Class 19: PA3 on Games:
TicTacToe and Nim
PA3: Games and DP: Background

Cs310 Slides/videos on tic tac toe:
• Intro in class 14 of March 24
• Continued in class 15 of March 26

Cs310 Slides/videos on Dynamic Programming:
• DP for change-making with non-US-coinage: code in class 12, worked-out example in class 15 of Mar. 26, hw4 problem
• class 15 slides from Prof. Haspel

4/8/2020
PA3: Games and DP: Background

From pa3.html:

• Read Wikipedia Dynamic Programming intro, and Sec. 1.3 and 2.2. Also there: a fascinating story on how DP got its name.
• Also see Geeks for Geeks Dynamic Programming Basic Concepts.
• Weiss Chap. 7, esp. 7.6., and 10.2.
Pa3 Provided files

- TicTacToe.java: from Weiss, but without alpha-beta pruning, which is specific to 2-player games
- Best.java: a Best object is used to carry the best move from chooseMove to the client, so class Best needs to be known outside TicTacToe.java. Thus it has its own file.
- PlayTicTacToe.java: provided client code for TicTacToe.
- Nim.java: code for playing Nim, another 2-player game
The provided files are all in the default package, so we put them in the src directory, and compile there.

You will set up a games package as part of this assignment, which needs a src/games directory.

So the directories of the setup project are:

- src for TicTacToe.java, etc., also TicTacToe.class to start with
- src/games for .java files in package games (originally empty)
- bin/games for .class files for java files in src/games (orig. empty)
- bin itself can hold TicTacToe.class, etc. if you (or your IDE) wants
Is TicTacToe.java encapsulated?

• This code certainly looks better encapsulated than JDK HashMap of pa2!
• It has one field, properly made private:
  ```java
  private int[][][] board = new int[3][3];
  ```
• It has 3 private methods, all helpers to chooseMove: place, squareIsEmpty, and positionValue
• All the other methods are public, so should be the API for calls from clients
TicTacToe’s API

- public TicTacToe() (constructor)
- public int[][] getBoard() \(\gets\) exports private data in bulk
- // Find optimal move
- public Best chooseMove(int side, int depth)
- // Play move, including checking legality
- public boolean playMove(int side, int row, int column)
- public void clearBoard()
- public boolean boardIsFull()
- public boolean isAWin(int side)

4/8/2020
Fixing TicTacToe’s API

• It’s not a good idea to let the client have direct access to the class’s private data.
• It’s tantamount to having the board[][][ ] public.
• With this method, the client could even change the data in the array, i.e., change what’s on the board without going through the legitimate API playMove.
• You’ll fix this encapsulation defect in pa3 step 1.
Enhancement in step 2

• Can we let the human use chooseMove when they want to, to harness its power to find the best move?
• Find out in step 2, where the user can enter -1, an impossible move value, to ask for this service.
Nim

- **Wikipedia entry**: “Nim is a mathematical game of strategy in which two players take turns removing (or "nimming") objects from distinct heaps or piles.”
- For example: For one move, take any number of matches from one row.
- Our Nim setup has 3 rows, with 1, 3, 5 “stars” in each row.
Step 3: Nim.java

- This is provided, though you could try to write it yourself if you want.
- Its main has a little test code to try out.
- Then write PlayNim3.java, by starting from PlayTicTacToe.java and modifying it as little as possible.
- Since Nim.java has no chooseMove for the computer’s move, you’ll need to have the human supply both human and “computer” moves.
Step 4: chooseMove for Nim

- Provide Nim with chooseMove (recursive search without dynamic programming) so that the computer plays a good game.
- Use Best.java for the best move (use “column” for the number of stars)
- Note that making a trial move in chooseMove requires changing one of the heaps and the nextPlayer value.
- The computer, once optimal, should choose move A3 (row1, take 3 stars) to start the game.
Step 5: Game interface

• Write an interface Game.java that covers the actions needed for the clients for both games PlayTicTacToe1 and PlayNim4.
• Many of these actions have different names in the two cases, so you need to change some method names, ending up with TicTacToe5 and Nim5.
• For example, clearBoard() in TicTacToe has the same generic action as init() in Nim, namely, to bring the game to its initial state. Clearly init() is the more generic name, so rename clearBoard in TicTacToe to init and put init in the interface.
Step 5, cont.

- Once you have aligned the APIs of the two game classes, and made them both “implement Game”, morph PlayTicTacToe or PlayNim into PlayGame, a single client that can handle either game.
- Now that we have a serious multi-class project, it deserves its own package. Put all the involved classes in the games package.
- Even Best.java needs to be copied and edited, since Java won't import a file from the default package to a named package.
Step 5: a new API call

• You’ll find that the two games need different user prompts that are not supplied by the API so far.
• One needs “row: “ and “column: “ and the other needs “row: “ and “stars :”.
• So we need to design a new API call to provide these prompt Strings.
• See the assignment for more info on this, or come up with your own idea.
Step 6: Optional Challenge

• We’ve now covered the required part of the assignment.

• Clearly there are unfinished things here:
  • DP for Nim’s chooseMove
  • Making a Move type that can bridge the two cases better than our kludge of using column for #stars in Best. BTW, kludge has a Wikipedia page.
  • Other games, like 4x4 tic tac toe, described in Wikipedia’s Tic-tac-toe page.