Chaos theory and the evolution of Traffic law

New Mexico
Super Computing Challenge
April 28, 2008
Team 13
Artesia High School

Team Members:
Phillip DeLaRosa
Alex Sifuentes
J.P. Rivera
Jake Green

Teacher:
Mr. Gaylor

Mentor:
Nick Bennett

Contents

Executive Summary

The purpose of this project was to model chaos theory, which states that complex systems evolve from simple sets of rules. We decided to model this in a Java-based traffic simulation. We hypothesized that the cars would evolve by learning the best paths for reaching their destinations in this traffic environment.

Introduction

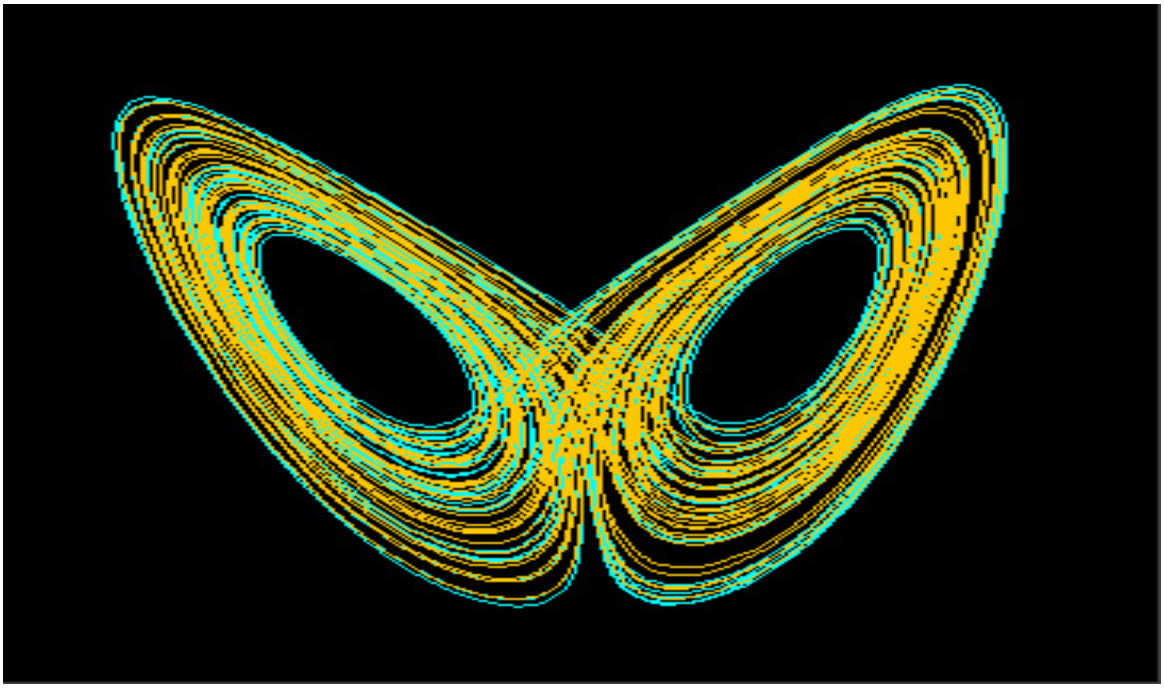
Ian Stewart, Does God Play Dice? The Mathematics of Chaos, stated:

"The flapping of a single butterfly's wing today produces at tiny change in the state of the atmosphere. Over a period of time, what the atmosphere actually does diverges from what it would have done. So, in a month's time, a tornado that would have devastated the Indonesian coast doesn't happen. Or maybe one that wasn't going to happen, does.(pg141)"

This quotation has become a classic statement for explaining "Chaos Theory". "Chaos Theory" is the idea, or theory that describes how systems of apparently random data develop mathematical patterns. "The Chaos Theory" describes apparent anomalies; one such result of "Chaos Theory" could be to observe how an existing complex system, such as Traffic Law, evolves from simple rules. The name "chaos theory" could come from the fact that the theory describes systems that are apparently disordered, but the theory is really about finding the underlying order in apparently random systems. A prime example of this is when in 1960 a meteorologist named Edward Lorenz was working on weather predictions. He had developed an equation to help with his weather predictions and had to use the same equation many times … so to speed up the process, he had a computer set up with a set of twelve equations to model the

weather. While the computer program did not predict the weather itself the computer program did theoretically predict what the weather "might be." Edward Lorenz had no problems with his computer generated predictions for almost a year until one day in 1961. Lorenz wanted to review a particular sequence once again and so, to save time, he started in the middle of the sequence, instead of the beginning. He entered the number from his printout and left the mathematical sequence running. When he came back later, the sequence had evolved differently then expected. Instead of the same pattern as before, it diverged from the original pattern, ending up with a wildly different pattern. After many tests and much time studying the pattern he eventually figured out what happened. The computer stored the numbers to six decimal places in its memory. To save paper, he only had it print out three decimal places. In the original sequence, the number was *.506127,* and he had only typed the first three digits *.506*. By all conventional ideas at the time, it should have worked. He should have gotten a sequence very close to the original sequence. In fact most, if not all, scientists consider themselves fortunate if they are able to attaint measurements with accuracy to three decimal places. Surely the fourth and fifth decimals, almost impossible to measure using reasonable methods, would have minimal, not a huge effect, on the outcome of the experiment. Lorenz proved this idea wrong and while Lorenz was studying the mathematical structure of the data of chaotic behavior, he used time-series analysis to ferret out hidden details and employed the technique to help produce an image bearing three dimensions of information. This image (an attractor) soon became extricably

tied to field of chaos (or nonlinear dynamics).



(A Lorenz-type image (above)… note it resembles a butterfly)

Within this image, one can detect an underlying fractal pattern. The apparently elliptical path is that followed by a particle drawing the trajectory in 3D. When this figure was drawn, the particle was seen looping around one lobe for a while and then jumping over to the other lobe as it continued looping. The path never retraced itself. (Though it appears to, that is simply a limitation of the drawing resolution of a computer screen.) This back-and-forth motion continued until the entire plot was generated. The image looks flat, but the tracing particle moved in three dimensions, alternately projecting out of the screen and then back in as it orbited the attractor.

This phenomenon, (“The Butterfly Effect,” named such after Lorenz’s speech, *Predictability: Does the Flap of a Butterfly’s Wings in Brazil set off a Tornado in Texas* , in which he states “While one butterfly's flapping wings could trigger off a tornado in the Lone Star state, another butterfly's flapping wings could prevent it.”(Lorenz)), is also known as a sensitive dependence on initial conditions, which can drastically change the long-term behavior of a

system. A great analogy for this is white water on a rapidly flowing section of a mountain stream. If you set two leaves in motion next to each other on the upstream side of the white water, they will most likely be widely separated by the time they reach the down stream side…to reiterate, in a system like this a small difference in its initial conditions (the position on the leaves) can result in a large difference in the outcome (Trefil118). Such a small amount of difference in a measurement might be considered an inaccuracy of the equipment of just white background noise. Such things are impossible to avoid in even the most isolated lab. With a starting number of two, the final result can be entirely different from the same system with a starting value of 2.000001!

This theory led Team Thirteen (Chaos theory and the evolution of traffic law) to ask the question: if such patterns could occur in seemingly random data, then could a seemingly organized system truly be the function of random data? The team then wondered if it was possible to create a system of order by giving a program, such as one programmed in Java, almost no data but only a few basic guidelines such as: (1) Don't allow agents to die (2) Have agents get from point A to point B. If such guidelines were given, would the basic properties of chaos theory prove true?

Model

The model is created in Java. It has four classes: car, move, grid, and test. The car class is used to create car objects. Car objects are objects that move around in the program's world. They know how to move, when they collide with other instances of the car class, and how to decide the shortest path. The move class decides in which direction an instance of the car class

will move. The grid class holds the data for the world. The test class creates a 10 by 10 world and places five instances of the car class on the world and decides their destinations. Instance 1 is created at (1, 1) and has a destination of (6, 6). Instance 2 is created at (2, 2) and has a destination of (7, 7). Instance 3 is created at (3, 3) and has a destination of (8, 8). Instance 4 is created at (4, 4) and has a destination of (9, 9). Instance 5 is created at (5, 5) and has a destination of (10, 10).

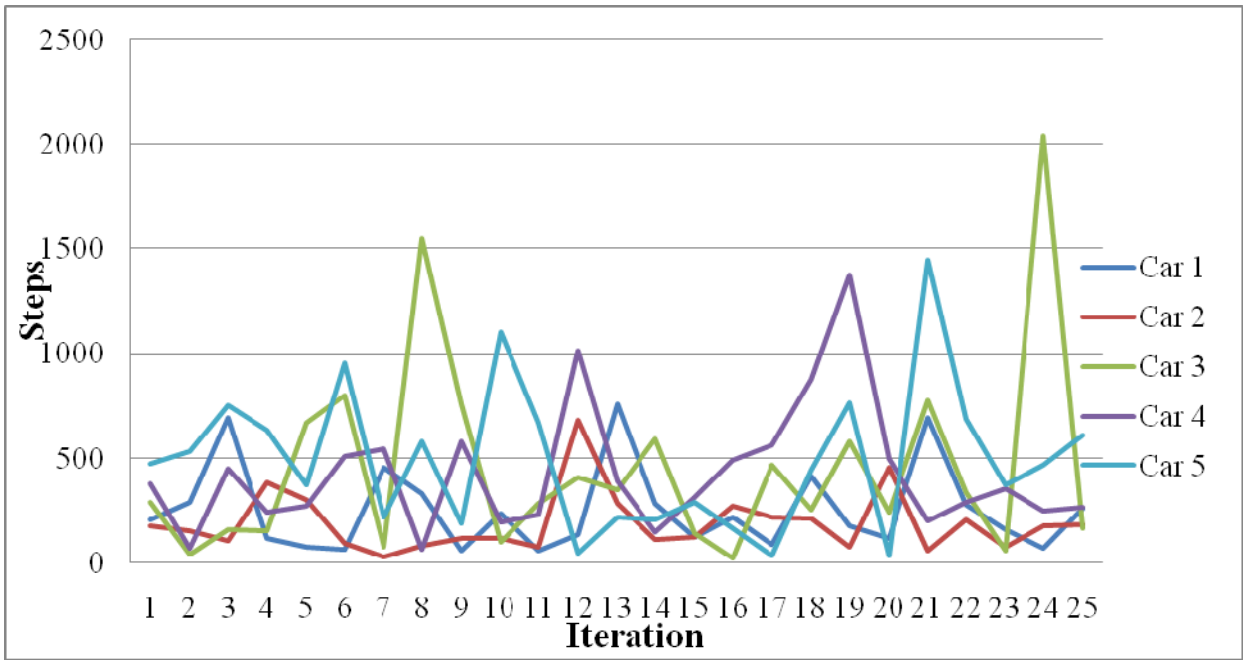
History

Our team had originally started out as a two person team (Alex S. and Phillip D.) but two weeks before the "SCC Kick-off" we gained our third team mate (Jake G.) We then attended the "Kick-off" and learned a lot of programming and got to meet many interesting people and even the programmer of Mathematica! Once we returned from the "Kick-off" we gained our fourth and final member of our team (JP R.) With all four of our team finally assembled we started to ritualistically meet every lunch on Tuesdays, Fridays and at certain weeks we also meet on Monday afternoons. The team did have its ups and downs however… for a few months (due to scheduling conflicts and break down in the line of communication) we, Team 13, almost ceased to exist! However, after many compromises and talks with our "SCC" teacher (Mr. Gaylor) our team had gotten back on track and was able to finish the project.

Tests and Results

Car 1	**Car 2**	**Car 3**	**Car 4**	**Car 5**
209	178	287	378	477
290	152	39	73	536
694	104	154	454	754

116	389	151	246	633
73	301	669	271	377
65	92	797	511	953
460	29	72	550	219
335	80	1549	66	580
58	120	756	581	189
239	119	96	196	1102
58	76	284	234	668
136	684	407	1019	48
761	284	346	390	220
284	110	592	150	210
123	123	139	316	290
218	276	20	493	169
88	217	465	567	39
414	213	251	880	443
177	78	579	1379	768
118	455	238	501	37
697	60	775	205	1447
280	208	335	294	687
159	76	54	355	376
67	178	2044	251	472
263	183	160	267	609



Conclusion

We found that the cars did not learn to ultimately develop shorter paths. Thus, our hypothesis was proven false.

Work citied

Trefil, James , The nature of science. Boston: Houghton Mifflin Company, (2003): 118-122

Lorenz, Edward, "Predictability - Does the Flap of a Butterfly's Wings in Brazil Set Off a Tornado in Texas?"(Paper presented to the American Association for the Advancement of Science, Washington, D.C., December 1972).

Stewart, Ian, Does God Play Dice? The mathematics of chaos. Malden, MA: Blackwell Publishers, Inc., 2002

Code

```
Package basic;


import java.awt.Point;
import java.util.ArrayList;



public class Test {

    public Test() {
    }



    public static void main(String[] args) {
        Grid grid = new Grid(10, 10, false);
        Car car1 = new Car();
        Car car2 = new Car();
        Car car3 = new Car();
        Car car4 = new Car();
        Car car5 = new Car();
        car1.setOrigin(new Point(1, 1));
        car1.getDestination().add(new Point(6, 6));
```

```
car2.setOrigin(new Point(2, 2));
car2.getDestination().add(new Point(7, 7));
car3.setOrigin(new Point(3, 3));
car3.getDestination().add(new Point(8, 8));
car4.setOrigin(new Point(4, 4));
car4.getDestination().add(new Point(9, 9));
car5.setOrigin(new Point(5, 5));
car5.getDestination().add(new Point(10, 10));
while (!car1.move() | !car2.move() | !car3.move() | !car4.move() | !car5.move()) {
    ArrayList<Car.Result> results = car1.getResults();
    System.out.println(results.get(results.size() - 1));
    results = car2.getResults();
    System.out.println(results.get(results.size() - 1));
    results = car3.getResults();
    System.out.println(results.get(results.size() - 1));
    results = car4.getResults();
    System.out.println(results.get(results.size() - 1));
    results = car5.getResults();
    System.out.println(results.get(results.size() - 1));
}
System.out.println(car1.getTotalMoves());
System.out.println(car2.getTotalMoves());
System.out.println(car3.getTotalMoves());
```

```
        System.out.println(car4.getTotalMoves());
        System.out.println(car5.getTotalMoves());
    }

}
```

```
package basic;

import java.util.Random;



public enum Move {
    NORTH,
    EAST,
    WEST,
    SOUTH,
    NONE;

    private static Random rng = new Random();

    public static Move getRandomMove() {
        Move result = NONE;
        switch (rng.nextInt(Move.values().length)) {
```

```
            case 0:
                result = NORTH;
                break;
            case 1:
                result = EAST;
                break;
            case 2:
                result = WEST;
                break;
            case 3:
                result = SOUTH;
                break;
            default:
                result = NONE;
                break;
        }
        return result;
    }
}
```

```
package basic;
```

```
import java.awt.Point;

public class Grid {

    private static Grid singleton;

    private Point dimensions;
    private boolean wrapped;
    private int[][] status;

    public Grid(int rows, int columns, boolean wrapping) {
        dimensions = new Point(columns, rows);
        this.wrapped = wrapped;
        status = new int [rows][columns];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                status[i][j] = 0;
            }
        }
        singleton = this;
    }
```

```
public static Grid getCurrentGrid() {
    return singleton;
}


public Point update(Point start, Move move){
    int x = start.x - 1;
    int y = start.y - 1;
    boolean crash = false;
    status[y][x]--;
    switch (move){
        case NORTH:
            if (y == dimensions.y - 1) {
                if (wrapped) {
                    y = 0;
                }
            }
            else {
                y++;
            }
            break;
        case EAST:
            if (x == dimensions.x - 1) {
                if (wrapped) {
```

```
            x = 0;
        }
    }
    else {
        x++;
    }
    break;
case SOUTH:
    if (y == 0) {
        if (wrapped) {
            y = dimensions.y;
        }
    }
    else {
        y--;
    }
    break;
case WEST:
    if (x == 0) {
        if (wrapped) {
            x = dimensions.x;
        }
    }
```

```
            else {
                x--;
            }
            break;
        default:
            break;
    }
    crash = (status[y][x] > 0);
    status[y][x]++;
    x++;
    y++;
    if (crash) {
        x = -x;
        y = -y;
    }
    return new Point(x, y);
}

public void updateIntersection(Point point) {
    int x = point.x - 1;
    int y = point.y - 1;
    status[y][x]++;
}
```

```
}



package basic;

import java.awt.Point;
import java.util.ArrayList;
import java.util.Iterator;



public class Car {

    public static class Result {

        private Move move;
        private Point destination;
        private boolean crash;

        public Result (Move move, Point destination, boolean crash) {
            this.move = move;
            this.destination = new Point(destination);
            this.crash = crash;
```

```
}

public Move getMove() {
    return move;
}

public void setMove(Move move) {
    this.move = move;
}

public Point getDestination() {
    return destination;
}

public void setDestination(Point destination) {
    this.destination = destination;
}

public boolean isCrash() {
    return crash;
}

public void setCrash(boolean crash) {
```

```
            this.crash = crash;
        }

        public String toString() {
            return String.format("Moved %s to (%d, %d). Crashed? %b", move, destination.x,
destination.y, crash);

        }
    }

    private Point origin;
    private ArrayList<Point> destination;
    private ArrayList<Result> results;
    private Point location;
    private boolean arrived = false;

    private int totalMoves = 0;

    public Car() {
        results = new ArrayList<Result>();
        destination = new ArrayList<Point>();
    }
```

```
public Point getOrigin() {
    return origin;
}

public void setOrigin(Point origin) {
    this.origin = new Point(origin);
    Grid.getCurrentGrid().updateIntersection(origin);
    this.location = this.origin;
}

public ArrayList<Point> getDestination() {
    return destination;
}

public ArrayList<Result> getResults() {
    return results;
}

public Point getLocation() {
    return location;
}

public void setLocation(Point location) {
```

```
    this.location = location;
}

public boolean move() {
    if (!arrived) {
        Move direction = Move.getRandomMove();
        Point tempResult = Grid.getCurrentGrid().update(location, direction);
        Result result;
        boolean crash = false;
        boolean deleting = false;
        Iterator<Result> iterator;
        if (tempResult.x < 0) {
            crash = true;
            tempResult.x = -tempResult.x;
            tempResult.y = -tempResult.y;
        }
        result = new Result(direction, tempResult, crash);
        iterator = results.iterator();
        while (iterator.hasNext()) {
            Result searchResult = iterator.next();
            if (deleting) {
                iterator.remove();
            }
```

```
            else {
                if (searchResult.getDestination().equals(tempResult)) {
                    iterator.remove();
                    deleting = true;
                }
            }
        }
        results.add(result);
        totalMoves++;
        location = tempResult;
        arrived = destination.contains(tempResult);
        return arrived;
    }
    else {
        return true;
    }
  }

  public int getTotalMoves() {
    return totalMoves;
  }
}
```