Project 1 Organisms 2 Final Report

Group 5

Bogdan Caprita (bc2008)
Mike Mulley (mm2041)
Rean Griffith (rg2023)

Introduction

Our approach to developing players was to implement a number of strategies, analyze their performance in single and multiplayer tournaments and see whether online or offline tuning lead to any improvements or explanations about the players' performance.

Background

We divide the game into two phases of interest, the initial Sunrise phase and the later Sunset phase.

The Sunrise phase represents the early rounds of the game; our definition of early is when the round number is less than three hundred (<300) for 5000 round games. This early phase is characterized by plentiful food, relative to the food available during the later phase of the game, and a sparse board. Under these conditions, food appears more often and the possibility of food doubling is greater, provided that $q > 0$, since the board is under populated. In the early phase, aggressively searching for food can have a higher payoff than later in the game. In this early phase, amoeba energy is higher allowing an amoeba to search longer and farther than it could later in the game when its energy level is likely to be lower. The initial gamble of whether to strike out early foraging for food is unavoidable, if there is no food around an amoeba it can always sit still, conserve its energy while slowly starving to death – which could be a great strategy in exceptionally harsh conditions – at least guaranteeing that a cell lasts M / s rounds where M is the maximum energy level per organism and s is the amount of energy used for staying put.

The Sunset phase represents the later rounds of the game; round number greater than three hundred (>300) for 5000 round games. This phase is characterized by less plentiful food and the board is likely to be crowded. Based on the discussions in class and running simulations with this year's and last year's players this proved to be the case since most players adopted the simple rule of thumb of "expand early in the game to avoid being crowded out of food later".

During both of these phases there can be huge cycles in population density. If food is plentiful there is an increased propensity to reproduce more often, if there is sufficient food then high reproduction rates can be sustained, however overpopulation usually leads to food depletion and amoeba starvation, making room for food to grow again and another possible surge in population density if there are surviving amoebas. Based on simulations the most violent surge occurs in the Sunrise phase as initial aggressive expansion takes place. During the Sunset phase surges are less violent because there is less open space on the board and less food available. It is not guaranteed however that the system reaches a self-regulating or stable state where the rate of reproduction balances the rate of automata dying and the food source remaining for the most part constant.

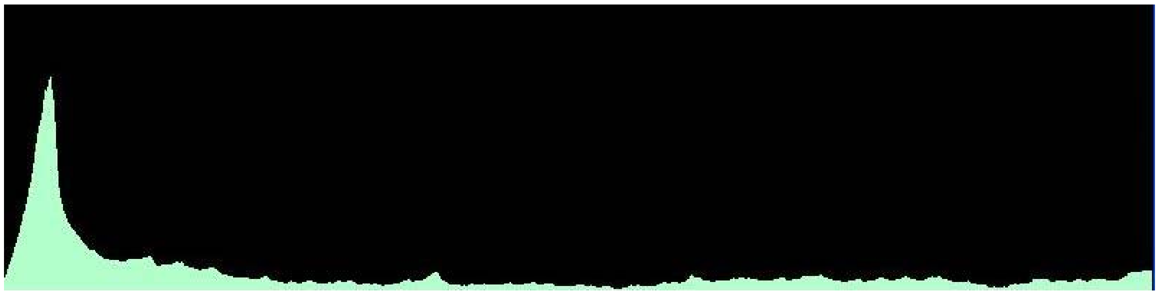


Figure 1 - Steady food supply in the Sunset Phase

Game Simulator Parameters

There are given nine (9) simulators parameters upon which we can base our player strategies on. Four (4) parameters are hidden from amoebas while they are aware of the remaining five (5).

Hidden parameters:

p – The probability of food spontaneously appearing
q – The probability of food doubling
m – The horizontal size of the grid
n – The vertical size of the grid

Visible parameters:

s – The amount of energy used to stay put
v – The amount of energy used for moving or reproducing
u – The amount of energy per unit food
M – The maximum energy per organism
K – The maximum food units per cell

Based on the two categories of parameters it was initially considered whether it would be a useful strategy to try to estimate some of the unknown parameters. We also examined how the visible parameters could be useful to any strategies we came up with.

Estimating Hidden Parameters

Estimating p and q

The discussions in class highlighted a number of drawbacks with estimating p and q.

1. Amoebas can only sense local information about their environment, estimates of the global values of p and q based only on limited local information are expected to be inaccurate, sometimes by as much as an order of magnitude [1] To get more accurate estimates, an amoeba must sit around for a longer period of time collecting data and refining its estimates. While sitting around conserves the most energy, this strategy can leave an amoeba vulnerable to starvation if it is not sitting on or near food.

2. In multiplayer games, sitting around for n rounds gives an n-round advantage to opponents which could potentially deplete the food on the board or block us into a foodless section of the board hastening our demise.
3. Even if you estimated the correct values of p and/or q what would you do with them? [Someone raised this in the last class on September 27th but we can't find it in the transcript]

Group1 presented a nice compromise to respond to drawback number 1. Their player uses the first fifty (50) rounds to gauge whether they are in harsh, moderate or good conditions based on a sliding scale of their own devising.
However, rather than sit still allowing opponents to get a fifty round head start they collect information as they are spreading out across the board reproducing and wandering. This appears to be a reasonable compromise based on their claims of consistency across players as well as across various simulator values of p and q. Also it highlights an observation that was raised by one of our own group members, based on the limitations of this simulation, information that is correct as to the order of magnitude is just as good as having the exact information.

Estimating m and n

Knowing how big the board is was not really considered to be parameters that amoebas really needed to know. Having a big board has a number of benefits:

1. It increases the likelihood that organisms from the same species (team) develop independently without being crowded by organisms from other species.
2. The bigger the board the more likely it is to be sparse which allows food to appear and double more often then if the board was smaller and crowded.

It was also discussed in class that trying the estimate the size of the board was not worth the hassle given a number of obstacles:

1. Amoebas don't know where they are, walking off one end of the board wraps around and the amoebas are none the wiser.
2. Amoebas could go off in opposite directions counting squares until they meet, but amoebas are not able to tell whether an amoeba it runs into is the one it is looking. Furthermore, if there are obstacles in an amoeba's path, getting back on the right track may be difficult e.g. if an amoeba runs into a wall of amoebas of another species. [ref notes from Sept 13]
3. It is more important to gauge whether the board is crowded which each amoeba can infer based on its local information e.g. using a count of how many other organisms it has met after moving n squares as a rough approximation.

Using the Visible parameters

For our players the only visible parameters we used were u – the amount of energy used staying put and v – the amount of energy needed for moving or reproduction and M – the maximum energy possible per organism.
We set our low energy threshold as energy left < 6v. The multiplier 6 was chosen arbitrarily. For details on how this was used see [Bogdan's section on the TilePlayer strategy]
We also used the ratio of u/v – the cost of moving to determine an initial moving probability. [see BigMommaStrategy section for details]

Basic Building Blocks

Communication

Two types of communication are allowed in this project; Amoeba-to-amoeba and parent-to-child.
Amoeba-to-amoeba communication is limited to and 8-bit channel but exchanges can take place more frequently over a sequence of turns. Every round an amoeba can sense the 8-bit state of amoebas adjacent to it.
Parent-to-child communication takes place only when an amoeba reproduces. The channel in this instance however is 32-bits wide.
Given the restricted channel of amoeba-to-amoeba communication and the possibility of spoofing and/or drawn out negotiation protocols we decided against implementing any form of amoeba-to-amoeba communication. We chose instead to make use of parent-to-child messages as our communication medium of choice.

```
3 3 2    2 2 2 2 2 2 2 2 2 1 1 1 1  1 1 1 1   1 1
1 0 9    8 7 6 5 4 3 2 1 0 9 8 7 6  5 4 3 2   1 0 9 8  7 6 5 4  3 2 1 0
Strategy Round #                    Trait4    Trait3   Trait2   Trait1
```

Key:

Strategy	which strategy from a predefined set should the child select as its primary one. The predefined set included five (5) core strategies: TilePlayerStrategy SexFiendStrategy SteadyStateStrategy CarefulBreederStrategy BigMommaStrategy See [section on Player Strategies] for the details on how each of these strategies works.
Round #	what round has the simulator reached?
Trait1	The index of the factor in a predefined table used to adjust the age after which an amoeba would begin reproducing.
Trait2	The index of the factor in a predefined table, used to adjust the probability that a cell reproduces.

Trait3	The index of the factor in a predefined table used to adjust the probability that an amoeba favors moving.
Trait4	The index of the factor in a predefined table used to adjust the probability that a cell sends its children far away from the position of the parent.

Estimating board crowdedness

Each amoeba estimates the crowdedness of the board based solely on local information. As an amoeba moves searching for its “position”, it keeps a moving average of the number of other amoebas it runs into. We use this information on crowdedness as feedback into varying the reproduction rate [see the TilePlayerStrategy section for details on how an amoeba calculates its position and the mechanics of reproduction rate control]

Keeping a history

At a late stage in the project we considered maintaining a history of the board conditions and the moves we made under those conditions, collectively referred to as a context, but we did not implement a strategy that leveraged historical information in any useful way before the submission deadline. Keeping the previous hundred moves or so might be beneficial for comparing past conditions with current conditions. Placing a limit on the length of the history kept would allow us to act on more recent information rather than being bogged down with a lot of old information. The tradeoff we see here however, is that older players become more valuable whereas our current strategies are age agnostic with regard to amoebas dying.

Simple rules of thumb

Some simple rules all of our players follow include:

1. Always squat on food
2. Truly random walking is wasteful, controlled walking is better. A controlled walk picks an initial direction at random and follows it for n steps. At the n^{th} step, if there is no food, another random direction is picked for another n steps, preference is given to any direction that does not backtrack unless that is our only option.
3. Always move towards food. If food is sensed while walking, go for it.

Player Strategies

Group5Player5 (“Strategy Chess Knight”)

Group5Player5 relies exclusively on the TilePlayerStrategy.
The Tile Player belongs to the generic class of sit-and-wait strategies, and evolved out of two crucial observations about the dynamics of the game:

1. Movement is expensive

Moving is costly in that energy is wasted with no clear gain. In the most simplified model, a cell which sits has the same mathematical probability of finding food as one that moves, since at any time it can see only 4 adjacent squares. This simplifying assumption assumes that there is no food already on the board, and thus all food that an amoeba encounters just pops in exactly when the amoeba is next to it. To see why such an assumption is not too far from reality, we note that, save for the initial expansion stage of the game, food lying around is a rarity, since there is bound to be some organism to pick up any food that is more than a couple of turns old. As we will see, our Tile Player not only acknowledges this state of affairs, it bases its strategy on it, and even contributes to it.

2. Life is cheap

An amoeba is born, lives, and dies as a law abiding citizen and nothing more. It does not educate itself very much, it doesn't try to help others; it doesn't even try not to hurt others. It is an energy carrier, and when its energy is low, it becomes worthless in our eyes. At that point, creating a new amoeba or having a starving amoeba move on an adjacent food cell cost the same. As we will see, the amoebas are food filters. They sit and pick up all the food around them, for the glory of The Tile Player and the demise of enemies.

The strategy we follow is thus almost trivial: we sit and eat all that is around us. The main feature of the player rests in where it sits, or more precisely, how. We want to span the board with amoeba such that there is an amoeba covering each cell on the board (a cell is covered by an amoeba if the amoeba sits on it, or is adjacent to it). We also want to do this with the largest efficiency possible, where efficiency = energy intake / energy expenditure. There are two major energy sinks: movement (which we minimize by sitting still), and existence, or energy used to keep an amoeba alive. The latter we optimize by employing the minimum number of amoeba necessary to cover the entire board. Our player thus introduces the L-tile positioning strategy.

L-tiling

We restrict amoeba to position themselves in an L-pattern, namely, if an amoeba sits at coordinates (0,0), then other amoeba should occupy positions
(2,1), (1,-2), (-2,-1), and (-1,2). This behavior is analogous to the movement of a chess knight.

Consider the positioning of amoeba A, B, C, D, E. The cells covered by an amoeba are denoted in lowercase letters corresponding to the name of the amoeba:

```
[.][.][e][.][.][.][.]
[.][e][E][e][.][b][.]
[.][.][e][a][b][B][b]
[.][d][a][A][a][b][.]
[d][D][d][a][c][.][.]
[.][b][.][c][C][c][.]
[.][.][.][.][c][.][.]
```

We note that out of the 8 cells around A, 4 are covered by A itself, and 4 by its L-neighbors. The main feature of this placement is apparent: all cells are covered, and there is no overlap (which makes this placement optimal in the density of amoeba needed to cover the entire board. Notice also the symmetry on a larger board segment, also apparent in the steady-state picture of the board in the single player game (Figure 2)

```
  5[ ][ ][ ][ ][x][ ][ ][ ][ ][x]
  4[ ][ ][x][ ][ ][ ][ ][x][ ][ ]
  3[x][ ][ ][ ][ ][x][ ][ ][ ][ ]
  2[ ][ ][ ][x][ ][ ][ ][ ][x][ ]
  1[ ][x][ ][ ][ ][ ][x][ ][ ][ ]
  0[ ][ ][ ][ ][x][ ][ ][ ][ ][x]
 -1[ ][ ][x][ ][ ][ ][ ][x][ ][ ]
 -2[x][ ][ ][ ][ ][x][ ][ ][ ][ ]
 -3[ ][ ][ ][x][ ][ ][ ][ ][x][ ]
   -4 -3 -2 -1  0  1  2  3  4  5
```

We define 'valid' positions to be positions that are consistent with the L-tile pattern. From looking at the above figure, it should become apparent that the valid positions form a module L (or an integer 'vector space') of dimension 2 over Z[i], the ring of Gaussian integers. In other words, looking at the board above, the 4 amoeba around the one at the origin of the coordinate systems (0,0) have Cartesian and L-coordinates as given below:

Amoeba	Cartesian	L
origin	0, 0	0, 0
NE	2, 1	1, 0
SE	1,-2	0,-1
SW	-2, 1	-1, 0
NW	-1, 2	0, 1

As another example, the cell at Cartesian (1, 3) has L coordinates: (1, 1).

We can define a linear transformation to take us from Z[i] (the rectangular Cartesian integer coordinate system) to the new 'vector space', which we found to be defined as:

T : L -> Z[i] given by the coordinate transformation

```
[ 2 -1 ] [a]   [x]
[      ].[ ] = [ ]
[ 1  2 ] [b]   [y]
```

where (a,b) is the L-point (coordinates in our 'valid' space) and (x,y) are normal Cartesian coordinates.

The reverse transformation T': Z[i] -> L is

$$\frac{1}{5}\begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix}\cdot\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

We immediately identify two major advantages to using this pair of transformations:

1. Since valid points represent 1/5 of all board cells (for example, count the number of amoeba in each row in the board model above), encoding an L-position takes less bits than encoding a Cartesian coordinate pair for the cases where we only care about representing valid points (such as for any Hillbill (stationary amoeba) location).

2. T' in particular offers a quick test for the validity of a coordinate point: (x,y) is a valid point if and only if the transformation takes (x,y) to a pair of integers.

We are ready to explain the behavior of our player.
Organisms have as their purpose in life the creation of the L-tile pattern. As such, organisms pass through two stages in life:

1. The Scout Stage
2. The Hillbill Stage

Hillbills are the amoeba sitting in a valid position, hence referred to as its 'hut'. They sit in that spot and whenever they see food around them, they either eat it, or, if they are full, they reproduce onto that food. Only Hillbills are allowed to reproduce, and only when they sit in a valid position. This gives the scouts a (0,0) consistent with the rest of the population (we will mention wraparound problems later). As an enhancement for the multiplayer game, the Hillbill is allowed to pursue food beyond the cells covered by it, and never leaves a cell that has food on it. Whenever it finds itself in a cell that is not its hut, the Hillbill has the option of moving to any food next to it. The first choice is given to food closer to its hut, so that the amoeba always has the tendency to return to its hut. If there is no food around it, the Hillbill amoeba also moves towards its hut. Basically, extending the analogy introduced by the farming strategies, we can compare our Hillbills with gardeners. They maintain a 4 square garden around their hut, from where they collect all the food. There is no fence around gardens, but there need not be one, since the gardener will most likely be the first to reach any food that grows around his hut.

Scouts are newly-born amoebas which go off in search of a valid cell to settle down into its own 'hut' and become a Hillbill. As mentioned, only a Hillbill in its hut (a valid cell) is allowed to reproduce. The reproduction invariant is that a scout's origin of coordinates is always a valid cell, and thus a scout can easily compute a valid point to go towards. Scouts are given by their parent a direction in L-space, and when they are born, the parent places the scouts in the cell that is best suited for the scout to depart in the chosen direction. If no special reproduction opportunities present themselves (i.e., food in an adjacent square that cannot be eaten by the parent), the parent reproduces spontaneously in that it round-robins among the 4 directions and sends scouts out every so often in each direction. Scouts then move away a variable distance in the determined direction. If their target cell is occupied, the scout continues its motion in its direction, until it finds an empty valid cell. At that point, it becomes a Hillbill. As with Hillbills, scouts are allowed to stray away from their direction of motion in pursuit of food, but will refocus on their destination as soon as the food is exhausted.

From the above discussion, the following advantages immediately come to mind:
Optimal energy consumption
Maximal energy intake
Simplicity

We present now how we apply various techniques to implement our strategy, and point out their benefic effects as appropriate:

* Reproduction threshold:

This is perhaps the most important variable in our strategy. The decision about how often to reproduce spontaneously, determines the energy/food/population balance of the board. Ideally, once all the Hillbills are in their 'huts' and the entire board is covered, there ought to be no need for new amoeba and thus no new scouts. Although Hillbills are very conservative, they do die off, mainly because the cells covered by them happen to not produce food for a long time. We are not the least compassionate about dying Hillbills, since they are just meant to 'garden' a given 5 cell cross. Instead, by randomly sending out scouts across the board, we fill up the gaps that the dying Hillbills leave behind. We do need to be careful about throttling our reproduction rate, lest excess scouts are doomed to wander around without finding an empty valid slot to settle down.
There are several conditions we are sensitive to when deciding to spontaneously reproduce. As a general guideline, most thresholds are soft, in that loaded coins are flipped with a probability proportional to the respective threshold:

- Minimum energy threshold

This is the only hard limit, in that a Hillbill will never reproduce if his energy falls below the threshold. This is set to 12 times the energy needed to make a move, such that the scout, born with 6 times the energy for a move, has enough resources to reach a valid cell at an L-distance of 2 (an L-distance of 1 requires 3 moves).

- Crowdedness

Each Hillbill keeps a moving average of the number of scouts seen. The more scouts it sees, the more likely it is that the board in its neighborhood has all valid L-spots occupied (and thus scouts are unable to settle down).

- Reproduction probability

This is just a probability that a Hillbill will reproduce at any round. This parameter was tuned off-line.

- Energy left

We weighted the reproduction probability by a sub unitary power of the ratio of energy left to maximum energy, mostly to give more parental opportunities to healthy Hillbills. The more energy an amoeba has, the better chance of survival the scout has (so it can reach its target cell), and the bigger the likelihood that that area is richer in food.

* Communication:

We kept communication to a minimum: apart from settling down in their L-pattern, amoeba had little else to coordinate. Furthermore, ensuring that the L-pattern was formed correctly was an easy task, as the scouts each kept their own coordinate system which guided them to a valid cell, and the parents transmitted them the direction of motion at birth (to ensure even spreading of scouts in the 4 directions).

The only instance where we do use the 1 byte external state for communication is in encounters between Hillbills and Scouts: as mentioned, Hillbills identify scouts to count their number. On the other hand, scouts also learn from Hillbills: each Hillbill that finds itself in its hut exposes that fact as a flag, which the scout can use to readjust its coordinate system (since it knows the Hillbill's hut is always on a valid square). This is the way we deal with wraparound conditions: instead of determining the board size or performing other cumbersome measurements, we allow any scout which has moved around the world and finds its coordinate system confused to readjust it based on what positions are considered valid in that neighborhood. We have found this decentralized strategy to preserve the L-pattern with only local discrepancies.

* Dealing with adversaries:

Initially, we wanted to block off any adversaries that tried to enter out L-pattern structure, by having Hillbills reproduce or move to create spontaneous walls in the face of foreigner. We soon realized that such explicit blocking, besides being expensive in energy and disruptive to our pattern, was actually not needed. Taking a look at Figure 3 should show why: since we are always the first to eat all the food in the gardens around us, enemy cells will find the intra-L-pattern terrain to be a wasteland of food and stay out of it.

Figure 2 exemplifies the L-pattern in the steady-state phase of a game with p = 0.003 on a 25 x 20 board. The darker colored amoeba are the Hillbills, when they are positioned in valid L-positions. We draw attention to the successful avoidance of any wrap-around problems: the pattern does not break down as amoeba travel across the board. The slightly brighter organisms are Hillbills that have moved outside their hut to gather food. An example would be the amoeba in cell (18,6) whose hut probably lies in (20, 7) and had followed a path of food to its present square. Finally, scouts are the bright colored amoeba such as the ones in (13,5) and (18,5). As Figure 1 implies, the amoeba are successful at consuming all the food on the board, without overpopulating. In Figure 2, scouts are few, since Hillbills adjust their reproduction rates to fit the needs of the L-pattern structure.

Figure 3 shows the end of a 5000 round multiplayer game (p = 0.005) where we are the only population left to fight off the very expansive yellow player. An important consequence of our strategy is that it avoids the initial sunrise phase by keeping few amoeba with high energy count, and expands with a slow growing exponential throughout the game. Thus, at round 5000 our player had been growing steadily from 2 cells (Round 200) to 73 (round 500) and the ascending trend does not seem to die off as we extend the running period.

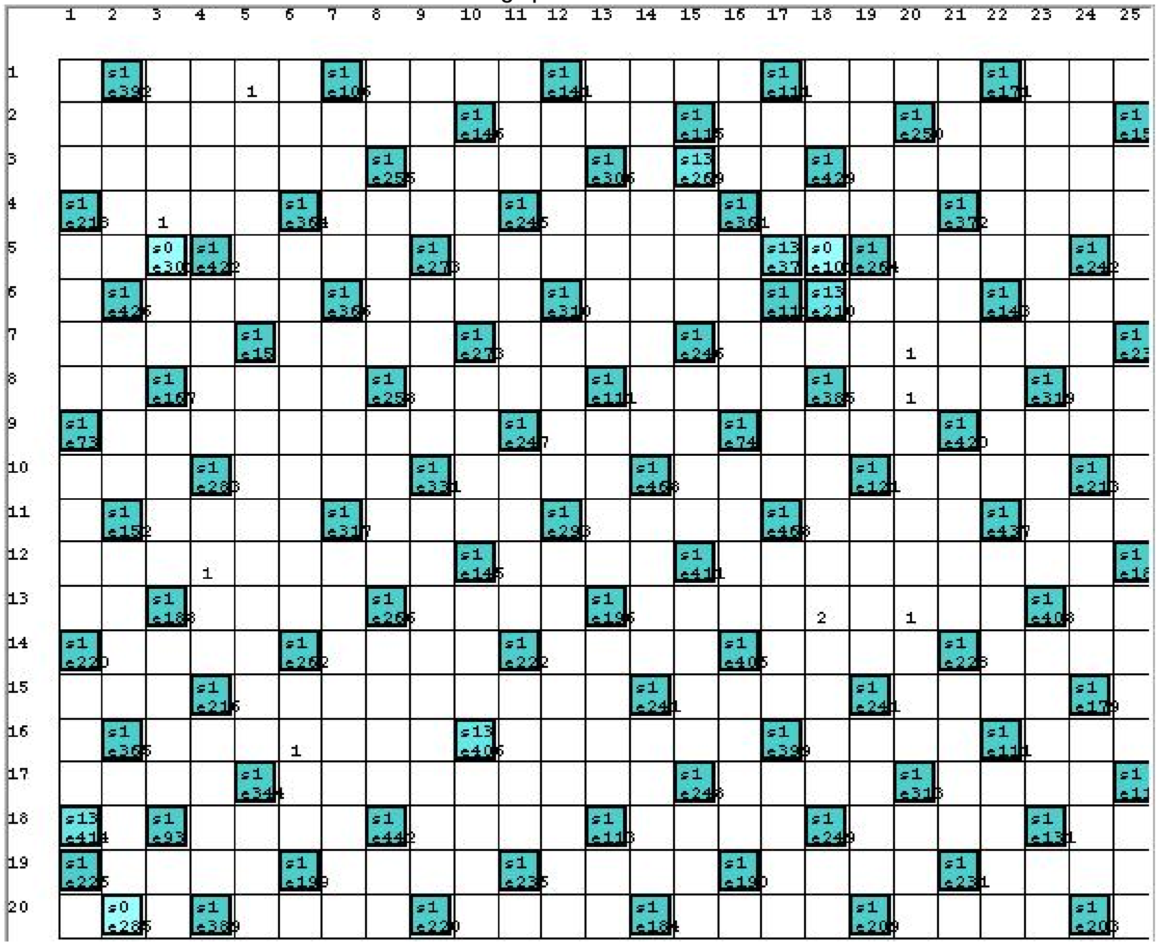


Figure 2 - Example of the Tile Pattern and Grid Covering (Scouts are Light Cyan, Hillbills are darker)

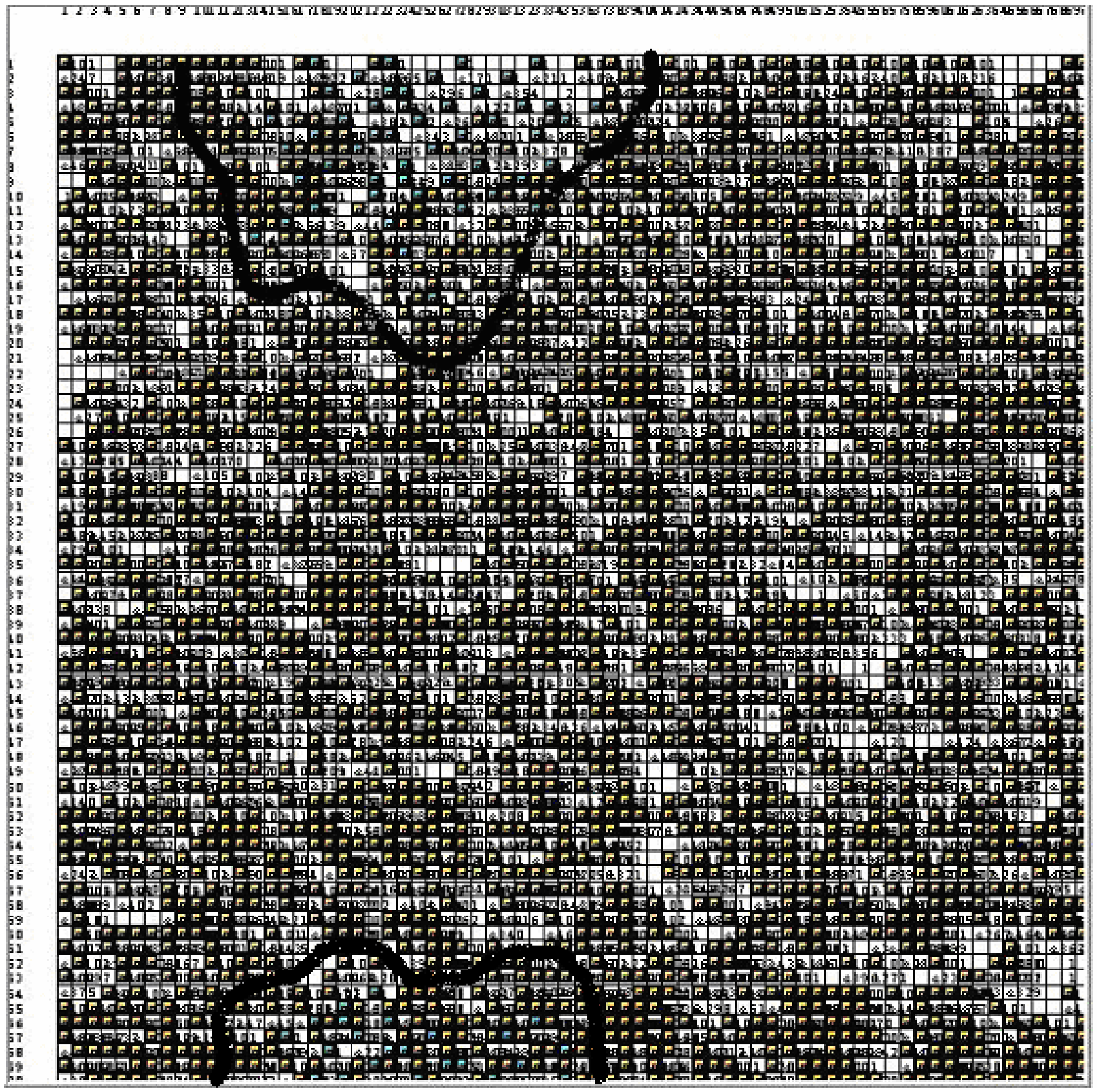


Figure 3 – Us (Chess Knight) holding off Group 4 (Yellow) using our L-tiling pattern

Group5Player9 ("Mrs. Betty Bjornson")

The central principle of our second player is the use of multiple strategies. In giving our organisms highly variable parameters and personalities, we aimed to make them adaptable to a wide range of conditions and situations.

Beyond this, Betty's design takes into account a few core principles:

- Gardening, not farming

For many board configurations, food doubling (q) is a better source of dinner than food blowing in (p). But for food doubling to be effective, the board must be left fairly open, and organisms generally need to avoid eating the last couple of food units on a square. In single-player games, this is easy to implement. But in a multiplayer game, leaving uneaten food in an empty square is simply an invitation to enemy organisms.

In class discussions and last year's players, farming was a popular solution to this problem. Food would be allowed to multiply--and enemy organisms would be kept away. But our attempts to implement farming were quickly met with frustration. Farms took a great deal of effort and coordination to set up, and it takes only one move from an enemy organism--into the home square of hungry farmer organisms--to destroy a whole farm. Proposed solutions to this generally involved ever-larger, multi-layered farms, which seem all-but-impossible to implement efficiently using only the insecure state system for communication.

Our approach was to try "gardening": leaving open squares, and depleting all the food in a particular location only if necessary. To prevent other organisms from abusing our goodwill, we temporarily shut off this behavior if there are aliens in our midst.

- Magic numbers are a pain

When working on our first player, we discovered that we didn't much like endlessly tuning scores of constants. This makes optimization not only difficult by endless.

Though our players still have their share of hard-coded constants and formulas, we also try to use inter- and intra-generational evolution to set our parameters for us.

- Game phase matters

Round 5 and round 500 are very different games. As outlined in the Background section on the phases of the game, different phases require different strategies. Risky moves early in the game can have substantially higher payoffs than risky moves taken later in the game.

- Life is cheap

One organism's death is no big deal. We'd rather produce three organisms with different strategies and lose two than produce one organism with a middle-of-the-road strategy and lose the game when its strategy proves unsuccessful. [Discussed more by Bogdan]

The Group5Player9 class is little more than a framework for different strategies. In any given time slice, the decision of which move to make is made solely by a strategy class. Instead of discussing the player as a whole, then, we will begin by discussing the individual strategies.

We used strategies as both a design principle and a programming concept. That's why you'll find, for example, a RandomWalkStrategy class. We're not content to create organisms that do nothing but walk randomly; instead, the class is used by higher-level strategies, when they decide that the best move in a given time slice is to move to a random square.

Other such mini-strategies include the StubbornWalkStrategy and the FoodSearchStrategy. The former picks a general direction, like Southeast, and then does its best to move roughly in that direction, stopping at any food it sees along the way. The FoodSearchStrategy uses the StubbornWalkStrategy to look for food, but also decides each turn whether it wants to move or just sit and watch neighboring squares. To make this decision, it uses a movement probability passed from parent to child, as well as the number of free adjacent squares--the more crowded the adjacent squares are, the less likely that food will appear, and the more likely the organism is to move.

We implement five high-level strategies. These are BigMommaStrategy, CarefulBreederStrategy, SexFiendStrategy, SteadyStateStrategy, and TileStrategy.

BigMomma

This is the strategy assigned to our first automaton, and it's never assigned to any other automata.

In general, it plays conservatively: if our first player dies before reproducing, there's no going back. It moves rarely at first, stepping up its rate of movement only if it has trouble finding food. It reproduces only when it has plenty of energy.

One of its significant functions is determining initial values of the movement and reproduction probabilities that future players will use. It initially chooses a movement probability based on the ratio of u and v. If a piece of food will give us enough energy for 15 moves, we'll move a lot; if it'll give us enough energy for 5, we'll tend to sit still. As it searches for food, it adjusts the parameters. If it finds it needs to move several squares before finding food, it will raise the movement parameter and lower the reproduction parameter. If it finds food plentiful, it does the opposite.

BigMomma's biggest duty, of course, is to reproduce. To avoid a premature demise, it reproduces only when it has significant energy and either has food itself or can put its child on top of food. Its children are sequentially assigned one of three strategies--CarefulBreeder, SexFiend, or Tile.

BigMomma is designed for the early rounds of the game. After round 250, it switches to the SteadyStateStrategy.

CarefulBreeder

This strategy is focused on reproducing frequently but not recklessly. It reproduces whenever it's very full (energy at roughly 90% of the maximum value), and whenever it has a reasonable amount of energy (roughly 45% of maximum) and can place its child on top of food.

Of course, it also searches for food. Like all other strategies, it subcontracts this to the FoodSearchStrategy, which uses the variable movement parameter to decide on its behavior.

All of CarefulBreeder's children are also CarefulBreeders. Around round 200 -- the exact point is random -- CarefulBreeder organisms switch to the SteadyStateStrategy.

When CarefulBreeder reproduces, it decides at random whether to alter the movement and reproduction parameters. The changes are made entirely at random; the assumption is that players with values better-suited to the current game will be able to reproduce more, and the fact that players are more likely to keep values as-is than tweak them will result in parameters slowly moving toward an optimal value.

SexFiend

This strategy reproduces, and then reproduces some more. It fills the board in short order, though of course the high population that results is often not stable. Our goal with the SexFiendStrategy was to be first to fill the board when conditions are favorable. When conditions aren't so favorable, well, at least we'll crowd our enemies for a while, and the presence of CarefulBreeder organisms means that we'll still have a fair number of more durable organisms on the board.

SexFiend reproduces whenever it has fairly high energy (roughly 70% of the maximum) and essentially whenever it can place its child on top of food. It also moves more frequently than other strategies, in order to spread out across the board.

All of SexFiend's children are also SexFiends. Around round 200 -- the exact point is random -- SexFiend organisms switch to the SteadyStateStrategy.

SexFiend uses the same evolutionary parameter-tweaking as CarefulBreeder.

TileStrategy

This is the same strategy that Group5Player5 uses exclusively; it's discussed in great detail in the previous section on Group5Player5.

SteadyState

The goal of the SteadyStateStrategy is to maintain its current population as smoothly as possible.

Players are only assigned the SteadyStateStrategy after a couple of hundred rounds, so the assumption is that our player has already reached sufficient population levels. Because of this, SteadyState organisms reproduce very rarely: only when they have more than (M - 2u) in energy, are adjacent to food, and are adjacent to at least two squares unoccupied by other organisms.

SteadyState organisms, then, tend to spend their lives just sitting around. But there's one important twist: when they haven't seen any foreign players in k rounds (k is 50 in the player we submitted), they start to behave cooperatively. They will move away from fellow organisms, in order to leave the board open, and they will move off squares with a fewer than 3 units of food (unless they're hungry enough to need those remaining pieces of food). This altruistic behavior is designed to take effect in single-player games and in pockets of multi-player boards inhabited mostly by our organisms.

Identification of friend and enemy organisms is done very simply. All of our organisms broadcast a constant state value, and this value is used to identify friends. This is, of course, highly vulnerable to spoofing in any given round. But because we act cooperatively only if we haven't seen a single player with a different state in 50 rounds, we believe that mistaking enemies for friends is very unlikely.

During reproduction, SteadyState uses the same evolutionary parameter-tweaking as SexFiend and CarefulBreeder.

But not all of SteadyState's children are themselves SteadyState organisms. Roughly two-thirds are -- but about one third are assigned a different strategy (SexFiend, CarefulBreeder, or TileStrategy) at random. Because of their nearly infertile ways, SteadyState organisms are particularly vulnerable to temporary changes in food supply, such as that which results from an unsustainable surge in an enemy organism's population. Organisms will die of starvation, but then will regenerate very slowly once food starts to reappear. But if breeder players are occasionally produced, they will take advantage of these fertile conditions and quickly replenish our population. (CarefulBreeder and SexFiend players that trace their ancestry to SteadyState players all revert to the SteadyState strategy after about 130 rounds.) This is also evolutionary behavior: if TileStrategy's approach is better-suited to board conditions, that occasional child should thrive, reproduce, and start taking over SteadyState players.

Strategy Tuning

As we tuned our players over the four – five week period we quickly realized that while hand tuning seemed to give us decent results, the more sophistication we wanted to add to our players the more magic numbers we had to add. Turning any of these extra “knobs” resulted in wildly oscillating behavior; clearly we needed a better way or some justification for selecting some of our magic constants.
To this end we found the concept of genetic mutation very appealing. We identified for key traits that we believe have a direct impact on a player’s success. These traits are:

The propensity to move
The propensity to reproduce
The propensity to send offspring farther away from the birth site
The age when reproduction starts

We augmented our parent-to-child communication to allow us to pass down a directive to the child that would cause it to adjust one or more of these traits in a positive or negative direction. In our final player Group5Player9 we chose to mutate either the propensity to move or the propensity to reproduce, and randomly selected a scaling factor from a table of multipliers for the child to use.

```
int[] table = { 0,1,2,4,5,8,10,50,100,-1,-2,-4,-5,-8,-10,-50 };
```

Given the four bits per trait, we randomly selected an index in this sixteen element table to pass onto the child. The child in turn would compute the new value for its trait.

As an example of some offline runs we did to determine reasonable initial move propensities. For set simulator parameters of p and q we use a test strategy that every time it reproduces picks a factor index at random. Each child is likely to have a completely different move probability than its parent.

For our tuning experiments we augment each player to have a unique id and we run each player in a 10000 round game. For every move the player prints out its id, age, movement probability and whether that trait has been mutated or not. We process this output after the tournament and examine which move probabilities are used by the amoebas that have the longest lifespan for the given simulator parameters of p and q. We then use these probabilities as the magic numbers in some of our player strategies depending on the simulator conditions we want to optimize for. While we do not think that this will lead to “optimal” parameters suitable for every scenario they may be used in a scheme suggested by Madhuri Shinde of Group 8 who suggested that a lookup table of <conditions,parameter value> tuples may be useful for organisms to have if they can indeed reasonably estimate conditions.

See a few examples of tuning runs below.

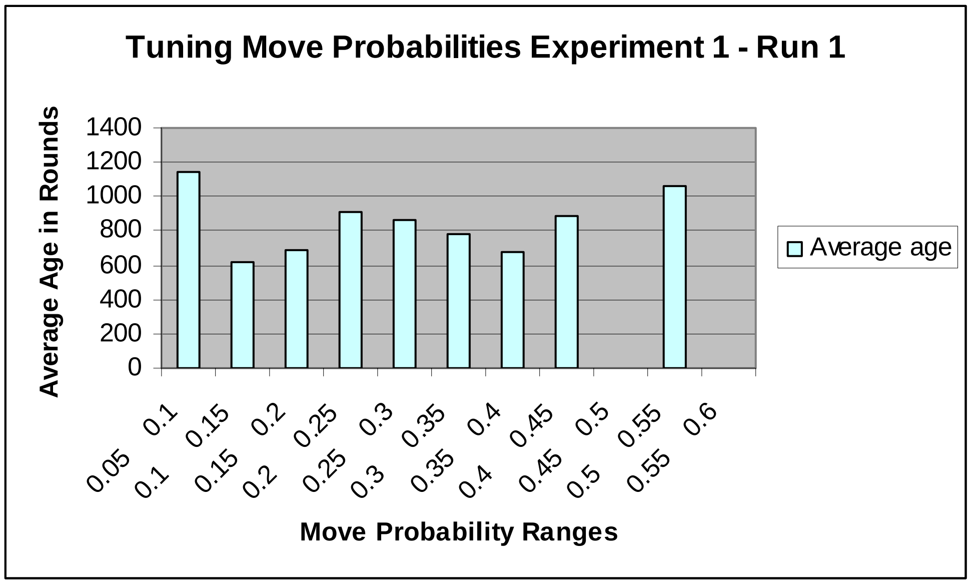
Tuning Move Probability

Default Move Probability	= 0.20
Move probability delta	= 0.03
Factor table	= 0,1,2,4,5,8,10,50,100,-1,-2,-4,-5,-8,-10,-50
p	= 0.001
q	= 0.005
v	= 10
u	= 100
M	= 500
K	= 80
m	= 25
n	= 20
Rounds	= 10000

Run 1

Raw Table

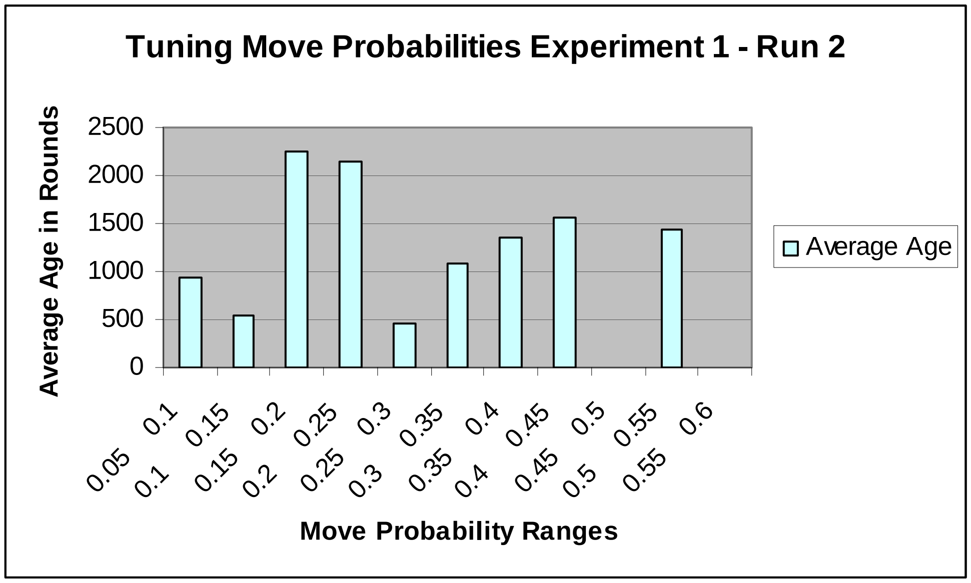
Move Probability Range		#Amoebas	Avg Amoeba Age
0.05	0.1	33	1143.60606060606
0.1	0.15	17	619.411764705882
0.15	0.2	10	688.8
0.2	0.25	106	915.745283018868
0.25	0.3	10	867
0.3	0.35	16	786.5625
0.35	0.4	13	676.846153846154
0.4	0.45	16	890.75
0.45	0.5		
0.5	0.55	15	1063
0.55	0.6		



Run 2

Raw Table

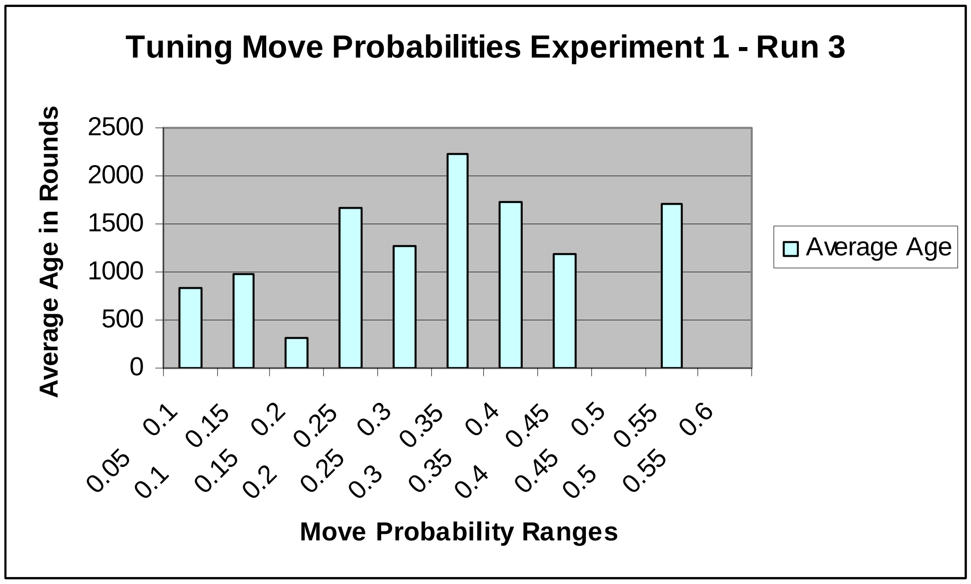
Move Probability Range		#Amoebas	Avg Amoeba Age
0.05	0.1	12	929.583333333333
0.1	0.15	13	550.076923076923
0.15	0.2	13	2253.61538461538
0.2	0.25	92	2155.44565217391
0.25	0.3	18	465.5
0.3	0.35	19	1077.68421052632
0.35	0.4	19	1348.36842105263
0.4	0.45	16	1557.5
0.45	0.5		
0.5	0.55	12	1427.5
0.55	0.6		



Run 3

Raw Table

Move Probability Range		#Amoebas	Avg Amoeba Age
0.05	0.1	27	837.407407407407
0.1	0.15	14	989.285714285714
0.15	0.2	11	321.545454545455
0.2	0.25	87	1665.34482758621
0.25	0.3	14	1262.64285714286
0.3	0.35	11	2227.90909090909
0.35	0.4	9	1731.55555555556
0.4	0.45	17	1189.47058823529
0.45	0.5		
0.5	0.55	9	1707
0.55	0.6		

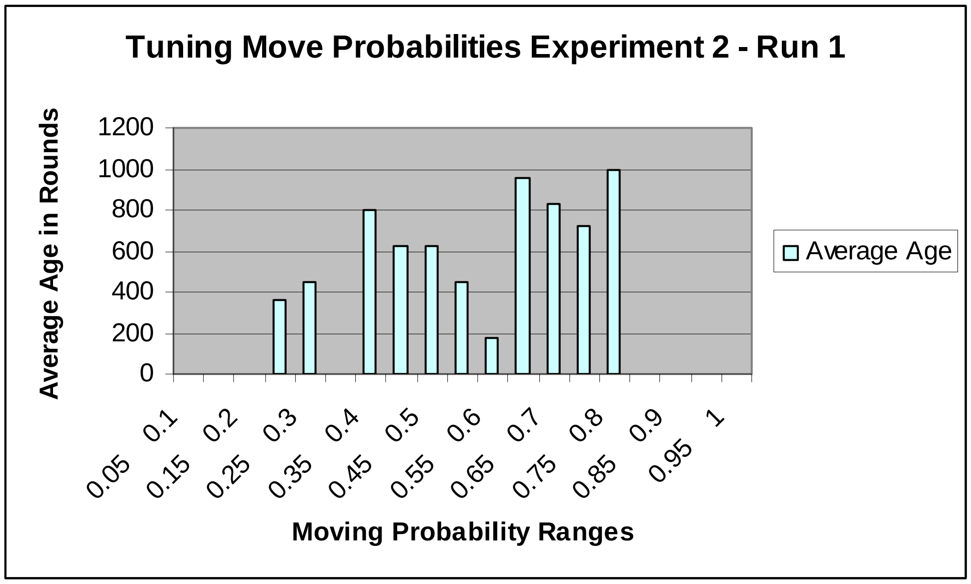


```
Default Move Probability= 0.50
Move probability delta  = 0.03
Factor table            = 0,1,2,4,5,8,10,50,100,-1,-2,-4,-5,-8,-10,-50
p                       = 0.001
q                       = 0.005
v                       = 10
u                       = 100
M                       = 500
K                       = 80
m                       = 25
n                       = 20
Rounds                  = 10000
```

Run 1

Raw Table

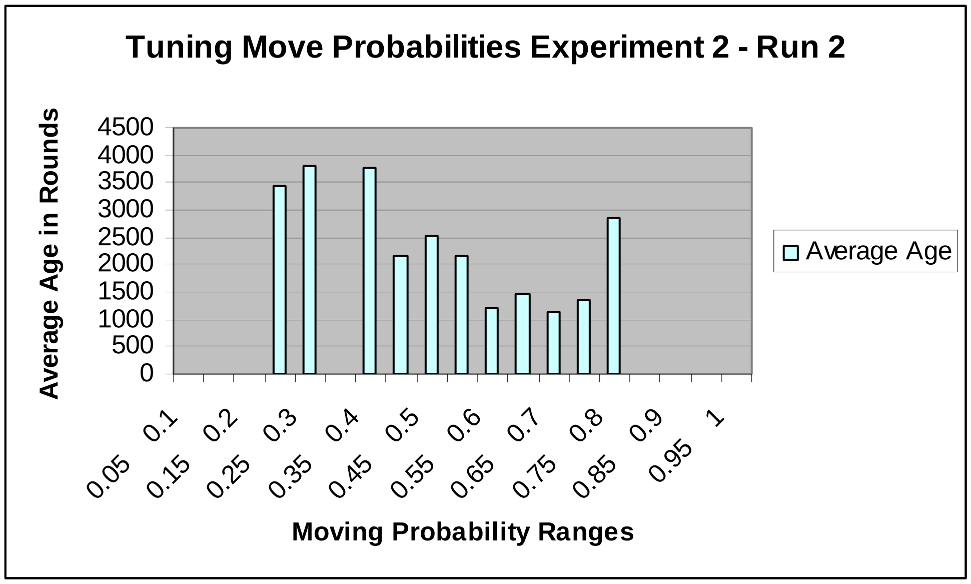
Move Probability Range		#Amoebas	Avg Amoeba Age
0.05	0.1		
0.1	0.15		
0.15	0.2		
0.2	0.25	14	363.714285714286
0.25	0.3	21	448.714285714286
0.3	0.35		
0.35	0.4	42	803.47619047619
0.4	0.45	27	619.925925925926
0.45	0.5	22	626.5
0.5	0.55	128	451.4765625
0.55	0.6	26	176.615384615385
0.6	0.65	19	956.736842105263
0.65	0.7	19	824.842105263158
0.7	0.75	19	723.894736842105
0.75	0.8	16	993.5625
0.8	0.85		
0.85	0.9		
0.9	0.95		
0.95	1		



Run 2

Raw Table

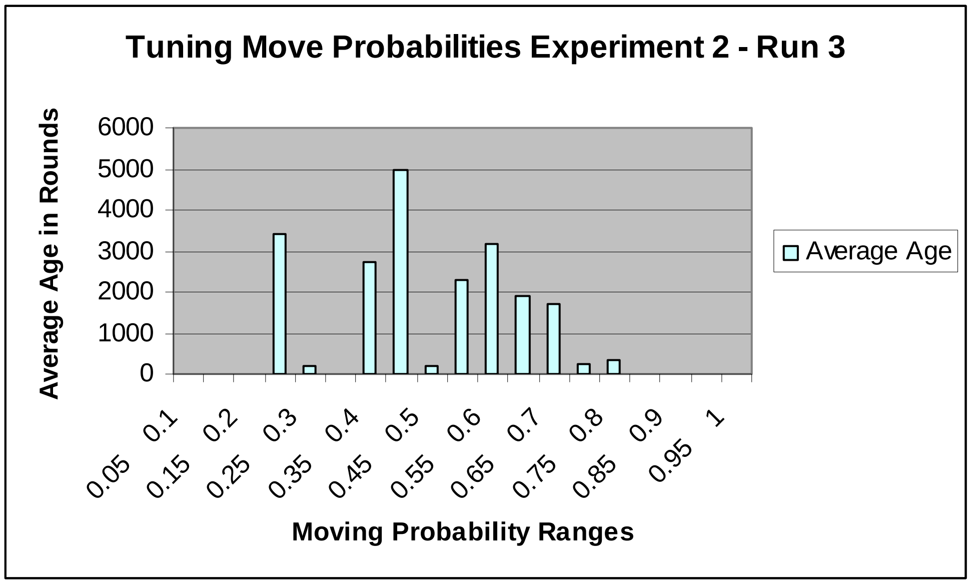
Move Probability Range		#Amoebas	Avg Amoeba Age
0.05	0.1		
0.1	0.15		
0.15	0.2		
0.2	0.25	11	3446.27272727273
0.25	0.3	8	3819
0.3	0.35		
0.35	0.4	25	3756.24
0.4	0.45	8	2168.375
0.45	0.5	11	2531
0.5	0.55	34	2151.32352941176
0.55	0.6	11	1200
0.6	0.65	7	1453.85714285714
0.65	0.7	9	1140.33333333333
0.7	0.75	6	1359
0.75	0.8	7	2837.28571428571
0.8	0.85		
0.85	0.9		
0.9	0.95		
0.95	1		



Run 3

Raw Table

Move Probability Range		#Amoebas	Avg Amoeba Age
0.05	0.1		
0.1	0.15		
0.15	0.2		
0.2	0.25	10	3430.7
0.25	0.3	5	197.8
0.3	0.35		
0.35	0.4	24	2723
0.4	0.45	12	4967.83333333333
0.45	0.5	9	206.222222222222
0.5	0.55	39	2288.89743589744
0.55	0.6	11	3148.18181818182
0.6	0.65	15	1913.86666666667
0.65	0.7	6	1700.5
0.7	0.75	6	266.166666666667
0.75	0.8	11	364.545454545455
0.8	0.85		
0.85	0.9		
0.9	0.95		
0.95	1		



Tournament Runs

Class Tournament Runs

Below are some sample results from the class tournament single player and multi-player runs. Based on the conservative foundations of our strategies we are not surprised by our overall performance. We consistently finish in the top five in terms of energy and we survive until the end of the simulation in single player games.

Our conservative strategy turns out to be our biggest weakness in multiplayer games. We are easily blocked in by opponents that aggressively reproduce early in the game. We do how ever put in a good fight holding them off until either the game ends or we are smothered [See Figure 3].

Single Player

```
[X=50, Y=50, u=100, v=10, p=0.0050, q=0.01]
Group0Player1           722.59:101580.1:-1
Group0Player3           763.75:101524.7:-1
Group0Player5           778.67:167761.0:-1
Group1PlayerFinal       132.84:28977.2:-1
Group2Player2           298.34:27577.0:-1
Group2Player3           999.55:192115.8:-1
Group3Player1           535.63:73418.1:-1
Group3Player2           510.86:114126.4:-1
Group4Player1           603.48:71237.6:-1
Group4Player2           720.49:97270.6:-1
Group5Player5           662.97:182340.1:-1   * 2nd Highest energy
Group5Player9           677.86:163138.9:-1   *
Group6PlayerAG2         755.8:127456.3:-1
Group6PlayerN           725.97:106215.0:-1
Group7PlayerFinal       790.81:168522.2:-1
Group8Player1           284.73:59142.9:-1
Group8Player2           717.61:160165.5:-1
Group9Player1           663.79:139538.5:-1
Group9PlayerJD          754.35:135562.9:-1

[X=40, Y=40, u=100, v=2, p=0.0080, q=0.0020]
Group0Player1           615.48:89288.7:-1
Group0Player3           682.8:92393.0:-1
Group0Player5           702.58:160518.2:-1
Group1PlayerFinal       612.79:160528.5:-1
Group2Player2           674.62:99069.0:-1
Group2Player3           578.87:75430.9:-1
Group3Player1           464.5:109066.5:-1
Group3Player2           456.24:105970.3:-1
Group4Player1           629.11:83731.5:-1
Group4Player2           696.61:94567.8:-1
Group5Player5           610.68:150442.2:-1 * 3rd Highest energy
Group5Player9           653.44:150621.4:-1 * 3rd Highest energy
Group6PlayerAG2         698.03:128466.7:-1
Group6PlayerN           664.54:104552.5:-1
Group7PlayerFinal       695.48:60230.7:-1
Group8Player1           530.31:123022.2:-1
Group8Player2           695.07:160515.7:-1
Group9Player1           691.85:147148.6:-1
Group9PlayerJD          698.25:141065.6:-1
```

```
[X=75, Y=75, u=100, v=10, p=0.0020, q=0.0]
Group0Player1            209.37:26292.7:-1
Group0Player3            709.5:92299.1:-1
Group0Player5            541.69:97708.5:-1
Group1PlayerFinal        103.35:22975.0:-1
Group2Player2            0.0:0.0:25
Group2Player3            0.0:0.0:23
Group3Player1            491.1:69671.4:-1
Group3Player2            484.29:110790.0:-1
Group4Player1            572.38:62516.6:-1
Group4Player2            664.27:78077.5:-1
Group5Player5            625.22:143829.3:-1  ** Highest energy
Group5Player9            1.0:343.2:-1        *
Group6PlayerAG2          0.0:0.0:499
Group6PlayerN            631.97:86640.0:-1
Group7PlayerFinal        713.53:126983.9:-1
Group8Player1            279.98:56348.5:-1
Group8Player2            695.0:153351.3:-1
Group9Player1            475.87:101951.4:-1
Group9PlayerJD           120.22:21709.9:-1

[X=25, Y=25, u=100, v=10, p=0.01, q=0.02]
Group0Player1            296.0:44510.1:-1
Group0Player3            286.04:38807.3:-1
Group0Player5            297.27:67696.9:-1
Group1PlayerFinal        66.93:15961.9:-1
Group2Player2            444.68:176051.8:-1
Group2Player3            246.53:33484.2:-1
Group3Player1            224.18:31260.0:-1
Group3Player2            197.98:45867.1:-1
Group4Player1            223.06:27993.4:-1
Group4Player2            273.43:38979.6:-1
Group5Player5            238.98:73755.3:-1 * 2nd Highest energy
Group5Player9            226.07:56601.1:-1 *
Group6PlayerAG2          273.1:49542.3:-1
Group6PlayerN            283.61:43957.9:-1
Group7PlayerFinal        288.22:58732.8:-1
Group8Player1            111.05:23720.3:-1
Group8Player2            274.29:63108.0:-1
Group9Player1            274.04:57743.8:-1
Group9PlayerJD           296.86:58462.5:-1
```

Multi-player

```
X=40, Y=40, u=100, v=2, p=0.0080, q=0.0020]
Group0Player1            0.0:0.0:4935
Group0Player3            96.93:13151.7:-1
Group0Player5            31.52:7150.3:-1
Group1PlayerFinal        27.93:6712.7:-1
Group2Player3            0.0:0.0:471
Group3Player1            0.0:0.0:2763
Group3Player2            0.0:0.0:2002
Group4Player1            28.44:3534.4:-1
Group4Player2            52.01:7276.1:-1
Group5Player5            67.26:15922.3:-1  *
Group5Player9            122.52:31301.3:-1 ** Highest energy
Group6PlayerAG2          0.0:0.0:283
Group6PlayerN            34.26:5328.8:-1
Group7PlayerFinal        26.09:2696.1:-1
Group8Player1            0.0:0.0:2898
Group8Player2            135.31:31296.9:-1
Group9Player1            0.0:0.0:697
Group9PlayerJD           38.35:7363.4:-1
```

References

[1] Project 4 Organisms II Group 1 2003 - Alexis Goldstein, Edan Harel, David Vespe

[2] Class notes 09-13 courtesy Abhinav Kamra

[3] Class notes 09-15 courtesy Abhinav Kamra

[4] Class notes 09-20 courtesy Abhinav Kamra

[5] Class notes 09-22 courtesy Abhinav Kamra

[6] Class notes 09-27 courtesy Abhinav Kamra