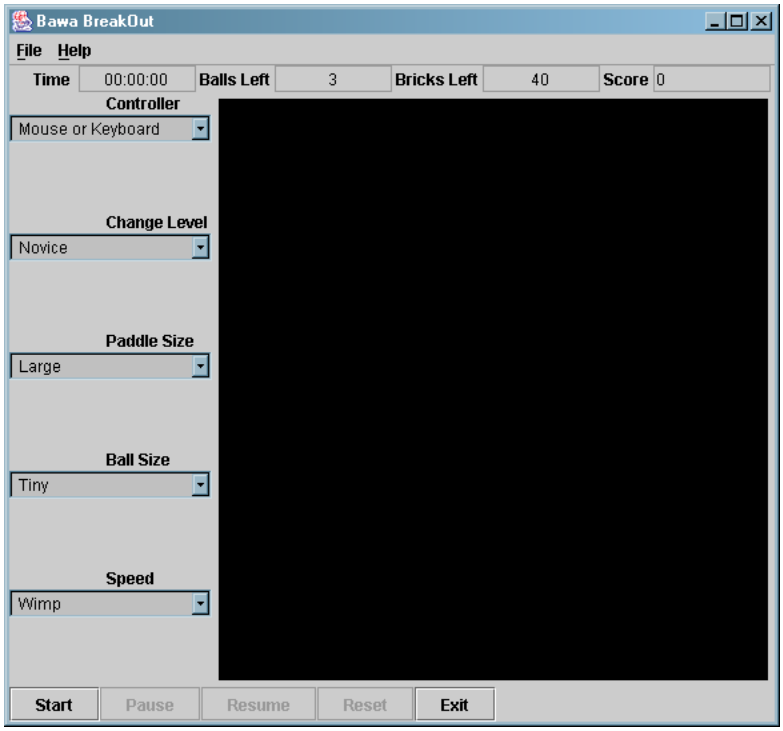
***Opening Screen***




**Choices :**

Change Controller --- use Mouse or Keyboard to control the paddle (NOTE:- to enable paddle movement with the keyboard it is required to click with the mouse at least once in the game panel);

Change Level --- Two levels exist "Novice" – where the game is played without a deflector.
"Advanced" – where the game has deflector which increases speed of the ball upon collision as well as changes the direction randomly

Once the level has been set it cannot be changed by the user during the process of the game. If the user has selected "Novice" and manages to survive for more than 5 minutes in the game then the level is automatically raised to "Advanced" and the deflector is set into play.

Paddle Size -- Three possible sizes for the Paddle – "Small", "Medium" and "Large" – and can be changed dynamically during a game.

Ball Size -- Three possible sizes for the ball – "Small", "Medium" and "Huge" – can be changed dynamically.

Speed -- Three possible speed settings for the ball – "Wimp", "Av Jo" and "Blitz" – can change dynamically.

**Set the game to play**

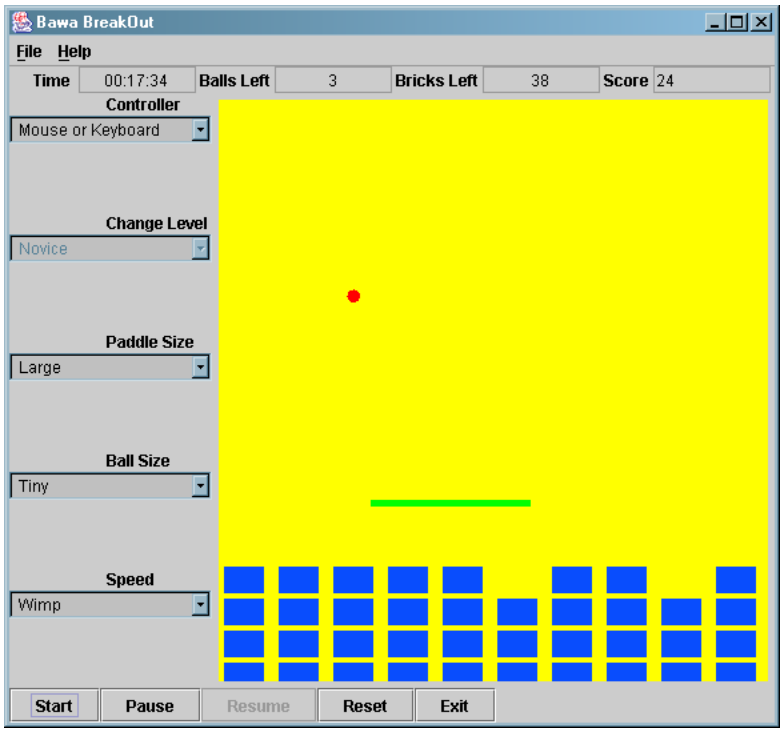
**Scoring :**

1. Each **blocking** of the ball with the **paddle** **+10**
2. Ball **hits the brick** **-5**
3. Survival of every 2 seconds **+10**

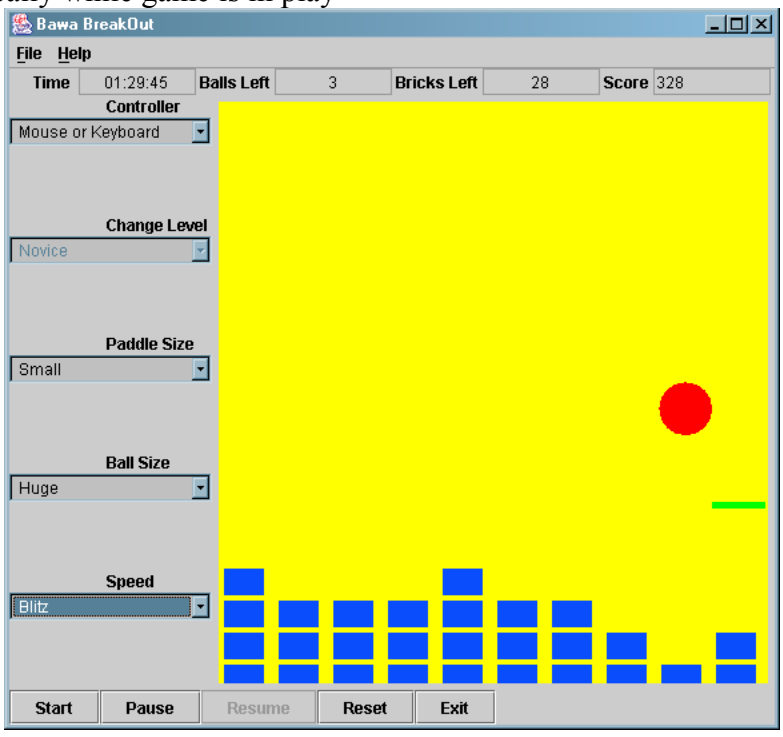
The **game is over** when

1. There are no balls left
2. There are no bricks left
3. The player survived for 20 minutes without losing all items

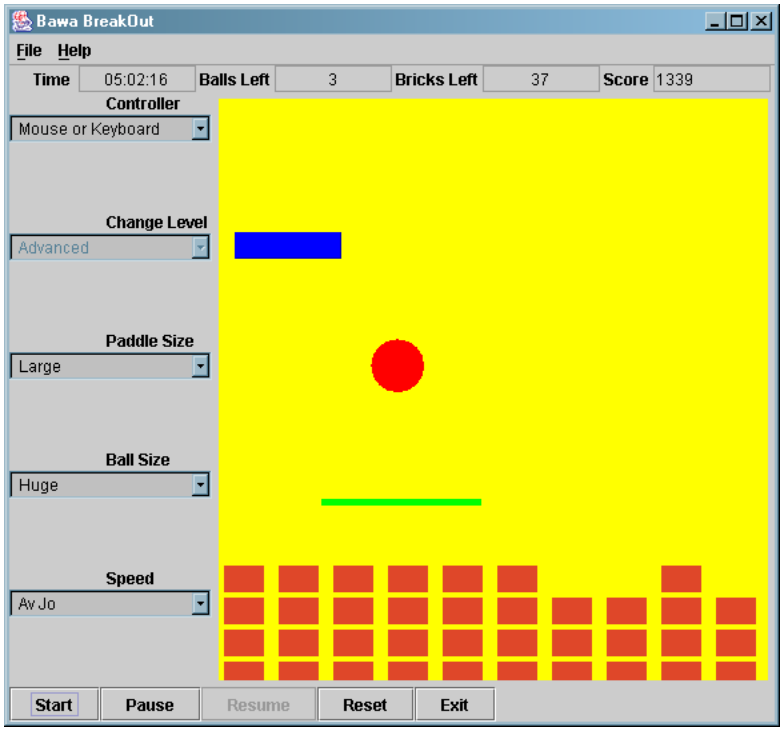
Game set into play



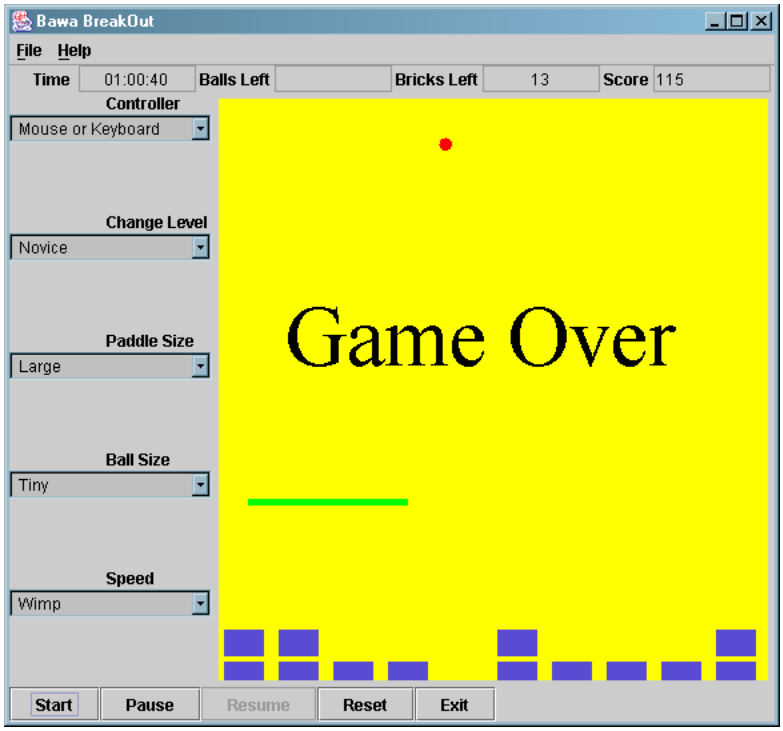

Changing choices dynamically while game is in play



Survival of 5 minutes changes level to **Advanced** (Note the player dynamically changed the paddle size)

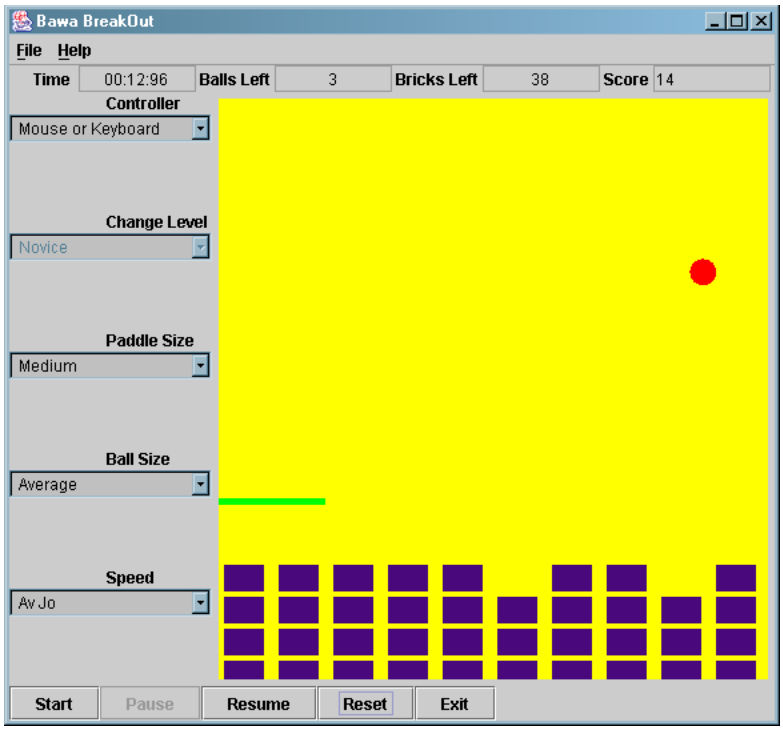



The Game is over --- no balls left

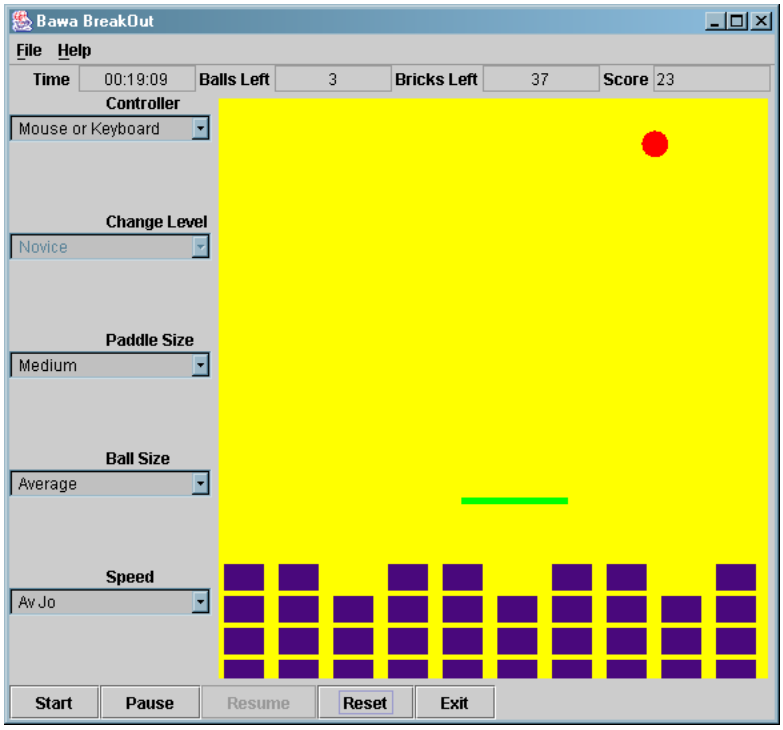


BUTTON CONTROLS  -- the button controls have been devised so as to ensure exclusivity of operations. Initially only pause is enaled --- hhowever on pressing pause, the **Resume** button gets enabled and all threads are interrupted

**Pause**

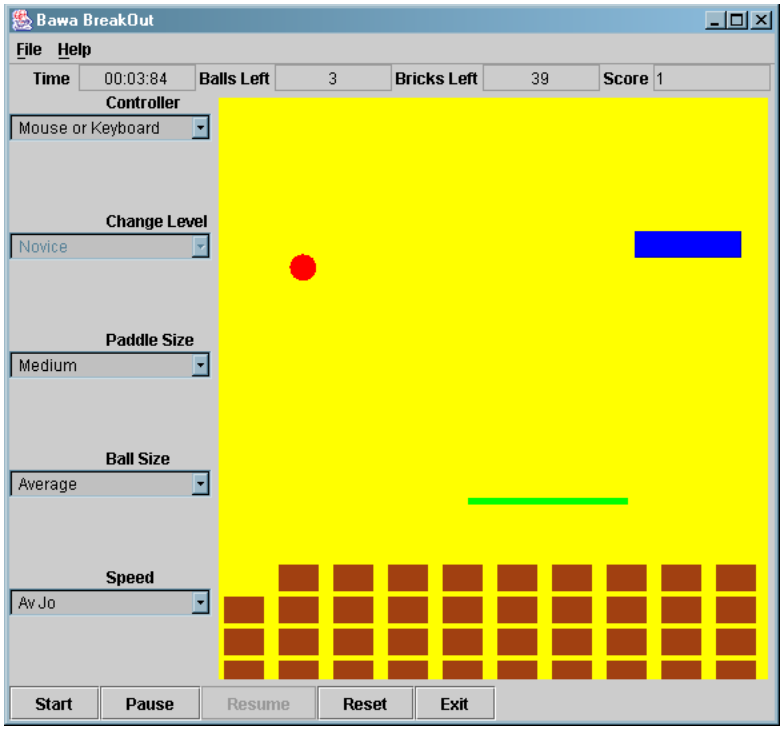



**Resume**




(snapshot taken after a delay)

**Reset**




(snapshot taken after a delay)

**The Java files:**

To get this game up and running compile the files in order: "***javac*** FileName.java"

1. **ball.java**
2. **brick.java**
3. **Paddle.java**
4. **Deflector.java**
5. **MyBoard.java**
6. **breakOutUI.java**

To run the game : ***java*** **breakOutUI**

**Have Phun!**

**THE CODE**

**ball.java**

```
/* author N Medhora*/
import java.util.Vector;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import java.io.*;
import java.util.*;
import java.lang.Thread;

public class ball extends Thread implements Runnable {
   private int oldx = 0;
   private int oldy = 0;
   private int dx = 3;
   private int dy = 3;

   private Canvas     box;
   private int                xpos;
   private int                ypos;
   private int                width;
    int                height;
   private int                xvel;
   private int                yvel;
   private Color      color = Color.yellow;
   private Paddle     p;
   private int                velSave;


   public ball(int xpos, int ypos, int xvel, int yvel, int width, int height,
            Canvas canvas1,  Color color) {
      setDaemon (true);
      this.xpos        = xpos;
      this.ypos        = ypos;
      this.xvel        = xvel;
      this.yvel        = yvel;
      this.width       = width;
      this.height = height;
                       this.box   = canvas1;
      this.color       = color;
      velSave          = Math.abs(xvel);
   }

  public void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D)g;
      g2.setColor(color);
      g2.fillOval(xpos, ypos, width, height);
   }
```

```
public boolean moveBall() {
    oldx = xpos;
    oldy = ypos;

    xpos += xvel;
    ypos += yvel;

    Dimension d = box.getSize();
    if (xpos < 0 || xpos > d.width - width) {
            xvel = -xvel;
    }

    if (ypos < 0) {   // || ypos > d.height - YSIZE) {
            yvel = -yvel;
    }

    if(ypos > d.height + 100) {
                                    ypos = (int) Math.floor(Math.random() * 100D);
                                    return(true);
  }
  if (xpos < 0) {
            xpos = 0;
            }
            if (xpos + width > d.width) {
                        xpos = d.width - width;
 }
  return(false);
}

  public void setColor() {
     color = new Color((int)Math.floor(Math.random() * 250D),
                                                (int)Math.floor(Math.random() * 250D),
                                                (int)Math.floor(Math.random() * 250D));
  }

  public void setXvel(int xvel) {
     this.xvel = xvel;
     velSave = Math.abs(xvel);
  }

  public void setYvel(int yvel) {
     this.yvel = yvel;
  }

  public int getXvel() {
     return this.xvel;
  }

  public int getYvel() {
     return this.yvel;
  }

  public int getXpos() {
     return this.xpos;
  }

  public void setXpos(int ypos) {
     this.xpos = xpos;
  }

  public int getYpos() {
     return this.ypos;
  }

  public void setYpos(int ypos) {
     this.ypos = ypos;
  }
```

```
    public void setColor(Color color) {
        this.color = color;
    }

    public void setSize(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public void setVel(int xvel, int yvel) {
        this.xvel = xvel;
        this.yvel = yvel;
        velSave = Math.abs(xvel);
    }
    public void setPos(int xpos, int ypos) {
            this.xpos = xpos;
            this.ypos = ypos;
    }

    public int getSize() {
            return width;
    }

    public Color getColor() {
            return      color;
    }
}
```

**brick.java**

```
import javax.swing.*;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.*;

public class brick
{
// fields
        int xpos;
   int ypos;
   int width;
   int height;
   int lasthit = 1;
   Color color;
   int active;
   int yStart;

   /* ctor */
   public brick (      int xpos, int ypos, int width, int height, Color color, int yStart) {
      this.xpos = xpos;
      this.ypos = ypos;
      this.width = width;
      this.height = height;
      this.color = color;
      this.yStart = yStart;
      this.active = 1;
   }


   void setColor(Color color) {
      this.color = color;
   }

   public boolean collision(int ballxpos, int ballypos, int ballwidth, int ballheight) {

        if ((active != 0) &&  (ballypos < ypos + height) && (ballypos + ballheight > ypos)
                        && (ballxpos < xpos + width) && (ballxpos + ballwidth > xpos)) {
                active = 0;
                return(true);
        }
                return(false);
   }

   public void paintComponent(Graphics g)  {
        Graphics2D g2 = (Graphics2D)g;

      if (active != 0) {
      g2.setColor(color);
      g2.fillRect(xpos, ypos, width, height);
     }
   }
}
```

**Paddle.java**

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.Dimension;

import java.util.*;
public class Paddle {
   private int          xpos;
    int      ypos;
   private int          width;
    int      height;
   private Color      color;
   private int          xmax;
   private int          xmin;
   private int          ymin;
   private int          ymax;

   public Paddle(Paddle p) {
      this.ypos                  = p.getYpos();
      this.width                 = p.getWidth();
      this.height                = p.getHeight();
      this.color                 = Color.green;
      this.xmin                  = p.getXmin();
      this.xmax                  = p.getXmax();
      this.ymin                  = ypos;
      this.ymax                  = ypos - 40;
   }

   public Paddle(int width, int height, int ypos, int xmin, int xmax) {
      this.ypos                  = ypos;
      this.width                 = width;
      this.height                = height;
      this.color                 = Color.green;
      this.xmin                  = xmin;
      this.xmax                  = xmax;
   }

   public void setColor() {
      this.color = new Color(   (int)Math.floor(Math.random() * 250D),
                                                (int)Math.floor(Math.random() * 250D),
                                                (int)Math.floor(Math.random() * 250D)  );
   }

   public void setColor(Color color) {
      this.color = color;
   }

   public Color getColor() {
      return this.color;
   }

   public int collision (int ballxpos, int ballypos, int ballwidth, int ballheight) {
                        if( (ballxpos + ballwidth        < xpos || ballxpos > xpos + width) ||
                            (ballypos + ballheight       < ypos || ballypos > ypos + height)        )
        return(-1);
       else {
          return(ballxpos - xpos);
       }
   }

   public void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D) g;


      g2.setColor(color);
      g2.fillRect(xpos, ypos, width, height);
   }
```

```
    public void setYpos(int y) {
            if (y > ymin) {
                    this.ypos = ymin;
            } else if (y < ymax) {
                    this.ypos = ymax;
            } else {
                    this.ypos = y;
            }
}

    public void setXpos(int x) {
            if (x < xmin) {
                    this.xpos = xmin;
       } else if (x > xmax - width) {
                    this.xpos = xmax - width + 10;
       } else {
                this.xpos = x;
      }
    }

    public void setWidth(int width) {
       this.width = width;
    }

    public int getXpos() {
       return this.xpos;
    }

    public int getYpos() {
       return this.ypos;
    }

    public int getWidth() {
       return this.width;
    }

    public int getHeight() {
       return this.height;
    }

    public int getXmax() {
       return this.xmax;
    }

    public int getXmin() {
       return this.xmin;
    }

}
```

**Deflector.java**

```
import java.util.*;
import java.awt.*;
import java.applet.*;

public class Deflector extends Thread implements Runnable{
   private int xpos;
   private int ypos;
   private int xvel;
   private int yvel;
   private int velSave;
   private int width;
   private int height;
   private Color color;
   private Canvas box;
   private boolean raised = true;

   /** ctor **/
   public Deflector(int width, int height, int ypos, Canvas box) {
            setDaemon (true);
      this.color = Color.blue;
      this.ypos = ypos;
      this.xpos = 250;
      this.xvel = 4;
      this.yvel = 0;
      this.width = width;
      this.height = height;
      this.box = box;
   }

   public boolean  collision(int ballxpos,int ballypos, int ballwidth, int ballheight) {
      if(    (ballxpos + ballwidth  < xpos || ballxpos > xpos + width) ||
             (ballypos + ballheight < ypos || ballypos > ypos + height))
        return false;
       else
        return true;
   }

   public void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D) g;
      g2.setColor(color);
      g2.fill3DRect(xpos,ypos,width,height, raised);
   }

   public void moveDeflector() {
     int oldx = xpos;
     int oldy = ypos;

     Dimension d = box.getSize();

               if (xpos > d.width - width) {
                        xpos = d.width - width;
               }
     if (xpos >= (d.width - width) || xpos < 0) {
                        xvel *= -1;
     }
     xpos += xvel;
   }

   public void setVel(int xvel, int yvel) {
      this.xvel = xvel;
      this.yvel = yvel;
      velSave = Math.abs(xvel);
   }

   public void setXpos(int x) {
       this.xpos = x;
   }
```

```
  public void setWidth(int width) {
        this.width = width;
    }

    public int getXpos() {
        return this.xpos;
    }

    public int getWidth() {
        return this.width;
    }
}
```

**MyBoard.java**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.io.*;
import java.util.*;

public class MyBoard extends Thread implements Runnable, KeyListener, MouseMotionListener{

  int constant = 20;                // paddle moves by this amount
  int level = 1;

  private brick       brickArray[];
  private Canvas      box;
  Paddle              thePaddle;
  Deflector           theDeflector1;
  ball                b;

  private long        tartTime;
  private int         seconds;
  private int         minutes;

  private Color       background;
  boolean gameOver            = false;
  boolean startNow             = false;
  int     ballsLeft           = 3;
  int    numberOfLines        = 4;
  int     bricksPerLine       = 10;
  int     brickWidth          = 30;
  int     brickHeight         = 20;
  int     bricksLeft          = 40;
  int     score               = 0;

  private boolean             paused;
  private Canvas              offscreen;
  private BufferedImage       bImage = null;
  private Graphics2D          gbuffer = null;
  private boolean             useMouse = true;
  private JTextField          tf_BallsLeft;
  private JTextField          tf_BricksLeft;
  private JTextField          tf_Time;
  private JTextField          tf_Score;

  private Choice              ch_Level;

          // ctor
  public MyBoard(Canvas theCanvas, ball b, Paddle thePaddle, Deflector theDeflector1, JTextField tf_BallsLeft, JTextField tf_BricksLeft, JTextField
                        tf_Time, JTextField tf_Score, Choice ch_Level) {
                    setDaemon(true);
                    this.box            = theCanvas;
                    this.gameOver       = false;
                    this.background     = Color.yellow;
                    this.b              = b;
                    this.thePaddle      = new Paddle(thePaddle);
                    this.theDeflector1  = theDeflector1;
                    this.setUpBricks(numberOfLines, bricksPerLine, brickWidth, brickHeight);
                    this.level = 2;
                    this.bImage = new BufferedImage(theCanvas.getSize().width,
                                                    theCanvas.getSize().height,
                                                  BufferedImage.TYPE_INT_RGB);

          Graphics2D gbuffer = bImage.createGraphics();
                    theCanvas.addKeyListener(this);
                    theCanvas.addMouseMotionListener(this);
                    theCanvas.isFocusable();
                    theCanvas.requestFocus();
```

```
                Image cursorImage = Toolkit.getDefaultToolkit().createImage( "NoCursor.gif" );       Point cursorPoint = new Point( 0,0 );

                Cursor NoCursor = Toolkit.getDefaultToolkit().createCustomCursor( cursorImage, cursorPoint, "NoCursor" );
                box.setCursor( NoCursor );
                this.tf_BallsLeft = tf_BallsLeft;

                tf_BallsLeft.setText(Integer.toString(ballsLeft));
                this.tf_BricksLeft = tf_BricksLeft;
                tf_BricksLeft.setText(Integer.toString(bricksLeft));
                this.tf_Time = tf_Time;
                startTime = new Date().getTime();
                tf_Time.setText(setTime());
                this.tf_Score = tf_Score;
                tf_Score.setText(Integer.toString(score));
                this.ch_Level = ch_Level;

}

public void pause(int time) {
    try {
       Thread.sleep(time);
       return;
    }
    catch(InterruptedException _ex) {
       return;
    }
 }


public void run() {
            ch_Level.setEnabled(false);
            startNow = true;
    try {
       while(!gameOver) {
          if (!paused) {
                   if (b.moveBall()) {
                            ballsLeft -= 1;
                            tf_BallsLeft.setText(Integer.toString(ballsLeft));
                                if (ballsLeft < 0) {
                                        gameOver = true;
                                        ch_Level.setEnabled(true);
                                        box.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
                            }
       }
          try {
                   theDeflector1.moveDeflector();
         } catch (NullPointerException e) {};
          paintComponent();
                            tf_Time.setText(setTime());
                           Thread.sleep(1L);
       }
           }
    } catch(InterruptedException interruptedexception) { }
 }


public void paintComponent() {
          Graphics g = box.getGraphics();
          Graphics2D g2 = (Graphics2D) g;

    Dimension d = box.getSize();
    gbuffer = bImage.createGraphics();

    gbuffer.setColor(background);
          gbuffer.fillRect(0,0,d.width,d.height);   // game playing rect
          int i = 0;
          boolean brickCollision          = false;
          boolean paddleCollision         = false;
          boolean deflectorCollision      = false;
          int paddleXCollision            = 0;
          int paddleYCollision            = 0;
```

```
boolean theCollision;

		if (gameOver) {
			gbuffer.setColor(Color.black);
			gbuffer.setFont(new Font("Times New Roman", Font.PLAIN, 64));
			gbuffer.drawString("Game Over",	50, 200);
			tf_BallsLeft.setText(" ");
			this.interrupt();
		}
		while(brickArray[i] != null) {
				if (!brickCollision) {
					theCollision = brickArray[i].collision(b.getXpos(), b.getYpos(), b.getSize(), b.getSize());
				  if (theCollision) {
					score -= 5;
					tf_Score.setText(Integer.toString(score));

					brickCollision = true;
					b.setYvel(b.getYvel() * -1);
					bricksLeft -= 1;
					tf_BricksLeft.setText(Integer.toString(bricksLeft));

					if (bricksLeft == 0) {
					    gameOver = true;
					}
				  }
			    }
					brickArray[i].paintComponent(gbuffer);
			i++;
		}

		paddleXCollision = thePaddle.collision(b.getXpos(), b.getYpos(), b.getSize(), b.getSize());
		if(paddleXCollision != -1) {
			if(  b.getYpos() > thePaddle.height + thePaddle.ypos) {
				score = score;			// no change in score if ball hits paddle from below
			}
			else
			score += 10;
			tf_Score.setText(Integer.toString(score));

			paddleCollision = true;
			b.setXvel((paddleXCollision - (thePaddle.getWidth() / 2)) / 10);
			b.setYvel( b.getYvel() *  -1)  ;
		}

		try {
		 if(theDeflector1.collision(b.getXpos(), b.getYpos(), b.getSize(), b.getSize())) {
			deflectorCollision = true;
			b.setXvel( b.getXvel() + 1);
			b.setYvel( b.getYvel() + 1);
		  }
		 } catch (NullPointerException e) {};

		gbuffer.setColor(Color.red);
		b.setColor(Color.red);
		b.setPos(b.getXpos(), b.getYpos());
		b.setSize(b.getSize(),b.getSize());
		b.paintComponent(gbuffer);

		thePaddle.paintComponent(gbuffer);

		try {
			theDeflector1.paintComponent(gbuffer);
		} catch (NullPointerException e) {};

		g2.drawImage( bImage,null, null);
}
```

```
  public void setUpBricks(int numberOfLines, int bricksPerLine, int brickWidth, int brickHeight) {
     int z = 0;
     int yincr = 4;
     int xstart = 4;
     int ystart = 2;
     int xcnt = 4;
     int ycnt = 350;
     this.numberOfLines = numberOfLines;
     this.bricksPerLine = bricksPerLine;
     this.brickWidth = brickWidth;
     this.brickHeight = brickHeight;
          int xincr = (box.getSize().width / bricksPerLine) - brickWidth;
     Color color;

     brickArray = new brick[numberOfLines * bricksPerLine + 10];
     int r = (int)Math.floor(Math.random() * 256D);
     int g = (int)Math.floor(Math.random() * 84D);
     int b = (int)Math.floor(Math.random() * 256D);
     for(int i = 0; i < numberOfLines; i++) {
        color = new Color(r, g, b);
        for(int x = 0; x < bricksPerLine; x++) {
           brickArray[z] = new brick(xcnt, ycnt, brickWidth, brickHeight, color, ystart);
           xcnt += xincr + brickWidth;
                    z++;
        }
        ycnt += yincr + brickHeight;
        xcnt = xstart;
     }
  }

  public void keyReleased(KeyEvent event) {
           }

  public void keyTyped(KeyEvent event) {
           }

  public void keyPressed(KeyEvent event) {

    int keyCode = event.getKeyCode();

    if( keyCode == KeyEvent.VK_LEFT ) {
      this.thePaddle.setXpos (thePaddle.getXpos() - constant);
    }
    else if( keyCode == KeyEvent.VK_RIGHT) {
          this.thePaddle.setXpos (thePaddle.getXpos() + constant);
   }
  }

   public void mouseMoved(MouseEvent e) {
                    if (useMouse) {
                    //          this.thePaddle.setYpos(e.getY());  // in case we want the paddle to move in x-y space
                              if (e.getX() < 0) {
                                        this.thePaddle.setXpos(0);

                              } else if (e.getX() > box.getSize().width) {
                                        this.thePaddle.setXpos(box.getSize().width);
                              } else {
                                        this.thePaddle.setXpos(e.getX());
                              }
            }
  }

  public void mouseDragged(MouseEvent e) {
  }


public ball getBall() {
          return(b);
  }
```

```
public void setBall(ball b) {
            this.b = b;
  }
  public Paddle getPaddle() {
            return (thePaddle);
            }

  public void setPaddle(Paddle thePaddle) {
            this.thePaddle = thePaddle;
            }

public void setUseMouse(boolean useMouse) {
            this.useMouse = useMouse;
  }

  public void setPaused(boolean paused) {
            this.paused = paused;
  }

  public String setTime() {

      Date date = new Date();
      long thisTime = date.getTime();
      long diffTime = thisTime - startTime;
      long hundOfSec = diffTime / 10L;
      if(hundOfSec > 99L) {
         startTime = thisTime;
         hundOfSec = 0L;
         seconds++;
         score += 2;
         tf_Score.setText(Integer.toString(score));
      }
      if(seconds > 59) {
         seconds = 0;
         minutes++;
     if (minutes == 5) {
            theDeflector1          = new Deflector( 80, 20, 100, box);       //set to advanced mode
            ch_Level.select(" Advanced ");
      }
    if (minutes == 20) {
            gameOver = true;
            gbuffer.setColor(Color.black);
            gbuffer.setFont(new Font("Times New Roman", Font.PLAIN, 44));
            gbuffer.drawString("You Won",            50, 300);
      }

   }

  String sec = Long.toString(seconds);
      if(seconds <= 9) {
         sec = "0" + sec;
      }

  String hund = Long.toString(hundOfSec);
      if(hundOfSec <= 9L) {
         hund = "0" + hund;
      }

  String min = Long.toString(minutes);
      if(minutes <= 9) {
         min = "0" + min;
      }

  return min + ":" + sec + ":" + hund;
   }
}
```

**BreakOutUI.java**

```
/* breakOutUI.java -- the UI */

import java.awt.*;
import java.awt.event.*;
import java.awt.Graphics2D;
import javax.swing.*;
import java.io.*;
import java.util.*;
import javax.swing.JOptionPane;


public class breakOutUI extends JFrame implements Runnable {

        // ctor
        public breakOutUI() {
                this.run();
                this.initUI();
    this.startNow = true;
  }

        public static final int WIDTH          = 500;
        public static final int HEIGHT         = 600;
        int constant = 20;

        // fields in the object for UI controls //
        // bottom most panel - Buttons for all standard operations //
  private  JButton     start;
  private  JButton     pause;
  private  JButton     resume;
  private  JButton    exit;
  private  JButton    reset;

        // panel to hold the buttons //
  private  JPanel      buttonPanel;

        // panel to hold the game
  private  JPanel              gamePanel;
  private  Canvas    myCanvas;

        // panel for time related activity
  private  JPanel              timerPanel;
        // labels and textfields
  private  JLabel              la_Time;
  private  JTextField          tf_Time;
  private  JLabel              la_BallsLeft;
  private  JTextField          tf_BallsLeft;
  private  JLabel              la_BricksLeft;
  private  JTextField          tf_BricksLeft;
  private  JLabel              la_Score;
  private  JTextField          tf_Score;

        // status panel
  private  JPanel              statusPanel;
  /// user customizable settings //
  private  JLabel              la_Level;
  private  Choice              ch_Level;                  // advanced, novice
  private  int                 level;
                                                                  // including just in case we wish to add such options later
 // private          JLabel              la_NumberOfBricks;
 // private          Choice              ch_NumberOfBricks;          // changes the number of bricks the user would like to play with

 // private          JLabel              la_BrickSize;
 // private  Choice              ch_BrickSize;       // changes the size of the bricks

  private  JLabel              la_useMouse;
  private  Choice              ch_useMouse;
```

```
private  JLabel              la_BallSize;
private  Choice              ch_BallSize;        // changes the size of the ball


private  JLabel              la_PaddleSize;
private  Choice              ch_PaddleSize;      // changes the size of the paddle

private  JLabel              la_Speed;
private  Choice              ch_Speed;                    // changes the speed of the ball | s

        // actual object //

 private int                 brickWidth = 2;
 private int                 brickHeight = 1;
 private int                 bricksPerLine;
 private int                 numberOfLines;
 private int                 arraySize;

 private int                 ballYstart;
 private int                 ballXstart;
 Color   color = Color.yellow;
 Color   background = Color.black;
 Paddle paddle;
 private int                 paddleWidth = 120;
 private int                 paddleXmax;

 Deflector deflector1;

 ball     ball;
 private int                 ballWidth = 10;
 private int                 ballHeight = 10;

 private Image               offscreenImg;
 private Graphics  offscreenGraphics;
 private Thread              thread;
 private int                 z;

 private Vector              v = new Vector();
 Enumeration e;
 private brick               brickArray[];

 private long                startTime;
 private Date                date;

 private boolean   startNow;
 private int                 xexit;

 private int                 oldVsize;

 private Font                f;
 private FontMetrics fm;
 private int                 fs;
 private int                 xpos;
 private int                 ypos;
 private int                 width;
 private int                 height;
 private int                 xvel;
 private int                 yvel;

 private Paddle              p;

 private int velSave;

 private MyBoard  theBoard;

 boolean             useMouse = true;
```

```
/**********PRIVATE METHODS ****************/
/*   Initialize UI controls   */
private  void  initUI() {

Container   cp;

   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   this.setTitle("Bawa BreakOut");

   this.setSize(WIDTH, HEIGHT);
   this.setResizable(true);

   cp = this.getContentPane();

  // Setup Menus
          // Create toolbar
          JMenuBar menuBar = new JMenuBar();
          setJMenuBar (menuBar);

          // Create a menu labeled File, accelerator F
          JMenu file = new JMenu ("File");
          file.setMnemonic (KeyEvent.VK_F);
          JMenuItem item;
          // Create a menu item Exit, accelerator x
          // Have it call doCloseCommand when selected
          file.add (item = new JMenuItem ("Exit"));
          item.setMnemonic (KeyEvent.VK_X);
          item.addActionListener (new ActionListener() {

          public void actionPerformed (ActionEvent e) {
          doExitCommand (0);
          }
          });

          // Add file menu to menu bar
          menuBar.add (file);

          // Create a menu labeled Help, accelerator H
          JMenu help = new JMenu ("Help");
          help.setMnemonic (KeyEvent.VK_H);

          // Create a menu item About, accelerator A
          // Have it call doAboutCommand when selected
          help.add (item = new JMenuItem ("About"));
          item.setMnemonic (KeyEvent.VK_A);
          item.addActionListener (new ActionListener() {
          public void actionPerformed (ActionEvent e) {
          //doAboutCommand();
          }
          });

          // Add help menu to menu bar
          menuBar.add (help);


// user customizable settings //
/// instantiate the panels individually first//
this.statusPanel                =           new JPanel();


/// instantiate components for panels and add resp ///
/// statusPanel ///
            this.la_Level       =           new JLabel(" Change Level ");
  this.ch_Level    =           new Choice();
                    ch_Level.add(" Advanced ");
                    ch_Level.add(" Novice ");
```

```
/*                      // in case we wish to use these options
    this.la_NumberOfBricks  =           new JLabel(" Bricks # ");
    this.ch_NumberOfBricks =            new Choice();
                        ch_NumberOfBricks.add(" 40 ");
                        ch_NumberOfBricks.add(" 50 ");
                        ch_NumberOfBricks.add(" 60 ");

            this.la_BrickSize               =           new JLabel(" Brick Size ");
            this.ch_BrickSize               =   new Choice();
                                ch_BrickSize.add(" Small ");
                                ch_BrickSize.add(" Medium ");
                                ch_BrickSize.add(" Large ");
            */

    this.la_useMouse                        =           new JLabel(" Controller  ");
    this.ch_useMouse                                    =           new         Choice();
                        ch_useMouse.add(" Mouse or Keyboard ");
              ch_useMouse.add(" Keyboard ");

    this.la_PaddleSize                      =           new JLabel(" Paddle Size ");
            this.ch_PaddleSize                          =   new Choice();
                                ch_PaddleSize.add(" Small ");
                                ch_PaddleSize.add(" Medium ");
                                ch_PaddleSize.add(" Large ");

    this.la_BallSize                        =           new JLabel(" Ball Size ");
    this.ch_BallSize                                    =           new Choice();
                        ch_BallSize.add(" Tiny ");
                        ch_BallSize.add(" Average ");
                        ch_BallSize.add(" Huge ");

    this.la_Speed    =          new JLabel(" Speed ");
            this.ch_Speed       =           new Choice();
                                ch_Speed.add(" Blitz ");
                                ch_Speed.add(" Av Jo ");
                                ch_Speed.add(" Wimp ");

            this.ch_Speed.select(" Wimp ");
            this.ch_BallSize.select(" Tiny ");
            this.ch_PaddleSize.select(" Large ");
            this.ch_Level.select(" Novice ");

          // add items to the statusPanel //
          this.statusPanel.setLayout(new BoxLayout(this.statusPanel,BoxLayout.Y_AXIS));
          this.statusPanel.add(la_useMouse);
          this.statusPanel.add(ch_useMouse);

          this.statusPanel.add( la_Level );
          this.statusPanel.add( ch_Level );
                        // in case we wish these choices
 //       this.statusPanel.add( la_NumberOfBricks );
 //       this.statusPanel.add( ch_NumberOfBricks );
 //       this.statusPanel.add( la_BrickSize );
 //       this.statusPanel.add( ch_BrickSize );
          this.statusPanel.add( la_PaddleSize );
          this.statusPanel.add( ch_PaddleSize );
          this.statusPanel.add( la_BallSize );
          this.statusPanel.add( ch_BallSize);
          this.statusPanel.add( la_Speed );
          this.statusPanel.add( ch_Speed );
          this.statusPanel.setSize( 540, 100);

  this.buttonPanel              =           new JPanel();
    this.start          =           new         JButton(" Start ");
    this.pause          =           new JButton(" Pause ");
```

```
  this.resume      =          new JButton(" Resume ");
  this.exit        =          new JButton(" Exit ");
  this.reset       =          new JButton(" Reset ");
        resume.setEnabled(false);
        pause.setEnabled(false);
        reset.setEnabled(false);



        // add items to buttonPanel //
        this.buttonPanel.setLayout(new BoxLayout(this.buttonPanel,BoxLayout.X_AXIS));
        this.buttonPanel.add( start  );
        this.buttonPanel.add( pause  );
        this.buttonPanel.add( resume );
        this.buttonPanel.add( reset );
        this.buttonPanel.add( exit   );

        this.buttonPanel.setSize( 440, 60 );


this.timerPanel                =          new JPanel();

        // ADD ITEMS TO THE TIMERPANEL //
        this.timerPanel.setLayout( new BoxLayout(this.timerPanel,BoxLayout.X_AXIS));
        this.la_Time          = new JLabel("      Time  ");
        this.tf_Time          = new JTextField(4);
                tf_Time.setEditable(false);
                tf_Time.setHorizontalAlignment(JTextField.CENTER);

        this.la_BallsLeft     =          new JLabel(" Balls Left ");
        this.tf_BallsLeft     =          new JTextField(4);
                tf_BallsLeft.setEditable(false);
                tf_BallsLeft.setHorizontalAlignment(JTextField.CENTER);

        this.la_BricksLeft    =          new JLabel(" Bricks Left ");
        this.tf_BricksLeft    =          new JTextField(4);
                tf_BricksLeft.setEditable( false );
                tf_BricksLeft.setHorizontalAlignment(JTextField.CENTER);

        this.la_Score                  =          new JLabel(" Score ");
        this.tf_Score                  =          new JTextField(4);
                tf_Score.setEditable(false);

        this.timerPanel.add( la_Time );
        this.timerPanel.add( tf_Time );

        this.timerPanel.add( la_BallsLeft );
        this.timerPanel.add( tf_BallsLeft );

        this.timerPanel.add( la_BricksLeft );
        this.timerPanel.add( tf_BricksLeft );

        this.timerPanel.add( la_Score );          // this is being added here as the timerPanel
        this.timerPanel.add( tf_Score );          // is essentially recording changes

        this.timerPanel.setSize( 50, 50);

        this.gamePanel                 =          new JPanel();
        myCanvas = new Canvas();
   myCanvas.setBounds(12, 36, 412, 436);
   myCanvas.setBackground(background);


        this.gamePanel.add(myCanvas);

this.gamePanel.setSize( 440, 440);
        for(bricksPerLine = myCanvas.getSize().width / brickWidth;
                ((brickWidth * bricksPerLine + bricksPerLine) - brickWidth) + 1 > myCanvas.getSize().width ;
                bricksPerLine--);
   offscreenImg = createImage(myCanvas.getSize().width, myCanvas.getSize().height);
```

```
    paddle          = new Paddle(paddleWidth, 5 , 300, 0, 400);
     ball           = new ball(ballXstart, ballYstart, 2, 2, ballWidth, ballHeight, myCanvas, color);
    deflector1      = new Deflector( 80, 20, 100, myCanvas);
    theBoard        = new MyBoard(myCanvas, ball, paddle, null, tf_BallsLeft,tf_BricksLeft, tf_Time, tf_Score, ch_Level);




  // ACTION LISTENERS //

  // start button
  this.start.addActionListener(new ActionListener()  {
            public void actionPerformed(ActionEvent evt) {
                    startAll();
      }
    });

  // pause button
  this.pause.addActionListener(new ActionListener()  {
            public void actionPerformed(ActionEvent evt) {
        theBoard.setPaused(true);
        pause.setEnabled(false);
        resume.setEnabled(true);
      }
    });
            // resume button
  this.resume.addActionListener(new ActionListener()  {
            public void actionPerformed(ActionEvent evt) {
                    try {
        pause.setEnabled(true);
        resume.setEnabled(false);
            theBoard.setPaused(false);
       } catch(IllegalMonitorStateException ilmse) {System.out.println("ILLEGAL");}
      }
    });

            // reset button
  this.reset.addActionListener(new ActionListener()  {
            public void actionPerformed(ActionEvent evt) {
                    theBoard.interrupt();
                    ball.setXpos(20);
                    theBoard = new MyBoard(myCanvas, ball, paddle, deflector1, tf_BallsLeft, tf_BricksLeft, tf_Time, tf_Score, ch_Level);
                    startAll();
                     }
    });

            // exit button
  this.exit.addActionListener(new ActionListener()  {
            public void actionPerformed(ActionEvent evt) {
        doExitCommand(0);
      }
    });

  // choice - use the mouse or keyboard
  this.ch_useMouse.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent evt) {
                    if(ch_useMouse.getSelectedItem().equals(" Mouse or Keyboard ")) {
                            theBoard.setUseMouse(true);
              } else {
                    theBoard.setUseMouse(false);
              }
  }
  });

            // choice - number of bricks - in case we wished this
/* this.ch_NumberOfBricks.addItemListener(new ItemListener()  {
                    public void itemStateChanged(ItemEvent evt) {
                            if (ch_NumberOfBricks.getSelectedItem().equals(" 40 ")){
                                    theBoard.setUpBricks(4,10, 30, 20);
                            }
```

```
                        else if (ch_NumberOfBricks.getSelectedItem().equals(" 50 ")){
                                theBoard.setUpBricks(5,10, 30, 20);
                        }
                        else if (ch_NumberOfBricks.getSelectedItem().equals(" 60 ")){
                                theBoard.setUpBricks(5,12, 30, 20);
                }
        }
});
*/


        // choice - ball size
this.ch_BallSize.addItemListener(new ItemListener()  {
                public void itemStateChanged(ItemEvent evt) {
                         if (ch_BallSize.getSelectedItem().equals(" Tiny ")) {
                                ballWidth = 10;
                                ballHeight = 10;
                        }
                        else if (ch_BallSize.getSelectedItem().equals(" Average ")) {
                                ballWidth = 20;
                                ballHeight = 20;
                        }
                        else if (ch_BallSize.getSelectedItem().equals(" Huge ")) {
                                ballWidth = 40;
                                ballHeight = 40;
                        }
                        theBoard.getBall().setSize(ballWidth, ballHeight);
                }
 });

        // choice - paddle size
this.ch_PaddleSize.addItemListener(new ItemListener()  {
                public void itemStateChanged(ItemEvent evt) {
                        if (ch_PaddleSize.getSelectedItem().equals(" Small ")) {
                                theBoard.getPaddle().setWidth(40);
                        }
                        else if (ch_PaddleSize.getSelectedItem().equals(" Medium ")) {
                                theBoard.getPaddle().setWidth(80);
                        }
                        else if (ch_PaddleSize.getSelectedItem().equals(" Large ")) {
                        theBoard.getPaddle().setWidth(120);
                  }
                }
        });

        // choice - brick size  -- in case we wished this
        /*
        this.ch_BrickSize.addItemListener(new ItemListener()  {
                public void itemStateChanged(ItemEvent evt) {
                        if (ch_BrickSize.getSelectedItem().equals(" Small ")){
                                theBoard.setUpBricks(theBoard.numberOfLines, theBoard.bricksPerLine, 30, 20);
                        }
                        else if (ch_NumberOfBricks.getSelectedItem().equals(" Medium ")){
                                theBoard.setUpBricks(theBoard.numberOfLines, theBoard.bricksPerLine, 40, 20);
                        }
                        else if (ch_NumberOfBricks.getSelectedItem().equals(" Large ")){
                                theBoard.setUpBricks(theBoard.numberOfLines, theBoard.bricksPerLine, 50, 20);
                }
        }
});
        */

        // choice - ball speed
        this.ch_Speed.addItemListener(new ItemListener()  {
                public void itemStateChanged(ItemEvent evt) {
                        if (ch_Speed.getSelectedItem().equals(" Blitz ")){
                                theBoard.getBall().setVel(5,5);
                        }
                        else if (ch_Speed.getSelectedItem().equals(" Av Jo ")){
                                theBoard.getBall().setVel(3,3);
                        }
                        else if (ch_Speed.getSelectedItem().equals(" Wimp ")){
                                theBoard.getBall().setVel(2,2);
```

```
                }
        }
});
```

```
        // choice - level
        this.ch_Level.addItemListener(new ItemListener()  {
                public void itemStateChanged(ItemEvent evt) {
                        if(ch_Level.getSelectedItem().equals(" Advanced ")) {
                                level = 2;
                                theBoard.interrupt();

                        theBoard = new MyBoard(myCanvas, ball, paddle, deflector1,tf_BallsLeft, tf_BricksLeft, tf_Time, tf_Score, ch_Level);

                }
                else if (ch_Level.getSelectedItem().equals(" Novice ")) {
                        level = 1;
                        theBoard.interrupt();

                        theBoard = new MyBoard(myCanvas, ball, paddle, null, tf_BallsLeft, tf_BricksLeft, tf_Time, tf_Score, ch_Level);
                                }
                }
        });

        cp.setLayout(new BorderLayout());
        cp.add( this.buttonPanel,       "South");
        cp.add( this.statusPanel,       "West");
        cp.add( this.gamePanel,         "Center");
        cp.add( this.timerPanel,        "North");

    this.pack();
    this.setVisible(true);
} // ends initUI
```

```
 // implement Runnable
  public void run () {
              }

  /** accessors **/
  public int getLevel() {
          return ( this.level );
  }

  /** displayers and UI checks**/
  public void setColor() {
      color = new Color(        (int)Math.floor(Math.random() * 256D),
                                                (int)Math.floor(Math.random() * 256D),
                                                (int)Math.floor(Math.random() * 256D) );
      Graphics g = getGraphics();
      Graphics2D g2 = (Graphics2D) g;
      g2.setColor(color);
      g2.fillRect(0, 0, getSize().width, getSize().height);
      setBackground(color);
   }
```

```
          public void startAll() {
    try {
        startNow = true;
       // setTime();
       pause.setEnabled(true);
                resume.setEnabled(false);
        reset.setEnabled(true);
```

```
                    theBoard.start();
                } catch (IllegalThreadStateException iltse) {};
    }
```

```
public void pause(int time) {
        try {
          Thread.sleep(time);
          //setTime();
           return;
        }
        catch(InterruptedException _ex) {
           return;
        }
    }

            public void doExitCommand (int status) {
            System.exit (status);
            }

    /** mutators **/
    public int setLevel( int level ) {
            return this.level       =           level;
    }


    public static void main(String args[]) throws Exception {
            breakOutUI bO       =           new breakOutUI();
            }

}
```

**ENJOY!!!!!**