Feeding Hungry Villages

New Mexico

Supercomputing Challenge

Final Report

April 2, 2014

Team 48

Desert Academy

Team Members

Jonas Kaare-Rasmussen

Ben Voter

Alex Kellam

Teachers

Jeff Mathis

Jocelyne Comstock

Mentors

Kim Kaare-Rasmussen

Table of Contents

Executive Summary

Hungry is a global issue. It is found everywhere. There are two major reasons for hunger. The first is not having money to buy resources, while the second is not having resources available. This project revolves around the idea of making resources readily available to everyone.

Our goal is to create a model that will accurately portray a real life scenario depicting hunger. From this model we hope to find conclusive results on the most efficient way to feed 6 impoverished villages from a central distribution center. We will model an ATV and a truck. The pair of vehicles can travel over two different types of roads. We will find the most efficient ratio of roads that trucks can pass against roads that ATVs can pass. Our variables include the amount of passable roads for the truck, time, and money spent on feeding the villages. Our controls include the length of the roads and speed of vehicles. We also have random natural events, like snowstorms. The most efficient ratio of roads is the ratio that yields the longest amount of time the simulation runs with the lowest amount of money spent on the villages.

Our results found the most efficient ratio of roads that a truck can pass to roads a truck cannot pass (but an ATV can) to be 1/6. With this data, we could implement it into a real world cluster of villages. This would increase the amount of food in the selected villages, and create a more sustainable living situation in those villages.

This is a very relevant problem in the entire world, and could be solved simply by implementing a system to distribute food around the world. We not only have the food to distribute it to ever place on earth, but we also have the technology to spread the food everywhere. We can send a package from a place to another place on the other side of the globe using the mail system. We should be

able to use the same system to send food to impoverished villages, nations and continents.

Introduction

Hunger is a major problem in developing countries. In some places this is because of the social hierarchy where the poor simply do not have the resources to nourish themselves. This is defined as poverty. In other places, hunger is a problem, but not because of any social structure. Some people are undernourished simply because they, and the society they are in, cannot get their hands on any food. The resources are simply not there. This is the problem that our project addresses.

A. Objective

Our objective throughout this project is to develop a method to successfully distribute resources to a group of villages surrounding a central distribution center. We plan on looking at efficiency based on the cost of the infrastructure projects (road improvements, etc.) we have implanted into the hypothetical landscape we have created. We hope to find a solution that is indeed viable, and try to implement it into a real life scenario. This model can then be used to find solutions for impoverished groups of villages where away to access such resources is limited.

B. Propose

Our team went for this project because it seemed like a problem that affects people globally, and could easily be solved with the correct use of resources. These resources are implemented into our code in the form of variables.

C. Hypothesis

We hypothesize that we can write a computer code to accurately depict a simple distribution model, and from this model can deduce an applicable solution to the models parameters, which could be used on the global problem.

D. Background

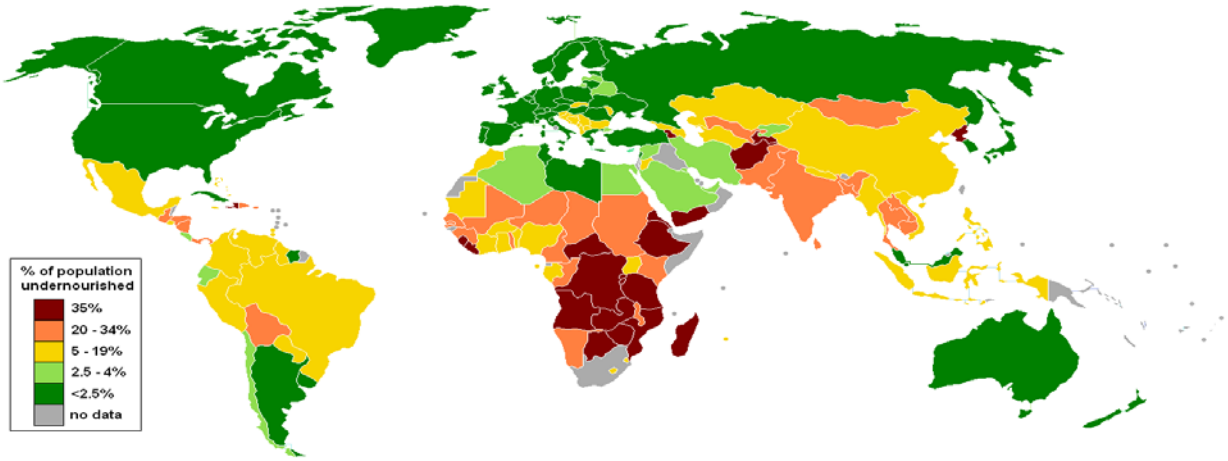

Figure 1: A map showing % of population that is undernourished per country
From (Global Patterns of Food Supply, C onsumption and Trade.)

As seen in Figure 1, the United States, along with the majority of Europe, Australia, and a couple other countries have the lowest poverty rates with less than 2.5% of their populations being undernourished. These areas are usually considered “developed” countries. Countries that we consider “underdeveloped” like Angola and the Central African Republic are countries were undernourishment is very high. Reportedly, 35% of the population in these countries is undernourished by American standards. We, as a global population, produce 1.5 times the food needed to feed every person on the planet. This means that part of

the problem is a distribution problem. In other words, we have the capability to feed everyone on earth but we are not capable of getting the food to the right places. About 40% of all the food in America ends up in a landfill somewhere. According to the NRDC (National Resource Defense Council) this is equivalent to $165 billion dollars worth if food per year. The NRDC continues to state that if only 15% of this waste were to be saved by not cooking massive meals where half of it is thrown out, it would feed 25 million Americans every year. This shows that we are not very efficient with our food production, transportation and usage as a whole. By making distribution more efficient, we would be less wasteful as a nation, and we could divert the extra 40% of waste into something more sustainable, like feeding hungry societies in developing countries. (Gunders 2012)

Description

Our model is written in Processing. Processing is a Java based language created by Ben Fry and Casey Reas. Processing was a good choice for this project because it allows us to make graphic representations much easier than using other languages. One of the major pitfalls is the fact that Processing uses Java's coordinate plane. The origin is not in the center, but rather the upper left corner. The x-coordinates are as the normally would be on the Cartesian plane, but, the y's grow positive in the opposite direction than that of the Cartesian plane. They increase, as the point moves down. When you are used to a Cartesian plane, it is confusing to switch to a Java coordinate plane. We fixed this by translating all the coordinates so that the origin was in the center.

We wanted to make a model that does not simply produce a number; rather it illustrates the process, and then provides a number. Because we could see the process, we could see the validity of the output numbers.

While hunger is a global problem, we narrowed it down to a hypothetical

group of 6 villages. Though never stated, we have always thought that that group of villages was in central Africa, one of the many starving nations. We attempted to construct a model, as seen in Figure 2, which has a defined amount of trucks and ATVs that navigate different roads with different conditions and carry different amounts of supplies. They move between the villages and a central distribution center. The idea is that canned food can be flown into a distribution center where then they can be further distributed to the nearby villages. The model is very controlled, within a chaotic background. It has set parameters, but almost all of them have a random element. For instance, the roads have a random condition, but it has to be one of the three specified. There are other parts of the model that are relatively random, and add many a layers to the project. All of these parts will be discussed in greater detail in the following sections.

A. Model

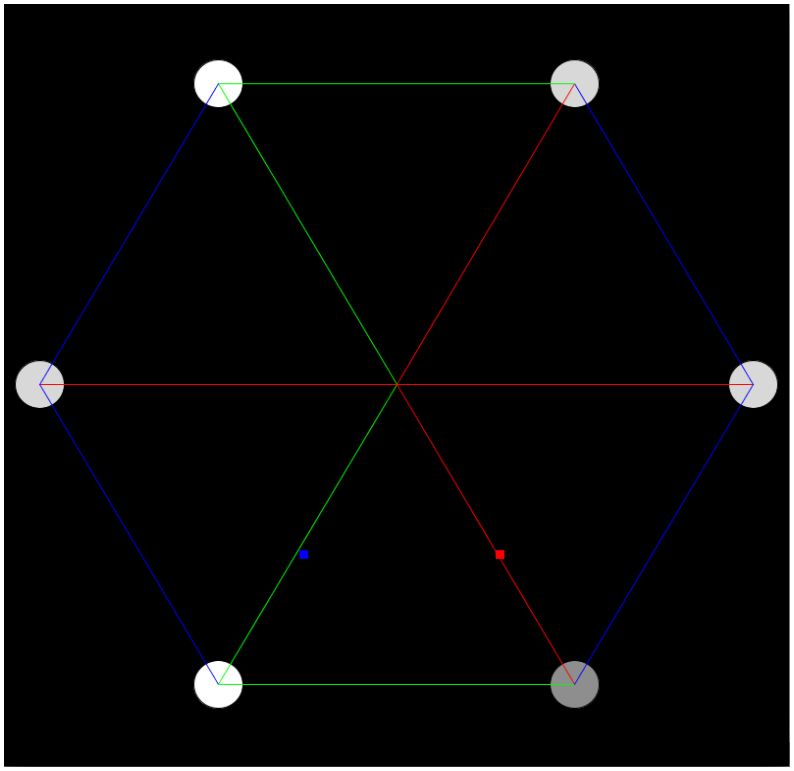

Figure 2: The model

The model is set up with 6 villages, two types of vehicles (an ATV and a truck), and 12 roads that can have 3 different terrain types. The ATV can traverse all three types of roads, whereas the truck can only pass two types of road conditions. In the center is the distribution center. The idea is that the distribution center sends the vehicles filled with food to outlying villages that then consume the food. The ATV then goes back to the center to repeat that process, whereas the truck can go to another village before returning to the center. To keep all the code neat, we used classes. This allows us to write the code once, and have the objects repeat themselves instead of rewriting the same code multiple times. The roads, the villages, the ATV, and the truck are all their own classes.

The roads are the simplest class. They only have one routine, and that is their display routine. The roads are defined by their location relative to the different villages, and the distribution center, as well as a third parameter. This parameter is the color (road condition), a random integer between 0 and 2. With these three options, the roads can be red, blue, or green. Red represents the most treacherous to pass. One could think of it as a trail, rather than a road. This is the road that only the ATV can pass. The Green roads are the most well maintained roads of all. It is thought of as a paved road that is prepared for all types of traffic. The blue road is somewhere in the middle, but at this point is has no effect on the code. It works in the exact same manner as the green road.

Another class is the ATV's class. It runs two subroutines, one of which simply displays it at its position as it moves along the roads. The other routine determines its current position as it moves along the roads. It does so by defining two variables. The ATV's origin, and destination coordinates. The movement is a simple application of the Pythagoreans theorem ($a^2 + b^2 = c^2$.) We define "a" as the difference of the ATV's origin x-coordinate, and the destination x-coordinate.

We define “b” as the difference of the origin’s y-coordinate and the destination’s y-coordinates. In other words, we define “a” and “b” as the legs of a right triangle. The road itself is the hypotenuse (“c”) of the triangle. In our code we simply define this as the distance between the two known points. The ATV then adds (a/c+speed) to its x-coordinate, where speed is a previously defined variable for the pace of the vehicles. Then the ATV proceeds to add (b/c+speed) to its y-coordinate making it move left or right respectively. This happens very fast, so we perceive it to just go in direct line.

The truck class is different from the ATV in one respect. The ATV can only go to a village and back; the ATV can only feed one village per trip. The truck, if the roads are passable, can feed two villages. This is what makes the truck a more powerful tool then the ATV. It can feed more people faster, but it requires better quality roads.

The village class is by far the most complicated class. It has a sub-routine for displaying the village. This routine is relatively simple as all it does is display a circle at the village’s position. The complicated part is finding the correct shade of grey for the village. This is simply done by decreasing a variable in a linear fashion equivalent to the amount of food the village has.

The village also runs a routine for receiving food. The village takes the food from the ATV or truck. Then it redirects the truck or ATV to its next destination. This is simple as a swap of the ATV’s destination and origin variable. The truck is a little more complex. It tests whether it is the trucks first or second destination. If it is the first, the village sends it to the second, if the roads are passable. Otherwise, the village sends the truck to the center. It might seem backwards to have the truck and ATV directed by the village code rather than its own code, but in reality, this way is much simpler because it does not have to refer to all 6 villages.

All of these classes work together with the distribution center to create

model of the hypothetical world. To add a more natural world aspect, we added random elements, like natural disasters. We did this by implementing a time variable, and every 10000 milliseconds we stopped food delivery for a random time less then 100 milliseconds. This helped model a more natural world with interruptions, and give use more precise data.

Results

Through running the simulation many times, we were able to find the most efficient ratio of red, blue and green roads. This was our preliminary experiment. From the data, we were able to make a guess that the best ratio of road colors is in fact 7 red roads, 5 blue roads, and no green roads. A graph of this preliminary data is included below. The points can be found in A. Appendix, Table 1.

Figure 3: The preliminary graph that shows the Time villages had food versus the money spent on the model.

This graph shows all but one of the 100 points of data taken. The point was omitted because the others are 100 times greater than all of the other points. It was so far out that it actually ruined the remaining 99 points distribution. This was a very special point. It was the rare case when there was only village accessible to the truck, so that the truck went back and forth from the center to that village. This meant that the village almost had a constant supply of food; therefore it took a very long time, and cost a lot more than the rest of the points. It was not very efficient.

To find the most efficient point, we need to find the data point that was lowest in cost, but also longest time wise. This point is the 67^{th} test (time spent = $1.28*10^8$, and Price paid = $4.25*10^8$). This point is highlighted in green on the graph.

To further prove this we ran another type of test where the roads were not randomly generated, but rather, a mixture of this ratio, as seen in Figure 4. This was used to provide a more conclusive picture of the previously shown ratio. We did this because the results change depending on where the roads are in relation to each other. For example, if all the blue roads were not accessible to the truck, rendering them completely useless, the efficiency of the model would drop considerably, compared to a model where the truck could drive to the blue roads. From this experiment, we concluded that this setup is the most efficient using the ratio decided upon in the preliminary test. The data did not yield as conclusive results as the first test, because the data was less scattered, and more linear. No one point stuck out more than the other, therefore we decided that the most efficient point was median of the data. The points can be found in A. Appendix, Table 2.

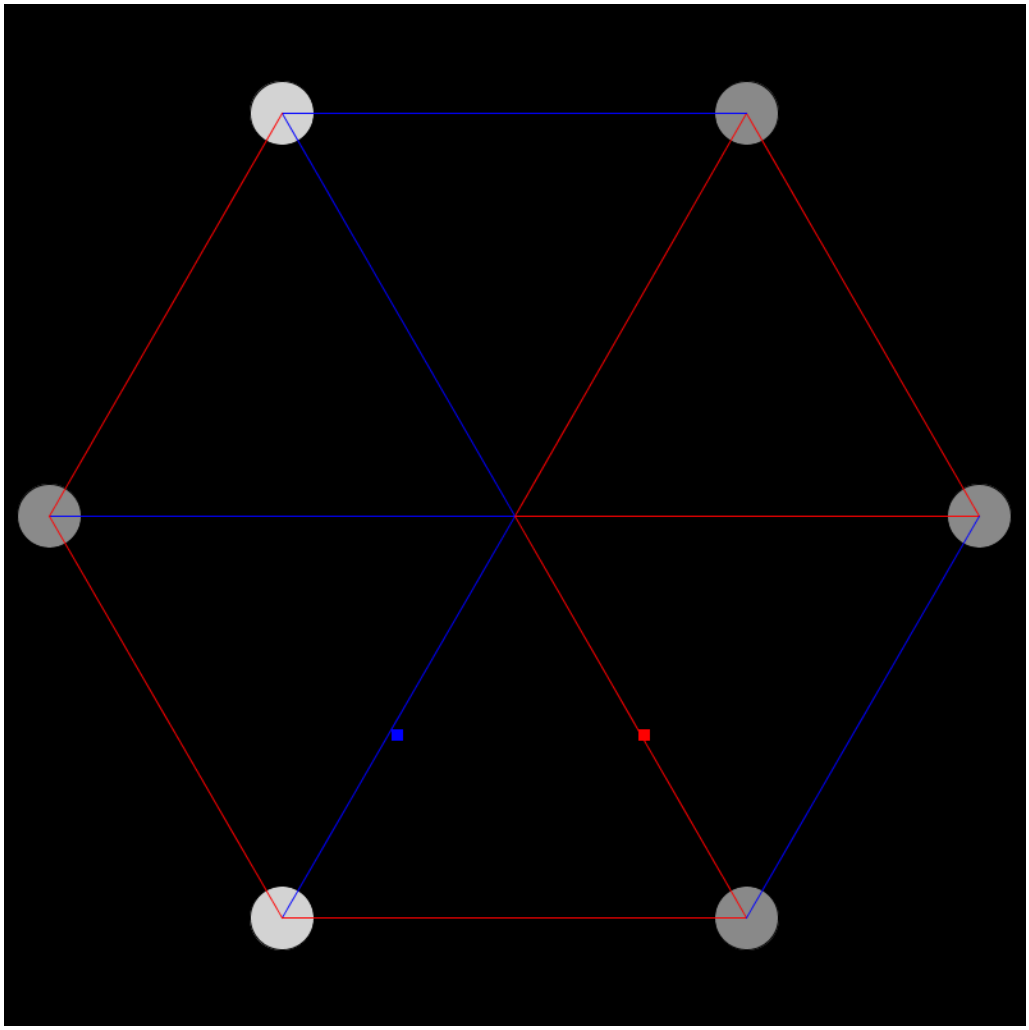

Figure 4: The most efficient model based on the results of the second test.

At this point, we realized that two of the blue roads were completely wasted. The ATV does not need the road to be blue for it to pass it, and the truck cannot reach those two roads. Therefore, we thought we could make it even more effect by making those two roads blue. This was our third and final test, as seen in Figure 5. In this test we eliminated three of the roads and found the most efficient way of keeping at least one village alive is with 2 blue roads, and 10 red roads. The points can be found in A. Appendix, Table 3.

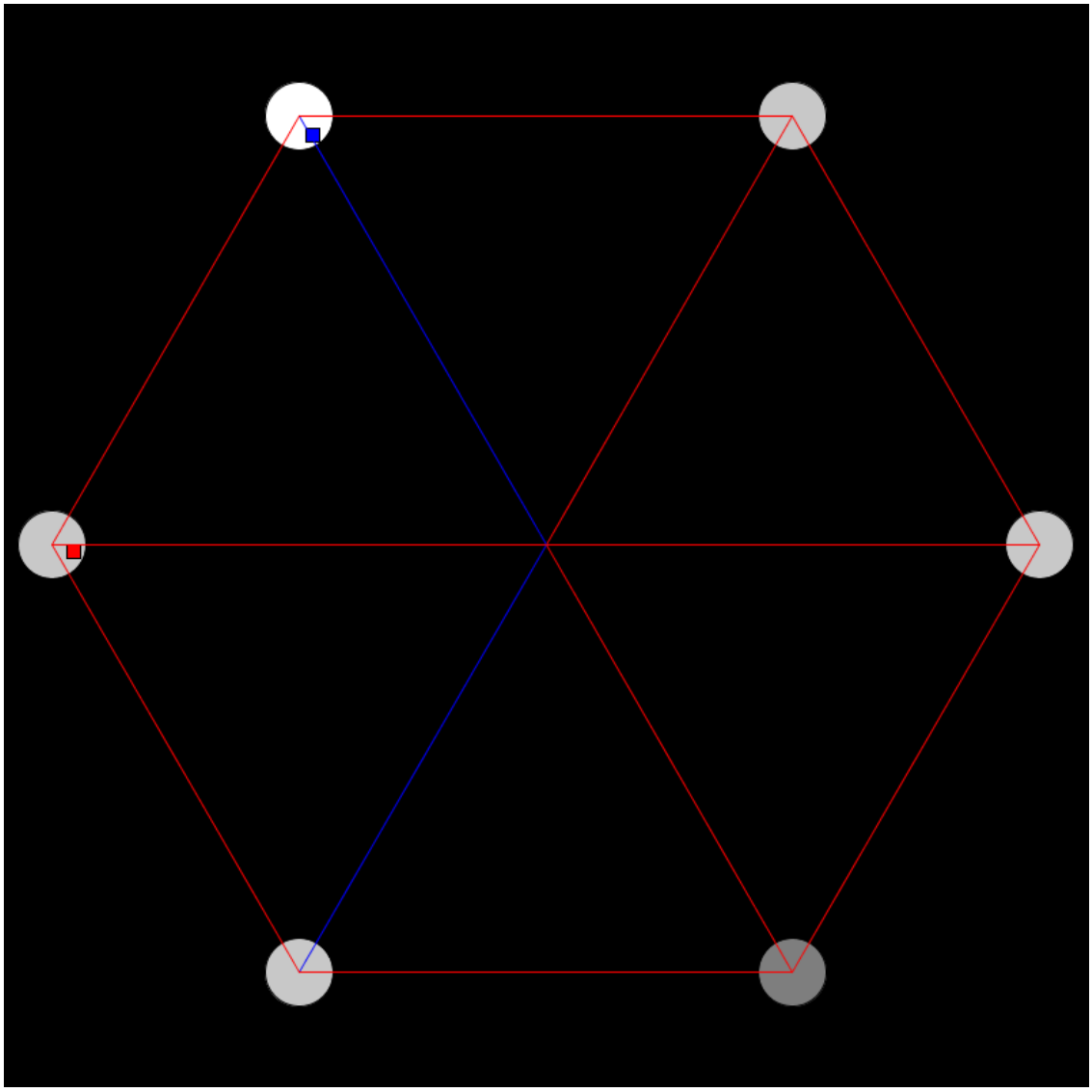

Figure 5: The most efficient model

Conclusion

A. Analysis

Our results verified our hypothesis. We found the most viable solution to our original problem. We found the best way of shipping supplies to a net of villages around a center. Our model displays the real world fairly accurately because of the pauses in the program where a theoretical storm passed, and the way the trucks and ATVs acted. Of course, the real world variables change depending on location, but for our purposes, the model's variables worked well. We can now

take our results here, and implement it into a real group of villages somewhere in central Africa. The idea we would implement would remain the same. There would be a distribution center between the villages. From there, two nice dirt roads would extend to two villages, and the rest of the ten roads would be mere trails.

The one variable our model does not account for is the fact that in the real world, villages are not built around a central point. Our model assumes that the villages are equidistant from each other and the distribution center.

Our model does in fact have a flaw in it. If a village dies, trucks and ATV's will still travel to it. This is pointless, and just wastes money, which decreases efficiency. Within the parameters of our program, we could not stop certain functions when the villages died. The program never recognized the villages as dead, rather, it just say the variable food under 0. We then translated that to the village is dead. This problem could have been fixed simply by acknowledging the fact that a food value under 0 meant that the village was dead, and then telling the ATV, and truck to not go to that village. This simple fix can be added in the future.

B. Future Development

In the future, we would like to fix the issues explained in the analysis section. This would help expand, and make a more accurate model. We would also like to create a model based off of a geologically accurate location in Central Africa. This would show how effective our model design is in real life. Lastly, we would also like to test the effects of adding villages, or subtracting villages, and expanding the fleet of vehicles. This way we could find an even more effective way to serve a multitude of villages. With this extra data, we could expand to a more realistic model.

Acknowledgements

We would like to thank Jocelyne Comstock and Jeff Mathis for their devotion to

our team. They gave use a lot of good advice, and they also helped us stay track. They also helped use proofread most of our work. We would like to thank Kim Kaare-Rasmussen for helping us with the more technical elements, and teaching us the basics of coding.

References

[1] Gunders, Dana, “WASTED: How America is Losing 40 Percent of Its Food From Farm to Fork to Landfill”, Web, August 2012, 26.

[2] Processing. "Overview." *Processing*. Processing, n.d. Web. 20 Mar. 2014.

[3] Coolgeography. "Global Patterns of Food Supply, Consumption and Trade."*Coolgeography*. Coolgeography, n.d. Web. 23 Mar. 2014.

Appendixes

A. Appendix

Table 1

	Route description	Time spent	Price paid
1	7 red; 3 blue; 2 green;	5.10E+07	7.47E+08
2	5 red; 4 blue; 3 green;	6.38E+07	1.49E+09
3	5 red; 4 blue; 3 green;	6.32E+07	1.22E+09
4	6 red; 1 blue; 5 green;	2.98E+07	1.16E+09
5	4 red; 4 blue; 4 green;	5.06E+07	1.39E+09
6	7 red; 3 blue; 2 green;	3.95E+07	6.58E+08
7	6 red; 6 blue; 0 green;	5.08E+07	3.21E+08
8	5 red; 4 blue; 3 green;	2.98E+07	8.35E+08
9	5 red; 2 blue; 5 green;	4.00E+07	1.39E+09
10	5 red; 4 blue; 3 green;	6.62E+09	1.26E+10

11	5 red; 3 blue; 4 green;	6.32E+07	1.49E+09
12	3 red; 7 blue; 2 green;	9.24E+07	1.30E+09
13	7 red; 3 blue; 2 green;	1.44E+07	3.94E+08
14	1 red; 8 blue; 3 green;	7.72E+07	1.62E+09
15	4 red; 4 blue; 4 green;	5.09E+07	1.39E+09
16	3 red; 4 blue; 5 green;	1.27E+08	2.67E+09
17	5 red; 4 blue; 3 green;	1.27E+08	1.74E+09
18	6 red; 3 blue; 3 green;	5.11E+07	1.04E+09
19	7 red; 2 blue; 3 green;	2.98E+07	7.53E+08
20	4 red; 6 blue; 2 green;	1.27E+08	1.44E+09
21	7 red; 2 blue; 3 green;	3.95E+07	8.69E+08
22	3 red; 4 blue; 5 green;	7.71E+07	2.08E+09
23	3 red; 2 blue; 7 green;	5.07E+07	2.16E+09
24	5 red; 4 blue; 3 green;	6.32E+07	1.22E+09
25	4 red; 6 blue; 2 green;	9.33E+07	1.23E+09
26	4 red; 6 blue; 2 green;	5.07E+07	9.07E+08
27	5 red; 1 blue; 6 green;	6.34E+07	2.03E+09
28	6 red; 4 blue; 2 green;	3.97E+07	7.05E+08
29	1 red; 4 blue; 7 green;	7.72E+07	2.80E+09
30	1 red; 7 blue; 4 green;	5.07E+07	1.55E+09
31	4 red; 6 blue; 2 green;	3.95E+07	8.00E+08
32	5 red; 4 blue; 3 green;	3.95E+07	9.64E+08
33	0 red ;4 blue; 8 green;	6.31E+07	2.86E+09
34	7 red; 3 blue; 2 green;	2.98E+07	5.70E+08
35	4 red; 6 blue; 2 green;	9.25E+07	1.23E+09

36	5 red; 4 blue; 3 green;	6.33E+07	1.22E+09
37	2 red; 5 blue; 5 green;	5.06E+07	1.73E+09
38	3 red; 6 blue; 3 green;	5.07E+07	1.20E+09
39	4 red; 7 blue; 1 green;	3.96E+07	5.89E+08
40	5 red; 5 blue; 2 green;	1.09E+08	1.26E+09
41	3 red; 3 blue; 6 green;	1.28E+08	3.05E+09
42	5 red; 1 blue; 6 green;	5.07E+07	1.81E+09
43	5 red; 4 blue; 3 green;	2.97E+07	8.35E+08
44	4 red; 4 blue; 4 green;	3.94E+07	1.22E+09
45	8 red; 3 blue; 2 green;	1.27E+08	7.21E+08
46	3 red; 6 blue; 3 green;	5.12E+07	1.20E+09
47	5 red; 5 blue; 2 green;	2.97E+07	6.52E+08
48	6 red; 1 blue; 5 green;	3.95E+07	1.34E+09
49	2 red; 6 blue; 4 green;	9.25E+07	1.95E+09
50	2 red; 5 blue; 5 green;	5.12E+07	1.73E+09
51	2 red; 5 blue; 5 green;	7.77E+07	2.14E+09
52	5 red; 4 blue; 3 green;	3.95E+07	9.64E+08
53	3 red; 6 blue; 3 green;	3.95E+07	1.06E+09
54	2 red; 7 blue; 3 green;	6.32E+07	1.40E+09
55	4 red; 6 blue; 2 green;	3.95E+07	8.00E+08
56	3 red; 4 blue; 5 green;	6.33E+07	1.88E+09
57	5 red; 5 blue; 2 green;	3.96E+07	7.52E+08
58	5 red; 2 blue; 5 green;	3.95E+07	1.39E+09
59	7 red; 4 blue; 1 green;	1.27E+08	8.06E+08
60	3 red; 5 blue; 4 green;	3.96E+07	1.27E+09

61	7 red; 3 blue; 2 green;	1.09E+08	1.10E+09
62	4 red; 6 blue; 1 green;	3.95E+07	1.86E+09
63	3 red; 5 blue; 4 green;	1.09E+08	2.47E+09
64	3 red; 5 blue; 4 green;	5.07E+07	1.44E+09
65	2 red; 5 blue; 5 green;	5.07E+07	1.73E+09
66	3 red; 6 blue; 3 green;	5.07E+07	1.92E+09
67	7 red; 5 blue; 0 green;	1.28E+08	4.25E+08
68	9 red; 1 blue; 2 green;	1.44E+07	3.37E+08
69	4 red; 5 blue; 3 green;	6.32E+07	1.82E+09
70	6 red; 3 blue; 3 green;	3.95E+07	9.17E+08
71	6 red; 1 blue; 5 green;	3.95E+07	1.34E+09
72	2 red; 6 blue; 4 green;	7.72E+07	1.84E+09
73	4 red; 6 blue; 2 green;	1.27E+08	1.44E+09
74	2 red; 2 blue; 8 green;	5.07E+07	2.45E+09
75	5 red; 2 blue; 5 green;	3.95E+07	1.39E+09
76	8 red; 3 blue; 1 green;	1.10E+08	6.68E+08
77	3 red; 5 blue; 4 green;	5.07E+07	1.44E+09
78	2 red; 2 blue; 8 green;	5.07E+07	2.45E+09
79	3 red; 6 blue; 3 green;	6.45E+07	1.34E+09
80	5 red; 3 blue; 4 green;	7.77E+07	1.65E+09
81	2 red; 3 blue; 7 green;	9.26E+07	2.99E+09
82	4 red; 7 blue; 3 green;	5.07E+07	1.15E+09
83	3 red; 7 blue; 4 green;	7.79E+07	1.78E+09
84	5 red; 2 blue; 5 green;	3.95E+07	1.39E+09
85	3 red; 6 blue; 3 green;	2.98E+07	9.16E+08

86	3 red; 4 blue; 5 green;	5.07E+07	1.68E+09
87	4 red; 5 blue; 3 green;	4.00E+07	1.01E+09
88	4 red; 5 blue; 3 green;	6.45E+07	1.28E+09
89	4 red; 3 blue; 5 green;	6.33E+07	1.82E+09
90	5 red; 2 blue; 5 green;	4.02E+07	1.39E+09
91	3 red; 7 blue; 2 green;	9.24E+07	1.30E+09
92	1 red; 5 blue; 6 green;	1.68E+08	3.70E+09
93	3 red; 5 blue; 4 green;	6.37E+07	1.61E+09
94	6 red; 2 blue; 4 green;	7.75E+07	1.58E+09
95	4 red; 3 blue; 5 green;	3.96E+07	1.43E+09
96	1 red; 2 blue; 9 green;	7.72E+07	3.39E+09
97	4 red; 2 blue; 6 green;	6.33E+07	2.09E+09
98	6 red; 5 blue; 1 green;	3.96E+07	4.94E+08
99	5 red; 4 blue; 3 green;	6.32E+07	1.22E+09
100	6 red; 2 blue; 4 green;	6.34E+07	1.43E+09

Note: Red row (#10) is the outlier referred to on page 11. The Green row(#67) is the most efficient result.

Table 2

	Description	Time spent	Price paid
1	0,2,2,2,2,2,0,0,0,0,0,0	2.97E+07	2.04E+08
2	0,2,2,2,2,0,0,2,0,0,0,0	3.95E+07	2.36E+08
3	0,2,0,2,0,0,2,2,2,0,0,0	1.09E+08	3.93E+08
4	0,2,2,0,0,0,2,2,2,0,0,0	6.61E+09	3.07E+09
5	2,0,0,2,0,0,0,0,2,0,2,2	1.28E+08	4.25E+08
6	0,0,0,0,0,0,0,2,2,2,2,2	1.43E+07	1.41E+08

7	0,0,2,2,2,0,0,2,0,0,0,2	4.28E+07	2.44E+08
8	0,0,2,2,2,2,0,0,0,0,0,2	2.98E+07	2.04E+08
9	0,0,2,2,0,0,0,2,0,2,0,2	1.09E+08	3.93E+08
10	0,0,2,2,0,2,0,0,0,2,0,2	2.97E+07	2.04E+08

Table 3

	Description	Time spent	Price paid
1	Original	3.95E+07	2.36E+08
2	Minus one road	3.97E+07	1.89E+08
3	Minus other road	3.95E+07	1.41E+08
4	Minus one other road, just as a test drive	1.09E+08	1.57E+08

B. Appendix

Code

```
//announce the fact that these are going to happen
Village V1, V2, V3, V4, V5, V6;
Road R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12;
ATV A;
TRUCK T;
//universal variables
int NX = 800;
int NY = 800;
//sc = scale
float sc = 22.0/NX;
float road_length=10;
int V_no=0;
float price = 0;
```

```
float time = 0;



//setup
void setup()
{
  float x, y;
  float d2r=PI/3.0;
  size(NX, NY);
  fill(255, 126);
  //set first village place
  x=road_length*cos(0.0*d2r);
  y=road_length*sin(0.0*d2r);
  V1 = new Village(x, y, 3, 3, 1);
  //set 2nd village place
  x=road_length*cos(d2r);
  y=road_length*sin(d2r);
  V2 = new Village(x, y, 2, 2, 3);
  //set 3rd village place
  x=road_length*cos(2.0*d2r);
  y=road_length*sin(2.0*d2r);
  V3 = new Village(x, y, 1, 3, 2);
  //set 4th village place
  x=road_length*cos(3.0*d2r);
  y=road_length*sin(3.0*d2r);
  V4 = new Village(x, y, 3, 1, 3);
```

```
//set 5th village place
x=road_length*cos(4.0*d2r);
y=road_length*sin(4.0*d2r);
V5 = new Village(x, y, 1, 2, 1);
//set 6th village place
x=road_length*cos(5.0*d2r);
y=road_length*sin(5.0*d2r);
V6 = new Village(x, y, 2, 1, 2);
//create ATV
A=new ATV(0, 0);
price = price + 20000;
//create Truck
T=new TRUCK(0, 0);
price = price + 45600;

//create all roads
//  R1 = new Road(0, 0, V1.x, V1.y, int(random(3)));
//  R2 = new Road(0, 0, V2.x, V2.y, int(random(3)));
//  R3 = new Road(0, 0, V3.x, V3.y, int(random(3)));
//  R4 = new Road(0, 0, V4.x, V4.y, int(random(3)));
//  R5 = new Road(0, 0, V5.x, V5.y, int(random(3)));
//  R6 = new Road(0, 0, V6.x, V6.y, int(random(3)));
//  //connecting roads
//  R7 = new Road(V1.x, V1.y, V2.x, V2.y, int(random(3)));
//  R8 = new Road(V2.x, V2.y, V3.x, V3.y, int(random(3)));
//  R9 = new Road(V3.x, V3.y, V4.x, V4.y, int(random(3)));
//  R10 = new Road(V4.x, V4.y, V5.x, V5.y, int(random(3)));
```

```
  //  R11 = new Road(V5.x, V5.y, V6.x, V6.y, int(random(3)));
  //  R12 = new Road(V6.x, V6.y, V1.x, V1.y, int(random(3)));
  //fake roads
  R1 = new Road(0, 0, V1.x, V1.y, 0);
  R2 = new Road(0, 0, V2.x, V2.y, 0);
  R3 = new Road(0, 0, V3.x, V3.y, 2);
  R4 = new Road(0, 0, V4.x, V4.y, 0);
  R5 = new Road(0, 0, V5.x, V5.y, 2);
  R6 = new Road(0, 0, V6.x, V6.y, 0);
  //connecting roads
  R7 = new Road(V1.x, V1.y, V2.x, V2.y, 0);
  R8 = new Road(V2.x, V2.y, V3.x, V3.y, 0);
  R9 = new Road(V3.x, V3.y, V4.x, V4.y, 0);
  R10 = new Road(V4.x, V4.y, V5.x, V5.y, 0);
  R11 = new Road(V5.x, V5.y, V6.x, V6.y, 0);
  R12 = new Road(V6.x, V6.y, V1.x, V1.y, 0);
}
//draw loop
void draw() {
  background(0);
  //diffrent functions
  display();
  Distribution_Center();
  nd();
  if (nd == false) {
    update_Villages();
  }
```

```
  update_vehicles();
  death();
  time = time+millis();
}
//defining villages
class Village {
  float x, y; // The x- and y-coordinates
  float food = 2000;
  int Center_Road;
  int Clockwise_Road;
  int Count_Clockwise_Road;
  boolean serviced = false;
  Village(float xpos, float ypos, int C, int CW, int CCW) {
    x =xpos;
    y =ypos;
    Center_Road=C;
    Clockwise_Road=CW;
    Count_Clockwise_Road=CCW;
  }
  //function for displaying villages
  void display() {
    int col;

    col=int(255.0/2000.0*food);
    if (col>255) col = 255;
    if (col<0) col = 0;
```

```
  fill(col);
  stroke(0);
  //disply
  ellipse(ConX(x), ConY(y), 50, 50);
 }
 //food consumption
 void eat() {
  food=food-2;
 }
 //getting food
 void receive() {
  float dis;
  //if distance < 2/3+speed, reset ATV destination and origin
  dis=sqrt((x-A.x)*(x-A.x)+(y-A.y)*(y-A.y));
  if (dis< 2.0/3.0 * A.speed) {
   food=food+576;
   A.OX=x;
   A.OY=y;
   A.DX=0.0;
   A.DY=0.0;
   serviced = false;
   price = price + 4319.37;
  }
  //if distance < 2/3+speed, reset ATV destination,
  //second destinationand origin
  dis=sqrt((x-T.x)*(x-T.x)+(y-T.y)*(y-T.y));
  if (dis< 2.0/3.0 * T.speed) {
```

```
      food=food+576.5;
      T.OX=x;
      T.OY=y;
      price = price+4319.895;
      if ((T.DX==T.DDX)&&(T.DY==T.DDY)) {
        T.DX=0.0;//return home
        T.DY=0.0;
      }
      else {
        T.DX=T.DDX;
        T.DY=T.DDY;
      }
      serviced = false;
    }
  }
}
//define roads
class Road {
  float x1, x2, y1, y2;
  int condition;
  Road(float xpos1, float ypos1, float xpos2, float ypos2, int c) {
    x1 = xpos1;
    y1 = ypos1;
    x2 = xpos2;
    y2 = ypos2;
    condition=c;
  }
```

```
  //show roads
  void display() {
    if (condition == 0) {
      stroke(255, 0, 0);
    }
    else if (condition == 1) {
      stroke(1, 255, 0);
      price = price + 120000;
    }
    else if (condition == 2) {
      stroke(1, 1, 255);
      price = price + 21875;
    }
    line(ConX(x1), ConY(y1), ConX(x2), ConY(y2));
  }
}
//define ATV
class ATV {
  float x, y; // The x- and y-coordinates
  float speed=0.1;
  float OX, OY, DX, DY;
  ATV(float xpos, float ypos) {
    x = xpos;
    y = ypos;
  }
  //show ATV
  void display() {
```

```
    fill(255, 0, 0);
    stroke(0);
    rect(ConX(x), ConY(y), 10, 10);
  }
  //drive (using pythagreons theorm)
  void drive1() {
    float a, c, b;

    a = DX-OX;
    b = DY-OY;
    c = dist(OX, OY, DX, DY);
    x = x + (a/c)*speed;
    y = y + (b/c)*speed;
  }
}
//define truck
class TRUCK {
  float x, y;                    // The x- and y-coordinates
  float speed=0.1;                  //speed adjustment
  float OX, OY, DX, DY, DDX, DDY;      // we need of rest period
  TRUCK(float xpos, float ypos) {
    x = xpos;
    y = ypos;
  }
  //show truck
  void display() {
```

```
    fill(0, 0, 255);
    stroke(0);
    rect(ConX(x), ConY(y), 10, 10);
  }
  //drive (using pythagreons theorm)
  void drive() {
    float a, c, b;

    a = DX-OX;
    b = DY-OY;
    c = dist(OX, OY, DX, DY);
    x = x + (a/c)*speed;
    y = y + (b/c)*speed;
  }
}
//define hungry
int hungry() {
  float L_food=10000;
  int V_no=0;
  //ask if v1 has least food, and its not serviced
  if ((V1.food<L_food)&(!V1.serviced)) {
    L_food=V1.food;
    V_no=1;
  }
  if ((V2.food<L_food)&(!V2.serviced)) {
    L_food=V2.food;
```

```
    V_no=2;
  }
  if ((V3.food<L_food)&(!V3.serviced)) {
    L_food=V3.food;
    V_no=3;
  }
  if ((V4.food<L_food)&(!V4.serviced)) {
    L_food=V4.food;
    V_no=4;
  }
  if ((V5.food<L_food)&(!V5.serviced)) {
    L_food=V5.food;
    V_no=5;
  }
  if ((V6.food<L_food)&(!V6.serviced)) {
    L_food=V6.food;
    V_no=6;
  }
  return V_no;
}
//truck version of hungry
//diffrence: checks if roads are passable
int hungry_truck() {
  float L_food=10000;
  int V_no=0;
  if ((V1.food<L_food)&&(!V1.serviced)&&(R1.condition>0)) {
    L_food=V1.food;
```

```
    V_no=1;
  }
  if ((V2.food<L_food)&&(!V2.serviced)&&(R2.condition>0)) {
    L_food=V2.food;
    V_no=2;
  }
  if ((V3.food<L_food)&&(!V3.serviced)&&(R3.condition>0)) {
    L_food=V3.food;
    V_no=3;
  }
  if ((V4.food<L_food)&&(!V4.serviced)&&(R4.condition>0)) {
    L_food=V4.food;
    V_no=4;
  }
  if ((V5.food<L_food)&&(!V5.serviced)&&(R5.condition>0)) {
    L_food=V5.food;
    V_no=5;
  }
  if ((V6.food<L_food)&&(!V6.serviced)&&(R6.condition>0)) {
    L_food=V6.food;
    V_no=6;
  }
  return V_no;
}
//define Center
void Distribution_Center() {
  float dis_ATV, dis_TRUCK;
```

```
 dis_ATV=sqrt(A.x*A.x+A.y*A.y);
 dis_TRUCK=sqrt(T.x*T.x+T.y*T.y);
 if (dis_ATV <2.0/3.0 * A.speed) {
  switch(hungry()) {
   //tells ATV where to go
  case 1:
   A.OX=0;
   A.OY=0;
   A.DX=V1.x;
   A.DY=V1.y;
   V1.serviced= true;
   break;
  case 2:
   A.OX=0;
   A.OY=0;
   A.DX=V2.x;
   A.DY=V2.y;
   V2.serviced= true;
   break;
  case 3:
   A.OX=0;
   A.OY=0;
   A.DX=V3.x;
   A.DY=V3.y;
   V3.serviced= true;
   break;
  case 4:
```

```
      A.OX=0;
      A.OY=0;
      A.DX=V4.x;
      A.DY=V4.y;
      V4.serviced= true;
      break;
    case 5:
      A.OX=0;
      A.OY=0;
      A.DX=V5.x;
      A.DY=V5.y;
      V5.serviced= true;
      break;
    case 6:
      A.OX=0;
      A.OY=0;
      A.DX=V6.x;
      A.DY=V6.y;
      V6.serviced= true;
      break;
    }
  }
  //tells truck where to go
  if (dis_TRUCK<2.0/3.0 * T.speed) {
    T.OX = 0.0;
    T.OY = 0.0;
    T.x = 0.0;
```

```
    T.y = 0.0;
    switch(hungry_truck()) { //switch works well for ATV, but not truck... new idea...
      //new hungry for truck.... but also checks road conditions
    case 1:

      T.DX = 0;
      T.DY = 0;
      T.DX = V1.x;
      T.DY = V1.y;
      V1.serviced = true;
      T.DDX = 0;
      T.DDY = 0;
      if ((R7.condition>0)&&(R2.condition>0)) {
        T.DDX = V2.x;
        T.DDY = V2.y;
        V2.serviced = true;
      }
      if ((R12.condition>0)&&(R6.condition>0)) {
        T.DDX = V6.x;
        T.DDY = V6.y;
        V6.serviced = true;
      }
      break;
    case 2:
      T.DX = 0;
      T.DY = 0;
```

```
    T.DX = V2.x;
    T.DY = V2.y;
    V2.serviced = true;
    T.DDX = 0;
    T.DDY = 0;
    if ((R7.condition>0) && (R1.condition>0)) {
      T.DDX = V1.x;
      T.DDY = V1.y;
      V1.serviced = true;
    }
    if ((R8.condition>0)&&(R3.condition>0)) {          //try to do not do anything
      T.DDX = V3.x;
      T.DDY = V3.y;
      V2.serviced = true;
    }
    break;
  case 3:
    T.DX = 0;
    T.DY = 0;
    T.DX = V3.x;
    T.DY = V3.y;
    V3.serviced = true;
    T.DDX = 0;
    T.DDY = 0;
    if ((R8.condition>0)&&(R2.condition>0)) {
      T.DDX = V2.x;
      T.DDY = V2.y;
```

```
      V2.serviced = true;
    }
    if ((R9.condition>0)&&(R4.condition>0)) {
      T.DDX = V4.x;
      T.DDY = V4.y;
      V4.serviced = true;
    }
    break;
  case 4:
    T.DX = 0;
    T.DY = 0;
    T.DX = V4.x;
    T.DY = V4.y;
    V4.serviced = true;
    T.DDX = 0;
    T.DDY = 0;
    if ((R9.condition>0)&&(R3.condition>0)) {
      T.DDX = V3.x;
      T.DDY = V3.y;
      V3.serviced = true;
    }
    if ((R10.condition>0)&&(R5.condition>0)) {
      T.DDX = V5.x;
      T.DDY = V5.y;
      V5.serviced = true;
    }
    break;
```

```
  case 5:
    T.DX = 0;
    T.DY = 0;
    T.DX = V5.x;
    T.DY = V5.y;
    V5.serviced = true;
    T.DDX = 0;
    T.DDY = 0;
    if ((R10.condition>0)&&(R4.condition>0)) {
      T.DDX = V4.x;
      T.DDY = V4.y;
      V4.serviced = true;
    }
    if ((R11.condition>0)&&(R6.condition>0)) {
      T.DDX = V6.x;
      T.DDY = V6.y;
      V6.serviced = true;
    }
    break;
  case 6:
    T.DX = 0;
    T.DY = 0;
    T.DX = V6.x;
    T.DY = V6.y;
    V6.serviced = true;
    T.DDX = 0;
    T.DDY = 0;
```

```
      if ((R11.condition>0)&&(R5.condition>0)) {
        T.DDX = V5.x;
        T.DDY = V5.y;
        V5.serviced = true;
      }
      if ((R12.condition>0)&&(R1.condition>0)) {
        T.DDX = V1.x;
        T.DDY = V1.y;
        V1.serviced = true;
      }
      break;
    }
  }
}
void death() {
  if ((V1.food <= 0)&&(V2.food <= 0)&&(V3.food <= 0)&&(V4.food <=
0)&&(V5.food <= 0)&&(V6.food <= 0)) {
    fill(255);
    tint(255, 127);
    rect(0, 0, width, height);
    stroke(0);
    fill(0);
    println("All Villages have died"+ "Time spent: " + time+".  Price paid: "+price);
  }
}

//change from JAVA (where X is the upper leg of display
```

```
//and y is the left leg of diplay
//to Cartesian system

float ConX(float x) {
  return x/sc+NX/2.0;
}
float ConY(float x) {
  return -x/sc+NX/2.0;
}
//compile all displays
void display() {
  V1.display();
  V2.display();
  V3.display();
  V4.display();
  V5.display();
  V6.display();
  A.display();
  T.display();
  //roads();
  R1.display();
  R2.display();
  R3.display();
  R4.display();
  R5.display();
  R6.display();
  R7.display();
```

```
  R8.display();
  R9.display();
  R10.display();
  R11.display();
  R12.display();
}
//compile all changes for villages
void update_Villages() {
  V1.eat();
  V2.eat();
  V3.eat();
  V4.eat();
  V5.eat();
  V6.eat();
  V1.receive();
  V2.receive();
  V3.receive();
  V4.receive();
  V5.receive();
  V6.receive();
}
//compile all changes for truck and ATV
void update_vehicles() {
  A.drive1();
  T.drive();
}
boolean nd = false;
```

```
int ndtime = 0;
void nd() {
  if (time==random(10000)) {
    nd = true;
    ndtime = millis();
    if (ndtime == random(100));
    nd = false;
    ndtime=0;
  }
}

//DONE!
```