Discovery Lab 2: Artificial Intelligence

CISC181 Spring 2014
Assigned: April 7
DL2 Due: April 13
Project Checkpoint 2 Due: April 27 (please see last page of this document)
Text references: Chapters 7, 8, 10, 11, 15, 22.4, 22.6

Discovery labs should be completed by a pair of students. Permission from the professor is required to work in a team of more than two students. Your work on the discovery lab must be completed without the help or consultation of students outside your team.

Discovery labs are designed to look at a particular area of Computer Science applications and research. In each lab you will develop a working prototype application that practices fundamental programming techniques. After completing the prototype, your group will apply these techniques to develop a checkpoint for your course project.

In this discovery lab you will explore the area of Artificial Intelligence.

Objectives:

1. Design, implement, test, and debug programs in an object-oriented programming language.
2. Explain how abstraction mechanisms support the creation of reusable software components.
3. Defend the importance of abstractions, especially with respect to programming-in-the-large.
4. Explain the relationship between the static structure of the class and the dynamic structure of the instances of the class.
5. Choose appropriate conditional and iteration constructs for a given programming task.
6. Create algorithms and heuristics for solving simple problems.
7. Write programs that use the following data structures: arrays, 2D arrays, lists.

* Problems you need to solve and turn in are listed in green text.

The field of Artificial Intelligence (AI) attempts to understand intelligent entities. This simple definition belies a vast historical debate over the term intelligence. What does it mean to *be* intelligent? Does it mean to *think* like a human? Does it mean to *act* like a human? Maybe it does not even involve humans, but a high-level concept like *rationality*. We will not answer this question during this Discover Lab, but instead will focus on the task of building programs to solve problems that are generally thought to require *intelligence*. The central problems of AI include such traits as reasoning, knowledge, planning, learning, communication, perception and the ability to move and manipulate objects.

A fundamental question of Artificial Intelligence is how a program can make intelligent decisions. Some of the most visible AI applications tackle this question "head-on" by pitting a program's ability to make an intelligent decision against humans:

Watson: IBM's DeepQA research project competes on Jeopardy against humans.

World chess champion, Garry Kasparov, competing in regulation chess with IBM's Deep Blue.

To illustrate how difficult this question is, let's consider how we make intelligent decisions on a daily basis. Suppose you were faced with the following cereal choices for breakfast:

- Cocoa Puffs
- Lucky Charms
- Corn Flakes

Which would you choose? And why? Would your choice change if you were told that it would cost $100 for a bowl of Cocoa Puffs/Lucky Charms, but that a bowl of Corn Flakes was free? What if the prices were $1, $1, and $0.50 instead?

To encode our decision making process into a program, we will loosely borrow from the highly *rational* model of utility theory (http://en.wikipedia.org/wiki/Utility). Every choice will be given a score, which is equal to benefit minus cost -- that is, the measure of relative satisfaction minus the cost of resources used to obtain the satisfaction. In our above cereal dilemma, suppose we measured our relative satisfaction of the choices as follows:

- Cocoa Puffs - 10
- Lucky Charms - 9
- Corn Flakes - 1

If we equate these level of satisfaction with dollar amounts, our utility theory would give scores of -90, -90, and 1 for our initial cost model, and 9, 8, and 0.5 for the second cost model. Given each of the cost models, we would actually make a different *rational* decision -- to choose Corn Flakes in the first model, and Cocoa Puffs in the second model.

In this Discovery Lab you will develop an Artificial Intelligence program that will play our TicTacToe5x5 game. You will add logic to the basic model for the game: actions that can change the game state, and a basic AI that will pick the highest scoring action from among a collection of all valid actions.

Things that have been added since TicTacToe5x5 checkpoint 1:

- Several framework classes have been added or modified. Since these will be used by ALL games you might want to look at how they are used for TicTacToe5x5:
    - The Game class now has additional logic related to starting, ending, performing actions, ticking, and general game management. All projects should extend this Game class with their own logic.
    - An AI class represents a computer player of a Game. Some methods are already written in this class. You will complete the algorithm for finding the best action. Each project will also have a specific subclass of AI that contains logic specific to the project.
    - A Tickable interface with a related Ticker class now handles time based events. These events include ticks for a Game and ticks for an AI so that it waits a determined amount of time between taking actions.
    - A GameListener receives notification when events take place in a Game. Specific life-cycle events (start, end) as well as onTick, onPerformAction are automatically generated, but Game subclasses can also broadcast custom events to listeners using broadcastEvent.

To get started, download DL2.zip from Sakai and import it into Eclipse.

Part 1: Implementing game update logic for TicTacToe5x5

- Modify the class for TicTacToe5x5Game.
    - Add a method to the TicTacToe5x5Game class:

      changeTurn() -- this method should mutate the TicTacToe5x5Game such that values of the turn and notTurn properties are swapped.

      Your class should now pass the test for changeTurn in TicTacToe5x5Tests.java. Look at the tests for PlacePieceAction before continuing to the next part.

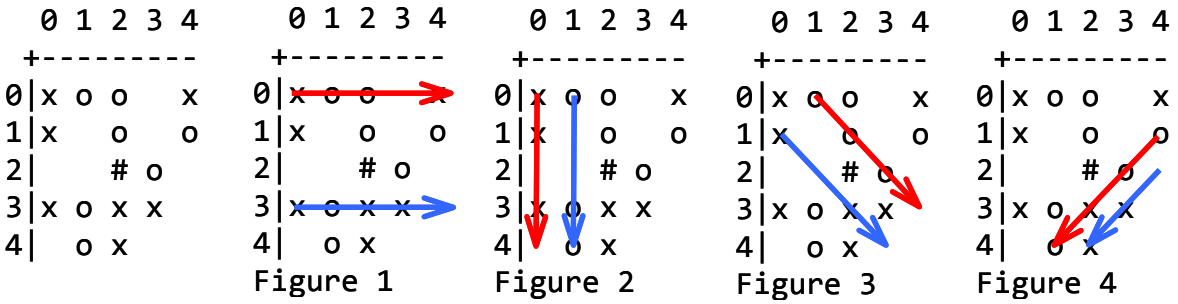
- Modify the class for PlacePieceAction.
    - Implement the update method so that the board and turn properties of the game are updated correctly for the test for PlacePieceAction.

Note that the game update logic for TicTacToe5x5 is *very* simple. Most games will have much more complicated update logic for different Actions, and also may have onTick logic. Please take a minute to look at the SnakeGame.onTick if you are doing a game that has time-based logic.

Part 2: Implementing scoring rules and a heuristic for TicTacToe5x5

- Implement the hasEmptySpace method on TicTacToe5x5Game. See comments and test for more details.

- Implement the getMaxSequence method on TicTacToe5x5Game. This method returns the maximum sequential appearences of a symbol along a linear path. **This method is relatively difficult.** The best way to explain the intended behavior is with pictures and the tests that are given. Here is mid game state 2, with several tests:



In each figure a red line and a blue line represent two different tests. In Figure 1 the red line traverses row 0. x has only 1 sequential appearance in this row. o has 2 sequential appearances in this row. The test for row 0 is:

```
assertEquals(2, game.getMaxSequence(0, 0, 0, 1, 'o'));
assertEquals(1, game.getMaxSequence(0, 0, 0, 1, 'x'));
```

Each test starts at row 0, column 0 with the linear path described as changing row by 0 each iteration and column by 1 each iteration, thus testing a left-to-right path.

Note that your method will be called with different starting row, columns and one of 4 combinations of dr, dc (change in row, column): left-to-right=[0, 1], top-to-bottom[1,0], down-right-diagonal[1,1], down-left-diagonal[1,-1]. These are sufficient to test max sequence in a TicTacToe5x5 game.

- Implement the getScore method on TicTacToe5x5Game. This method will call getMaxSequence many times for one player symbol: once for each unique left-to-right, top-to-bottom, down-right-diagonal, and down-left-diagonal in TicTacToe5x5. The return value is the max value of all results returned from getMaxSequence.

If completed correctly, tests should now pass for hasEmptySpace, getMaxSequence, getScore, isWinner, and isEnd.

Part 3: Creating a basic "greedy" AI

Time to build the AI! Look at the tests for getAllValidActions, getHeuristicScore, and getBestAction before continuing.

- Add code to getAllValidMoves in the TicTacToe5x5AI class. This method takes a TicTacToe5x5Game as a parameter and returns a list of ONLY valid actions for the game.

- Add code to getHeuristicScore in the TicTacToe5x5AI class. This method computes the score for a speculative action by mutating the board, computing the game's score, and then undoing the mutation. The return value for the heuristic is the computed game's score.

- Complete the implementation of getNextAction in the AI class. **NOTE:** this class is in the game framework package. This implementation will look at all of the possible valid actions, compute heuristic score for each of them, and then greedily choose the one with the max score (or if there are more than one max score actions, choose one of these randomly). Read the pseudocode in the method comments for a guide.

  Make sure that your code passes the tests for all three of these methods.

Once your code has passed all tests, you should be able to run TicTacToe5x5Game and see two instances of TicTacToe5x5AI play verses each other. They are not very good because the heuristic function they use is very basic -- it only tries to maximize connected sequences of pieces.

Export and submit your DL2 project to Sakai as a .zip archive.