ITC527 Assignment 1

Andrew James

March 24, 2011

Contents

1 Task

Extend Lea's ParticleApplet program as an Application that provides the following additional functionality;

1. When the particle moves out of the displayed area, the thread controlling that particle will terminate.
2. The number of particles existing at any stage will be displayed in an appropriate area of the GUI.
3. A button will be provided on the GUI, which allows the user to create a new particle (and controlling thread) at any stage. The particle will be created at the usual initial position.
4. The user can remove a particle (and terminate its controlling thread) by clicking on the particle.

2 Solution Approach

The program was first implemented in a skeleton Application framework that provided the exact same functionality of the original ParticleApplet by Lea.

In order to accomplish this a simple Main class was created as the entry point of the program. A ParticleWindow class was defined to contain the code required for creating and updating a window. The Main.main function then just needed to create a new ParticleWindow object and tell it to start running. The ParticleWindow was derived from the Java.awt.Frame class, and thus only needed to define a few basic properties about itself (such as name and size) then define a canvas to be used for painting and a method to handle quit requests. The initialisation and execution code remained identical to the ParticleApplet program.

Once this framework was operational the requirements were implemented as follows:

2.1 When the particle moves out of the displayed area, the thread controlling that particle will terminate

A method was added to the Particle class that returned true if the particle was partially within a rectangle specified by the user. For simplicity the function only takes two arguments which are the width and height of the rectangle. It is assumed that the rectangle is bordered with the edges defined by $[0, width, 0, height]$ ($[left, right, bottom, top]$ respectively).

The makeThread function was modified such that each time the particle was moved, the controlling thread checked to ensure the particle was still within the boundary of the canvas. If at any point this function returned false the thread would stop running.

A new thread was created that monitored the list of Threads to ensure that any dead threads were removed. This controller would iterate through the list of threads 10 times a second, calling Thread.isAlive() for each thread. If this ever returned false it would remove the thread from the collection and then continue.

In order to accomplish this functionality the declaration of Threads[] (and all associated code) had to be modified to use the List interface, with the array created as an ArrayList instead. At this point the list of particles was not being modified after initialisation, and so the declaration was not altered.

2.2 The number of particles existing at any stage will be displayed in an appropriate area of the GUI

As the array of particles was not being modified, its size was not able to be used as a reference for the count of live particles. A new variable called activeParticles was created in the ParticleCanvas class and a setActiveParticles method was provided to allow the ParticleWindow class to update the count when a thread was killed.

This variable and method was later removed and replaced with a more natural and efficient approach. (Mentioned in section 2.4)

The canvas would then draw the value of activeParticles as a string in the top left of the window as part of the paint method.

2.3 A button will be provided on the GUI, which allows the user to create a new particle (and controlling thread) at any stage. The particle will be created at the usual initial position

The "usual initial position" requirement was interpreted to mean the center of the canvas, as this seemed a natural place for new particles to appear.

A button was created in the construction of the ParticleWindow class and attached to the bottom of the window. The ParticleWindow class was then modified to implement the ActionListener interface and registered as a listener for the button. A method was defined to allow the ParticleWindow class to respond to events generated by the button.

When an event is received the event type is checked, if it corresponds to the type generated by the Button, a new particle is created in the current centre of the canvas and a thread is created to control that particle. In a synchronised block of code the particle and thread are added to their associated collections, and the thread is started.

2.4 The user can remove a particle (and terminate its controlling thread) by clicking on the particle

A method was added to the Particle class that would check if the particle was within a certain radius of the point. For simplicity it checks if the distance between the point and the particle centre is less than the radius (thus assuming the particle has a spherical shape, not square).

The ParticleWindow class was modified to implement the MouseListener interface. This interface requires the implementation of five methods, but only one is used by this application. The mouseClicked method is called when a mouse button is pressed and released, so this is the method that needs to be implemented. When an event is passed to the method a check is performed to see which button was pressed. If it corresponds to the left mouse button the

(x, y) position of the mouse is retrieved and passed to the nearPoint method of all particles.

At this point the particles array needed to be altered to use the ArrayList type and each method dealing with the particles array was updated to expect a List interface instead. The controller code was updated to remove the particle that corresponds to the associated thread at the same time it removes a thread (thread[i] and particle[i] are removed at the same time). This also meant that the ParticleCanvas.activeParticles method was redundant as particles.size() would be accurate at all times.

The mouseClicked method would then interrupt and also remove the thread and particle if Particle.nearPoint returned true (which would imply that the mouse was within the bounds of the particle when a click was registered). The method would then return after culling the first match.

2.5 Additional modifications

Double buffering was employed in order to eliminate the visual tearing effect that was present when drawing directly to the active canvas. The calls to canvas.repaint() were moved from the makeThread method to the controller thread in order to stabilise the frame rate and improve application performance with large particle counts.

3 Limitations and Bugs

A bug that was present in Lea's code has been intentionally preserved. The Particle.move method contains the lines:

```
public synchronized void move() {
  x += rng.nextInt(10) - 5;    // Should be nextInt(11)
  y += rng.nextInt(20) - 10;   // Should be nextInt(21)
}
```

The Random.nextInt(n) method returns a number in the interval $[0, n)$, which means that it is more likely for the particles to move left or up the screen (as the origin, $(0, 0)$, is located in the top left corner).

It can be difficult to select the particles with the mouse due to the large range of vertical motion permitted, the hit region has been set to be slightly larger than the Particle to make it easier to remove a Particle with a mouse click.

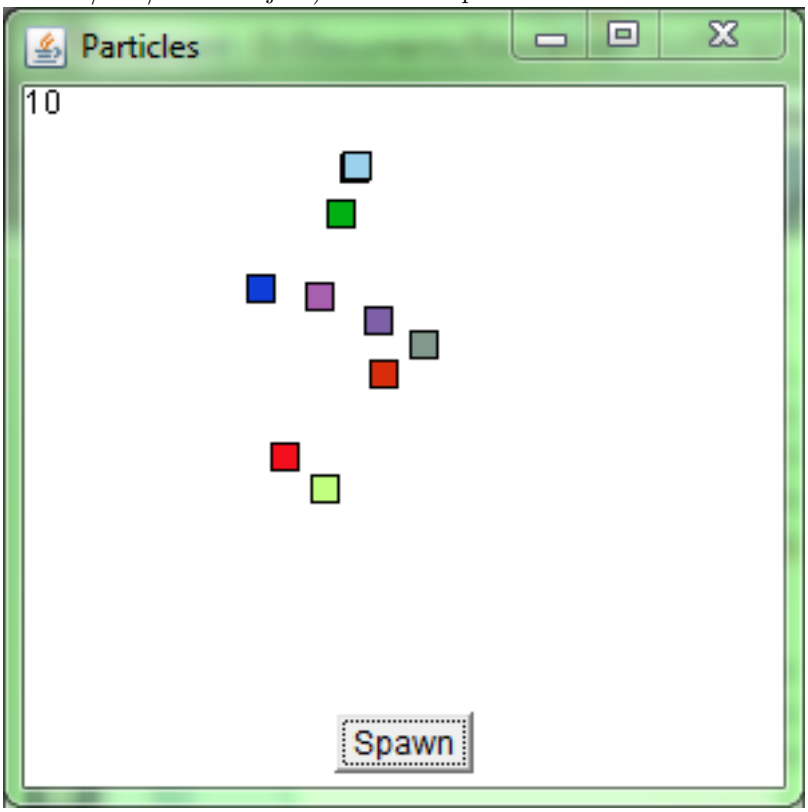
There is no upper limit on the number of particles (and thus, threads) that can be created. With large window sizes this can cause performance problems. With the default window size approximately 140 particles can be created before the application reaches a stable state (Particles exiting the window at the same rate as new particles can be created).

4 Structure of the source code

For a complete listing of classes and methods, please see the javadoc generated documentation located at "Assignment 1/Particles/dist/javadoc/index.html"

5 User guide

The program can be run by executing Particles.jar (located at "Assignment 1/Particles/dist/Particles.jar") with a compatible Java Runtime Environment.



When first launched a window similar to the above will appear. The particles will move around and as they move off the screen the counter in the top left will decrement.

The window can be resized, particles will retain their position relative to the top left corner unless the new dimensions of the window would place them outside the viewing area, in which case they will be removed and the counter will decrement.

You may click on a particle to remove it. The counter will decrement and the particle will disappear.

Clicking on the button labelled "Spawn" will cause a new particle to appear in the center of the window and start moving around.

The program can be exited by clicking on the X in the top right of the window (top left for MacOS).

6 Programmers guide

The Particle folder is a Netbeans project folder and can be opened and run from that IDE. If using another IDE or the command line compiler the files Particle.java, ParticleCanvas.java, ParticleWindow.java, and Main.java will need to be compiled in that order, then run Main.class or the generated .jar file.