Smart Grid

New Mexico

Supercomputing Challenge

Final Report

Team 75

Los Alamos Middle School (LAMs)

Team Members -

- Colin Redman
- Michael Englert Erickson
- Sudeep Dasari

Teacher -

- Clara Vigil

Project Mentors -

- Andrew Erickson
- Jim Redman
- Venkat Dasari

Executive Summary: The project that we did this year is on the Smart Grid. The Smart Grid is an electric grid that can communicate with household appliances and reschedule nonessential appliances around times of peak load. We have examined benefits of deploying smart grid type of control technologies in a semi-urban community like our hometown Los Alamos New Mexico. Peak load is the daily power surge. It happens between four o'clock to nine o'clock pm in Los Alamos New Mexico. It is caused when most commuters arrive at their home and turn on most if not all of their appliances. By rescheduling nonessential appliances and adding price benefits we can drastically reduce peak load.

We created an agent based model that uses an applet simulation to uses vectors to store instances for houses. Because conventional agent based models, such as NetLogo, can't handle a very large number of agents simultaneously we have decided to develop and use our own agent-based model in Java environment. This Java environment was created in Eclipse. In this report we present and discuss the objectives and the results of the model. For example, our simulations indicate that deployment of smart grid together with even the simplest of control strategies can save energy use by as much as five% and "shave-off" the peak load use in excess of thirteen%. We recognized through the simulation that bad control strategies coupled with incomplete understanding of grid conditions can result in unstable operating conditions. For example, all the appliances can shut-off at the same time resulting in grid voltage overshooting safety limits. Or all the appliances could turn on at the same time which will result in a collapse of the grid. Our team programmer managed to fix most of the

problems plaguing our simulation but there are still a few left. He will work hard to fix the final problems with our program for the Expo.

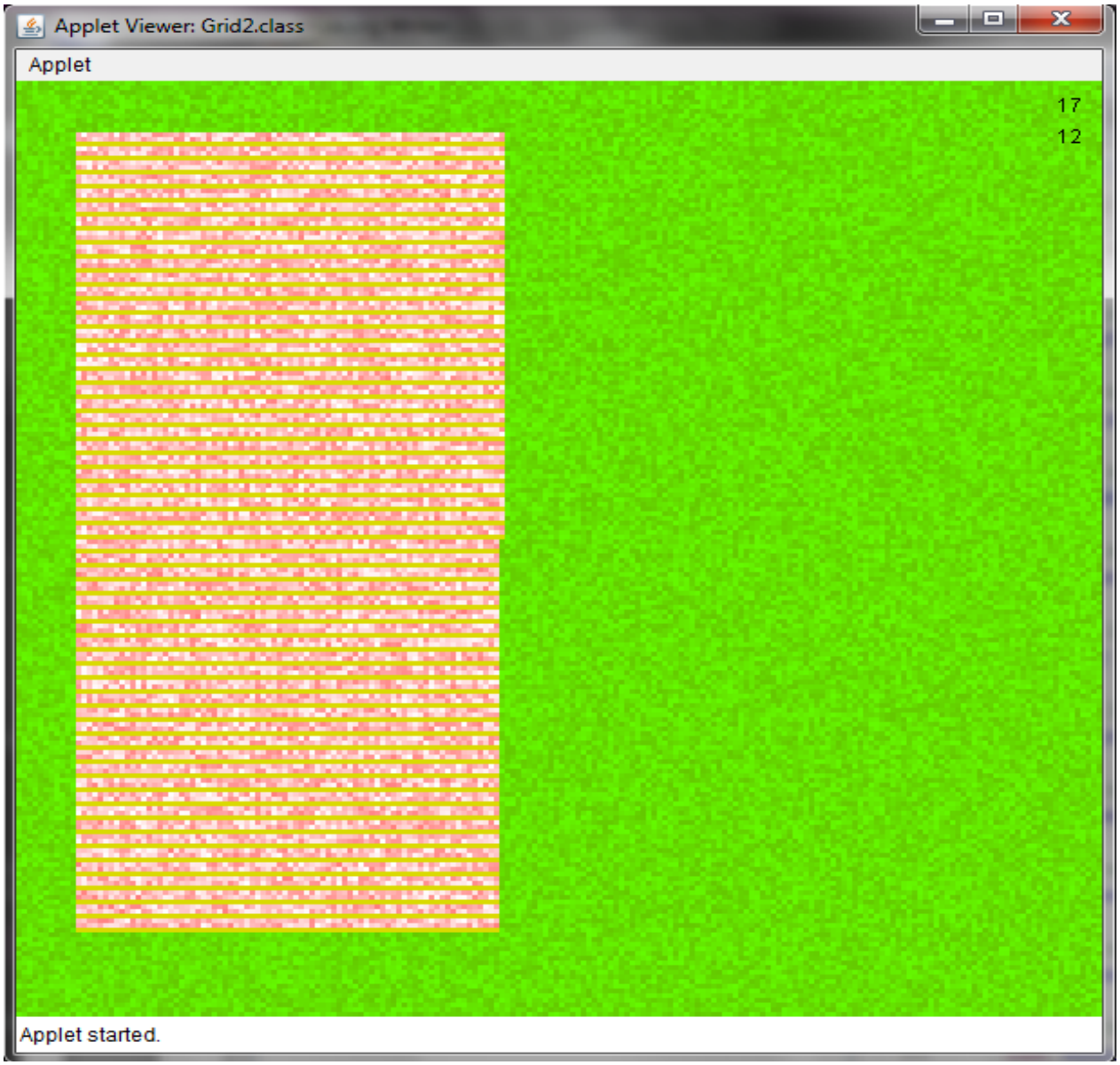
Introduction: During a typical day between 5:00 pm to 9:00 pm there is a peak in the amount of power used by appliances when people return from work and rush to finish dinner and other daily jobs. In the figure below in our results (Figure 1) red-line illustrates typical power use profile in Los Alamos. In Los Alamos that peak occurs at about 8:00 PM and can be as high as 16 MW compared to average power consumption of 13 MW during the day. During peak load inefficient backup generators or "peakers" are used to compensate for the sudden increase in power demand. In addition these peak loads can also result in excess energy loss due to Ohmic losses (that is heating of the electric wires by current as given by I^2R equation)

Smart Grid's goal is to reduce this power peak by transferring power used by noncommercial appliances to other parts of the day (like at night). The Smart Grid works by using the zigbee system. Every appliance would be hooked up to the zigbee network. The zigbee system connects all appliances in a house and then each house in the community is connected to the city grid system. This network allows appliances to communicate with each other and with the grid. This way allows appliances in a household to continuously monitor the grid condition in terms of its voltage and determine when the other appliances on the zigbee network are on or off. When appliances recognize that peak conditions are reaching, then they can turn themselves off completely or operate at reduced consumption. Communication between appliances is essential otherwise all of them can shutdown at the same time leading to unstable grid conditions. Our hypothesis is that such a smart control strategy will greatly reduce the peak load.

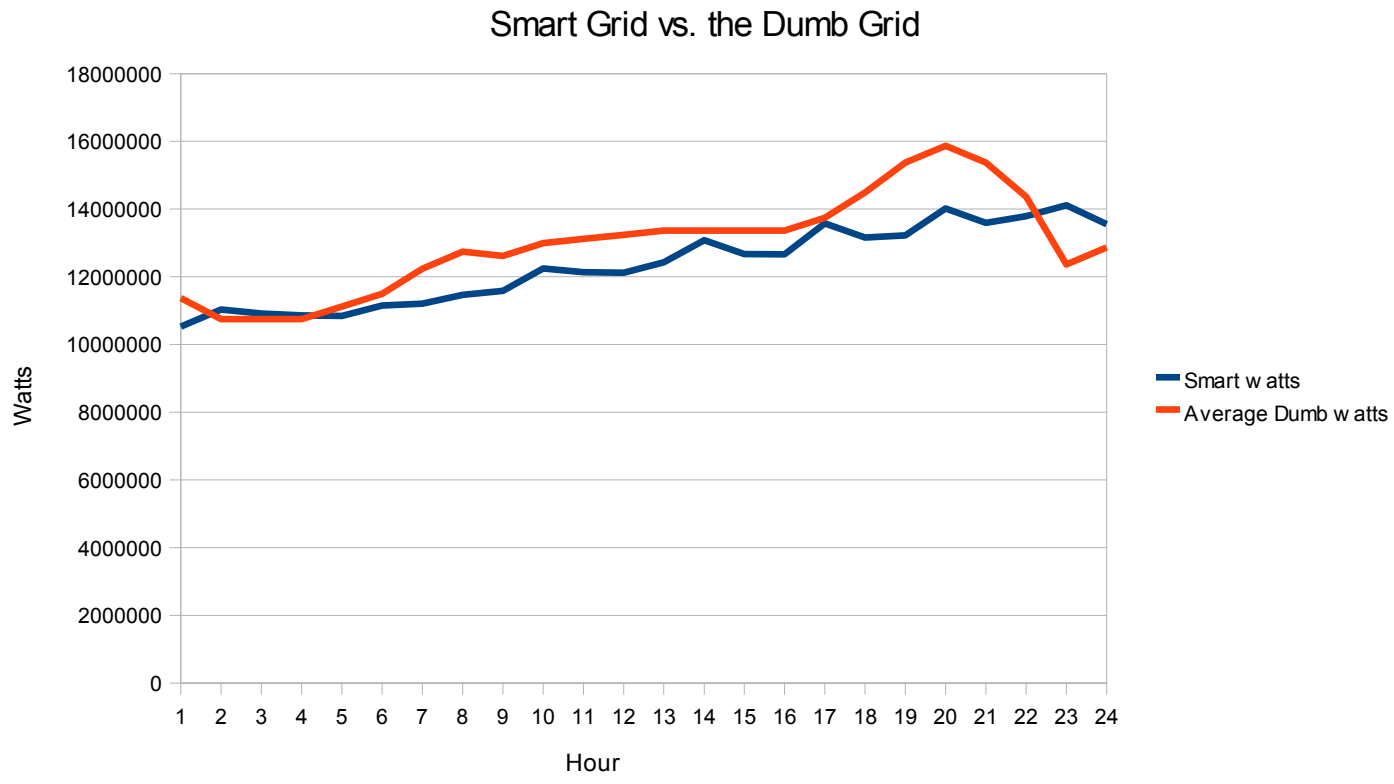
Description: Our project's goal is to accurately simulate the advantages of Smart Grid. Our team programmer has chosen agent-based modeling approach for the simulation. We have selected Los Alamos County as the model community. In the model we created several thousands of dots (or agents) that form the community (In LA for example we have approximately 9000 residential users). Each individual dot has its own figures for ID, color, and power usage. Each dot is assigned a specific amount of appliances and types of appliances. We will create a simulated Smart Grid and use a graph from LANL on the average dumb grid power use. The Smart Grid simulation should show less power usage during peak load than the LANL graph did. In our simulation the Smart Grid will allow each appliance to use zigbee to communicate with one another and distribute the amount of energy used over the day.

Commonly available agent-based models, such NetLogo, can't accommodate these many agents in one simulation. They also do not provide flexibility and control we desired. So we developed our own agent-based model in the Java environment.

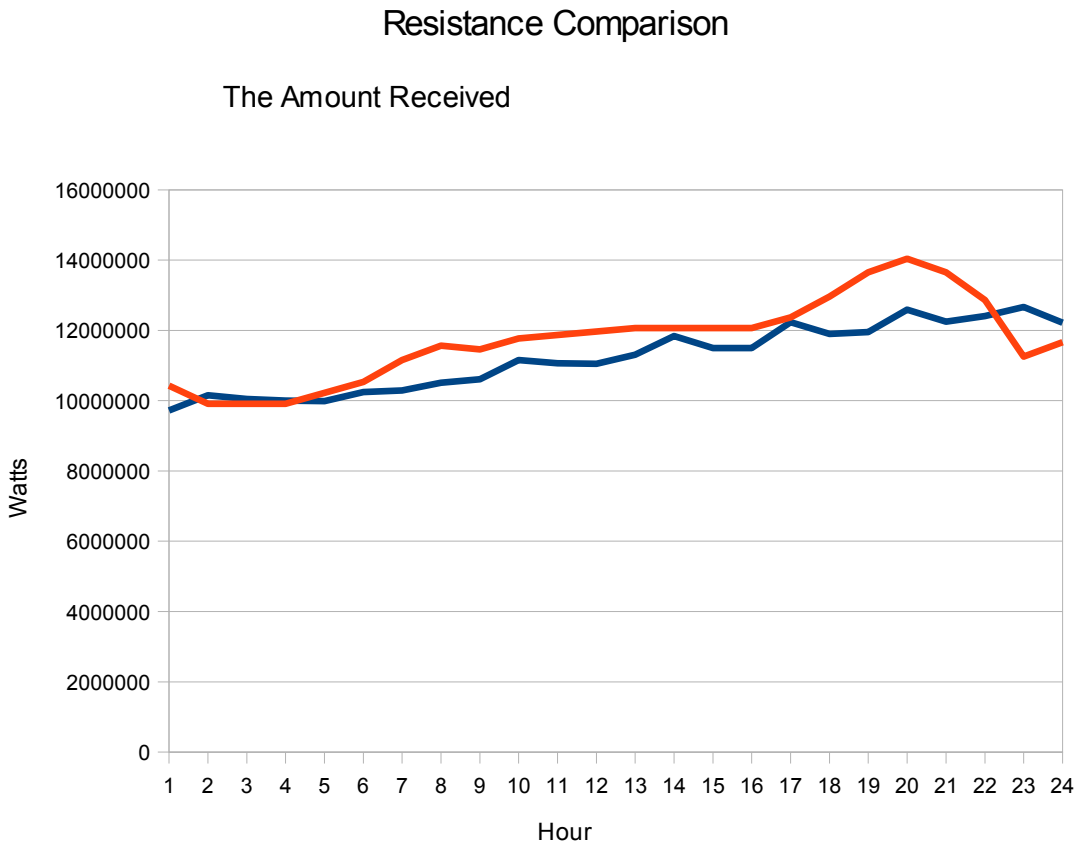
The Program: The program is an applet which uses vectors to store instances of objects and information for the paint program. There is one primary class or object in the simulation which is the house. The house stores it's usage information and has methods for the other appliances that can be moved away from peak times. The main class tells the house objects when they can use energy. During the simulation the buildings are red-scaled to show energy use, allowing the energy use to be easily visible.



Results: In Los Alamos county our existing grid (or “dumb grid”) reaches peak use conditions at approximately 8 PM as residences (or agents) turn on their appliances. This peak is as high as 16000 kilowatts. In dumb grid setting we baselined our agent-based community to closely resemble LA County power use (see Red Line in Figure 1 below). After that we turned on our smart grid setting. Smart watts (or energy usage under smart grid setting or shown as blue line). We can clearly see that smart controls have resulted in lowering the peak substantially. According to our computer program the smart grid was able to save on average 637.09 kilowatts which is about 4% of the total use. Reduced energy use and heating losses contributed to lowered energy consumption. The computer model was able to efficiently and effectively redistribute around 1779 kilowatts from the peak load which is a 13% reduction. These results demonstrate that smart grid technologies can result in good energy savings and peak reductions. Of course in our simulation we assumed 100% residential participation in smart grid, less participation does not result in such greater savings.



almost 2 mega watts (2 million watts) are lost during the peak load times, and all this energy would have to be resent, causing more loss.



Conclusion: We concluded that smart grid implementation can indeed both save energy and reduce peak load in Los Alamos and communities similar to LA County. Agent-based modeling approach can be used to accurately simulate smart grid functionality.

There are several limitations in our “proof-of-principle” model. For example, we used semi-biased random sampling for representing agent behavior and assumed 100% participation of all residential users. In reality pricing of electricity may not drive

such high participation. We plan to refine our model to examine impacts of less than perfect participation. We have also not done a very good job at modeling electrical transients (for example, voltage fluctuations). In the future we propose to use a better electrical model together with queuing theory to turn off and on the appliances.

Acknowledgment and Resources: We would like to thank Andrew Erickson, Jim Redman, and Venkat Dasari for their help and support in this project. Also the LANL report on the smart grid was an invaluable research tool for us.

Grid2.java

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.Vector;

import javax.swing.Timer;

public class Grid2 extends Applet implements MouseListener {
	// Create the vectors that store the information about the houses and the
	// painting
	Vector houses = new Vector();
	Vector paint = new Vector();

	// The total amount used by the grid every hour (used later)
	int gridUseage = 0;

	// The population of the town
	int popSize = 82000;

	// The number of people living in each house
	double peoplePerHouse = 2.5;

	// The size of the Applet
```

```
int size = 200;

// How scaled each dot is
int sizeFacter = 3;

// How long the simulation has run
int day = 0;

// How long it should run for before stopping
int daysRun = 50;

// The normal (average) power hourly per house in watts
int powerPerHouseNorm = 7000;

// The maximum power used in an hour in a house in watts
int powerPerHouseMax = 0;

// The normal power used by the grid.
int norm = (int) ((popSize / peoplePerHouse) * powerPerHouseNorm);

// The hour of the day
int hour = 1;

// Color of the grass; changes day / night
int grassColor = 0;

int[] finalUse = (new House()).hourlyUsage;

int finalTotalUse;

// Frames per second, if 1000 or over it will go multi-threaded.
```

```
int fps = 33;

// Number of houses
int housesNumb = 0;

// Are the graphics on?
boolean graphics = false;

// This will be displayed on the applet
String display = "";

// Both are for double-buffing (to make the screen change smoothly)
Image offscreenImage;
Graphics offscreenGraphics;

// Called by the timer
ActionListener taskPerformer = new ActionListener() {

	public void actionPerformed(ActionEvent evt) {
		passTime();

	}
};

class SmartThread extends Thread {

	SmartThread(int sleep) {

		// Just letting us know
		System.out.println("Thread created");
```

```
        }

        // Run the thread
        public void run() {
                while (day < daysRun) {
                        passTime();

                }
        }
}

// The timer
Timer time = new Timer((int) 1000 / fps, taskPerformer);

// Called at the beginning
public void init() {

        // Figure out how much energy should be used
        int i = 0;
        while (i < 24) {
                finalUse[i] = finalUse[i] * 8950;
                finalTotalUse = finalTotalUse + finalUse[i];
                i++;
        }

        // Resize the applet to the right size
        this.resize(size * sizeFacter, size * sizeFacter);
        offscreenImage = createImage(getSize().width, getSize().height);
        offscreenGraphics = offscreenImage.getGraphics();

        // Ensure that vectors will hold everything
```

```
        houses.ensureCapacity((int) (popSize / peoplePerHouse));
        paint.ensureCapacity((size ^ 2));

        // Let the mouse be detected
        addMouseListener(this);

        // Create a new world
        newWorld();

        // Check the frames per second
        if (fps < 1000) {
                graphics = true;
                time.start();
        } else {
                // If you really want power
                threadStarting();
        }

        // If there are no graphics, shrink the display
        if (!graphics) {
                this.resize(1, 1);
        }

}

// This just paints the applet
public void paint(Graphics g) {
        if (graphics) {
                for (int i = 0; i < paint.size(); i++) {
                        String input = (String) paint.elementAt(i);
                        int y = Integer.parseInt(input.substring(
```

```
                                        input.indexOf("-") + 1, input.indexOf(":")))
                                        * sizeFacter;
                        int x = Integer
                                        .parseInt(input.substring(0,
input.indexOf("-")))
                                        * sizeFacter;
                        // This section here finds the color
                        String color = input.substring(input.indexOf(":") + 1);
                        Color c;
                        if (color.substring(6, 9).equals("grs")) {
                                c = new Color(Integer.parseInt(color.substring(0,
3)),
                                                Integer.parseInt(color.substring(3, 6))
                                                                - (grassColor / 2),
grassColor);

                        } else {
                                c = new Color(Integer.parseInt(color.substring(0,
3)),
                                                Integer.parseInt(color.substring(3,
6)), Integer
                                                                .parseInt(color.substring(
6, 9)));
                                // System.out.println(color);
                        }
                        offscreenGraphics.setColor(c);
                        offscreenGraphics.fillRect(x, y, sizeFacter, sizeFacter);

                }
                offscreenGraphics.setColor(Color.black);
                offscreenGraphics.drawString(hour + "", getSize().width - 25, 20);
                offscreenGraphics.drawString(day + "", getSize().width - 25, 40);
                offscreenGraphics.drawString(display + "", 10,
                                getSize().height - 25);
```

```
                g.drawImage(offscreenImage, 0, 0, this);
        }
}

public void passTime() {

        int lastUse = 0;
        int totalHourUse = 0;
        int debugPowerUseClear = 0;
        for (int i = 0; i < houses.size(); i++) {
                House house = (House) houses.get(i);

                int use;
                // Run the house
                if (lastUse < house.run(hour, true)) {
                        use = house.run(hour, true);
                        debugPowerUseClear++;
                } else {
                        use = house.run(hour, false);
                }
                // Reset it when the hour is 1
                if (hour == 1) {
                        house.powerUsed = false;
                        houses.setElementAt(house, i);
                }
                // If there is a new record, write it down
                if (use > powerPerHouseMax) {
                        powerPerHouseMax = use;
                }
                // Record the usage
                totalHourUse = totalHourUse + use;
```

```
                        gridUseage = gridUseage + use;

                        // Paint stats from the paint vector

                        String paintStats = (String) paint.get(house.place);

                        String redShade = "";

                        if ((int) (255 - (155 * (((double) use / (double)
powerPerHouseMax)))) <= 255) {

                              // if (use / powerPerHouseMax < 1) {

                              redShade = ""

                                          + (int) (255 - (155 * (((double) use /
(double) powerPerHouseMax))));

                        } else {

                              redShade = "100";

                        }

                        paint.set(house.place, paintStats.substring(0, paintStats

                                          .indexOf(":") + 1)

                                          + "255" + redShade + redShade);

                        lastUse = use;


                  }

                  if (!graphics) {

                        System.out.println(totalHourUse);

                  }


                  // Depending on the time of day make it change light

                  if (hour <= 7) {

                        grassColor = 200;

                  } else if (hour > 7 && hour < 20) {

                        grassColor = 000;

                  } else if (hour >= 20) {

                        grassColor = 200;

                  }
```

```
			// Pass time
			hour++;
			if (hour == 25) {

				hour = 1;
				gridUseage = 0;
				day++;

				// Just test if it is actually working
				if (houses.size() < housesNumb) {
					System.out.println("Bad! People don't wash clothes! Only "
							+ debugPowerUseClear + " people are good out
of "
							+ housesNumb + "!");
				}
				// When it needs to stop it does
				if (day == daysRun) {
					time.stop();
				}
			}
			// Test to repaint
			if (graphics) {
				repaint();
			}

		}

		// Override the update to make it double-buffered
		@Override
		public void update(Graphics g) {
			paint(g);
```

```
	}

	@SuppressWarnings("unchecked")
	public void newWorld() {
		int x = 0;
		int place = paint.size();
		housesNumb = 0;
		while (x < size) {
			int y = 0;
			while (y < size) {

				if ((x > 10 && x < popSize / peoplePerHouse)
						&& (y > 10 && y < Math.sqrt((double) popSize
								/ peoplePerHouse)) && housesNumb
< 8950) {
					paint.add(x + "-" + y + ":255255255");
					place = paint.size() - 1;
					House house = new House();
					house.place = place;
					houses.add(house);

					// place--;
					housesNumb++;
					y++;
					paint.add(x + "-" + y + ":255255255");
					// house.place = place;
					house = new House();
					place = paint.size() - 1;
					houses.add(house);
					house.place = place;
					housesNumb++;
```

```
                                y++;
                                paint.add(x + "-" + y + ":" + "218218000");
                        } else {
                                paint.add(x + "-" + y + ":" + "100"
                                                + ((int) (Math.random() * 50L + 200)) +
"grs");
                                // place = paint.size();
                        }

                        y++;

                }
                x++;

        }

    }

    @Override
    public void mouseClicked(MouseEvent me) {

    }

    @Override
    public void mouseEntered(MouseEvent me) {

    }

    @Override
    public void mouseExited(MouseEvent me) {
        time.start();
```

```
        display = "";
    }

    @Override
    public void mousePressed(MouseEvent me) {
        time.stop();
        int xPos = me.getX() / sizeFacter;
        int yPos = me.getY() / sizeFacter;

        for (int i = 0; i <= houses.size() - 1; i++) {

            House house = (House) houses.get(i);

            String elementXY = paint.elementAt(house.place).toString()
                    .substring(
                                    0,
                                    paint.elementAt(house.place).toString()
                                            .indexOf(":"));

            if (elementXY.equals(xPos + "-" + yPos)) {

                display = house.power + "";
                repaint();
                i = houses.size();
            }
        }

    }

    @Override
    public void mouseReleased(MouseEvent me) {
```

```
            time.start();
            // display = "";
        }

        // When speed is wanted, it's for going multi-threaded
        public void threadStarting() {

            // Cut the graphics off
            graphics = false;

            // While the number of threads is not right...
            for (int i = 0; i < 8; i++) {

                // Start another
                SmartThread smart = new SmartThread(i * 20);
                smart.start();

            }
        }

}
```

House.java

```
public class House {

	int place;

	int power;

	int fridgeTemp = (int) ((Math.random() * 10L) + 30);

	int freezerTemp = (int) ((Math.random() * 8L) - 7);

	int freezerMax = 0;

	int fridgeMax = 39;

	int reduction = (2000 / 24) + (2800 / 24) + (1500 / 24);

	int[] hourlyUsage = { 1271, 1201, 1201, 1201, 1243, 1285, 1368, 1424, 1410,
			1452, 1466, 1480, 1494, 1494, 1494, 1494, 1536, 1620, 1718, 1773,
			1718, 1606, 1382, 1438 };

	int[] yesterUse = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0};

	Boolean powerUsed = false;

	Boolean fridgeOn = false;

	Boolean freezerOn = false;


	public int run(int hour, boolean clearedForPower) {


		/*
		 * Java counts strangely (starts at 0), so subtracting 1 to compensate.
		 */
		hour--;
		power = (int) (((Math.random() * hourlyUsage[hour]*1.97)))-reduction;
		while (power<0){
			power = (int) (((Math.random() * hourlyUsage[hour]*1.97)))-
reduction;
		}
```

```
fridgeTemp++;
if (fridgeTemp > fridgeMax) {
	fridgeOn = true;
} else {
	fridgeOn = false;
}
if (fridgeOn) {
	fridge();
}
freezerTemp++;
if (freezerTemp > freezerMax) {
	freezerOn = true;
} else {
	freezerOn = false;
}

if (freezerOn) {
	freezer();
}

// There isn't very much power being used and
// the stuff ain't done
if (clearedForPower && !powerUsed) {
	dishes();
	washer();
	powerUsed = true;
}
if (hour == 24 && !powerUsed) {
	dishes();
	washer();
	powerUsed = true;
```

```
            }
            yesterUse[hour] = power;

            return power;
        }

        public void fridge() {
            fridgeTemp = fridgeTemp - 3;
            power = power + 200;
        }

        public void freezer() {
            freezerTemp = freezerTemp - 3;
            power = power + 200;
        }

        public void dishes() {
            power = power + 2800;
        }

        public void washer() {
            power = power + 1500;
        }

}
```