Amy Biasella

CSCI E10B Graduate Final Project Writeup

May 4, 2018 (Spring 2018 semester)

TA: David Habermehl

Files included:

**DotsAndBoxes.java** *(run this one)*

**Cell.java**

**Player.java**

**Background and Rules**

The objective of this two player “pen and pencil game” is for an opponent to capture as many boxes as possible. The playing area starts as an empty grid of dots, and players alternate turns placing a single line segment between two orthogonal dots. If a line segment completes the fourth side of an individual cell, the player that placed that segment gets a point, and the chance to place another line segment. Once a terminal position is reached (all possible 1x1 boxes have been enclosed), the player with the most points wins.

**Implementation Notes**To use: compile and run **DotsAndBoxes.java**This project does not use AI - two players will alternate turns.The main program (DotsAndBoxes) is responsible for initializing a game board, initializing Players, and initializing the 16 individual cells that make up the grid. Each cell is an object - cell borders are activated when a user clicks it (see **Border Activation** below). A “line” is created when a border *and its neighbor’s border* are both activated. This requires the main program to facilitate communication between clicked and neighboring cells.

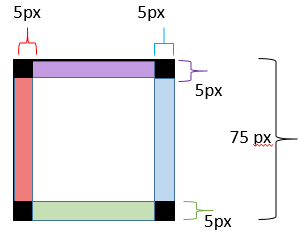
The main program is also responsible for maintaining turn order. P1 is automatically set as first player. Play switches when a cell is NOT enclosed. Whether or not a cell is closed is determined by the Cell itself.

**Template class 1 of 2:** **Player.java**

Players are simple. Each Player has two fields - ID and Points. The id is passed in from the main program to create players P1 and P2. Methods include a private ID setter and a public setter for being able to alter points when a cell is enclosed, as well as publically being able to reset Player point data.

**Template class 2 of 2: Cell.java**

*Representation of a single cell.*

A cell object is a modified JPanel. It has four borders that are toggled on during gameplay to represent enclosing the cell. Colors are just for demonstration and not seen in game.

It is also drawn with four black corners, that when grouped in the main game’s GridLayout will form the “Dots” of the main GUI grid. (See note in Startup and Gameplay that acknowledges a shortcoming with this approach.)

Each border has a TRUE/FALSE state that is set as TRUE when clicked. This is needed to prevent borders from being redrawn and gaining points if a user clicks the same area twice.

When the user clicks, the main program registers the Cell clicked, X coordinate, and Y coordinate and passes the coordinates to the clicked Cell. Cells are responsible for determining if a user’s click is in a valid hitbox, and which one. (Gritty details in “Border Activation” below). The main program is responsible for finding out if the clicked cell has a neighbor, and if one exists, sends in what border should be activated to that neighbor.

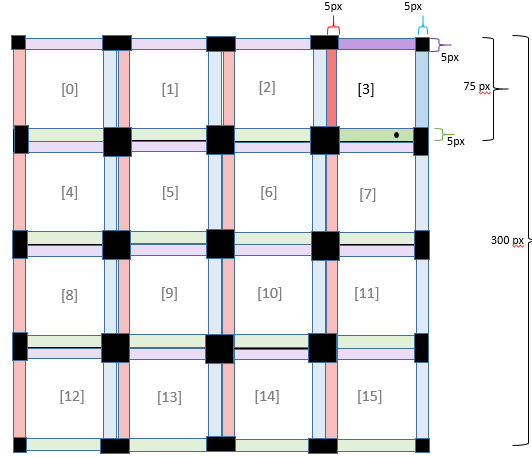
A cell also has a isEnclosed field that is set as TRUE when all four border objects are TRUE. When this happens, the active player gains a point and the cell background is changed to a player’s identifying color.

A cell has a reset function that will reset all borders to false, set enclosed as false, clear background, clear borders, and reset the player value that represents who captured it.

**Border activation**

The logic of border activation is diagrammed as follows. The colored borders in each cell represent a directional (**top**, **bottom**, **left**, **right**) hitbox that will activate that border. Clicking in either the center of a cell or in the black grid markers in the corners will do nothing.

Activation means setting that cell border to be visible, as well as setting its related boolean value to TRUE. If a user clicks again in the same spot, nothing will happen.



Click (●) at:

Cell [3] at x= 60, y=70

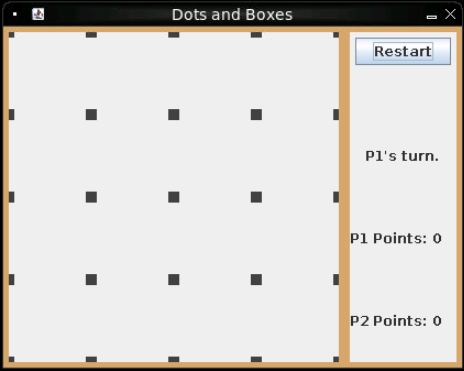
- This activates the **bottom border** of element [3] in the array.  
  
It will also activate the **top border** of Cell [7] - see getAdj() method in the main DotsAndBoxes program. These borders are then both rendered by setting the border. Border setting involves saving the old border object and combining it with the new border that was just clicked.

After a border is set, the Cell (both the clicked cell and neighboring cell, if one exists) asks “Are *all* of my borders activated?” If so, the cell is considered enclosed, the cell is colored, and a point is awarded to the active player.

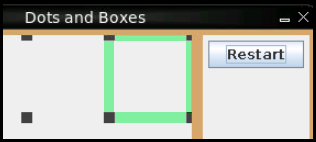
**Startup and Gameplay**

Below are screenshots for what the user will see when launching DotsAndBoxes.java as well as other various gameplay screens. The initially loaded black squares are grid marks and users will click between these dots to set a line.

*View at start:*

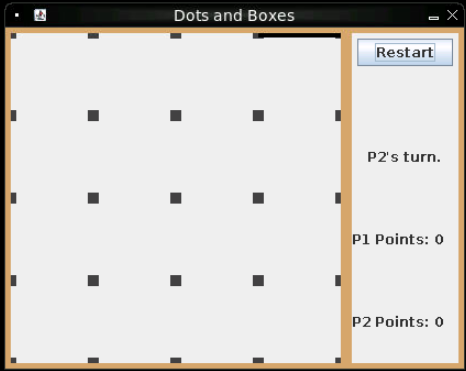


**Note** that the corner and border dots are smaller than middle dots. I made the Cell objects responsible for drawing boxes in their corners but was not able to figure out how to make the dots appear uniform.



Therefore the hitboxes are unfortunately smaller along the edges and slightly more fiddly to click, but are still functional. Note the example to the right, showing the hitboxes to activate respective borders in Cell [3].

*View after P1 clicks the top border of Cell 3*:



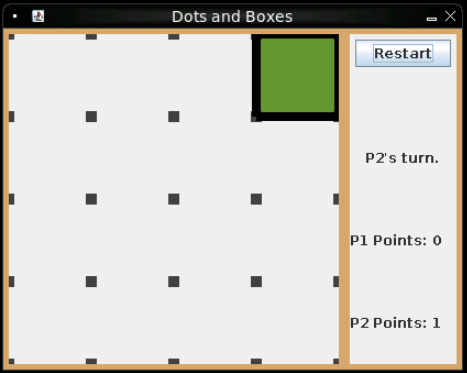
After P1 clicks the top border of cell 3, a line appears.

Play passes to P2.

*Not shown - P2 clicks Cell 3, right border. Play passes to P1.*

*Not shown - P1 clicks Cell 3, left border. Play passes to P2.*

*P2 clicks the bottom border, closing Cell 3:*



After P2 clicks:

- The border is set at the bottom of the cell.

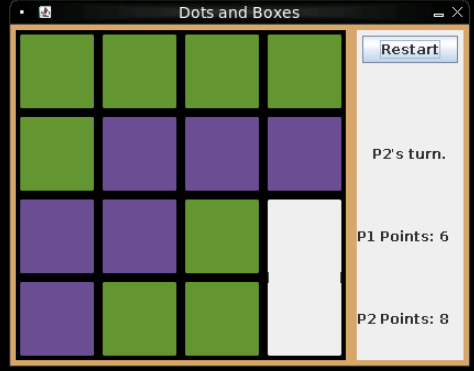
- The cell takes P2’s color.

- P2 gains a point

- P2 is still the active player and must make another move.

**Game End**

Example showing nearly complete board with P2 having the last valid move:



After the last move is made, P2 encloses both cells and therefore gains two points at once.

A prompt is shown congratulating the winner (or announces a tie). The game board is not reset when the user clicks OK. The Restart button must be clicked for a new game.

