Programming and Problem Solving:
A Transcript of the Spring 1999 Class

Columbia University Computer Science Technical Report Number CUCS-018-99

Kenneth A. Ross and Simon R. Shamoun

May, 1999

This report contains edited transcripts of the discussion held in Columbia's Programming and Problem-Solving course, taught as W4995-01 during Spring 1999. The class notes were taken by the teaching assistant so that students could focus on the class material. As a service to both the students and to others who would like to get some insight into the class experience, we have drawn all of the class handouts, discussion, and some of the results into this technical report.

Contents

1 Introduction

The material handed out on the first day of class, including the problem statements, is presented in the next section. The class transcripts assume the reader is familiar with the problem statements.

1.1 The First Handout: Course Description and Problem Statements

Spring, 1999
Mondays and Wednesdays, 1:10–2:25, Prof. K. Ross
486 CS Building, the CLIC Lab (enter through Fairchild)

Information sheet.

The Course

This course is like no other course you've taken. There are no exams. There are no homeworks. There are 5 projects, but all of the projects are open-ended: there are no "correct" project answers. The idea is to develop problem solving skills using techniques that you have learned during your CS training to solve the project problems. Lectures will be relatively informal, with most of the class interaction being cooperative problem solving and discussion rather than a fixed set of lectures with questions.

Some of the supplied problems may be too tough to make much progress on in the 3 weeks you have. If so, you should try to find special cases of the problem that can be solved, or to simplify the problem in such a way that it still remains interesting, but may be solved.

Conversely, some of the problems may be too easy. In that case you should try generalizing the problem. To what other kinds of problem can your solution also be applied?

All projects will be done in groups of four. There are 5 separate projects, and groups will change from project to project. *You cannot be in the same group as another person more than twice.*

A large proportion of your grade will come from class participation. *Class attendance is mandatory.* Do not sign up for this class if you think you will need to miss classes. An attendance record will be kept by the TA.

This is the kind of course in which you can invest a little or a lot, and for which your enjoyment and fulfilment will come in proportion to the amount of energy you put into the course.

Prerequisites

W3156 (Software Engineering), W3139 (Data Structures), and W3824 (Computer Organization). **Permission of the instructor is required for all students.** Senior CS or Computer Engineering majors and CS graduate students only. (Advanced juniors whose CS preparation is comparable to that of graduating seniors will also be considered.) Enrollment will be limited to 40 students.

Office Hours

Instructor: Prof. Ken Ross, 510 Computer Science Building, 212-939-7058, `kar@cs.columbia.edu`

Office hours: Mondays 2:30–3:30, Wednesdays 4:30–5:30.

Teaching Assistant: Simon Shamoun

Grading

Projects will be graded by the instructor. However, there is no predefined grading scheme. A grade will be given based on the approach used to solving the given problem, including

- the novelty of the approach
- the thoroughness of the project report

- the clarity of the report presentation
- the quality of the code
- the correctness and generality of the proposed solution
- the efficiency of the proposed solution

Several of the projects have competitions to see whose program "wins" against other programs. The outcome of these competitions is not a factor in the project grades. The competitions are intended to stimulate the development of new ideas, and it is the ideas that are important. The competition ranking are for fun and pride only.

There is no restriction on communication between different groups. However, if a certain idea X is proposed by group A, and group B thinks the idea is so cool that they want to use it too, then group B can do so *as long as group B explicitly cites group A's contribution in their report.* So, for example, suppose that during the second class discussing project 1, a student Jane Smith reports to the class that she has discovered that a certain algorithm in the literature applies to the problem. Then other groups are still free to use that algorithm too, but in their report they must include an "acknowledgements" section with text something like

> We acknowledge the contribution of Jane Smith who observed that the 17-starving-philosophers-named-Josephus algorithm applies to this problem.

Any information supplied by the instructor or teaching assistant need not be acknowledged in this way. Published sources and Web sources should be formally cited.

If a group develops a tool that is potentially useful to other groups, then you are encouraged to share it (and it's use should be acknowledged). However, this sharing should be limited to tools that aid the development of solutions (eg., visualizers, compilers, etc.) and not the solutions themselves. Similarly, external data that might be useful should be shared. For example, if we had a map-drawing project, then if somebody downloaded from the Web some datasets describing real-world maps that could be used to test project code, that data should be shared and used with acknowledgement. (For the purposes of this course, the use of other people's contributions without acknowledgement is considered cheating.)

Outside people may be consulted for general references, but shouldn't be asked to contribute to solving the problems themselves. This is a course whose primary purpose is to give you the opportunity to tackle challenging problems yourselves. If you consult outside people, they should also be acknowledged.

While groups are permitted to communicate with each other, they are not encouraged to work so closely together that they really could be described as an 8-person group. Groups must submit their own reports, and the reports should be written independently of other groups.

Here are the grading proportions:

Project Reports and Presentations	60%
Class Participation	40%

All members of a group will get the same grade for their project report.

In addition, you are required to attend all classes. You can miss two classes without penalty. However, the third class missed will result in a drop in the *final grade* of one fractional grade (eg., B+ to B, or A to A-). If you miss four classes, you will drop a full grade (eg., A- to B-). If you miss five or more classes, you will fail the course. Arriving in class more than 15 minutes late counts as missing the class. (In the event of a medical or other emergency, consult the instructor.)

Policies

There will be no extensions for projects. Hand in what you have on the due date.

An electronic newsgroup will be set up for all course-related announcements. Check there regularly for any course related announcements. Questions or comments of a general nature that would be of interest to the whole class should be posted there. The instructor and the TA will be reading the newsgroup regularly.

A web site for the course is at `http://www.cs.columbia.edu/~kar/4995`. The TA will be taking notes of our discussions during class, and those notes will be posted on the web site. Links to other relevant resources will be put on the web page.

You are required to get CS accounts for use in room 486 (the CLIC lab). These accounts can be used in class for demos during presentations, and also for work in the CLIC lab outside of scheduled class hours. All students in this class will have access to the CLIC lab whenever it is open, outside of scheduled class hours. There is a $45 charge to get a CLIC account — account application details will be provided. You can use any computing platforms you want as long as your programs conform to the interface specifications described within each project.

The number of times that a single student may contribute during one class is limited to three, in order to allow for as many participants as possible.

Background Reading

A more advanced version of this course (aimed only at PhD students) was given at Stanford University by Don Knuth. Your instructor was a TA for one of those courses. The course documents can be found at `http://www.cs.columbia.edu/~kar/pubsk/seminar.html`. Reading the notes from that class will give you an idea of how the course works.

Tentative Timetable

Jan 20 Class introduction and overview (video). Students submit questionnaires.

Jan 23 Announcement of selected students on the class web page.

Jan 25 Discussion of all 5 project problems. Formation of teams for project 1. HW for Jan 27: Read thoroughly the description of project 1.

Jan 27, Feb 1, 3, 8 Project 1 discussion.

Feb 10 Project 1 report presentations and submission. Formation of teams for project 2. HW for Feb 15: Read thoroughly the description of project 2.

Feb 15, 17, 22, 24 Project 2 discussion.

Mar 1 Project 2 report presentations and submission. Formation of teams for project 3. HW for Mar 3: Read thoroughly the description of project 3.

Mar 3, 8, 10, 22 Project 3 discussion.

Mar 24 Project 3 report presentations and submission. Formation of teams for project 4. HW for Mar 29: Read thoroughly the description of project 4.

Mar 29, 31, Apr 5, 7 Project 4 discussion.

Apr 12 Project 4 report presentations and submission. Formation of teams for project 5. HW for Apr 14: Read thoroughly the description of project 5.

Apr 14, 19, 21, 26 Project 5 discussion.

Apr 28 Project 5 report presentations and submission.

May 3 Course wrapup.

Registering for the Course

This course has a limited enrollment of 40. This is a hard limit, due to the size of the room, and the need for consistent close interaction during the course. As a result, it is possible that there may be more people wanting to do the course than we are able to accomodate. To sign up for this class you **must** have a signature from the instructor. *You cannot sign up for the class in advance.*

In the first class, students will be asked to fill out a questionnaire. Based on the answers supplied, the instructor will select 40 students, to be announced before the second class on the class web page. *You must attend the first class to submit the questionnaire. Late questionnaire submissions will receive lower priority for class admission.* The broad criteria that will be used to select students include:

- Selecting a balanced class, including undergraduate seniors, MS students, and PhD students.
- Selecting students who indicate a strong commitment to the class.
- Selecting students who are willing and able to participate in class.
- Selecting students with a broad preparation in CS courses.
- Selecting students with good grades in relevant courses.
- Selecting students who would benefit the most from the course.

1.1.1 Project 1: Toetjes III

This problem was suggested by Sape Mullender, who described it as follows:

> In Amsterdam, where I grew up, dessert is usually referred to as "toetje" (Dutch for "afters"). The problem of allocating a left-over toetje to one of the children in my family became the Toetjes Problem. The algorithm was the following: First my mother would choose a secret number between one and a hundred. Then the children, in turn, youngest to oldest, could try to guess the number. After the last guess my mother would tell whose guess was closest to her secret number and the winner would get the toetje.
>
> It quickly became clear to us kids that a clever strategy for choosing the number would help to increase one's chances of winning. In a family of two children, for instance, the first child would have to choose the middle number, 50, and the other child could then choose either 51 or 49. Years later, when our reasoning skills were more developed we could even do the optimal choice for three kids. (Naturally, we assumed that each child would attempt to maximize his or her chances without resorting to conspiracy with one of the others.) The first child had to choose 25 (or 75), the second 75 (or 25), and the third could choose any number between 26 and 74, influencing the other two kids' chances, but not his or her own.
>
> Now that I have a degree in mathematics, the problem still puzzles me: Given that the secret number is chosen randomly from a fixed interval what is the optimal strategy for choosing a number for the i,th child in a family of n children? The i,th child knows what the first $i-1$ children chose, and knows that all the children choose optimally (*i.e.*, choose to maximize their own chance without consideration for the chances of any other child in particular).
>
> I grew up in a family of five children, and I never worked out the optimal strategy for $n = 5$. Fortunately, I was the oldest, so choosing optimally was easy for me. But I don't think it mattered very much what I chose; I think my mother cheated. I think she chose the number after we had all announced our guesses, because I can't remember anyone ever winning two times in succession.

CS304 students at Stanford investigated the Toetjes Problem twice; see the 1989 Stanford CS304 technical report at `http://www.cs.columbia.edu/~kar/pubsk/seminar.html` for more details. Unfortunately, the problem turned out to be much harder than it looked at first glance. Rather than analytically try to find the optimal solution to the problem, you are going to write computer programs to play a version of the Toetjes game against programs written by other groups. The goal is to win as many Toetjes as possible.

Our setup is going to be slightly different from the original formulation. Some of these differences may make major differences in game strategy. You will write a program that takes a variable number of command-line arguments. If your program is called toetjes7 (say for group 7) then your program will be called as

```
toetjes7 10000 8 234 3435 9543
```

which means:

- There are 10000 numbers to choose from, ranging from 1 to 10000.
- There are 8 players in this game.
- Three previous players have chosen the numbers 234, 3435, 9543. Thus, you are the fourth player. The previous-move list will be sorted. You cannot assume anything about the order of play of the previous players. Also, you don't know who the players are.

- Your program selects a number in the given range, and writes it to standard output followed by a newline, then terminates.

We calculate the payoff for each program (in this example) as the number of integers in the range 1 to 10000 that are closer to that program's selection than any other program's selection. In case of ties, the program choosing the lower endpoint scores the point.

You can submit up to two programs per group. Once submitted, the program is available for challenges by other groups. More details on challenges later. For this project, your code must execute under Solaris, since we'll run challenges on the machines in the lab.

Programs must terminate! A program that does not terminate within sixty seconds of CPU time will be "unsubmitted" and not eligible for challenges until resubmitted.

At the end of the project we'll run a Toetjes competition on the final versions of the programs.

1.1.2 Project 2: Where can we go from here?

Have you ever tried to read the back pages of airline magazines showing where they fly? Have you ever been bothered by how hard it is to read the maps with a multitude of lines joining many pairs of cities? If so, here's your chance to do something about it.

The purpose of this project is to automatically draw a map that communicates the information about flights in as clear a manner as possible. There are many dimensions to "clarity" and I will not attempt to define the term: There are many ways to achieve clarity. Examples of *nonclarity* include: clutter, ambiguity (eg., lines that merge then separate: which line goes where?), visual strain (eg., so much visually similar information in an area that it's hard to follow a visual object from one region to another), approximation (i.e., displaying only a subset of the information).

The input to your program is two lists. One list contains airports (denoted by their 3-letter codes, eg. SFO or JFK) and their latitude and longitude. The second list contains pairs of airports for which there are flights between those airports. Your program should draw a picture on a computer screen (no bigger than the screen itself) showing the route map. You are free to use whatever graphical software you can find, as long as it is freely available and runs on one or more machines in the lab. If you prefer, you can generate an image in a standard image format, and view the image using a viewer or browser.

You are free to render the maps in any way you think communicates the information clearly. You do not have to use the exact coordinates for the airports if you think alternative locations would be better. (Bear in mind, though, that seeing SFO to the east of JFK might be unintuitive to the average viewer.)

Your code will be run on a supplied test data set. At the end of the project, you should be prepared to run your code on a data set that you have not previously seen. Your maps will be judged by a panel of guest experts. You may assume that the dataset has properties typical of real-world flight plans. For example, the set of flights won't be "all pairs of cities."

1.1.3 Project 3: Battlesplat

You've probably played battleship before. The aim of that game is to place your "ships" in a grid, and then try to guess the coordinates of the enemy ships before the enemy guesses yours. A ship consists of a contiguous (straight) sequence of 3, 4, or 5 squares. If you guess a square occupied by an enemy ship, you are told you have a "hit." Once you have a hit, it is relatively easy to pinpoint the orientation and location of the ship, and to sink it by attacking all of its squares.

Attack technology has been somewhat diminished, and it is now only possible to attack a square adjacent (horizontally or vertically, not diagonally) to the square that was previously attacked.

Your task is to write two programs. The first program (the "placer") generates three ships (one of length 3, one of length 4, and one of length 5) and hides them on a grid. The second program (the "guesser") chooses locations to attack, and receives information about whether each attack is successful. A moderating program will be supplied that interacts with both the guesser and placerer. The expected interfaces for the guesser and placer will be supplier later.

The goal of the placer is to place the ships in locations that are hard to find. The goal of the guesser is to destroy all enemy ships with the fewest guesses.

Simple Competition We'll run various groups' placers and guessers against one another to see which algorithms do best. This first competition will be a "simple" competition in that a single game will be played at a time, with neither the guesser nor the placer knowing the identity of the opponent.

Advanced Competition The guesser and placer will be run against one another multiple times within the same execution sequence. It is possible during that time for the guesser to use information about the placer's behavior in prior games to modify its guessing strategy. Similarly, the placer can use information about the guesser's prior behavior to alter its ship placement strategy.

1.1.4 Project 4: Egalitarian Island.

The people of Egalitarian Island are so egalitarian that every resident is allocated a plot of land of equal area. Whenever a new resident arrives, or an old resident leaves, the land is reallocated in equal proportions. The residents don't build on or farm the land, so it's easy for them to switch locations frequently. They also don't care about the shape of their plots. The islanders are a very private group, and build high fences on the borders between their plots of land.

The problem that the islanders face is that they have an extremely limited supply of fencing material. Your challenge is to help them choose plots of land that divide the island into equal-area pieces, while minimizing the total amount of fencing material needed.

More concretely, you are given a (nonoverlapping) polygon that closely approximates the shape of the island. You are also given a number n of residents that need to be allocated plots of land. Your job is to generate a finite set of straight-line segments (representing the plot boundaries) such that the plots so defined are of equal size, and such that the total length of the line segments is as small as possible.

The input consists of a file containing n (an integer), followed by pairs of floating-point numbers denoting vertices of the island polygon on the real plane. So, for example, the file

```
5
1.1 2.2
3.1 0
1.5 -1
-1.5 -1
-2 1.4
0 5
```

denotes a situation with 5 residents on a 6-sided island with the specified vertices. All entries in this file can be separated by any whitespace characters.

The output is a list of line segments, for example

```
1.1 2.2 1.5 -1
0 5 0 0
0 0 3.1 0
...
```

would describe a line segment between $(1.1, 2.2)$ and $(1.5, -1)$, a line segment between $(0, 5)$ and $(0, 0)$, and a line segment between $(0, 0)$ and $(3.1, 0)$. The orientation of line segments is unimportant, so that `0 5 0 0` and `0 0 0 5` are equivalent.

To make the job of the verification program easier, you are required to output a solution with the following characteristics:

- *The line segments describing the outline of the island must be included.* While the amount of fencing does not depend on the perimeter of the island, it does not hurt to add this number to the total line segment length because the perimeter is a constant. (This requirement will help for drawing pictures, and for the requirement below.)
- No pair of line segments may intersect, except at the endpoints of both line segments. Thus the pair `-1 -1 1 1` and `-1 1 1 -1` of line segments should be replaced by four line segments each terminating at $(0, 0)$. Similarly, `-1 0 1 0` and `0 0 0 1` is not allowed; the first of these needs to be replaced by two line segments throught the intermediate point $(0, 0)$. If necessary, you must replace a line segment on the perimeter by a set of shorter line segments. Since the measure we're trying to optimize is total line segment length, and not the number of line segments, this transformation does not change the merit of a solution.

A correct solution will have the following properties:

- All line segments are within the island.
- No endpoint of a line segment is isolated. In other words, for every endpoint of a line segment, there must be another line segment with exactly the same endpoint. Make sure that you don't generate problems due to rounding where one endpoint is $(2.6666667, 0)$ and the other is $(2.6666666, 0)$. We will test for floating-point equality.
- The areas defined by the line segments are each equal to $1/n$ times the total area of the island. We'll perform this test to within some small value ϵ to allow for rounding errors.

1.1.5 Project 5: Organisms

Consider an electronic world consisting of an m by n grid. Virtual "organisms" can exist on this grid, with an organism able to occupy a cell on the grid. Organisms have *energy* that can be gained or lost in a variety of ways. When an organism runs out of energy it dies, and vacates the cell it formerly occupied. An organism may do one of several things during a virtual time cycle:

- Move one cell horizontally or vertically in any direction. The world wraps, so that an organism travelling off the right edge of the grid appears on the left edge, and similarly for the top and bottom edges. A move uses some energy.
- Stay put and do nothing. This move uses no energy.
- Attack a neighboring organism. Assuming that the cell being attacked is occupied, the attacker gains some energy, but the organism being attacked loses more than the attacker gains. Thus, there is a net loss of energy in this interaction.
- Support a neighboring organism. This operation gains no energy for the supporter, but the organism being supported gains some energy. *This is the only way that a net gain of energy can occur.*
- Reproduce. An organism can split in two, placing a replica of itself on an adjacent square. Each resultant organism has slightly less than half of the initial energy of the original organism since reproduction costs some energy.

An organism's "brain" is a program that you will write. The programming language is relatively limited. The brain of an organism is a list of instructions, where an instruction consists of a condition and an action. Each instruction in the list is examined in turn until an instruction is found that has its condition satisfied. The organism then executes the action of that instruction as its action for that time cycle.

For example, consider the following pseudo-code for an organism's brain:

	Condition	Action
1	My energy is higher than 100 units and the cell to my north is vacant.	Reproduce, putting my descendent in the cell to my north.
2	The cell to my west is occupied, and a random bit is set to 1.	Support the organism to the west.
3	I am in an "explore" state, and the cell to my east is vacant.	Move east.
4	All my neighboring cells are occupied and I'm in an "explore" state.	Change to a "stay put" state and support the organism to my south.
5	I was supported from the west on the previous move, and the cell to the west is occupied.	Support the organism to the west.

There are several important aspects to this example. The instructions are checked one by one, from 1 to 5. If instruction 1 does not match, but instruction 2 does, then the action for instruction 2 is executed, and instructions 3, 4 and 5 are not consulted. If none of the instructions match, then the organism does nothing.

A condition may involve a number of parameters. An organism can sense its immediate neighboring cells to detect whether they are occupied. (A move or split can only succeed if the cell in the appropriate direction is vacant; if it's occupied then the move defaults to a "do nothing" operation.) Conditions can check a random number. This random number is generated once per cycle; the same number is used in all instructions on a single cycle. Thus you can have one check for "random bit set" followed by another check for "random bit unset" and expect one of the two checks to hold.

Organisms can have internal states. These states are the only way that the organism can remember long-term history. In the example above, the programmer has chosen an "explore" state, presumably indicating that an organism in this state should be encouraged to travel. Another instruction in this example shows a situation where the state changes from "explore" to "stay put" in response to overcrowding.

Organisms cannot identify their neighbors. Neighbors may be of the same species (i.e., have the same programmed brain), or of a different species (i.e., have a different brain). The only input an organism gets about its neighbors is whether it was attacked or supported (or neither) from a given direction on the previous move.

Organisms act one-at-a-time from top-left to bottom-right, row by row. We number the top-left cell as (1,1) and the bottom-right cell as (m, n). That means that the state of the virtual world seen by an organism at (x, y) reflects the situation in which all organisms in positions (x', y') lexicographically less than (x, y) have already made their moves, while organisms in positions lexicographically after (x, y) have not yet moved. This convention allows all operations to happen without any need for resolving conflicts between organisms (for example trying to move to the same cell). However, it leads to some slightly unintuitive effects:

- An organism may be attacked from the west even though the state of the world after the move cycle shows no organism to the west. For example, if there are organisms at (3,3) and (3,4), then the organism at (3,3) might attack the organism at (3,4), followed by the organism at (3,4) moving to (4,4).
- An organism may be attacked (or supported) twice from the same direction, or even simultaneously attacked and supported. Continuing the previous example, if there was another organism at (4,3), that organism would see the newly moved organism, and would have an opportunity to attack or support it. For similar reasons, an organism moving north (i.e., up) cannot be attacked (or supported) from the east. (There's an exception to this observation: what is it?)

Organisms are placed randomly on the grid, and don't know their coordinates.

We'll provide an organism "simulator" that reads in one or more organism brains, places one organism for each such brain randomly on the grid, and lets the organisms behave according to their brains' instructions.

There are several goals for this project:

1. Your organism should be able to survive and replicate in an environment where it is the only kind of organism present. Some relevant metrics might be (a) the fraction of the grid that (after some large number of moves) becomes occupied with copies of the organism; (b) the number of moves taken to achieve a certain coverage (say 25%) of the grid.
2. Your organism will be tested in environments containing other organisms. Can your organisms populate the grid faster than their competitors? How might you program your organisms to "recognize" different organisms based on their behavior? If you can distinguish members of your species, how might you behave differently to members of another species? Your goal here is not necessarily to obliterate other species, but to maximize the fraction of the population containing your brain.
3. In the discussion so far, we haven't limited the size of organisms' brains. How might your programs change if (to simulate an organism's limited brain size) we limited the length of the brain code? Another way to look at this question is "How critical is brain size for reproductive success?" Would a 1000-line brain have a substantial advantage over a 250-line brain, or would the difference be marginal?

We'll run several simulations for each of these scenarios to see how the organisms behave. We'll also have some special categories of simulation, such as "best 10-line brain", "best 50-line brain" etc.

The simulator, together with the precise programming language for organism brains, will be provided later (see Section 8.1).

1.2 Participants

The following people have registered for the course. Their identifying intials (as used in the class notes) are also given.

Initials	Name	email (@columbia.edu unless specified)
JA	Johan Andersen	johan
PC	Phil Capetanopoulos	pjc20
GC	Gaurav Chowdhary	gc122
KC	Kristian Concepcion	kjc9@cs
GE	Gregory Estren	gregory
MG	Mukul (Micky) Goyal	mg282
PG	Philip Gross	png3
PJ	Patrick Johnson	plj9
DK	Dave Kalina	dave
AK	Anastio Kesoglidis	aek19
PM	Pritesh Motipara	pvm8
LO	Lawrence Ogrodnek	ljo5
JP	Janak Parekh	jjp32@cs
BES	Brian Shicoff	shicoff
RS	Ravi Shah	ras72
JCS	Jeff Sherwin	commando@cs
GS	Gavin Snyder	gks4
JHS	Jonathan Spiegel	jhs49
MS	Min Suk Song	mss43
PW	Peter Wang	pw51
KR	Ken Ross	kar@cs
SS	Simon Shamoun	shamoun@cs

1.3 Group Membership for Projects

The following assignments were made during the course of the class. Groupings were random, except that people who had previously worked together were (as far as possible) not grouped together for subsequent projects.

Project 1

1. GC, PJ, PW, MG
2. MS, PM, JA, RS
3. DK, GE, BES, GS
4. PG, JHS, JCS, LO
5. AK, JP, PC, KC

Project 2

1. GC, MS, DK, PG
2. AK, PJ, PM, GE
3. JHS, JP, PW, JA
4. BES, JCS, PC, MG
5. RS, GS, LO, KC

Project 3

1. PW, PG, KC, BES
2. JA, MG, DK
3. JP, MS, PM
4. JHS, GS, AK
5. JCS, GE, RS
6. PC, LO, PJ, GC

Project 4

1. PW, PC, MS, GS
2. PG, PJ, RS
3. KC, GC, GE
4. BES, LO, JA
5. MG, PM, JHS
6. DK, JP, JCS, AK

Project 5

1. JA, PC, GC
2. JP, PJ, BES
3. PM, JCS, KC, GS
4. MG, PG, LO, AK
5. MS, JHS, GE
6. DK, PW, RS

2 Toetjes

2.1 Monday January 25, 1999

KR was taking attendance as SS walked into class. All students were present. KR seems to be feeling better. Welcome back.

Initials of class members are used for semi-anonymity. BS requested to be referred to as BES. Therefore, BES is the equivalent of BS.

KR began the class by briefly reviewing the game of Toetjes and then played a game on the marker board. There were five players who had to guess in the range of 1-100. These were their guesses along with their likelihood of winning:

```
GF    73     1/100
KC    72    23/100
JA    26    26/100
PG    74    27/100
BES   27    23/100
```

The correct answer was 87 so PG would have won. The class then proceeded into a discussion of how to handle ties. The problem description states that a number exactly in between two guesses goes to the lower guesser. However, KR decided that it may be better to split it in half. JCS suggested we stick with halves because in the theory of probability, half the time one wins and half the time another does.

Another sample game was played, this time with two players:

```
JS    50    50/100
PC    51    50/100
```

There could not be a more optimal strategy for either player. If PC chose anything higher than 51, then he would have to split the intermediate numbers with JS. If he chose 49, then his probability would have been 49/100. Anything lower would have reduced that number. Likewise for JS, and if JS chose 51 and PC chose 50, it would have been an equivalent game.

This differs from the original problem at Stanford where players chose on the real number scale from $0-1$. The first guess is optimally at 0.5. However, any other number, short of the number infinitesimally close to 0.5, would leave an interval to be divided, thus leaving the second player with a lower score. Therefore, the problem was slightly different. PJ also noted the difficulty of truly choosing a random number on that scale, with limitations on computer precision. KR left the issue as problematic. KR suggested that this case of real intervals takes on significance for us when dealing with a large integer range, like $1 - 1000000$.

The class returned to the issue of ties. PC noted that a distinction can me made that when there are two players who guess 49 and 51, and the correct answer is 50, halves would give each an equal probability while going to the lower guesser favors that player. SS pointed out that if this was a real world simulation, their would be no ties. PW suggested that tie goes to the first guesser. The decision was to stick with splitting the score. The class proceeded into questions and answers about the game. There are no hidden guesses. That is, each player knows all the previous guesses. There is no knowledge of who the opponents are, so guessing will not be based on history. Each player will receive guesses in sorted order to reinforce the no-history rule. Finally, PC asked if we can assume that the other players will make the best possible choice, which led into the important part of the class discussion, how to choose.

First discussed was how should the last player play. One answer was to choose the middle integer in the largest interval. JS suggested taking the first integer above the starting number of the largest interval. For example, for two consecutive choices X and Y, where $Y - X$ is the largest interval among the guesses, one suggestion is to choose $X + (Y - X)/2$. Another is to choose $X + 1$. Any choice in that interval will yield the same results.

KR also suggested whole endpoints. That is, in a range of 1-100, if the largest guess was 79, then the last player can choose 80 and gain all number from 80-100. If this interval is larger than half of any mid intervals, then this is the optimal choice.

```
|-------|-----------------|---------------|
A   x   B        y        C       z       D
```

The optimal choice in this case would be any guess in between B and C, unless $z > y/2$. In which case $C + 1$ is optimal. The formula is as follows:

score=max(size of largest $interval/2$, size of largest end interval)

While an optimal solution can be determined for 2 or 3 players, there is no general solution for n players. If all players are random, then the first player does best by choosing in the middle. For the sake of discussion, we wil assume that all players choose optimally.

In response to a question by PG, KR stated that having an "unknown" number of players would a very different (and probably harder) problem. However, we will stick to our problem of a known number of players.

Players will be rated by their performance in different positions. So if there are ten players, then each player will be rated for its average payoff in position 1, 2, 3, etc.

The groups for this project are:

1	2	3	4	5
GC	MS	DK	PG	AK
PJ	PM	GE	JHS	PK
PW	JA	BES	JCS	PC
			LO	KC

2.2 Wednesday January 27, 1999

KR began the discussion with the question: What is the optimal solution for 3 players? There was was much discussion in class, especially over the calculations, and this was one proposed answer:

```
Player 1 (P1): 25
Player 2 (P2): 76
Player 3 (P3): 50 (or any other number between 26 and 75)
|--------------------------------|
1      25              76       100
  24/100  25/100       24/100

The payoff for each player would be:

P1: 24/100 + (avg (P3-25+1)/2 where 26<=p3<=75)

Calculated as follows:

75              50
avg (i-25+1)/2 = avg (i+1)/2
i=26            i=1
   50
= Summa (i+1)/100
   i=1
=(1/100)(51(51+1)/2-1)

P1: 3/8 (this is an approximation)
P2: 3/8
P3: 1/4
```

KR repeated the calculations offline and came up with the following corrections to the calculations:

```
Player 1 payoff = Player 2 payoff = 37.25/100 and
Player 3 payoff = 25.5/100.
```

The error in the original calculation was that $(i + 1)/2$ should have been $(i - 1)/2$.

If P2 chose 75, P3 would choose 76 (since he would get less than 1/4 in the interval) yielding 1/4, P2 would get $< 1/4$, and P1 would receive $> 3/8$. If P2 chose lower than 75, the payoff would be even worse.

Likewise, if P2 chose greater than 76, P3 would gain more from the middle interval than before and P2 would have a lower payoff. Therefore, P2, and P3 played optimally. The question of whether or not P1 played optimally led to the following discussion:

PG asked if we could do a state space search of P1's choices. However, that involves calculating the guesses of P2 and P3 as well in an optimal game. KR proposed that resulting payoff function on P1's guesses would be piecewise linear, meaning there will be breaks and jumps at different points of the graph. He proposed that one can take select points in the graph, and if there is a suggestion that several points are part of one "piece," one can do a "binary search" outwards to find the endpoints of that piece. The optimal solution would be at one of these endpoints.

JCS stated his "magic number conjecture," that there a number among a small set of guesses that is the optimal solution. The numbers in this set would correspond to the endpoints of KR's pieces. All can find proofs for or arguments against the JCS "magic number conjecture."

JCS stated a second conjecture, that the first and second players get the same payoff.

MG suggested to divide the interval by four (for 3 player game) and consider the points at those intervals.

JCS brought up the issue of random guesses and unanticipated guesses again. How a program performs depends on the aims of its competitors. Optimal guesses may not be optimal against random players.

There are then two types of solutions:

```
C1: Attempting to be optimal.
C2: C1 union everything else. For example, always pick 17.
```

Teams in this class can devise players in C1 and C2. Players in will only play other players in this class. In this course, we are striving towards C1.

GE requested a definition of optimality. For the last player, it would be the maximum payoff given the previous choices. For player i: given previous choices, assuming the next players follow optimal strategy, optimize current strategy. This definition is inductive.

MS asked if self playing is optimal. KR said that even if a player played optimally against itself, that is not a proof that it plays optimally against all other players.

GS offered an alternative definition of optimality. PW offered a suggestion for what optimal for P1.

PC said to look backward in finding the optimal strategy. What would be best for P3? See strategically, not numerically.

We recapped the optimal strategy for last player:

- Either anything in largest interval (L), one less than lowest choice(L1), or one more than highest choice (L2).
- $max(L/2, L1, L2)$
- If all three are equal, then choose randomly between all three.

Second to last player:

SS said to assume half of L goes to the last player.

JP said that the last player will go as far possible from the previous choice.

GS said we don't want to pick right next to anyone due to risk of being sandwiched.

MS suggested we use brute force: try all choices and see what the P3's choice will be. While that is a valid strategy, it does not offer insight into general strategy.

LO said that endpoints seem nice because they are never split by players. BES said that to assume the best score for any player your optimal is k/n, and work inductively to the number aiming for that score.

KR says k/n contradicted by 3 player example.

KR, as a meta-comment, suggested that if a class member has a different approach in mind, rather than sitting quietly and saying nothing while other approaches are discussed at length, try to find an opportunity to bring up your new idea and steer the discussion along a different path.

JCS then proposed his "1st and 2nd players conjecture," that the first two players get the same payoff.

RS suggested the "decreasing payoff conjecture," the subsequent players have decreasing payoffs.

PG offered some intuition that might be extendible to a proof. If a later player chose optimally and had a higher payoff, you did not choose the proper number.

PJ resurrected BES's earlier suggestion in a modified form: maybe k/n is a good starting point to find lower and upper bounds on the payoff for any given player. KR added that perhaps it's easier to find these bounds than to find exact optimal payoffs, and that the bounds could be used to prune a search.

2.3 Monday February 1, 1999

The class began with a discussion on how to run tournaments. The users will specify the number of games, the number of players, and the player pool. Players will be chosen at random from that pool for each game. The class was divided on how to handle a program that did not work properly. The class decided to figure out how to handle it if the issue comes up.

The class the returned to the previous week's discussion, from which there remained two open questions:

1. Did player 1 play optimally in our 3-player game by choosing 25?
2. What is optimal strategy for the second-to-last player?

GE went up to the computer to sketch a proposed answer to Q1.

1. P1 wants to choose a number that P2 and P3 will choose on one side of it.
2. P2 wants the maximum such number.
3. P2 wants smallest # that P3 will go to its left, assuming P2 to the right of P1.

This suggests an optimal strategy for all players, which P1 followed.

BES went on to Q2. He stated that if the opponents choose optimally, then there is a strategy to be followed. If not, then optimal is not well defined for P2.

DK said to take the largest interval L and the end intervals $L1$ and $L2$. If $L/2 > L1, L2$ choose in L, else if $L1 > L2 > L/2$ then choose $L2$. KC noted an important point, that players don't want opponents to take their endpoints.

A sample game was then played with the following intervals on the board. The purpose was to find a general strategy.

```
L1 16     L 157      L2 25
```

JP said that no matter what the second to last player does, the last player will play in L for the largest payoff. MS said that if the second longest interval L'=146 say, the last player will play in L' because that will yield the highest payoff. The class's intuition for 2nd to last player was that he doesn't need to consider more than the 2 largest intervals and end intervals. How does the second to last player P2' play? SS said in L. Another qualified by saying the middle of the interval. This reduces the amount that P1' (the last player) will take from P2's "domain". PG said that P2' should play anywhere in L that results in ranges less than L'.

```
Payoffs:
P2': 157/2
P1': 146/2
```

Is there a situation where P2' would pick in L'? JA said that P2' does not play optimally by choosing in L'. KC tried to provide an example where it would be optimal to do so. He tried to find a situation where, if P2' chose in L, it will be optimal for P1' to pick in a resulting subrange.

```
   e.g. L1 6   L 55   L2 7  L' 25
P2' Plays the middle of L, leaving 27 on each side
P1' then chooses in L': 25/2
If P1' chooses in L, payoff(P1')=27/2 and payoff(P2')=27/4+27/2=20-1/4 > 25/2.
```

The example was inadequate and the suggestion was left unproven.

KR then gave the following scenario. P3' (the third-to-last player) knows that P2' has 5 choices with different payoffs, so P3's payoff is the average of those choices and should aim for that payoff as the optimal solution. However, if P3' could sense that P2' will play suboptimally, then P3' could choose more wisely.

PC then reminded the class that since the successors are unknown, P3' has no idea how they will guess. If the nature of the algorithms was known, then there could be a basis for choosing more wisely among those options.

PW stated that we should assume that all players play optimally. PM reinforced PC's statement. P2' should play in L1, L2, L.

PJ asked how does one predict what the next players will choose, if you have several choices and they do?

KR then set the goals for Wednesday: Every group has to have one program submitted. That program would not havto play especially well, but having a submission for each group would get the ball rolling for code development.

The class discussed the issue of open source code. The pro of open source is that teams can get ideas from each other. There is the risk that some teams might copy other's code. JHS was also concerned that one might look into another's code for weakness to better tweak his program, which doesn't necessarily lean to an "optimal" solution for all games. KR suggested that looking into weaknesses could be a way of analyzing strategy, and improving the quality of the program. The final vote was to keep the source code open, and available in the same directory as the executable.

2.4 Wednesday February 3, 1999

KR suggested that all students should read the Stanford tech report of the 1989 problem solving class. There is some discussion of toetjes there that is relevant and useful for this project.

KR demonstrated a toetjes program he wrote based on the "Stanford conjecture." The Stanford conjecture is to choose a number at one end of the interval to make it appealing for other players to choose away from you. The critical point is how long that interval needs to be. Too small and you don't get a big enough payoff. Too large and somebody else will be tempted to play in the interval.

According to the Stanford conjecture, the following is how to choose good (optimal?) moves an n player game with range up to m.

([] means round down)

```
P1: [m/(2n-2)] from one end
P2: [m/(2n-2)] from the other end
P3: 2[m/(2n-2)] from one endpoint of longest region
```

PW had his own conjecture on determining the optimal payoffs. The scores in the 3 player game were 3/8, 3/8, and 1/4. In a similar fashion, he said to divde the interval by 2 to the nth power and assign each player as equal a portion as possible. For a three player game, 8 portions are diviedup with three people. KR's intuition, in response, is that the payoff distribution will not be equal, and the generalization won't hold.

GS noted that it seems that your payoff depends on choices made before and after. Your choices will be compromised if other players don't play the way you expect. The fact that other players won't be optimal is a factor in your strategy.

JHS said to look at two edges, neither is good, or one is good. Various cases ensued depending on the relative sizes.

We then returned to the Stanford strategy. KR provided an example on the board. If you choose at an edge, you want to choose an interval that gives you a large space, but not so large that someone will snatch it.

```
|---------|-----------------------------------|
m/(2n-2)     (2n-3)m/(2n-2)
```

You will end up with intervals that are at least twice as large as your end, so players will have incentive to choose outside of your end. We are using pigeonhole reasoning, that is, one of the gaps beyond your choice will be larger and following players will be more likely to choose in one of those gaps. The idea behind [m/(2n-2)] is that it leaves (2n-3)m/(2n-2) for the remaining n-1 players, and if they divide it close to evenly, the non-end intervals would be of size at least 2[m/(2n-2)], meaning that the intermediate intervals are more attractive than player 1's ned interval. If the subsequent players did not divide the remaining space evenly, then some of the intervals would be even larger, and even more tempting than player 1's end interval. Similar reasoning can be applied to subsequent players.

PS said that we haven't proven that we have maximized our payoff. We've only guaranteed a certain payoff for the end interval.

JCS provided the following situation for the third-to-last player.

```
  L 200 (this is the largest interval)
  L' 180 (this is the second-largest interval)
  Suppose the end intervals are both small.
P3' (third to last) plays in L, P2' plays in L', and P1' plays in remainder
of L. Payoffs are:
  P3': 100/2+50/2 =75
  P2': 90/2+90/2  =90
  P1': 100/2      =50
OR P3' plays in L', P2' plays in L, and P1' plays in L. Payoffs are:
  P3': 90/2+90/2  =90
  P2': 100/2+50/2 =75
  P1': 100/2      =50
```

It was to P3' advantage to choose in L', based on the play of P2' and P1'. This solution is more optimal than the Stanford strategy. It also demonstrates a case where playing in the largest interval is not optimal.

According to the Stanford strategy, P1 and P2 choose close to the ends, and subsequent players move inwards. The only randomness is whether to move inwards from the left or right side.

GS says that there is a significant chance that other players won't choose in your end zone, so it may be safe to choose a little further inwards. JA says that if you did so, P3 might have found it more tempting to choose in your end interval (since his estimated payoff is lowerer than P1's zone).

The class then broke of into discussion about the actual programs and policy.

PG said that 2 days notice for a program submission was a bit tight. KR asserted that deadlines are needed to make progress, and this deadline did not require great precision. JCS said that it is not an issue of work load, so much, but the fact that it is a group project that makes the time constraint difficult. JCS says that one issue in the project is that not everyone lives on campus, so coordinating can be difficult. Once KR reminded us that programs must be complete by Monday and presentations by Wednesday, JCS emitted an expicitive. The benefit of having the program done in advance is that the writeup will be more complete. Most of the class thought that the requirement was reasonable, given that no strict criteria were imposed on the quality of the guessers submitted.

Format of the reports and demonstrations:

Demos are explanation of achievements. The report describes what methods were tried, what worked, didn't work, and why not. Proofs of any results, simulations, results of competitions, and graphs may be provided. Teams should make arguments about optimality. The report should be a coherent paper that can be understood by a professor who is familiar with the problem only, and not familiar with the class discussion or the background reading. The background of the discussions should be included in an introduction, and all contributions should be acknowledged. The report should focus on what you have contributed.

The presentation can be run as you like. The team determines who presents. KR prefers that more members participate in the presentation. It shoud be ten(10) minutes or less.

JP asked how much work needs to be done in research and references. KR said that some is nice, and that there are a few documents related to this problem. No prescribed length of the reports. They must be in hardcopy (that is, printed). Recap: by end of Monday, the submitted code will be used for the tournaments. The source of the code does not need to be in the report, but there should be a pointed to the code for review.

Back to the problem.

JP suggested that we can add enhancements, not stick to the exact formal points provided by Stanford. GE still has a concern that not everyone will be playing optimally, and so you may suffer because they will throw off your strategy.

PG suggested that we have a random player and volunteer to write up such a program. JP already implemented that in their program. Please post the name to the newsgroup.

PC said that for the second to last, it is optimal to choose in the largest interval (modulo some details about the end intervals). AK thought he might be able to find a counter example to PC's statement. RS said that he tried hard but couldn't find a counterexample to PC's situation.

KC worked out a proof of the optimal strategy for the second-to-last player. He broke it down into different cases

1. $L > L1, L2$
2. $L < L1, L2$
3. $L = L1, L2$

etc.

See his proof online `~kjc9/proof.html`, and if there is a hole, notify him.

MK suggests a 3 player game.

```
P1 25
P2 76
P3 L
```

P1 may not have played optimally if P3 chooses close to him. If there is a degree of arbitrariness, then that will bias some programs. For example, even though P3 could play anywhere, his logic might bias him to always choose 26 in this situation. If P1 had known that, he would have chosen 76 instead of 25 for his first move. JP suggested that you can add an element of learning. If you know the patterns of the players, you can guess what they play and adjust your moves accordingly. JCS countered this by stating that if you don't know that you are playing the same person each time, your training set is invalid. KR observed that you might try to identify players by noticing numerical quirks in their guesses. Other class members countered that if programs knew of this behavior, they could try to use it to their own advantage.

By Monday, make sure the players and links are current. After Monday's class we'll run various toetjes tournaments. If you detect a problem with another player, inform them. Maintain the source code in SS's directory (`~shamoun/4995/tootje/src`).

2.5 Monday February 8, 1999

The teams reported on the performance of their programs, which ones passed the stress test, and their progress. Group 5 submitted two programs. Since the problem was so complex, JHS said they simplified their algorithms and cut out the AI. The Stanford technique appeared to be very good.

JCS spoke about the "Riverfront property," in which the end points are most desirable. The remaining intervals will be divided into k segments where there are k players left.

Group 4 searched for optimal games to analyze. JHS said that his group uses a brute force procedure for guessing optimal moves. This method was feasible for games of up to four players. However, it was realized during discussion of JHS's brute force search that it selected just one of the possibly many optimal moves to play at any given move. This might be biased. For example, suppose that the algorithm systematically chose the smallest among all optimal moves. While this doesn't seem to bias the current player's payoff, it does bias earlier players payoffs. A move that was good for player 1 for most of player 2's optimal choices may seem bad when evaluated against just one of those choices.

Group 4 also admitted to allocating an array of size equal to the range of the game. (No other group had done so.) This allocation caused problems on the stresstest because there was an example with a very large range that caused the memory allocation to fail.

BES said that they maintain constants like the proportion of competitors using the Stanford algorithm. By keeping logs of previous games played, they can tune the values of these parameters. For example, if they see moves that could potentially be made by a Stanford algorithm (e.g., a move just the right distance from the edge) they surmised the presence of a Stanford-like competitor. When the tournaments are run, care should be taken that the internal logs due not overload their directories.

The class then set the ground rules for tournaments, which will be run shortly after class. The goal is to have them all finished by 1 PM on Tuesday. The results will be published and can be incorporated into the reports.

The tournaments will be broken down into different categories. On January 27, it was stated that there will be two broad categories: C1, all programs attempting to find an optimal solution, and C2, all solutions, including random guessers.

These are the programs that fall into C1:

```
team  program
---------------
1     team1
2     group2
2     group2a
3     g3
3     brian
3     greg
4     killer
4     team4
5     pajk
5     pajk2
      stanford
```

The are the programs that fall into C2: random

The following tournaments will be run, where n=number of players, m=range. The the pool of players are those in C1:

```
n    m   m
------------
3   100 103
4    ..  ..
5
..
10  100 103
50  100 103
99  100 103
```

PJ objected to the 99/100 because he did not believe it will reveal much. However, the results are not obvious and it is worth testing.

Tournaments will also be run for a range of one hundred million, for the same n as above, plus n=1000 and n=673.

The following suggestions were made about how to handle the data when there is a crash:

1. Accept the sub tournament if the number of games played is large enough (>1000)
2. Separate them into a different set of tournaments
3. Eliminate programs with errors and re-run the tournaments
4. Exclude a program that bombs in a tournament from that one only

Suggestions 1 and 4 were accepted.

Pairwise comparisons will also be made. JP observed that a program can be its own worst enemy.

There are 55 pairwise tournaments which will be run for n=11 and m=100. The following method of ranking the tournaments was proposed:

1. Rank each player in a position for each tournament by payoff
2. Add the ranks for each player

Player scores may be determined for the following:

1. Position and number of players.
2. Position in all games
3. Overall score

To find best player, average all those scores. To find best player at a particular position, average all the scores at that position.

Presentations are due on Wednesday.

2.6 Wednesday, February 10, 1999–Presentations

Today was an exciting day for the class, because each group presented their projects, explaining the methods they used for guessing, and the performance of their programs.

Group 1 was the first to present. PJ stated the how the formula that players used to guess. The first player P1 chose the position m/(2n-2), where m is the range and n is the number of players. The second player chose the point the same distance from the other endpoint. The third player P3 chose twice the first players guess from one end of the interval, and so on for the first n/2 players. PW stated that in addition to using the strategy described in class for the second-to-last player, they also analyze the end intervals and choose in one or the other such that the last player would not choose in that interval. In the example he provided, L=76, L1=38, L2=29, so the second-to-last player should choose 28, which leaves a smaller interval than L2. MG said they wanted to refute the conjecture playoffs are inverse proportional to player position, so their player would choose numbers in a way that reduces the scores of previous players. Tests confirmed his anti-increasing payoffs method worked. PJ described a more conservative strategy for the middle players. When there are k players left, then choose the kth largest interval and guarantee that interval to that player. MG said that the performance was good in the beginning and end cases. The program did not play so well in the third to last position because it did not first check the end intervals before choosing in one of the largest intervals. Unfortunately, group 1's tournament submission had a bug (some diagnostic printf statements were not removed) that meant it was disqualified for all games with more than 4 players. Nevertheless, group 1 was proud of their code, and thought that it would have done well if it wasn't for the bug.

Each member of group 2 presented their proposed solutions. Several approaches were independently pursued. MS said they tried to use the brute force method. After P1 chooses, P2 has 99 choices, P3 has 98 choices, and so on. The number of games for 5 players in a range of 100 is 100 choose 5, which is over 9 billion. The process is long and tedious but easy to code. They were able to prune the code to choose among three categories: the largest interval and the two end intervals, giving a branch factor of 3 and 125 [sic] games to choose from. The first process is slow but accurate. The second method is not guaranteed success at finding the optimal solution. This method was not successfully implemented.

RS used an alternative to the Stanford report. He agreed that if the ends were chosen properly, they can be ruled out. He said to then create an interval that would be slightly smaller than the end. He said it worked better for smaller games. His implementation did not pass the stress test.

JA suggested taking a recursive view of the problem, making each choice a repetition of the first choice over a smaller interval, amking it very easy to code and O(1) time complexity. This was also a midification of the Stanfard algorithm, except that it divided the largest interval by the remaining number of players, and not by n. This implementation was called "group2" and came in 3rd or 4th on average. The disadvantage of this algorithm was that it threw away previous moves.

PM presented their final attempt at an optimal player. He felt that the Stanford algorithm was sufficient for the first two players, and the strategy for the final two players was clear. The algorithm began by creating a matrix of all the intervals, excluding the endpoints. Then, if the current player chooses in interval i, if it is likely that the next player will choose in i then reduce the payoff accordingly. This was not implemented.

KR helped clarify that JA's algorithm will exlude the possiblity that optimality may be in the the second largest interval.

Group 3 introduced themselves as the Dutch Masters (GE, BES, DK, and GS who was absent). They worked on three algorithms, with "greg" coming in first, "brian" second, and all other algorithms last in the tournaments. They intended to implement a learning strategy to determine how the other players in a tournament were playing based on patterns in the choosing. If there program could determine what algorithms were being used, then it would present counter algorithms.

DK presented the solution from a programming point of view. Since they had to quickly analyze all intervals, they listed them in descending order and used the appropriate algorithm. The "brian" algorithm simply chose the midpoint of the largest interval.

BES spoke about g3 algorithm in GS's stead, which gathered plenty of info about the games and tried to make predictions about the outcomes.

GE presented an algorithm, implement in "greg," that assumed all players were using the Stanford algorithm. It would choose slightly further into the largest interval than Stanford would to siphon off a few points.

Group 3 could not fully implement the learning aspect of the project and had the turn in off during the tournament. That's why they submitted 3 programs, and not one program with three different behaviors.

PG begain Group 4's presentation by describing their work as the least organized development process. They wanted to implement an "AI" approach but were limited due to time constraints. JCS discussed one algorithm. The kth player would divide the total space, excluding the end regions, by k, chose based on the following conjecture about payoffs: $m/k < payoff(x) < (3/2)(m/k)$ The numbers were chosen to stick within those bounds. This algorithm went unimplemented.

JHS tried to analyze some optimal games to find out how to choose optimally. He noticed late in the process that the players closest to the endpoints were always m/(2n-2) from the endpoint. He used this as the lower bounds for an optimal score. He used a conservative algorithm based on Stanford by creating the right length intervals such the following players would choose outside of them.

LO said he was explaining how to be the last runner in a competition (joke). His algorithm tried to take advantage of players that made grossly sub-optimal moves.

Group 5 presented last. JP said they followed two basic principles in their implementation. The first was to play conservatively so they would do decently by the end. The second principle was to use a modular strategy, that uses multiple algorithms that served as fallbacks for each other. If one did not play well, then it would pass the problem to another algorithm.

These were the algorithms. The first two algorithms were KC's ideas.

1. Opening moves: Determine what interval the player deserves, and tries to stake out that interval.

2. n-m: a generalization of the algorithm for the second-to-last player. It would look at the smaller intervals as it moved further back.

3. Attack (AK's idea): The #1 priority was choose towards the ends and reduce the score of player with the best score.

4. Naive: Sort the intervals and choose the one it feels it deserves.

An interesting aspect of their presentation was their analysis of their own programs' chosen strategies used in the tournaments. In their analysis, they discovered that Opening Move ran way too often. Attack was more aggressive in pajk2.

Some ideas they had for the future were to look more closely at the second largest interval and to find a optimal 4 player game.

Some final thoughts of MG on the toetjes problem were that, although the algorithms were not playing optimally, they were beating the stanford algorithm. Their attempts were more at winning, and not at determining the optimal solution.

The class was then divided into groups for project 2 by reading down the columns of the chart of groups for project 1. These are the groups:

GC	MS	DK	PG
AK	PJ	PM	GE
JHS	JP	PW	JA
BES	JCS	PC	MG
RS	GS	LO	KC

3 Where Do We Go from Here?

3.1 Monday February 15, 1999

Before starting the discussion on the maps project, KR said that the presentation quality of some of the projects could have been improved. The documents, as well as the presentations, could have been better structured, with the appropriate headings for each section.

The discussion about project 2, drawing airline routes, began. The goal of the project is to write a program that airlines could use to update their route maps with changes in service, and to improve the quality of these maps. The maps to be drawn for this project are static, like those found in the back of airline magazines. Possibile extensions, such as interactive use over the internet, would be interesting but beyond the scope of this project.

PG asked if the maps will be global or rectangular regions, since the maps in airline magazines vary that way. Both styles can be worked with during the project. The only restriction is that the map has to be displayable on a computer screen and of printable quality. The labels for cities are not critical.

Data sets for this problem are hard to come by on the internet. It will be incumbent on the groups to manually encode the routes of several airlines. SS will also write a program to generate artificial routes. All data sets will have latitude and longitude coordinates of the cities. LO asked about the background for these maps. If it is a known region, then the image can be imported. For artificial data, any background suffices. The distance between two cities does not have to be accurately represented. However, that is a factor in readability. Anyone who knows the map of the U.S. will expect cities to be at (or near) their correct positions. Non-technical judges will help review the quality of these maps.

JCS wanted to know the distinction between the present problem and the research work done on graph drawing. KR acknowledged the point and encouraged the class to look at text books that have been written on graph drawing. The project will focus specifically on route maps. JCS felt that this project would simply involve drawing lines between points on a map. JP noted stations in subway maps are mildly shifted to make them aesthetically pleasing, but are geographically constrained to make sense; geographical constraints are not as important in graph drawing. KR added that there may be better ways of representing a link between a pair of cities than drawing an edge between those cities.

PG brought up the issue of red/green color-blindness. The class spent some time discussing whether this was a sufficiently important issue to consider for this project. Apparently, about five percent of the population cannot distinguish red and green. JCS wanted to know whether the judges that KR was going to choose were red/green color-blind. JA suggested that color-blindness was not at the heart of the problem. It will be up to the groups to decide how much to take this issue into account.

The class then when into a discussion and analysis of the route maps of several major airlines. The URLs can be found in the newsgroup.

America West Airlines: The major hubs in this map are Las Vegas, Phoenix, and Houston. RS said that it is hard to follow links from NY to the west. Although PJ did not like how edges intersect with each other, he liked that the city names do not intersect with the edges. The class spoke about the drawing of the lines. GE felt that Las Vegas and Phoenix are too close, so that it is not obvious which distant cities connect to the hubs. PG pointed out that it is possible to distinguish to which hub flights are destined by how they curve. JP suggested that for marketing, it looks good if there is a lot of activity on the map. According to PC, there are two types of travelers: those that want a direct connection, and those who are willing to connect. This map makes it clear that Phoenix and Las Vegas are the major hubs, so all flights could connect through those cities. KR said that doesn't help in all situations. For example, it is not immediately obvious that a direct connection exists between Columbus and Washington D.C.

The class tried to figure out how to make the map more readable. GC suggested using separate colors for the hubs. RS said that some links can removed, and instead indicate what hubs each city connects to by a letter next to the city. JCS suggested having a channel to a region that splits into specific routes for the cities in that region. He further suggested grouping cities in a region and sending on edge from a hub to the group as a whole, rather than to each city individually.

In the final comments on the AMerica West map, PG pointed out that specific airports in the same city, like NY, have been artificially pushed apart. GS also noted that the arcs are not accurate to flight paths.

Reno Air: This was a map with few connections located primarily in the west. By looking at the map, PG pointed out that, in drawing connections between cities, one has to make the distinction about when a flight goes over a city or stops at a city. If an arc goes too close to a city, it becomes ambiguous whether we have one or two links. Similarly, if two opposite edges out of a city have the same slope, it might be interpreted as one long edge that coincidentally comes close to a city, rather than two edges.

Continental Airlines: JCS suggested coloring regions of the country so that viewers are not distracted by lines in other regions. MS felt that thin lines make it hard to distinguish between colors. The suggestions till now were to draw links between cities or indicate on the cities what connections were available. LO felt that the links are more intuitive and gives a better sense of what connections are available. JP said that a map with links is subconsciously more pleasing.

GS said there appear to be duplicate lines between cities on the Continental map. Perhaps this was deliberate, in the spirit of the "activity" marketing perspective mentioned previously.

American Airlines: Each city indicates that they are "served by" the airline. The presumption is that there is too much information to express on the map.

Midwest Express: JCS immediately suggested that the map is ugly. JP said the straight lines make it less appealing. PJ said he liked it at first, until he realized that Atlanta is too far out west. The class then noted other ways in which cities were not in their proper locations. However, because the lines are straighter, and the map is in black in white, the routes are more distinct. MS liked the boldness and directness of this map. JHS said that map works well only because it is small with few connections.

The American Eagle Code Share map available from the Midwest Express map shows all the cities that are served by American Eagle without even creating links, simply by listing the cities near their hubs. Because American Eagle is regional, there is much more locality in their routes. Tis map also omitted details of many states that didn't contain routes.

These are the goals for next class: Each group will transcribe the route maps of different airlines. There will be a data file with the airport abbreviations and there latitude/longitude coordinates (expressed as decimals). The other file with have the connection pairs. Each pair will be alphabetical. That is, JFK comes before LAX. Data in these files will be tab delimited, with one city or connection per line.

These are the groups and the maps they must transpose:

```
Group 1: Delta (domestic)
Group 2: AA (domestic)
Group 3: TWA (global)
Group 4: Continental (domestic)
Group 5: Reno and American Eagle
```

Finally, KR asked the class to begin thinking about algorithmic aspects for next class.

3.2 Wednesday February 17, 1999

The class reported their (partial) progress in the transcribing of airline routes.

The class began with a discussion about what heuristics can be used for drawing a map. In order to determine the threshold for the number of arcs out of a city, PG suggested drawing a circle around the city and finding out how many arcs are in a given degree span. This gives an idea of what the density of edges is coming from a node. JP said measures of density can be incorporated with "Jeff's Band Vision," suggested by JCS last class, to determine when the band should be used. JCS said we are assuming that the computer knows where to place the cities to start with. Issues to consider for displaying these graphs are: the graph representation, distance of cities, shapes of edges, and placement of nodes.

RS suggested creating a region around each actual city location to delineate the region in which it is considered ok to draw the city. JCS observed that where you place a city affects where others are placed so they stay in their relative locations. This could lead to an iterative process of refining the positions of cities, and termination might be an issue. One reason for moving nodes, according to DK, is if they are too close. For example, one might separate LGA, JFK and EWR so that flights to each could be distinguished. KR offered another situation where moving nodes may be desirable. In order to draw the links between three cities that are lined up, the edges will have to be arced. It may better to shift the cities and draw straight lines.

BES asked if an alternative method to maps can be used for representing the airline routes. An adjacency matrix may work for small maps, but it may not work for large maps. BES suggested an intermediate sytem, like showing the connections between cities without accurately placing them. JP pointed the class to the New Jersey Transit Map (`http://www.nj.com/njtransit/njtmap.gif`) which has a semblence of geography but primarily shows the routes. MS noted, however, an airline map is much different from a train map, because of the many connections, making it less linear.

The class analyzed the route map for Alaska Airlines. Their is no underlying map with political boundaries. KR made a quick observation that a city overrides a link; that is, the name of the city blocks out the links when they occupy a common space. Most of the other maps used the opposite convention. The map contains a lot of spaghetti, (JCS likened it to a fireworks display) which is what makes it an interesting discussion piece. PG said that it is not easy to locate a city if a person does not know where they are relatively located. For example, foreign travellers who don't recognize American geography might be totally lost.

KR suggested that it might be possible to design a special map that biases emphasis towards frequently travelled routes. In the absence of actual passenger volume statistics, one could bias towards routes between cities with larger numbers of connections. JCS suggested drawing something like a ciruit diagram. When two lines cross but don't intersect, then one should skip over the other.

PC had a proposal for "Airline routing using buses," which is based on the circuit model given above. He proposed a rectilinear grid of bands that are drawn like computer buses with smaller edges emanating perpendicularly from the buses.

KR said there is another good application of the "skipping over" idea. When a city falls in a route between two other cities, then it can be "skipped over" instead of curving the arc, so that straighter, more direct edges can be used.

BES suggested that coloring would improve understandability.

PC said that the Alaska Airline map is atypical because the hubs are not so major. 95 percent of Continental's domestic flights go to one of their hubs, so their representation on a route map might be done differently.

The class then set goals for Monday's class. By Monday, there should be a design of the programs. An airport code among the several airports for a city can be arbitrarily assigned to that city if it is unclear what airports are used by that airline. There is no required image format. The programs need to be able to handle an arbitrary set of airports (supplied with their latitute, longitude and city names). The programs should also be general enough to handle arbitrary sets of flight pairs. In particular, the programs should be able to generate international maps. GE was unclear about how to handle world maps. KR said that international airlines typically have separate maps for the different regions with an indication to where they fly. That approach would be acceptable for this project.

Some further criteria:

- Maps should be readable in a browser and printable on a standard printer. The department has a good color printer that is available for use for the final submissions.
- There are no limitations to color.
- Groups can submit two maps if they want to try different approaches.
- Groups will have a day to produce the maps. So the programs had better be able to complete several maps within a 24 hour period.
- The maps have to fit on a computer screen, in particular, the projector screen at the front of the CLIC lab. The desktop on the CLIC overhead computer has a resolution of 1024x768. The images won't be resized, so they should try to make maximum use of the available screen real-estate.
- Providing the city name on the map is optional; airport codes are sufficient.

The class discussed their initial ideas of platforms on which to implement their designs. There was much talk about Java and what classes to use. KC suggested a package called GIM. JA thought of using one program to create the representation, like a Perl script, and one to display it, like TCL/Tk.

In summary, for Monday, the following has to be provided:

1. Design
2. Platform (e.g. drawing software, image formats, etc.)
3. Completed transcriptions, with the file locations posted to the newsgroup.

Implementation is not required by Monday. However, it will help to validate choices. KR would provide a file with latitude and longitude info.

JP suggested the tempting possiblity of a startup based on this project, with a sellable prduct. There was some enthusiasm for a class t-shirt with the winning map on the front.

3.3 Monday February 22, 1999

After reporting on their progress transcribing the route maps, the class discussed what progress they made on the project and what algorithms they thought of implementing.

JP said his group fleshed out what rules to use for drawing the maps. They liked arcs in particular. They used the Continental Airlines map as their model, and decided to apply the JCS concept of "bands" or "trunks" out of hubs to different regions. Arcs that are going in the same direction can be group into bands using "quanta," a threshold of when to band, as suggested by JA. PG said that instead of using a "starburst" at the end of the "trunk," they would have little branches shooting off at a certain angles as the trunk passed a city.

JCS suggested using the "New York Walking Algorithm." Divide the map into a grid much like the streets of New York, and find a path between cities much like getting between locations in New York. As a route is traversed by more people, the path thickens. He claims that although the map will not be pretty, the algorithm will be much clearer and easier to code.

JA said the discussions were a bit biased since they all focused on a U.S. map or a world map, and not concentrating on other types of map. He suggested taking the map and pasting it a globe and then focus on the region in question. When an arc falls of that region, indicate the destination. This is like trans-Atlantic flights on a map of the U.S. continent.

LO wanted to take a completely different approach using colors. He proposed to "shade" around the hubs. A hub and its destinations will all be assigned a color, as well as the region covered. When a city falls in two regions so that the coloring is a bit unclear, then an arc can be drawn. This was termed the "Venn diagram approach," due the similarity with Venn diagrams.

A combination of the two main ideas was suggested by MG. There should be bands out of the hubs with the region it services being colored. He then spoke about having layers of translucent colors to indicate an overlap of service between different hubs.

KR noted that although these approaches are good in terms of reducing arcs, they may make viewing more difficult because the viewers attention will be split between arcs and colors.

PJ voiced objection to the "Venn diagram approach" because the mixing of colors will cause confusion. GC felt that the effects of the different layout would be different if the nature of the connections changed, but KR reassured the class that the data will be from an airline map or similar that of, in which there are several major hubs.

Instead of overlapping colors over cities to represent connections to hubs, KR suggesting using a pie for each city, with each slice representing a hub. PM said that the resolution of the cities could be adjusted in order that the pie would be visible, which was a concern of PG. KR also thought it would be hard to follow the different colors. SS suggested that a color key be used if that is the case, but that a key with too many colors would be hard to follow.

Analyses of flight distributions, like PG's of Continental, are legitimate for use in reports, particularly for justifications of algorithm design.

PJ noted the recurring theme in the class was to look at empirical data and make assumptions about how the graphs are layed out. KR elaborated that this is a common theme in research. One often doesn't know what are the "most important" parts of a problem, and so in order to resolve trade-offs between the various parts one can gather empirical evidence to see what kinds of configurations actually occur in practice.

LO tried to resuscitate the "Venn diagram" proposal, indicating that the shapes of the outlines will disambiguate overlapping regions.

The discussion then shifted to rendering software that can be used for this assignment.

MS said that the awt Java package functions well in some some instances and not so well in others. JA suggested generating the text after the maps were scaled down since text does not scale well. JP said that it is better to deal with integer coordinates at high resolution.

For JA, the software used is not important because the final image is what counts. His group intends on downloading a background map from the internet and indicating in the program where the longitude/latitude coordinates are on the map.

Most people are familiar with curved maps, and KR asked how they intend on maping from a flat map to a curved map. If a map is distorted too much, then it is not recognizable. JP did not think a flat map would be a problem, since that is common as well.

It appeared that the whole class planned on using java.awt. LO wanted to use gimp scripting, but only the old version is installed on the network.

MS spoke about moving around cities and wondered whether that issue was related to having only integers for city coordinates. The consensus seemed to be that they were unrelated, as long as the integer precision used for the latitude and longitude were high enough.

Every group should target to produce an image by Wednesday. The presentations are on the following Monday, and the competitions will follow. JP thought that by knowing who the judges are, then they will have idea of how people will respond to the maps. KR asked for a template of the judges. This is what the class suggested:

1. Child, 8-10 years old
2. Adult, non flyer
3. Artist/architect/graphic designer
4. Airline employee
5. Frequent flyer

The main ideas for map drawing suggested so far are coloring and the use of bands. SS asked there was little discussion about the use of arcs, as in the airline diagrams we have been discussing. KR asked the class whether they thought they'd do better than the airlines' maps. Most thought they would, at least in as much as communicating whether there are links between cities. PG said that arc diagrams are cluttered by design for marketing purposes. PM thought that the Continental map was fairly professional looking and not too cluttered. PW was also happy with the Continental map and did not find unclear at the hubs.

MG asked (somewhat toungue in cheek) whether a textual matrix of cities containing entries describing their connections would be in the spirit of this project. KR hesitantly said “yes”, but he doubted that the presentation would be as transparent as a map. He was hesitant because he didn't want the groups to settle for an easily implemented method that was anticipated in advance to be inferior.

JCS wanted to know the math behind drawing arcs. PG said that Bessian curves are used with a control point. JA said that Java uses bounding rectangle for creating arcs. The man page can be found for the Java method drawArc.

GS said they have code that renders the screen. They want to try rendering a few maps before deciding on what methods to use.

3.4 Wednesday February 24, 1999

The class decided on conventions for listing latitude and longitude coordinates. North latitudes will have positive coordinates (to 90 degrees), south will be represendted as negative coordinates; east longitudes will be positive, and west negative.

An unseen data set has been selected. The data set represents a world airline. That is, coordinates are not limited to the U.S. or any other continent.

The class then showed some of their preliminary maps and their techniques for production. JA and JP showed a flat map of the U.S. as the background since plotting coordinates is straightforward. A flat map is when the lines of latitude and longitude are horizontal and vertical respectively. They worked with both a color map and a black and white map. The colors on the color map have to be tweaked to improve contrast. They are drawing hubs and their outgoing edges with the same color. PW wants to add an outline to the hub city to distinguish it from the arcs.

JCS told the class about problems his group was having with longitudes and latitudes. They had chosen a nice-looking nonflat map of the US that they downloaded over the Internet. However, hey didn't know how the map was generated, i.e., what projection was used to draw the map. So they had trouble matching latitudes and longitudes properly. Some coastal cities ended up in the ocean, for example.

JCS and MG showed their map. Hubs are red and the routes out of a hub are blue. The destination cities are green and connections between non-hub cities are drawn with black lines. (Several people remarked that it was difficult to tell the blue and black lines apart.) They had addressed the problem described above by including a "fudge factor" representing the necessary offset of the y coordinate to place a city in the correct locations. PC played with the x and x$\hat{2}$ coefficients of a parabolic function to find the correct offset. Their fudge factors were good (but not perfect) on the US map they presented. Due to their choice of a map without a known projection method, their program is hard to extend to other maps. They are considering accepting user input to configure (or fine-tune) the map to world coordinates on non-US maps.

The class discussed their feelings on the issue of how to configure the maps to coordinates. The majority did not want to allow major revisions to programs to tailor the output to the given input. Too much of this kind of activity would amount to manual creation of maps. PG was opposed to any code changes after the submission deadline. However, some kind of configuration is necessary since the class does not know what the map will look like, that is, a world map or an Asian map, etc. While automatic configuration by the program is the ideal, some groups may be generating specialized maps for some regions (e.g., the US) without having complete coverage of the world. The final decision is that for the unseen dataset for the competition, the data will be made available at 7:30 PM Monday and groups will have two hours to tweak their programs to work with the map and email printable versions of the final route map. By limiting the tweaking time in this way, the class thought that only minor tweaking would be feasible.

City labels were omitted from the maps seen so far. JA was in the process of writing that code, but it wasn't ready for presentation.

PG showed his group's image. Hubs are represented using green dots. MS defined a hub as a city to which more than 10% of the connections belong. He also implemented the "Min Coloring" method for routes. Two routes that are close to each other are drawn with different colors so that they can be easily distinguished.

While the next group pulled their map onto the screen, the class decided on the data sets to be used for testing. There will be four sets of data:

1. The Secret world airline
2. Reno airlines
3. Continental airlines
4. Shamoun special (from mapgen.tcl)

LO and GS presented thier map. They noted that on their map the city label overrides an arc. The cities are shifted at random for a better drawing, so that the same map may not be produced with each call to the program on the same set of data. On one version of the map, Burbank was drawn to the north of San Francisco. On subsequent versions, it was placed more accurately. The class decided that the random seed should be a command line argument so the maps can be generated repeatedly. LO also demonstrated a "Venn diagram" map, with color keys indicated the hubs and the connecting cities. This was still in an experimental state.

The issue of randomness brought up another interesting issue. LO's group could generate a large number of maps each slightly different due to randomness, and choose the one that looked best to them. This could be seen (to some extent) as manual rather than algorithmic design since the choice of the best map would be done by a human. The class didn't seem bothered by this fact, since whichever map was chosen was itself generated algorithmically.

GE and AK showed their map. They still had to tune it so cities don't fall into the ocean. Also, currently only straight lines are used for arcs. They are also working on an algorithm to separate cities that are very close to each other. They also had the same issue with curved maps as JCS and MG. PW said there is a web site that will help find the x,y coordinates of cities on such a map. The website is posted to the newsgroup. KR suggested that groups try to generate maps according to various algorithms presented on that web site, and see which methods and parameters match most closely with the maps being used. Once a close match is identified, the method and parameters could be used to generate coordinates more accurately than an approximate method like fudge factors.

GE and AK's group also have not decided on how to handle arcs that fall off a map, for example, a U.S. map with a flight from Miami to South America.

JHS suggested accepting the curvature of the map as parameters to the program. He also spoke about generating a group of maps and using a heuristic to decide on the best map.

PG asked whether simulated annealing might be used to come up with maps in a totally different way. By creating good functions to measure the goodness of a particular map, one could use simulated annealing as a method to try to maximize goodness.

KR introduced the class to simulated annealing. Annealing is a metallurgical process in which mixtures of metals are heated to high degrees and allowed to cool with precisely measured changes in temperature over certain time intervals, so that take some desirable form. Then can be heated a little more to add more turbulence in the sytem and allowed to reset. The process is somewhat self adjusting.

Simulated annealing is a computational procedure based on this that attempts to avoid local minima. It was used for one of the Stanford projects and can probably be applied later to the Egalitarian Island project. Simulated annealing might be used, for example, as a way of disambiguating arcs. At first the system will be given high "energies," allowing edges to move away from each other according to some model of edge repulsion, and then allowed to set. Then some energy is added to the system, and the process is repeated until an acceptable display of the arcs is created.

Maps will be rated next week on a 10 point scale by judges and then ranked by their total ratings.

3.5 Monday March 1, 1999–Presentations

PG began Group 1's presentation with an explanation that there were two parts to their project: map scaling and graph drawing.

DK said they wanted to use a map that was uniform, with no curvature, and got one from JP. They used the java.awt package to scale it.

MS said they took a practical approach to making the map look nice. If it did not look good, then they chose a different method for drawing it. The firt issue was drawing the lines. By alternating between cool and warm colors, lines could be more easily distinguished. The use of thick and thin lines also aided in distinction except around the hubs. The second task was labelling the cities. The city icons had to be placed somewhere on the map and the labels drawn on a different space than the egdes. The size of the labels had to be managed so that they would not overlay edges. 18x18 pixel icons seemed appropriate for the cities. These icons were circles with varying color schemes, and black and white together worked best.

The group used the trunk and branch approach to drawing the graphs. PG implemented the detailed algorithm for drawing the arcs. He chose some radial angle around a node and the maximum arcs allowed in that angle. If, for each node, that limit is exceeded, then all the arcs get replaced with a trunk. A thick line goes to the first intersection point, and thin lines branch out from there.

GC said there were many limitations in implementing their design: java, colors, overlays, etc. They had to find a different formulation for attacking the problem. The process was trial and error due to the unforeseen limitations. Also, several of the pieces developed for the project didn't fit together as well as was hoped.

GE introduced Group 2's presentation by formulating two principles: (a) display as much information as necessary, and (b) reduce the clutter to make the map clear. PM said their first issue was the selection of a map. Instead of using a world map, they used a collection of regions. PM said that their algorithm chose a map based on the coordinates. AK did the conversion of map coordinates to pixel coordinates. He found a web site that generates a map in the x,y plane, and it worked out close enough.

PJ worked on the positioning of labels. In some instances, there were more labels in an area that can fit. His algorithm could not space them out.

GE wrote the edge drawing functionality. He used the branching method. His algorithm divides the area around a hub into regions. Lines are then drawn out of that hub to its regions. Those regions are subdivided and lines are drawn the those subregions. When the program was actually run, not all lines were drawn.

Group 3 made their presentation using Powerpoint. JP said they used "Node-edge aggregation bands", another phrase describing JCS's "band vision". They used arcs instead of straight lines because arcs are visually pleasing and reduce crossing between lines. A node was determined to be a hub if the number of connections it had was 2 standard deviations above mean.

JHS explained their use of "band vision." First he tried the "explosion" method, in which a band "explodes" at the first cities and spreads out to all others. A city close to the hub causes a problem, though, becuase the band explodes to early. Instead , a band is drawn the middle of the region, with arcs incident to the band at 45 degrees going off to the cities. A band was drawn for the densest 30 degrees around a hub.

JP spoke about zooming in on the map to generalize the problem. They used a generic global map, calculated the bounding box for this particular map, and then scaled the new map. He had to add extra steps to preserve the aspect ration. He also learned how to generate a gif of an image.

JP also explained the coloring used on the map. A unique color is assigned to each hub, and non hub cities are colored red. All edges out of a hub are the same color. An arbitrary choice was made for connections between hubs.

The positioning of labels was described by PW spoke about position labelling. The algorithm finds the most open interval around a city and centers the label in that interval along a 45 degree edge. He used Java2D for the labelling.

One problem with the map was that cities tended to shift locations in the drawing process.

Group 4 began by showing a map. PC noted that the cities were perfectly located on a curved projection. This required some math on his part (and ultimately in the program). He found the x,y coordinates of cities on the the same latitude and determined the parabolic equation for the line. For the longitudes, since the curvature changes, he looked at the lines on at the extremes to an equation for the change in curvature.

MG used band notion in certain instances for drawing the arcs, but didn't know how to obtain thickness in drawing them.

JCS explained that a node is determined to be a hub if 25nodes are connected to it. The graph is then subdivided 30 degrees around each hub to determine the component clusters. The trunk to that cluster branches off at the first node.

MG said that all component clusters have the same color.

If the map is of Europe, then the program can be called with different command line arguments and works with a different set of equations.

One of the known bugs is that nothing has been done to handle the drawing over lines.

RS said the goal of Group 5 was to reduce the clutter apparent in other maps. One issue is to make sure that objects are not drawn on top of each other, so they decided to shift around the cities. Because of this, there was no underlying map. Also, to reduce the number of arcs, they instead labelled the cities with the connection. LO was not able to fully implement his original "Venn diagram" idea in which the background was colored to represent sets in a Venn diagram. However, they did implement two methods that were close in spirit to the original idea.

KC showed a map to the class. Each node has a small colored square if it connects to the hub drawn with that color. A key is printed at the bottom of the map. LO showed another map, in which two circles were overlapped on a city to indicate connections to two different hubs. On a third map, two non-hub cities were connected using a line. It was evident from a fourth map that there was a lot of shifting in the location of cities.

GS said the program uses a deterministic algorithm to move the cities. On the Continental map, everything was progressively moved west. While LA was east of some cities, it had its own "pocket" that distinguished it. LO said the Venn Diagram was also a good marketing ploy, because it had so many circles that were attractive and that gave the impression of good coverage.

A hub is determined by the number of cities connected to it. They added LAX to the list in one map because it made the map look nice. The colors were chosen arbitrarily.

KR assigned groups for the next project, Battlesplat. The class voted on and agreed to forming groups of three. Groups were formed through a process of elimination, by assigning students to work with people they haven't worked with yet. These are the groups:

1	PW	PG	KC	
2	JA	MG	DK	
3	JP	MS	PM	
4	JHS	GS	AK	
5	JCS	GE	RS	GC
6	PC	LO	PJ	

3.6 Wednesday March 3, 1999

Five judges were used for the map competition: Alice, the departmental receptionist, qualified in the category of "nonflyer", having not flown in the last 2 years. An artist friend of the instructor was the second judge. The third judge was a nine year old girl (a neighbor of the instructor). The fourth judge was the instructor. The fifth judge was Terry Lentz, a travel agent at STA Travel. Thanks to all the judges who generously donated their time.

KR read of some comments made about the maps during the competition. (The list of comments is available on the class web page; see Section 8.4.) One comment was that the mountains on a map made the routes hard to see. The artist liked the use of colors, although others were confused by them. Some people had trouble reading names. One person did not like the shoots off a branch, and Group 2 got a “10/10 for comedy” on one map.

The results of the competition were in as well. Group 5’s Venn diagram maps seemed to do best. Group 4 scored well on many maps as well. Group 5 will get to choose a map for a t-shirt, and GC will be in charge of making them. All results and comments are available on the class web page.

4 Battlesplat

4.1 Wednesday March 3, 1999 (continued)

The class then began discussion on the third project, Battlesplat. Battlesplat is a modified version of the original Battleship game in which there are only three ships, one each of length 3, 4, and 5. There is one player placing ships and one trying to sink them by guessing their locations. The guesser can choose only in squares adjacent to the last move. The analogy is to an airplane, which can not jump from one location to another but rather drops bombs as it moves along. PG proposed that guesser be allowed to choose adjacent to *any* previous guess, but the class chose to stick with KR's original specification. MS thought that it would waste time to require the guesser to move only to adjacent squares. That makes the problem interesting, according to KC. KC also observed that if one gets four hits in a row, it can not be determined so quickly if that was a ship of length three or four.

The map size will be a parameter. The placer will get to hear the guesser's choices choices so it can update its strategies for future games. If log files are being maintained, programmers should assume that multiple instances of their programs will be run at the same time. The class debated on the time limit per move, and settled on 30 seconds, which many considered to be a long time anyway. 60 seconds will be allowed at the end of the game, since the placer and guesser have to update their strategies. The limit on board size will be such that the product of height and width will be no greater than 10,000.

There was some discussion related to whether a guesser should be told when a ship was "sunk". The argument in favor was that battleship is traditionally played that way, and that it would simplify the game analysis. The counterargument was that the problem would be more interesting with a degree of challenge in the game analysis. Also, announcing a sinking of ships would complicate the game interface. KR had a strong preference not to announce the sinking of ships, and the class settled on this choice.

The class then played a game on a 10x10 board. GS suggested the first move. He use the "empty square principle." By checking rows 4 and 7, it can be determined if there are any ships placed vertically in between. This is because the smallest ship hss length three, and if it starts on row 5, it must end on row 7. DK suggested using the "snake around" principle, that by "snaking" around the board and sectioning off blocks, the same determination can be made about horizontal and vertically placed ships. A 2x2 block surrounded by misses cannot contain a ship. JCS said to move diagonally to eliminate rows and columns simultaneously, but did not realize that the guesser can not move diagonally.

PC suggested a "spiral approach" out of the center. This led to some discussion of the placer's strategy, and a suggestion that the edges might be good places to put ships. On the other hand, if guessers see this happenning, they can adjust their strategy accordingly.

JCS felt that since nothing is known at the beginning, a random walk might be a good idea. PC objected, stating that it may not cover the board efficiently.

Moves were made by committe. This was the board state after a certain number of moves (M means Miss, H means Hit, and . means untouched):

```
          X
    1 2 3 4 5 6 7 8 9 10
  1 . . . . . . . . . .
  2 . . . M . . . . . .
  3 . . M M . . . . . .
  4 . . M . . . . . . .
Y 5 . . M H H H H M . .
  6 . . . . . . . . . .
  7 . . . . . . . . . .
  8 . . . . . . . . . .
  9 . . . . . . . . . .
 10 . . . . . . . . . .
```

The starting position was (4,2). Once the class made four hits in a row, they stopped for discussion. JCS said that only the positions above and below need to be checked to see if this is a ship of length 4 or one of length 3 placed alongside a vertical 4 or 5. RS said to assume that it was a 4 ship, and if the class was wrong, to go back later. JP noted that this is in fact a learning parameter because it suggests how the placer placed the ships. MG said to not worry about what ships are sunk, and to instead focus on making the 12 hits necessary to win. KR pointed out that the set of remaining ships constrained the search space in a way that might be profitable to exploit. For example, if we know the 3-ship is gone, we could modify DK's snake principle to trace out 3x3 squares rather than 2x2 squares.

MS was also against going back to check what ship was hit. PW said that if the above "empty square" principle is used, then a row or two can be skipped and more data can be gathered. PG brought in the old game of "core war," which he thought had certain similarities with the present situation. The class continued the game to yield the following board state:

```
           X
    1 2 3 4 5 6 7 8 9 10
  1 . . . . . . . . . .
  2 . . . M . . . . . .
  3 . . M M . . M M M M
  4 . . M . . . . . . H
Y 5 . . M H H H H M . H
  6 . . . . . . . M . H
  7 M M M M M M M M . H
  8 M . . . . . . . . H
  9 M . . . . . . . . M
 10 M M M M M M M M M M
```

At this point, it was obvious that the ship on the right was a 5 ship, because if the ship on the left was a 4, then the the right cannot have a 4 plus some other ship, and there are only three ships on the board. In fact, the ship on the left must be a 4, because it can not be a 3 attached to any other ship other than one of length 3, of which there is only one on the board.

GE was placed as commander, and after snaking around the board with no announced strategy, sank all three ships. The total number of moves made was 83.

The groups for the project have been updated as follows:

1	PW	PG	KC	BES
2	JA	MG	DK	
3	JP	MS	PM	
4	JHS	GS	AK	
5	JCS	GE	RS	
6	PC	LO	PJ	GC

4.2 Monday March 8, 1999

SS will make information about the interface available on the newsgroup. Programs will be available in a directory on SS's account. KC suggested reducing threshold for number of moves beyond which the game terminates. KR originally chose 3 times the number of squares on the board to allow "really bad" guessers to be distinguished from those that are just "slightly bad". A case that might come close to the threshhold would be a random walker, because it will have retrace its path many times to cover the whole board. Some threshhold is necessary in case there's a guesser that never reaches some square containing a ship.

Groups can submit up to 4 placers and guessers. JHS asked if a 1x10000 board is acceptable. Certainly it is, and the class discussed strategies for such a game.

GS said that the endpoints need to be covered by the placer to guarantee for optimal play. If there is a ship on both ends, the guesser needs at least 10,000 moves. BES suggested starting at position 1 or two squares in. Since starting in from the endpoint may require somebacktracking, the best strategy is a straight line starting at one of the endpoints. JCS said it would be a tie in all games.

Then a 2x1000 board was discussed. A good strategy (but one that was not proven optimal) for the placer is again to play at both ends, but this time on different rows. This guarantees a minimum of 1001 guesses. The following strategies for the guesser assume this placement strategy.

JCS suggested snaking around the board, like this:

```
XXX  XXXX
  XXXX  X...
```

This requires 4 moves for every 3 horizontal positions moved, which means that the number of moves is (4*1000)/3 = 1333. This is approximately the upper bound on the number of moves for the guesser. By "approximately", we mean that we also need to factor in a small number of extra moves to knock out a ship when we hit it.

JP suggested going in a straight line halfway across the board, and then switching to the other row half way across. If a ship was missed, then backtrack along the squares that were not yet checked. With probabilities factored in, the cost of this solution is 0.5*1000+0.5*1500. This is determined by saying that there is a 1/2 chance that all three ships are hit when first crossing the board, and 1/2 chance that the third ship is only hit at some point while returning to the other side; on average we expect to hit that ship half way along. The average cost of this strategy is 1250, which seems to be better than the previous one.

GS suggested a modification of JCS's strategy, using that strategy until the ship in the middle of the grid is found. After that, switch to JP's straight-line method. The average cost until the ship is found is (4/3)*500, and after the ship is found it is 500, for an overall average of 1167 guesses. PW said that if we see misses as we go across, then we should switch to a different row. KR thinks this game is interesting enough to include in the tournament. KC observed that if the 3-ship is hit at the very beginning, then the snaking algorithm can leave larger gaps.

The class moved on to a 3x1000 board. PJ suggested snaking across all three rows, in order to catch a vertical ship. Snaking on a 3xN board looks like this:

```
XXX  XXX
  X  X
  XXXX
```

BES suggesting that if the 3- and 4-ships are found, then the spacing (snaking) algorithm can move by even greater increments than if only the 3-ship was found. This observation suggests a strategy for the placer to place the 3- and 4-ships far apart to prevent such an increasing of increments.

AK wondered if we'd too hastily dismissed the middle of the board as a good starting place. The class soon agreed that it is not good to make the first guess in the middle of the board because the guesser will still have to go to both ends.

DK says that if you start in a corner, then you don't have to go back there. However, if the placer realizes this, then it may start placing ships in the middle.

RS thought that we should treat all unchecked squares as if they had an equally likely chance of containing a hit. KR wondered whether the claim that unchecked squares are equally likely to contain a hit was true, and suggested a discussion of the following scenario in which we assumed that the placer had placed the ships randomly:

```
      X
   1 2 3 4 5
 1
 2
Y3
 4 H H M M
 5
```

We've played at (1,4) through (4,4) and have some partial information. Is it now true that all unchecked squares are equally likely to contain a hit.

BES said it is known that the two squares above the hits, i.e., (1,3) and (2,3), are also hits. KC said that, with no further information, all the other squares have an equal probability of a hit. KR pointed out that at least one of the two squares (1,2) and (2,2) is a hit, giving them a higher probability. KC tried to amend his statement as being a lower bound on the probability of a hit.

According to GE, depending on the placement of the ships, the probabilities of the locations of other ships changes. For example, if the two hits belong to a 4 and 3 ship, then for any such configuration there are only two locations for the 5 ship. GE estimated that the probability of square (4,3) being a hit is 1/6. He obtained this number by considering all possible cases for the two ships we've hit, and computing a probabilistically weighted sum. It was not clear, however, that he had taken all possibilities into account. His conjectures could be verified using the Monte Carlo method, i.e., generating millions of boards and for those satisfying the hit pattern described above, checking how many have a hit at (4,3).

JHS pointed out that to gather the information about a square that might have a hit may cost a lot in itself. Thus it might be better to investigate a nearby square that is less likely to have a hit, rather than a distant square that is more likely to have a hit.

KR then wondered whether it would ever be a good idea, for the squares that are known hits such as in the scenario above, to schedule an attack on those squares later (as opposed to sinking the ship immediately). LO said that on a larger board it may make a difference, but in a small board that will not change the number of guesses significantly. KR noted that this observation leads to size specific behavior. Others observed that it probably pays to sink a ship sooner because the information about the length of ship sunk could be used to streamline the remaining search.

To continue the game above, PW would like to explore the perimeter of the square. MS asked what the behavior should be when there is a hit. MG said that if there is a hit when going down, then it is not necessary to explore upwards.

KR set the tasks for Wednesday. Every group must submit a placer, even a dumb or random placer. For the first Monday after break, a guesser will be due as well. Tournament results will be available for the reports. which will be due the Friday after presentations.

4.3 Wednesday March 10, 1999

KR took general questions and comments from the class. The interface will validate moves, so the placer does not need to duplicate the work of validation. PG advertised a resource for "camouflaging" ships by making their placement ambiguous. For example, placing a 3-ship perpendicular to one of the endpoints of a 5-ship might make the 3-ship look like a 4-ship.

JCS made a request to discuss the types of AI desired in the programs. KR said we'd get to that later in the class.

The class played a game against a program called placer2. The class discussed what to do when three ships are placed next to each other, but it is ambiguous how they are placed. They decided that it is best to choose the closest square if it is equally likely a ship is there.

This prompted to LO bring another dimension into placer strategy. If a guesser is assuming that the placer is putting the ships far apart (a heuristic we discussed earlier) then he suggests that the placer actually put ships together so that the guesser wastes time going in circles. However, KR said that if the guesser gets M H H M in a path, he knows that two ships are next to each other, so the guesser might end up better off.

It was suggested to occasionally use a different placer strategy to throw off the guesser's counterstrategy. PG said they will have to solve a constraint propagation problem. Once a ship is hit, it is necessary to find out where the other ships are. JP anticipated programs switching back and forth on each other. KR made an analogy to the game "Rock, Paper, Scissors," (which he called "Scissors, Rock, Paper") in which the opponents both play strategies that win against some and lose against others, (rock loses to paper but beats scissors, etc.), and therefore must learn to switch between the strategies. PJ suggested switching strategies randomly in the game of Rock, Paper, Scissors so the opponent can not run metrics. LO thought one can play a fixed strategy in Battlesplat, which is to circle around a ship until it is definitely sunk. This strategy is determinisic, and after some time, the placer can play many games to find out what placement causes the most moves. JCS suggested to adapt the placer strategy to the guesser strategy.

For the guesser, PJ suggested choosing randomly between three robust strategies, like snaking from the corner, spiral from the center, and some third method. In that case, the placer can find a strategy that on average causes the guesser to do worst for all three methods.

The class tried to figure out what the placer should do, then, if there are three deterministic guesser strategies. One suggested random placement. JP thought that the three strategies are enough, because learning what strategies are used is not trivial. PG suggested using neural networks for learning strategies. JP was opposed to his suggestion because neural networks take a long time to learn. Since some people did not know what a neural network was, JCS described it as a directed graph with a series of input nodes, internal node layers, and output nodes. Each node has a simulated function that outputs an answer based on the input. A neural network can learn how to add, for example, by seeing many examples. JP said it simulates the brain because there are multiple layers doing math functions. KR described the general learning problem as programming some behavior based on a training set, and then observing its behavior on unseen data. For neural networks, it's the mathematical functions that are determined by the training data. A neural network might be good at predicting what move will be made, but it is ignorant of strategy. JCS assumed they will have time and space to create neural networks, with a three valued input for all squares. It will predict where other hits will be.

LO said that one takes a gamble by adapting the strategy to the opponent. KR simulated a game. If the placer is random and the guesser snakes around the board, the expected score is the average distance of the furthest ship along the snakelike path. (LO wanted to see a strategy better than snaking, because the placer may place in three corners.) If the placer knows in advance the path the guesser is likely to take (ignoring local diversions to sink ships that are hit), then the placer should put at least one ship at the end of this path. Suppose the snaking path has length 1000. Then this placer's strategy would mean that the guesser needs to traverse the entire path, using at least 1000 moves. On the other hand, a random placer would have ships at three random places along the path. The number of guesses needed would be the expected value of the maximum of three random numbers drawn from the range 1 to 1000. This expected value is certainly significantly less than 1000 (KR challenged the students to calculate it). The bottom line is that even with a relatively thorough guesser, a guesser that is too deterministic is vulnerable to being taken advantage of.

PJ thought performance would be better with a fixed strategy than a game which learns within a single game, because learning waste moves. While learning within a game is wasteful, KR said that learning across games doesn't waste moves, and can be beneficial.

GC had a problem with both Battlesplat and Toetjes. He likened the projects to birds. One could be sitting with the other attacking, or both could be flying and then the attack is more difficult. While it appeared to him that it would be difficult to choose a strategy, KR pointed out that they can learn from each other's behavior. A good example of this would be fighter planes. KR said to think of the player strategies like survival of the fittest. They are not originally designed to be the best, but are good in their usual environment.

MS asked how much learning is required until real lessons can be learned. BES said that learning rates (parameters in the code) can be set, much like growth. A baby expects things to be alike, but understands better with experience. JA thought that predictability, in terms of determing the algorithm being used, is an interesting subproblem. GE wondered why random placement can not be used. This was the whole rock, paper, scissors argument. A random solution is intrinsically not learnable. JA said this is like the first project. In the first project, the midpoint chooser was the winner. PG noted that the programmers learned that their opponents were using a variation of the Stanford algorithm and chose accordingly.

PG said the thirty second move was too long because statistics can not be gathered. This prompted KR to remark that some programmers may choose to put a sleep call into the programs so that others can not run many tests to learn the strategies. However, he asked that the class does not add a sleep command to their programs.

JP suggested a way to use the learned statistics to derive weights for the various strategies. As he tournament progresses, the weights are adjusted, leading to changes in the overall behavior.

KR returned again to the snaking path issue, in which the placer takes advantage of the guesser's determinism. He suggested snaking in a loop instead of a straight line. The starting point and direction would be chosen randomly as part of the strategy. PJ thought it is good strategy because the guesser will not double over. The average cost is the average distance between the starting point and the furthest ships, which works out to be the same cost as for the random placer above. PW said that it is possible to start next to a ship and move in the wrong direction. The class agreed, but reasoned that averaged over many games, it will all even out.

KR suggested using fractal-like space-filling curves for "inner snaking." DK preffered conservative strategies. For example, when snaking, he suggested starting a different place and choosing the direction based on experience. AK said it is worth taking a risk of missing a ship by leaving 3x3 empty squares while snaking instead of 2x2 empty squares. The question is how much better is this strategy.

GS defined the project as a structural question. He asked what ontology should be used to talk about strategy. BES said to define a completed state pattern description of where the ships are placed. GS wanted an abstraction for how the ships are placed. KR gave an example of abstractions as saying "in the corners, all lined up, etc."

PJ asked if programs will be ranked or scored. KR prefers running tournaments, taking the average number of moves between a guesser and placer and for the guesser/placers along all games. The ranks will be proportional to the scores. JP requested a second table of minimum and maximum scores.

The placers must be completed in the next few days, and by Monday, there should be a guesser. The final versions of all programs should be done on Wednesday. The results of the tournaments will be in by Thursday, and the reports are due Friday.

4.4 Monday March 22, 1999

The class returned from spring "break." KR told the class about a cool collection of maps he saw in the British Airways in-flight magazine. GC reported on the T-shirts: They could be done for eight dollars each if only two colors were used. The class seemed to want more colors (and were willing to pay $15 each) and several students volunteered to help find other T-shirt options.

The Battlesplat programs will be run on Solaris, so those who are using Java will not run into "path" problems. These are the programs that are now available for each group:

group	guesser	placer
1	guesser	placer
	guesser2	placer2
2	guess.sh	splat.sh
	Spiral...	
3	mjp.guesser1	mjp.placer1
4	guesser4	placer4
5	gguesser	gplacer
6	g6guesser	g6placer

The class discussed the ideas they had started working with. After running many random placaments, RS assigns a probability for each square that it contains a ship. He is searching for a way for to calculate those probabilities without running so many games. JP wondered whether one would get sufficient bang for the buck for doing all that work. In particular, he imagined large regions with close to uniform probabilities. As a result of the class discussion, RS suggested sampling if there is a large search space.

This was only half the problem. Once these probabilities (or sampled probabilities) were determined, RS was not sure how to best use the information. As has been discussed in previous classes, it might still pay to check a nearby low-probability square in preference to a far-away high-probability square.

BES and his group wrote a brute force guesser that checks each square in a linear fashion. They want to use it as a control, because a strategy should do no worse. (This comment was interesting in light of a game played at the end of class: The other guesser implemented by BES's group took 121 moves on a 100-square board, largely because its local search for the remainder of partially hit ships traversed many previously explored squares.)

JHS was devising a strategy for the placer he is developing. He thought of keeping track of the number of moves it a guesser to reach a square and how often it is reached, for each square. He was considering using that information to seek out untouched regions, or regions that the guesser reaches late in the game, for placement. MG was concerned that it will not take board symmetry into account.

JCS was working on a placer that takes as parameters a centroid and a level of closeness that define a region for ship placement. As the ships get closer to the centroid, their configuration looks more like a U. Further out, the placement is more random. He felt that he will be playing guessers that will choose based on area and closeness. He felt his method is good because the parameters can be tuned. Two parameters would be more easily tuned than more parameters.

JP thought of choosing zones in the board and randomly permuting ship placement in that zone.

The class discussed how to counteract JHS's strategy. PJ mentioned an idea from a previous lecture of choosing a random starting point on the "snake loop." JHS said that this requires an efficient snaking algorithm. LO said that if one knows that the opponent is using this strategy, then he can also keep track of the regions that are being used, and anticipate the placer's placement. KR called this the return of the anti-Stanford algorithm, because the class may start programming in reaction to one method. GS said that the placer should program based on the guesser programming based on the placer strategy. This cycle could go on forever.

These were the general descriptions of the current placer strategies for each group:

1. (a) Place far apart; (b) place close together to confuse
2. Place far away from the first hit in last game
3. Pseudo random but degree of "fixed-ness"
4. Chooses three of 9 zones in a large board

5. Segfault (ha ha)

6. Somewhat random; anticipating differing rules for board size

JA said that his placer handles collision on small boards by not dealing with boards smaller than 5x5. The class chose the following board sizes for the tournaments:

```
small 10x10 8x23
medium 43x43 27x64
large 100x100 9x1000
```

Rank will be determined by the average number of moves per game.

The tournament will consist of pairing each guesser G against each placer P for some number of games (say 100). The score for this pairing will be the average number of moves taken. Thus for each board size we will have a p by g matrix of scores, where there are p placers and g guessers. The placers will be ranked based on the average score across the whole of the placer's column, and symmetrically for guessers across rows.

The class also discussed and voted on a plan for dealing with crashes or programs generating improper output. The matrix will be constructed as above, with entries based on average scores over fewer games if one player crashes during play. However, any guesser that crashes will have its whole row deleted before the ranking phase. Similarly, any crashing placer will have its column deleted before the ranking. That way the ranking won't be biased by some placer doing badly against one opponent while crashing against another. The class thought that the interface for this project was simple enough that a program that generated inappropriate output deserves not to be ranked.

An alternative suggestion (that was voted down) was to penalize the faulty program by assigning it the maximum moves in case of a faulty guesser, or a minimum number of moves (say 12) for a faulty placer. The exact details of the tournament will be posted to the newsgroup.

Presentations will be on Wednesday and limited to eight minutes. The tournament will begin after class, and results will be posted on Thursday by 5pm. Written reports are due Friday by 5pm.

4.5 Wednesday March 24, 1999– Presentations

PW began the presentation for group 1. They wrote four guessers and four placers. Two of the guessers used a brute force algorithm that scans every line until all ships are hit. One of them randomized the starting corner. KC presented "jerry," which implemented a modification of the snaking algorithm that increases the size of its gaps after the smallest ship is sunk. The fourth guesser uses an algorithm called killship, which recursively searches a nearby area when a hit occurs.

The first of their placers, presented by BES, places the ships far apart based on the height and width of the board. One is placed in each of two opposing corners and the third is placed in the middle. The ship orientation runs parallel with the larger side. PG described the second placer, pinta, which uses the place cluster algorithm. Ships are placed in a cluster which has a bounding box and origin.

Group 2 presented next. The goal of JA in writing the placer, splat.sh, was to enforce a minumum bound for the number of moves made by the guesser. He decided to do this by hiding the three ships as best as possible. The four and five ships are placed in the corners so that a minumum of width+height moves is required. It adjusts the placement of the three ship based on the previous moves of the opponent.

MG wrote the spiral guesser, which starts at the center and spirals around the board, covering every row. It keeps track of the hits for each game and adjusts the starting location.

DK described the nothing guesser, which is another variation of the snaking algorithm. It tries to maximize information about the board by evaluating the possiblilty of its in areas it sections off. No special actions were set up in case of a hit, but the program tends to explore the space around it and ends up sinking a ship right away. A delta is calcuated for which direction to turn based on the information it has.

MS spoke about the group 3 placer. It picks a random starting location and tries to maximize the distance between ships. It then randomizes between strategies, such as placing in corners or in specific zones. JP worked out the base cases for the group 3 guessers, and PM worked on the general algorithms. He started with a linear algorithm that uses the board size as an upper bound on the search time. Another algorithm starts in a random corner and tries to reduce backtracking.

JP spoke about the integration of their algorithms. They tried to set up pseudo-learning. The same initial weights are given to the different algorithms and adjusted as the tournament continued until the best player is chosen. Their programming was done in Java because it is simple to plug in new modules. Some disadvantages of using Java were that it was difficult to handle different thresholds and degenerate cases. One concern that group 3 expressed was that they devoted a lot of effort to the learner that was somewhat wasted because they hadn't implemented enough good strategies to have something useful to learn.

Group 4 presented next. GS said that implementing the snaking algorithm was the hardest problem, which kept them from looking at higher level problems. He had an idea to divide the board into areas and to search out the ships by exploring some areas and not the others, but it was hard to implement. He called it a gruelling project. AK wrote the sinker, i.e., the subalgorithm for sinking a ship after a hit has been achieved. He had (in his opinion) a cool algorithm which, unfortunately, became complex because it needed to check for being near the boundaries, and was finally scrapped. The basic method implemented for the guesser was to follow a line of hits, and then find the way down the board line by snaking. It did not check for edges.

JHS worked on the placer. In the program, he maintains two parameters for every position on the board: the number of games in which it was hit, and the average number of moves till it was hit. The board is broken into 5x5 regions into which any ship can fit. Based on the weights of the positions, the program decides on the ship position and orientation. The quality of the game improves at first but levels off rapidly.

GE said that the group 5 guesser was similar to others. His guesser has 3 steps. First identify the placer pattern in previous games, that is, whether ships are centered, on the edges, or scattered. It then uses the snaking algorithm, and performs some optimization tests. RS did not write another version, but was a big fan of ranking positions from the beginning.

JCS wrote the placer. He found out from JHS how to maintain information about the board. In the program, he keeps a buffer the size of the board that maintains constant knowledge of the guesses. A square with lots of traffic has less points. It then takes the highest bump in this space using the neighboring scorse to determine orientation.

PJ said that group 6 wrote three main modules: one placer and two guessers. The first guesser uses a basic snaking algorithm which starts somewhere on the general path and chooses a direction. It is guaranteed to hit all ships because it only leaves 2x2 blocks. PC had an idea to incorporate something like gambling, which varies the snaking path. It hits many ships, but sometimes misses them because it leaves 3x3 blocks. Nevertheless, their calculations suggest that the benefits outweigh the risks.

PJ wrote the hit function. It handles the base cases well. If a hit occurs, the program continues in its direction. If a miss occurs, then it knows that the ship is perpendicular to its path. Two hits in a row bounded by misses means that two ships are next to each other, while three hits make the situation harder to distinguish. It then returns to the original location to continue the snaking path.

LO spoke about the placer. It places the ships somewhat randomly. It keeps track of its performance on that board, and modifies it accordingly. The program should occasionally produce an unplayed board that the guesser will still do poorly on. In the final comments, JP suggested non-symmetric skipping for the snaking algorithm. LO said that if the placer is doing well, it will still sometimes be forced to generate a new board.

KR asked if anyone in the class saw placer/guesser pairs that they might predict the outcome for. Cluster will do poorly against spiral. JCS says their placer will do badly with even distribution. LO found out that the hit function is easy to fool, so clusters will not necessarily lose badly if they are stumbled upon.

Groups for project 4 have been posted to the web. A student that can find groupings for project 5 that minimize crossovers will get a booby prize.

5 Egalitarian Island

5.1 Monday March 29, 1999

Group 1 received a travel version of Battleship as a prize for their impressive results on project 3. All stood for a photo. GC is collecting \$15 for t-shirts by Wednesday. The money is also being used to subsidize t-shirts for four of the judges.

KR took questions and ideas about project 4. KC asked what assumptions could be made about the shape of the island. KR responded that the island polygons can be concave, and can have an arbitrary number of sides. They will, however, be nonoverlapping.

The class gave ideas for calculating the area of a polygon. JCS suggested breaking the polygon up into small triagles. A ten line algorithm of a different nature can be found at the following location: `http://www.mhri.edu.au/~pdb/geometry/polyarea`. KR suggested starting with a vertex of the polygon and forming triangles between that vertex and all other adjacent pairs of vertices. If, from the viewpoint at that vertex, the third vertex is to the left of the second, the area of that triangle should be subtracted from the sum of all other triangle areas.

PG pointed the class to `http://alphaworks.ibm.com/tech/gfc`, which has classes dealing with graphs and graph drawing. KR estimates that 0.1 percent will be a generous percentage of error for equal subdivision of the graph. Nice looking maps are not required, in answer to a question from BES. The only requirement is to minimize the sum of the perimeters of all subdivisions, i.e., the amount of fencing the islanders require.

To subdivide the map, PG suggested placing n reference points inside the map, where n is the number of residents. Those points define a Voronoi diagram that partitions the space as follows: A point on the island is associated with its nearest reference point. Thus, we would get n regions with boundaries corresponding to lines that are perpendicular bisectors of line segments between reference points.

The question then arose as to whether this arrangement would lead to an equal division of the area of the island. If not, PG argued, then you could shift the reference points around until you got it right. There was still some concern though that the shifting process might not converge, since shifting any reference point affects all of the neighboring regions too. Further, there was little intuition as to why such a solution would give small perimeter values.

GS suggested dividing a polygon into small pieces, and join as many as necessary to get the area needed. However, he had no estimate on the total perimeter. PW suggested creating a "fan" from a starting point on the boundary, rotating across the island and putting in an edge when the area was just right. However, that may actually maximize area since there are long boundaries running close to parallel to each other. Further, the approach might not work for concave islands.

JP and JCS suggested transforming an island in to one that is simpler to deal with. The broad idea would be to find some transformation T that maps an island into an island of shape that is easy to deal with. The transformation T would need to have some special properties, such as being invertible, and mapping perimiters in a monotonic way. While appealing in concept, they did not have a candidate mapping to propose.

PC proposed to find the center of gravity for the island and create a wheel. Then he said to rotate the wheel until the areas are just right. (This is similar to PW's idea above, but using a point in the interior of the island, rather than on the boundary.) For convex islands, the resulting partition looks something like a pizza. By looking at various starting angles, one could get a set of solutions and choose the one with minimum perimeter. PG suggested, however, that much better solutions might exist. For example, rather than n "pizza slices" one might be better off having a central region and n-1 slices surrounding that region.

JCS suggested making a sub-pizza in the center. He thought it would still be inefficient even if it reduced fencing compared to the basic pizza approach. JHS suggested using curves, with the intuition that circles minimize the perimeter-to-area ratio. PC countered by saying that there would be a region on the other side of the curve, and that a straight line might be better. KR challenged the class to try to prove something conclusive about the use of curves.

KR created some data sets located in `~kar/4995/island`. One looks like Australia and another like Manhattan. He does not expect there to be more than 100 inhabitants per island. However, there won't be an explicit limit placed on the number of edges in the island boundary.

PW wanted to know how to tell when their programs optimally divide to map. That cannot be answered since that is the definition of the assignment!

KR demonstrated a three-way partition and asked how the perimeter can be minimized. GS suggested equalizing angles. KR asked how increasing the value of one angle would affect the calculations. He tried to determine if increasing the theta of one angle would decrease the perimeter. The trigonometry started getting a bit complicated, so KR asked the class to work out the math off-line. The intuition behind equalizing the angles at 3-way intersections is that one could lengthen one edge by a small amount and decrease the lengths of the other two edges. The decrease would outweigh the increase in some circumstances.

KR then told the class about a reference suggested to him by a colleague. The colleague said that the problem reminded him of bubbles. Soap film bubbles tend to form surfaces between them that minimize surface tension, and there seems to be a close analogy (in 2D rather than 3D though). KR brought up a page which has a bubble simulator and a description of the program that produces it. The source is available at `http://www.geom.umn.edu/software/evolver`. It may not be quite the problem the class is working on, because the perimeter of the island is fixed. Looking at some of the figures reachable from the evolver page, it seemed that the curves are on the outside of the figure, and that the interior regions were separated by lines that were either straight or only very slightly curved.

For the next class, the students should read the evolver material and try to apply it to Egalitarian Island. They should also start thinking about algorithmic and coding issues, since coding brings out subtleties of the problem and last minute coding causes bugs. Looking back at the previous project, some class members thought that (in retrospect) more time during the battlesplat discussion could have been devoted to how to implement snaking, since there were some nontrivial algorithmic details that were not realized until the last minute. Hopefully we'll learn from that experience this time around.

5.2 Wednesday March 31, 1999

(This writeup based on notes taken by BES in the absence of SS.)

KR began the class by asking who took a look at the bubble simulation software called evolver that was referenced in the last class. Five or six students raised their hands.

PG thought that the evolver program can be applied to the problem but it will take effort to do so. It seems you can constrain it to set fixed boundaries and vectors... more or less what we're looking for. The format for the files is non-trivial. MG pointed out that evolver used the fundamental shape of the triangle in its data representation.

PJ said "If there is an algorithm to get a minimum amount of fencingfor an arbitrary case, I just do not see it." He thought one can definitely divide up the island and get equal areas, but it seems like there is no (obvious) way to divide up the island (no absolute algorithm) to guarantee minimal fencing. The bubble software does not guarantee a solution, as it is an iterative solution.

KR asked whether PJ was already giving up on finding an absolute algorithm. While PJ acknowledged there always existed an optimal solution, it seemed to him that you cannot always find it. If so, KR suggested, perhaps we will not be able to solve the general case, but we can come up with heuristics to attempt to solve the problems. JA agreed with PJ on the difficulty of finding an exact algorithm.

GE wondered whether we can gather information for the general case, from examining the specific cases. KR thought this was a good approach, and wondered how useful or comparable the information we gather from the specific cases will be in solving the general cases. PJ added that for regular shapes and small numbers of inhabitants, it is easy to see the solutions because the borders "cooperate" with you.

KR then drew an example: a square island with two inhabitants.

```
      ----------------- (1,1)
      |       |       |
      |       |       |
      |       |       |         CASE 1: one method for partitioning in two
      |       |       |
      |       |       |
      -----------------
(-1,-1)
```

```
      ----------------- (1,1)
      |     -----     |
      |     |   |     |
      |     |   |     |         CASE 2: could also be a valid partitioning
      |     |   |     |         (this was a circle on the board).
      |     -----     |
      -----------------
(-1,-1)
```

```
      ----------------- (1,1)
      |          /    |
      |       /       |
      |    |          |        CASE 3: another partitioning
      |       \       |
      |          \    |
      -----------------
(-1,-1)
```

MG wondered about recursively subdividing the problem by chopping the island into pieces by inserting a boundary half way along (for an even number of inhabitants, at least) and then recursively solving the two halves. This seemed like it might work, but there wasn't any reason to think it would do well as far as fence length minimization goes.

Returning to the examples on the board, GS thought that you want to consider the angles, and perhaps select lines that are perpendicular to borders.

GE wondered whether the angles are a by-product of the solution. If we're dealing with the square here, we want to minimize the use of the perimeter. To do this, we want to use as much perimeter of the original square as possible. KR was confused about what metric can one use to compare the amount of original perimeter in various solutions. It seemed to him that all solutions use the perimeter to exactly the same extent. After some discussion it became apparent that usage of the island's perimeter was not an issue.

MS asked whether an islander can be given a noncontiguous region ? KR said that we answered this last time. An islander must have a completely contiguous region for him or herself. In other words, he/she must be able to walk through the entire expanse of his or her own area without crossing over into another's area. Remember, these are very high fences.

KR had taken a quick look at the evolver manual, and it seems that springs might be a nice abstraction of the problem. The endpoints could be moved (say, along the island's perimeter), and tensions on the springs could be changed, while making sure to keep the areas the same. Tight springs should tend to lead to short segments, and small path length.

GC thought the spring analogy is a really great one. Think about the bubbles - you have pressure, volume, temperature, etc. If we take this problem into chemistry, you can look at the surface tensions... the bubble will form itself in a way so that the surface tension will be minimized.

KR reiterated that the spring stuff is there in the bubble software. So it seems there are two approaches if you want to do something based on springs – you can either attempt to edit the bubble software (or use it as a subroutine), or implement a much smaller amount of similar functionality yourselves.

BES mentioned that the bubble program will not compile out of the box. PG remarked that one colon should be a semicolon.

GS observed that applications of pressure in different regions would tend to make convex regions. KR concurred that this seems like a reasonable heuristic, even though it is not an algorithm yet.

PJ suggested running spring models from many different initial configurations. KR pointed out that exactly how the springs are going to form will depend very much on their initial configuration.

```
        ---------------- (1,1)
        |       |       |
        |      /        |
        |\__/--\___/---|          CASE 5: one method
        |      /        |
        |      |        |
        -----------------
(-1,-1)
```

PJ and GS wanted to minimize the total tension on all the springs, since tension is proportional to length. KR drew the diagram above with many springs on each path. RS wondered why have many springs on a path. KR responded that many springs would allow better approximation of curves, while single springs would give just straight lines. The point KR wanted to make was that one might introduce a bias if one had different numbers of springs along different paths.

A second point that KR wanted to make was that the topology of the starting configuration is important, since topology is preserved by spring length adjustment. If, instead of the picture above, we had a triangular central region with three branches to the island peritmeter, the final result would look very different.

PC said "Perhaps I am out of line here, but I am a lowly C programmer. Realistically, I wonder about the implementation of an algorithm." He proposed a solution for solving the problem for a square island with any number of residents. At each stage we partition the remaining space into two with a perpendicular slice across the shortest side of the remaining rectangle, to slice off a region of just the right size for one resident. He conjectured that this algorithm was optimal.

As conjectures go, this one didn't last long. JA observed that our simple quadrant solution for the square beat PG's algorithm for four residents. Nevertheless, the implementability of the algorithm appealed to JA, who suggested the pizza algorithm for a circular island as a similarly implementable (but not necessarily optimal) algorithm.

KR pointed out that the class has only have 2 weeks to implement this, so the class will have to consider implementability as an issue. JP liked the greedy aspect of PC's algorithm. He wondered if there is an intermediate algorithm that could be implemented. KR offered that using the evolver software is another way to achieve implementability, since much work has already been done. The bubble software has been developed over a period of 9 or 10 years.

RS wondered whether the spring approach will converge. KR thought it would. JP asked whether the points on the spring are variable or fixed? KR said they are variable, like the hooks on a shower curtain. KR thought that the resolution of spring tension will lead to curved lines.

MS wanted to try PC's approach on a square with five players, but KR wanted to move on.

PG asked whether we know anything about the NP completeness of this problem. It seems to have NP complete characteristics. KR wasn't certain, although he had some ideas. He didn't pursue a proof himself because that was the job of the class! JA wondered whether a proof that this problem is NP complete would be enough, instead of writing an implementation. KR said that a formal proof of NP-completeness would be great, but that an implementation (probably using heuristics) would still be required. JP joked that KR is under contract to solve the Yugoslavia problem... KR replied that unfortunately the solution will come too late.

KR moved on to a different kind of special case. What would you do if you had a really long and skinny island?

```
--------------------------------
|                              |
--------------------------------
```

KC wanted to start chopping vertically. PM suggested an amended strategy. For this shape, there would be a certain point at which the relative height of the pieces grows large enough such that it would be more useful to make a slice across the horizontal. The utility of this amendment was verified for 200 residents, with a length of 100 and a height of 1.

KR said "Let me throw another spanner in the works." Everybody responded with "a wrench!" What about making a bulge in the rectangle?

```
--------------------------------
|                              |
-------------/  \--------------
            /    \
            |    |
             \__/
```

Does this have local effects or global consequences ?

JA thought that the effect would be global, although for the most part the lengths of the lines will probably stay approximately the same.

BES suggested heuristics based on geometric considerations, for example, decomposing a complex shape into simpler shapes and using different approaches for different sub-shapes.

MS wondered how to choose a first place to start building fences. Some initial choices may not lead to good optimizations. KR described some potential pitfalls. For example, one might be tempted to move a straight line across a shape until the right amount of area is swept out. However, this straight line might define a disconnected area due to a concave (or zig-zagging) boundary. It might not even be possible to adjust the line (say by rotation) because the boundary is sufficiently jagged. JP wondered how bubbles will do in those jagged-boundary conditions.

KR reminded the class that, unlike most CS classes, one should not expect the instructor to tell you the algorithm to implement. The problems of this class are intentionally open-ended, and defining a good algorithm is part of the project. If the problem is too hard to do in 2 weeks, consider working on special cases. The hard problems in this class should be specialized to more tractable sub-problems, while problems with straightforward solutions should be generalized.

KR reported that SS developed a program to draw the map and the regions chosen by an algorithm. The areas of the regions are displayed. A feature to display the total perimeter will be added.

After some discussion, the class settled on the following delierable for Monday: Write a program to handle the 3-resident case for any map, and demo it on the map of australia ("oz") for 3 inhabitants.

5.3 Monday April 5, 1999

KR began the class by taking questions on details of the project. An error of 0.1% means that the area of a plot should be within 0.1% of the expected area (as opposed to being within an absolute amount of 0.001 units). The subdivisions will be judged visually, so line crossing by minute amounts due to floating point calculations will not be penalized. The drawit program will be made runnable on the HPs.

JCS had some ideas about the problem. He discussed a method of dividing the polygon by drawing a line from a vertex to an opposing edge and trying to find the proper crossing point to achieve the desired area.

```
5   2
|\ / \
| 1   \
|      \
|       \
4--------3
```

In the figure above, for example, a line would be drawn from 2 to some point between 4 and 3 until a region is created with the necessary area. Alternatively, a similar process could be used with 4 as the fixed point, and the opposite point varying along the edge between 2 and 3. Sometimes the line will cross the perimeter. In that case, he would reject that cut and look for another place to cut. Once a successful cut is found, the problem is solved recursively for one fewer residents. (KR wondered if there always exists an internal cut of the form described for arbitrary shapes.) JCS does not see a possibilty of finding an algorithm to minimize the perimeter.

JA liked this idea better than a similar idea they had been working on to cut off pieces of island. LO suggested a modification to JCS's algorithm which is to vary both endpoints, one between 2 and 3 and the other between 3 and 4, to get the right area and minize the length of that edge.

The question, "Does anyone have an input for the evolver program?" prompted a moment of silence. BES proposed forming a coalition of Evolver people to work on issues regarding the program. Several people had examined evolver, but were finding it hard to learn. They regretted that no one in the department that can assist with it. KR suggested emailing the author for hints and help.

BES wanted to come up with n areas of minimal perimeter, and then adjust the areas. KR pointed out the obvious problem, which is that a subdivision with near-minimal perimeter and no constraint on areas would be a degenerate one with many epsilon-length edges in a corner of the map. JP suggested determining the lower and upper bounds for perimeters to avoid the problems of the BES suggestion.

KR suggested another heuristic to minimize the perimeter. Divide the region in two with the smallest perimeter cut. Move cut as necessary to get the correct area. KC mentioned that if you know you have a rectangle, you can make more assumptions. He proposed pretending the island is a rectangle, and accepting the potential extra perimeter caused by the fact of that approximation.

KR showed the class another idea for an island, which resembles a jack (from the game) with three bells, like this:

```
         ***
        *   *
         * *
        / /
 *     / /
* *---  /
* *---  \
 *     \ \
        \ \
         * *
        *   *
         ***
```

To divide among three residents, a little Y can be drawn in the middle. If one of them dies or moves, then the third bell will have to be split in half, meaning some perimeter must be used down one of the long "necks". The number of residents can make a big difference in perimeter.

KR attempted to show that the perimeter is dependent on the starting configuration as well. A fourth resident can share a bell with an existing resident and then the perimeter will be bubbled out into the neck. Alternatively, the fourth resident can be placed in the center and its area will cover the tubes and part of each bell, using (probably) less fencing. JP did not see that there was a feasible method to determine a good starting configuration. He suggested using the Monte Carlo method to find find a good starting position.

KR asked if the class though the problem was so hard that they would not be able to find a solution to all four maps. He suggested finding simpler subproblems to solve. AK noted that the perimeter was the crucial aspect of the assignment and was not even the focus of the discussions. KR made it clear that that the assignment is to obtain solutions using a small amount of fencing, not necessarily to find the optimally minimum fencing. He is not sure if the problem is NP-complete or even NP-hard and leaves it for the class to determine. DK said that they should try working with convex maps only, and not those like the figure above. BES suggested placing constraints on the number of inhabitants and vertices, although JA didn't think that the number of island vertices was the right measure of the problem's complexity. The class decided on the following three subproblems to solve:

1. Divide a general map into equal areas with no concern for perimeter.
2. Divide a convex map into equal areas with minimum perimeter.
3. Divide a general map into equal areas with minimum perimeter and a small number (at most 4) of residents

Did the class have any lower level implementation details to discuss? JP said he had trouble knowing when lines fall outside of a polygon. He proposed to add the area bounded by the new line and existing lines to see if it is greater than the total area of the island. KR had a different idea. When walking along the boundary of an island, i.e., on the beach, the internal areas will always be to the left, or always be to the right, depending on the direction you start walking in. If the line is on the opposite side, then it is external.

The following deliverables are expected for Wednesday. The maps due for today should be completed and another that divides the longandskinny island into 7 regions should also be ready. Remember, presentations are next week!

5.4 Wednesday April 7, 1999

(These notes are based on notes taken by JP in the absence of SS.)

KR apologized for the error in the drawit script such that it only draws convex polygons (incorrect call in the OpenGL routines). It made it look like some students' partitions went outside the region when in fact they didn't. In the meantime, don't get too bothered if some things go outside the edge.

JP said that this will make it difficult for us to debug the code. KR agreed, and suggested we draw our own pictures to verify it doesn't go outside the boundaries. A fix will be made soon. KC suggested we make a map file out of the results and take an outline (e.g. convert the regions into a map that would be filled, and then use the script).

Groups 2 and 4 did not produce oz maps last time. Group 4 has a lot of maps, and now has them loaded into the directory. BES demonstrated it on his machine. MG asked why there are so many different maps; BES replied that it's because they are generating all possible different solutions starting with those points. Group 2 has not yet produced its deliverables.

KR asked if anyone wanted to show something for the long and skinny map for today. No volunteers. KR asked if anyone has anything to show. LO says that they have maps, although they're more of the same thing. BES explains that (unfortunately) edges sometimes cross outside the island perimeter. JP remarked that the long and skinny island broke the code quite frequently. He also mentioned that he and JCS have solved the orientation problem discussed last class.

KR asked what other problems students had with "long and skinny". BES thought they couldn't solve it by hand, so therefore it's more difficult to derive an algorithm out of it. (An empty long and skinny map appeared on the screen.) KR asked what is difficult about doing this by hand – just cut off parts off the "sausage" as you've attained 1/7th of each of the areas. MG wondered how that concept would be conceptualized into a program. BES expanded by saying that there is no way to do absolute minimization; if that is not the goal, fine, but he stipulates that there is no way to accomplish absolute minimization by hand while checking boundary conditions, polygon data, etc. LO added that it's difficult for programs (as opposed to humand) to realize the long and skinny shape, and thus apply a sausage technique.

GE wanted to think only about "real-life" islands and that since this island is not realistically shaped, real-life solutions would not be applied here. KR disputed the realism of the long and skinny island: "Why, it's a tropical Norwegian island with a lagoon and a fjord!" GE says that he would prefer to see some more common shapes, such as convex islands. KR observed that that was one of the subproblems discussed last time. He says it would be perfectly legitimate to limit oneself to simple shapes given the time constraints given in class (w/sufficient explanation). He says that discussion of what can be handled should be put into the report.

GE wanted to know whether there will be any test cases when this code is submitted. KR suggeststhat by Friday every group should email him one test case (presumably one that their solution can do well), and he will post them as test cases for everyone to try and do. He says the maps expected to be done (for both the presentation and report) will be the four in this directory plus these additional six. He suggests that this will help distinguish algorithms that would handle convex islands well, for example, as opposed to concave islands.

The class agreed to submit such maps by 5pm Friday. KR will post a message to the newsgroup shortly thereafter.

PC says that the fencing is not computed correctly by the drawit program; Peter says that with the square it was computing the perimeter incorrectly. KR promised to find and fix the bug.

MG showed an Evolver screen with (a) the original polygon, and (b) with the "equalized" areas. He didn't know how to calculate the area (PG says Shift-P followed by Option 4). PG remarked that the points are not moving along the island perimeter, and the internal boundaries are not bending. He suggested that the latter can be done, but the former is a problem as he has not seen that in the documentation. He says he cannot find the syntax to leave it free yet constrained to the outer boundary.

KR asked MG what the shape of this island is: MG says it's a trapezoid. Given the shape and the areas on the inside, it's supposed to minimize the area differences. Some people noticed that the outside seems to move. It is confirmed that it moved. PG suggested that MG do a refine operation before the relax operation in Evolver. The results looked good, with curved boundaries. PG asked if wraparound of the outside vertices can be done. MG thought it can be done.

KR asked how much effort was required with the surface evolver. MG reported that it required a lot of communication with the creator of the program, and he sent MG a basic file to help him out. PG expressed interest in seeing this same file. MG popped it up on the display. PG pointed out that you are limited to a maximum of 20 different constraints per file, so an island with 100 edge segments would not be able to be captured in this manner. KR wondered whether this limit of 20 is a recompilable constant, but PG did not know. He said it might not be one the creator intended to be modifiable.

PG wanted to mesh the previous evolver example with the Oz map. MG said that the next step was to parameterize the outsides at runtime. PG mentioned that output can then be done by Shift-P, option 4 (points output).

KR asked if any other groups had any progress using evolver. He said that we have seen a proof-of-concept here, i.e., it can be done. GS asked if using evolver requires some means of tesselating the polygons. PG said no, because evolver does this itself.

BES had talked to Prof. Grossberg about the problem to get a mathematician's view. Prof. G. said that we might want to enumerate all of the solutions possible, i.e., create the set of solutions with some line segments with varying lengths of perimeter. BES then adds from that space of solutions the minimization should be found. Mathematically speaking, we are generating a very large space.

KC ventured, however, that the set of potential solutions is infinite for this problem as little movements would create more elements in this set. He was trying to figure out a way to show NP-completeness, and the problem is that for the reduction he can't see a way to do it. He thought that the knapsack problem was a good starting point, but couldn't see a way to achieve the required reduction.

BES responded that with integers it's usually easy to show problems like this to be NP-complete, but for this problem, with real numbers for the island parameters, it will be more difficult to show NP-completeness. Prof G. had visualized the problem as energy fields over polygons, which PG thought was similar to the evolver's conceptualization. BES reported on several sources he spent some time researching for references, but without success.

PG described one of the examples in the evolver – apparently there is a famous problem of space-packing – how packing of space would be accomplished by minimizing the surface between the shapes. Apparently Lord Kelvin's solution lasted a century or so, and shows how the evolver can be used to reach a better solution for this. He adds, however, it is tough to get the periodicals for this problem.

KR says that the problem of explicitly minimizing the perimeter is probably not the way to go about the proof. KR would look for the following in an NP-completeness proof: {he adds that knapsack isn't ideal as all residents have an equal amount of space...}

> Suppose we can minimize the perimeter for any given configuration of the island. Parameters: number of residents, shape of the island.
>
> Given an instance of an NP-complete problem on some number n, map that to (island, number-of-residents) configuration so that if he achieves a certain perimeter on this island/resident solution he would have the original solution.
>
> The gist of this would be that you create a weird-shaped island, such as

```
<--                  long l                -->
***********|**********************|************ <-- epsilon thick
*  *                  *  *                 *  *
*  *                  *  *                 *  *
*  *                  *  *                 *  *
*  *                  *  *                 *  *
*  *                  *  *                 *  * h
*  *                  *  *                 *  *
*  *                  *  *                 *  *
*  *                  *  *                 *  *
*  *                  *  *                 *  *
****                  ****                 ****
                                            w
```

> so that the area for each prong is wh. And then to maximize the packing, the total perimeter is 2 epsilon for 3 residents.
>
> But then if we have four residents, you need lines someways further down, and the lower bound for the perimeter is (probably) w.

Set a threshold between $2e$ and w. Given a particular instance of the original (NP-complete) problem, construct the island and residents in a polynomial way; if we can solve the island problem, then by construction, we will have solved the NP-complete problem. The special property here is that you have a really small perimeter for three but you can construct an island as many forks as a parameter of your original NP-complete problem (e.g. bins/buckets to divide – imagine the forks as bins). If there is an efficient solution, small perimeter; else large perimeter. This is just a sketch, and not an actual construction.

KR asked the class whether they know the difference between NP-complete vs NP-hard? NP-completeness requires a reduction from a known NP-complete problem, as well as a proof of membership in NP. NP-hardness requires just the former. A problem is in NP if it can be solved *nondeterministically* in polynomial time. An intuitive way of understanding this definition is that a problem is in NP if, given a potential solution to the problem, one can decide whether the proposed solution is in fact an actual solution in polynomial time. It's not obvious whether the Egalitarian Island problem is in NP, because there's no obvious way, in general, to efficiently check a proposed solution to see whether it is optimal. Poll: about half the class has taken a course that covers NP and associated concepts.

What's required for next Monday:

- 8-minute presentations
- Pictures from our terminals OR (preferably) from the main console
- Reports (explain what you tried that didn't work as well as what did); as for NP completeness stuff, there is no expectation of an NP-hardness proof, but if you do it's welcome. Actual maps should be included, either on paper (do a screen capture?) or submit a reference to the map

BES asked KR what motivated the choice of this problem. KR was looking for a variety of kinds of problems, one being at least somewhat geometric. PG added that this is a good description of this problem. JCS adds that computer scientists hate geometry. Some in the class disputed this assertion. KR expected this to be a hard problem where some approximation would be required. Again, if you choose a problem that is too easy or too hard you can generalize or specialize as appropriate. KR thought it would be the right mix, and if it's harder it's still ok to solve special cases.

JP commented that it was more difficult to discuss this problem as it is more difficult to "toss about" geometric algorithms on-the-fly. JCS said it's hard to code this project, given fencing, tough polygons – it's becoming harder to model as the situation becomes harder to code. JHS found some code at `http://www.mike95.com` – Vector class for C/C++. In his opinion, it made life easier.

LO thought the best solution may be "more than one solution". If it's done manually by a human, different algorithms are used. In his opinion, the best thing may be to have the program recognize different map configurations. PC liked the problem; he suggests limiting the problem to convex islands, as the complexity is more than doubled for concave islands.

JCS rebuts the idea of limiting the problem to convex islands; he says it takes out some of the challenges to the problem. He cites the fact that JP and he came up with some algorithms to deal with the concavity of the problem. PC says that he ignored concave islands at first, and he and PW came up with a solution that extends to most (but not all) concave islands.

BES asked about linear programming. He wonders if a solution using Linear Programming would have been possible. (Much of the class does not know about linear programming.) KR explains that it's basically the minimization of a linear function given linear constraints; it's not inconceivable that liner programming would help, but it's not obvious how it would work.

KR then gave a quick preview of Project 5. He shows on the display a short simulation of an organism moving, replicating, etc. KR demonstrates how supporting organisms gain energy, to give some flavor of what the fifth project will be like. The project has 3 goals:

- Grow faster in isolation!
- Do well in competition with other organisms.
- Both of the above with limits placed on the organisms brain size.

These will be used to measure the success of the organism.

BES: "Cool."

5.5 Monday April 12, 1999–Presentations

KR gave the class pointers to project 5. There is a detailed description on the homepage of the organism language. The group assignments are also available.

Project 4 Presentations

The first groups to present were those that needed to use Windows NT on the machine up front.

Group 1 presented first. PC said that their first algorithm takes the center of gravity of the polygon and draws a line to the right, performing some math to find what edge it intersects. Another function figures out how far to sweep out to close a region with the desired area. It moves along to each vertex of the bounding polygon and calculates the area. It the moves back from the first vertex that yields an area greater than that desired. The algorithm does not work well for concave polygons, and had some funny results for the more complicated structures.

Their second algorithm uses a recursive subdivision technique. If there is an even number of inhabitants, it cuts the polygon in half and recurses on those two halves. If there are an odd number of inhabitants, it finds a line that creates a region with the desired area and passes the rest to the subdive. In both cases the shortest such cut is chosen. PC started to put in code for concave maps, but still had difficulty with complex maps.

MS spoke about the NP-completeness of the problem. A problem that is NP complete must be reduced to Egalitarian Island. He said that to show Egalitarian Island was in NP, one must reduce it to a decision problem, i.e., a problem that has a yes/no answer. To do this, one can add an extra parameter representing the perimeter threshold and ask whether there is a solution with perimeter less than the threshold. A proposed solution can then be verified in polynomial time. He thought that the bin packing problem (which is known to be NP-complete) could be reduced to the decision problem, but did not have time to present his result fully. KC requested that the reduction be posted to the newsgroup.

Group 4 was next. BES explained why they called it Gilligan's Egalitarian Island. They got lost, and the highest priority was to get off the island. Similarly for this project.

They were faced with the classic decision of whether to use an off-the-shelf or in-house implementation. JA explained the use of the surface evolver. He was very impressed with it, but he thought the problem was it conveniently implements the integral of the island problem, i.e., 3-dimensional concepts (areas and volumes) rather than the 2-dimensional ones. The other problem was getting the data it produced into the required form. He felt the evolver was best used in conjunction with an in-house implementation.

Not theirs, according to LO. They borrowed some ideas from JCS. The program takes an arbitrary point, forms a triangle to two other points and backtracks till the area requirement is met. With help from PG they were able to figure out the exact point needed. Unfortunately, no working solution is available, just some modern art.

BES spoke about NP-completeness. Over integers there are known reductions. Over the reals it is unkown, but he said that it is most likely NP-complete.

Group 6 followed. JP said they established the ground rules at the beginning. They took a non-surface-evolver approach because of JCS's vision. JCS spoke about the basic algorithm, implemented in C, Lisp, and Java, none of which worked. It takes a root node, forms a basic polygon, and subdivides. He tried to use an A* search to come up with a complete and optimal solution. It starts with an inside vertex and chooses two points that are permuted until a large enough polygon is found. JCS said he used the algorithm from PG to move on from that point. JP said the initial problem was completeness was not guaranteed and the program didn't divide the island correctly. Some possible optimizations were to find smaller fence sizes and to modify the subpolygon reduction. Another issue was intersection code, which AK found. For lines falling outside the boundaries, they used the concept of orientation.

Groups 2 3 and 5 needed to use drawit on the Linux OS.

KC said that group 3 had a number of ideas. His idea was to go with horizontal and vertical lines that chop off just the right amount of area. It was not good for concave polygons. Since it was not a complex algorithm, it also did not perform so well in terms of perimeter. There were many deficiencies. Another idea (not implemented) was to rotate the polygon before reapplying the algorithm. As was apparent by the display, his code did not do as well on the square as group 1, since the first vertical line chopped of 1/4 of the area rather than 1/2.

Their implemented program split the interior space orthogonally to optimize the fencing. After reviewing the drawings, KR asked what the program missed that it used so many vertical lines in Australia, when it is more economical to use some horizontal lines. KC explained that whenever there was a concavity in the vicinity, lines intersection the concavity were excluded. The necessary horizontal lines for oz would have hit concavities. GC explained GE's unimplemented idea, which evolved from the bubble idea. He would split the interior into several polygons and have bubbles push out. The idea was similar to the bubble evolver.

PG said that group 2 tried to use the evolver as much as possible. They came to the conclusion that it is beyond their understanding. RS said there is more than one way to give shapes to the evolver and if they had more time they may have been able to do more. Lines in the middle start off as straight lines and evolve to curves using the evolver. They showed the egg map and Manhattan. It was hard to get the evolver to place points properly. Right now it describes polygons as segments, but PG thought there might be a way to describe it as one segment to give to the evolver. They had trouble setting constraints to draw the inside of an island; some points ventured outside the island perimeter. PG suggested (humorously) that if the Egalitarian Island problem is to be given again for another instance of this class, that the problem be set in 3 dimensions rather than 2.

Group 5 presented last. PM said they shared everybody's frustration. The problem was hard to conceptualize. They tried to get the evolver working but had trouble. MG spent a lot of time working with it. It can't work dynamically, nor could he set constraints. JHS was in the process of breaking an island into regions and throwing them into the evolver. JHS spoke about the divide and conquer method. For every edge, determine what edges can be drawn from each endpoint, and divide in half. As the clock ticked, they realized it was difficult to generate a file for evolver, so they adapted an algorithm to chop off areas.

The numbers below represent the best perimeters that the groups' programs generated. Only groups 1 and 3 solved more than one configuration, and three groups had no figures to present.

	egg	oz	manhattan	g1square	g2a	g3bat	g6a
1	14020	44297	12337	20	12500	754	16134
2	?	-	?	20	-	-	
3	13967	70301	13233	24.16	24447	876.6	

Several additional maps, including long-and-skinny, had stumped all the groups.

The class members started making suggestions about the format of the course, in light of their experience with project 4. (Further discussion along these lines will happen on the last day of class.) Some suggested having only four projects with more time allotted for each. KR said that choosing the scope of the problem to attack, a major issue for project 4, is an issue that comes up often in research. One may start with a big problem, but find it better to cut of a smaller but tractable part to work on. RS said project 4 was deceptively hard. JCS said it was like calculus, max-this min-that, and hard to represent on a computer. Desperation induced cooperation, according to JA. JP felt the nature of groups implies teams, which implies competition. KR took a poll to see how many people felt the class is too competitive and should be more collaborative. Most thought that the balance was good. The problem with a collaborative class environment, according to MG, is that talkers drown out others. JA complimented those he worked with who don't talk much, because he found them to be great to work with.

These are the goals for Wednesday. Read the full description for project 5, including the detailed description of the organism brain language. Look at the test brain KR wrote, and either run the simulator or just look at HTML output. The project is due the last class in April.

6 Organisms

6.1 Wednesday April 14, 1999

KR announced that he signed up for a student/faculty dinner on Tuesday April 27, and extended a warm invitation to the class to join him. (Exactly who is eligible was unclear, and would be clarified later.) The cost is covered by the department.

KR tried to make sure the class has a clear idea what an organism can do for the last project. JP verified that an organism has only one turn in any action-cycle. For example, if it moves south, it does not get scanned by the next line and move again. Also, when a split occurs, the new organism does not move until the next cycle. KR confirmed these points. KR also pointed out that organism death happens after all organisms have performed their actions, so it is possible for an organism to go down to zero (or less) energy and then go back up again (via support by other organisms on the same turn) and survive.

KR also described how the simulator worked by considering one organism at a time from top-left to bottom-right. That way, there are no difficult problems of resolving conflicts such as when two organisms want to move to a single cell. JA said that Organisms could be programmed in a way that is similar to the game of Diplomacy, where all moves are resolved simultaneously. KR said there are known scenarios in Diplomacy in which the rules don't disambiguate the moves. He finds Diplomacy depressing because you end up hating everyone at the end.

All of the information about organisms, including the precise energy costs and benefits, is provided on the class web page (see Section 8.1). JA noted an interesting scenario that contradicted KR's assertion that the only way that there's a net gain of energy is when there's a support. An organism with energy 7 can split into A and B each with energy 1. An organism C that attacks A will gain five points, and A will disappear. The same organism C can then attack B, gaining 5 more points, and B disappears. C has gained 10 points while only 7 points were "used", a net gain. KR said that, nevertheless, this kind of coordination would be difficult to program. (A "fix" would be to check for organism death after each action, but KR thought that the benefits were not worth the effort of programming a frequent death check.)

JCS wondered what should be done with the one byte of memory that the organism has. KR suggested one simple possibility: 256 different strategies, one ofr each state, with appropriate transitions between them. PG and JCS thought that maybe it would be better to use only 4 bits for strategy information, and 4 bits to record something important about the environment, such as whether the organism in each direction is considered friend or foe.

Organisms can not directly identify their neighbors. PM suggested establishing a protocol, in which organisms have some sort of handshake, or sequence of events, to recognize each other. GE was worried that another organism might take advantage of a handshake to attack. MG asked if the moves required for a handshaking protocol are worth it. JA said he will use a simple protocol without memory: Any organism that supports me is a friend. PW's concern was how to set up a protocol. JP said it is difficult to keep track of a handshaking protocol because it is unknown when organisms have switched locations.

Any organism that moves north, as JA noted, is impervious to attack from the east, while one that moves south can be attacked twice from the west. This is a two-sided sword, because the organism is also impervious to support.

DK thought of using a genetic algorithm. JP attempted an explanation. Begin by generated random players. Then they reproduce by crossing their components and mutating. Then evaluate their scores on some trial to decide who makes it to the next generation. KR gave a broad framework for GA. Generate a large population of (somewhat) random programs and see which do well to move on. This is a process that provides selective breeding of programs. JA thought genetic algorithms will work well, but he, apologizing for what he considered a supremacist attitude, does not want to mutate a well functioning, precisely crafted and fine-tuned brain. KR responded that mutations or organism breeding would not have to be done randomly. Perhaps there is a way to combine chunks of organism brain corresponding to functional units.

KR warned the class to be careful not to have the initial rules apply too often, because the rules that occur later on would rarely get executed.

There are three goals in the design of an organism. One is to eventually reproduce and fill a substantial fraction of the board (say 25 percent) when on its own. KR prefers not to place limits on how fast it fills the board. He also didn't want to require organisms to fill every cell, or even most cells, since some robust strategies might naturally tend to leave certain cells empty, by biasing attacks towards those cells.

The second goal is to have the highest population when in competition with other organisms. This is a count of the organisms, not energies. The third goal is to repeat the two tests above, while placing a limit on the number of instructions, because a brain capacity is naturally limited. For each of the three goals, the class will decide later, in the light of experience with the organisms, what the thresholds should be.

The discussion turned to implement strategy while limiting the number of instructions used. MS suggested a defensive algorithm. Never attack, and run away when attacked. SS felt the two most important instructions are split and support, since no successful organism can avoid them. However, GE thought that such a determination is relative to who else is in the game. It may be in one's best interest to attack everyone. The class was quick to observe that an "attack everyone" strategy would go extinct as a result of its success. That is why a protocol might be needed to establish friends.

GS related the organisms problem to the prisoner's dilemma, which KR explained to the class. Two gangsters are arrested and taken to separate prison cells for interrogation. They are each offered deals by the prosecution. A prisoner can cooperate (C) with his friend and not spill the beans, or defect (D) by informing on his friend. The following is an example of the payoff matrix.

```
       prisoner 1
2 |       C      D
-------------------
C |      1,1     0,5
D |      5,0     4,4
```

If both cooperate, then they each get a 1 year sentence (for lack of strong evidence). If both defect, they get 4 years each. If one defects and the other cooperates, the cooperator gets 5 years and the defector 0. The prosecutor tells both it is in their best interest to snitch: No matter what the other guy does, you're better off if you defect.

Our problem is similar to a variant of the problem known as the iterated prisoners dilemma. Repeat the prisoner's dilemma many times, with both partners able to remember the entire history of the interaction. There is an interesting article worth reading by Axelrod which suggests strategies. The reference can be found on the web and has been posted to the newsgroup by JP. One strategy is a tit-for-tat maneuver: do what the other guy did last time. That the problem applies to organisms can be demonstrated with the following matrix:

```
        organism 1
2 |      S       A
---------------------
S |      +1,+1   +6,-7
A |      -7,+6   -2,-2
```

It seems better to attack, no matter what the other organism does, but iterated may times, there may be better actions. KC suggested using a mixed strategy. Set probablities for the moves based on the expected payoff for player 2.

JCS thought this is the like Borg from Star Trek. An organism can be in protocol with several neighbors, and evenually they will form a blob and respond robustly to outsiders. He wondered whether an organism could signal for help.

PJ asked if an organism knows when it killed another organism. According to KR, they will simply disappear from the neighboring cell, which can mean that they either moved or died.

JP picked up on JCS's idea. He suggested creating a phalanx. Create an organism that does not take over immediately, but strengthens itself. KR said it is like a cancer, which grows (fast) locally and grows bigger.

AK said to split early. KR asked if it is better to split early, or to keep an energy reserve and split only a little bit? GS said to split as soon as possible, depending on size, except on small boards. JA went out on a limb and said it is always better to split. The more organisms there are, the more supporting (and attacking) that can happen.

MS asked what do in the absence of a protocol. BES said to propagate forever and rely on locality. Organisms in the vicinity are likely to be clones and should be supported. PJ proposed a simple altruistic strategy: Never attack, support neighbors, split. Since they never attack, they will eventually fill the board.

MG suggested a snaking protocol. Have a head organism, and allow the others to follow. Reproduce at the tail by splitting occasionally. Orient all supports towards the tail of the snake. KR said it is a risk. If a snake only goes straight, it will only fill one row. It must actually snake around the board. JP generalized the example to a congo line.

KR pointed out that straight congo lines travelling west could be compact, with no gaps between organisms, since the westward organism gets processed first. However, eastward travelling congo lines need gaps (or will move more slowly) because the eastward organism gets processed last. This makes a big difference to support actions, because support requires adjacency. MG proposed a congo snake, one that splits itself. JP likened a congo snake to a centipede from the video game. It splits when shot in the middle.

PC suggested starting in a central location, expanding outwards, and using state bits to have a notion of distance from the center. The edges should have higher energy and be more willing to attack, and the central organisms should reproduce. KR thought that the opposite would generally happen. An cluster's center has higher energy, because it has more neighbors that are likely to support it, and reproduces less because it is surrounded. BES likened it to the Roman empire, which spread out and supported the outside regions.

For Monday, each group should program an organism that has a behavior they can talk about.

6.2 Monday April 19, 1999

The class had noticed the organism simulator sometimes caused a segmentation fault. KR had begun looking for the problem, and said he'd announce to the newsgroup when he'd fixed it. KR observed that students had made progress on writing brains.

KC commented on his experiences so far with the simulator. He had written a brain he called "mad cow disease". KC ran his brain against GE's brain (called "psycho"): psycho went nuts and destroyed mad cow disease. KC's algorithm tries to take over the board and rid it of opponents. Madcow doesn't defeat psycho because psycho builds tough cores and kamikazes Madcow with offshoot organisms. GE found it odd because his organism usually looks weak. Psycho walks randomly around the board and attacks wildly when it finds something.

Anthrax, a modification of madcow, starts off better than psycho. After 90 cycles in a simulation, psycho had much success, while anthrax began to shrink. Finally, at 400 cycles, psycho obliterated the remaining organisms. When asked, GE was not sure what strategy would be good against psycho. He said that the only time one psycho organism supports another is in an initial nurture phase, when an organism first splits. Then it goes off and attacks wildly.

The following quote from GE is excerpted from the newsgroup:

> "I tried playing against BES's Frans and the results were quite interesting. In the short-term, Frans would build up a much smaller, yet much stronger, close-knit population. psycho would just spread out all over the place with a bunch of semi-weak organisms. In the long-run, psycho would kick frans' ass, eventually obliterating frans entirely in every trial I did. Take a look at the stats, and you'll notice that psycho is INCREDIBLY more volatile than frans: Deaths, births, and splits can be easily 10 times as much, and attacks can be... well let's just say it's not called psycho for nothing. :)"

JA added that his results correllate with KC's observations. JA showed results that his brain "Support-Attack" did in fact put down psycho. The three main elements of his strategy are to attack if not attacked, support if supported, and split at certain points where the energy level is greater than 6. He observed that "mummy", a brain written by GS, performs better in the beginning. After 1000 iterations it cleared out SupportAttack. KR said that an organism that only supports will never eliminate another organism unless the other organism self-destructs, to which MG responded by saying, "That's what we do."

Could the class deduce any principles based on the results? JCS said it doesn't seem to matter how many cells an organism occupies. JP asked if anyone ever played boulder-dash? There is a creature called an amoeba that grows slowly, and the player's job is to contain it. If it is not contained, it turns into a rock. It seems in these simulations that an organism which quietly grows and builds up energy can be dangerous. RS said that it is good to surround others. AK said that the random numbers can have a big effect on the outcome.

In a competition between mummy and AK's brain called "virus", a rectilinear pattern emerged, with one organism compeletely filling six columns, and the other four columns. GS said that mummy moves east and south. If the eastward moving organisms are killed, then the rest do not try to invade other columns.

In response to a question from JA, KR asked the class who thought it would be interesting for a child to retain the state of a parent after a split? PG thought it is a good idea because he can plan different strategies, rather than having every new child start in the same default state. On the other hand, PG thought it would be frustrating to not know who is the child if the parent and child looked the same.

How does a child organism realize that it's just been born as a result of a split given that it must start in the default state? GS found that by splitting only at certain energy levels, he could use a test on the energy as a surrogate for state information. He admitte, though, that the energy information was vulnerable if the new organism was immediately attacked or supported. RS suggested "slapping" the child by attacking to let it know who is the parent. JP said that may not work because of the order in which organisms are evaluated. Another student suggested that propogating down and to the right might avoid JP's objection. KR was reluctant to change the simulator because groups have already determined some of their strategies.

GS wanted to know if anyone wrote an organism that moves? It was already established that psycho does. The purpose in that, GE explained, was to hunt and attack other organisms. Also, once nurturing is complete, a parent and child may attack each other if they don't move apart. Most simulations were run on a 10x10 board, so JA asked if anyone ran simulations on different size boards. JHS tried it on a 25x25 board and his brain ("herpes") performed well. JA thought that maybe his own brain would do poorly on a small board because the fringes die out quickly.

Everyone was focused on the the second goal of dominating the opponent, but KR wanted to know if the class took the first goal (being able to populate a substantial fraction of the environment) for granted? Are they sure they will succeed at the first goal? KR initially defined success as filling 25% of the board, but thought that maybe long term survival is good enough. DK suggested that the first goal will be achieved by almost everyone. He felt that most of the work is in achieving the other goals.

GE said a virus needs others to live, so maybe it won't do well. PJ said there are two different goals that may be in conflict with each other. It is easy to program a brain that will populate the board, but difficult to write one that will survive in the presence of others. That should be the primary focus. On the other hand, KC said it is a natural goal to take up all the space.

DK suggested style points for how the organisms behave. Due to lack of enthusiasm, and subjectivity inherent in any notion of style, there will be no style competition. However, the presentations and reports will report on any interesting developments.

KR will only change the organisms code to fix bugs. JP is working on a genetic algorithm, but either needs something to evaluate fitness or a mini-parser to pull out the values needed. He said there are two different approaches to a genetic algorithm. One is using the platform GCP++. He is working on a program that will OR lines together. He says there is some cool code available, but it is very generalized and hard to adapt. KR said that students can post any modifications they want to make to the organisms program, but are responsible for fixing the bugs in their versions on their own.

BES ran a large run on everybody's "brains" using a 20x20 grid. Several brains didn't attack. By 1200 runs, virus died off.

How does the class define success for the secound round competition? Should the program run till kingdom come? JA said to run it till a fixed round number and then count the energy levels. He had support from PW, who said a small blob withh high energy can eventually spread out. High energy translates into potential. This suggested energy criterion was different from the original criterion that measured just organism populations.

SS is creating a directory to store the brains. To ensure a brain remains stable (not modified) during a simulation, it should be copied into a local directory.

The process of writing out brain code and working out the number for position and states can be tedious. KR asked whether anyone thought of writing a compiler that produces brain code? JA said that cpp and M4 can run macros to do a lot of the tedious work.

In regards to genetic algorithms, JHS thought it would be nice to write a C program that extracts everything. It is too big a space to start evolving truly random organisms. JP said to take one algorithm that does well and let it perturb, not mutate. The problem is that the number of choices can become vast very soon. It is important to choose the right parameters for optimization. One problem for JHS is the crossover function, since the order of rules makes a difference. He thought of writing a program to figure out how to order the rules most efficiently. KR pointed out that brains can be interleaved and the resulting code "makes sense" in that the instruction executed would be one that one of the component organisms would have executed.

More simulations were run on varying size boards. "TitForTat" by JA seems to do well on a 25x25 board.

Mummy and niceguy (by GS) do well. JA said he added 2 more rules to his base 3. Support in directions that children are known to exist, and then where children are believed to exist.

GS said that if you are running peaceful strategies, you must hold grudges. JA agreed to some extent, but suggested eventually turning off attacking, because otherwise you will kill yourself off. To avoid that, GS said to expand quickly.

BES wanted to know if there is a way to set up the initial state? KR said no, but a random seed can be supplied to the generator, so that if the initial position is not desired (for example, two organisms start right next to each other) one can re-run with another seed. BES was more ambitious. He wanted to be able to configure organisms like a war game, setting up armies covering certain terrains, and then to program their behavior. Many games come with editors, and if organisms is rewritten, JA said it could be done like a java applet, with drag-and-drop capabilities.

JP wondered if an organism can be encoded that conforms to rules of game of life. In the Game of Life, an organism has 8 neighbors, dies if not surrounded by the right number of neighbors, remains alive if it does, and a new organism appears in an empty square if it is surrounded by the magic number of other organisms. The class quickly realized that many of the features of Life could not be simulated.

KR wanted to know what directions the class planned to take during the next few days? JP and PG will work on a better model for programs to adapt to genetic programming. JA asked, seemingly out of context, if it is worthwhile to be offensive in an environment by yourself? His initial results say it is bad, but maybe because his organism split a lot and killed itself. KC said that if it can determine who is an opponent, then attacking is good.

KC observed a diagonal of empty squares in one simulation and explained that it had to do with how the organism moved. KR thought that these kinds of macroscopic patterns were interesting, and encouraged the class to look for them and to come up with explanations based on the brain code.

KR established the goals for this Wednesday. He would like to have an informal mini tournament, if possible. BES suggested picking each group's four best brains and enter them into the tournament. KR did not want to fix the tournament structure, like board size and number of opponents, but KC disagreed, saying that some organisms perform better with only one opponent, on small boards, etc. KR said we'd try to look at a variety of configurations: crowded/sparse, square/rectangular (as suggested by MG), big/small. Everyone will select their four best brains and place them in SS's directory. Class members can perform some simulations in advance if they like.

6.3 Wednesday April 21, 1999

PG's general strategy in designing a brain was to try different energy level thresholds (for example, the threshold at which a split happens) to see which performed best. He usually jumps straight to the final map to see how he did versus the opponent. Whichever threshold gave the best results was the one he chose to use.

JCS said movement is not much of an issue in terms of memory (retaining the board state). They are working on the 007 method against TitForTat. Since TitForTat (the program that seems to do the best overall, so far) responds positively to support, they take advantage by cooperating and trying to grow rapidly. Their brain also tries to impersonate other organism's protocols to get an advantage.

MS is trying to randomly print out all combinations of the rules by mixing conditions and actions. Then they will pick the "best" third according to some (not very clear) criteria. Because arbitrary rules will not make any sense, there are some constraints, but they are still fairly random. JP said that the brains can be so inherently complex that such broad-based genetic programming will be a very difficult process. His group thought of working more on small mutations instead. KR said that those doing evolution of organisms could have to run the evolution for millions of generations for good results because mutations are mostly for the worse. PG suggested starting with some working brains first, because random brains will not mutate very well.

BES is still striving to create a good smaller brain. He wrote one with about 8 lines, and it usually comes in third or fourth against the better performing brains. JCS wanted to know the best minimum threshold for splitting. JA said their brain splits at the minimum level, which is 7. PG said that they were following a nuturing strategy which used a higher level threshold of about 12.

KR then demonstrated an organism he wrote called "shag". Shag is an acronym for "support, hold a grudge." KR noticed that mummy was not splitting rapidly, so he put in a bias to do so more quickly. His organism branches out orthogonally as it grows in order to fill the space more quickly. If an organism has three vacant neighboring cells, it chooses to split into the middle one of the three, leading to more branch-like development.

Nevertheless, in a simulation against tittat, it seemed that tittat spread better, both by the 50th run and 150th run. JA (tittat's author) theorized why. If an L is formed, then a child receives two supports and splits faster. PJ suggested that board position has an effect on the win. KR has a threshold for the internal organisms so they split at a higher energy threshold. This is because he wanted to make sure that the peripheral organisms were supported sufficiently to reproduce themselves first.

JCS came up with the the "spawn theory." Send out a small organism with enough energy that it will reproduce in a new region and form a new blob. KR observed that this strategy might do well because the perimeter-to-area ratio for two blobs would be bigger than one blob of double the size.

DK's satan reproduces slowly so it may be overrun by others. It is a compromise between psycho and tittat. In a game against tittat, it did poorly. AK said that middle cells have so much energy while growing that the fringes should be pushed out so the middle cells can reproduce. KR suggested putting in a threshold so the more energetic organisms move.

In tittat vs. mummy3, mummy3 was doing better. JA said that by the 400th round, they were still fighting, but was sure that the weak tittat's would die.

KR went ahead to run a big simulation with all of the brains participating. It seems the performance depends on next to whom one starts. KR explained that the basic defensive strategy of shag is to attack in the direction it was attacked and forever pursue anything in that direction, theorizing that the opponent came from that direction. (That's where the grudge part of shag's name comes from.)

PJ suggested coming up with formulas describing how fast, in principle, any organism can expand. The class seemed to think it was exponential. With a formula, one could then assess how rapidly other organisms were doing compared with the best possible rate.

MS suggested interweaving all the strategies. KR told the class to look at the blend program in his directory which scrambles a number of brains together by interleaving the instructions in some weighted fashion. KR's experience, thoughm was that the scrambled brains do not work; they die out pretty quickly. JCS suggested to taking chunks of rules at a time, one for each state. KR agreed this would be better than his naive blend program, but that nevertheless the states of the two organisms are likely to be used so differently that the automaton formed by combining the sets of rules would probably yield degenerate behavior. KR likened the situation to Buridan's ass. A donkey stands between two piles of straw, not deciding from which to eat, and starves to death. JP said that if one brain uses only 16 states and the another one only uses 16, then put them together by offsetting one of them so they don't overlap. Even in this case, though, it is hard to imagine a good way to coordinate behavior between the two suborganisms.

PJ wondered whether the parameters (costs of actions) can be set differently to see how that affects brain behavior. KR though that this could be an interesting exercise, and that the source code could easily be so modified. However, since the parameters so clearly influence the way the class has written their brains so far, we won't consider other parameter settings for the competitions.

JP volunteered to rewrite the code in Java. He wanted to do this in order to be able to run large numbers of simulations within his genetic programming environment. KR described an extension of such an organism simulator as an applet, with the parameters adjustable before any simulation.

On Monday, the rules will be set for the tournaments. PG said this is a cool project. There is always immediate feedback from the simulator. JA contrasted the immediate feedback with Egalitarian Island. Writing and testing programs for that project was similiar to writing in Prolog, which, to JA's perpetual annoyance, always seemed to tell him "No."

Are there ideas for tournament guidelines? JP said to try an odd board configuration, like 30x7. A few experiments were tried on the spot. Tittat was beating mummy and mummy3 on a 30x7 and 30x2 board. KR added that shag wouldn't be included in the tournaments.

Remember to place working code in the brain repository.

6.4 Monday April 26, 1999

Five students have signed up for the department dinner with KR. Master's students are not actually allowed to sign up, but KR won't be checking. Meet at Tomo's at 6:30.

Tournaments will be run between today and tomorrow so results will (hopefully) be available this time tomorrow. Try to come up with smaller brains for the small brain categories. KR has a 3 line brain that fills the board. Small programs may expand very quickly but may not attack effectively, and be vulnerable to other attackers. JCS suggested separating aggressive and passive brains in the tournaments. There was little support for this proposal, however.

AK implemented his concept of moving the outer cells so the inner cells will have room to expand. PG noticed the common idea of a hop and spread strategy. That is, organisms, hop, split, and then expand. That might give more rapid expansion since the intermediate gaps are easier to fill by the inner organisms later.

In GE's experience, moving doesn't help. He implemented an idea for the purpose of beating titfortat that multiplies faster. Edge organisms that have low energy will step a little to the side to allow middle ones to split. GS said to split 1 energy level from minimum needed to split effectively, and use that extra point to move out one cell. This way the organism expands faster. JHS came up with two tweaks. One is to support only children. If that square is empty, then support a random neighbor. He also attempts to push energy outwards. If there is one neighbor, then a splitter splits away from the neighbor. His brain is named gherpes, for great herpes. KR incidentally requested that vulgar words be removed from comments in the brain code.

GE had a strategy that if there is an organism on the left and right, and the one on the left moves, then split to the left. After a simulation was run, GE pointed out that there was a large number of organisms with low energy count. In the long run, it is a win because there is a higher overall energy. By 400 cycles, his brain called "the pope" beat titfortat. In another simulation, JHS's g5_combined, which is the pope and herepes combined, seemed to perform similarly to "the pope." It splits with random supporting of the outer edge. g5_combined kept winning against McBain, which is AK's earlier described implementation. In one case, McBain spread too quickly and left all its organisms isolated so that they couldn't support one another. (Only after the competing organism reached them did they expand further, because the competing organism provided the necessary support!) In another competition with other brains, a rectangle was left unclaimed because it was surrounded by organisms that did not expand in the necessary directions.

The class discussed how to have an attacking organism in four lines. KC has a 7 liner that attacks pretty well, but could not make one in less. The essential problem is that an organism will end up attacking itself. g3-gout-7lines was run against a few of the other 3 liners and began to destroy them all. It only attacks in two directions, and spreads in two others.

Has the class focused on small boards? JHS said that mummy does well on small boards, because it performs well if it attacks before another is fully developed.

The main CS file server went down at that moment, and that was the end of the simulations, as well as computerized notes.

The class discussed what types of simulations to run. In brief, the following was agreed upon: Competitions among the brains with more than ten lines, those with 10 or less, those with 7 or less, and those with 4 or less. For each of those categories, there will be competitions on 12x12 boards, 7x7 boards, and 25x25 boards. The class did not think that rectangular boards would contribute much, with the exception of RS, whose brain functions differntly on those sized boards. JP suggested that groups run their own simulations to illustrate certain features, and include those in the report. For each board size, there will be pairwise competitions and group-wsie, with 4 organisms at a time on the 7x7 board, 7 on the 12x12, and 9 on the 25x25. Finally, there will be a super competition between all organisms on a 50x50 size board.

6.5 Wednesday April 28, 1999–Presentations

The results of the tournaments have come in, some of them late last night. KC wanted to see how his ten-liner performed in the big tournament, but the competition was limited to only greater than 10 line brains because the simulator would not handle more than 30 brains. JCS proposed another t-shirt, but no one was enthusiastic. JCS gave KR the Lollipop Award for the professor who codes. The students were impressed with the professor's coding abilities.

JA kicked off group 1's presentation by saying proudly "We wrote tittat." He said that many games can be reduced to the organisms game and their principles applied. Their main guiding principle was "spreading speed is the key." He wrote the brain for an organism that can split as fast and wide as possible. Since he did that at the beginning, everyone learned from it, and planned their strategies around it and it did not do as well in the end. Moving at the beginning is not beneficial because an organism loses energy and the association of its peers. A star formation is good because it allows four splits. Tittat is at a disadvantage if it starts close to a different brain. It splits fast, and if the other attacks, it gets eliminated. GC programmed a brain to first check for an attack, and attack back twice or go into a reproduction phase if there was no attack. Then it goes into a support phase, which follows a series of splits. There was a variant of tittat called tattat that attacked back when attacked on the fringes. If it couldn't support outwards, it would support inwards to build the core instead of randomly supporting. Their brains averaged fourth place. How closely did their final brains match the tit-for-tat policy motivated by Axelrod's description of the iterated prisoner's dilemma? At first, there was a close resemblance because it attacked when attacked, and then went through a growth phase. The later refinements didn't resemble it as much. For example, attacking twice is not exactly tit for tat.

JP said that group 2 took two approaches to the brain writing task. There were hand tuned brains and computer generated brains. BES stated that a total of 16 brains were submitted. He described the hand tuned brains. He specialized in the small brain categories. The "Dinosaur" series had varying sizes with no great improvements. They were designed for splitting and attacking. The small brains didn't attack. He was impressed with one 3 liner that supports well enough to fill the whole board. Since their small brains did not attack, they were all wiped out by group 3 brains in a competition.

PJ worked on two brains, based on the tit-for-tat strategy. Each had a little twist. Splitter splits and runs away from a fight. It attacks, but instead of pursuing like shag, it retreats for support. Isolationist tries to only support itself. On big boards, it would get support from other organisms without supporting back, which gave it more energy. The problem with the brains is that they were usually small when attacked, and hence, killed.

JP described the computer generated brains. He tried to use a package called GBJPP. The difficulty is the dependency of lines on each other. He had to rewrite the simulator to handle many games so he could run the generation process. The rewrite was not completed until today. JP applauded KR for his work on the simulator, seeing how difficult it was to write.

BES concluded by saying that they were successful at providing lunch for other brains.

JCS introduced group 3. He explained the complexity of the problem. There are two stated goals–fill the board and eliminate foreign organisms. The questions are then, can you identify the foreigners and implement a strategy for defeating them? KC worked on the small brains. The all need to split in two directions. The three liners split in two directions and support. He ran a quick simulation of the 10 liner against the larger brains and it won against them. The mummy strategy, which is GS's implementation, does support in a ring formation. If one is not supported, it knows the previous organism was attacked and goes back to support.

GS said that mummy3 won 7x7 and 12x12 tournaments. It was complicated to assign roles to the different cells so they interact differently. That is why originally they wanted to do something like titfortat, which has a general strategy.

Group 4 was up next. MG described marge, which splits in four directions and attacks. He didn't fully implement the snaking idea. He realized that it is not very effective because it doesn't expand very fast. LO wrote one that split slowly. He realized that it was too conservative, so he decided to move first and then split instead of splitting right away. He saw that the organisms eventually fell into equilibrium and thought of taking advantage. His organism killed itself instead.

PG wanted to implement the genetic programming approach by mutating brains. He created more brains than others in group, but did not submit any. They would mutate and play against their parents. He came to the conclusion that he would probably have to write 10's of thousands of brains to have positive results.

Next up was group 5. MS first started with random brain lines, but they did not work well. He chose to randomize and interleave line by line. He didn't do so well with more than five lines. He also ran it against herpes, but it still didn't do well. Five lines were the maximum for okay performance.

GE said that genetic programming was baloney for him. He thought it would require so many evolutionary iterations that it could not do well. He wanted to beat titfortat. Titfortat's advantage is that it multiplies fast. He realized that it gains more energy from others in a given turn and can grow beyond belief. "The pope" takes advantage of the same principle, but it a more detailed way. He drew diagrams to explain the functioning. There is internal support until an energy level threshold is reached. After six turns, he has the same number of organisms as titfortat but with three units more energy. In smaller boards, mummy3 beat pope, because it didn't have time to multiply.

JHS found titfortat a good theme. It mimics the prisoner's dilemma. It could be like the iterated prisoner's dilemma if the is only atack and support. Since there is interaction between species, though, it is not as simple. He started with titfortat. It was weak because it did not split fast enough. He used two heuristics to modify titfortat. He would rather support himself than opposition. The only way to know for certain that a neighbor is friendly is by giving birth to it. The other heuristic is to branch out, not evolve as an oval. There is a greater chance of being surrounded if it forms a dense cluster. He showed a simulation in which it cut off different directions on the board so no organisms could cross that border. As long as it does not play an aggressive attacker, it does well. He also had different versions of child support.

Finally, there was group 6. RS wrote “cock” (cooperate or continuously kill). He had two main ideas. Cooperation should work, because many organisms will be helping. Attack has the opposite effect. Similar to MG, he chose to expand lengthwise. It grows and supports horizontally. He found it worked well on 15x15 boards with 6 players. Continuously killing followed KR’s shag strategy of persistently attacking in the direction an attack came from.

PW worked on two simple algorithms. One was like MG’s idea of a chain. “Wednesday” tries to wrap around the board as fast as possible to support itself, and attacks at the endpoints to complete the chain. He thinks of it as a callous, because it froms a barrier. “Thursday” has four organism support each other in a cylce. Then they split apart and each forms another 4-cluster. It was fun to see how it worked, although PW admitted it wasn’t competitive because it didn’t multiply fast enough.

DK described satan. It employs safe strategies, like support and attack only if attacked. He want to mix it with riskier strategies, which move and attack. A satan cluster builds up the cells, reaches a threshold, and sends out emissaries with a certain health level to eliminate other organisms. It seems the best strategy is to split and grow. Scabies would try to cover more ground first. Rabies tried to stay close to home. Movement is not a good strategy because it wastes energy and leaves exposure. He spoke briefly on genetic programming. It seems like the ideal approach. A proper domain needs to be established. There are so many rules and branches that it seems hard to do.

KR accepted comments about the project. RS said to turn organisms into a class. As a biology major, he though that biology students would be interested. PG liked the fast turnaround time between editing a brain, running it, and seeing the results. KR got the idea for organisms from a project in the Stanford class called ants. That project has similar features, with ants able to lay and sense scent; the goal for the ants was to seek food and return to the nest with it. KR wrote the code for that one, too, on a MacIntosh.

7 Looking Back

7.1 Monday May 3, 1999: Final comments

Today's class is a look back on the course. The discussion is valuable in itself, and serves as a communal assessment of what was learned in the course.

The opening question is: What did you learn in this class? JP said that principles they learned in software engineering were applied in this class, such as setting up timelines, implementing methods they know will work, and not being overambitious. PG said the constant reforming of groups did not allow for elaborate overhead. The groups had to find different ways of working with each other.

Did anyone find the class like research? RS said since the projects were open ended, it was like research, but he always viewed research as taking a longer time. KC said it resembles research on a smaller scale. GE said that unlike research he'd been exposed to, planning was not a big part. This class was intended to approach research, according JP, but is not a real research class.

This class showed that group work can work in a class environment, according to JP, and suggested that this approach can be applied to other classes. PG pointed out that this is a self-selected group as opposed to other classes. It can work in general, according to GE. Do any students think this approach should be applied to other classes? JP asked "realistically" or "ideally"? At PG's request, KR clarified the question as refering to discussion and cooperation, as opposed to strict regulations about sharing solutions with others. DK thinks most classes should use this approach, but the sheer size of the classes inhibit that. PM said the problems were also more open ended in this class than others. In other classes, not everyone would learn by cooperation on the same solution. JP said that in the lower level courses it would not work, but for the advanced classes it would. The majority of the class agreed that 4000 and 6000 level courses should be of this style. MG said it makes people learn, because in other classes they can get away without learning anything until the exams. DK said that group work was only a small part of his overall experience. JHS said it wouldn't work in a class like data structures because there is no discussion about what a linked list is, or other similarly concrete ideas. BES agreed. In this class, the groups could split up and try different approaches. GC said that no curriculum starts with a colloquium. The same goes for the approach used in this class. JP said that the dynamics of the class leads to the openness. The search for a solution as opposed to the pressure to only write working code improves discussion.

What are the one or two most interesting technical aspects learned on the five projects? GE said that toetjes was most extraordinary in that it seemed so simple but turned out to be so complex. RS said that learning that the problems existed was cool enough. PG liked how LO's approach to maps was so off the wall yet turned out to be the best. Also the constraints of the brains were interesting. Battlesplat most stuck in KC's mind. They spent so much time talking about the snaking algorithm until they tried to code it and found out it was difficult. GC saw similarities between toetjes, battlesplat, and organisms, in the interaction of the players. Randomness was sometimes the key to a good effort. LO's most feared problem was egalitarian island, but on reflection it was beneficial for him to face his fear of calculus. BES said they could all come up with good solutions to apparently NP complete problems. GS thought it was interesting to move cities around in the maps project.

PC thought he learned a lot more about people in this class. He said people melt from project to project. Their styles and interaction changed from problem to problem. Once they coded all night, another time they planned on a white board. He gained insight into how he likes to work. PM saw similarities between all of the problems. JP found it scary how many of the algorithms they learned during their first three years of school they applied in class. Also, they learned to integrate formal methods with lower level implementation. The challenge for JHS was how to split up the projects. It was also useful to present every two weeks. He hadn't done that in college in CS before. MG found the experience humbling. Some of the seemingly simple ideas were in fact very difficult. AK agreed with JHS. Everyone was in charge of one aspect of maps, so each had to come through. MS was always thinking about how well the simulators were doing. He wondered if more runs would change performance, and wanted to do more testing. He felt that probability suggested he would do better over more runs. It was good for DK to use things he hadn't learned in a while. JCS liked the ability for members to throw out everything classic about the problem and come up with something creative and off the wall. PW saw that sometimes the most basic and intuitive solutions did the best, like in battlesplat.

How accurate was their initial impression of the class? JCS said this class was, as claimed, not like about any other class he had taken. He said that it is kind of like a research project, in which the professor gives a problem and there is no set solution. Everyone was working toward the same goal but not following the same route. He felt he got a hint of that from the description. JP was surprised that the projects turned out as they did; he had thought the problems would be too hard. DK said the web page pretty much described everything that would happen. He was apprehensive because he never spoke in class during the last two years. But he managed to overcome this fear, and did participate in class. He also enjoyed having real contact with other CS majors.

MS pointed out the problem that this cannot be a CVN class. His idea that maybe in ten years it could be taught with teleconferencing. GS remembers the day he walked in. He was amazed. He saw smiles on faces. It was inclusive. It met his expectations. PW was interested in the class because of the small size of the class. He was disappointed when he found out that it was taught in the CLIC lab because it is not good for conversations. There was some disagreement about the benefits of the CLIC lab. A vote showed a majority (15 to 5) thought the benefits outweighed the drawbacks. JP agreed that this room is not good for conversations. A computing environment should be available, and maybe something between 251 (with recessed screens) and CLIC. PG said that having computers in the room was important during organisms because people changed their programs during class. GE liked the way the room was not a formal setting. He saw the class more as an informal gathering. RS listed all the things he liked about the room, at which point KR interjected, "But you wanted room service!"

JP suggested lowering the students' screens six inches. GS thought it was nice to be able to work on something while the professor is talking. BES would have preferred a circular room. JP pointed out that attention span is very short. The ability to shift attention between the class discussion and working at a terminal was helpful. BES disagreed, saying that it took away from the discussion. SS suggested switching between discussion and coding sessions. GE said the availability of computers in class was vital for organisms.

Did the class teach students how to think about computer science problems? JP said they did not have new ways of thinking, but they organized what they new. PG said this is the first class he's been in with a set of unsolvable problems, which required a different approach. PW doesn't think he learned how to think, but he was able to contribute. LO thought it was interesting to see how other people think. It led him to think in other ways. GC said the typical CS class spends time setting up axioms and formulas in a predetermined way that obstructs the path to the interesting problems. That did not happen in this class. He also liked that KR did not comprehensively plan the structure of the class hours in advance or lead the discussion in a very directed way. BES liked how KR presented the Stanford algorithm in the second discussion and led students on a discussion. JP said the clearly defined problems and organisation was essential to the success of the class.

KR thought that writing a textbook for this class might be something he'd like to do. Would a textbook be useful in this class? There was a resounding no. GE said maybe a journal would be good. KR said to call it a "companion book". LO said it might be good to read about related experiences. Also, in this class, not everyone understood what NP-completeness meant. Having a quick reference to such concepts would have helped. MG said it depends on the purpose of the class: either to make progress with the problems or to learn what others have done. A textbook would allow one to progress in the problems by building on what others have done, assuming that similar problems were set. JP said the course (or any course for that matter) should not be taught out of the book. He would see it more of a reference. AK thinks it would have been nice to see what another group came up with before starting the class. (AK admitted, though, to not having read the Stanford report about toetjes.)

PC thought that for each problem, everyone has to go through all of the phases of learning about a problem to really experience it. A companion book might have helped, but students would still have had to go through all the cases. BES said that groups of 3 were better than groups of 4. With 3, you need a strong leader and division of work. BES also suggested a stronger emphasis on writeups. More emphasis on polished writeups would have led them into the research world. PG disagreed, saying that it would have been too much work for a 3 credit course.

JP brought up the issue of the number of projects. Having to switch projects gave them a breath of fresh air. KR got the number five from the Stanford class. He wanted to cover different areas of CS.

JCS said it was important to work in groups, but it didn't always work. The first assignment should maybe involve assigning roles. There was little agreement with JCS on this issue. KR's opinion is that it would not be good because it could become a blinkering device wehre students don't explore beyond their immediate responsibilities. KR suggested that the group dynamic can be set by each group for itself. Very few in the class would have liked to set aside time to discuss group dynamics. JP doesn't think it would be a necessary part of the class.

Are there final words to sum up the class? For some, this was their last Columbia class. MS wanted the class videotaped. KR went part of the way by recording the notes, which he will compile into a technical report. He does not feel that video recording is feasible. MS thought it would be nice for CVN students to see how the class works. BES said t-shirts were a nice addition. KR enjoyed teaching the class, and more so than databases (the other class he taught this academic year). He thinks he prepared more for this class, but it seemed like less because he enjoyed it. JCS complimented him again for his coding on the last project. Thanks to SS for taking the notes and writing some of the code. GE wants to know if the class can be taken twice. KR did not have an answer for that one. Thank you to all for making this class fun to teach. Good luck to all that are graduating.

8 Supplementary Material

8.1 Organisms Detailed Description

Here is a sample line from an organism brain:

```
n 0 n 1 = 8 > 20 n 0 n 0 * s w a 255
```

The first 13 values are part of the *condition* of this rule, i.e., tests on the current configuration to see whether this rule applies. The final 4 values form the *action* of the rule, determining what the organism does if this rule is chosen.

This particular rule says (reading from left to right): "Regardless of my initial state, if there is no organism to my west, and a random number is equal to 8, and my energy is more than 20 units, regardless of whether I was attacked on the previous round, and regardless of whether I was supported on the previous round, and regardless of whether my previous move was successful, split west and retain the same state I started in."

Breaking commands down into their components:

State The first two values have the form `op num`. `op` can be one of $\{=, <, >, a, n\}$. `num` can be any number in the range 0..255. The first three operators test for equality or inequality of the organism state when compared with `num`. So "< 100" succeeds when the organism's state is in the range 0..99. The "a" operator is a binary AND. So "a 4" will succeed if the third bit of the organism's state is set. "n" is a NAND, with "n 4" succeeding if the third bit of the organism's state is not set.

Neighbors The next two values have the form `op num`. They check the neighborhood of the organism for other organisms. The neighborhood is described by a 4-bit number (i.e., from 0 to 15). From low order to high order bits these are north, east, south, west (i.e., clockwise starting from the north). The neighborhood is be compared with `num` via one of the operators $=, a, n$ as above.

Random The next two values have the form `op num`. A random number between 0 and 15 is generated. It is compared against `num` via one of the operations $\{=, <, >, a, n\}$.

Energy The next two values have the form `op num`. The organism's current energy is compared with `num` via one of the operators $=, <, >$.

Attacks The next two values have the form `op num`. They check whether the organism was attacked on the last move from each direction. The format is similar to the neighborhood operations above, with the pattern of attacking neighbors described by a 4-bit number. The neighborhood is be compared with `num` via one of the operators $=, a, n$ as above.

Supports Same as Attacks, but for supports.

Previous-Success The only case that a move does not succeed is when you try to move or split to an occupied space, or to attack/support an unoccupied space. Usually you can perform sufficient checks to avoid such actions, but if they happen the previous-success flag is set to 0; otherwise it's 1. A "*" for this field means that you don't care about the success of the previous action. A "1" means the rule applies only after a successful action. A "0" means the rule applies only after an unsuccessful action.

Action The next two values have the form `action direction`. `direction` may be "n", "s", "e", or "w". Action may be "m" (move), "s" (split), "c" (cooperate, i.e., support), "d" (defect, i.e., attack), or "n" (no-op, i.e., do nothing).

New-state The state may be modified via a pair of the form `op num`. `op` is one of $\{=, a, o, x, i\}$. They correspond to assignment with `num`, logical ANDing with `num`, logical ORing with `num`, logical XORing with `num`, and incrementing by `num` and taking the result modulo 256.

If no rule matches the current configuration, the organism does nothing. On a split, (and "in the beginning") the new organism starts in state 0.

An organism starts with 50 units. A move costs 1 unit. A split costs 5 units. A supported organism gains 1 unit per support. An attacked organism loses 7 units per attack. An attacker gains 5 units.

The simulator is available in `~kar/4995/organisms/organisms`. You'll find the source code in the same directory if you want to play with it. The simulator runs a simulation, and writes selected states of the simulation to html files. You can view these html files to study the results of the simulation. The first file is always called `0.htm`.

The command line is `organisms config-file seed`. The seed is optional, and is used to drive the random number generator. The config file has the following format, one entry per line:

```
width
length
directory to put the html files in
number of iterations to simulate
number of iterations between display steps
name of brain
name of brain
...
```

You may put as many brains as you like on the field (compiled limit is 30) and one organism will be created (and placed randomly on the field) for each brain. You can use this for debugging to, putting many copies of your organism on the field and following their progress separately. Different brains can be placed on the field in this way too.

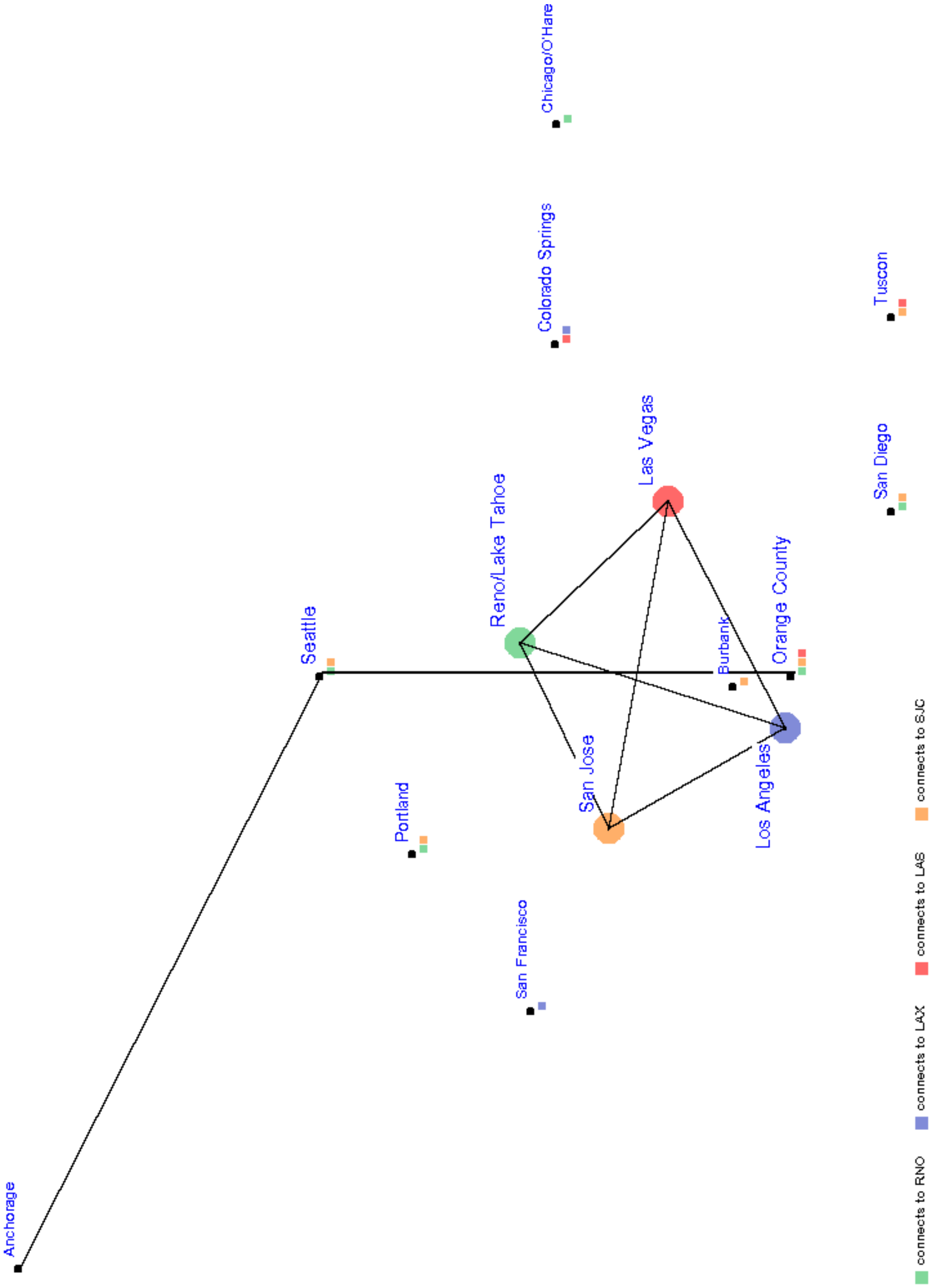
Organisms Source Distribution

If anybody else wants to use the organisms code, they are welcome to do so for any noncommercial purpose. Please inform the author (`kar@cs.columbia.edu`) of any interesting use of the code.

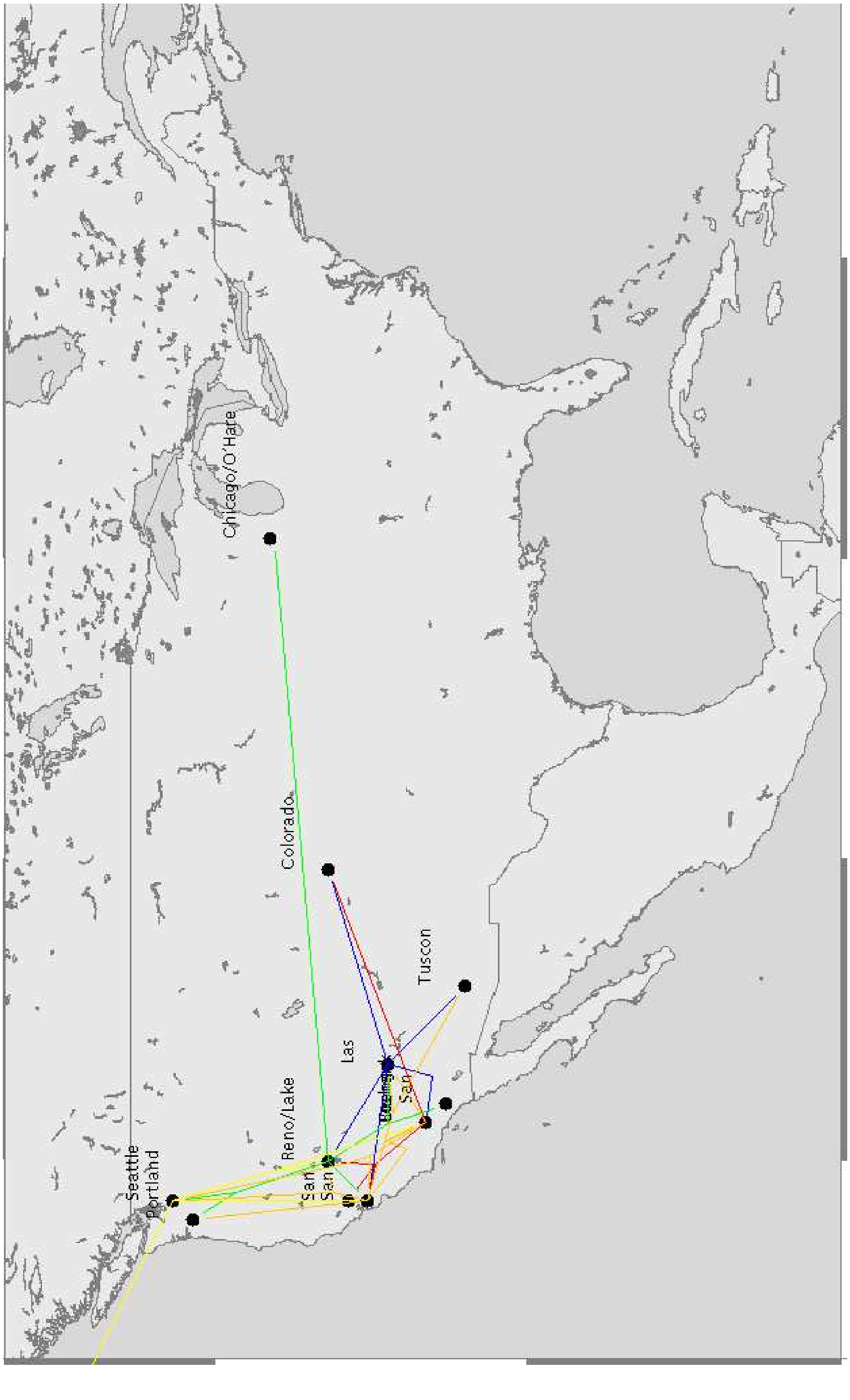
The code (a tar file) can be found at `http://www.cs.columbia.edu/~kar/organisms.tar`. It is written in C++.

8.2 Sample Maps

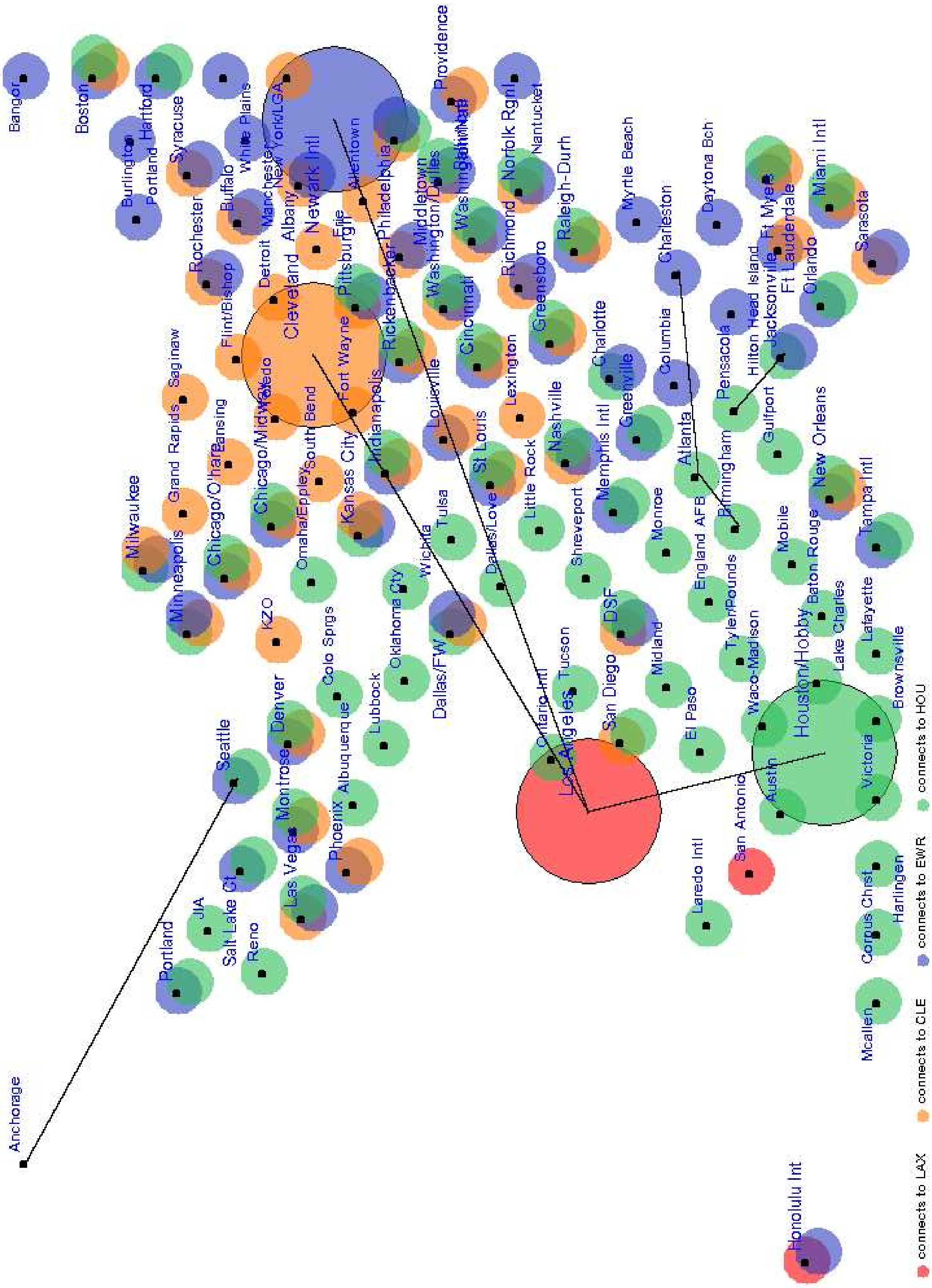
We present here a sample of the maps produced for project 2. The maps are best viewed in color. The following map of the Reno system by group 5 was used on the front of the class T-shirts.



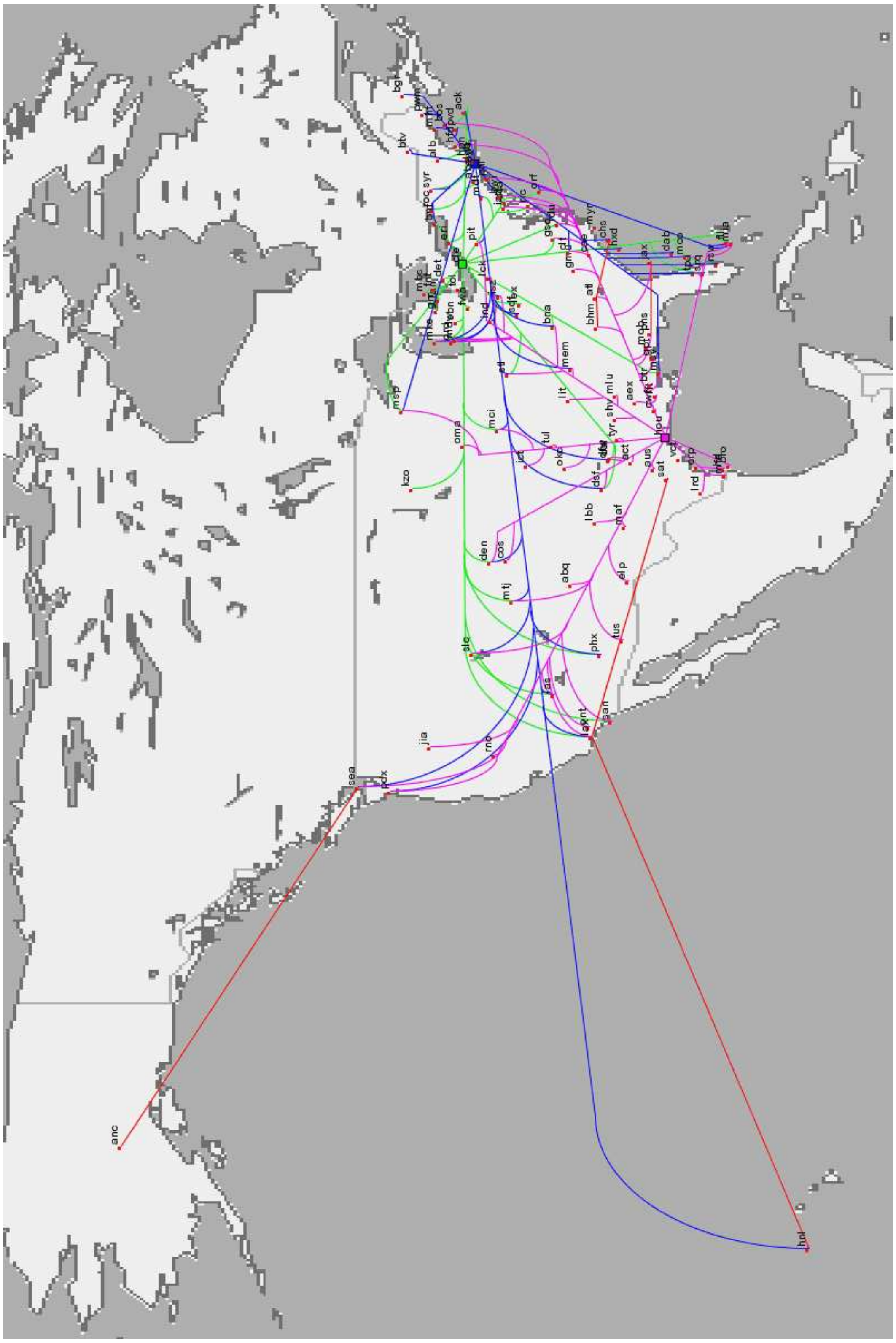
This is group 2's version of the Reno map.



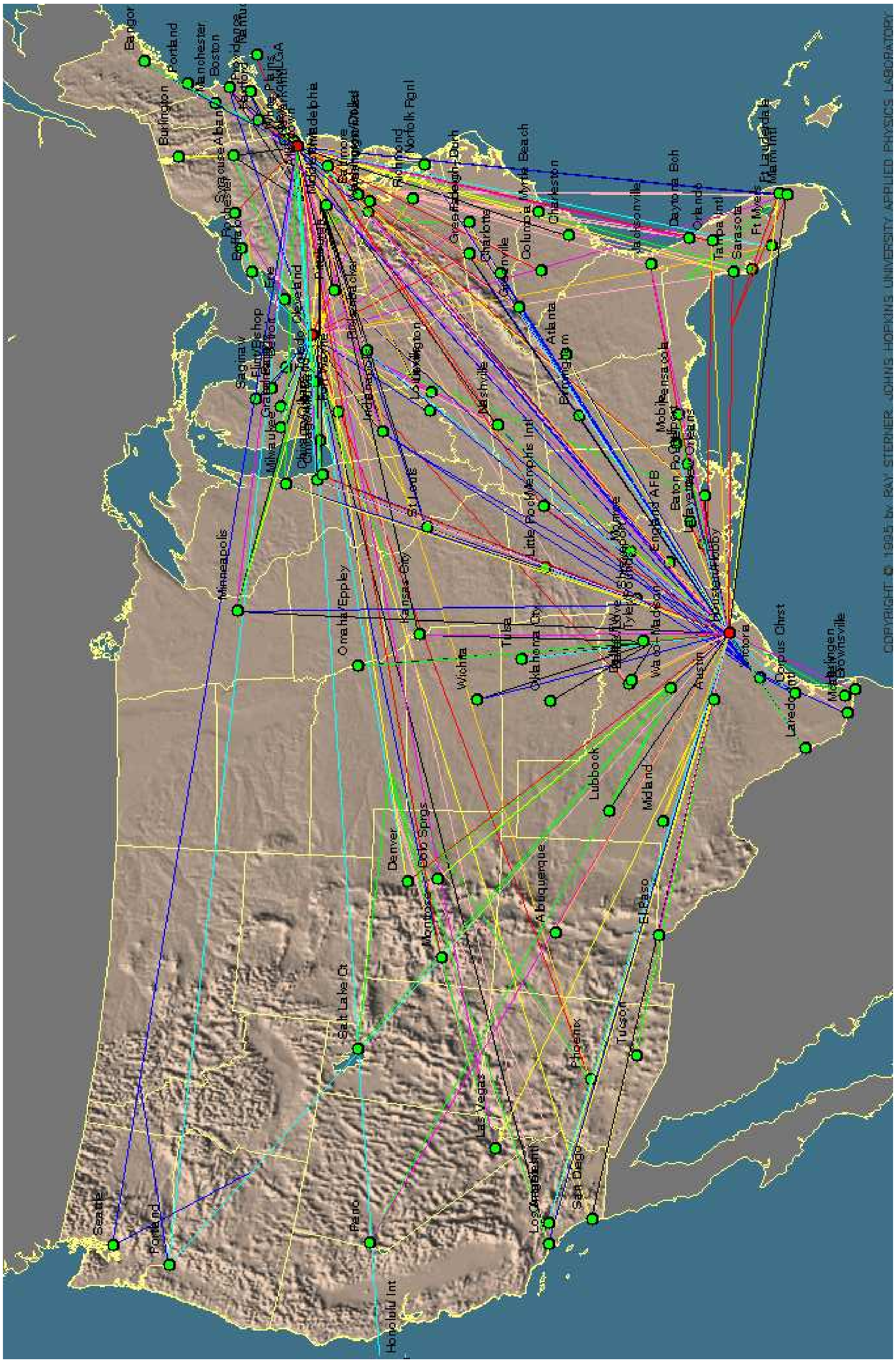
The following map of the Continental system uses group 5's alternative style.



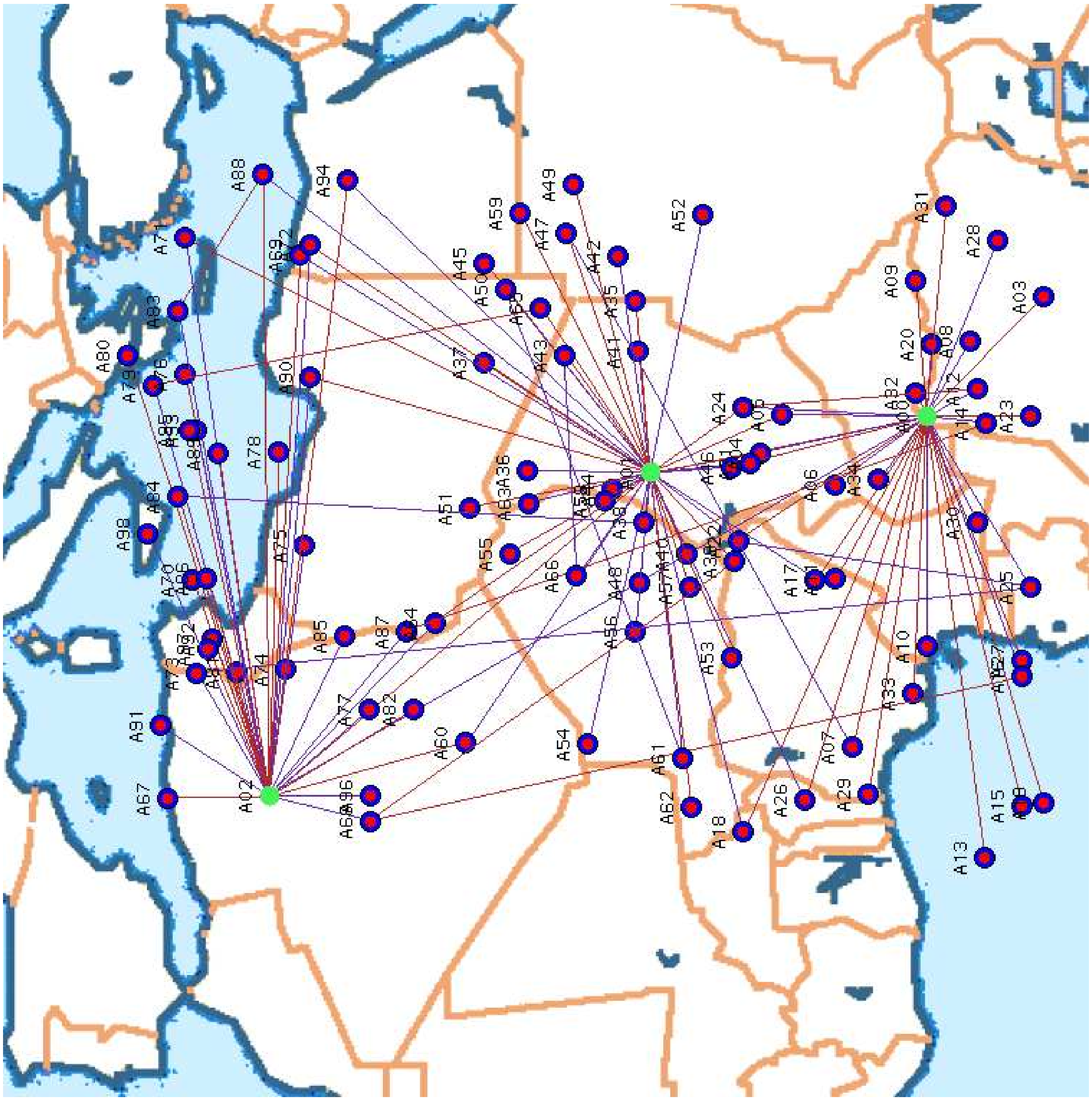
The following map of the Continental system by group 3 illustrates their fingering branches.



The following map of the Continental system by group 4 illustrates their method of branching, and their use of color.



The following map of the Air Shamoun system by group 1 uses actual longitudes and latitudes, leading to a background map of Africa.



8.3 Toetjes Pairwise Tournament Results

	brian	greg	group2	k2	pajk	pajk2	stanf	team4	Average payoff
brian		100.013	107.856	114.53	100.957	99.3919	101.207	98.1807	103.1622286
greg	99.982		105.024	114.206	109.67	109.058	103.133	100.681	105.9648571
group2	92.7572	95.3045		116.802	102.354	101.192	105.34	99.524	101.8962429
k2	85.361	85.547	82.9952		84.1659	86.0457	85.3355	88.8028	85.46472857
pajk	99.1438	90.9292	97.0361	115.955		100.947	104.437	98.2678	100.9594143
pajk2	100.523	90.9225	98.5758	113.323	99.2745		103.088	96.1963	100.2718714
stanf	98.8045	97.0644	94.7466	114.806	95.6305	96.9109		97.812	99.39641429
team4	101.515	99.6378	100.608	111.344	101.722	103.874	102.316		103.0024

8.4 Map Judging Results

The judges' tabulated scores are given below. Assorted comments follow.

Reno	G1	G2	G3	G4	G5	G5V
nontraveller (Alice)	4	5	6	7	5	10
Artist	6	5	5	7	7	8
Child	4	4	6	7	5	6
Instructor	6	6	6	7	8	9
Travel Agent	7	5	9	6	8	8
	5.4	5	6.4	6.8	6.6	8.2

Continental	G1	G2	G3	G4	G5	G5V
nontraveller (Alice)		4	4	5	5	4
Artist		4	6	7	8	9
Child		3	4	4	5.5	5
Instructor		4	7	7	8	8
Travel Agent		6	8	8	10	10
	0	4.2	5.8	6.2	7.3	7.2

Swissair	G1	G2	G3	G4	G5	G5V
nontraveller (Alice)	2	3	4	3	6	4
Artist	4	4	5	7	9	8
Child	4	3	4	5	3	5
Instructor	3	2	4	2	8	7
Travel Agent	5	4	6	7	9	9
	3.6	3.2	4.6	4.8	7	6.6

Air Shamoun	G1	G2	G3	G4	G5	G5V
nontraveller (Alice)	4	4	4		4	4
Artist	4	4	5		8	10
Child	5	4	5		6	6
Instructor	8	3	4		7	8
Travel Agent	8	4	5		9	9
	5.8	3.8	4.6	0	6.8	7.4

	G1	G2	G3	G4	G5	G5V
Overall	3.7	4.05	5.35	4.45	6.925	7.35

Some notable comments:

- "Are the blobs there for a reason?"
- "Maps help interpret the data."
- "Not enough color."
- "Too small."

- “Don’t like city abbreviations. Would give non-English speakers a problem.”
- “Does a dot inside a dot mean something special?”
- “Too cluttered.” *A common complaint!*
- “Need to look very close to follow links.”
- “The dots bother me.”
- “A key would help.”
- “Where is Anchorage?” “Where is Honolulu?”
- “Hard to make out Europe.”
- “Edges don’t reach cities.”
- “Bigger dots for cities would be good.”
- “What does RNO mean?”
- “Large color circles visually appealing. Lines around circles help.”
- “Whole words are good.”
- “The writing overlaps itself.”
- “What do the differences in city color mean: green vs. red/blue?”
- “Do cities with little yellow squares connect to each other?”
- “Links are like spider legs. It’s hard to tell what’s important.”
- “The mountains in the nice background make the lines hard to follow.”
- “Yellow lines are hard to see on a white background.”
- “The geography shift doesn’t bother me.”
- “The venn diagram on Continental looks beautiful.”
- “The black squares and the colored squares are confusing being the same size and shape.”
- “I like straight lines.”
- “I can’t read the place names.”
- “What do those offshoots from lines mean?”
- “The three-letter codes are hard to work out.”
- “Blurry!”
- “The group 2 map of Swissair gets 10/10 for comedy! What an artistic splash!”
- “I can’t see anything.”
- “The hub letters should be even bigger.”
- “Blue writing interfered with by blue dots.”
- “3 collinear cities make edges ambiguous.”
- “Nice separation of colors.”
- “Maps with a background map are better.”

- "I don't understand the colors."
- "The dots are too far from the city names."
- "Too bunched on the east."
- "Nice spread, but try for better geographical accuracy."
- "Cities aren't exactly where they're supposed to be."
- "Keys are good."
- "Air Shamoun seems more real with a background."
- "The branching lines look nice."
- "The lines go through other things."
- "Why are there dots in the ocean?"
- "Dog legs look awkward."
- "Lines overlap."
- "Unbalanced."
- "Colorful and attention-grabbing."
- "The red dot inside blue is more attention-grabbing than the green dot given to the hubs."
- "Some unnecessary line crossings."
- "2 maps is good, but the Europe map looks faked."
- "CHI and BOM are misplaced." *My fault — KAR.*
- "This map is truncated."
- "Why lines *and* colors?
- "There's info missing."
- "Looks like the map shows Cleveland to Omaha, but I know they don't fly that route." (The checked this fact on his travel agency computer.)
- The group 5 maps are "easy to read. Probalbly the best way to do them."
- "Should be larger."
- "Really bad."
- *laughter*

8.5 Battlesplat tournament results

10 by 10

	gplacer	maria	mjp-placer	nina	pinta	placee	placer4	santa	splat.sh	
elaine	101	87	102	99	61	102	^	110	118	97.5
george	89	81	81	75	73	81	83	93	92	83.1
gguesser	91	153	85	78	58	87	93	73	92	90.0
guess.sh	89	90	84	74	72	78	92	91	95	85.0
jerry	96	94	89	89	57	89	100	91	102	89.7
kramer	91	89	84	77	69	79	91	91	95	85.1
mjp-guesser	90	96	83	78	75	78	84	95	87	85.1
nothing	90	91	86	83	83	85	<timeout	93	88	87.4
spiralGuesser	97	91	79	72	70	78	90	91	95	84.8
	92.7	96.9	85.9	80.6	68.7	84.1	90.4	92.0	96.0	

Placer ranking	Guesser ranking
maria	george
splat.sh	spiralGuesser.sh
gplacer	guess.sh
santa	kramer=
mjp-placer	mjp-guesser=
placee	jerry
nina	gguesser
pinta	elaine

43 by 43

	maria	nina	pinta	placer4	santa	
elaine	1605	861	892	1019	1746	1224.6
george	1733	1013	1112	1038	1849	1349.0
gguesser	1339	1402	747	3336	1366	1638.0
guess.sh	1716	881	1190	1766	1849	1480.4
jerry	1050	766	658	1251	1064	957.8
kramer	1720	1055	1069	1762	1849	1491.0
mjp-guesser	<timeout	1141	1074	1387	1841	1360.8
nothing	1434	1305	1264	1293	1437	1346.6
spiralGuesser.sh	1812	976	1113	1793	1810	1500.8
	1551.1	1044.4	1013.2	1627.2	1645.7	

Placer ranking	Guesser ranking
santa	jerry
placer4	elaine
maria	nothing
nina	george
pinta	mjp-guesser
	guess.sh
	kramer
	spiral
	gguesser

100 by 100

	maria	nina	pinta	placer4	santa	
elaine	9386	4698	5876	4832	9770	6912.4
george	9800	5691	5290	5163	9936	7176.0
guess.sh	9710	4875	6143	9447	9901	8015.2
jerry	5923	3683	3019	6442	5439	4901.2
kramer	9760	5090	5630	9827	9901	8041.6
mjp-guesser	9933	4091	4358	7265	9923	7114.0
nothing	7524	6031	6482	7154	7017	6841.6
spiralGuesser.sh	9899	5679	5218	9458	9901	8031.0
	8991.9	4979.8	5252.0	7448.5	8973.5	

Placer ranking	Guesser ranking
maria	jerry
santa	nothing
placer4	elaine
pinta	mjp-guesser
nina	george
	guess.sh
	spiral
	kramer

8 by 23

	gplacer	maria	mjp-placer	nina	pinta	placee	placer4	santa	splat.sh	
elaine	177	172	168	141	106	168	156	195	^	160.4
george	172	163	147	112	104	143	119	184	^	143.0
gguesser	219	159	182	115	93	166	284	170	129	168.6
guess.sh	159	163	148	109	105	141	174	184	205	154.2
jerry	145	140	132	113	82	129	151	146	181	135.4
kramer	159	163	141	114	107	144	174	184	157	149.2
mjp-guesser	158	170	149	146	142	146	137	170	181	155.4
nothing	160	158	150	149	135	151	<timeout	143	<timeout	149.4<not rankable
spiralGuesser.	166	163	144	115	107	136	172	184	181	152.0
	168.3	161.2	151.2	123.8	109.0	147.1	170.9	173.3	172.3	
									^not rankable	

Placer ranking	Guesser ranking
santa	jerry
placer4	george
gplacer	kramer
maria	spiralGuesser
mjp-placer	guess.sh
placee	mjp-guesser
nina	elaine
pinta	gguesser

27 by 64

	maria	nina	pinta	placer4	santa	
elaine	1608	686	739	935	1688	1131.2
george	1664	830	938	926	1711	1213.8
gguesser	1263	1516	718	2409	5184	2218.0
guess.sh	1670	1059	934	1670	1702	1407.0
jerry	1039	659	497	1163	988	869.2
kramer	1663	889	1036	1717	1702	1401.4
mjp-guesser	1717	1145	941	1064	1707	1314.8
nothing	1375	1280	1204	1212	1391	1292.4
	1499.9	1008.0	875.9	1387.0	2009.1	

Placer ranking	Guesser ranking
santa	jerry
maria	elaine
placer4	george
nina	nothing
pinta	mjp-guesser
	kramer
	guess.sh
	gguesser

9 by 1000

	maria	nina	pinta	placer4	santa	
elaine	8980	4056	5133	4972	9012	6430.6
george	8978	5546	4555	5378	8993	6690.0
guess.sh	8978	4509	4461	8984	8992	7184.8
jerry	5719	2863	3708	5604	5422	4663.2
kramer	8981	5027	3600	8976	8992	7115.2
mjp-guesser	8601	7248	6776	7304	8551	7696.0
nothing	7307	TBA	TBA	7172	7441	7306.7
	8220.6	4874.8	4705.5	6912.9	8200.4	

Placer ranking	Guesser ranking
maria	jerry
santa	elaine
placer4	george
nina	kramer
pinta	guess.sh
	nothing
	mjp-guesser

8.6 Organism Tournament Results

8.6.1 Brains of more than 10 lines

	50x50 group	25x25 group	25x25 pairs	12x12 group	12x12 pairs	7x7 group	7x7 pairs
McBain	183	79	393	17	76	11	22
barney	107	59	319	16	73	10	25
g2-anklo.br	10	9	8	7	8	5	7
g2-stego.br	45	31	274	10	63	7	23
g3-ebola	30	17	145	5	26	2	8
g3-hanta	97	53	249	19	57	12	20
g3-hyper	227	99	440	20	87	11	24
g3-mummy3	178	120	385	66	115	38	46
g5-dontrunmepope	297	112	457	24	89	11	26
g5-psycho	0	0	26	0	8	0	3
g5-thepope	275	113	469	25	91	10	26
g5-combined	293	122	471	23	91	17	27
gherpes	270	125	454	26	95	17	29
marge	74	5	0	14	65	9	23
otto	34	4	0	9	30	8	15
tattat	173	78	390	25	82	14	26
tittat	183	116	426	27	89	14	28

8.6.2 Brains of at most 10 lines

	25x25 group	25x25 pairs	12x12 group	12x12 pairs	7x7 group	7x7 pairs
3brain	10	203	2	47	5	16
g2-10l.br	32	250	4	54	5	17
g2-3l.br	7	183	2	42	2	14
g2-3lb.br	2	131	0	24	2	9
g2-6l.br	12	277	1	58	7	18
g2-7l.br	7	170	1	39	5	13
g2-rex.br	0	0	0	0	0	0
g2-slimy.br	1	56	0	9	2	6
g3-dengue	194	512	54	118	27	40
g3-gout-7lines	125	407	41	103	26	37
g3-influenza-3lines	11	197	3	46	6	16
g3-menengitis-8lines	205	521	56	118	31	39
g3-minimummy-9lines	98	500	38	116	32	35
g3-monkeypox-9lines	201	510	52	118	31	39
g3-rickets-5lines	26	320	5	75	10	24
g3-smallpox-10lines	281	551	68	122	29	39
g3-strep-4lines	19	298	3	68	7	22
g5-6brain	18	297	4	65	3	21
maggie	13	277	4	60	4	19

8.6.3 Brains of at most 7 lines

	25x25 group	12x12 group	7x7 group
3brain	44	14	11
g2-3l.br	36	11	10
g2-3lb.br	20	6	5
g2-6l.br	64	18	9
g2-7l.br	30	9	7
g2-rex.br	0	0	0
g2-slimy.br	11	5	5
g3-gout-7lines	283	85	37
g3-influenza-3lines	39	14	12
g3-rickets-5lines	94	28	18
g3-strep-4lines	83	22	14
g5-6brain	75	24	14

8.6.4 Brains of at most 4 lines

	25x25 pairs	12x12 pairs	7x7 group	7x7 pairs
3brain	264	62	12	22
g2-3l.br	242	56	9	20
g2-3lb.br	161	35	5	12
g3-influenza-3lines	250	60	10	21
g3-strep-4lines	444	101	18	32