notez que la construction de tous les jeux possibles utilise presque le même code que la méthode **generateAllUniqueGames** de **ListOfGamesGenerator** du devoir 3. La seule différence est que **TicTacToeGame** est remplacé par **PerfectTicTacToeGame**. Nous avons recopié le code dans le constructeur de **PerfectTacTacToeGame** au lieu de réutiliser la méthode précédente pour simplifier le problème. Vous pouvez faire la même chose pour votre implémentation de **ComputerMenacePlayer**.

L'essentiel de MENACE est qu'il pré-calcule toutes les parties possibles (tenant compte des symétries), et pour chaque partie, il fournit initialement un certain nombre de billes pour chaque coup possible. Lorsqu'il joue une partie, MENACE trouve la partie correspondant à l'état actuel et choisit au hasard une des billes qu'il possède pour cette partie. Plus un coup donné possède de billes à ce stade, plus il a de chances d'être sélectionné. Une fois la partie terminée, MENACE met à jour le nombre de billes pour chacun des coups utilisés au cours de la partie, en fonction du résultat : si la partie a été perdue, les billes qui ont été sélectionnées seront retirées, ce qui rendra moins probable la sélection de coups similaires dans le futur. Cette approche pose un problème : il est possible de retirer la dernière bille d'une partie de cette manière, auquel cas la menace sera bloquée si la partie est à nouveau jouée dans le futur, sans qu'aucun coup n'ait la moindre chance d'être sélectionné. Pour éviter cela, nous n'enlevons une bille que si elle n'est pas la dernière de la partie. Si le jeu est nul, la bille est remise en place et une autre bille similaire est ajoutée à chaque fois, augmentant un peu plus la chance de sélectionner ce coup dans le futur. Si la partie est gagnée, la bille est remise en place et trois autres billes similaires sont ajoutées.

On vous demande de fournir une implémentation de **ComputerMenacePlayer** qui se comporte comme prévu. Vous devez également implémenter **MenaceTicTacToeGame**, une sous-classe de **TicTacToeGame**, que les instances de **ComputerMenacePlayer** utilisent pour pré-calculer toutes les parties possibles et enregistrer le nombre actuel de billes pour chaque coup possible de chaque partie possible.

Dans le document, on suppose que MENACE joue toujours le premier coup (et nos expériences suggèrent en fait que MENACE apprend beaucoup mieux à être un premier joueur qu'un deuxième joueur). L'instance principale de MENACE dans notre système est également réglée pour toujours jouer en premier (bien que vous puissiez facilement changer cela dans le code), mais puisque nous pouvons jouer MENACE contre MENACE, nous devons avoir une implémentation qui peut jouer à la fois X et O. Le document précise le nombre initial de perles pour X seulement. Nous utiliserons des nombres légèrement différents pour X (8,4,2,1 au lieu de 4,3,2,1) et nous utiliserons des nombres similaires pour O. En d'autres termes, le nombre initial de perles est

- First move (X) : 8 beads
- Second move (O) : 8 beads
- Third move (X) : 4 beads
- Fourth move (O) : 4 beads
- Fifth move (X) : 2 beads
- Sixth move (O) : 2 beads
- Seventh move (X) : 1 bead
- Eighth move (O) : 1 bead
- Ninth move (X) : 1 bead

Le fonctionnement du système que nous fournissons est le suivant : dans un premier temps, une nouvelle instance de MENACE est créée. Cette instance n'a pas été entraînée et est donc très faible. Vous pouvez choisir l'adversaire, qui est soit un ordinateur (aléatoire, parfait ou MENACE), soit un humain (vous). Si vous choisissez un adversaire contrôlé par l'ordinateur, le système lancera automatiquement 500 parties entre MENACE et cet adversaire, au cours desquelles MENACE devrait améliorer son jeu (surtout la première fois) et devrait ensuite être beaucoup plus difficile à battre. Nous avons également inclus une option pour réinitialiser MENACE, afin que vous puissiez facilement recommencer une étape d'apprentissage à partir de zéro.

Voici un exemple de fonctionnement du système. Nous allons d'abord effectuer deux tours contre un humain :

```
> java TicTacToe
************************************************************
*                                                          *
*                                                          *
*                                                          *
*                                                          *
************************************************************

(1) Human to play menace
(2) Train Menace against perfect player
(3) Train Menace against random player
(4) Train Menace against another menace
(5) Delete (both) Menace training sets
(Q)uit
1
   |   |
-----------
   | X |
-----------
   |   |

0 to play: 1
 0 | X |
-----------
   | X |
-----------
   |   |

0 to play: 8
 0 | X |
-----------
   | X | X
-----------
   | 0 |

0 to play: 4
 0 | X |
-----------
 0 | X | X
-----------
   | 0 | X

0 to play: 7
 0 | X |
-----------
 0 | X | X
-----------
 0 | 0 | X
```

```
Result: OWIN
This player has won 0 games, lost 1 games and 0 were draws.
(1) Human to play menace
(2) Train Menace against perfect player
(3) Train Menace against random player
(4) Train Menace against another menace
(5) Delete (both) Menace training sets
(Q)uit
1
 X |   |
-----------
   |   |
-----------
   |   |

O to play: 5
 X | X |
-----------
   | O |
-----------
   |   |

O to play: 3
 X | X | O
-----------
   | O |
-----------
 X |   |

O to play: 4
 X | X | O
-----------
 O | O |
-----------
 X | X |

O to play: 6
 X | X | O
-----------
 O | O | O
-----------
 X | X |

Result: OWIN
This player has won 0 games, lost 2 games and 0 were draws.
(1) Human to play menace
(2) Train Menace against perfect player
(3) Train Menace against random player
(4) Train Menace against another menace
(5) Delete (both) Menace training sets
(Q)uit
```

Comme on peut le voir, jusqu'à présent, MENACE ne joue pas très bien et a perdu deux fois de suite bien qu'il ait été le premier joueur. Entraînons-le maintenant contre un joueur parfait :

```
This player has won 0 games, lost 2 games and 0 were draws.
(1) Human to play menace
```

```
(2) Train Menace against perfect player
(3) Train Menace against random player
(4) Train Menace against another menace
(5) Delete (both) Menace training sets
(Q)uit
2
player 1: This player has won 0 games, lost 36 games and 466 were draws. Over the last 50 games,
this player has won 0 games, lost 0 games and 50 were draws.
player 2: This player has won 34 games, lost 0 games and 466 were draws. Over the last 50 games,
this player has won 0 games, lost 0 games and 50 were draws.
```

MENACE (qui est ici le joueur 1) a perdu 34 matchs contre le joueur parfait sur les 500 qui ont été joués, et aucun au cours des 50 derniers matchs.

Essayons encore une fois contre un joueur humain :

```
(1) Human to play menace
(2) Train Menace against perfect player
(3) Train Menace against random player
(4) Train Menace against another menace
(5) Delete (both) Menace training sets
(Q)uit
1
   |   |
-----------
   | X |
-----------
   |   |

0 to play: 1
 0 |   | X
-----------
   | X |
-----------
   |   |

0 to play: 7
 0 |   | X
-----------
 X | X |
-----------
 0 |   |

0 to play: 6
 0 |   | X
-----------
 X | X | 0
-----------
 0 | X |

0 to play: 2
 0 | 0 | X
-----------
 X | X | 0
-----------
 0 | X | X
```

```
Result: DRAW
This player has won 0 games, lost 36 games and 467 were draws. Over the last 50 games,
this player has won 0 games, lost 0 games and 50 were draws.
```

Maintenant, MENACE est un bien meilleur joueur, et joue en effet très bien et a obtenu un match nul. Notez cependant qu'il n'est pas parfait et qu'il peut encore être battu. Regardez la prochaine partie :

```
(1) Human to play menace
(2) Train Menace against perfect player
(3) Train Menace against random player
(4) Train Menace against another menace
(5) Delete (both) Menace training sets
(Q)uit
1
   |   |
-----------
   | X |
-----------
   |   |

O to play: 2
   | O |
-----------
   | X |
-----------
 X |   |

O to play: 3
   | O | O
-----------
   | X |
-----------
 X |   | X

O to play: 1
 O | O | O
-----------
   | X |
-----------
 X |   | X

Result: OWIN
This player has won 0 games, lost 37 games and 467 were draws. Over the last 50 games,
this player has won 0 games, lost 1 games and 49 were draws.
(1) Human to play menace
(2) Train Menace against perfect player
(3) Train Menace against random player
(4) Train Menace against another menace
(5) Delete (both) Menace training sets
(Q)uit
q
>
```

Nous avons réussi à le battre à nouveau. Il y a dans ce cas une raison assez simple à cela : notre premier coup (humain) a été de jouer 2 sur une partie où l'on a d'abord joué 5. Si vous y réfléchissez bien, c'est en fait un très mauvais coup, une perte garantie contre un joueur compétent. Mais comme il s'agit d'un coup perdant, notre joueur parfait ne le jouera jamais, et donc MENACE n'a pas du tout été formé pour le gérer. Cela illustre certains des défis de l'apprentissage machine, mais c'est une toute autre discussion !

Intégrité académique

Cette partie du devoir vise à sensibiliser les étudiants au plagiat et à l'intégrité académique. Veuillez lire les documents suivants.

- https://www.uottawa.ca/administration-et-gouvernance/reglement-scolaire-14-autres-informations-impor
- https://www.uottawa.ca/vice-recteur-etudes/integrite-etudes

Les cas de plagiat seront traités conformément au règlement de l'université. En soumettant ce travail, vous reconnaissez :

1. J'ai lu le règlement académique sur la fraude académique.
2. Je comprends les conséquences du plagiat.
3. À l'exception du code source fourni par les instructeurs pour ce cours, tout le code source est le mien.
4. Je n'ai collaboré avec aucune autre personne, à l'exception de mon partenaire dans le cas d'un travail d'équipe.
    - Si vous avez collaboré avec d'autres personnes ou obtenu le code source sur le Web, veuillez alors indiquer le nom de vos collaborateurs ou la source de l'information, ainsi que la nature de la collaboration. Mettez ces informations dans le fichier README.txt soumis. Des points seront déduits proportionnellement au niveau de l'aide fournie (de 0 à 100%).

Directives

- Suivez toutes les directives disponibles sur la page Web Consignes pour la remise de tous vos travaux pratiques.
- Soumettez votre devoir par le biais du système de soumission en ligne campus virtuel.
- Vous devez préférablement réaliser le travail en équipe de deux, mais vous pouvez également le faire individuellement. Toutefois, si vous effectuez le travail en équipe, votre devoir ne doit être soumis qu'une seule fois sur Brightspace. Si les deux partenaires soumettent le devoir, une pénalité de 20 % pour le devoir 2, de 30 % pour le devoir 3 et de 40 % pour le devoir 4 sera appliquée.
- Vous devez utiliser les classes de modèles fournies ci-dessous.
- Si vous ne suivez pas les instructions, votre programme fera échouer les tests automatisés et, par conséquent, votre devoir ne sera pas noté.
- Nous utiliserons un outil automatisé pour comparer toutes les travaux les uns par rapport aux autres (y compris les sections française et anglaise). Les soumissions qui sont signalées par cet outil recevront la note 0.
- Il vous incombe de vous assurer que BrightSpace a bien reçu votre travail. Les soumissions tardives ne seront pas notées.

Fichiers

Vous devez soumettre un fichier **zip** (aucun autre format de fichier ne sera accepté). Le nom du répertoire principal doit avoir le format suivant : **a4_3000000_3000001**, où 3000000 et 3000001 sont les numéros d'étudiant des membres de l'équipe qui soumettent le travail (il suffit de répéter le même numéro si votre équipe compte un seul membre). Le nom du dossier commence par la lettre "a" (minuscule), suivie du numéro du devoir, ici 4. Les parties sont séparées par le trait de soulignement (et non par le trait d'union). Il n'y a pas d'espace dans le nom du dossier. L'archive a4_3000000_3000001.zip contient les fichiers que vous pouvez utiliser comme point de départ. Votre soumission doit contenir les fichiers suivants.

- README.txt
    - Un fichier texte qui contient les noms des deux partenaires pour les devoirs, leurs numéros d'étudiant, la section et une courte description du devoir (une ou deux lignes).
- CellValue.java

- ComputerMenacePlayer.java
- ComputerPerfectPlayer.java
- ComputerRandomPlayer.java
- GameState.java
- MenaceTicTacToeGame.java
- HumanPlayer.java
- Player.java
- PerfectTicTacToeGame.java
- StudentInfo.java
- TicTacToe.java
- TicTacToeGame.java
- Transformation.java
- Utils.java

Questions

Pour toutes vos questions, veuillez consulter le site web de la Piazza du cours :

- https://piazza.com/uottawa.ca/winter2020/iti1521/home

Resources

- https://images.math.cnrs.fr/Une-machine-en-boites-d-allumettes-qui-apprend-a-jouer-au-Morpion.html, visitée le 9 mars 2020.

Dernière modification : 10 mars 2020