ITI 1121. Introduction to Computing II
Winter 2020

Assignment 4
(Last modified on March 8, 2020)

Deadline: April 5, 2020, 11:30 pm

Learning objectives

- Inheritance and Abstract Classes
- Introduction to machine learning

Introduction

We have created the basis for our implementation of MENACE, an automated system learning how to play Tic-Tac-Toe. In this assignment, we will use the solution of assignment 3, you must thus complete it first. We will use our previous work to create our own implementation of Donald Michie's 1961 paper.

MENACE

We are now ready to implement **Menace**. If you have not already done so, you should really read the paper published by Donald Michie in 1961 in *Science Survey*, titled *Trial and error*. That paper has been reprinted in the book *On Machine Intelligence* and can be found on page 11 at the following URL: https://www.gwern.net/docs/ai/1986-michie-onmachineintelligence.pdf. You can also watch https://www.youtube.com/watch?v=R9c-_neaxeU.

In the following, we are only going to deal with 3x3 games, because that is what was defined in the paper. You do not have to worry about other board sizes, only 3x3.

We have given you everything that is required for this part, so you only need to focus on the implementation of **ComputerMenacePlayer** and **MenaceTicTacToeGame**. You need however to use your working solution of assignment 3.

In our solution, we have our MENACE player, playing against a range of possible players: a human, a random player, a prefect player, or another MENACE player.

The human player and the random player were already implemented in the previous assignments. We do provide an implementation of a perfect player. You do not need to understand how it works precisely (but of course, you can!) but you should definitely have a look at the code since this will help you a lot for the implementation of MENACE.

We have done a couple of changes in the design: first, we now would like our Players to share some additional methods. It used to be that all a Player had to do was to give a concrete implementation of the method **play**. Now, we want to be able to inform the Player that a new game is starting and that a game is finished. We want the player to keep some stats about its performance: how often it won and lost overall, as well as over the last 50 games (to track progress). The implementation of some of these methods are common to all Players, so we would like to add the code directly in **Player**. However, **Player** was an interface which prevented us to do this. So we transformed it in a full fledged class. We still cannot provide a default implementation for the method **play**, so that method is still **abstract**, therefore **Player** is now an **abstract class**. We have provided the implementation for all the other methods of the class **Player**

Second, some of our players will need specialized version of **TicTacToeGame**. For example, our perfect player needs to enhance **TicTacToeGame** instances to record which of the possible moves are winning and which ones are losing. So we created the class **PerfectTicTacToeGame**, a subclass of **TicTacToeGame** that **ComputerPerfectPlayer**

uses to record the additional information. The way the class works is that when an instance of **ComputerPerfectPlayer** is created, it first creates the list of all possible games, as we did in Assignment 3. The list is however made of **PerfectTicTacToeGame** instances instead of **TicTacToeGame** instances as we had in Assignment 3. Once the list is created, the **ComputerPerfectPlayer** instance precomputes all the moves from all the possible games to figure out which ones should be played and which ones should be avoided. It is then ready to play games. When its method **play** is called, it receives an instance of **TicTacToeGame** as parameter, which is the current state of the game. It looks through its list of all precomputed games to find which one corresponds to the current state of the game (up to symmetry) and then selects from there one of the best possible moves, as precomputed during initialization. **ComputerMenacePlayer** will work very much in the same way.

Note that the construction of all the possible games uses almost the same code as the method **generateAllUniqueGames** of **ListOfGamesGenerator** in assignment 3. The only difference is that **TicTacToeGame** is replaced with **PerfectTicTacToeGame**. We have copied over the code in **PerfectTicTacToeGame**'s constructor instead of reusing the previous method to simplify the problem. You can do the same thing for your implementation of **ComputerMenacePlayer**.

The gist of MENACE is that it precomputes all possible games (up to symmetry), and for each game, it initially provides a certain number of beads for each possible move. When playing a game, MENACE finds the game corresponding to the current state and randomly selects one of the beads it has for that game. The more a given move has beads at that stage, the more likely it is to be selected. Once the game is finished, MENACE will update the number of beads for each of the moves used during that game, based on the outcome: if the game was lost, then the beads that were selected will be removed, making it less likely that similar moves will be selected in the future[1]. If the game is a draw, then the bead are put back, and another similar bead is added each time increasing a little bit the chance of selecting that move in the futur. If the game is a win, then the bead is put back and 3 similar beads are added.

You are asked to provide an implementation of **ComputerMenacePlayer** that behaves as expected. You have to also implement **MenaceTicTacToeGame**, a subclass of **TicTacToeGame**, which **ComputerMenacePlayer** instances use to precompute all possibly games and record the current number of beads for each possible move of each possible game.

In the paper, it is assumed that MENACE always plays the first move (and our experiments actually suggest that MENACE learns much better how to be a first player than to be a second player). The main instance of MENACE in our system is also set to always play first (though you can easily change that in the code), but since we can play MENACE against MENACE, we need to have an implementation that can play both X and O. The paper specifies the initial number of beads for X only. We will use slightly different numbers for X (8,4,2,1 instead of 4,3,2,1) and we will use similar numbers for O. In other words, the initial number of beads are

- 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

The implementation of the system that we provide works as follow: initially, a new instance of MENACE is created. That instance has not been trained and therefore is very weak. You can chose the opponent, which is either computer driven (random, perfect or MENACE), or a human (you). If you choose a computer driven opponent, then the system will automatically run 500 games between MENACE and that opponent, during which MENACE should improve its play (especially the first time). and should be then much tougher to beat. We also have included an option to reinitialize MENACE, so that you can easily restart a learning stage from scratch.

Here is a sample run of the system. We will first run two rounds against a human:

[1] This approach has as problem: it is possible to remove the **last** bead of a game this way, in which case Menace will be stuck if that game is seen again in the futur, with no move having any chance of being selected. To avoid that, we only remove a bead if that is not the last one in that game.

```
% 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 |
-----------
   |   |

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

O to play: 8
 O | X |
-----------
   | X | X
-----------
   | O |

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

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

As can be seen, so far MENACE is not really good and lost twice in a row despite being first player. Let's now train it against a perfect player:

```
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 (which is player 1 here) has lost 34 games against the perfect player in the 500 that were played, and none during the last 50 games.

Let's try it again against a human player:

```
(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.
```

Now, MENACE is a much better player, and plays indeed very well and got a draw. Note however that it is not perfect and can still be beaten. Look at this next game:

```
(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
%
```

We managed to beat it again. There is in that case a fairly straightforward reason for this: our first (human) move was to play 2 on a game in which 5 was first played. If you think about it, this is actually a very bad move, a guaranteed loss against a capable player. But because this is a losing move, our perfect player will **never** play it, and therefore MENACE has actually not been trained at all to handle it. This illustrates some of the challenges in machine learning, but that is an entire different discussion!

Academic Integrity

This part of the assignment is meant to raise awareness concerning plagiarism and academic integrity. Please read the following documents.

- https://www.uottawa.ca/administration-and-governance/academic-regulation-14-other-important-informat

- https://www.uottawa.ca/vice-president-academic/academic-integrity

Cases of plagiarism will be dealt with according to the university regulations. By submitting this assignment, you acknowledge:

1. I have read the academic regulations regarding academic fraud.
2. I understand the consequences of plagiarism.
3. With the exception of the source code provided by the instructors for this course, all the source code is mine.
4. I did not collaborate with any other person, with the exception of my partner in the case of team work.
    - If you did collaborate with others or obtained source code from the Web, then please list the names of your collaborators or the source of the information, as well as the nature of the collaboration. Put this information in the submitted README.txt file. Marks will be deducted proportional to the level of help provided (from 0 to 100%).

Rules and regulation

- Follow all the directives available on the assignment directives web page.
- Submit your assignment through the on-line submission system virtual campus.
- You must preferably do the assignment in teams of two, but you can also do the assignment individually.
- You must use the provided template classes below.
- If you do not follow the instructions, your program will make the automated tests fail and consequently your assignment will not be graded.
- We will be using an automated tool to compare all the assignments against each other (this includes both, the French and English sections). Submissions that are flagged by this tool will receive the grade of 0.
- It is your responsibility to make sure that BrightSpace has received your assignment. Late submissions will not be graded.

Files

You must hand in a **zip** file (no other file format will be accepted). The name of the top directory has to have the following form: **a4_3000000_3000001**, where 3000000 and 3000001 are the student numbers of the team members submitting the assignment (simply repeat the same number if your team has one member). The name of the folder starts with the letter "a" (lowercase), followed by the number of the assignment, here 4. The parts are separated by the underscore (not the hyphen). There are no spaces in the name of the directory. The archive a4_3000000_3000001.zip contains the files that you can use as a starting point. Your submission must contain the following files.

- README.txt
    - A text file that contains the names of the two partners for the assignments, their student ids, section, and a short description of the assignment (one or two lines).
- 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

For all your questions, please visit the Piazza Web site for this course:

- https://piazza.com/uottawa.ca/winter2020/iti1121/home

Last modified: March 8, 2020