Boxed

CSC 252 – Sections A and B

**Assigned:** 21 August 2014 **Due**: 28 August 2014

**Overview**

Dots and Boxes is a simple yet fascinating game when it comes down to what strategies work to regularly beat your opponent. You may not think it, but there is strategy (and good strategy at that!) to regularly blowing your friends and neighbors out of the proverbial water.

**Outline**

Dots and Boxes is a game where you start with a grid of dots. Each person takes a turn by drawing a line from one horizontal or vertical line to an adjacent dot. Players draw these lines with the intent of capturing boxes. Once four lines connect to make a square, whoever drew the fourth line will “capture” that box. Each time a player captures a square, he is required to place an additional line on the grid. The game ends when there are no lines left to draw; the winner is the one who captured the most boxes.

*If you’ve never played this game before, you can find more detail online at* [*http://en.wikipedia.org/wiki/Dots\_and\_Boxes*](http://en.wikipedia.org/wiki/Dots_and_Boxes)

For this lab, you will create a simple representation for the Dots and Boxes game that can be used to analyze different aspects of the board.

***Graph.java***

This class is the graph that represents the Dots and Boxes game board. There are two possible implementations, Adjacency Matrix and Adjacency List. Adjacency Matrix is recommended (no pointers!), but you may implement either one. The ADT for the Graph class is as follows:

new Graph(int v); // initializes a graph of *v* vertices with no edges

int vcount(); // returns number of vertices (whether connected or not) in the graph

int ecount(); // returns the number of edges in the graph

int first(int v); // returns the first vertex (in natural order) connected to vertex *v*. If there are none, then vcount() is returned

int next(int v, int w); // returns the vertex (in natural order) connected to vertex *v* after vertex *w*. If there are no more edges after *w*, vcount() is returned

void addEdge(int v, int w, int wt); // adds an edge between vertex *v* and vertex *w*.

void removeEdge(int v, int w); // removes edge between vertex *v* and vertex *w*.

boolean isEdge(int v, int w); // returns whether there is a connection between vertex *v* and vertex *w*

int degree(int v); // returns how many edges depart from vertex *v*

int getMark(int v); // returns any graph coloring for this vertex

void setMark(int v, int m); // colors vertex *v* color *m*

A nice implementation of many of these methods is provided in the Shaffer book in chapter 11.

***BfsGraphTraversal.java*** and ***DfsGraphTraversal.java***

This class performs a traversal of a Graph and emits the visit order of the graph’s vertices. The method signature is as follows:

List<List<Integer>> traverse(Graph g);

The inner List represents a traversal of a connected component in the graph. The outer list represents all the connected components.

You must implement both Depth-First Search and Breadth-First Search, though the actual Dots and Boxes class does not need to exercise both. Both need to adhere to their classic efficiency classifications.

***DotsAndBoxes.java***

This class is for playing the actual game. It will implement the following methods:

new DotsAndBoxes(int rows, int columns);

int drawLine(int player, int x1, int y1, int x2, int y2); // draws a line from (x1, y1) to (x2, y2) **(0,0) is in the upper-left corner**, returning how many points were earned, if any

int score(int player); // returns the score for a player

boolean areMovesLeft(); // returns whether or not there are any lines to be drawn

int countDoubleCrosses(); // returns the number of double-crosses on the board

int countCycles(); // returns the number of cycles on the board

int countOpenChains(); // returns the number of open chains on the board

*drawLine.* Representation for this lab is key. A well-researched implementation is that of Strings and Coins. It turns out that Strings and Coins is isomorphic to Dots and Boxes (for every state in Dots and Boxes, there is a one-to-one relationship with a state in Strings and Coins). To represent Dots and Boxes as Strings and Coins, the graph needs to be one row and one column bigger than the user requests. Then, an edge is drawn in the Strings and Coins graph wherever there is **no** edge in Dots and Boxes:

# # # # # #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# # # # # #

Strings and Coins

Dots and Boxes

When a player wants to draw a line in dots and boxes it is the same as *cutting* a string in Strings and Coins:

# # # # # #

**. ------- . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . ------- . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# # # # # #

Two Lines Drawn in Dots and Boxes

# # # # # #

**. ------- . ------- . | . | .**

# **|** # **---** # **|** # **---** # **---** #

**. ------- . ------- . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

**. | . | . | . | .**

# # # # # #

Six Lines Drawn in Dots and Boxes (six strings now cut in Strings and Coins)

This turns out to be very useful when analyzing the board for significant elements.

# # # #

**. ------- . ------- . |**

# **|** # **---** # **|** #

**. ------- . ------- . |**

# **---** # **---** # **---** #

*Double-cross.* A double cross is a shape in which there are six lines that form into a two-box rectangle. Specific to the double-cross, one line drawn will earn the player two boxes. In the game of Strings and Coins, it is when there are to coins joined together by one string and are disconnected from every other coin.

# # # # # #

**. ------- . ------- . ------- . | .**

# **|** # **---** # **---** # **|** # **---** #

**. | . ------- . | . | .**

# **|** # **---** # **---** # **|** # **---** #

**. ------- . ------- . ------- . | .**

# **---** # **---** # **---** # **---** # **---** #

*Cycle.* A cycle is a “hallway” of dots and boxes lines that wraps back onto itself. Specifically, if one player draws any line in a cycle, the next player can obtain all the squares in the cycle.

# # # # # #

**. ------- . ------- . ------- . | .**

# **---** # **---** # **---** # **|** # **---** #

**. ------- . ------- . | . | .**

# **---** # **---** # **|** # **|** # **---** #

**. | . | . | . | .**

# **---** # **---** # **---** # **---** # **---** #

*Chain.* A chain is an open “hallway” of dots and boxes lines that **does not** wrap back onto itself and is at least three boxes long. If one player closes one end of the hallway, then the next player can capture all the boxes in the hallway **or** can leave a double-cross at the end.

The class DotsAndBoxes should create graph of Strings and Coins. When a Dots and Boxes line is drawn, a String is cut. When one of the *count* methods is called, the Strings and Coins graph is traversed into its connected components and analyzed.

When four lines are drawn to form a single box, the player that drew the last of the four lines obtains that box. He now owns it, and receives one point.

**What to Study**

Read Sections 3.5 in the Levitin text. While not directly related, sections 8.4 and 9.1-9.3 may prove helpful.

There are several articles on Dots and Boxes strategy and implementation hints:

<http://www.geocities.ws/ilanpi/tutorial1.html> - Also includes clear diagrams and definitions of chains, cycles, etc.

<http://ojs.academypublisher.com/index.php/jsw/article/viewFile/jsw0702256262/4379>

<http://www.aaai.org/ocs/index.php/aaai/aaai12/paper/download/5126/5218>

<http://www.rowlandoflaherty.com/wp-content/uploads/2012/04/Project1_Paper_DotsAndBoxes_OFlaherty_Wu.pdf>

For the stretch goal, Ilan Vardi’s geocities page (above) goes through several Dots and Boxes strategies.

**What to Hand In**

Pass-offs will be done via one or more unit tests. Submit to your GitHub url **Graph.java, BfsGraphTraversal.java, DfsGraphTraversal.java, DotsAndBoxes.java,** and any other files that are necessary for your code to work.Please also submit a jar file with all your code.

**How You Will Be Graded**

You will receive **5%** for a correct implementation of Graph, **10%** for each correct traversal class, **9%** each for drawLine, score, areMovesLeft, and the three count methods, and **15%** for the verbal defense of your code.

**Stretch Goals**

* Create an AI that leverages DotsAndBoxes’s analysis of the board. The counting methods that you wrote are specifically designed for a minimax approach; however, you are welcome to do what you like. For **5%**, show that it regularly beats a player that places lines randomly. For an additional **10%**, show that it beats more than half the time a player that greedily closes any 3-line box that it finds. Remember that whenever a player scores a point, he **must** take another turn (so long as there are moves left). I have an old test harness that I can dust off if you would like to use it.
* Add countClosedChains, which is a chain that is closed on both ends. Also add countHalfOpenChains, which is a chain that is open at only one end. (**+5%**)