

CS-259 Data Structures with Java

Project 2: Breakout v2:
Collision Detection in 2D



Instructor: **Joel Castellanos**

e-mail: joel@unm.edu

web: <http://cs.unm.edu/~joel/>

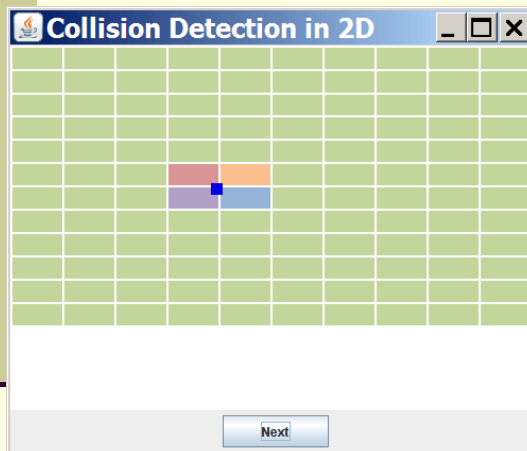
Course website:

<http://cs.unm.edu/~joel/cs259/>



10/9/2009

Breakout: Project 2, Milestone 2

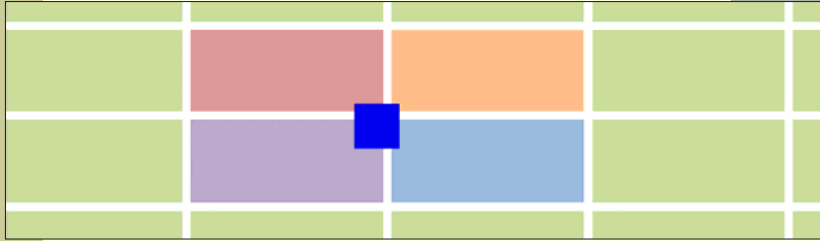


Due Sunday, Oct 11.

- Button added to ContentPane of JFrame.
- Square "ball" moves on button click.
- Ball bounces on boundaries of JPanel.
- Blocks that do not intersect with the ball are colored some shade of green.
- A block that intersects with the ball is colored according to which corner of the ball it intersects.

2

Collision Detection in 2D: Collision Colors

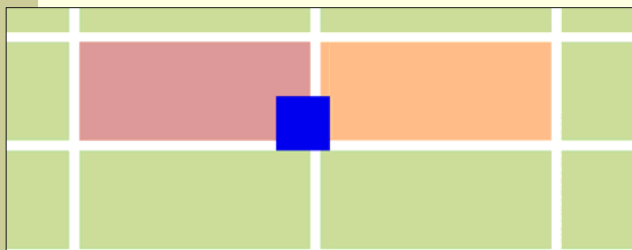


A block that intersects a corner of the ball must be colored as specified:

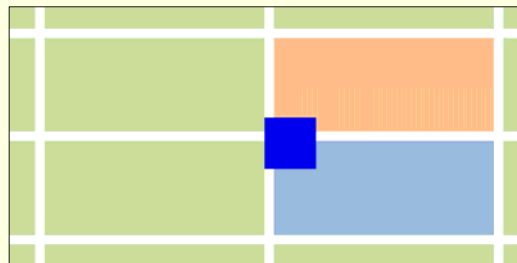
- Upper Left corner of ball: some shade of red.
- Upper Right corner of ball: some shade of orange.
- Lower Right corner of ball: some shade of blue.
- Lower Left corner of ball: some shade of purple.

3

Collision Detection in 2D: Gaps

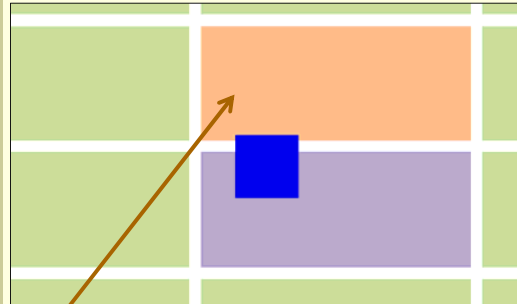


Corners of the ball in gaps do not collide with a block.



4

Collision Detection in 2D: Multi-Corner



If more than one corner falls within a single block, then the block may be given the color of any of those corners.

May be colored red or orange.

May be colored red, orange, blue or purple.

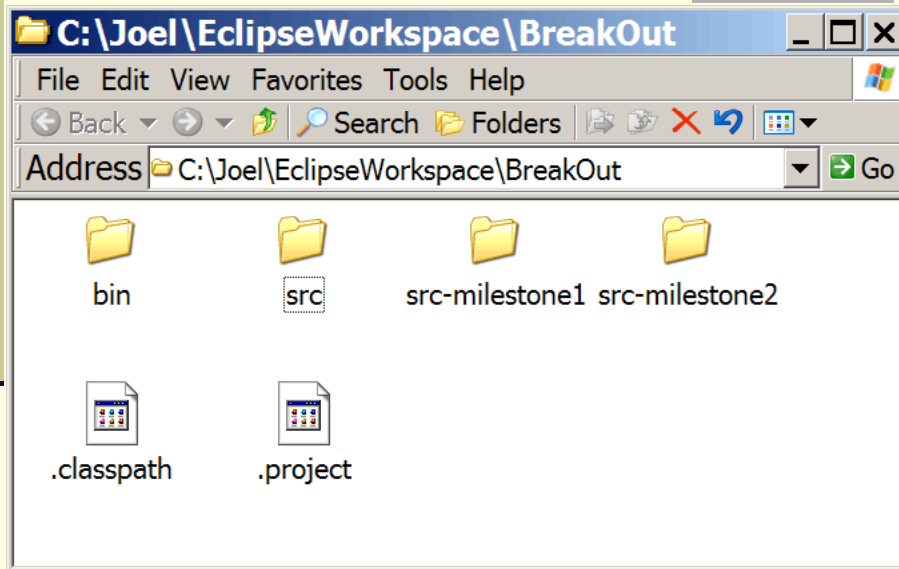
5

Project 2, Milestone 2: Grading Rubric

- 3 Points:** Adheres to CS-259 coding standard.
- 2 Points:** Adheres the given class, field and method names.
- 2 Points:** The GUI looks like the given sample: correct number of blocks drawn in the correct locations with the correct size, and correct button in correct location.
- 2 Points:** Clicking the button advances the turn 1 step.
- 2 Points:** The ball bounces correctly.
- 7 Points:** The blocks are colored correctly.
- 2 Points:** On each time step, only the blocks that change color are redrawn.

6

Primitive Version Control



Collision Detection in 2D: Class Structure

You program must consist of two files.

■ Breakout.java

```
public class Breakout extends JFrame implements  
    ComponentListener, ActionListener, WindowListener
```

This JFrame instantiates BreakoutDraw, handles resizing (ComponentListener calls BreakoutDraw.setBounds), handles window closing events (WindowListener), instantiates and contains a JButton, and handles the button click event (ActionListener calls BreakoutDraw.nextTurn()).

■ BreakoutDraw.java

```
public class BreakoutDraw extends JPanel
```

This JPanel does all the graphics, figures out ball collisions, and updates block colors.

8

Breakout.java: Required Packages

```
import javax.swing.JFrame;
import javax.swing.JButton; //For button that calls nextTurn()
import java.awt.Container; //On which to add button and JPanel
import java.awt.Insets; //To get window frame borders

//Listen to frame resize events
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;

//Listen to button clicks
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

//Listen to window closing events
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
```

9

BreakoutDraw.java: Required Packages

```
import javax.swing.JPanel;

import java.awt.Color; //Color of ball, blocks, background

import java.awt.Graphics; //For paintComponent() callback

//For off-screen graphics buffer
import java.awt.image.BufferedImage;
import java.awt.Graphics2D; //For drawing in BufferedImage

//For random starting location and velocity of ball.
import java.util.Random;

//For internal error checking of BreakoutDraw's method arguments
import java.lang.IllegalArgumentException;
```

10

Initial Ball Position and Velocity

At the start and after each frame resize, the ball should be set to an initial position and velocity:

- Initial position of the ball must be a random position along the bottom edge of the panel.
- The ball's initial vertical velocity must be +10 pixels/turn. Throughout the "game" the ball's vertical velocity should always be ± 10 pixels/turn.
- The ball's initial horizontal velocity, must be a random, between $[-10, 10]$, but not 0 pixels/turn.

11

BreakoutDraw JPanel: Block Fields

You may choose your own colors, but each of these fields must be defined

```
private final int BLOCK_GAP = 2;      //Pixels
private final int BLOCK_ROWS = 12;
private final int BLOCK_COLUMNS = 10;
```

```
//BLOCK_HEIGHT is a constant regardless of the frame size.
//blockWidth is calculated based on the inside frame width so that the
// BLOCK_COLUMNS with a BLOCK_GAP between each fills the
// available width.
```

```
private final int BLOCK_HEIGHT = 20; //Pixels
private int blockWidth;              //Pixels
```

```
//Amount of extra space in pixels on each side of block area when
// BLOCK_COLUMNS + BLOCK_GAP does not evenly divide the inside
// frame width.
```

```
private int blockBoarder;           //Pixels
```

12

BreakoutDraw JPanel: Required Methods

You may have additional methods.

```
public void setBounds(int left, int top,  
                    int width, int height) ☒
```

```
public void clear() ☒
```

```
private int columnToPixelLeft(int column) ☒
```

```
private int pixelToColumn(int x) ☒
```

```
private int rowToPixelTop(int row) ☒
```

```
private int pixelToRow(int y) ☒
```

```
public void nextTurn() ☒
```

```
public void paintComponent(Graphics g) ☒
```

13

BreakoutDraw: setBounds()

```
public void setBounds(int left, int top,  
                    int width, int height) ☒
```

This method is called by the parent JFrame after BreakoutDraw is instantiated and each time the parent JFrame is resized.

Responsibilities:

1. call `super.setBounds()`
2. Create an off-screen BufferedImage of the full Panel size.
3. Set the class variable graphics handle to the BufferedImage.
4. call `this.clear()`.

14

BreakoutDraw: clear()

```
public void clear() ☒
```

This method is called by the `this.setBounds()`.

Responsibilities:

1. Paint the full draw area with the background color.
2. Set the initial ball location and velocity.
3. Draw the ball and the fill set of blocks in the off-screen buffer.
4. Call `this.repaint()`.

15

HelloWorld: IllegalArgumentException()

```
1) public class HelloWorld
2) { private double hourlySalary;
3)   public void setSalary(double x)
4)     { if (x<5.25 || x>1000)
5)       { throw new IllegalArgumentException();
6)     }
7)     hourlySalary = x;
8)   }
9)   public static void main(String[] args)
10)  { HelloWorld fu = new HelloWorld();
11)    fu.setSalary(2.50);
12)  }
13)}
```

```
Exception in thread "main"
java.lang.IllegalArgumentException
at HelloWorld.setSalary(HelloWorld.java:5)
at HelloWorld.main(HelloWorld.java:11)
```

16

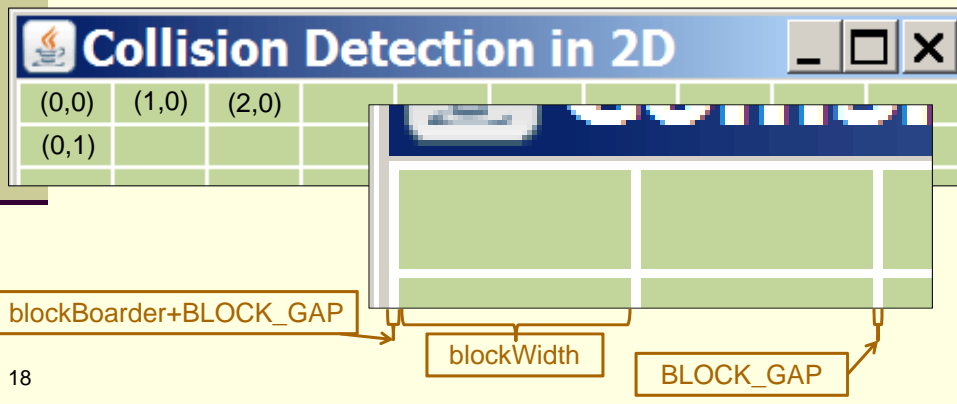
Improved: IllegalArgumentException()

```
private double hourlySalary;
public void setSalary(double x)
{ if (x<5.25)
  { throw new IllegalArgumentException("Below minimum Wage");
  }
  if (x>1000)
  { throw new IllegalArgumentException("You are nuts!");
  }
  hourlySalary = x;
}
public static void main(String[] args)
{ HelloWorld fu = new HelloWorld();
  double[] x = {6.00, -50000.00, 5.00, 50000.00};
  for (int i=0; i<x.length; i++)
  { try
    { fu.setSalary(x[i]);
    }
    catch (Exception e)
    { System.out.println(x[i]+" " + e.getMessage());
    }
  }
}
```

17

Block Drawing Assumptions

- The code snippets on the next few pages assumes a particular block layout and row, column numbering system.
- If your block layout is different, then you will need to modify this code.



18

BreakoutDraw: columnToPixelLeft()

```
private int columnToPixelLeft(int column) ☒
```

Given a column number between 0 and BLOCK_COLUMNS-1, this method returns the x-coordinate of the upper left corner of block.

Responsibilities:

1. If input column is out of range, then
`throw new IllegalArgumentException()`
2. Return the correct value.

```
private int columnToPixelLeft(int column)
{ if (column < 0 || column >=BLOCK_COLUMNS)
  throw new IllegalArgumentException();

  return column*(blockWidth+BLOCK_GAP) +
    blockBoarder+BLOCK_GAP;
}
```

19

BreakoutDraw: pixelToColumn(int x)

```
private int pixelToColumn(int x) ☒
```

Given a pixel value for the x-coordinate, this method:

1. Returns the column number that pixel fall within *if the pixel value is inside one of the columns.*
2. Returns -1 if the pixel value is not inside a column (either off the view area or within a gap between columns).

```
private int pixelToColumn(int x)
{ if (x< blockBoarder+BLOCK_GAP) return -1;
  if (x>(blockWidth+BLOCK_GAP)*BLOCK_COLUMNS return -1;

  ☒ //Determine whether x is in a vertical gap.
  column = (x - blockBoarder-BLOCK_GAP)
    / (blockWidth+BLOCK_GAP);
  return column;
}
```

20

Determine Whether x is in a Vertical Gap.

For an example, let `BLOCK_GAP = 2`, `blockWidth = 3` and `blockBoarder=0`.

Find an equation that identifies the gaps.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|
| | | X | X | X | | | X | X | X | | | X | X | X | | | X |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |

```
int BLOCK_GAP = 2;
int blockWidth = 3;
for (int x=0; x<=17; x++)
{ if (x % (blockWidth+BLOCK_GAP) < BLOCK_GAP)
  { System.out.println(x +": GAP");
  }
  else System.out.println(x+": Block")
}
```

21

Reproducing a Run State

- In most cases, each instance of a program should have ONLY ONE instance of `Random()`.
- When you notice a bug, you often want to reproduce that bug while watching the code, events, control flow and data more closely.
- Therefore with running with a random seed, it is helpful to always get and display the first random long value in the sequence, then use this to reset the random seed.

```
1) rand = new Random();
2) long seed = rand.nextLong();
3) rand.setSeed(seed);
4) System.out.println("Random seed: "+seed);
5) //rand.setSeed(-4377093108882599865L);
```

22