Analytical Fire Modeling: Expanding upon the Elliptical Fire Theory

New Mexico Supercomputing Challenge
Final Report
April 1, 2007

Team 84
Rio Rancho High School

Team 84
Catherine Fessler
Christopher Morrison

Teacher Sponsors
Janet Penevolpe
Debra Loftin

Advisors
Mark Finney
Nadyne Shimada

Table of Contents **Page**

Executive Summary

The purpose of this project was to expand upon the *Elliptical Fire Theory* to further develop a computer application to better model the flow of fire across a virtual representation of a forest. The Elliptical Fire Theory essentially states that two-dimensional fire flow, under perfect conditions, forms a perfect circle, imperfect fire flow can be found by lengthening and shortening the radii on the circle, in accordance to the factors bearing upon fire flow, to form a multidimensional ellipse. This project aims to expand upon and validate to a further degree the Elliptical Fire Theory. Small scale empirical fire experimentation was conducted and the results were compared against the computer application. The results reflected definite similarity between the two and support the creditability of the modeling program.

The Elliptical Fire Theory is a template that can be added to. Once a variable's effect upon the fire flow process is known then it can easily be implemented upon the simulation. Small scale empirical testing was performed to determine the effect of basic kinds fuels upon fire spread rate. This data has been incorporated into the program and used to qualify the Elliptical Fire Theory. Color scanning of satellite photography was used to design real world forest inputs into the program. As more fuel type data is gained the greater degree the fore modeling process can be used effectively.

The java modeling program, also known as *Phoenix*, encapsulates the Elliptical Fire Theory and was created with the idea that it could someday be used as a tool to combat forest fires. The empirical evidence in support of the Elliptical Fire Theory gives Phoenix creditability. Now that the satellite photograph forest scanning has been developed, larger scale validation of the program can be made. Still, due to the sheer number of confounding variables, more time and research is needed to make Phoenix an accurate fire modeling program.

Introduction

The fire phenomenon has troubled humankind since its beginnings. In ancient times forest fires raged wildly across the environment. However, humans inherently attempted to protect their lifestyles and began to stop the blazes. In the United States in the early to mid-twentieth century humans were very successful in effectively dousing every forest fire. This led to the build up of perilous underbrush that eventually served as fuel for such devastating fires as the Yellowstone Fire which burned 793,000 acres or about 36 percent of the entire park (www.nps.gov). Soon people began to recognize that forest fires could not be prevented indefinitely, and that an accumulation of underbrush would fuel vast, sweeping fires that would leave no form of life in its wake. Eventually a method, known as controlled burns, was implemented. This plan proposed burning vast swaths of underbrush before it became abundant enough to propagate uncontrollable fires. However, even with such practices in place, many times fires would rage out of control. The Cerro Grande fire of 2000 burned 47,650 acres of pristine New Mexican forest and destroyed portions of Los Alamos, NM (Masse, 2003). This fire began as a prescribed burn that unexpectedly got out of hand due to weather conditions. Every year in the United States about 4.3 million acres burn, and they cost over $1 billion to contain (en.wikipedia.com) despite all efforts to control them.

The far reaching purpose of this project is to develop a fire modeling program that could be used to help prescribe controlled burns, help firefighters manage wildfires, by predicting fire direction and spread as well as predict the need for evacuation. These potential real world applications would save money and lives as well as preserve natural habitats for recreation and wildlife.

Literature Review

There are several modeling programs related to fire modeling dating as far back as 1971 (Kourtz & O'Regan). There has been a range of types of modeling including grid models that attempt to measure fire flow as flow across a square, triangular, or other geometric-shaped grids tracking fire progression (French 1992; Kourtz 1977; O'Regan 1976; Green 1983; Feunekes 1991). The percolation model, implemented by an agent based modeling system, follows more of a logic code rather than actual fire laws (Beer & Enting 1990; Borlawsky 2000). Fractal algorithms have been used to reflect uncertainty in fire flow (Clark 1994). All of the above modeling procedures are considered inaccurate in viable prediction of fire flow through a forest environment and have been abandoned.

There are some fire modeling programs (FDS, CFAST) that strive to model the precise happenings within a fire. These simulations have been validated; however, they are not good estimates for forest fires because of the innumerable degrees of precision that would bog a computer down. They are better for modeling very small-scale fires and are often used in urban arson litigation.

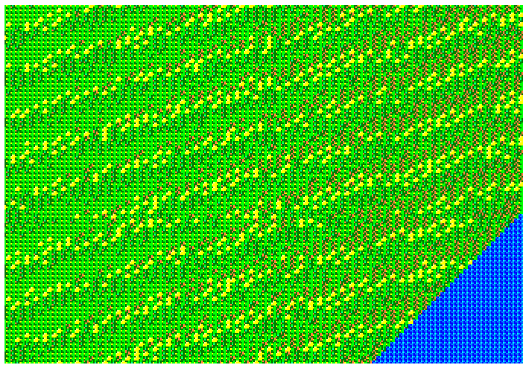
However, Huygen's Principle, a system of equations that originally described light wavelength, has been successfully applied to the forest fire modeling process. The greatest example of the success of Huygen's viability is FARSITE, the United States Forestry Service's greatest asset in wild fire control (Finney 1998), but there have been a few others (Coleman & Sullivan 1996; Richards & Bruce 1995). Huygen's Principle, along with the treatment of fire procreation as vectors, gave birth to the first pragmatic forest fire modeling approximation. FARSITE calculates one-dimensional fire flow using Rutherford's model and contains a fire line intensity and flame length models.

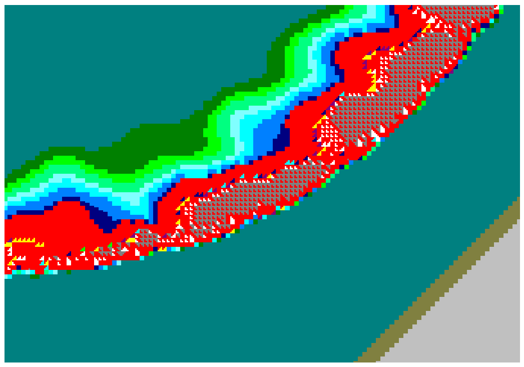
What This Project Learned and Borrowed

This project has been most influenced by FARSITE. Specifically, the treatment of fire as a vector came from FARSITE. However, Huygens's equation and the *Elliptical Fire Theory* differ significantly in their approaches to derive vector lengths and directions. On a related note, much of the inner workings of FARSITE are protected. Therefore, only general information was available for review.

Previous Years' Progress

2004-2005 Fire in the Bosque





Christopher Morrison was the only team member after his original teammate dropped out of the Challenge. A basic two-dimensional pseudo fire model was developed that used heat flow as an approximation for fire flow. Essentially, it classified fire as a patch whose temperature was over the flashpoint of the patch. It used a square grid geometric system to track fire flow. The program was written in C++ using the SDL graphics library.

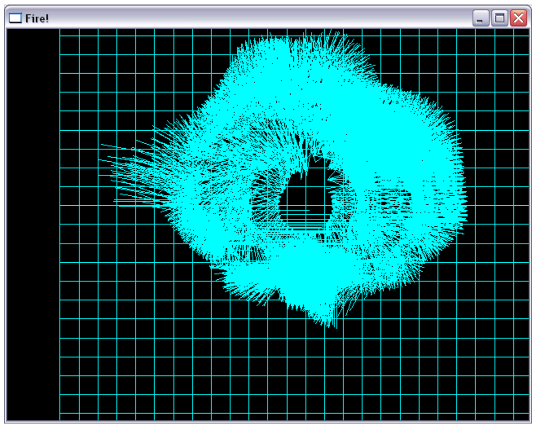
The program was limited in the fact it was a pseudo logical fire flow model. Its use of heat flow to approximate fire flow was not valid. The heat flow, the basis of entire model, was also flawed. Its methods of thermodynamics were based upon logic and perpetually ignored certain fundamental characteristics such as radiation and could only vaguely approximate heat generated by a fire. The square geometric method of tracking fire flow was also a basis for error due to fire's inability to be modeled geometrically (French 1992; Kourtz 1977; O'Regan 1976; Green 1983; Feunekes 1991).

The significant finding of this project was that a better system of fire flow needed to be developed---something that had a scientific basis rather than a pseudo logic system. This led to the development of the *Elliptical Fire Theory*.

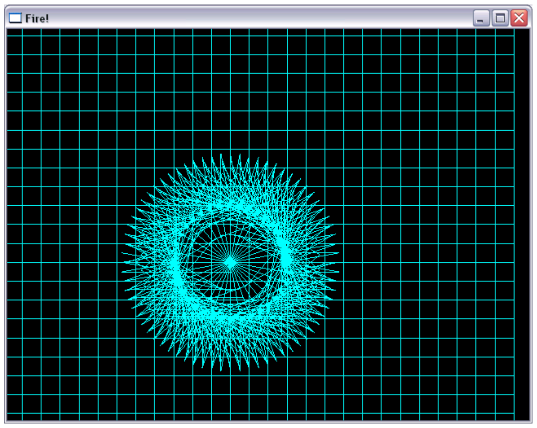
For more information, please view the final report at:
http://www.challenge.nm.org/archive/04-05/finalreports/50.pdf

2005-2006 Analytical Fire Modeling: Fire in Its Environment

Nicholas Kutac and Christopher Morrison worked diligently to develop the first viable forest fire modeling program. The fire flow process was differentiated from the heat flow and encapsulated into the *Elliptical Fire Theory*. The development of the E*lliptical Fire Theory* and the *fire flow process* highlighted the accomplishments of this phase. The treatment of fire as a vector eliminates the geometric grid problem from the year before. The program was written in C++ using the OpenGL library for graphics.



The there were a range of limitations. The most prominent is the large array of unknowns. A basic framework was developed that had a solid analytical basis for theoretical fire flow. However, no empirical data was derived to polish the program into an accurate model. On a more minor note, the C++ program was not adequate in its ability to interact with the user.



Essentially, last year a framework was developed with the E*lliptical Fire Theory*. The E*lliptical Fire Theory* is dynamic in its ability to model a fire and take into account all of the factors bearing upon vector fire flow. This current year's work concentrates on expanding upon the concepts developed last year.

For more information, please view the team website and final report:
www.analyticalfiremodeling77.com
http://www.challenge.nm.org/archive/05-06/finalreports/77.pdf

<u>Problem/Definition</u>

<u>Statement of Goals</u>

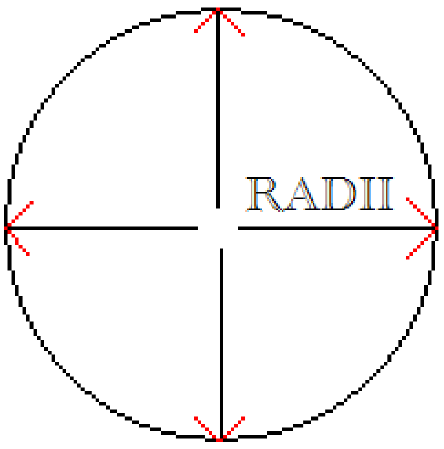
The fundamental theme of this project is to create a computer application that can accurately model fire flow through virtual forest. Last year a fire framework, centering on the Elliptical Fire Theory, was theorized and developed to track fire progression through a *virtual forest*. This year's ambition is to expound upon that theory. This project has four main goals that stem from expanding upon the E*lliptical Fire Theory* and other concepts developed in previous years.

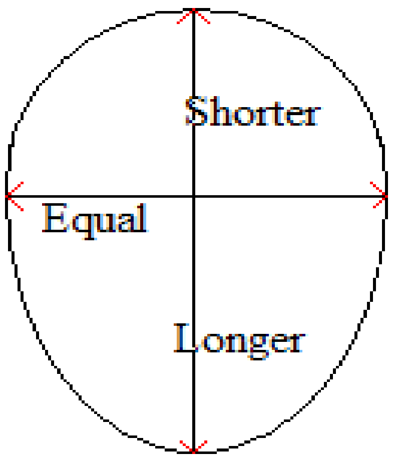
- **<u>Expansion of the *Elliptical Fire Theory* concepts</u>**
  The *Elliptical Fire Theory*, developed last year, is the basis of this year's fire flow process. The goal was to expand upon the theoretical concepts of the *Elliptical Fire Theory* specifically in the interest of vector propagation and differential calculus.
- **<u>Empirical testing and validation of the theoretical principles through small scale empirical processes</u>**
  This goal basically stands to validate the project's goals. It involves creating small scale fire experiments for comparison against the program. First, tests would be conducted to gain real world data. Then an experimental setup would be created in the real world and in the program with previously derived data. Then the experimental fire and the program fire could be compared and a judgment of the program and the *Elliptical Fire Theory* could be given.
- **<u>Use of satellite photography to model accurate terrain for a forest fire</u>**
  The rationale of this goal was simple. To create a practical, realistic fire modeling program a realistic forest must also be created. So, using satellite photographs as a basis, RGB colors (red green blue) can be simplified into different fuel types to create a virtual representation of a real forest (*virtual forest*).
- **<u>Development of an advanced graphical user interface</u>**
  A further goal of this project was to develop the program to be highly interactive. An advanced graphical user interface, named *Phoenix*, was created, using the Java swing library, to allow the user optimum control over the virtual forest environment.

Fire Theory

The *Elliptical Fire Theory*

The Elliptical Fire Theory is the underlying principle within this project. It is essentially the hypothesis. The *Elliptical Fire Theory* consists of two parts. The first is that fire flow, under perfect conditions in a two-dimensional plane, will form a perfect circle. On a three-dimensional plane, under perfect conditions, fire will form a perfect sphere. This spherical pattern of fire growth is seen in space where gravity does not affect fire flow. However, only two-dimensional fire flow is accounted for within this program because gravity would destroy the simplicity of this theory. Last year it was established that this basic principle holds true after conducting several small scale fire tests.





The second part of the *Elliptical Fire Theory* is often known as the *corollary to the Elliptical Fire Theory.* It states that two-dimensional imperfect fire flow can be accounted for by adjusting the radii (*fire radii* or *fire vectors)* on the perfect circle to form some sort of irregular ellipse (irregular circular).

Independent/Dependent Relationship

The length of the radii is essentially the dependent variable**.** The independent variables are the factors affecting the length of the radii (wind, fuel type, temperature, elevation). The Elliptical Fire Theory is the fundamental building block of this project. It breaks down fire flow into quantifiable variables that can be accounted for in a scientific approach. The *Elliptical Fire Theory* is dynamic in its ability to account for confounding variables, variables that typically throw off the results because they are uncontrolled. It can account for independent variables once their affect upon one-dimensional fire flow is known.

Accounting for Variables

There is no possible way to account for all factors bearing upon a fire. Because of the potentially unlimited number of variables that affect fire. Rather, you must attempt to model the most significant variables. Some of the most significant variables identified include the following:

- *Fuel Type*: *Fuel Type* affects fire flow because the chemical composition's inclination to fire and the surface area (or typical formation the fuel is found in) can have a profound impact upon the speed of the spreading fire.
- *Temperature*: Hotter temperatures are more inductive to fires. There is a property of every fuel called a *flash point*. It is the temperature the fuel must reach before it can become flammable. A *secondary flash point* is the temperature at which the fuel will spontaneously combust. Every mass has these two flash points. The hotter the temperature, the faster the fire will spread because the flash points of the fuels will be reached sooner.
- *Wind*: Wind propels fire flow in the direction of the wind. It is typically associated with wildfires and can often propel flames faster than a human can run.
- *Humidity*: Water will evaporate off burning fuel, straining off heat. The more moisture there is in the air or the fuel, the slower the fire spreads.
- *Elevation*: Fire flows faster uphill because on a three-dimensional plane heat travels upward pushing fire on its currents.
- *Air Pressure*: The higher the air pressure, the greater the oxygen content. Fire at sea level typically burns faster than fire at higher altitudes.
- *Gravity*: Gravity tends to push hotter, excited particles into the atmosphere and keep cooler particles at the ground level. This creates an upward air current as well as making the fire burn in an upward direction. This is why fire burns in a flame shape on earth under the influence of gravity and in a spherical shape in space.

- *Oxygen Content*: Oxygen is a limiting factor of the combustion chemical reaction that creates fire. Therefore, fire will burn at much higher speeds in places where there is more oxygen.

The Elliptical Fire Theory is set up in such a way that these variables can be treated as constants until their effects upon fire can be known. A working model must be able to describe the effects of these variables as well as what affects the variables themselves. This is especially complicated in the case of the heat model.

Assumptions

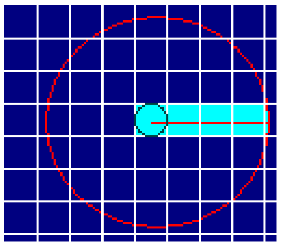
There were several underlying assumptions that serve as the foundation for the basis of the fire theory. They form the core basis upon which the program is built.

- The vital *Elliptical Fire Theory* is the first and most important. It assumes irregular elliptical growth as the basis for fire propagation.
- The *Vector Assumption* assumes fire flow can be approximated by a vector. The fire vectors are propagated out from ignition points located upon the perimeter of the fire.
- The *Wind Assumption* assumes wind's effect upon fire can be accounted for by vector multiplication of the fire growth vector by a vector proportional to the wind speed and direction.
- The *Elevation Assumption* assumes that elevation does not affect fire flow, yet gravity is influenced. Gravity is a vector whose magnitude is dependent upon the heat generated by fire which affects fire in the third dimension. Because of gravity, heated, lighter particles rise, creating an upward wind. On level ground the dot product of fire growth and elevation has no affect upon two-dimensional fire growth. However, when the elevation slopes uphill, fire will speed up to account for the effect of elevation change. This assumption has not been implemented as of yet.

One-Dimensional Fire Flow

One-dimensional fire flow is the basis for two-dimensional fire flow. In essence, it expresses a one-dimensional length based upon the independent variables. Each one of the *fire radii* on the *Elliptical Fire Theory* circle is derived from the integral below.

$$\int_0^t \overrightarrow{\text{Spread Rate}(t)} \bullet \overrightarrow{\text{Wind}(t)}\, dt = \text{Radius}$$



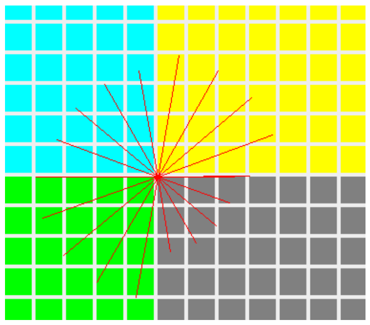
The picture to the right depicts the one-dimensional *fire radius* extending out from a central ignition point. The one-dimensional length is then imposed upon a two-dimensional plane.

Two-Dimensional Fire Flow

Two-dimensional fire flow can be derived by integrating the one-dimensional fire flow for 360 degrees. This gives a polar equation to describe the fire's growth out from the central ignition point. This essentially gives a fire perimeter that is, as the *corollary* describes, an irregular ellipse.

$$\int_0^{2\pi}\int_0^{t}\left(\overrightarrow{\text{Spread Rate}(t)}\cdot\overrightarrow{\text{Wind}(t)}\,dt\right)d\theta=\text{irregular ellipse}$$

The integral of the one-dimensional integral from 0 to 2π with respect to feta equals the irregular ellipse.



The picture to the right illustrates the multiple one-dimensional radii extended out from the central ignition point on a two-dimensional plane (*fire arc*). To save computer processing power on a computer, the integral above is approximated with the sum limit below. The fire arc pictured above is basically an 18-sided polygon approximation of the irregular ellipse. As the number of vertices (*fire radii)* increases, the greater the processing power required to compute the fire arc.

$$\text{Lim}\sum_{d\theta\to 0}^{2\pi}\int_0^{t}\overrightarrow{\text{Spread Rate}(t)}\cdot\overrightarrow{\text{Wind}(t)}\,dt=\text{Approximate Ellipse}$$

This is an approximation of the two-dimensional irregular ellipse dθ can never reach 0.

Further Breakdown of Mathematics

The one-dimensional fire spread is broken down into two vectors. The reason for the breakdown is due to the different directions of the forces' action upon fire flow.

$$\int_0^t \overrightarrow{\text{Spread Rate}(t)} \bullet \overrightarrow{\text{Wind}(t)}\, dt = \text{Radius}$$

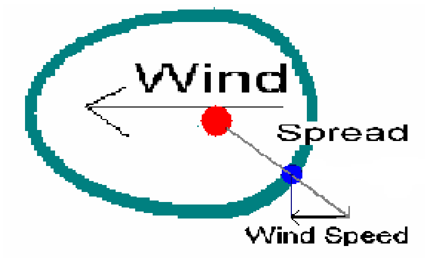
The integral from time equals zero to time t of the vector spread rate multiplied by the vector wind equals a single one dimensional radius.

- **Spread Rate Vector**

This vector operates in a direction outward from the central ignition point. It is the composed of all environmental independent variables except wind (elevation, fuel type, temperature). This vector represents the fire propelling itself outward. The function to describe the spread rate vector is not continuous because a forest environment cannot be described an equation. Rather, it is described by the environmental traits (pine tree here, bush over there). The spread rate vector is measured in centimeters per second and was determined empirically for each fuel or arbitrarily assigned.

- **Wind Vector**

This vector operates in the direction of the wind and is proportional to the wind speed. The proportionality constant is known as the *wind coefficient.* It is separated from the spread rate vector because it operates in the direction of the wind, whereas the spread rate vector operates in a direction outward from the central ignition point. The picture the right depicts, the vector dot multiplication of the wind speed to the spread rate.



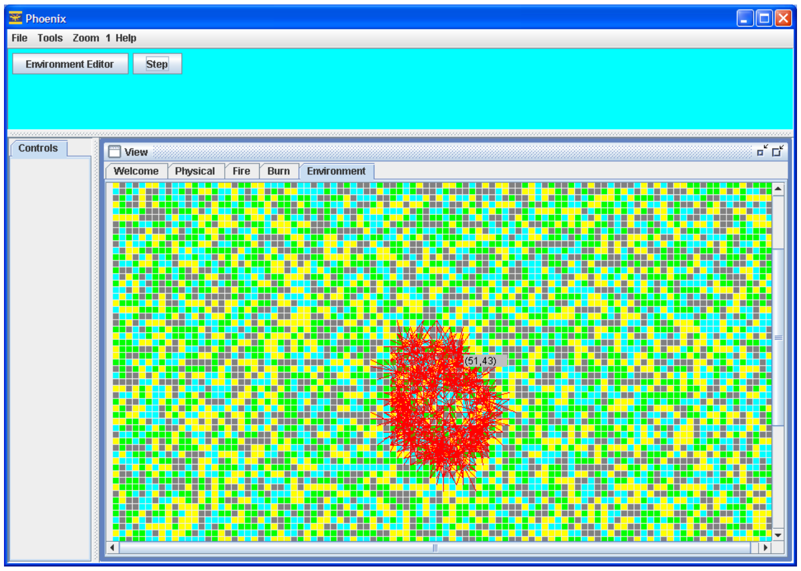
Program- Phoenix

Java Programming Specifics

Phoenix is a Java application built on the NetBeans 5.5 IDE with the Java JDK 5.0. Phoenix encompasses 16 .Java files. The graphical user interface is implemented by the Java swing library. There are approximately 4,000 lines of code throughout the .Java files all of which are original. The fire flow process is encapsulated within the .Java file FireFlow.java. The processing requirements for this program are directly proportional to the area of the burning fuels. In other words, the bigger the fire, the greater the processing requirements to compute.

The Basic Environment

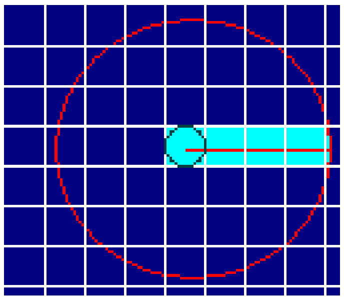
The environment in which the fire persists is a system of *patches*. Within those *patches*, there are more *mini-patches*. The *patches* encapsulate their own specific set of variables which include the fuel type, temperature, wind, and other environmental variables. The collection of patches, as a whole, is known as the *virtual forest*. Each patch type is unique and causes a fire to either accelerate or slow down as it passes over the patch. The patches then have individual sections within them called *mini-patches*. These *mini-patches* are then used to further track the exact movement of the fire as it crosses the *virtual forest*.



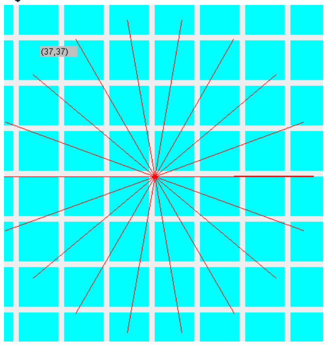
Fire Flow Process

The different *patches* in the *virtual forest* contain different values of what is called *spread rate*, which is the rate at which fire can spread through the patch. The spread rate is be manipulated by factors such as heat and humidity. This is the same spread rate that is used in the one-dimensional integral. For each time step, the fire spreads across the patches until the accumulated value of the time steps reaches a point called the maximum spread rate (top picture). This process is repeated until a fire arc is formed, making the perimeter of the fire for the certain time step. The *fire arc* shown here (second from top) is the same process as shown before, just taken out many more iterations. The approximate ellipse summation basically represents the fire arc with a 2π/dθ sided polygon. The endpoints of the fire arc are then stored, and during the next step, are transformed into fire arcs themselves. This process is computed once per time step, producing fire perimeter results such as in the bottom picture. This picture depicts the result of several time steps in a non-uniform virtual forest and shows the created *fire perimeter*.

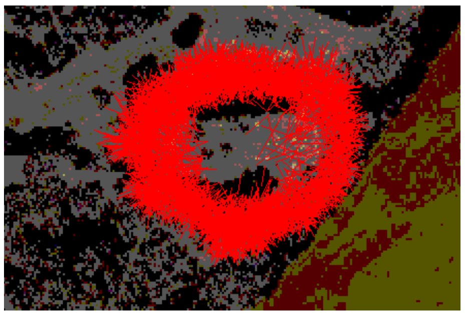
$$\int_0^t \overrightarrow{\text{Spread Rate}}(t)\cdot\overrightarrow{\text{Wind}}(t)\,dt = \text{Radius}$$



$$\text{Lim}\sum_{d\theta\to 0}^{2\pi}\int_0^t \overrightarrow{\text{Spread Rate}}(t)\cdot\overrightarrow{\text{Wind}}(t)\,dt = \text{Approximate Ellipse}$$



Fire Perimeter Reduction



As the fire grows larger, only the perimeter is needed to map the fire flow. If the computer took every fire endpoint and arced it, its efficiency would be x^N, whereas 'x' is the number of endpoints the fire arc contains and 'N' is the number of time steps. Therefore, a system was developed to record previously burned spots and reject new fire arcs in those burned spots. This would increase the efficiency of the program by preventing "burned" patches from being burned a second time.

Program Accountancy

The *Elliptical Fire Theory* allows for all unaccounted variables to be held constant. This allows for unknown confounding variables to be held constant. This project accounts for two specific variables- wind and fuel type. These were the only variables the team felt were able to be reasonably approximated. The temperature model from last year still exists; however, it has been deemed unrealistic. A central ethic in the programming has been to allow for the incorporation of various variables and their models, but not yet implement them until a reliable model can be developed.

- Fuel Type Model: The fuel type model is simply the different rates fire spreads over a material. This model essentially assigns a spread type to each kind of fuel. Real world values can be determined empirically. For example, the speed at which fire spreads over similar brush can be determined and assigned to all brush fuel types.
- Wind Model: The wind model essentially states that wind and fire flow can be expressed as a vector and the vector dot product of the fire path vector and the wind vector expresses wind's direct impact upon the fire flow. This is discussed in more detail on pg. 13.

Miscellaneous Interactive Features

One specific goal was to make Phoenix as interactive as possible. This would give the user greater control over the fire and allow for greater scope of modeling. Some of the notable features include the following:

- A multilayer Java swing library interface and display system maximizes the visual capabilities of the program.
- Four separate views of the fire, physical, environmental, burned, and fire, allow for a better view of exactly what is going on inside the fire.
- The Environment Editor allows for real time input of variables such as wind speed and direction.
- Satellite Photograph Scanning can be used to make user defined realistic *virtual forests.*
- The Zoom in and out feature allows for a micro and macro view of fire progression.

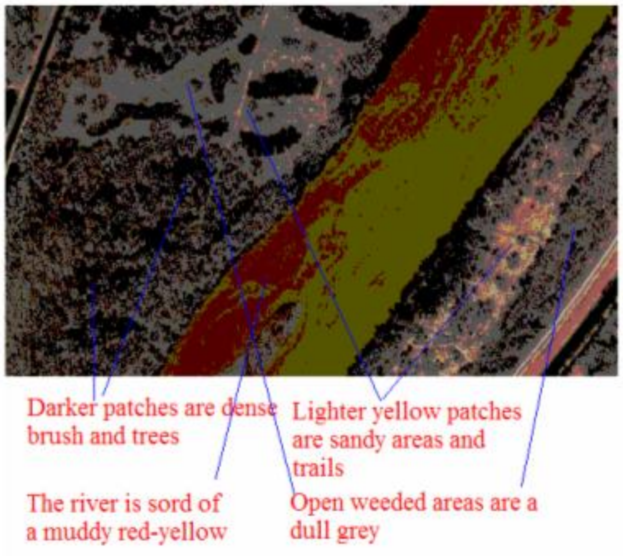
- The patch trait information can display the state of any individual patch.
- A statistics system keeps track of several key points of information.
- The user can start a fire anywhere within the *virtual forest*.

  Expect many more features to be added in the near future.

Satellite Terrain Mapping

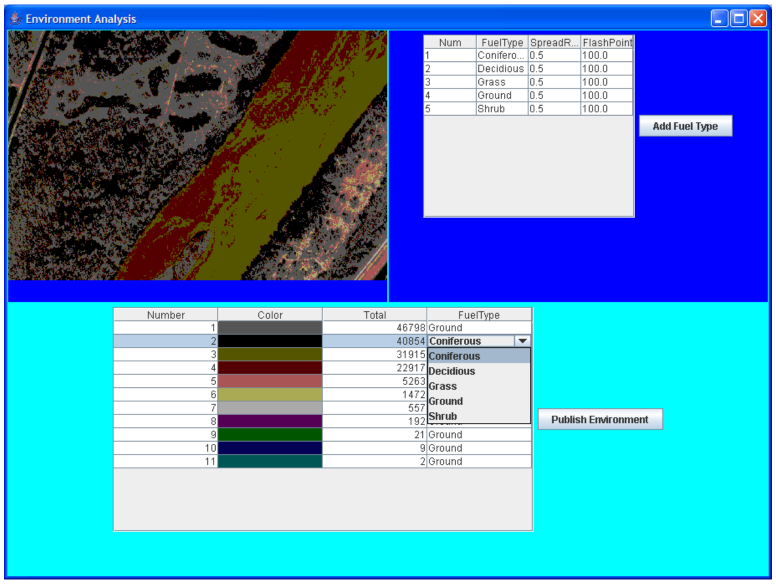
Creating a Realistic Forest

The basic idea behind using a satellite photograph to create a realistic forest was that different *fuel types* exhibit different colors. By breaking down a satellite photograph into its basic Red-Green-Blue components, you can group similar RGB values as the same fuel type. The top picture shows a picture of the Bosque South of the Montaño Bridge in full 256 color (256^3 colors available). The second picture is the same picture only in 3 colors (3^3 or 27 available colors total). The program essentially rounds every pixel in the original photograph into the closest of one of 27 available colors (*spectrum classification*).


Darker patches are dense brush and trees
Lighter yellow patches are sandy areas and trails
The river is sord of a muddy red-yellow
Open weeded areas are a dull grey

```
int Red= Round(red);
int Green= Round(green);
int Blue= Round(blue);

public int Round(int color)
{
            return (int)Math.floor((color/(255/colorNum))*255/colorNum);
}
//ColorNum is the color (ex: 256 color)
```



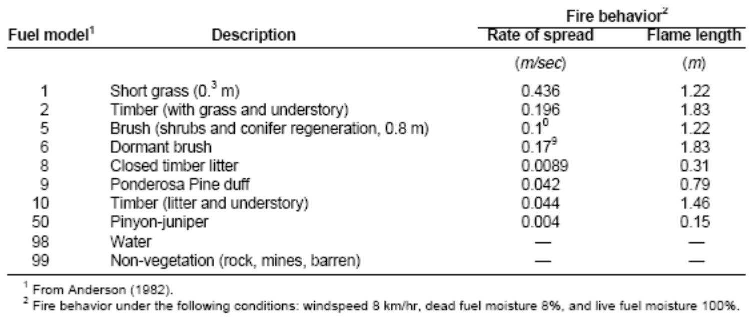
Num	FuelType	SpreadR...	FlashPoint
1	Conifero...	0.5	100.0
2	Decidious	0.5	100.0
3	Grass	0.5	100.0
4	Ground	0.5	100.0
5	Shrub	0.5	100.0

Number	Color	Total	FuelType
1	⬛	46798	Ground
2	⬛	40854	Coniferous
3	🟫	31915	Coniferous
4	🟥	22917	Decidious
5	🟥	5263	Grass
6	🟨	1472	Ground
7	⬜	557	Shrub
8	🟪	192	Ground
9	🟩	21	Ground
10	🟦	9	Ground
11	🟩	2	Ground

This basic color model serves as a fuel type approximation map. The bottom picture shows the process by which the user initializes the data. A table is generated containing the intrinsic colors and the user specifies the type of fuel associated with the color. Fuel types can be generated, added, and changed on another table.

<u>Field Testing</u>

Once a particular area is picked, several traits must be known about the area before the program will model it. Several of these traits (*spread rates* of native flora) have to be obtained or approximated from the ecosystem. The purpose of field testing was to gain information about the fuel types of an area. Once the fuel types could be determined, then either from empirically-derived data or from existing research, a realistic forest environment could be mapped.



Fuel model[1]	Description	Fire behavior[2]	
		Rate of spread	Flame length
		(*m/sec*)	(*m*)
1	Short grass ($0.^{3}$ m)	0.436	1.22
2	Timber (with grass and understory)	0.196	1.83
5	Brush (shrubs and conifer regeneration, 0.8 m)	0.1^{0}	1.22
6	Dormant brush	0.17^{9}	1.83
8	Closed timber litter	0.0089	0.31
9	Ponderosa Pine duff	0.042	0.79
10	Timber (litter and understory)	0.044	1.46
50	Pinyon-juniper	0.004	0.15
98	Water	—	—
99	Non-vegetation (rock, mines, barren)	—	—

[1] From Anderson (1982).
[2] Fire behavior under the following conditions: windspeed 8 km/hr, dead fuel moisture 8%, and live fuel moisture 100%.

The data above is existing research (Keane, Robert 2000) that could be used to render a basic approximation of a native environment once a field test is conducted to determine the fuel types associated with the *spectrum classification* different colors. Appendix C details a minor *field study* conducted upon an area of the Albuquerque Bosque.

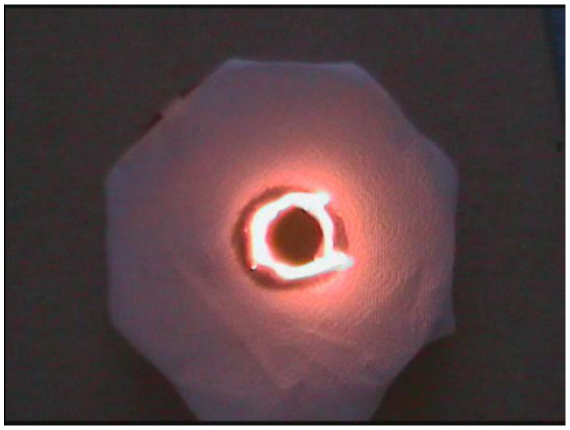
<u>Implications</u>

The addition of satellite terrain mapping to *Phoenix* increases the scope of *Phoenix* to real world fire situations. Now, empirical validation on a grand scale could occur. If data could be collected, both from a thorough *field study* and from a real fire, then this program could have a real world assessment of its modeling capabilities.

Empirical Validation

This project has reached a point to where it can viably go no further without being validated. The *Elliptical Fire Theory* is essentially the hypothesis. Last year the first clause of the *Elliptical Fire Theory* was Validated. Fire under perfect conditions will grow in a perfect circle. However, the corollary has not been validated. For this project to progress, the underlying assumptions that compose it must be validated. Therefore, a series of empirical experiments were devised to test small scale real-world fire experiments against the Phoenix.

Experimentation



The empirical experimentation consists of two parts. The purpose of the first portion was to attain the real world inputs for *Phoenix*, the computer application. The second part consisted of comparing results from a real world experiment against the program. Spread rates could be determined by recording the fire with a camcorder and freeze framing the video at half second intervals. Based upon the angle of the lens, the length of the diameter, and the number of pixels burned an approximate spread rate can be determined. The spread was measured at the point where the charred black line intersected the colored fuel.

$$\text{Spread Rate} = \frac{\text{width} \times \Delta\text{pixels}}{\text{width in pixels}}$$

- Real World Inputs: The spread rate is the ultimate ambition of the first phase of testing. Several fuels were tested including: Brawny paper towel, tissue paper, notebook paper, computer paper, and construction paper. The experiments were limited because larger fuels such as cardboard and wood and conglomerations of fuels such as would be found on a forest floor were either too large for the safe small-scale experimentation or required a much higher flash point then could be delivered without a laboratory condition.

- Comparison: Once real world inputs (*spread rates)* are derived, they can be placed within the program. A real world and virtual experiment can be set up that uses the previously derived data. The experiment can be any variation of studied fuels. The experiments can be conducted and compared. Similar results will support the program and the *Elliptical Fire Theory*.

Comparison

A basic visual comparison can be used to generate a qualitative assessment of the program's effectiveness (Appendix B). A more quantitative approach in progress is known as *interpolation comparison. Interpolation comparison* essentially is a ratio inter-lapping area between the real world scenario and the computer program. This requires the writing of a computer program that can determine area burned from a picture. The interpolation program is still in development stages. Also a linear regression can correlate the size of the real and experimental fire. Pearson's r correlation statistic can be used to assess the degree of likeness between the fires. Once the burned area program is finished, a finite quantitative evaluation of *Phoenix* can be produced.

Implications

The results of the validation are essentially the judge of the program and all the work that has been done. If they are not alike, then there is a problem with *Phoenix.* If the results are similar, then *Phoenix* must have some ability to model fire flow, at least on a small scale.

Conclusion

The validation experimentation supports the Elliptical Fire Theory hypothesis as a valid basis for a fire flow model. The empirical experimentation results confirmed Phoenix's ability to model simplistic fire on a small scale. The successful incorporation of the *Elliptical Fire Theory* into *Phoenix* is a strong and viable start on the road to creating a feasible forest fire modeling program.

This project was named Analytical Fire for a reason. After all, forests are far too precious to be burned at a whim. However, now that a basis has been laid, it needs real world values to be placed in variable areas. Efforts to determine spread rates of different fuel types have only begun. Heat's effect upon fire is known, but not precisely defined. Therefore, this project needs an empirical basis as well as an analytical basis to become a more accurate fire flow forecaster.

Another inaccuracy of this project involves the sheer number of factors involved in determining fire flow. Fire flow cannot be accurately modeled without knowing these factors and their affect upon fire. However, the nature of the Elliptical Fire Theory allows for these variables to be accounted for easily once their affects are known. As more and more variables are accounted for, more accurate fire flow approximation can be included in the program.

Satellite terrain mapping helped to diminish the number of unaccounted variables. By using colors, a forest can effectively be mapped for its *fuel type* distribution. In addition, it yielded the interesting prospect of large scale empirical experimentation. Once in depth *field testing* is done, a more rigorous test of the Elliptical Fire theory could be arranged.

The conclusion is that this project is a success. The amount of progress accomplished in such a short amount of time is remarkable. The flow of fire across the virtual environment is rational. The results of this project are viable. The program only needs more work, time, research, and an empirical basis to become an asset which could be used in real world applications.

Limitations

Real World Data

The largest limitation to this project was the large amount of field data needed to make it into a viable forest fire modeling program. The theoretical basis of the program is sound; yet, for it to be practical, many real world values must be empirically derived. There is some research done by the United States Forestry Service in these areas; however, this research is only semi-compatible with this program because of the difference in mathematical models between *Phoenix* and other forest fire modeling programs. The data that is compatible, mainly the spread rate, has been implemented somewhat in the satellite photographs. However, this program has only just reached the level to where it can use this existing data.

Also, the data is often as hindering as it is helpful. Much data available is dependent upon even more specific data that could only be provided by empirically testing the native area where the fire is being simulated.

- The spread rates for the differing fuels needs to be empirically derived. Many typical forest fuels would need to be derived on a larger scale for composite fuels (leaves, sticks, rocks, grass combined) that would be seen in a forest.
- A large amount of empirical data would need to be collected to determine the effect that temperature has upon fire flow, to determine the *wind coefficient*, to determine the effect humidity wind has upon fire flow, the effect of gravity upon fire flow, et cetera.
- An even larger amount of empirical data is needed to determine the effectiveness of the program in modeling fire flow.

There are many fire experts who have worked their whole lives gathering similar, or perhaps, identical data. It would be advantageous much avail to this project to use their data. Yet, much of the data is copyrighted and withheld from easy access.

Laboratory Conditions & Equipment

A limiting factor in this project was the availability of a laboratory setting and equipment needed to increase efficiency. For example, the spread rate for medium and heavy fuels could not be determined because their *flash points* were too high to ignite under outside non-laboratory conditions. Also, the empirical tests were not done under perfectly controlled circumstances because outside conditions fluctuate. A laboratory condition would have fixed this problem. Equipment was missing as well. A thermometer that can read temperatures within a fire (infrared) for the empirical testing and a hand held GPS for satellite field testing would have been helpful.

Heat Model & Other Important Variables

The missing heat model is perhaps the greatest hindrance to *Phoenix's* continued progress. In real life the spread rate is highly dependent upon the heat model. Without a heat model, the *flash point* aspect of a real forest fire cannot be successfully implemented. Because of this *flash point limitation, Phoenix* has one definite flaw; it burns everything to ashes. The only force to stop is if the spread rate of the material is so low that *Fire Perimeter Reduction* eliminates the ignition points. Everything combustible in the forest would be burned down. This is not realistic. The heat model from last year still exists. It was deemed unrealistic because it had a flawed basis. Now that the *Elliptical Fire Theory* has been validated, more research should be put into working on the heat model. Two more models that deserve attention are the humidity precipitation models.

Three-Dimensional Fire Flow

The fact is the world is not flat. Two-dimensional fire flow is a good approximation. A majority of professional forest fire modeling programs are two dimensional. *Phoenix* has the capability to model land slope with the *elevation assumption* (although it requires a working heat model). When the third dimension is added, a lot more factors become issues. For example, in a realistic fire, fire flows both across the ground and in the trees. Many forest fires spread to a greater degree in the trees and to a lesser degree on the ground. *FARSITE* and a few other models like it are 2.5 dimensional models. They contain equations that calculate flame height and assume trees have a canopy height. If the flames are long enough to reach the canopy, then a fire begins at the top of the tree and is known as a *crown fire*. A crown fire model for *Phoenix* is in development. Certain programs are fully three dimensional, but they would not be applicable to forest fires. The amount of processing time is extraneous and would be of no avail in a real time fast-paced forest fire.

Satellite Mapping Limitations

There are three errors that are associated with using a single satellite photograph to determine fuel type.

- Color aliasing: Some fuel types have similar colorings. This would cause an RGB approximation to have aliased similar colored fuel types, causing errors.
- Underbrush: Underbrush cannot be determined from an aerial view. This could only be approximated for a large area by a field study.
- Elevation Corrections and Shadows: Elevation will cause errors in a straight up view. Shadows are often increased in areas with a high density of trees. These shadows throw off the *spectrum classification*.

Something that could solve these errors would be to analyze multiple satellite photographs and analyze them during different seasons. This would maximize the contrasts of color, increasing the accuracy of the *spectrum classification.*

Discussion

Analytical or Empirical

The title “Analytical” fire modeling is perhaps contradicted by the number of times “empirical” is mentioned. The basis of this project is analytical. The *Elliptical Fire Theory* is an analytical foundation, and along with other assumptions mentioned in the *Fire Theory* section of this paper, they form the backbone of the modeling process. Now, this project has grown out of the analytical stage in which it was founded. Now real world data values must be found; real world comparisons must be made to qualify the program. This project was formulated with the idea it could be used to model real world forest fires. So the line must be crossed from the theoretical to the tangible.

X Y Relationship

Science is all about taking something complex and simplifying it into something quantifiable, something methodical. The scientific method basically relates one variable to another, some amount of x equals y. In this case the Elliptical Fire Theory can express y, the length of the radii, through a multivariate x. Once the relationship between the variables and results is found, then it would only be a matter of plugging the variables into an equation and obtaining y.

Satellite Photography

The satellite mapping went well, producing many realistic terrains. It is a completely different realm of experimentation as compared to the fire tests. Once the environmental local fuel types are tested for their traits and that data placed in the program, it would be interesting to actually compare large scale fires against the program. There were several limitations such as underbrush and color aliasing that cannot be accounted for; however, an approximation is much better than nothing. In the future elevation could be accounted for by layering the satellite photo with a rasture map (elevation map with lines depicting altitude).

Why Not Use Existing Fire Models?

FARSITE and a few other forest fire modeling programs are excellent forest fire models. They are the product of many individuals' life work. Why not just leave the experts to find the answers? The truth is, without a lifetime of work, *Phoenix* can only hope to compare to the seasoned, professional models. However, this team is in search of knowledge. The purpose of the Challenge is to inspire students to take on daunting challenges head on. This is what team 84 is attempting. Perhaps, *Phoenix* will grow to become a heavyweight fire modeling program. Perhaps some of the ideas in this paper have great merit and will evolve to improve. Only time will tell. One thing is for sure, team 84 has improved their skills and abilities because of their attempts to defy the odds.

Mentor, Mentor, Anywhere?

This project has had an extremely difficult time attracting mentors that are knowledgeable in the area of fire science. Several fire experts were contacted, mainly through e-mail. Responses ranged from "good job" and "good luck" to no response at all. None felt obligated enough to brandish the "mentor title." Mark Finney is listed as a project advisor because he was one of the only fire experts to respond and give a few positive comments about the program. However, contact was lost with him for unknown reasons. Several contacts were made in the forestry service; however, none really had the technical expertise in the area of forest fire modeling. An effort was made to contact knowledgeable mentors.

The Future

This project has much potential. Since the *Elliptical Fire Theory* was proven to have a high degree of validity, then it can be used as a cornerstone to build upon. It would only be a matter of empirical testing to determine the effects of each confounding variable upon the length of the radii in the *Elliptical Fire Theory*. The next steps would be to compare the program against actual forest fires and to add a heat model to allow for accommodation of *flash points*.

Bibliography

Beer, T, Enting, I. (1990). Fire spread and percolation modeling. Mathl. Comput. Modeling 13(11):77-96.

Bonsor, K. (2004). *How Wild Fires Work.* Retrieved October 3rd 2005. from http://science.howstuffworks.com/wildfire.htm

Borlawsky T. (2000) Forest fire simulation using percolation theory. (Independent Research).

Colemen, J.R. and A.L. Sullivan. 1996. A real-time computer application for the prediction of fire spread across the Australian landscape. Simulation 67 (4):230-240

EFunda.com (2005). Heat Transfer. *EFounda.com*. Retrieved December 12th 2005 from http://www.efunda.com/formulae/heat_transfer/home/overview.cfm

Finey, M. (2002) Australian Mathematical Society *Fire Growth Using Minimum Travel Time Methods.*

Finney, M. (1998). FARSITE: Fire area simulator- model development and analysis. *USDA USFS.* (Research Paper). RMRS-RP-4 Revised

Finney, M and Patricia L Andrews. (1994). The *FARSITE* fire area simulator: Fire management applications and lessons of summer 1994. (Research Paper).

Feunekes, U. 1991. Error analysis in fire simulation models. M.S. thesis, University of New Brunswick, Fredericton.

Fire.org. Public domain software for the wildland fire community.

French, I.A., D.H. Anderson, and E.A. Catchpole. 1990. Graphical simulation of bushfire spread. Math. Comput. Model. 13(12): 67-71.

Green, D.G. 1983. Shapes of simulated fires in discrete fuels. Ecol. Mod. 20:21-32

Harris, T. (2005). *How Wild Fire Works* Retrieved December 13th 2005 from http://science.howstuffworks.com/fire.htm

Heidorn, Keith C. (2000) Weather Almanac for October 2000; The Great Fires of October 1871. Retrieved December 8th, 2005 http://www.islandnet.com/%7Esee/weather/doctor.htm

Keane, Robert E., Mincemoyer, Scott A., Schmidt, Kirsten M., Long, Donald G., Garner, Janice L. (2000). Mapping vegetation and fuels for fire management on the Gila National Forest Complex, New Mexico, [CD-ROM]. Gen. Tech. Rep. RMRS-GTR-46-CD. Ogden, UT: U.S. Department of Agriculture, Forest Service,Rocky Mountain Research Station, 126 p.

Kourtz, P. and W.G. O'Regan. (1971). A model for a small forest fire…to simulate burned and burning areas for use in a detection model. For. Sci. 17(2):163-169.

Masse, B. & Nisengard, J. (2003). *Cerro Grange Fire Assessment Project* Cultural Resources Report No. 211.

Morrison, C,G, Kutac N, J. (2005) Analytical fire modeling: Fire in its environment. *AISC* (Research Paper).

O'Driscoll, P. (2005). Studies at odds over logging after wildfires. *USA Today* Nov. 2nd

Rein, G, Amnon B, Carlos Fernandez-Pello, A, Norman, A. (2005). A comparison of three models for the simulation of accidental fires. *Journal of Fire Protection Engineering* 16(3) 183-21.

Richards, G, Bryce, R. (1995). A computer algorithm for simulating the spread of wildland fire perimeters for heterogeneous fuel and meteorological conditions. Int. J. Wildl. Fire. 5(2):73-80.

Rona, A. (2003). *Conduction.* Retrieved December 6th, 2005 http://www.le.ac.uk/engineering/ar45/eg1100/eg1100w/node12.html

Taftan Data (1998). *Fourier's Law of Conduction.* Retrieved December 6th, 2005 http://www.taftan.com/thermodynamics/FOURIER.HTM

The Official Website of Yellowstone National Park, Wildland Fire http://www.nps.gov/yell/nature/fire/index.htm

Wang, T (2004). Environmental assessment for the Bosque wildlife project, Bernalillo, Sandoval counties, New Mexico. *Army Corp of Engineers* (Research Paper).

Wikipedia: The Free Encyclopedia
http://en.wikipedia.org/wiki/Newton's_law_of_cooling
http://en.wikipedia.org/wiki/Thermal_conductance
http://en.wikipedia.org/wiki/Heat_conduction

www.nifc.gov (last updated, 2002). Historical Wildfire Statistics. Retrieved December 6th 2005 http://www.nifc.gov/stats/historicalstats.html

Acknowledgements

Nick Bennet: Thank you for all you programming help over the past four years.
Consult@challenge.nm.org : Thank you for being so flexible and accommodating for busy teenagers schedules at the February Evaluations.
Mark Finney: Thank you for being the only fire expert to reply back and for your suggestions.
Debbie Loftin: Thank you for your inspiration and never ending support.
David Morrison: Thank you for helping us build a "professional board."
Cheryl Morrison: Thank you for editing our paper.
Janet Penevolpe: Thank you for being our teacher sponsor and your criticisms.
Nadyne Shimada: Thank you so much for your uncanny ability to give great advice.
Team's Parents: Thanks to the moms and dads for their financial, encouragement, and labor that contributed to making team 84 a success.

Appreciation

On a more personal note, this is Christopher Morrison's last Challenge. He has been participating and enjoying the challenge for four years. He would like to thank everyone who has worked to make the Challenge what it is. Because of his experience in the challenge, he participated in an internship at Sandia National Laboratories last summer and is going to be rehired this summer. The challenge made a major difference in his life. So thank you David, Celia, Betsy, Nick and everyone who has made the Challenge what it is.

Appendix A: Glossary

Ambient Heat – Starting from the forest's ambient temperature and raising as a fire moves, approaches, and decreases as the fire passes by or dies our for lack of fuel or oxygen.
Ambient Temperature- Temperature to which environment will cool based upon *Newton's Law of Cooling*
Corollary to the Elliptical Fire Theory- This is the second part of the *Elliptical Fire Theory* which basically states irregular fire flow can be accounted for by lengthening and shortening radii to form an irregular ellipse
Dry Fuel- Fuel classified as having a lower moisture content requiring a lower *flash point* than *wet fuel.*
Elevation Assumption- Changes in ground level affect fire because of wind generated by gravity.
Elliptical Fire Theory- Theory that fire will form a perfect ellipse under perfect burn conditions. This implies that to properly account for *fire flow,* the polar radii can be shortened and lengthened accounting to different variables' effects on fire.
FARSITE- Forestry Service's fire approximation program based on *Huygen's Principle*.
Field Test- An environmental study to decipher what type of fuels are present in a forest.
Fire – The chemical reaction of fuel, ignition heat, and oxygen which generates additional heat burning fuel and oxygen. Fire is self-perpetuating as long as sufficient fuel and oxygen remain.
Fire Acceleration- As fire grows larger, it accelerates faster. The heat assumption may account for this phenomenon.
Fire Arcs- The collection extensions of the *fire radii* from a single point.
Fire Flow – The most commonly understood definition of a fire, defined as the actual, observable movement of a fire through a forest.
Fire Flow Process- The process of starting with initial ignition points fire arcing them and creating a new *fire perimeter.*
Fire Perimeter- Collection of *fire arcs* that, with *fire perimeter reduction,* form a perimeter around the fire.
Fire Perimeter Reduction- For better efficiency, unneeded *fire arcs* are deleted in the middle of the fire.
Fire Radii- Polar extensions from a central ignition point. For example, 3 units at 30 degrees. Fire radii are susceptible to changes in lengths based upon factors involved in *fire flow*.
Fire Vector- *See Fire Radii.*
Flash Point- The temperature a fuel must reach before it can be burned. A *Secondary Flash Point* is the temperature at which a fuel spontaneously combusts.
Flash Point Limitation- Because Phoenix is lacking an accurate heat flow model, flash points cannot be instantiated. This causes the current program to burn everything to ashes.
Fuel- Every patch has fuel with different moisture contents. These moisture contents are simplified into two types of fuels requiring different prerequisites to burn then.

Fuel Type – Different types of fuels where each contain its own heat generation factors, ignition point values, dry and wet fuel ratings, etcetera. Some examples include trees and shrubs.
Fuel Type Assumption- As fire flows over different types of ground coverage, it will spread faster or slower depending upon the fuel type's atomic structure and typical surface area.
Heat Flow – The transfer of heat in a forest as a fire moves, as well as, during everyday circumstances. Generally connected to *fire flow* because of the natural properties of fire which gives off heat as a byproduct, but can also be associated with ambient warming and cooling with the time of day.
Heat Assumption- Different amounts of heat cause fire to spread differentially. The hotter it is, the faster the *fire flow*.
Huygen's Principle- *FARSITE* equation that uses wind and elevation to determine an ellipse that the fire will follow.
Ignition Point – Minimum temperature at which a fuel will ignite and become self sustaining based on fuel type and environmental factors.
Interpolation Comparison- A process of comparing fire progress between two a model and real life visually by overlaying pixels (model fire pixels/ real fire pixels).
Moisture Assumption- More moisture will slow fire spread.
Mini-Patches- *Patches* divided into 100 equal parts to better keep track of certain aspects in the program.
Patches- Division of fuel type exactness and area. One *patch* is approximately 1 sq meter.
Phoenix- The name of the java computer application that encapsulates the fire flow process.
Piloted Flash Point/ Instantaneous – Point at which patch will combust whether or not fire is present.
Rapid Oxidization- The scientific term for the chemical reaction of fire. Rapidly taking of the oxygen in the fuel.
Spectrum Classification- The process of simplifying a typical 256-color picture into a lesser grade. This is used to group colors on a satellite photograph and create a virtual forest once a *field test* has been conducted.
Spot Fires- A term for sparks that fly into the air and create fires in new places down wind.
Spread Rate- The rate at which fire can spread across the ground. Influenced by many environmental factors, heat, wind, and other factors.
Unpiloted Flash Point/ Basic – Temperature which is required to burn a certain fuel.
Virtual Forest- Collection of patches that make up the forest environment.
Wet Fuel- Fuel classified as having a higher moisture content requiring a higher *flash point* than *dry fuel* .
Wind Assumption- Wind's effect upon fire can be described as a vector of wind speed and fire spread.
Wind Coefficient- The wind proportionality constant.
Wind Vectoring- The process of correcting fire flow for wind.

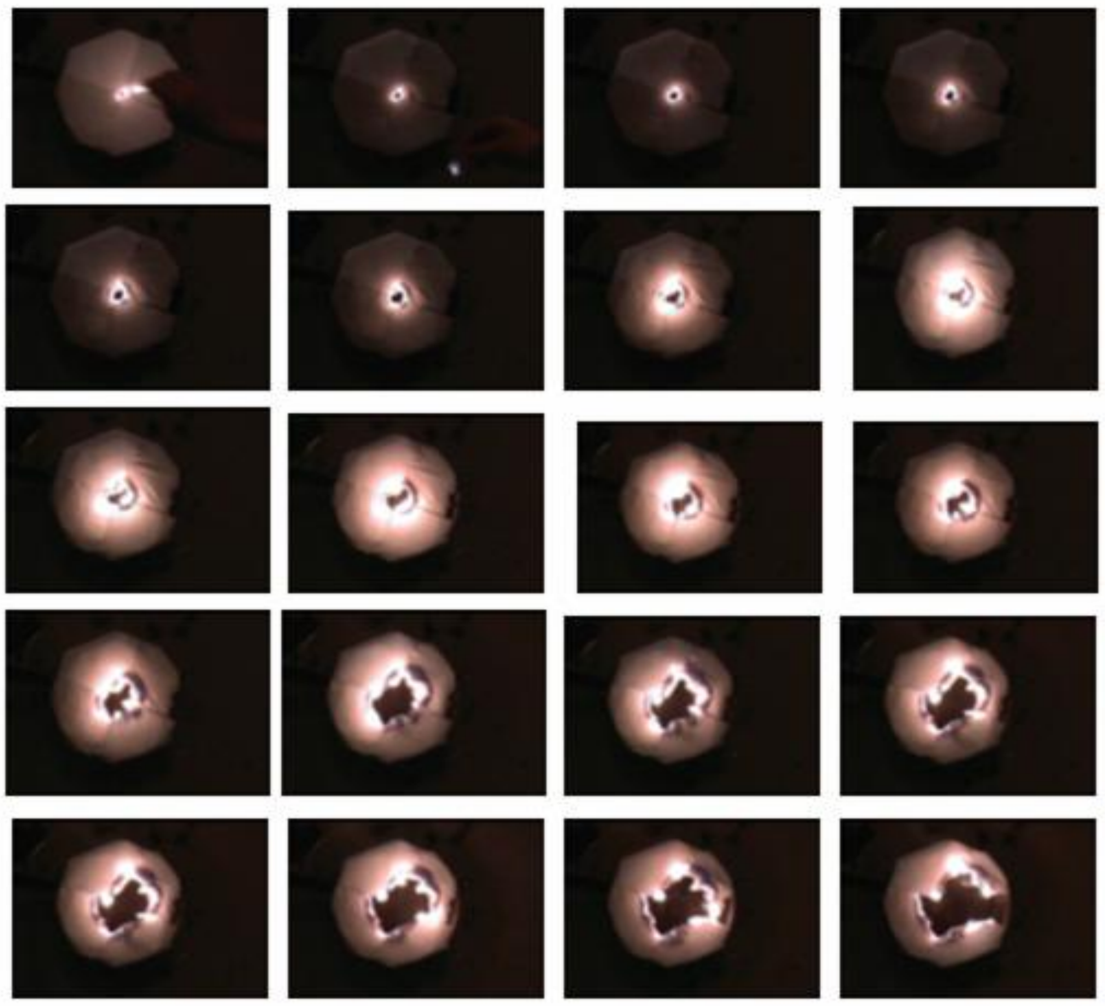
Appendix B: Tables of Validation Comparison

Analysis of Input Data

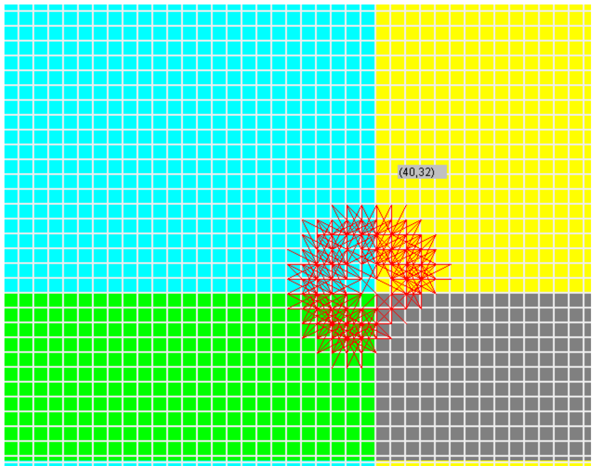
Variable	N	Mean (cm/s)	StDev	Median
Brawny Paper Towels	54	0.7350	0.2096	0.7805
NS Computer Paper	42	0.4320	0.1749	0.4320
Notebook Paper	28	0.8192	0.2921	0.7965
Tissue Paper	20	0.797	0.473	0.7380

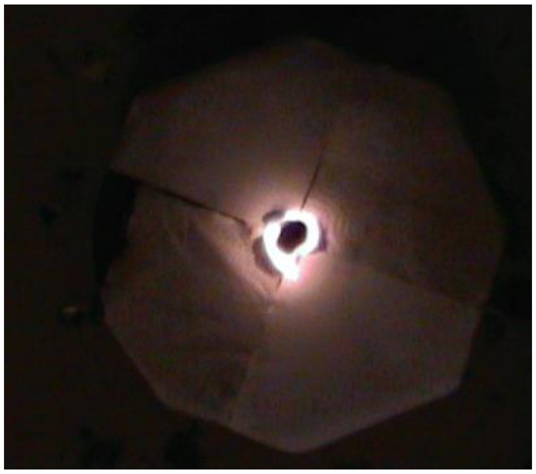
(Temperature about 20 degrees C, humidity at about 18 percent no wind)

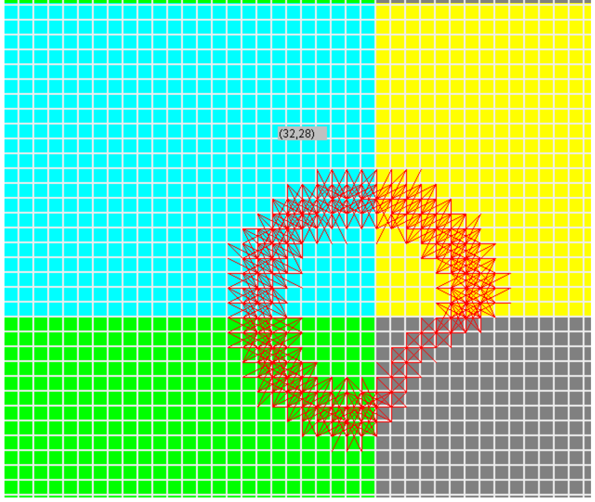
The data summary above shows the found spread rate distribution of the four tested materials under the given conditions. Two trials were conducted for each material. The following picture set illustrates the experiment fire test. The pictures were taken at half second intervals.

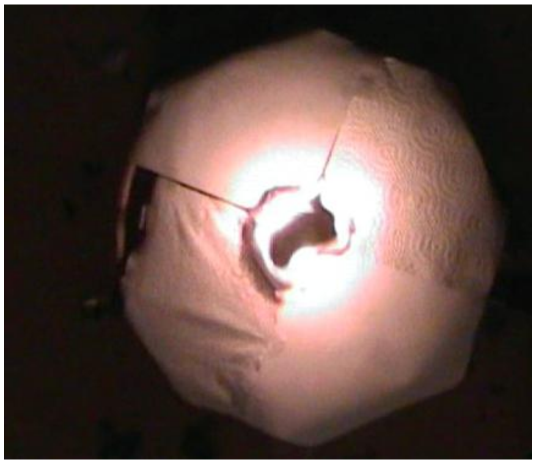


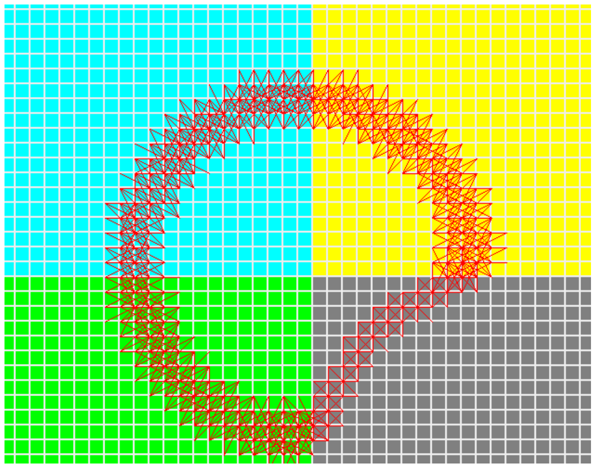
The experiment was conducted using the four tested in a quadrant configuration

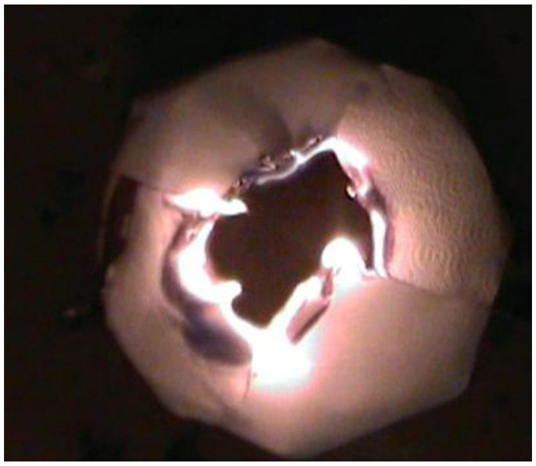












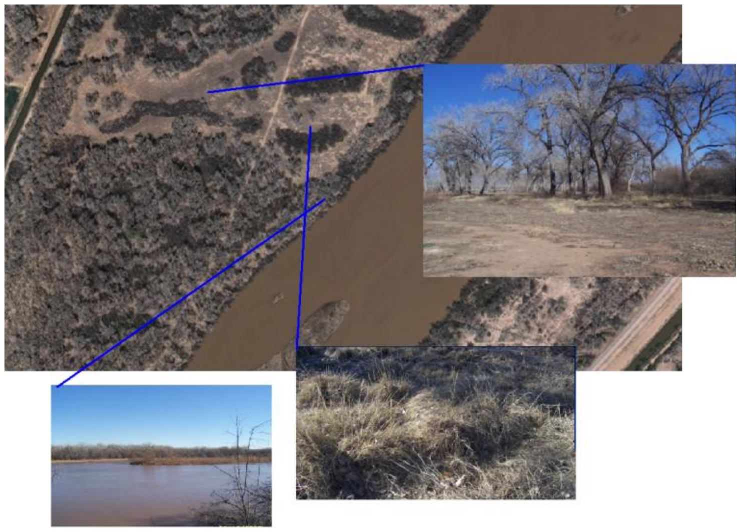
Assessment

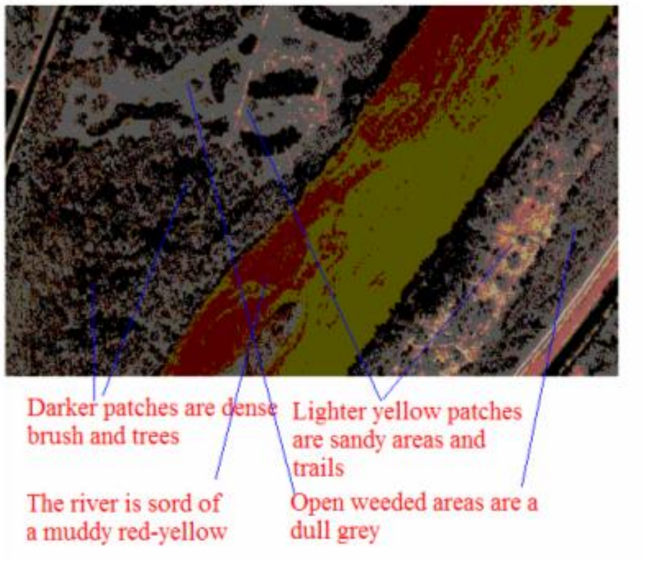
These visual comparison is without a doubt remarkably similar. This empirical test supports *Phoenix* to a high degree. Perhaps the best comparison is the visual comparison. The i*nterpolation comparison* tool will complement this qualitative assessment in the near future.

Appendix C: Satellite Field Testing

Bosque Field Testing

A field test was conducted on an area south of the Montaño Bridge in Albuquerque, New Mexico. The area tested was 500 by 300 meters and covered a span over the river. There were five specific fuel types found and documented in the Bosque. The picture to the right depicts pictures taken from the indicated spots in the Bosque.




Darker patches are dense brush and trees
Lighter yellow patches are sandy areas and trails
The river is sord of a muddy red-yellow
Open weeded areas are a dull grey

- Cottonwood: Cottonwoods are large typically less flammable trees. They appear in many of the darker regions of the photograph.
- The River: The River is obvious in its path straight through the middle of the photograph.
- Sandy Open Areas and Trails: Sandy open areas are characterized by lighter colored areas and contain sand and light concentrations of weeds.
- Brush and Grassy Areas: Bush grassy areas are depicted by the moderate grayish color. They contain large concentrations of weeds ground duff and brush. Some of the material is alive and some is dead.
- Russian Olive: Russian Olive concentrations are hard to distinguish from Cottonwoods and they often overlap.

Rationale

Once the fuel types are known from a *field test* they can be given an already researched spread rate or perhaps the spread rate could later be derived. The spread rates are essentially the same for all fuels of the same type and consistency. Eventually a fully developed fire map could be easily created and realistic fire flow could presumable be modeled quickly and easily.

Appendix D: Program Code

The Code

Analytical Fire modeling Phase II
Expanding upon the Elliptical Fire Theory

Patent in Progress

Table of Code Contents

```
// Background.java
// Created on November 7, 2007, 7:49 PM
// @author Christopher Morrison
//Draws the pane behind the internal frame window
package phoenix;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.image.*;

public class BackGround extends JDesktopPane //Area behind the Viewer
{
    Image PhoenixImg=Toolkit.getDefaultToolkit().getImage("Phoenix.jpg");

    public BackGround()
    {
        this.setBackground(Color.BLUE);
    }
     public void paintComponent(Graphics G) //draws the image behind the Internal Frame
    {
      super.paintComponent(G);
      G.drawImage(PhoenixImg,25,25,this.getWidth()-50,this.getHeight()-50,this);
    }
}
```
(20)

```
// ColorNum.java
// Created on March 7, 2007, 7:49 PM
// @author Christopher Morrison

package phoenix;
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.awt.Event.*;

public class ColorNum //Records the number of instances of a certain RGB
{                    //in the photograph and the color of them
   int Number;
   Color C;

   public ColorNum(Color c)
   {
       Number=1;
       C=c;
    }
    public int getNumber()
    {
       return Number;
    }
    public Color getColor()
    {
       return C;
    }
    public void another()
    {
       Number++;
    }
}
(33)
```

```
// CustomEnvironment.java
// Created on February 17, 2007, 5:49 PM
// @author Christopher Morrison
// Designed to coordinate satellite pictures with creating
// a new Environment
package phoenix;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.image.*;
import java.util.*;
import javax.swing.border.*;
import javax.swing.table.*;

public class CustomEnvironment extends JFrame implements WindowListener, ActionListener,Runnable
{
   Dimension D=new Dimension(600,650);
   JPanel JPA=new JPanel(new GridLayout(2,1));
   JPanel JP1=new JPanel(new GridLayout(1,2));
   JPanel JP2=new JPanel();
   ScanStats SS;
   ImageArea IA=new ImageArea();          //shows intrepreted data image
   SecondaryPane SP;//=new SecondaryPane();
   Table ProcessingTable;                 //table to select fuel type of color
   Image IM;
   FTAdd FTA=new FTAdd();                 //Add fueltypes for the table
   JComboBox JCB;
   ArrayList FuelTypes=new ArrayList();
   Thread thread;


   public CustomEnvironment(Image I,ScanStats S) //Pass Image and its scan data
   {
      super("Environment Analysis");
      this.setSize(D);
      SS=S;
      IM=I;
      this.initFuelTypes();
      JCB=Box();
      ProcessingTable=new Table(SS.GetColorInfo(),JCB,FuelTypes);
      SP=new SecondaryPane();

      this.add(JPA);
      JPA.add(JP1);
      JP1.add(IA);
      JP1.add(SP);
      JPA.add(ProcessingTable);

      FTA.setVisible(false);

      SP.AddFuelType.addActionListener(this);
      FTA.Pane.OK.addActionListener(this);
      ProcessingTable.Publish.addActionListener(this);

   }
   public JComboBox Box()
```

```
{
   JComboBox Box=new JComboBox();
   for(int x=0;x<FuelTypes.size();x++)
   {
      Box.addItem((FuelType)FuelTypes.get(x));
   }
   return Box;
}
public void initFuelTypes()         //initialize the basic fuel types
{
   FuelTypes.add(FuelType.BasicConiferous);
   FuelTypes.add(FuelType.BasicDecidious);
   FuelTypes.add(FuelType.BasicGrass);
   FuelTypes.add(FuelType.BasicGround);
   FuelTypes.add(FuelType.BasicShrub);
}
public void addFuelType(FuelType FT) //Add fuel type to use in
{                                   // processing table
   SP.addRow(FT,true);
   JCB.addItem(FT.getName());
   ProcessingTable.addFuelType(JCB);
}
public void Publish()
{

}
public class ImageArea extends JPanel
{
   public ImageArea()
   {
      this.setBackground(Color.BLUE);
      this.setBorder(new LineBorder(Color.CYAN));
      this.setSize(250,250);
   }
   public void paintComponent(Graphics G)
   {
      super.paintComponent(G);
      SS.redrawPic(G,this.getWidth(),this.getHeight());
   }
}
public class SecondaryPane extends JPanel
{
   Vector Rows=new Vector();
   Vector Columns=new Vector();
   JButton AddFuelType=new JButton("Add Fuel Type");
   TableEditor TEdit;
   JComboBox jcb;
   JTable JT;
   JScrollPane Pane;


   public SecondaryPane()  //holds data of fueltypes in a Table form
   {
      this.setSize(250,250);
      this.setBackground(Color.BLUE);
      this.setBorder(new LineBorder(Color.CYAN));
```

```
        TEdit=new TableEditor(Rows,Columns);
        JT=new JTable(TEdit);
        JT.setPreferredScrollableViewportSize(new Dimension(250, 200));
        Pane=new JScrollPane(JT);

        for(int x=0;x<FuelTypes.size();x++)
        {
            this.addRow((FuelType)FuelTypes.get(x),false);
        }
        this.add(Pane);
        this.add(AddFuelType);
    }
    public void addRow(FuelType FT,boolean firstTime)
    {
        if(firstTime)
        {
            FuelTypes.add(FT);
        }
        Vector V=new Vector();
        V.add(Rows.size()+1);
        V.add(FT.getName());
        V.add(FT.getSpreadRate());
        V.add(FT.getFlashPoint());
        Rows.add(V);
        JT.addNotify();
    }
    public class TableEditor extends DefaultTableModel
    {
        public final String[] Titles={"Num","FuelType","SpreadRate","FlashPoint"};
        public Vector Rows;
        public Vector Columns;

        public TableEditor(Vector rows,Vector column)
        {
            super(rows,column);
            Rows=rows;
            Columns=column;
            this.setColumnIdentifiers(Titles);
        }
    }
}

public void windowActivated(WindowEvent E)
{

}
public void windowDeactivated(WindowEvent E)
{

}
public void windowOpened(WindowEvent E)
{

}
public void windowClosed(WindowEvent E)
{
```

```
            this.dispose();
    }
    public void windowClosing(WindowEvent E)
    {

    }
    public void windowIconified(WindowEvent E)
    {

    }
    public void windowDeiconified(WindowEvent E)
    {

    }
    public void actionPerformed(ActionEvent E)
    {
        if(E.getSource()==SP.AddFuelType)
        {
            FTA.setVisible(true);

        }
        if(E.getSource()==FTA.Pane.OK)
        {
            if(FTA.check())
            {
                this.addFuelType(FTA.getFuelType());
                ProcessingTable.FuelTypes.add(FTA.getFuelType());
                FTA.setVisible(false);
            }
        }
        if(E.getSource()==ProcessingTable.Publish)
        {
            thread=new Thread(this);
            thread.start();
        }
    }
    public void  run()  //run this thread when publishing the new environment
    {
          if(JOptionPane.showConfirmDialog(null,"Are you sure you want to change
enviroments?")==JOptionPane.OK_OPTION)
          {
              try
              {
                  FuelType[] FT=ProcessingTable.Publish();
                  FuelType[][] Env=SS.Publish(FT);
                  PatchClass NewEnv=Data.universalPC;//new PatchClass(Env);//Memory Leak
                  Data.universalPC=NewEnv;
                  Data.V=new Views(NewEnv);
                  Data.EnvChanged=true;
                  this.dispose();
              }
              catch(Exception E)
              {
                  JOptionPane.showMessageDialog(null,E.toString());
              }
           }
    }
}
```

```
// CustomEnvironmentLoader.java
// Created on February 25, 2007, 3:57 PM
// @author Christopher Morrison
//Goals- sliders more interactive
// Allows for data and satellite scanning before user Environment Makes

package phoenix;
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;

public class CustomEnvironmentLoader extends JFrame
{
   int prec=1;
   int areaPrec=1;
   double envh=1;
   double envw=1;

   Dimension D=new Dimension(330,270);
   CELPane CELP=new CELPane();

   public CustomEnvironmentLoader() //essentially a big data Field
   {
      super("Env Specifications");
      this.setSize(D);
      this.setPreferredSize(D);
      this.setBackground(Color.BLUE);
      this.add(CELP);
   }
   public class CELPane extends JPanel
   {
      JButton Cancel=new JButton("Cancel");
      JButton OK=new JButton("OK");
      CELPCenter CELPC=new CELPCenter();
      JPanel Bottom=new JPanel();
      JProgressBar JPG=new JProgressBar(0,100);

      public CELPane()
      {
            super(new BorderLayout());
            this.add(CELPC,BorderLayout.CENTER);
            this.add(Bottom,BorderLayout.SOUTH);
            Bottom.setBackground(Color.BLUE);
            Bottom.add(JPG);
            Bottom.add(Cancel);
            Bottom.add(OK);
      }


      public class CELPCenter extends JPanel implements ActionListener
      {
         JLabel Precision=new JLabel("Color Precision       ");
         JLabel ENVH=new JLabel     ("Environment Height (m)");
         JLabel ENVW=new JLabel     ("Environment Width (m) ");
JLabel Swath=new JLabel    ("Area Precision        ");

JLabel SwathN=new JLabel(Integer.toString(areaPrec));
JLabel PrecisionN=new JLabel(Integer.toString(prec));
JLabel ENVHN=new JLabel(Double.toString(envh));
JLabel ENVWN=new JLabel(Double.toString(envw));

JSlider JS=new JSlider(1,255);

JTextField HeightField=new JTextField(Double.toString(envh),5);
JTextField WidthField=new JTextField(Double.toString(envw),5);
JTextField SwathPrec=new JTextField(Integer.toString(areaPrec),5);
JTextField ColorPrec=new JTextField(Integer.toString(prec),5);

JButton JB1=new JButton("Set");
JButton JB2=new JButton("Set");
JButton JB3=new JButton("Set");
JButton JB4=new JButton("Set");

JPanel Prec=new JPanel(new GridLayout(2,1));
JPanel HWPanel=new JPanel(new GridLayout(2,1));
JPanel JP1=new JPanel(new FlowLayout(FlowLayout.LEFT));
JPanel JP2=new JPanel(new FlowLayout(FlowLayout.LEFT));
JPanel JP3=new JPanel(new FlowLayout(FlowLayout.LEFT));
JPanel JP4=new JPanel(new FlowLayout(FlowLayout.LEFT));

public CELPCenter()
{
   super(new GridLayout(2,1,5,5));
   this.setBackground(Color.BLUE);

   JP1.setBackground(Color.CYAN);
   JP2.setBackground(Color.CYAN);
   JP3.setBackground(Color.CYAN);
   JP4.setBackground(Color.CYAN);

   Prec.setBorder(new TitledBorder(null,"Precision"));
   HWPanel.setBorder(new TitledBorder(null,"Height Width Specs"));

   JB1.addActionListener(this);
   JB2.addActionListener(this);
   JB3.addActionListener(this);
   JB4.addActionListener(this);
   ColorPrec.addActionListener(this);
   SwathPrec.addActionListener(this);
   HeightField.addActionListener(this);
   WidthField.addActionListener(this);
   Cancel.addActionListener(this);

   this.add(Prec);
   this.add(HWPanel);

   Prec.add(JP1);
   Prec.add(JP2);
   HWPanel.add(JP3);
   HWPanel.add(JP4);
```

```
    JP1.add(Precision);
    JP1.add(ColorPrec);
    JP1.add(JB1);
    JP1.add(PrecisionN);
    JP2.add(Swath);
    JP2.add(SwathPrec);
    JP2.add(JB2);
    JP2.add(SwathN);
    JP3.add(ENVH);
    JP3.add(HeightField);
    JP3.add(JB3);
    JP3.add(ENVHN);
    JP4.add(ENVW);
    JP4.add(WidthField);
    JP4.add(JB4);
    JP4.add(ENVWN);
}
public void actionPerformed(ActionEvent E)
{
    if(E.getSource()==JB1||E.getSource()==ColorPrec)
    {
        try
        {
            prec=Integer.parseInt(ColorPrec.getText());
            PrecisionN.setText(Integer.toString(prec));
        }
        catch(Exception EE)
        {
            PrecisionN.setText(Integer.toString(prec));
        }
    }
    if(E.getSource()==JB2||E.getSource()==SwathPrec)
    {
         try
         {
            areaPrec=Integer.parseInt(SwathPrec.getText());
            SwathN.setText(Integer.toString(areaPrec));
         }
         catch(Exception EE)
         {
            SwathN.setText(Integer.toString(areaPrec));
         }
    }
    if(E.getSource()==JB3||E.getSource()==HeightField)
    {
        try
        {
            envh=Double.parseDouble(HeightField.getText());
            ENVHN.setText(Double.toString(envh));
        }
        catch(Exception EE)
        {
            ENVHN.setText(Double.toString(envh));
        }
    }
```

```
                if(E.getSource()==JB4||E.getSource()==WidthField)
                {
                    try
                    {
                        envw=Double.parseDouble(WidthField.getText());
                        ENVWN.setText(Double.toString(envw));
                    }
                    catch(Exception EE)
                    {
                        ENVWN.setText(Double.toString(envw));
                    }
                }
                if(E.getSource()==OK)
                {
                    //action handled in Environment Editor Class
                }
                if(E.getSource()==Cancel)
                {
                    this.setVisible(false);
                }
            }
        }
    }
     public boolean check()
     {
        try
        {
            prec=Integer.parseInt(CELP.CELPC.ColorPrec.getText());
            CELP.CELPC.PrecisionN.setText(Integer.toString(prec));
            envw=Double.parseDouble(CELP.CELPC.WidthField.getText());
            CELP.CELPC.ENVWN.setText(Double.toString(envw));
            envh=Double.parseDouble(CELP.CELPC.HeightField.getText());
            CELP.CELPC.ENVHN.setText(Double.toString(envh));
            areaPrec=Integer.parseInt(CELP.CELPC.SwathPrec.getText());
            CELP.CELPC.SwathN.setText(Integer.toString(areaPrec));
            if(envw<=0||envh<=0)
            {
                throw new Exception("Environment Bounds Incorrect");
            }
            if(prec<1||prec>255)
            {
                throw new Exception("Precision Must be between 1 and 255- reccomended 3");
            }
            if(areaPrec<1)
            {
                throw new Exception("Swath Must be greater than or equal to 1");
            }
            return true;
          }
          catch(Exception EE)
          {
             JOptionPane.showMessageDialog(this,"Insufficient data: "+EE.toString());
             return false;
          }
     }
}
```

```
//Data.java
//@ Christopher Morrison
// Holds static Environment Data easy to get to
package phoenix;
import java.awt.*;

public class Data
{
    public static double increment=.5;//=D.getIncrement();           //fire step length
        public static double MSR=6;//Math.sqrt(2);//=D.getMSR();          //interval*fuela length per step
        public static double s=0;//=D.getSeperation();                 //current interval*fuela
        public static int degrees=18;//=D.getDegrees();                 //number of extensions on fire arc
    public static double WindToSpread=1;
    public static Degrees WindAngle=new Degrees(0,Degrees.degrees);
    public static double WindSpeed=0;
    public static PatchClass universalPC=new PatchClass(/*100,100,*/0.0d,0.0d,0.0d);
    public static double FireArcPrecision=360/18;
        public static boolean Wind=false;
    public static Views V=new Views(universalPC);
    public static boolean EnvChanged=false;

    public static FuelType FuelKind(int ft)  //For patch class Constructors
    {
      return (new FuelType(FuelName(ft),SpreadRate(ft),FlashPoint(ft),color(ft)));
    }
    public static Color color(int ft)
        {
                if(ft==0)
                {return Color.cyan;}
                if(ft==1)
                {return Color.green;}
                if(ft==2)
                {return Color.yellow;} //
                if(ft==3)
                {return Color.gray;} //comp paper
                if(ft==4)
                {return Color.blue;}
                if(ft==5)
                {return Color.red;}
                if(ft==6)
                {return Color.orange;}
                if(ft==7)
                {return Color.white;}
                if(ft==8)
                {return Color.magenta;}
                if(ft==9)
                {return Color.black;}
                return Color.RED;
        }
        public static double SpreadRate(int ft)
        {
                if(ft==0)   //notebook paper
                {return 1/.6392;} //.8192
                if(ft==1)   //Tissue Paper
                {return 1/.797;}
                if(ft==2)   //Paper Towl
```

```
			{return 1/.7350;}
			if(ft==3)   //Computer Paper
			{return 1/.4320;}
			if(ft==4)
			{return 3.5;}
			if(ft==5)
			{return 2.5;}
			if(ft==6)
			{return 2;}
			if(ft==7)
			{return 4;}
			if(ft==8)
			{return .75;}
			if(ft==9)
			{return 6;}
			return 1;
	}

public static String FuelName(int ft)
	{
			if(ft==0)
			{return "fuel type 0";}
			if(ft==1)
			{return "fuel type 1";}
			if(ft==2)
			{return "fuel type 2";}
			if(ft==3)
			{return "fuel type 3";}
			if(ft==4)
			{return "fuel type 4";}
			if(ft==5)
			{return "fuel type 5";}
			if(ft==6)
			{return "fuel type 6";}
			if(ft==7)
			{return "fuel type 7";}
			if(ft==8)
			{return "fuel type 8";}
			if(ft==9)
			{return "fuel type 9";}
			return "unknown fuel type";
	}
public static double FlashPoint(int ft)
	{
			if(ft==0)
			{return 100;}
			if(ft==1)
			{return 100;}
			if(ft==2)
			{return 100;}
			if(ft==3)
			{return 100;}
			if(ft==4)
			{return 100;}
			if(ft==5)
			{return 100;}
```

```
                if(ft==6)
                {return 100;}
                if(ft==7)
                {return 100;}
                if(ft==8)
                {return 100;}
                if(ft==9)
                {return 100;}
                return 100;
        }
}
```

```
// Degrees.java
// Created on February 7, 2007, 9:04 PM
// @author Christopher Morrison
//This class encapsulates angle measurements mostly for
// wind and fire propagation

package phoenix;
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.awt.Event.*;

public class Degrees
{
   public static int radians=1; //specifies degrees or radians
   public static int degrees=2; // in the constructor
   private double RADIAN;
   private double DEGREE;

   public Degrees(double size,int form)
   {
      if(form==1)
      {
         DEGREE=180*size/Math.PI;
         RADIAN=size;

      }
      if(form==2)
      {
         DEGREE=size;
         RADIAN=Math.PI*size/180;

      }
   }

   public double getDegree()
   { return DEGREE; }

   public double getRadian()
   { return RADIAN; }

   public void addDegrees(double deg)
   {
      DEGREE+=deg;
      RADIAN+=Math.PI*deg/180;
   }

   public void addRadians(double rad)
   {
      RADIAN+=rad;
      DEGREE+=rad*180/Math.PI;
   }

   public static Degrees CalculateAngle(double X,double Y)
   {
      double Z =(Math.sqrt(Y*Y+X*X));
```

```
        if(Z==0)
        {
           return new Degrees(0,radians);
        }

        if(X>=0&&Y>=0)
        {
           return new Degrees(Math.asin(Y/Z),radians);
        }
        else if(X<=0&&Y>=0)
        {
           return new Degrees(Math.asin(Math.abs(X)/Z)+Math.PI/2,radians);
        }
        else if(X<=0&&Y<=0)
        {
           return new Degrees(Math.asin(Math.abs(Y)/Math.abs(Z))+Math.PI,radians);
        }
        else if(X>=0&&Y<=0)
        {
           return new Degrees(Math.asin(X/Z)+3*Math.PI/2,radians);
        }
        return new Degrees(0,radians);
    }
}
```

```
//Desktop.java
//@ Christopher Morrison
//This class is the origional JFrame where all the GUI is stacked

package phoenix;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.image.*;

public class Desktop extends JFrame implements ActionListener, WindowListener, MouseListener,
MouseMotionListener
{
   JSplitPane TopBottom=new JSplitPane();
   JSplitPane LeftRight=new JSplitPane();
   PatchClass Env=Data.universalPC;                //main Environment
   JInternalFrame JIF=new Internal("View");
   JMenuBar menu=new JMenuBar();
   JMenu File=new JMenu("File");
   JMenu Tools=new JMenu("Tools");
   JMenu Zoom=new JMenu("Zoom");
   JLabel ZoomLabel=new JLabel("1");
   JMenu Help=new JMenu("Help");
   JMenuItem Exit=new JMenuItem("Exit");
   JMenuItem EnvEdit=new JMenuItem("Environment Editor");
   JMenuItem ZoomIn=new JMenuItem("Zoom In");
   JMenuItem ZoomOut=new JMenuItem("Zoom Out");
   JComboBox ZoomBox=new JComboBox();
   OptionPanel OP=new OptionPanel();
   BackGround BG=new BackGround();
   SideBar SB=new SideBar();
   FireFlow FF=new FireFlow(Env);                  //Fire Flow Processes
   Views V=Data.V;                                 //fire view panel
   EnvironmentEditor EE=new EnvironmentEditor(Env);
   Thread thread;

   public Desktop()
   {
      super("Phoenix");
      this.setJMenuBar(getMenu());
      this.setIconImage(Toolkit.getDefaultToolkit().getImage("Phoenix.jpg"));
      this.add(TopBottom);
      this.setSize(Toolkit.getDefaultToolkit().getScreenSize().width-
200,Toolkit.getDefaultToolkit().getScreenSize().height);

      TopBottom.setTopComponent(OP);
      TopBottom.setOrientation(JSplitPane.VERTICAL_SPLIT);
      TopBottom.setBottomComponent(LeftRight);
      TopBottom.setDividerLocation(175);

      LeftRight.setRightComponent(BG);
      LeftRight.setLeftComponent(SB);
      LeftRight.setDividerLocation(150);

      BG.add(JIF);
      JIF.add(V);
   this.Exit.addActionListener(this);
   this.ZoomIn.addActionListener(this);
   this.ZoomOut.addActionListener(this);
   this.addWindowListener(this);
   this.OP.Step.addActionListener(this);
   this.OP.EE.addActionListener(this);
   V.BV.addMouseListener(this);
   V.EV.addMouseListener(this);
   V.FV.addMouseListener(this);
   V.PV.addMouseListener(this);
   V.BV.addMouseMotionListener(this);
   V.EV.addMouseMotionListener(this);
   V.FV.addMouseMotionListener(this);
   V.PV.addMouseMotionListener(this);
   SetUp();

}

public void SetUp() //stats a basic fire
{
    FF.addFire(new Location(40,40,0));

}
public void actionPerformed(ActionEvent E)
{
   if(E.getSource()==Exit)
   {
      System.exit(0);
   }
   if(E.getSource()==ZoomIn)
   {
      V.zoomOut();
      double d=8/V.zoom;          //sets zoom number
      ZoomLabel.setText(Double.toString(d));
   }
   if(E.getSource()==ZoomOut)
   {
      V.zoomIn();
   }
   if(E.getSource()==OP.Step)
   {
      thread=new Thread(FF);
      thread.start();
   }
   if(E.getSource()==OP.EE)
   {
      EE.Activate(Env);
   }
}

public class Internal extends JInternalFrame
{
   public Internal(String Name)//,ImageIcon ii)
   {
      super(Name);
```

```
        setResizable(true);
        setSize(800,750);
        setClosable(false);
        setMaximizable(true);
        setIconifiable(true);
        setVisible(true);
      }
}

public JMenuBar getMenu()
{
    menu.add(File);
    File.add(Exit);
    menu.add(Tools);
    Tools.add(EnvEdit);
    menu.add(Zoom);
    ZoomBox.addItem("Zoom In");
    ZoomBox.addItem("Zoom Out");
    ZoomBox.addItem("Zoom Normal");
    ZoomBox.setEditable(true);
    Zoom.add(ZoomIn);
    Zoom.add(ZoomOut);
    menu.add(ZoomLabel);
    menu.add(Help);
    return menu;
}

public void windowActivated(WindowEvent E)
{

}
public void windowDeactivated(WindowEvent E)
{

}
public void windowClosed(WindowEvent E)
{

}
public void windowClosing(WindowEvent E)
{
        System.exit(0);
}
public void windowIconified(WindowEvent E)
{

}
public void windowDeiconified(WindowEvent E)
{

}
public void windowOpened(WindowEvent E)
{

}
public void mouseExited(MouseEvent E)
```

```
    {
        V.inside=false;
        V.repaint();
    }
    public void mouseEntered(MouseEvent E)
    {
        V.inside=true;
        V.updateLocation(E);
    }
    public void mouseReleased(MouseEvent E)
    {

    }
    public void mousePressed(MouseEvent E)
    {

    }
    public void mouseClicked(MouseEvent E)
    {
        if(E.getButton()==MouseEvent.BUTTON1)  //add fire
        {
            Point P=E.getPoint();
            P.x=(P.x-V.zoom)/V.zoom;
            P.y=(P.y-V.zoom)/V.zoom;
            if(JOptionPane.showConfirmDialog(this,"Add Fire on point
("+P.x+","+P.y+")")==JOptionPane.OK_OPTION)
            {
                FF.addFire(new Location(P.x,P.y,0));
            }
        }
        if(E.getButton()==MouseEvent.BUTTON2)   //check patch stats
        {

        }
    }
    public void mouseMoved(MouseEvent E)
    {
        V.updateLocation(E);
    }

    public void mouseDragged(MouseEvent E)
    {

    }
}
```

```
//Draw.java
//@ Christopher Morrison
// This class draws all of the fire arcs from a starting point

package phoenix;
import java.util.*;
import java.awt.*;
import javax.swing.*;

public class Draw extends ArrayList
{
            double xstart;
            double ystart;

    public Draw(double xs,double ys,ArrayList coordinates)
    {
            xstart=xs;
            ystart=ys;

            if(coordinates.size()!=0)
            {
                        for(int x=0;x<coordinates.size();x++)
                        {
                                    this.add(coordinates.get(x));
                        }
            }
    }
    public Draw(double xs,double ys)
    {
            xstart=xs;
            ystart=ys;
    }

    public void DrawArray(Graphics G,int zoom)
    {
        G.setColor(Color.red);
            if(this.size()!=0)
        {
            for(int x=0;x<this.size();x++)
            {

            G.drawLine((int)((xstart)*zoom+zoom),(int)((ystart)*zoom+zoom),(int)((((Location)this.get(x)).xl
oc)*zoom+zoom),(int)((((Location)this.get(x)).yloc*zoom+zoom)));
             //JOptionPane.showMessageDialog(null,(xstart)*zoom+zoom+" "+(ystart)*zoom+zoom+"
"+(((Location)this.get(x)).xloc)*zoom+zoom+" "+(((Location)this.get(x)).yloc*zoom+zoom));
            }
        }
    }
}
```

```
//DrawList.java
//@ Christopher Morrison
// This class encspsulates all the Draw classes and draws the
// Fire Perimeter
package phoenix;
import java.util.*;
import java.awt.*;
import javax.swing.*;

public class DrawList extends ArrayList
{

   public DrawList()
   {
   }
   public void DrawIt(Graphics G,int zoom)
   {

      if(this.size()!=0)
      {
         for(int x=0;x<this.size();x++)
         {
            ((Draw)this.get(x)).DrawArray(G,zoom);
         }
      }
   }
}
```

```
// DrawPanel.java
// Created on January 31, 2007, 3:40 PM
// @author Christopher Morrison
// Class designed to graph data

package phoenix;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.print.*;
import java.awt.image.*;
import java.io.*;
import java.util.*;

public class DrawPanel extends JPanel implements Printable
{
   Vector Xinfo;                  //Varible info
   Vector Yinfo;
   double XL,XH,YL,YH;            //keep track of data min max
   public int Xdividingfactor=10; // divisions of the  x and y axis
   public int Ydividingfactor=10;
   public boolean rounding=false;
   int gottenX,gottenY;
   int xbase=35,ybase=400;
   int ticksize=4;
   double extraSpacing=1.1f;       //percent distance between last peice of data and graph maximum
   boolean initHL=true;

   public DrawPanel(Vector X,Vector Y,int x,int y)
   {
      gottenX=x;
      gottenY=y;
      Xinfo=X;
      Yinfo=Y;
      initHighLow();  //dynamic allocation based on min and maxes of data
   }

          public void paintComponent(Graphics G)
          {
             super.paintComponent(G);
             drawAxisLabels(G);
             drawInformation(G);
          }

          public void drawAxisLabels(Graphics G)  //draws the asis labels and such
          {
             G.drawLine(0,ybase,435,ybase);  //X axis
             G.drawLine(xbase,0,xbase,435);  //Y axis

             for(int x=0;x<=Xdividingfactor-1;x++)
             {
                 G.setColor(new Color(200,200,255));
                 if(x!=0)
                 {
                     G.drawLine(Math.round(x*400/(Xdividingfactor))+xbase,ybase-
ticksize,Math.round(x*400/(Xdividingfactor))+xbase,0);}
```

```
                    G.setColor(Color.BLUE);
                    G.drawLine(Math.round(x*400/(Xdividingfactor))+xbase,ybase-
ticksize,Math.round(x*400/(Xdividingfactor))+xbase,ybase+ticksize);
                    if(XH-XL>400||rounding)
                    {
                       G.drawString(Long.toString((Math.round((x*(XH-
XL))/(Xdividingfactor)))+Math.round(XL)),Math.round(x*400/(Xdividingfactor))+xbase-27,ybase+30);
                    }
                    else
                    {
                       G.drawString(Float.toString(Float.parseFloat(Double.toString(((x*(XH-
XL))/(Xdividingfactor))+XL))),Math.round(x*400/(Xdividingfactor))+xbase-27,ybase+30);
                    }
              }

            for(int y=0;y<=Ydividingfactor;y++)
              {
               G.setColor(new Color(125,255,255));
               if(y!=0)
               {
                  G.drawLine(400+xbase,ybase-(Math.round(y*400/(Ydividingfactor))),xbase+ticksize,400-
(Math.round(y*400/(Ydividingfactor))));
               }
               G.setColor(new Color(0,150,255));
               G.drawLine(xbase-ticksize,ybase-(Math.round(y*400/(Ydividingfactor))),xbase+ticksize,400-
(Math.round(y*400/(Ydividingfactor))));
               if(YH-YL>400||rounding)
               {
                  G.drawString(Long.toString((Math.round((y*(YH-
YL))/(Ydividingfactor)))+Math.round(YL)),8,ybase-(Math.round(y*400/(Ydividingfactor)-15)));
               }
               else
               {
                  G.drawString(Float.toString(Float.parseFloat(Double.toString(((y*(YH-
YL))/(Ydividingfactor))+YL))),8,ybase-(Math.round(y*400/(Ydividingfactor)-15)));
               }
            }
         }

         public void  drawInformation(Graphics G) //draws the points on the scatter plot
         {
            G.setColor(Color.GREEN);
            if(Yinfo.size()!=Xinfo.size())
            {
                JOptionPane.showMessageDialog(null,"Input Error");
                return;
            }
            for(int x=0;x<Xinfo.size();x++)
            {
               G.drawRect((int)Math.round(((Double.parseDouble(Xinfo.get(x).toString())-XL)*400/((XH-
XL)))-2+35),
                     400-(int)Math.round(((Double.parseDouble(Yinfo.get(x).toString())-YL)*400/((YH-
YL))))-2, 4,4);
            }
         }
```

```
private void initHighLow()  //finds min and max of x and y data
{
   XL=Double.MAX_VALUE;XH=Double.MIN_VALUE;
   YL=Double.MAX_VALUE;YH=Double.MIN_VALUE;
   for(int x=0;x<Xinfo.size();x++)
   {
      if(Double.parseDouble(Xinfo.get(x).toString())>XH)
      {
         XH=extraSpacing*Double.parseDouble(Xinfo.get(x).toString());
      }
      if(Double.parseDouble(Xinfo.get(x).toString())<XL)
      {
         XL=(double)Math.floor(Double.parseDouble(Xinfo.get(x).toString()));
      }
   }
   for(int x=0;x<Yinfo.size();x++)
   {
      if(Double.parseDouble(Yinfo.get(x).toString())>YH)
      {
         YH=extraSpacing*Double.parseDouble(Yinfo.get(x).toString());
      }
      if(Double.parseDouble(Yinfo.get(x).toString())<YL)
      {
         YL=(double)Math.floor(Double.parseDouble(Yinfo.get(x).toString()));
      }
   }

   if(XL>=XH)
   {
      XL=0;
   }
   if(YL>=YH)
   {
      YL=0;
   }
}

public void setVectors(Vector X,Vector Y)//reset x and y data values
{
      Xinfo=X;
      Yinfo=Y;
      initHighLow();
      this.repaint();
}

public int print(Graphics g, PageFormat pf, int pgIndex)//printable implementation to print the
graph
{
   if(pgIndex>0)
   {
      return Printable.NO_SUCH_PAGE;
   }
   g.translate((int)(pf.getImageableX()),(int)(pf.getImageableY()));
   paint(g);
   return Printable.PAGE_EXISTS;
}
```

```
}
```

```
//EnvironmentEditor.java
//Created on January 8, 2007, 6:03 PM
//Christopher Morrison
//Goals - add in progress bar multithreading add fore controls
//Allows user input for wind and other environmental variables
package phoenix;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.print.*;
import java.awt.image.*;
import java.io.*;
import java.util.*;
import javax.swing.border.*;
import java.math.*;
import javax.swing.event.*;
import java.beans.*;

public class EnvironmentEditor extends JFrame implements ActionListener, WindowListener, Runnable
{
   Dimension D=new Dimension(500,550);
   Color ThemeColor=Color.CYAN;
   Image IC=Toolkit.getDefaultToolkit().getImage("Phoenix.jpg");
   Image IM=null;
   EETab Tab=new EETab();
   int height;
   int width;
   PatchClass Environment;
   CustomEnvironmentLoader CE=null;//=new CustomEnvironmentLoader();
   CustomEnvironment CEE;//=new CustomEnvironment();
   ScanStats SS=null;//=new ScanStats();
   Thread thread;
   Thread ProgressThread;

   public EnvironmentEditor(PatchClass PC)
   {
      super("Environment Editor");
      Environment=PC;
      this.setPreferredSize(D);
      this.setSize(D);
      this.setIconImage(IC);
      this.setVisible(false);
      int height=this.getHeight();
      int width=this.getWidth();
      this.add(Tab);

      Tab.EL.Browse.addActionListener(this);
      Tab.EL.Cancel.addActionListener(this);
      Tab.EL.OK.addActionListener(this);
      Tab.E.Exit.addActionListener(this);

   }

   public void Activate(PatchClass PC)
   {
      Environment=PC;
```

```
    this.setVisible(true);
}

public class EETab extends JTabbedPane
{
    Editor E=new Editor();
    EnvLoader EL=new EnvLoader();
    DrawPanel DP=new DrawPanel(new Vector(0),new Vector(0),10,10);

    public EETab()
    {
        this.addTab("Environment Center",E);
        this.addTab("Environment Creater",EL);
        this.addTab("Analysis",DP);

    }
    public class Editor extends JPanel
    {
        JButton Exit=new JButton("Exit");
        JButton Save=new JButton("Save Changes");
        Top T=new Top();
        Bottom B=new Bottom();
        Center C=new Center();


        public Editor()
        {
            super(new BorderLayout(2,2));
            this.add(T,BorderLayout.NORTH);
            this.add(C,BorderLayout.CENTER);
            this.add(B,BorderLayout.SOUTH);
        }
        public void ResetValues()
        {

        }
        public class Center extends JPanel
        {
            JPanel A=new JPanel(new BorderLayout(2,2));
            JPanel B=new JPanel(new BorderLayout(2,2));
            JPanel AA=new JPanel(new GridLayout(2,2));
            JPanel BB=new JPanel(new BorderLayout(20,20));
            JPanel R=new JPanel();
            JPanel R1=new JPanel();
            JPanel W=new JPanel(new BorderLayout());
            JPanel C=new JPanel();
            JPanel WindSpeedSlider=new JPanel();
            JPanel WindSpeedText=new JPanel();
            JPanel Blah=new JPanel();
            JPanel Blah1=new JPanel(new GridLayout(2,1));
            JLabel WindSpeed=new JLabel("Wind Speed: "+Data.WindSpeed+" m/s");
            JLabel WindAngle=new JLabel("Wind Angle: "+Data.WindAngle.getDegree());
            WindSlider WS=new WindSlider(120);
            JTextField JTF=new JTextField(Double.toString(Data.WindSpeed),6);
            JButton SetWindSpeed=new JButton("Set Speed");
            Font Large=new Font("LucidaBrightDemiBold",Font.BOLD,16);
```

```
		TitledBorder TBF=new TitledBorder(null,"Fire
Factors",TitledBorder.LEFT,TitledBorder.TOP,Large);
		TitledBorder TBW=new TitledBorder(null,"Wind
Factors",TitledBorder.LEFT,TitledBorder.TOP,Large);
		TitledBorder Wspeed=new TitledBorder(new LineBorder(Color.BLUE),"Wind Speed");
		TitledBorder Wangle=new TitledBorder(new LineBorder(Color.BLUE),"Wind Angle");
		JSlider JS=new JSlider(0,50,0);
		JCheckBox SliderBox=new JCheckBox("Slider");
		JCheckBox TextBox=new JCheckBox("Manual");
		JCheckBox Wind=new JCheckBox("Wind");

		public Center()
		{

		   super(new GridLayout(2,1,4,4));
		   WS.addMouseListener(new ML());
		   JS.addChangeListener(new CL());
		   JTF.addActionListener(new AL());
		   SetWindSpeed.addActionListener(new AL());
		   TextBox.addActionListener(new AL());
		   SliderBox.addActionListener(new AL());
		   Wind.addActionListener(new AL());

		   JS.setMinorTickSpacing(5);
		   JS.setMajorTickSpacing(10);
		   JS.setPaintLabels(true);

		   TBF.setTitleColor(Color.CYAN);
		   TBW.setTitleColor(Color.CYAN);
		   Wangle.setTitleColor(new Color(45,45,45));
		   Wspeed.setTitleColor(new Color(45,45,45));

		   A.setBorder(TBF);
		   B.setBorder(TBW);
		   C.setBorder(Wspeed);
		   W.setBorder(Wangle);
		   WindSpeedSlider.setBorder(new LineBorder(new Color(45,45,45)));
		   WindSpeedText.setBorder(new LineBorder(new Color(45,45,45)));

		   W.setBackground(Color.CYAN);
		   R.setBackground(Color.CYAN);
		   R1.setBackground(Color.CYAN);
		   A.setBackground(Color.BLUE);
		   B.setBackground(Color.BLUE);
		   AA.setBackground(Color.CYAN);
		   BB.setBackground(Color.CYAN);
		   C.setBackground(Color.CYAN);

		   WindSpeedSlider.setBackground(Color.CYAN);
		   WindSpeedText.setBackground(Color.CYAN);
		   SliderBox.setBackground(Color.CYAN);
		   TextBox.setBackground(Color.CYAN);
		   Blah.setBackground(Color.CYAN);
		   Wind.setBackground(Color.CYAN);

		   this.add(A);
```

```
    this.add(B);
    A.add(AA,BorderLayout.CENTER);
    B.add(BB,BorderLayout.CENTER);
    BB.add(W,BorderLayout.WEST);
    W.add(WS,BorderLayout.CENTER);
    W.add(R,BorderLayout.WEST);
    W.add(R1,BorderLayout.NORTH);
    W.add(R,BorderLayout.EAST);
    W.add(WindAngle,BorderLayout.SOUTH);
    BB.add(C,BorderLayout.CENTER);
    Blah.add(Wind);
    Blah.add(WindSpeed);
    C.add(Blah,BorderLayout.NORTH);
    C.add(Blah1,BorderLayout.CENTER);
    Blah1.add(WindSpeedSlider,BorderLayout.CENTER);
    Blah1.add(WindSpeedText,BorderLayout.SOUTH);

    WindSpeedSlider.add(SliderBox);
    WindSpeedSlider.add(JS);
    WindSpeedText.add(TextBox);
    WindSpeedText.add(SetWindSpeed);
    WindSpeedText.add(JTF);

    Wind.setSelected(false);
    SliderBox.setSelected(true);
    SetWindSpeed.setEnabled(false);
    JTF.setEnabled(false);
    disableSlider();
    disableText();
    SliderBox.setEnabled(false);
    TextBox.setEnabled(false);
}
public void disableSlider()
{
    JS.setEnabled(false);
    SliderBox.setSelected(false);
}
public void enableSlider()
{
    JS.setEnabled(true);
    SliderBox.setSelected(true);
}
 public void disableText()
{
    TextBox.setSelected(false);
    JTF.setEnabled(false);
    SetWindSpeed.setEnabled(false);
}
public void enableText()
{
    JTF.setEnabled(true);
    SetWindSpeed.setEnabled(true);
    TextBox.setSelected(true);
}
public class ML implements MouseListener
{
```

```
                    public void mouseEntered(MouseEvent E) {}
                    public void mouseExited(MouseEvent E) {}
                    public void mouseReleased(MouseEvent E)
                    {
                        BigDecimal Bg=new BigDecimal(WS.CurrentAngle.getDegree());
                        BigDecimal bG=Bg.setScale(1,BigDecimal.ROUND_HALF_UP);

                        if(E.getButton()==MouseEvent.BUTTON1)
                        {
                            if(JOptionPane.showConfirmDialog(C,"Set Wind Angle:
"+bG.doubleValue()+"?")==JOptionPane.OK_OPTION)
                            {
                                WS.WindAngle=WS.CurrentAngle;
                                Data.WindAngle=WS.WindAngle; //program specific
                             }
                        }

                        WindAngle.setText("Wind Angle: "+Double.toString(bG.doubleValue()));
                    }
                    public void mousePressed(MouseEvent E) {}
                    public void mouseClicked(MouseEvent E) {}
                }
                public class CL implements ChangeListener
                {
                    public void stateChanged(ChangeEvent E)
                    {
                       double ws=((JSlider)E.getSource()).getValue();
                       WindSpeed.setText("Wind Speed: "+Double.toString(ws)+" m/s");
                       Data.WindSpeed=ws;
                    }
                }
                public class AL implements ActionListener
                {
                    public void actionPerformed(ActionEvent E)
                    {
                        if(E.getSource()==JTF||E.getSource()==SetWindSpeed)
                        {
                            try
                            {
                                double ws=Double.parseDouble(JTF.getText());
                                Data.WindSpeed=ws;
                                WindSpeed.setText("Wind Speed: "+Double.toString(ws)+" m/s");
                            }
                            catch(Exception e)
                            {
                                JOptionPane.showMessageDialog(C,"Input Error"+e.toString());
                                JTF.setText(Double.toString(Data.WindSpeed));
                            }
                        }
                        if(E.getSource()==TextBox)
                        {
                            if(TextBox.isSelected())
                            {
                                disableSlider();
                                enableText();
                            }
                else
                {
                    enableSlider();
                    disableText();
                }
            }
            if(E.getSource()==SliderBox)
            {
                if(SliderBox.isSelected())
                {
                    enableSlider();
                    disableText();
                    Data.Wind=true;
                }
                else
                {
                    disableSlider();
                    enableText();
                    Data.Wind=false;
                }
            }
            if(E.getSource()==Wind)
            {
                if(!Wind.isSelected())
                {
                    disableSlider();
                    disableText();
                    SliderBox.setEnabled(false);
                    TextBox.setEnabled(false);
                }
                else
                {
                    enableSlider();
                    SliderBox.setEnabled(true);
                    TextBox.setEnabled(true);
                }
            }
        }
    }
}
public class Top extends JPanel
{
    public Top()
    {

    }
}
public class Bottom extends JPanel
{
    public Bottom()
    {
        super(new FlowLayout(FlowLayout.RIGHT,5,5));
        this.add(Save);
        this.add(Exit);
    }
}
```

```
        }
        public class EnvLoader extends JPanel
        {
           JButton Browse=new JButton("Browse");
           JButton OK=new JButton("Scan");
           JButton Cancel=new JButton("Cancel");
           Top T=new Top();
           Picture P=new Picture();
           Bottom B=new Bottom();
           JScrollPane JSP=new JScrollPane(P);

           public EnvLoader()
           {
               super(new BorderLayout(2,2));
               this.add(T,BorderLayout.NORTH);
               this.add(JSP,BorderLayout.CENTER);
               this.add(B,BorderLayout.SOUTH);
           }
           public class Top extends JPanel
           {
              public Top()
              {
                 this.add(Browse);
              }
           }
           public class Bottom extends JPanel
           {
              public Bottom()
              {
                 super(new FlowLayout(FlowLayout.RIGHT,5,5));
                 this.add(OK);
                 this.add(Cancel);
              }
           }
           public class Picture extends JPanel
           {
              public Picture()
              {
                 this.setPreferredSize(new Dimension(this.getHeight(),this.getWidth()));
                 this.setBackground(ThemeColor);
              }
              public void paintComponent(Graphics G)
              {
                 super.paintComponent(G);
                 if(IM!=null)
                 {

G.drawImage(IM,(int)(1.0/16.0*this.getWidth()),(int)(1.0/16.0*this.getHeight()),(int)(14.0/16.0*this.getWi
dth()),(int)(14.0/16.0*this.getHeight()),this);
                 }
              }
           }
        }
        public class Analysis extends JPanel
        {
```

```
        public Analysis()
        {
           super(new GridLayout());
        }
     }
  }

public void actionPerformed(ActionEvent E)
{
   if(E.getSource()==Tab.EL.Browse)
   {
      JFileChooser JFC= new JFileChooser(".");
      JFC.setFileFilter(new Input(1));
      JFC.setAccessory(new ImageFileView(JFC));
      JFC.setVisible(true);
      int retval = JFC.showOpenDialog(null);
      if (retval == JFileChooser.APPROVE_OPTION)
      {
         File myFile = JFC.getSelectedFile();
         try
              {
           IM=Toolkit.getDefaultToolkit().getImage(JFC.getSelectedFile().toString());
           Tab.EL.repaint();
         }
              catch(Exception e)
              {
            JOptionPane.showMessageDialog(null,"Open file failed: "+e.toString());
         }
       }
        }
   if(E.getSource()==Tab.EL.Cancel||E.getSource()==Tab.E.Exit)
   {
      this.setVisible(false);
   }
   if(E.getSource()==Tab.EL.OK)
   {
      CE=new CustomEnvironmentLoader();
      CE.setVisible(true);
      CE.CELP.OK.addActionListener(this);
   }
   if(CE!=null)
   {
      if(E.getSource()==CE.CELP.OK)
      {
         if(CE.check())
         {
            SS=new ScanStats(IM,CE);
            //CE.CELP.JPG=new JProgressBar(0,SS.progress()[0]);
            //Thread Th=new Thread(new ProgressMonitorThread());
            //Th.start();
            thread=new Thread(SS);
            thread.start();
            ProgressThread=new Thread(this);
            ProgressThread.start();
            //CE.setVisible(false);
```

```
                }
            }
        }
    }
    public void windowActivated(WindowEvent E)
    {

    }
    public void windowDeactivated(WindowEvent E)
    {

    }
    public void windowOpened(WindowEvent E)
    {

    }
    public void windowClosed(WindowEvent E)
    {

    }
    public void windowClosing(WindowEvent E)
    {
            this.setVisible(false);
        CE.dispose();//setVisible(false);
        CEE.dispose();//setVisible(false);
    }
    public void windowIconified(WindowEvent E)
    {

    }
    public void windowDeiconified(WindowEvent E)
    {

    }
    public void run()
    {
        CE.CELP.JPG=new JProgressBar(0,SS.progress()[0]);
        do
        {
            try
            {Thread.sleep(100);}

            catch(Exception e)
            {
                JOptionPane.showMessageDialog(null,e.toString());
            }
            CE.CELP.JPG.setValue(SS.progress()[1]);
        }while(CE.CELP.JPG.getValue()<SS.progress()[0]);

    CEE=new CustomEnvironment(IM,SS);
    CE.setVisible(false);
    CEE.setVisible(true);
    this.setVisible(false);
    }
}
```

```
// FTAdd.java
// Created on March 10, 2007, 7:27 PM
// @author Christopher Morrison
//Designed to allow for increased amount fuel types

package phoenix;
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;

public class FTAdd extends JFrame
{
   Dimension D=new Dimension(320,240);
   String name="";
   double spreadrate=1;
   double flashpoint=0;
   Color colors=null;
   FTAPane Pane=new FTAPane();

   public FTAdd()
   {
      super("Add Fuel Type");
      this.setSize(D);
      this.setPreferredSize(D);
      this.setBackground(Color.BLUE);
      this.add(Pane);

   }

   public class FTAPane extends JPanel
   {
      JButton Cancel=new JButton("Cancel");
      JButton OK=new JButton("OK");
      FTACenter FTAC=new FTACenter();
      JPanel Bottom=new JPanel(new FlowLayout(FlowLayout.RIGHT));

      public FTAPane()
      {
            super(new BorderLayout());
            this.add(FTAC,BorderLayout.CENTER);
            this.add(Bottom,BorderLayout.SOUTH);
            Bottom.setBackground(Color.BLUE);
            Bottom.add(Cancel);
            Bottom.add(OK);
      }


      public class FTACenter extends JPanel implements ActionListener
      {
         JLabel FuelName=new JLabel      ("Fuel Name                 ");
         JLabel SpreadRate=new JLabel    ("Spread Rate (cm/s)  ");
         JLabel FlashPoint=new JLabel    ("Flash Point (dg C)     ");
         JLabel Colors=new JLabel        ("Color                          ");
JLabel FuelNameN=new JLabel(name);
JLabel SpreadRateN=new JLabel(Double.toString(spreadrate));
JLabel FlashPointN=new JLabel(Double.toString(flashpoint));

JTextField NameF=new JTextField(name,5);
JTextField SpreadF=new JTextField(Double.toString(spreadrate),5);
JTextField FlashF=new JTextField(Double.toString(flashpoint),5);

JButton JB1=new JButton("Set");
JButton JB2=new JButton("Set");
JButton JB3=new JButton("Set");
JButton JB4=new JButton("Set Default Color");

JPanel Top=new JPanel(new GridLayout(4,1));
JPanel JP1=new JPanel(new FlowLayout(FlowLayout.LEFT));
JPanel JP2=new JPanel(new FlowLayout(FlowLayout.LEFT));
JPanel JP3=new JPanel(new FlowLayout(FlowLayout.LEFT));
JPanel JP4=new JPanel(new FlowLayout(FlowLayout.LEFT));

public FTACenter()
{
   super(new GridLayout(1,1,5,5));
   this.setBackground(Color.BLUE);

   JP1.setBackground(Color.CYAN);
   JP2.setBackground(Color.CYAN);
   JP3.setBackground(Color.CYAN);
   JP4.setBackground(Color.CYAN);

   Top.setBorder(new TitledBorder(null,"Fuel Type Specs"));

   JB1.addActionListener(this);
   JB2.addActionListener(this);
   JB3.addActionListener(this);
   JB4.addActionListener(this);
   NameF.addActionListener(this);
   SpreadF.addActionListener(this);
   FlashF.addActionListener(this);
   Cancel.addActionListener(this);

   this.add(Top);

   Top.add(JP1);
   Top.add(JP2);
   Top.add(JP3);
   Top.add(JP4);

   JP1.add(FuelName);
   JP1.add(NameF);
   JP1.add(JB1);
   JP2.add(SpreadRate);
   JP2.add(SpreadF);
   JP2.add(JB2);
   JP2.add(SpreadRateN);
   JP3.add(FlashPoint);
   JP3.add(FlashF);
   JP3.add(JB3);
   JP3.add(FlashPointN);
   JP4.add(Colors);
   JP4.add(JB4);
}
public void actionPerformed(ActionEvent E)
{
   if(E.getSource()==JB1||E.getSource()==NameF)
   {
      try
      {
         name=FuelName.getText();
         FuelNameN.setText(name);
      }
      catch(Exception EE)
      {
         FuelNameN.setText(name);
      }
   }
   if(E.getSource()==JB2||E.getSource()==SpreadF)
   {
       try
       {
         spreadrate=Double.parseDouble(SpreadF.getText());
         SpreadRateN.setText(Double.toString(spreadrate));
         if(spreadrate<=0)
         {
            throw new Exception("Spread Rate out of bounds");
         }
       }
       catch(Exception EE)
       {
         SpreadRateN.setText(Double.toString(spreadrate));
       }
   }
   if(E.getSource()==JB3||E.getSource()==FlashF)
   {
      try
      {
         flashpoint=Double.parseDouble(FlashF.getText());
         FlashPointN.setText(Double.toString(flashpoint));
      }
      catch(Exception EE)
      {
         FlashPointN.setText(Double.toString(flashpoint));
      }
   }
   if(E.getSource()==JB4)
   {
      try
      {
         Color C=JColorChooser.showDialog(this,"DefaultColor",Color.WHITE);
      }
      catch(Exception EE)
      {
                }
              }
              if(E.getSource()==OK)
              {
                  //action handled in Environment Editor Class
              }
              if(E.getSource()==Cancel)
              {
                  this.setVisible(false);
              }
          }
      }
  }
   public boolean check()
   {
      try
      {
          name=Pane.FTAC.NameF.getText();
          Pane.FTAC.FuelNameN.setText(name);
          spreadrate=Double.parseDouble(Pane.FTAC.SpreadF.getText());
          Pane.FTAC.SpreadRateN.setText(Double.toString(spreadrate));
          flashpoint=Double.parseDouble(Pane.FTAC.FlashF.getText());
          Pane.FTAC.FlashPointN.setText(Double.toString(flashpoint));
          if(spreadrate<=0)
          {
              throw new Exception("Spread Rate out of bounds should be greater than 0");
          }
          if(name.equals(""))
          {
              throw new Exception("Fuel Type must have a name");
          }
          return true;
        }
        catch(Exception EE)
        {
          JOptionPane.showMessageDialog(this,"Insufficient data: "+EE.toString());
          return false;
        }
   }
   public FuelType getFuelType()
   {
       return new FuelType(name,spreadrate,flashpoint,colors);
   }
}
```

```
//Fire Flow.java
//Christopher Morrison
//This class encapsulates the whole fire flow process

package phoenix;
import javax.swing.*;
import java.util.*;

public class FireFlow implements Runnable
{
   PatchClass environment;
   FireList      FL=new FireList();
   DrawList      DL=new DrawList();
   PerimeterList PL=new PerimeterList();

   int width;
   int height;

   public FireFlow(PatchClass PC)
   {
      environment=PC;
      width=environment.getWidth();
      height=environment.getHeight();
   }

   public synchronized void run()
   {
                        DL.clear();
                        FL.clear();

      if(PL.size()!=0)
           {
            for(int x=0;x<PL.size();x++)
            {
               FireClass tmp=new FireClass(((Location)PL.get(x)).xloc,((Location)PL.get(x)).yloc);
               FL.add(tmp);
            }
           }
           PL.clear();
           if(FL.size()!=0)
           {
         for(int x=0;x<=FL.size()-1;x++)
         {
            FireClass A=(FireClass)FL.get(x);
                        A.FireFlow();
         }
           }
      Data.V.Update(FireDataPackage());
   }

   public FirePackage FireDataPackage()
   {
      return new FirePackage(FL,DL,PL);
   }

   class FireClass extends Thread
   {
            public double increment=Data.increment;           //fire step length
                        public double MSR=Data.MSR;                       //interval*fuela length per step
                        public double s=Data.s;                   //current interval*fuela
                        public int degrees=Data.degrees;                 //number of extensions on fire arc
                        public final double pi=Math.PI;
                        public double xstart;
                        public double ystart;
                        public double zstart;

                        public FireClass(double x,double y)
                        {
                                  xstart=x;ystart=y;
                        }
                        public void run()
                        {
                                  FireFlow();
                        }
                        public void FireFlow()
                        {
               Draw coordinates=new Draw(xstart,ystart);

               Degrees WindDeg=new Degrees(0,Degrees.degrees);

for(/*WindDeg=0*/;WindDeg.getDegree()<=360;WindDeg.addDegrees(Data.FireArcPrecision))
               {
                        boolean done=false;
                                  double spread=0;
                                  s=0;

                      do
                      {
                         double xa=xstart+s*Math.cos(WindDeg.getRadian());
                         double ya=ystart+s*Math.sin(WindDeg.getRadian());
                         if(xa<0){xa=0;}
                         if(ya<0){ya=0;}
                         if(xa>width){xa=width;}
                         if(ya>height){ya=height;}

                         int a=(int)xa;
                         int b=(int)ya;
                         spread+=increment*environment.getPatch(a,b).Type.getSpreadRate();
                         s+=increment;

                         if(spread>=MSR)
                        {
                           if(Data.Wind)
                           {
                              xa=WindVectoring("x",xa,ya,Data.WindAngle);
                              ya=WindVectoring("y",xa,ya,Data.WindAngle);
                           }

                           if(FirePerimeterReduction(xa,ya,WindDeg))
                           {
                              coordinates.add(new Location(xa,ya,0));
                              PL.add(new Location(xa,ya,0));
```

```
                }
            done=true;
            }

        }while(!done);
    }
    DL.add(coordinates);
 }
 public boolean FirePerimeterReduction(double xa,double ya,Degrees degslope)
 {
    double length=Math.sqrt(Math.pow(xa-xstart,2)+Math.pow(ya-ystart,2));
    int aa=0,ba=0,ca=0,da=0;

    for(double x=0; x<=length; x+=.09)
    {
                    aa=(int)(xstart+x*Math.cos(degslope.getRadian()));
                    ba=(int)(ystart+x*Math.sin(degslope.getRadian()));
                    ca=(int)((xstart+x*Math.cos(degslope.getRadian()))*10);
                    da=(int)((ystart+x*Math.sin(degslope.getRadian()))*10);

        if(aa<0){aa=0;}
        if(ba<0){ba=0;}
        if(ca<0){ca=0;}
        if(da<0){da=0;}

        if(!environment.getPatch(aa,ba).PreviousFire(ca%10,da%10)&&x<length-2*Math.sqrt(.02))
            {
            environment.getPatch(aa,ba).Ignite(ca%10,da%10);
        }
    }
    if(!environment.getPatch(aa,ba).PreviousFire(ca%10,da%10))
    {return true;}
    return false;
}

            public double WindVectoring(String component, double xa,double ya,Degrees Angle)
 {
    double length=Math.sqrt(Math.pow(xa-xstart,2)+Math.pow(ya-ystart,2));
    double na;
    Degrees WA=Data.WindAngle;

    if(WA.getDegree()>180)
    {
        if(WA.getDegree()>270)
                    {
            na=270+(WA.getDegree()-270);
        }
                    else
        {
            na=270+(270-WA.getDegree());
        }
    }
    else
    {
            if(WA.getDegree()>90)
        {
```

```
                        na=90-(WA.getDegree()-90);
                      }
                      else
                                  {
                        na=90+(90-WA.getDegree());
                      }
                    }

                    Degrees feta=new Degrees(Math.abs(Angle.getDegree()-na),Degrees.degrees);
                    double newLength=length-Data.WindSpeed*Data.WindToSpread*Math.cos(feta.getRadian());
                    double nxa=xstart+newLength*Math.cos(Angle.getRadian());
                    double nya=ystart+newLength*Math.sin(Angle.getRadian());

                    if(nxa<0){nxa=0;}
                    if(nya<0){nya=0;}
                    if(nxa>width)          {nxa=width;}
                    if(nya>height)         {nya=height;}
                    if(component.equals("x")) {return nxa;}
                    if(component.equals("y")) {return nya;}
                    return 0;
                         }
                }
        public void addFire(Location L)
        {
          PL.add(L);
        }
}
```

```
//FireList.java
//Christopher Morison
//This class encapsulates a list of all the locations of points
//of interest

package phoenix;
import java.util.*;

public class FireList extends ArrayList
{

    public FireList()
    {
    }


}
```

```
//Fire Package.java
//Christopher Morrison
//This class encapsulates all the information about fire points
//arcs and perimeters for the environment
package phoenix;

public class FirePackage
{
            FireList FL;
            DrawList DL;
            PerimeterList PL;

   public FirePackage(FireList fl,DrawList dl, PerimeterList pl)
   {
            FL=fl;
            DL=dl;
            PL=pl;
   }

   public FireList getFireList()
   {
            return FL;
   }

   public DrawList getDrawList()
   {
            return DL;
   }

   public PerimeterList getPerimeter()
   {
            return PL;
   }

}
```

```
// FuelType.java
// Created on March 10, 2007, 10:30 AM
// @author Christopher Morrison
//encapsulates constructor for all fuel types and a few static base fueltypes

package phoenix;
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.awt.Event.*;

public class FuelType
{
   private String Name;        //Name Spread Rate and flashpoint are th only implemented
   private double SpreadRate;  //Fuel Type traits
   //private double DryFuel;
   //private double WetFuel;
   //private double DryFlash;
   //private double WetFlash;
   private double FlashPoint;  //simplified flashpoint for now
   private Color DefaultColor;
   public static FuelType BasicGrass=new FuelType("Grass",.5,100,Color.GREEN);
   public static FuelType BasicGround=new FuelType("Ground",.5,100,Color.ORANGE.darker());
   public static FuelType BasicShrub=new FuelType("Shrub",.5,100,new Color(50,255,25));
   public static FuelType BasicDecidious=new FuelType("Decidious",.5,100,Color.GREEN);
   public static FuelType BasicConiferous=new FuelType("Coniferous",.5,100,Color.GREEN.darker());

   public FuelType(String name,double spreadrate,double flashpoint,Color defaultcolor)
   {
      Name=name;
      SpreadRate=spreadrate;
      FlashPoint=flashpoint;
      DefaultColor=defaultcolor;
   }
   public String toString()
   {
      return Name;
   }
   public String getName()
   {
      return Name;
   }
   public double getSpreadRate()
   {
      return SpreadRate;
   }
   public double getFlashPoint()
   {
      return FlashPoint;
   }
   public Color getDefaultColor()
   {
      return DefaultColor;
   }

}
```

```
// ImageFileView.java
// Created on January 30, 2007, 6:30 PM
// @author Christopher Morrison
//some material borrowed from http://java.sun.com/docs/books
//tutorial/uiswing/components/filechooser.html#filters
//This edits a JFile Chooser to show an image of any image files

package phoenix;
import javax.swing.*;
import java.beans.*;
import java.awt.*;
import java.io.File;

public class ImageFileView extends JComponent implements PropertyChangeListener
{
   ImageIcon thumbnail = null;
   File file = null;

   public ImageFileView(JFileChooser fc)
   {
      setPreferredSize(new Dimension(100, 50));
      fc.addPropertyChangeListener(this);
   }

   public void loadImage()
   {
      if (file == null)
      {
         thumbnail = null;
         return;
      }

      //Don't use createImageIcon (which is a wrapper for getResource)
      //because the image we're trying to load is probably not one
      //of this program's own resources.
      ImageIcon tmpIcon = new ImageIcon(file.getPath());
      if (tmpIcon != null)
      {
         if (tmpIcon.getIconWidth() > 90)
         {
            thumbnail = new ImageIcon(tmpIcon.getImage().getScaledInstance(90, -
1,Image.SCALE_DEFAULT));
         }
         else
         { //no need to miniaturize
            thumbnail = tmpIcon;
         }
      }
   }

   public void propertyChange(PropertyChangeEvent E)
   {
      boolean update = false;
      String prop = E.getPropertyName();

      //If the directory changed, don't show an image.
```

```
        if (JFileChooser.DIRECTORY_CHANGED_PROPERTY.equals(prop))
        {
            file = null;
            update = true;

        //If a file became selected, find out which one.
        }
        else if (JFileChooser.SELECTED_FILE_CHANGED_PROPERTY.equals(prop))
        {
            file = (File) E.getNewValue();
            update = true;
        }

        //Update the preview accordingly.
        if (update)
        {
            thumbnail = null;
            if (isShowing())
            {
                loadImage();
                repaint();
            }
        }
    }

    protected void paintComponent(Graphics g)
    {
        if (thumbnail == null)
        {
            loadImage();
        }
        if (thumbnail != null)
        {
            int x = getWidth()/2 - thumbnail.getIconWidth()/2;
            int y = getHeight()/2 - thumbnail.getIconHeight()/2;

            if (y < 0)
            {y = 0;}
            if (x < 5)
            {x = 5;}
            thumbnail.paintIcon(this, g, x, y);
        }
    }
}
```

```
//Location.java
//Christopher Morrison
//A basic class that holds a location x,y,z

package phoenix;

public class Location
{
		double xloc;
		double yloc;
		double zloc;

   public Location(double x,double y,double z)
   {
		xloc=x;
		yloc=y;
		zloc=z;
   }
}
```

```
//Main.java
//ChristopherMorrison
//The driver of the whole Program
//resets itself undercertain conditions
package phoenix;
import javax.swing.*;

public class Main implements Runnable
{
   Thread Reset;
   Desktop D;
   /** Creates a new instance of Main */
   public Main()
   {
      D=new Desktop();
      D.setVisible(true);
      Reset=new Thread(this);
      Reset.start();
   }
   public void run()
   {
      do
      {
         try
         {Thread.sleep(100);}

         catch(Exception e)
         {
            JOptionPane.showMessageDialog(null,e.toString());
         }
         if(Data.EnvChanged)
         {
            D.dispose();
            //D.setVisible(false);
            D=new Desktop();
            D.setVisible(true);
            Data.EnvChanged=false;
         }
      }while(true);
   }
   public static void main(String[] args)
   {
      Main M=new Main();
   }

}
```

```
//OptionPanel.java
//Christopher Morrison
//Top Panel Holds ToolBar and other data
package phoenix;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.border.*;

public class OptionPanel extends JPanel
{
   ImageIcon EEI=new ImageIcon("EnvironmentEditor.png");
   ImageIcon StepI=new ImageIcon("Step.png");
   JButton EE=new JButton(EEI);
   JButton Step=new JButton(StepI);
   JLabel Zoom=new JLabel("Zoom");
   JTextField ZoomLevel=new JTextField("8",5);
   JToolBar JTB=new JToolBar("Tool Bar");

   public OptionPanel()
   {
      this.setLayout(new FlowLayout(FlowLayout.LEFT));
      this.add(JTB);
      JTB.add(EE);
      JTB.add(Step);
      EE.setToolTipText("Environment Editor");

      this.setBackground(Color.CYAN);

   }
   public void paintComponent(Graphics G)
   {
      super.paintComponent(G);
   }
}
```

```
//PatchClass.java
//Christopher Morrison
//Holds all data about the virtual forest/environment
package phoenix;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;


public class PatchClass
{
            ArrayList Ground=new ArrayList();  //Array List Holding all patch information
      Random rand=new Random();
            Data c=new Data();
            int width;
            int height;
      public double WindSpeed;
      public double AmbientTemp;
      public double WindAngle;
      int size=5;

      public PatchClass(double Ambient,double WindA,double WindS) //4 kind enviroment used in
empirical testing
            {
         for(int x=0; x<80;x++)
              {
             Ground.add(new ArrayList());
             for(int y=0; y<80;y++)
                      {
                                if(x>=40)
                   {
                      if(y>=40)
                      {
                         ((ArrayList)Ground.get(x)).add(new patches(Data.FuelKind(3)));
                      }
                      else
                      {
                         ((ArrayList)Ground.get(x)).add(new patches(Data.FuelKind(2)));
                      }
                   }
                   else
                   {
                      if(y>=40)
                      {
                         ((ArrayList)Ground.get(x)).add(new patches(Data.FuelKind(1)));
                      }
                      else
                      {
                         ((ArrayList)Ground.get(x)).add(new patches(Data.FuelKind(0)));
                      }
                   }
                      }
         }
```

```
            AmbientTemp=Ambient;
            WindAngle=WindA;
            WindSpeed=WindS;
            height=Ground.size();
            width=((ArrayList)Ground.get(0)).size();
              }

        public PatchClass(int XSize,int YSize,double Ambient,double WindA,double WindS) //Random
shuffle Environment Constructor
              {
            int max=rand.nextInt(9);
                for(int x=0; x<XSize;x++)
                 {
               Ground.add(new ArrayList());
               for(int y=0; y<YSize;y++)
                        {
                                ((ArrayList)Ground.get(x)).add(new
patches(Data.FuelKind(rand.nextInt(max+1))));
                        }
            }
            AmbientTemp=Ambient;
            WindAngle=WindA;
            WindSpeed=WindS;
            height=Ground.size();
            width=((ArrayList)Ground.get(0)).size();
              }

        public PatchClass(double Ambient,double WindA,double WindS,int XSize,int YSize) //random Box
Environment
        {
            int max=rand.nextInt(9);

            for(int x=0; x<XSize;x++)
            {
               Ground.add(new ArrayList());
               for(int y=0; y<YSize;y++)
               {
                  ((ArrayList)Ground.get(x)).add(new patches(Data.FuelKind(1)));
               }
            }
            for(int a=14; a>=1;a--)
            {
               int yl=rand.nextInt(height-3*a);
               int xl=rand.nextInt(width-3*a);
               int type=rand.nextInt(max+1);

               for(int x=0;x<=a*3;x++)//(int)Math.floor(Math.pow(3/4,a)*width);  x++)
               {
                  for(int y=0;y<=a*3;y++)//(int)Math.floor(Math.pow(3/4,a)*height);  y++)
                  {
                     ((ArrayList)Ground.get(x+xl)).set(y+yl,new patches(Data.FuelKind(type)));
                  }
               }
            }

            AmbientTemp=Ambient;
        WindAngle=WindA;
        WindSpeed=WindS;
        height=Ground.size();
        width=((ArrayList)Ground.get(0)).size();
}
public PatchClass(FuelType[][] CustomEnv)  //Satellite Photo Env Constructor
{
   width=CustomEnv.length;
   height=CustomEnv[0].length;
   JOptionPane.showMessageDialog(null,height+" "+width);
   for(int y=0;y<width;y++)
   {
       ArrayList AL=new ArrayList();
       for(int x=0;x<height;x++)
       {
            AL.add(new patches(FuelType.BasicConiferous));//CustomEnv[y][x]));
       }
       Ground.add(AL);
   }
}


public class patches
{
    double temperatureC;        //inividual patch traits
          FuelType Type;
          public boolean[][] active=new boolean[10][10];

          public patches(FuelType FT)
          {
        Type=FT;
        temperatureC=AmbientTemp;
        for(int a=0;a<=9;a++)
               {
            for(int b=0; b<=9;b++)
                        { active[a][b]=false;}
               }
          }

          public boolean PreviousFire(int a,int b)
          {
        if(size==5)
        {
            return active[(int)Math.floor(a/2)][(int)Math.floor(b/2)];
        }
        return active[a][b];
    }

    public void Ignite(int a, int b)
          {
       if(size==5)
        {
           active[(int)Math.floor(a/2)][(int)Math.floor(b/2)]=true;
           return;
        }
       active[a][b]=true;
```

```
        }
      }

    public patches getPatch(int x,int y)
    {
       return ((patches)((ArrayList)Ground.get(x)).get(y));
    }

    public void DrawEnv(Graphics g,int zoom,boolean spacing)  //draws pactches
    {
        int spacer=zoom;
        if(spacing&&zoom>1)
        {spacer=Math.round(7*zoom/8);}

        for(int x=0;x<width;x++)
             {
           for(int y=0;y<height;y++)
                {
              g.setColor(getPatch(x,y).Type.getDefaultColor());
                   g.fillRect(x*zoom+zoom,y*zoom+zoom,spacer,spacer);
                }
              }
     }

     public void drawBurnt(Graphics g,int zoom,boolean spacing) //draws burnt patches (black) and unburnt
(green)
     {
        int spacer=zoom;
        if(spacing)
        {
           spacer=Math.round(7*zoom/8);}

           for(int x=0;x<=width-1;x++)
           {
              for(int y=0;y<=height-1;y++)
              {
                 g.setColor(Color.green);
                 for(int xa=0;xa<=9;xa++)
                 {
                    for(int ya=0;ya<=9;ya++)
                    {
                       if(getPatch(x,y).active[xa][ya])
                       {g.setColor(Color.black);  }
                    }
                 }g.fillRect(x*zoom+zoom,y*zoom+zoom,spacer,spacer);
              }
           }
        }

        public void EditStats() //soon to be recording device
        {

        }
        public int getHeight()
        {return width;}
        public int getWidth()
        {return height;}
}
```

```
//PerimeterList.java
//Christopher Morrison
//Holds all fire points not eliminated from fire perimeter Reduction
package phoenix;
import java.util.*;

public class PerimeterList extends ArrayList
{

   public PerimeterList()
   {
   }
}
```

```
// ScanStats.java
// Created on February 24, 2007, 12:07 PM
// @author Christopher Morrison
// Goals -Add env sizing, get full pixel range
//This class takes the satellite Photograph and turns it into statistical data
//for processing
package phoenix;
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.awt.Event.*;
import java.awt.image.PixelGrabber;

public class ScanStats implements Runnable
{
   int precision=3;                    //Within 10 color units the guestimation will be save CPU time
   int Precision=30;
   int area;                           //generalized pixel area
   ArrayList Colors=new ArrayList();   //Array of color Information
   Image Pic;
   ArrayList ColorArray=new ArrayList();
   int height;
   int width;
   CustomEnvironmentLoader CEL;
   double EnvWidth;
   double EnvHeight;
   int total;
   boolean working=false;


   public ScanStats(Image I,CustomEnvironmentLoader CE)
   {
      working=true;
      Pic=I;
      ColorArray.clear();
      Colors.clear();
      EnvHeight=CE.envh;
      EnvWidth=CE.envw;
      height=Pic.getHeight(null);
      width=Pic.getWidth(null);
      total=height*width;

   }
   public void run()
   {
      total=0;
      int[] pixels=new int[height*width];
      PixelGrabber pg = new PixelGrabber(Pic,0,0,width,height,pixels,0,width);
      try
      {
         pg.grabPixels();
      }
      catch (Exception E)
      {
         JOptionPane.showMessageDialog(null,"Error in fitlering pixels"+E.toString());
         return;
```

```
    }

    for(int y=0;y<height;y++)
    {
        ArrayList Row=new ArrayList();
        for(int x=0;x<width;x++)
        {
            total++;
            Color C=getColor(x,y,pixels[width*y+x]);
            Row.add(C);
             CompareColor(C);
        }
        ColorArray.add(Row);
    }
    working=false;
    JOptionPane.showMessageDialog(null,""+getSize());
}
public Vector GetColorInfo()
{
    Vector V=new Vector();
    ArrayList AL=Colors;

    while(AL.size()>0)
    {
        int base=0;
        int remove=0;
        for(int x=0;x<AL.size();x++)
        {
            if(((ColorNum)AL.get(x)).getNumber()>base)
            {
                base=((ColorNum)AL.get(x)).getNumber();
                remove=x;

            }
        }
        //JOptionPane.showMessageDialog(null,(((ColorNum)AL.get(remove)).getNumber())+" ");
        V.addElement((ColorNum)AL.get(remove));
        AL.remove(remove);
    }

    return V;

}

public ArrayList SwathArea()
{
    ArrayList AL=new ArrayList();

    int NumX=(int)((ArrayList)ColorArray.get(0)).size()/Precision;
    int NumY=(int)ColorArray.size()/Precision;
    int RemainX=(int)((ArrayList)ColorArray.get(0)).size()%Precision;
    int RemainY=(int)ColorArray.size()%Precision;
    int Prec2=Precision*Precision;

    for(int y=0;y<NumY;y++)
    {
            ArrayList al=new ArrayList();

            for(int x=0;x<NumX;x++)
            {
                int red=0;
                int green=0;
                int blue=0;

                for(int yy=0;yy<Precision;yy++)
                {
                    for(int xx=0;xx<Precision;xx++)
                    {
                        Color C=(Color)((ArrayList)ColorArray.get(y*Precision+yy)).get(x*Precision+xx);
                        red+=C.getRed();
                        green+=C.getGreen();
                        blue+=C.getBlue();
                    }
                }
                Color color=new
Color(Math.round(red/(Prec2)),Math.round(red/(Prec2)),Math.round(red/(Prec2)));
                al.add(color);
            }
            AL.add(al);
        }
        return AL;
    }

    public Color getColor(int x, int y, int pixel)
    {
        int alpha = (pixel >> 24) & 0xff;
        int red   = (pixel >> 16) & 0xff;
        int green = (pixel >>  8) & 0xff;
        int blue  = (pixel      ) & 0xff;
        try
        {
            red=Precision(red);
            green=Precision(green);
            blue=Precision(blue);
            if(red>255){red=255;}
            if(green>255){green=255;}
            if(blue>255){blue=255;}
            return new Color(red,green,blue);
        }

        catch(Exception E)
       {
          JOptionPane.showMessageDialog(null,red+" "+green+" "+blue+" "+E.toString());
          return new Color(255,212,222);
       }
    }
    public int[] progress()
    {
        int[] i=new int[2];
        i[0]=height*width;
        i[1]=total;
        return i;
```

```
}
public int Precision(int prec)
{
  return (int)Math.floor(Math.floor((prec/(255/precision)))*255/precision);
}

public void redrawPic(Graphics G,int width,int height)
{
   if(!working)
   {
      int xSize=(int)width/((ArrayList)ColorArray.get(1)).size();
      int ySize=(int)height/ColorArray.size();
      int realsize;

      if(xSize>0&&ySize>0)
      {
         if(xSize<=ySize)
         {
            realsize=xSize;
         }
         else
         {
            realsize=ySize;
         }

         for(int y=0;y<ColorArray.size();y++)
         {
            for(int x=0;x<((ArrayList)ColorArray.get(y)).size();x++)
            {
               G.setColor((Color)((ArrayList)ColorArray.get(y)).get(x));
               G.fillRect(x*realsize,y*realsize,realsize,realsize);
            }
         }
      }
      else
      {
         double xa=((ArrayList)ColorArray.get(0)).size();
         double ya=ColorArray.size();
         double Inx=1.0d/(width/xa);
         double Iny=1.0d/(height/ya);
         int InxSize=(int)Math.ceil(Inx);
         int InySize=(int)Math.ceil(Iny);
         int Inrealsize;

         if(InxSize<=InySize)
         {
            Inrealsize=InxSize;
         }
         else
         {
            Inrealsize=InySize;
         }

         for(int y=0;y<ColorArray.size();y+=Inrealsize)
         {
            for(int x=0;x<((ArrayList)ColorArray.get(y)).size();x+=Inrealsize)
```

```
                {
                    G.setColor((Color)((ArrayList)ColorArray.get(y)).get(x));
                    G.fillRect((int)x/Inrealsize,(int)y/Inrealsize,1,1);
                }
            }
        }
    }
}
public void CompareColor(Color C)
{
    if(Colors.size()!=0)
    {
        for(int x=0;x<Colors.size();x++)
        {
            if(C.equals(((ColorNum)Colors.get(x)).getColor()))
            {
                ((ColorNum)Colors.get(x)).another();
                return;
            }
        }
    }
    Colors.add(new ColorNum(C));
}
public int getSize()
{
    return Colors.size();
}
public FuelType[][] Publish(FuelType[] Breakdown)
{
    FuelType[][] Env=new FuelType[width][height];
    for(int y=0;y<height;y++)
    {
        for(int x=0;x<width;x++)
        {
            Env[x][y]=sort(Breakdown,(Color)((ArrayList)ColorArray.get(y)).get(x));
        }
    }
    return Env;
}
private FuelType sort(FuelType[] Breakdown,Color C)
{
    for(int x=0;x<Breakdown.length;x++)
    {
        if(Breakdown[x].getDefaultColor().equals(C))
        {
            return Breakdown[x];
        }
    }
    return null;
}
}
```

```
//SideBar.java
//Christopher morrison
//This class holds the side panel and statistical information
package phoenix;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;


public class SideBar extends JTabbedPane
{

   RunControls RC=new RunControls();
   JSplitPane JSP=new JSplitPane();

   public SideBar()
   {
      this.addTab("Controls",RC);
   }

   public class RunControls extends JPanel
   {
      int PointsOfInterest;
      //int

      public RunControls()
      {
      }
      public void paintComponent(Graphics G)
      {
         super.paintComponent(G);
      }
   }
}
```

```
// Statistics.java
// Created on January 30, 2007, 8:39 PM
// @author Christopher Morrison
//This class will soon hold a statistical record of fire and its path

package phoenix;
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.awt.Event.*;

public class Statistics
{

    public Statistics()
    {

    }
}
```

```
//Table.java
//Christopher Morrison
//This class holds the JTable that is used to determine user input
//for the satellite scanning rgb
package phoenix;
import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.awt.image.*;
import java.awt.event.*;
import java.awt.print.*;
import javax.swing.table.*;
import java.awt.event.*;
import javax.swing.event.*;


public class Table extends JPanel implements TableModelListener
{
   Vector Rows=new Vector();
   Vector Columns=new Vector();
   JButton Publish=new JButton("Publish Environment");
   TableEditor TEdit;
   JComboBox jcb;
   JTable JT;
   JScrollPane Pane;
   Vector colors;
   ArrayList FuelTypes;

   public Table(Vector V, JComboBox JCB,ArrayList FT) //creates JPanel and sets correct alignment and
numerical format
   {
      FuelTypes=FT;
      TEdit=new TableEditor(Rows,Columns);
      JT=new JTable(TEdit);
      jcb=JCB;
      JCB.setSelectedIndex(0);
      JT.setPreferredScrollableViewportSize(new Dimension(500, 250));
      JT.setSize(500,150);
      Pane = new JScrollPane(JT);
      this.setBackground(Color.CYAN);
      colors=V;
      for(int x=0;x<V.size();x++)
      {
         this.addRow((ColorNum)V.get(x));
      }
      TableColumn colorColumn = JT.getColumn("Color");
      colorColumn.setCellRenderer(new DTCR());
      TableColumn comboColumn = JT.getColumn("FuelType");
      comboColumn.setCellEditor(new DCE(JCB));
      init();
      this.add(Pane);
      this.add(Publish);

   }
   public void addFuelType(JComboBox JCB)
   {
        TableColumn comboColumn = JT.getColumn("FuelType");
        comboColumn.setCellEditor(new DCE(JCB));
    }
    public void addRow(ColorNum CN)
    {
            Vector C=new Vector();

            C.add(TEdit.getRowCount()+1);
            C.add(CN.getColor());
            C.add(CN.getNumber());
            C.add("");
            Rows.add(/*index*/C);
            JT.addNotify();
    }
    public FuelType[] Publish()
    {
        int length=TEdit.getRowCount();
        FuelType[] FT=new FuelType[length];
        for(int x=0;x<length;x++)
        {
          FuelType X=Search(TEdit.getValueAt(x,3).toString());
          FT[x]=new
FuelType(X.getName(),X.getSpreadRate(),X.getFlashPoint(),(Color)TEdit.getValueAt(x,1));
        }
        return FT;
    }
    public FuelType Search(Object FTS)
    {
        for(int x=0;x<FuelTypes.size();x++)
        {
            if(((FuelType)FuelTypes.get(x)).toString().equals(FTS))
            {
                return (FuelType)FuelTypes.get(x);
            }
        }
        JOptionPane.showMessageDialog(null,FTS+" Bad Bad Bad");
        return null;
    }
    public void init()
    {
       for(int x=0;x<TEdit.getRowCount();x++)
       {
          TEdit.setValueAt("Ground",x,3);
       }
    }
    public void tableChanged(TableModelEvent source)    //checks user input on table
    {
         TableModel tabMod = (TableModel)source.getSource();
          switch(source.getType())
          {
               case TableModelEvent.UPDATE:
               try
               {
                   //double
d=Double.parseDouble(DataTable.getValueAt(DataTable.getSelectedRow(),DataTable.getSelectedColumn
()).toString());
```

```
            }
            catch(Exception E)
            {
                //DataTable.addNotify();
            }
            break;
        }
    }
public class TableEditor extends DefaultTableModel
{
    public final String[] Titles={"Number","Color","Total","FuelType"};
    public Vector Rows;
    public Vector Columns;

    public TableEditor(Vector rows,Vector column)
    {
        super(rows,column);
        Rows=rows;
        Columns=column;
        this.setColumnIdentifiers(Titles);
    }


    /*public int getColumnCount()
    {
        return Titles.length;
    }

    public int getRowCount()
    {

        return super.getRowCount();
    }

    public String getColumnName(int col)
    {
        return super.getColumnName(col);
    }

    //public Object getValueAt(int row, int col)
    //{
    //    return data[row][col];
    //}*/

    /*
     * JTable uses this method to determine the default renderer/
     * editor for each cell.  If we didn't implement this method,
     * then the last column would contain text ("true"/"false"),
     * rather than a check box.
     */
    public Class getColumnClass(int c)
    {
        return getValueAt(0, c).getClass();
    }

    /*
```

```
     * Don't need to implement this method unless your table's
     * editable.
     */
    public boolean isCellEditable(int row, int col)
    {
        //Note that the data/cell address is constant,
        //no matter where the cell appears onscreen.
        if (col==3)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    /*
     * Don't need to implement this method unless your table's
     * data can change.
     */
    /*public void setValueAt(Object value, int row, int col)
    {
        if (data[0][col] instanceof Integer)
        {
            //If we don't do something like this, the column
            //switches to contain Strings.
            try
            {
                data[row][col] = new Integer((String)value);
                fireTableCellUpdated(row, col);
            }
            catch (NumberFormatException e)
            {

            }
        }
        else
        {
            data[row][col] = value;
            fireTableCellUpdated(row, col);
        }


    }*/

}
public class DTCR extends DefaultTableCellRenderer
{
    public void setValue(Object value)
    {
                if (value instanceof Color)
            {
                        Color c = (Color) value;
                    this.setBackground(c);
                    //setForeground(c.getTextColor());
```

```
                         //setText(c.toString());
                           }
              else if(value instanceof JComboBox)
              {
                  this.setText(((JComboBox)value).getSelectedItem().toString());

              }
              else
              {
                              super.setValue(value);
                          }

          }
      }
      public class DCE extends DefaultCellEditor
      {
          JComboBox JCB;

          public DCE(JComboBox C)
          {
              super(C);
              JCB=C;

          }

      }
  }
```

```
//Views.java
//Christopher morrison
//This displays the different views of the environment on JTabbedPane
package phoenix;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class Views extends JTabbedPane //implements MouseListener
{
   int zoom=8;
   Welcome WV;
   PhysicalView PV;
   FireView FV;
   BurnView BV;
   EnvView EV;
   PatchClass Env;
   JSP WS;
   JSP PS;
   JSP FS;
   JSP BS;
   JSP ES;
   DrawList DL=new DrawList();
   int xlocation=0;
   int ylocation=0;
   boolean inside=false;

   public Views(PatchClass PC)
   {
      Env=PC;
      FV=new FireView();
      BV=new BurnView();
      EV=new EnvView();
      PV=new PhysicalView();
      WV=new Welcome();
      FS=new JSP(FV);
      BS=new JSP(BV);
      ES=new JSP(EV);
      PS=new JSP(PV);
      WS=new JSP(WV);
      this.add("Welcome",WS);
      this.addTab("Physical",PS);
      this.addTab("Fire",FS);
      this.addTab("Burn",BS);
      this.addTab("Environment",ES);
      this.setPreferredSize(new Dimension(800,600));
   }

   public class PhysicalView extends JPanel
   {


      public PhysicalView()
      {
         this.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
```

```
        }
        public void paintComponent(Graphics G)
        {
            super.paintComponent(G);
            Env.DrawEnv(G,zoom,true);
        }

    }

    public class FireView extends JPanel
    {


        public FireView()
        {
            this.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
        }
        public void paintComponent(Graphics G)
        {
            super.paintComponent(G);
            DL.DrawIt(G,zoom);
            if(inside)
            {
                drawLocation(G);
            }
        }

    }

    public class BurnView extends JPanel
    {


        public BurnView()
        {
           this.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
           this.setBackground(Color.CYAN);
        }
        public void paintComponent(Graphics G)
        {
            super.paintComponent(G);
            Env.drawBurnt(G,zoom,true);
            if(inside)
            {
                drawLocation(G);
            }
        }
    }

    public class EnvView extends JPanel
    {

        public EnvView()
        {
            this.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
        }
        public void paintComponent(Graphics G)
        {
            super.paintComponent(G);
            Env.DrawEnv(G,zoom,true);
            DL.DrawIt(G,zoom);
            if(inside)
            {
                drawLocation(G);
            }
        }
    }
    public class Welcome extends JPanel
    {
        public Welcome()
        {

        }
        public void paintComponent(Graphics G)
        {
            super.paintComponent(G);
            G.drawString("Welcome. Press the step button to progress the fire         Phoenix Release version
0.0", 10,10);
        }
    }
    public JPanel makeScroll(JPanel V,JScrollPane JSP)
    {
        JPanel JP=new JPanel(new BorderLayout());
        JP.add(V,BorderLayout.CENTER);
        //JP.add(JSB,BorderLayout.WEST);
        //JP.add(JSB,BorderLayout.NORTH);
        return JP;
    }

    public void Update(FirePackage FP)
    {
        DL=FP.getDrawList();
        this.repaint();
    }

    public void drawLocation(Graphics G)
    {
        G.setColor(Color.LIGHT_GRAY);
        G.fillRect(xlocation+15,ylocation-20,53,15);
        G.setColor(Color.BLACK);
        G.drawString("("+(xlocation-zoom)/zoom+","+(ylocation-zoom)/zoom+")",xlocation+16,ylocation-
8);
    }
    public void updateLocation(MouseEvent E)
    {
        JScrollPane tmp=((JScrollPane)this.getSelectedComponent());
        inside=false;
        this.repaint(xlocation+15-tmp.getHorizontalScrollBar().getValue(),ylocation+5-
tmp.getVerticalScrollBar().getValue(),56,16);
```

```
        inside=true;
        xlocation=E.getX();
        ylocation=E.getY();
        this.repaint(xlocation+15-tmp.getHorizontalScrollBar().getValue(),ylocation+5-
tmp.getVerticalScrollBar().getValue(),56,16);
    }

    public void zoomOut()
    {
        if(zoom>1)
        {
            zoom=Math.round(zoom*2);
            this.repaint();
            this.BV.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
            this.EV.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
            this.FV.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
            this.PV.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));

        }
    }
    public void zoomIn()
    {
        zoom=Math.round(zoom/2);
        this.BV.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
        this.EV.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
        this.FV.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
        this.PV.setPreferredSize(new
Dimension(Env.getWidth()*zoom+2*zoom,Env.getHeight()*zoom+2*zoom));
        this.repaint();
    }

    public static class Border extends JPanel
    {

    }

    public static class JSP extends JScrollPane
    {
        public JSP(Component C)
        {
            super(C);
        }
    }
}
```

```
// WindSlider.java
// Created on February 4, 2007, 2:48 AM
// @author Christopher Morrison
//This class is the component that gives wind angle
package phoenix;
import javax.swing.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.math.*;

public class WindSlider extends JComponent implements MouseListener, MouseMotionListener
{
   Dimension D;
   Color C;
   Degrees UserAngle=new Degrees(0,Degrees.degrees);
   Degrees CurrentAngle=new Degrees(0,Degrees.degrees);
   Degrees WindAngle=Data.WindAngle;
   double WS=Data.WindSpeed;

   public WindSlider()
   {
      this.addMouseListener(this);
      this.addMouseMotionListener(this);
      D=new Dimension(100,100);
      this.setSize(D);
      this.setPreferredSize(D);
   }
   public WindSlider(int size)
   {
      this.addMouseListener(this);
      this.addMouseMotionListener(this);
      D=new Dimension(size,size);
      this.setSize(D);
      this.setPreferredSize(D);
   }
   public void paintComponent(Graphics G)
   {

      super.paintComponent(G);
      Graphics2D G2d = (Graphics2D)G;
      GradientPaint greyish = new GradientPaint(0,0,new Color(120,120,120),100,
0,Color.LIGHT_GRAY);
      G2d.setPaint(greyish);
      G2d.fillRect(0,0,D.height,D.width);
      G2d.setColor(Color.BLACK);
      G2d.drawOval(10,10,D.width-20,D.height-20);
      G2d.drawOval((int)D.width/2-5,(int)D.height/2-5,10,10);
      G2d.drawLine(8,(int)D.height/2,12,(int)D.height/2);
      G2d.drawLine((int)D.width/2,8,(int)D.width/2,12);
      G2d.drawLine(D.width-8,(int)D.height/2,D.width-12,(int)D.height/2);
      G2d.drawLine((int)D.width/2,D.height-8,(int)D.width/2,D.height-12);


      BigDecimal BG=new BigDecimal(CurrentAngle.getDegree());
      BigDecimal bg=BG.setScale(1,BigDecimal.ROUND_HALF_UP);
```

```
        double rounded=bg.doubleValue();
        G2d.drawString(Double.toString(rounded),D.width-35,D.height-5);

        BG=new BigDecimal(WindAngle.getDegree());
        bg=BG.setScale(1,BigDecimal.ROUND_HALF_UP);
        rounded=bg.doubleValue();
        G2d.drawString(Double.toString(rounded),D.width-35,15);



G2d.drawLine(D.width/2,D.height/2,(int)(D.width/2*Math.sin(CurrentAngle.getRadian()+Math.PI/2)+D.w
idth/2),
                (int)(D.width/2*Math.cos(CurrentAngle.getRadian()+Math.PI/2))+D.height/2);

    }

    public Degrees getAngle()
    {
        return WindAngle;
    }
    public void mouseExited(MouseEvent E)
    {
        CurrentAngle=WindAngle;
        this.repaint();
    }
    public void mouseEntered(MouseEvent E)
    {
       double X =(E.getX()-D.width/2);
       double Y =(D.height/2-E.getY());
       CurrentAngle=Degrees.CalculateAngle(X,Y);
       repaint();
    }
    public void mouseReleased(MouseEvent E)
    {
        /*if(E.getButton()==MouseEvent.BUTTON1)  //add fire
        {

             double X =(E.getX()-D.width/2);
             double Y =(D.height/2-E.getY());

            if(JOptionPane.showConfirmDialog(this,"Set Wind Angle:
"+CurrentAngle.getDegree()+"?")==JOptionPane.OK_OPTION)
            {
                WindAngle=Degrees.CalculateAngle(X,Y);
                Data.WindAngle=WindAngle; //program specific

            }
        }*/ //moved into parent class
        repaint();
    }
    public void mousePressed(MouseEvent E)
    {

    }
    public void mouseClicked(MouseEvent E)
    {
    }
    public void mouseMoved(MouseEvent E)
    {
        double X =(E.getX()-D.width/2);
        double Y =(D.height/2-E.getY());
        CurrentAngle=Degrees.CalculateAngle(X,Y);
        repaint();
    }

    public void mouseDragged(MouseEvent E)
    {

    }
}
```