Building a Java First-Person Shooter

Episode 8 – Alpha Support and More [Last Update 5/6/2017]

Objective

In this episode we introduce a game timer and ensure that the display screen (minus border insets) is exactly WIDTH x HEIGHT.

URL

https://www.youtube.com/watch?v=TQUzsyWmQ14

Discussion

Game Timer

CHERNO introduced a new class name Game.java. I would have preferred something like GameTimer.java but I suppose the intention will be to put into this class the game elements.

Create the following new class:

Table 1 - Game.java

```
package com.mime.minefront;

public class Game {

	public int time;
	public void tick() {
		time += 10;
	}
}
```

The `Screen.java render()` method will now be updated to use the above `tick()` method rather than depend on the `System.currentTimeMillis()` method. The above will ensure that the game clock is what is controlling how things move on the screen.

Table 2 - Updated Screen.java

```
package com.mime.minefront.graphics;

import java.util.Random;

import com.mime.minefront.Game;

public class Screen extends Render {

	private Render test;
	private final int BLOCK_SIZE = 256;	// temporary used for testing
```

```
	public Screen(int width, int height) {
		super(width, height);
		Random random = new Random();
		test = new Render(BLOCK_SIZE, BLOCK_SIZE);
		for (int i=0; i < BLOCK_SIZE * BLOCK_SIZE; i++) {
			test.pixels[i] = random.nextInt();
		}
	}


	public void render(Game game) {
		// let's first clear the screen
		for (int i = 0; i < (width * height); i++) {
			pixels[i] = 0;
		}
		int xCenter = (width - BLOCK_SIZE) / 2;
		int yCenter = (height - BLOCK_SIZE) / 2;

		for (int i = 0; i < 30; i++){
			int anim1 = (int) (Math.sin( ((game.time % 1000 + i) / 1000.0) *
2 * Math.PI) * 100);
			int anim2 = (int) (Math.cos( ((game.time % 1000 + i) / 1000.0) *
2 * Math.PI) * 100);
			draw(test, xCenter + anim1, yCenter + anim2);
		}
	}
}
```

In addition, Display class `tick()` method will be updated to invoke the `game.tick()` method.

```
	private void tick() {
		game.tick();
	}
```

The above requires the introduction of a new variable game into the Display class. In addition, it will need to be initialized in the Display constructor class.

This is the complete Display.java class:

Table 3 - Display.java for episode 8

```
package com.mime.minefront;

import java.awt.Canvas;
import java.awt.Graphics;
import java.awt.Insets;
import java.awt.image.BufferStrategy;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferInt;

import javax.swing.JFrame;
```

```
import javax.swing.SwingUtilities;

import com.mime.minefront.graphics.Screen;

public class Display extends Canvas implements Runnable{

	private static final long serialVersionUID = 1L;

	public static final int WIDTH = 800;
	public static final int HEIGHT = 600;
	public static final String TITLE = "Minefront Pre-Alpha 0.01";

	public static final int FRAMES_PER_SECOND = 60;

	private Thread thread;
	private boolean running = false; // indicates if the game is running or not

	private Screen screen;
	private Game game;
	private BufferedImage img;
	private int[] pixels;

	public Display() {
		screen = new Screen(WIDTH, HEIGHT);
		img = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
		pixels = ((DataBufferInt)img.getRaster().getDataBuffer()).getData();
		game = new Game();
	}



	private void start() {
		if (running) {
			return;
		}

		running = true;
		thread = new Thread(this);
		thread.start();
		System.out.println("Working");
	}

	private void stop() {
		System.out.println("stop() method invoked.");
		if (!running) {
			return;
		}
		running = false;
		try {
			thread.join();
		} catch (Exception e) {
			e.printStackTrace();
			System.exit(0);
		}
	}
```

```
	private void tick() {
		game.tick();
	}

	private void render() {
		BufferStrategy bs = this.getBufferStrategy();
		if (bs == null) {
			createBufferStrategy(3);
			return;
		}

		screen.render(game);

		for (int i = 0; i < WIDTH * HEIGHT; i++) {
			pixels[i] = screen.pixels[i];
		}
		Graphics g = bs.getDrawGraphics();
		g.drawImage(img, 0, 0, WIDTH, HEIGHT, null);
		g.dispose();
		bs.show();
	}


	@Override
	public void run() {
		// holds the number of frames per second
		int frames = 0;
		// accumulates time
		double unprocessedSeconds = 0;
		// start counting
		long previousTime = System.nanoTime();
		// ideal frame rate 60 ticks/sec
		double secondsPerTick = 1 / 60.0;
		// the number of ticks which should be 60 ticks/second
		// and we report the frame rate to the screen
		int tickCount = 0;
		// our signal indicator that a frame should be rendered
		boolean ticked = false;

		while(running) {
			// holds the time now
			long currentTime = System.nanoTime();
			// holds the time between now and last time
			long passedTime = currentTime - previousTime;
			// remember for the next check
			previousTime = currentTime;
			// real time elapsed
			unprocessedSeconds += passedTime / 1000000000.0;

			// has 1/60th sec elpased - i.e. a tick?
			while (unprocessedSeconds > secondsPerTick) {
				tick();		// do the update!
				// remember the time that went over a tick
				unprocessedSeconds -= secondsPerTick;
```

```
                        // signal to render a frame to the screen
                        ticked = true;
                        tickCount++;
                        // is it time to print the frame rate to the screen
                        if (tickCount % 60 == 0) {
                                System.out.println("" + frames +  " fps");
                                System.out.println("w: " + this.getWidth() + ";h: "
+ this.getHeight());
                                // add a fudge factor
                                previousTime += 1000;
                                frames = 0;
                        }
                }
                // always draw the screen as fast as possible
                render();
                frames++;
        }
    }


    public static void main(String[] args) {
        Display game = new Display();
        JFrame frame = new JFrame();
        frame.add(game);
        frame.setTitle(TITLE);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(WIDTH, HEIGHT);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
        frame.setVisible(true);

        System.out.println("Running...");
        game.start();
    }


}
```

This video episode introduces a horrible kludge in order to get the canvas to match our expected size of `WIDTH` x `HEIGHT`. I have learned a bit since I did the video four years back – how to measure the canvas size. I have decided the best approach was to re-do the code a bit. I changed the `Display` class so that is our `JFrame` and have it create and manage a `Canvas` object directly. Here is the new version of Display.java

```
package com.mime.minefront;

import java.awt.Canvas;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Insets;
import java.awt.image.BufferStrategy;
```

```
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferInt;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import com.mime.minefront.graphics.Screen;

public class Display extends JFrame implements Runnable{

	private static final long serialVersionUID = 1L;

	public static final int WIDTH = 800;
	public static final int HEIGHT = 600;
	public static final String TITLE = "Minefront Pre-Alpha 0.01";

	private Thread thread;
	private boolean running = false; // indicates if the game is running or not

	private Canvas canvas;
	private Screen screen;
	private Game game;
	private BufferedImage img;
	private int[] pixels;

	public Display() {
		// Set the preferred size of the Canvas
		canvas = new Canvas();
		Dimension size = new Dimension(WIDTH, HEIGHT);
		canvas.setPreferredSize(size);
		canvas.setMaximumSize(size);
		canvas.setMinimumSize(size);
		screen = new Screen(WIDTH, HEIGHT);
		game = new Game();
		img = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
		// Get the pixel array of the the ACTUAL SCREEN
		pixels = ((DataBufferInt)img.getRaster().getDataBuffer()).getData();
	}

	private void start() {
		if (running) {
			return;
		}

		running = true;
		thread = new Thread(this);
		thread.start();
		System.out.println("Working");
	}

	private void stop() {
		System.out.println("stop() method invoked.");
		if (!running) {
			return;
		}
```

```
			running = false;
			try {
				thread.join();
			} catch (Exception e) {
				e.printStackTrace();
				System.exit(0);
			}
		}

		private void tick() {
			game.tick();
		}

		private void render() {
			BufferStrategy bs = canvas.getBufferStrategy();
			if (bs == null) {
				canvas.createBufferStrategy(3);
				return;
			}

			screen.render(game);
			// draw 'screen' to the REAL SCREEN
			for (int i = 0; i < WIDTH * HEIGHT; i++) {
				pixels[i] = screen.pixels[i];
			}
			Graphics g = bs.getDrawGraphics();
			g.drawImage(img, 0, 0, WIDTH, HEIGHT, null);
			g.dispose();
			bs.show();
		}


		@Override
		public void run() {
			// holds the number of frames per second
			int frames = 0;
			// accumulates time
			double unprocessedSeconds = 0;
			// start counting
			long previousTime = System.nanoTime();
			// ideal frame rate 60 ticks/sec
			double secondsPerTick = 1 / 60.0;
			// the number of ticks which should be 60 ticks/second
			// and we report the frame rate to the screen
			int tickCount = 0;
			// our signal indicator that a frame should be rendered
			boolean ticked = false;

			while(running) {
				// holds the time now
				long currentTime = System.nanoTime();
				// holds the time between now and last time
				long passedTime = currentTime - previousTime;
				// remember for the next check
				previousTime = currentTime;
```

```
                    // real time elapsed
                    unprocessedSeconds += passedTime / 1000000000.0;

                    // has 1/60th sec elpased - i.e. a tick?
                    while (unprocessedSeconds > secondsPerTick) {
                          tick();       // do the update!
                          // remember the time that went over a tick
                          unprocessedSeconds -= secondsPerTick;
                          // signal to render a frame to the screen
                          ticked = true;
                          tickCount++;
                          // is it time to print the frame rate to the screen
                          if (tickCount % 60 == 0) {
                                System.out.println("" + frames +  " fps");
                                System.out.println("w: " + canvas.getWidth() + ";h:
" + canvas.getHeight());
                                // add a fudge factor
                                previousTime += 1000;
                                frames = 0;
                          }
                    }
                    // always draw the screen as fast as possible
                    render();
                    frames++;
              }
        }

        public void resizeToInternalSize(JFrame frame, int internalWidth, int
internalHeight) {
              Insets insets = frame.getInsets();
              final int newWidth = internalWidth + insets.left + insets.right;
              final int newHeight = internalHeight + insets.top + insets.bottom;

              Runnable resize = new Runnable() {
                    public void run() {
                          frame.setSize(newWidth, newHeight);
                    }
              };

              if (SwingUtilities.isEventDispatchThread()) {
                    try {
                          SwingUtilities.invokeAndWait(resize);
                    } catch (Exception e) {
                          // ignore ...but will be no no if using Sonar!
                    }
              } else {
                    resize.run();
              }

              frame.validate();
        }


        public static void main(String[] args) {
              Display game = new Display();
```

```
                game.add(game.canvas);
                game.setTitle(TITLE);
                game.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                game.pack();
                game.setLocationRelativeTo(null);
                game.setResizable(false);
                game.setVisible(true);

                // ensure the display canvas is the right size!
                game.resizeToInternalSize(game, WIDTH, HEIGHT);

                System.out.println("Running...");
                game.start();
        }


}
```

The above works the same but with no screen kludges.

Setting the Canvas to WIDTH x HEIGHT

The Display.java class shown in the above table highlights a new method that I added to the class – `resizeToInternalSize()`.

This details on how `resizeToInternalSize()` was discussed in my episode 1 notes. What we are basically doing is ensuring that the canvas created (`Display`) comes to exactly WIDTH x HEIGHT. In all previous episodes it was slightly less than that because of the pixels the frame window was taking.

My version of the code is different. Nothing irks me more than kludgy code being added when teaching someone new concepts. Do "real" programmers ever add kludgy code? Sure. But we know we have encumbered a debt of sorts. We have to go back and figure out what we don't understand well enough to avoid such things. When teaching someone something new there is never any reason to do such things. It can only mean we are not ready to teach the subject matter.

Last comment: CHERNO went back to discussing alpha but he has not figured out yet that he is losing color in the "colored square" because he is eliminating too many pixels (probability dictates it is about half of them).

Test how Alpha should work

We are going to create a new image where black pixels will be used to indicate "don't draw pixel" in order to draw a donut as shown below:

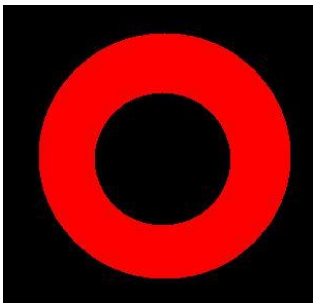

Figure 1 - Red donut

In order to make sure it is truly working we will make the background color white and print many versions as we did the "colored squares."

Here are the changes I made to the code to "see" how alpha (the correct version I implemented) truly works.

- ⇨ Copy EPISODE_08 to EPISODE_08_B. All changes describe change to EPISODE_08_B.

We only change one class the Screen.java class to print a red donut as shown above.

Table 4 - Screen.java (that print test donut)

```
package com.mime.minefront.graphics;

import java.util.Random;

import com.mime.minefront.Game;

public class Screen extends Render {

	private Render test;
	private final int BLOCK_SIZE = 256; // temporary used for testing

	public Screen(int width, int height) {
		super(width, height);
		Random random = new Random();
		test = new Render(BLOCK_SIZE, BLOCK_SIZE);
		for (int i=0; i < BLOCK_SIZE * BLOCK_SIZE; i++) {
			test.pixels[i] = random.nextInt();
		}
		createDonut(test);
	}

	private boolean isPointInCircle(int xCenter, int yCenter, int radius,
int x, int y) {
		return ((x-xCenter) * (x-xCenter) + (y - yCenter) * (y -
yCenter)) < (radius * radius);
	}

	private void createDonut(Render test) {
		int circleRadius = 75;
```

```
			// make the entire test area black
			for (int i=0; i < BLOCK_SIZE * BLOCK_SIZE; i++) {
				test.pixels[i] = 0;
			}

			// draw the red circle
			int xCenter = BLOCK_SIZE / 2;
			int yCenter = xCenter;
			for (int y=0; y < BLOCK_SIZE; y++) {
				for (int x=0; x < BLOCK_SIZE; x++) {
					if (isPointInCircle(xCenter, yCenter, circleRadius,
x, y)) {
						test.pixels[x + (BLOCK_SIZE*y)] = 0x00ff0000;
					}
				}
			}

			// now draw inner black circle
			circleRadius = 25;
			for (int y=0; y < BLOCK_SIZE; y++) {
				for (int x=0; x < BLOCK_SIZE; x++) {
					if (isPointInCircle(xCenter, yCenter, circleRadius,
x, y)) {
						test.pixels[x + (BLOCK_SIZE*y)] = 0x00000000;
					}
				}
			}

		}

		public void render(Game game) {
			// let's first clear the screen
			for (int i = 0; i < (width * height); i++) {
				pixels[i] = 0xffffff;
			}

			int xCenter = (width - BLOCK_SIZE) / 2;
			int yCenter = (height - BLOCK_SIZE) / 2;

			for (int i = 0; i < 30; i++){
				int anim1 = (int) (Math.sin( ((game.time % 1000 + i) /
1000.0) * 2 * Math.PI) * 100);
				int anim2 = (int) (Math.cos( ((game.time % 1000 + i) /
1000.0) * 2 * Math.PI) * 100);
				draw(test, xCenter + anim1, yCenter + anim2);
			}
		}
}
```

The render screen is essentially the same except for the fact that the screen is cleared to white. This will prove that a “black” pixel in our test Render object is not drawn to the screen.

We created two new methods. The `createDonut` creates a red donut shape into the test area (I could have done this all in one loop) the test area is completely made black then a red circle is drawn and then a smaller black circle (our donut) is drawn within it in order to get the donut shape outlined in the figure above.

When you run the program you will see the following:

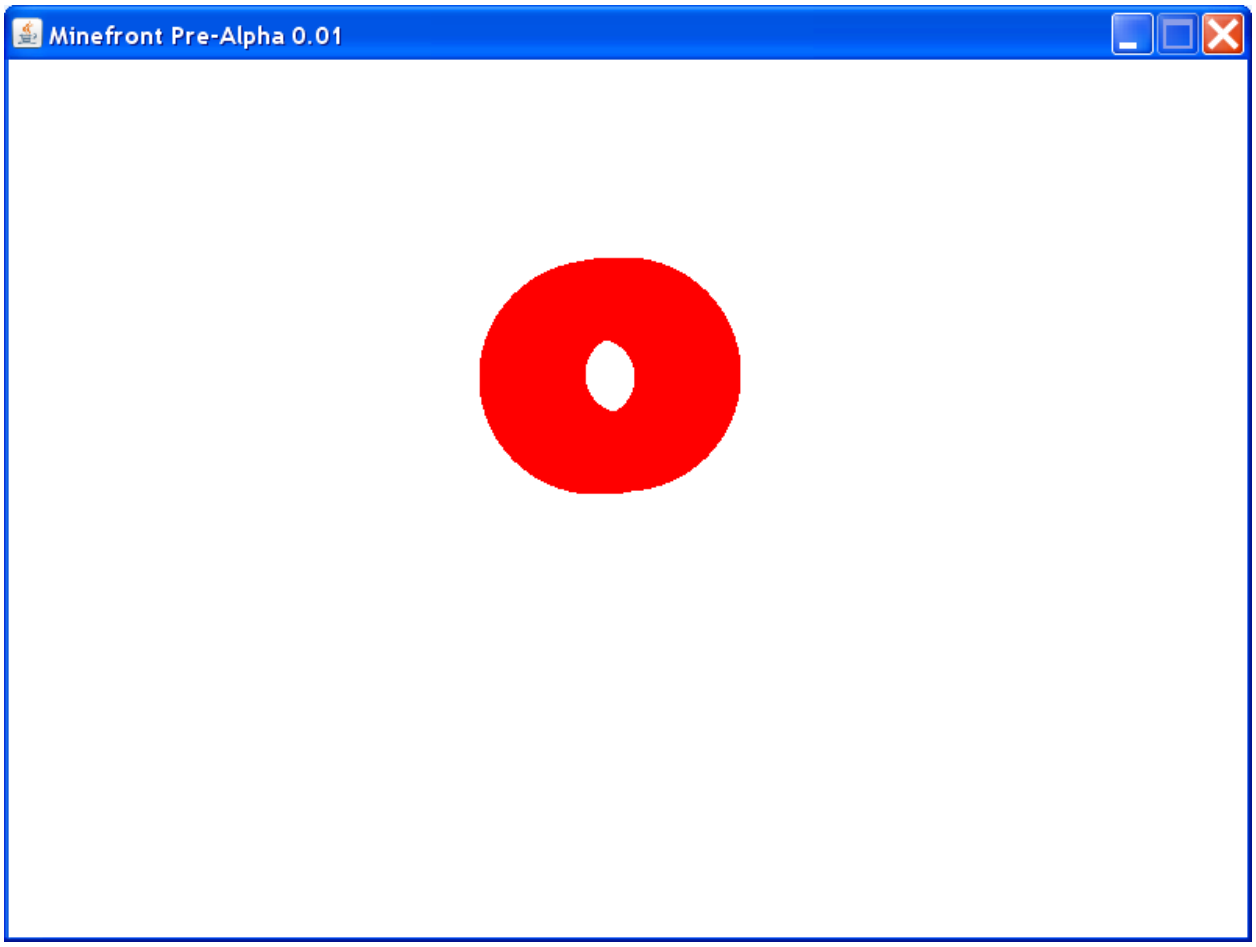

Figure 2 - Drawing a donut to illustrate how alpha works