

University of Richmond

UR Scholarship Repository

Honors Theses

Student Research

4-23-1999

Shout with the largest Mob : toward a model for primitive communication in mobile automata

Rebecca A. Weber
University of Richmond

Follow this and additional works at: <https://scholarship.richmond.edu/honors-theses>



Part of the [Computer Sciences Commons](#), and the [Mathematics Commons](#)

Recommended Citation

Weber, Rebecca A., "Shout with the largest Mob : toward a model for primitive communication in mobile automata" (1999). *Honors Theses*. 495.

<https://scholarship.richmond.edu/honors-theses/495>

This Thesis is brought to you for free and open access by the Student Research at UR Scholarship Repository. It has been accepted for inclusion in Honors Theses by an authorized administrator of UR Scholarship Repository. For more information, please contact scholarshiprepository@richmond.edu.

UNIVERSITY OF RICHMOND LIBRARIES



3 3082 00687 9034

Mathematics/Computer
Science

Web

Shout With the Largest Mob: Toward a Model for Primitive Communication in Mobile Automata

Rebecca A. Weber
Honors thesis¹

Department of Mathematics & Computer Science
University of Richmond

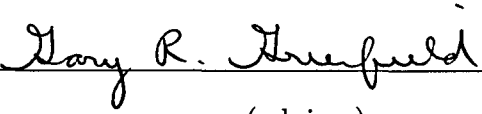
April 23, 1999

¹Under the direction of Dr. Gary R. Greenfield


Abstract

We consider the problem of simulating communication between independent, autonomous agents, or machines, using only local rules with no global control over the agents' behavior. First, we construct an algorithm by which the machines will avoid collisions with each other and with boundaries or obstacles. Noting that collision avoidance alone will not result in higher-level behavior, and with the goal of creating agents which would self-organize, we begin to develop a signalling system by which agents can communicate. This leads to a new method for modeling agent motion in the plane. Throughout, we are motivated by possible linkages between our agent models and the biological realm.

This paper is part of the requirements for honors in mathematics. The signatures below, by the advisor, a departmental reader, and a representative of the departmental honors committee, demonstrate that Rebecca Weber has met all the requirements needed to receive honors in mathematics.



(advisor)



(reader)



(honors committee representative)

1 Introduction

The theory of cellular automata has had a significant impact, as evidenced by the continuing interest in John H. Conway's game of "Life" [4] [7] [9] [10].¹ From the point of view of the theory of computation and formal models, however, progress towards understanding the computational nature of biological life, rather than the computations that can be performed via the mechanics of the game of "Life," has not proceeded at the pace one might expect. Cellular automata can be used to simulate universal Turing machines, and the study of computation in connection to CA's has been that of how such computations can be represented. The study of computation in connection to biological life must be at a higher level, where the mechanics of computation are hidden, and the *results* of the computations are focused upon.

Artificial Life, a field so named and promoted by Chris Langton [12], proposes that the study of digital organisms may be used to investigate biological life. Research in robotics, communication, and other fields has been rejuvenated with this new approach. There has been close to a decade of Artificial Life research, and strong focus has been put on global behavior, leading some to wonder if study of the underlying theory and rationale, in other words, the *formalism*, has been outpaced.

¹In personal conversation with Professor Conway on November 13, 1998 we learned first hand that Conway himself has grown weary of the heavy burden of fame inherited from his invention.

We shall investigate the simplest of formal models for mobile digital organisms — swarm automata — which must cope with solving such elementary problems as avoiding collision and developing rudimentary signaling and receiving capability. We are interested in a path of development that helps to illuminate the computational processing required to solve such problems and that demonstrates the integration of such traits into organisms. This is an endeavor that requires great care. The most complete formal model we are aware of [2] is seriously flawed.

The lack of formal models in the literature causes the question to arise as to whether or not formalism is a hindrance to producing results. Formal models do require more time and attention to detail, and so results are often slow in coming, but they allow the investigator complete control over every parameter of the model. Thus, as a secondary goal, we would like to determine whether there is a reason to avoid strict formalism; that is, whether there is some inherent limitation formalism imposes on models of high-level behavior.

Throughout our investigation, we are motivated and guided by possible connections to the biological world. We would like our model to be plausible as a model of a natural system, even though at the outset we have no specific system in mind.

2 Collision Avoidance on a Plane Tiled by Squares

The departure point for this thesis is a paper by Adamatzky and Holland [2] investigating excitation in systems of static agents and of mobile ones; in other words, cellular automata and lattice swarm systems. The intent of the model in [2] is clear, but the reader should be aware that there are several misprints. Even after attempting to correct the misprints, however, the formal model given seems incomplete and inconsistent, so it is not possible to duplicate the results found therein.

Adamatzky and Holland defined a lattice swarm system to be a collection of Turing machines with two external tapes that they mutually share. Each tape is two-dimensional, and each machine has two read-write heads, with one head on each tape. One tape is for direction state and one is for excitation state. On the direction state tape, the possible symbols are the eight compass directions, a symbol indicating “collision,” and a “blank” symbol the tape is presumably initialized with. The possible symbols on the excitation state tape represent excitation, rest, and a refractory state. The resting state is the default state; the excited state is induced by local conditions, and the refractory state is a brief period of unresponsiveness following excitation.

Each machine executes a three-step sequence during each clock cycle. It first reads the symbols on each tape in the squares on which its read-write heads are placed, as well as the symbols on the neighboring squares. Based

on this information, a state change may or may not occur. The machine then changes the position of each of its heads and writes a symbol on the square at each head's new position. The final position of each head and the internal state of the machine are dependent on the current direction and excitation state of the machine and its neighbors, as well as the value of an internal timer. When the machine is in the rest or refractory state, its timer has a value of zero. When the machine is initially excited, the timer is set to a pre-chosen integer, the *duration of excitation* parameter. The timer value decreases by one every time step, and when it becomes zero, the machine changes state from the excited state to refractory state.

The motion of a machine's heads is controlled by its direction state and the occupancy of neighboring squares. The machine's direction remains the same unless it detects a possible collision, meaning a situation in which the intended destination is occupied. If there is no collision, the machine moves in the direction indicated by its internal state. In the case of a collision, the machine changes direction according to a "deviation" function and moves forward in the new direction if the new destination is not occupied. If the new destination is occupied, as the original destination was, the machine remains in its original space until the next time step.

Two deviation functions are considered in the paper, each of which gives a set of directions from which the new direction of the machine is chosen probabilistically, based on its current direction. The π -deviation returns the

set containing the opposite direction and the directions 45 degrees to either side of the opposite direction, so that North would return {South, Southeast, Southwest}. There is equal probability of choosing any direction from this set. The α -deviation returns the set containing the original direction and the two directions 45 degrees to either side, as well as the two directions 90 degrees away from the original direction. The three former directions are given more weight than the two latter, which were included solely to prevent wall-following.

We now consider our modifications to the design in [2]. Because our model strove for simplicity and was directed at following rather than simple group dynamics, we did not adopt the probabilistic method of initiating direction change. Our machines use only one read-write head and one two-dimensional external tape. For visualization of motion dynamics we view the machine as occupying a particular square on the tape itself, instead of just the read-write head of the machine doing so. The formal model we next describe appears in the Appendix as Model 1A.

Since the first task the machines must accomplish is collision avoidance, it would be counter-productive for two machines to approach each other and either freeze or attempt to pass through each other. A machine must also not be disabled by a wall or other obstacle. To this end, each machine has two internal states, σ_D^t and σ_C^t . The first, σ_D^t , is the internal directional state at time t . It can be any of the eight cardinal or diagonal directions,

either represented as letters for ease of interpretation (e.g. N, E, SW), or as ordered pairs for ease of computation (e.g. (0, 1), (1, 0), (-1, -1)). The state σ_G^t is the internal collision state at time t . It has two possible values, Λ as the default, and $\#$ when the machine detects a possible collision. We call $\#$ the *collision alert state*, and say that a collision has been *resolved* when the machines involved have turned sufficiently to have a destination which is unoccupied and which, furthermore, no other machine has as its destination.

Each square of the external tape, which we think of as the environment for the machine, has one symbol printed on it. Ordinarily that symbol is Λ , but it can also be any of the directional symbols or special wall symbols. The wall symbols are h , or (0, 3), for a horizontally-running boundary, and v , or (3, 0), for a vertically-running boundary.

At initialization, the external tape square occupied by a machine has written on it that machine's current direction. At the beginning of each time step, the machine uses (0, 0) to refer to the space it occupies. This simplifies computation, as all coordinates are then relative to the square the machine occupies. The machine's destination square then has the same coordinates as its direction state. The relative coordinates of the squares surrounding the destination square can be added to the directions written on those squares to get the destinations of the other machines occupying them. In this way one machine can determine if another has the same destination. For example, let machine A be the machine whose perspective we take, that is, the machine

which to us is at $(0, 0)$. Moreover, let machine A be moving to the north, so its direction state is $(0, 1)$. Therefore $(0, 1)$ are also the coordinates of A's destination square. Now let machine B be in the square with coordinates $(1, 1)$ in machine A's coordinate system. Say that machine B is moving northwest, so its direction state is $(-1, 1)$. Machine A reads $(-1, 1)$ written on square $(1, 1)$ and calculates that machine B's destination is $(0, 2)$, and thus discovers that there is no danger of collision between the two machines. If machine B's direction were west, or $(-1, 0)$, machine A would calculate B's destination as $(0, 1)$, the same as its own, and would therefore go into the collision alert state.

During each clock cycle, the machine executes a sequence of four steps. First, it reads the symbol on the square it occupies and changes its internal state. Second, it writes the "blank" symbol, Λ , on its current square. If the machine is not in the collision alert state, its third step is to move to its desired next square. Finally, it writes its new direction, σ_D^{t+1} , on its new position. The state change in the first step is governed by the destination square and the squares surrounding it. If the machine does not detect a possible collision, the new direction state is the same as the direction written on the square the machine currently occupies. However, if there is another machine occupying the destination square, or another machine with the *same* destination square, the new collision state is $\#$, and the machine's new direction state is 45 degrees counter-clockwise from that written on the square

it currently occupies. The machine also goes into the collision alert state if the destination square is a wall, and changes its direction objective so that it reflects off the wall as if it were a pool ball. That is, it turns 180 degrees if it strikes the wall head-on, and 90 degrees if it strikes the wall at an angle. All rotations are accounted for by a change of internal state, without motion on the plane. Thus, turning effectively pauses the machine for one time step, or for two time steps if the machine has reached a corner, since it reflects first off one wall and then the other.

Since our model does not incorporate probabilistic direction change, as long as machines are not colliding with each other, there are fixed “paths” on which they move. On a space with a square boundary, the paths that machines follow all eventually trace out rectangles, with sides parallel either to the sides of the boundary, or to the diagonals connecting opposite corners of the boundary. The non-degenerate rectangles, that is, those with four distinct sides, all have edges parallel to the diagonals. All paths with sides parallel to the edges of the boundary are lines, or degenerate rectangles, a category into which diagonal paths running from corner to corner of the space also fall. In a space whose boundary is not square, the paths may be more complicated (Figure 1). There may, in fact, be no non-degenerate rectangular paths, but degenerate paths will still exist.

When there is more than one machine occupying the plane, they may collide multiple times before settling into steady paths. The possibilities we

must consider are that the paths could be distinct for each machine, the machines may be on the same path, in a sense *following* each other, or there may be a configuration which results in periodic collisions. We conjecture that there is a finite number of collisions possible between two machines on the plane, *i.e.*, that there does not exist a configuration leading to periodic collision. We were unable to prove this conjecture, but we did not find any periodic configurations in our attempts. Using two machines on a 5×5 square, the largest number of collisions we found for any initial configuration was six.

3 Where Are You? The Need for Signaling

The next question we investigated was whether two machines could enter into a following pattern when equipped with only the collision avoidance algorithm. For following to occur, there would have to be one collision which set the machines on the same path. There is a relatively small finite number of collision configurations possible between two machines on the plane, so we examined them all and found that only a head-on collision where two machines were facing each other heading North-South or East-West, with one empty square in between them, could potentially put two machines on the same path (Figure 2). All other collisions, independent of the size or shape of the boundary of the space they are in, were resolved with the machines tracing out different non-diagonal paths, a diagonal path and a non-diagonal

path, or two distinct diagonal paths. We determined, though, that again independently of the boundary of the space, the sought-for head-on collision can occur only if the machines are already on the same path (Figure 3). That is, the steps leading up to a North-South or East-West head-on collision all occur on the same line as the collision. This is independent of the size and shape of the space, because a machine on a horizontal or vertical line cannot leave that line without invoking collision avoidance.

It is clear, then, that some form of communication between the machines is necessary if they are to appear to work in concert. Our goal has been to keep that communication as simple as possible to preserve the non-deterministic nature of the system, so we started with a simple "Here I Am" call. The formal model was extended by giving each machine a timer and an additional internal state, and modifying the external tape so that it held two symbols per square (Model 1B). The second tape symbol is the call symbol, usually \cdot (no call), but $+$ when a machine within range is calling. The additional internal state distinguishes between two types of machines, one which calls and one which listens. The idea is for the listener to freeze when it hears a call, and not to resume motion until a fixed length of time has passed. The caller, or emitter machine, has two calling states: on and off. The states are controlled by the internal timer, which counts down from two to zero, and then resets to two. The emitter machine is *on* when the timer is nonzero, and *off* when the timer is zero. In the *on* state, it writes a $+$ to every square

in its calling “neighborhood.” The listener, or receiver machine, also has two states: paused and unpaused. Normally, the receiver machine is unpaused. When the receiver reads a + on the square it occupies, it pauses and sets its timer to a predetermined maximum value, which then decreases by one with every time step until it is zero again. The receiver remains paused as long as its timer is nonzero, and the timer is reset to the maximum value every time the receiver reads a +.

Since the receiver is paused when in close proximity to the emitter, it is probable this will affect the number of times the collision avoidance algorithm is invoked. The emitter machine, of course, will be unchanged in its use of collision avoidance, but the receiver machine, when paused, will not check for potential collisions and thus will not turn. We discovered that some configurations of two machines on a 5×5 square space had a decreased number of collisions before resolution into a stable motion pattern, as compared to resolution under Model 1A, and some had an increased number of collisions. However, pausing did not cause following.

We next looked for a correlation between the direction of emitter movement, the point of entrance of the receiver into the emit neighborhood, and length of receiver pausing. For a 5×5 emit neighborhood centered at the emitter and for a 3×3 emit neighborhood oriented “ahead” of the emitter, in the direction of its movement (Figure 4), there were no discernable correlations (Table 1). The 3×3 emit neighborhood also highlighted the

inherent asymmetry of the square-tiled plane. An emitter moving North, South, East, or West will emit to five squares adjacent to it, and to three squares at a distance of two squares from it. An emitter moving Northeast, Northwest, Southeast, or Southwest will emit to three adjacent squares and to five squares which are two squares away from it.

4 Symmetry: Using a Plane Tiled by Hexagons

The plane tiled by squares is a convenient and seemingly natural environment for our machines, with intuitive notation, but as we have seen it lacks parity between a neighborhood oriented in a cardinal direction and one oriented along a diagonal. To eliminate this problem and introduce symmetry, we turn to the plane tiled in hexagons, which has fewer directions to choose from, but no discrepancies between neighborhoods oriented in any of the directions. The basic collision avoidance model needs little modification to be adapted to the “hex plane” (Model 2A).

The most significant change to the model is a notational one. Since there is no standard way to abbreviate directions based on sixty degree increments, as there are compass points for directions based on forty-five degree increments, and no simple way to use ordered pairs to denote locations, the machines now operate on six-bit direction strings. Such a string will consist of five zeros and a single one, with the position of the one denoting the direction of intended movement as follows: down-right, right, up-right, up-

left, left, down-left. Thus the bitstrings 100000 and 000100 indicate opposite directions, as do all pairs with ones three places apart. Since moving down-right and then up-left, or any other sequence of two movements in opposite directions, leaves a machine in the same place it started from, adding these pairs must give 000000, a net effect of leaving the machine stationary. Location on the plane is also indicated by strings. Each machine, as in Models 1A and 1B, uses relative coordinates, viewing itself in hex-tile 000000 at the beginning of each time step. Location strings may have more than one position entry set to one, or entries set to values higher than one, to indicate hex-tiles not immediately adjacent to the machine. The location strings are interpreted as the result of the motion which would result from direction bitstrings which sum to the location string. For example, the string 210000 is interpreted as the hex-tile a machine would be in after moving down-right twice and right once (Figure 5).

Each hex-tile on the plane has a six-bit direction code written on it. The tape representing the plane is initialized with 000000 written in all spaces; this is equivalent to the Λ of the internal collision state. The machines follow the same sequence of four steps as in Model 1A. First, they read the direction written on their hex-tile — the direction they wrote the previous time step — and change state according to what is written on the hex-tiles around their destination. Second, they write 000000 on their hex-tile, and then, if they are not in the collision-alert state, move to their destination. Lastly,

each machine writes its new direction state on its new hex-tile.

Collisions are handled just as in the square-tile model. The machine enters the collision-alert state when its destination is a wall or is occupied by another machine, or when there is another machine with the same destination. A machine will turn 60 degrees counter-clockwise in the case of collision with another machine. Since, in the hex plane, there are no directions of movement which take the machine through the vertex of a tile's edge (as the diagonal movement in the square plane would), all collisions with walls are head-on, and in the hex-tiled case, the machine pivots 180 degrees.

Because of the constant 180 degree bounce off of walls, the stable path for a single machine is a line, which we recall is the degenerate case of the plane tiled in squares. An investigation of the possible collision configurations between two machines showed that there are no collisions which can put the machines on the same path (Model 2A). As in the case with the square-tiled plane, we conjecture there is only a finite number of collisions possible between two machines, and our search for a configuration with periodic collision failed to turn up any counterexamples.

Having determined that following behavior could not result from collision avoidance alone, we again added a signaling system (Model 2B). Signaling was modeled by adding an additional six-bit code to each hexagon on the plane, for the emitting machine's use, and an additional pause-bit to the direction code on each hexagon. Each hex-tile, then, holds a six-bit code

which the emitter will use to broadcast its presence, and a seven-bit code each machine will use to broadcast its intended direction and whether or not it is paused. As in the square-tiled model, receiving machines will pause when they detect a signal from the emitting machine, and remain paused until they either fall out of range for a sufficiently long time, or successfully determine the direction of the emitter machine's motion.

The model also incorporates an additional internal state for the emitter, and additional values for the receiver's internal collision state. The emitter machines have an internal emit state, which can be *first*, *second*, or *off*. The emitter calls when it is in *first* or *second* state, and is silent when it is in *off* state. The three states cycle so that *first* follows *off*, *second* follows *first*, and *off* follows *second*. Receiver machines have two additional possible values for their internal collision state. They retain the default state, Λ , and the collision alert state, $\#$, but they also can be in the *following* state, Ω , indicating they are following the emitter, or in the *following-collision-alert* state, \natural , which is to Ω what $\#$ is to Λ , and will be explained shortly.

The emitting machine writes to a convex neighborhood of hex-tiles consisting of five rows of tiles running parallel to the emitter's direction of movement. The center row, which is a total of seven hex-tiles long, extends four tiles ahead of the emitter and two tiles behind it.² The two rows directly next

²Initially we had the neighborhood extending only one hex-tile behind the emitter, but determined later that swarm size increase was hampered by that restriction. Temporary emitters (see next section) would then join the swarm only out to the sides, giving the whole the appearance of a flock of geese.

to the center row are six hex-tiles long, and the two outermost rows are five hex-tiles long. The bitstring it writes to each space is similar to a direction string (Figure 6). The string on each hex-tile is the sum of the direction strings indicating the directions the emitter must move in to reach that hex-tile. That is, if there is a straight-line path from the emitter, there will be one bit set to one, indicating the direction of the path, and if there is no straight path, there will be two bits set to one, indicating the two directions that must be taken by the emitter.

There is a simple set of computations the receiver must make to determine the direction of the emitter's movement (Model 2B). It will not always be possible to perform the computations, as the receiver may be in the emit neighborhood for only one time step, or only one time step during which the emitter is active. However, in most cases the receiver will be able to calculate the emitter's direction with two successive emit cycles, at which time it will enter the following state, Ω . The receiver will then remain paused until it is in one of the ten hex-tiles in the emit neighborhood behind or alongside the emitter machine (Figure 7), and begin motion in the same direction as the emitter. It will remain in the following state as long as its timer is nonzero; that is, as long as it remains in the emit neighborhood.

There are certain restrictions which must be made on receiver movement when it is in the following state. It must not get knocked off course when it is following, and also must not leave the following state if it is still in the emit

neighborhood and moving in the correct direction. Hence we introduced the following-collision-alert state, \natural , which causes a 180 degree turn upon any collision, and reverts to the following state when the collision is resolved. This allows for swarms (the followers attracted by an emitter) to form which are more than one machine deep, so that when the swarms collide with a wall, they will reflect back neatly (Figure 8). The machines will end up in reversed order, but will not fly off at 60 degrees to avoid collision with each other. The only exception to this is the emitter, which *will* make a 60-degree turn when colliding with another machine. Therefore the only stable configurations are those with no other machines in the emitter's line of motion.

Configurations which are stable, though, are very stable. Since a temporary emitter turns 180 degrees when it collides with a wall or another temporary emitter, even if it leaves the swarm it will rejoin after it and the swarm have both reflected from the walls. A receiver just entering the swarm might collide with a temporary emitter, but the receiver will be paused at the time and so it will not be affected. The temporary emitter would have to collide again once it has reverted to the default state (Λ) to be knocked out of the swarm's line of movement. When the isolated machine rejoins the swarm, it will cause a 180-degree deflection of the machine which initially deflected it out of the swarm. Eventually we will have a stable but dynamic, or fluid, swarm where machines deflect each other 180 degrees upon rejoining the group, after having been deflected themselves.

5 The More the Merrier: Temporary Emitters and Swarm Size Increase

All of the care we have taken to preserve following behavior in swarms that are several machines deep is necessary only if we can create such swarms. The fixed size of the emit neighborhood puts a limit on the number of machines which can be in the following state at any one time. Since the following state is preserved only while the machine's internal timer is nonzero, the machine must be physically in the neighborhood of the emitter. We would like for there not to be such a small limit on the size of the swarm, and in fact we would like to have the potential for swarms to reach any arbitrary size, given enough receivers.

In order for the swarm to grow larger, receivers farther out must be able to learn about the emitter's signal. One way to accomplish the "spreading of the word" is for receivers which are in the following state to become emitters as long as they are following. We call such receivers "temporary emitters," and distinguish them from the "main emitter" only by the fact that temporary emitters do not have the capacity to turn 60 degrees upon collision, but will always turn 180 degrees.

A non-following receiver, then, can start following via either the main emitter's signal or by the signal of any of the temporary emitters. This allows for growth of the swarm to be limited only by the number of machines available. However, there is a problem: the temporary emitters do not po-

tentially *need* the main emitter in order for them to stay in the following state. Once two temporary emitters are created, if they are in each other's neighborhoods, then they will keep each others' timers set and thus keep each other in the emitting state. If the main emitter were to shut down or to be knocked off course, there is the potential for "perpetual motion:" the two temporary emitters will be emitting to each other, unaware that the signal they "want" to be following is no longer there. It is undesirable to have these small splinter groups.³

In order to prevent a collection of temporary emitters with no main emitter, we introduced a second timer, our "long timer," as opposed to the already-established "short timer." The long timer is an automatic shutoff for the temporary emitters. It is set to a predetermined maximum value when the machine first goes into following state, and decreases by one with every time step. When it reaches zero again, the machine goes back to the non-collision receiver state; that is, Λ . The machine will then pause when it reads a signal and will need to recalculate the direction of emitter motion. If the short timer is short enough, say, three time steps, the existence of the long timer will prevent perpetual motion. In the case of two machines, when one's long timer runs out, the other will only emit until its short timer runs out, which should not be long enough for the first to re-follow. When there

³Hence the title of this thesis, from Dickens' *Pickwick Papers*: "It's always best on these occasions to do what the mob do." "But suppose there are two mobs?" suggested Mr. Snodgrass. "Shout with the largest," replied Mr. Pickwick.

are more than two machines, we see a “ripple effect” as the long timers on machines which began following earliest run out, then the long timers belonging to later additions to the swarm run out. It is possible with a larger group to have re-following by the earliest of the swarm members, but all the machines will eventually return to receiver state.

6 This Noisy World: Signal Discrimination in a Swarm of Signalers

There is an additional problem posed by a swarm of signaling machines; namely, noise. Where the emit neighborhoods overlap, so do the signals, and it would be difficult for a machine to calculate the direction of the group’s movement. For example, if a receiver reads 111001, it will not automatically be able to ascertain whether that code is the result of one machine writing 100001 and another writing 011000, or one machine writing 101000 and the other 010001.

There is biological precedent for allowing the machines to differentiate signals. In a crowded room, one individual can pick out another’s voice without difficulty. There are many abilities contributing to this one, including proximity, voice recognition, triangulation, and sight. Primarily, though, it is the fact that language consists of only a small subset of all possible strings of letters, spaces, and punctuation that allows humans to pick out one signal. Change a few letters in a sentence, and in all likelihood, the

string you end up with is not a valid sentence. A person hearing one of these altered strings of letters can figure out what valid sentence the string is closest to and thus what was meant. The same is not true of bitstrings — no string of two ones and four zeros or a single one and five zeros that a receiver examines has any more validity than any other. With the biological precedent that animals have signal discrimination, however, we can justify using computational mechanisms to give machines the same ability.

Even with signal discrimination, we are still concerned with keeping the main emitter's signal from getting lost in the crowd. In effect, we would like for the main emitter to be able to shout. This can be implemented in the model as an additional bitstring on each hex-tile, which is reserved for the main emitter's use. It is a high (or *loud*) channel the main emitter can dispatch a signal to, avoiding the temporary emitter static of the low channel. The receivers within range can then read from that channel to calculate the direction of the main emitter's movement.

A limited number of machines, however, can be in range of the main emitter. Continuing with the shouting analogy, we might increase the range of the emit signal. However, we would like for the emit neighborhood to increase only as necessary, which requires that the main emitter know how many machines are following it. This appears to be a delicate problem. The simplest way to solve it is to allow the main emitter the ability to sense all machines in its emit neighborhood, and then increase the size of the

neighborhood as necessary, according to the number of following machines, in order to provide room for additional swarm members. That ability, though, would be the second added functionality that distinguished the emitter from the receiver: not only could it call louder and farther, but it could also hear or read from farther away. We would like to minimize the difference between emitters and receivers, since there is no difference in the sensory abilities of the lead animal in a flock of birds or school of fish from those of the rest of the animals in the group.

Another potential solution involves adding a counter to the low channel, the channel the temporary emitters use. Each temporary emitter would make a record of its attempt to write to a hex-tile by adding one to the counter. Then, if at the end of the time step, three machines had tried to write to the hex-tile, the counter would read 3. This in and of itself does not help the main emitter determine how many machines are following it; all of the temporary emitters would have to be writing to spaces the main emitter could read, and for that to occur the swarm would necessarily have to be small. The counter could theoretically be used by the temporary emitters to communicate the number of machines following, though. A temporary emitter could read the largest value written in its emit neighborhood, add one to it, and use that as the value it adds to the hex-tiles around it. However, it is difficult to conceive of an implementation for modeling this idea which would not cause the values to snowball, producing numbers much larger

than the actual number of following machines due to counting each machine multiple times. It is clear that a better solution is necessary.

In passing, we also considered the possibility of using two clocks. One clock would synchronize the time steps we have worked with all along — each machine would be able to move once per time step. The other clock would run faster, with several of its time steps elapsing in every tick of the first clock. The fast clock would govern communication, so that several signals could occur between each motion time step. This would eliminate machines pausing while changing direction to resolve collisions, and would provide the opportunity for call-and-response signaling.

It would seem that call-and-response signaling could provide for a relay system, whereby temporary emitters could determine how many machines lie between them and the main emitter, and the main emitter could determine how many machines are following it. There are logistical problems associated with a relay, however. The temporary emitters must be able to discern the shortest path to the main emitter, and the main emitter must be able to determine when it has received all of the responses it is going to get, without repeat responses. We did not explore the two-clock model thoroughly enough to test solutions to each of the above, or to determine just how much of a difficulty there is in describing them in a formal model. We suspect that the shortest-path temporary emitter problem would be easier to solve, and the number of following machines problem would be more difficult to solve. Of

course, if the temporary emitters know whether or not they are following the main emitter, they would no longer need to read the main emitter's signal, so the relay eliminates the need for shouting. Without shouting, it would not be necessary for the main emitter to know how many are following it, and thus not necessary to solve the difficult problem of determining that value.

7 Doing it Backwards: Scattering

The problem of main emitter shutdown was considered partly as a way to clarify the dynamics of the temporary emitters, but it also has an analogy to the real-life systems we are attempting to mimic. If the leader of a flock of birds or school of fish should be killed or in some other way disabled from calling, would the group maintain its cohesiveness? A related issue which we have not explored in depth is that of developing a "scatter signal," some special call from the emitter machine which would cause the receiver machines to immediately stop following and move off in other directions, breaking up the group rapidly. The receivers would need to first recognize the signal as a call to scatter, which could be accomplished by adding a seventh bit to the emitter's signal bitstring, which would be zero normally but set to one for the scatter signal. Upon recognizing a scatter signal, the receivers would then need to determine a direction to move in. Since the direction bits in the emitter's call that the receivers use to determine following direction would be unchanged from following call to scatter call, each receiver could

read the direction from the emitter to its hex-tile, and move away from the emitter. The direction the scattering receiver moves in could simply be the direction of the path from the emitter (with a set choice when the direction bitstring has two bits set to ones), or it could be a pseudo-random direction calculated using the receiver's current timer value. We must note that this method of scattering was conceived of using an emitter with an increasing emit neighborhood size (although the mechanisms for increasing the neighborhood were unclear), and modifications would have to be made to adapt scattering to a model incorporating relaying.

8 Keeping it Real: Microtubules

The model developed in this thesis is of a system of autonomous, deterministic machines, but the problem of creating communicating agents which will collect into groups may be considered using a nonautonomous model. The analogy of our autonomous model to a flock of birds or other social organization of animals is an imperfect one — the machines are deterministic, and the animals are not. That is, a bird put into the same environment or given the same stimulus twice will not necessarily react the same way both times, but our machines will produce the same results every time they are put into the same initial configuration. Animals are autonomous, however, so an analogy between animals and a *nonautonomous* deterministic model is even less appropriate. One must look to a lower level of biological complexity

than birds or fish. In our search for appropriate biological systems to apply our model to, we found microtubules.

Microtubules are one of the primary components of the cytoskeleton. They also make up flagella and cilia, but the organization of those two structures is much more rigid than that of the cytoskeleton and so less applicable to the model. Microtubules are composed of tubulin dimers, which form long strands, in which the dimers are oriented in a helical fashion. The dimers are formed of one α -tubulin molecule and one β -tubulin molecule, and are attached to the “plus end” of the growing strand with a consistent orientation, giving the strand polarity. In the cytoskeleton, the “minus end” of the strand is anchored to a microtubule organizing center (MTOC), which is in fact the base upon which the strand is built. In the cytoskeleton, the MTOC is in the centriole. It has recently been determined that the “active ingredient” of the centriole, the MTOC itself, is an offset, or broken, ring of γ -tubulin, a form of tubulin which is not incorporated into the main body of the strand [15]. It is hypothesized that γ -tubulin will attach to only one of the other two forms of tubulin, and thus to only one end of each dimer, which would give the strand its polarity.

Hameroff et al modeled microtubules using cellular automata [12], using dimers within already-formed microtubule strands as stationary cells. They noticed that if the helix of dimers which forms the strand is cut along the strand and flattened, the result is a twisted hexagonal pattern (Figure 9).

Each cell has six neighbors, two to each side, one to the top, and one to the bottom, although the neighbors to one side are higher than those to the other side. The primary appeal of microtubules as a basis for Hameroff et al's model is that each dimer has two possible states, the α -state and the β -state, depending on whether a free electron is oriented to the α -tubulin end of the dimer or to the β -tubulin end. Which state a given dimer is in depends on its previous state and the state of the dimers surrounding it, based on electrostatic influence. Based on these rules, Hameroff et al were able to generate robust gliders, blinkers, and selectively-propagating patterns in either of the states against a background of the other.

The microtubule organizing center is the primary appeal of microtubules as a motivation for our nonautonomous model. The MTOC is composed of essentially the same components as the rest of the microtubule, but with a specific configuration and a certain "activation" over the remainder of the strand. In that way it is similar to a grouping of receiver machines becoming active as an emitter conglomerate when they are in the appropriate configuration. We believe it is a problem best considered as a nonautonomous model, because the logistics of moving machines into the correct places and having them "know" when to start emitting would be difficult at best in our formal, fully autonomous model.

9 Conclusions

The study of signaling and of the emergence and integration of different sensory channels in Artificial Life organisms is an active area of research. In fact, the forthcoming 1999 Genetic and Evolutionary Computation Conference will feature a workshop entitled "Evolution of sensors in nature, hardware, and simulation." The use of formal models is rare, however. There are certainly drawbacks to using formal models: every aspect of the model must be explicitly stated and thoroughly checked for consistency, and that causes refinement of the model to take place slowly. The reward, though, is that the model is then precise and complete, and the investigator has complete control of all parameters and complete knowledge of what each parameter affects. If unexpected behaviors arise within the model, it is possible to find their source, and all results are reproducible. We find no reason to avoid formalism, that is, no inherent limitation to it. In short, formal models show great promise, but are an underutilized resource in the study of Artificial Life.

The refinement and analysis of our model is far from complete, but there are some conclusions to be drawn from our work so far. We found that symmetry is important on a very basic level of the model. Without it, there is a need for separate rules for the diagonal and cardinal directions, a condition not consistent with our intuitive understanding of motion in the plane.

There are strong possibilities for making connections between biological

systems and this model. Microtubules seem to be the most easily related, but there could be others of a similar complexity level. It is possible that the behavior of unicellular organisms such as amoebas could be mimicked with modifications to the original model, since simple organisms are more hardwired, or deterministic, than complex ones like birds and fish.

References

- [1] A. Adamatsky, *Identification of Cellular Automata*, Taylor & Francis Inc., Bristol, PA, 1994.
- [2] A. Adamatsky and O. Holland, Phenomenology of excitation in 2-d cellular automata and swarm systems, *Chaos, Solitons & Fractals*, **9** 1998, 1233–1265.
- [3] P. Arnaud, Group locomotion of mobile robots based on auditory information, C. Wilke, S. Altmeyer, and T. Martinetz (eds.), Verlag Harri Deutsch, Frankfurt am Main, 1998.
- [4] E. Berlekamp, J. Conway and R. Guy, *Winning Ways for Your Mathematical Plays, Volume 2*, Academic Press, Orlando, FL, 1985.
- [5] S. Cowan, Rudy Rucker's CA lab, *Pixel*, **1** January/February 1990, 36–38.

- [6] A. Dewdney, Five easy pieces for a do-loop and random number generator, *Scientific American*, **252** April 1985, 20–30.
- [7] A. Dewdney, The game Life acquires some successors in three dimensions, *Scientific American*, **256** February 1987, 16–24.
- [8] A. Dewdney, The cellular automata programs that create wireworld, rugworld and other diversions, *Scientific American*, **262** January 1990, 146–149.
- [9] M. Gardner, The fantastic combinations of John Conway’s new solitaire game of “life”, *Scientific American*, **223** October 1970, 120–123.
- [10] M. Gardner, On cellular automata, self-replication, the Garden of Eden and the game of “life”, *Scientific American*, **224** February 1971, 112–117.
- [11] M. Gardner, *Wheels, Life, and Other Mathematical Amusements*, W. H. Freeman, New York, NY, 1983.
- [12] S. Hameroff, S. Rasmussen and B. Månsson, Molecular automata in microtubules: basic computational logic of the living state? *Artificial Life, SFI Studies in the Science of Complexity*, Ed. C. Langton, Addison-Wesley Publishing Company, 1988.
- [13] B. Hayes, The cellular automaton offers a model of the world and a world unto itself, *Scientific American*, **250** March 1984, 12–21.

- [14] R. Herken (ed.), *The Universal Turing Machine : A Half-Century Survey*, Springer-Verlag, New York, NY, 1995.
- [15] B. Oakley, A nice ring to the centrosome, *Nature*, **378** 7 December 1995, 555-556.
- [16] C. Pickover, *Mazes for the Mind: Computers and the Unexpected*, St. Martin's Press, New York, NY, 1992.
- [17] J. Pulsifer and C. Reiter, One tub, eight blocks, twelve blinkers and other views of life, *Computers & Graphics*, **3** 1996, 457-462.
- [18] T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*, MIT Press, Cambridge, MA, 1987.

Model 1A: A mobile automaton with collision avoidance, but no other form of “communication.”

Notation:

p^t : position of machine at time t .

σ^t : internal state at time t , an ordered pair (σ_C^t, σ_D^t) , where

σ_D^t : internal direction state at time t ,

σ_C^t : internal collision state at time t .

$D = \{(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1)\}$,
also written as $\{N, S, E, W, NE, SE, NW, SW\}$.

$C = \{\Lambda, \#\}$, where Λ is the default state and $\#$ is the collision alert state.

$N(\alpha) = \{\alpha + a(1, 0) + b(0, 1) : a, b \in \{-1, 0, 1\}\}$, the neighborhood of square α which includes α and the eight squares surrounding it.

s_α = symbol on square α . Note that s_α is an ordered pair,

$s_\alpha \in D \cup \{\Lambda, h, v\}$, where

$\Lambda := (0, 0)$,

$h := (0, 3) \equiv$ horizontal boundary (across the top or bottom),

$v := (3, 0) \equiv$ vertical boundary (down each side)

Execution Steps:

1. Read s_{p^t} and change the internal state σ according to state change rules below.

2. Write: $s_{p^t} = \Lambda$.

3. Move:

$$p^{t+1} = \begin{cases} p^t & \text{if } \sigma_C^{t+1} == \# \\ \sigma_D^{t+1} & \text{otherwise.} \end{cases}$$

4. Write: $s_{p^{t+1}} = \sigma_D^{t+1}$.

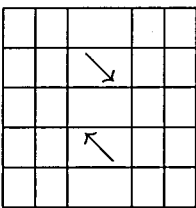
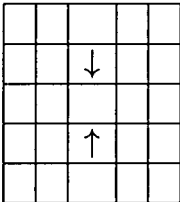
State Change Rules:

$$\sigma^{t+1} = \left\{ \begin{array}{ll} (s_p, \Lambda) & \text{if } s_{s_p} == \Lambda \text{ and for } n \in N(s_p), \\ & n \neq s_p, n \neq p^t, s_n + p_n \neq s_p \\ (s_p + x, \#) & \text{if } s_{s_p} == h \\ & \text{where } x = \begin{cases} (0, -2) & \text{if } s_p == N, NE, NW \\ (0, 2) & \text{if } s_p == S, SE, SW \end{cases} \\ (s_p + y, \#) & \text{if } s_{s_p} == v \\ & \text{where } y = \begin{cases} (-2, 0) & \text{if } s_p == E, NE, SE \\ (2, 0) & \text{if } s_p == W, NW, SW \end{cases} \\ (s_p + z, \#) & \text{otherwise} \\ & \text{where } z = \begin{cases} (1, 0) & \text{if } s_p == S, SW \\ (0, 1) & \text{if } s_p == E, SE \\ (-1, 0) & \text{if } s_p == N, NE \\ (0, -1) & \text{if } s_p == W, NW \end{cases} \end{array} \right.$$

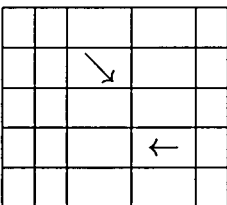
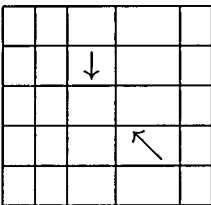
Collision Resolution Schemes Resulting from Model 1A

This is a list of initial collision configurations and their resolutions. The arrows indicate the machines' current positions and directions.

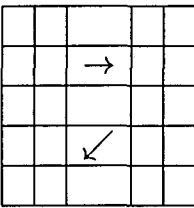
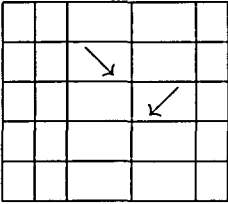
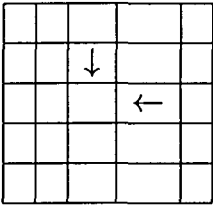
1. One machine moving south and one moving north, into the same square, resolves in one time step.



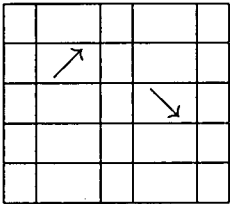
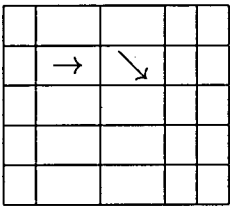
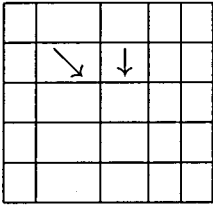
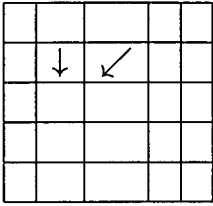
2. One machine moving south and one moving northwest also resolves in one time step.



3. One machine moving south and one moving west requires two times steps to resolve.



4. One machine moving south and one moving southwest requires three times steps to resolve.



Model 1B: A mobile automaton incorporating collision avoidance and primitive communication.

Notation:

p^t : position of machine at time t .

σ^t : internal state at time t , a tuple $(\sigma_D^t, \sigma_C^t, \sigma_E^t, \sigma_R^t)$, where

σ_D^t : internal directional state at time t ,

σ_C^t : internal collisional state at time t ,

σ_E^t : emitter state at time t ,

σ_R^t : receiver state at time t .

$D = \{(0,1), (0,-1), (1,0), (-1,0), (1,1), (1,-1), (-1,1), (-1,-1)\}$,
also written as $\{N, S, E, W, NE, SE, NW, SW\}$.

$C = \{\Lambda, \#\}$, where Λ is the default state and $\#$ is the collision alert state.

$E = \{on, off\}$, where the machine emits when *on*. (Receiver machines are permanently *off*.)

$R = \{paused, unpaused\}$, where the machine is normally *unpaused*, but goes into *paused* state when it detects an emit signal. (Emitter machines are permanently *unpaused*.)

$N(\alpha) = \{\alpha + a(1,0) + b(0,1) : a, b \in \{-1, 0, 1\}\}$, the neighborhood of array square α which includes α and the eight squares surrounding it.

$s_\alpha = s_\alpha(D, E)$ symbol set on square α . Note that s_α consists of an ordered pair and an additional symbol, where the ordered pair $s_\alpha(D)$ is in

$D \cup \{\Lambda, h, v\}$, where

$\Lambda := (0, 0)$,

$h := (0, 3) \equiv$ horizontal boundary (across the top or bottom),

$v := (3, 0) \equiv$ vertical boundary (down each side),

and the additional symbol $s_\alpha(E)$ is one of $\{\cdot, +\}$, where $+$ indicates the square is within the emit neighborhood of an active emitter machine.

Execution Steps:

1. Read $s_{p^t}(D, E)$ and change the internal state σ and internal timer value according to rules below.

2. Write: $s_{p^t} = \Lambda$.

3. Move:

$$p^{t+1} = \begin{cases} p^t & \text{if } \sigma_C^{t+1} == \# \text{ or } \sigma_R^{t+1} == \textit{paused} \\ \sigma_D^{t+1} & \text{otherwise} \end{cases}$$

4. Write: $s_{p^{t+1}} = \sigma_D^{t+1}$, and emit (see below) if $\sigma_E^{t+1} == \textit{on}$.

Emitter Rules:

Timer:

$$\varepsilon^{t+1} = \begin{cases} 2 & \text{if } \varepsilon^t == 0 \\ 1 & \text{if } \varepsilon^t == 2 \\ 0 & \text{if } \varepsilon^t == 1. \end{cases}$$

State Change Rules:

$$\sigma_{(D,C)}^{t+1} = \begin{cases} (s_p(D), \Lambda) & \text{if } s_{s_p(D)}(D) == \Lambda \text{ and for } n \in N(s_p(D)) \text{ such that} \\ & n \neq s_p(D), n \neq p^t, \text{ it is the case that} \\ & s_n(D) + p_n \neq s_p(D) \\ (s_p(D) + x, \#) & \text{if } s_{s_p(D)}(D) == h \\ & \text{where } x = \begin{cases} (0, -2) & \text{if } s_p(D) == N, NE, NW \\ (0, 2) & \text{if } s_p(D) == S, SE, SW \end{cases} \\ (s_p(D) + y, \#) & \text{if } s_{s_p(D)}(D) == v \\ & \text{where } y = \begin{cases} (-2, 0) & \text{if } s_p(D) == E, NE, SE \\ (2, 0) & \text{if } s_p(D) == W, NW, SW \end{cases} \\ (s_p + z, \#) & \text{otherwise} \\ & \text{where } z = \begin{cases} (1, 0) & \text{if } s_p(D) == S, SW \\ (0, 1) & \text{if } s_p(D) == E, SE \\ (-1, 0) & \text{if } s_p(D) == N, NE \\ (0, -1) & \text{if } s_p(D) == W, NW \end{cases} \end{cases}$$

$$\sigma_E^{t+1} = \begin{cases} \text{off} & \text{if } \varepsilon^{t+1} == 0 \\ \text{on} & \text{otherwise.} \end{cases}$$

Emitting:

If the emit machine's state σ_E^{t+1} is *on*, it will write a + to the squares in a 5×5 neighborhood centered at the emitter's position p^{t+1} .

Receiver Rules:

Timer:

$$\varepsilon^{t+1} = \begin{cases} 2 & \text{if } s_{p^t}(E) == + \\ 1 & \text{if } s_{p^t}(E) == \cdot \text{ and } \varepsilon^t == 2 \\ 0 & \text{otherwise.} \end{cases}$$

State Change Rules:

$$\sigma_R^{t+1} = \begin{cases} \text{unpaused} & \text{if } \varepsilon^t == 0 \\ \text{paused} & \text{otherwise.} \end{cases}$$

$$\sigma_{(D,C)}^{t+1} = \begin{cases} (s_p(D), \Lambda) & \text{if } \sigma_R^t == \text{paused,} \\ & \text{or if } s_{s_p(D)}(D) == \Lambda \text{ and} \\ & \text{for } n \in N(s_p(D)) \text{ such that } n \neq s_p(D), n \neq p^t, \\ & \text{it is the case that } s_n(D) + p_n \neq s_p(D) \\ (s_p(D) + x, \#) & \text{if } s_{s_p(D)}(D) == h \text{ and } \sigma_R^t == \text{unpaused} \\ & \text{where } x = \begin{cases} (0, -2) & \text{if } s_p(D) == N, NE, NW \\ (0, 2) & \text{if } s_p(D) == S, SE, SW \end{cases} \\ (s_p(D) + y, \#) & \text{if } s_{s_p(D)}(D) == v \text{ and } \sigma_R^t == \text{unpaused} \\ & \text{where } y = \begin{cases} (-2, 0) & \text{if } s_p(D) == E, NE, SE \\ (2, 0) & \text{if } s_p(D) == W, NW, SW \end{cases} \\ (s_p + z, \#) & \text{otherwise} \\ & \text{where } z = \begin{cases} (1, 0) & \text{if } s_p(D) == S, SW \\ (0, 1) & \text{if } s_p(D) == E, SE \\ (-1, 0) & \text{if } s_p(D) == N, NE \\ (0, -1) & \text{if } s_p(D) == W, NW \end{cases} \end{cases}$$

Model 2A: A mobile automaton on the plane tiled in hexagons, with collision avoidance but no other communication.

Notation:

p^t : position of machine at time t ; 6 bit code of 1's and 0's representing relative coordinates so that 000000 is the machine's current relative position at the beginning of each clock cycle.

σ^t : internal state at time t ; consists of:

σ_D^t : internal direction state at time t ,

σ_C^t : internal collision state at time t .

$D = 6$ bit code of 5 zeros and 1 one. The position of the one corresponds to the direction indicated, in the following order:

down-right, right, up-right, up-left, left, down-left.

$C = \{\Lambda, \#\}$, where Λ is the default state, and $\#$ is the collision alert state.

Let e_x be a six-bit code with 1 in the x^{th} position, 0 elsewhere.

$N(\alpha) = \{\alpha\} \cup \{\alpha + e_x : 1 \leq x \leq 6\}$ is the neighborhood of hex-tile α which includes α and the 6 hex-tiles surrounding it.

s_α : six-bit string on hex-tile α

unoccupied: 000000

occupied: direction code (one bit set to one)

wall: 101010

Note that addition of all bit strings satisfies the following relations:

1. 100100 \equiv 000000
2. 010010 \equiv 000000
3. 001001 \equiv 000000

Execution Steps:

1. Read s_{p^t} and change state σ according to rules below.
2. Write: $s_{p^t} = 000000$.
3. Move:

$$p^{t+1} = \begin{cases} p^t & \text{if } \sigma_C^{t+1} == \# \\ \sigma_D^{t+1} & \text{otherwise.} \end{cases}$$

4. Write: $s_{p^{t+1}} = \sigma_D^{t+1}$.

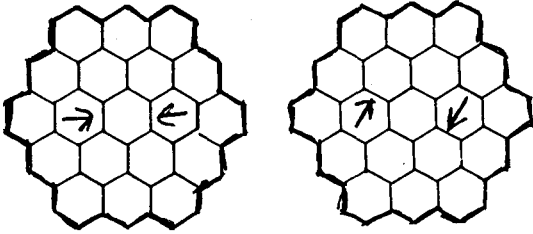
State Change Rules:

$$\sigma^{t+1} = \begin{cases} (s_p, \Lambda) & \text{if } s_{s_p} == \Lambda \text{ and for } n \in N(s_p), \\ & n \neq s_p, n \neq p^t, s_n + p_n \neq s_p \\ (e_{x+3(mod6)}, \#) & \text{if } s_{s_p} == 101010 \\ & \text{where } s_p = e_x. \\ (e_{x+1(mod6)}, \#) & \text{otherwise; notation as above.} \end{cases}$$

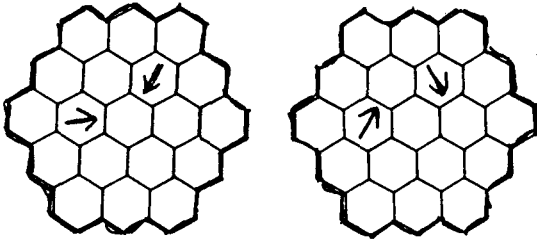
Collision Resolution Schemes Resulting from Model 2A

This is a list of initial collision configurations and their resolutions. The arrows indicate the machine's current positions and directions.

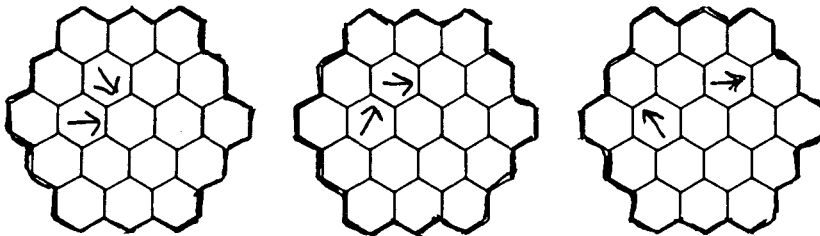
1. One machine moving to the right and one moving to the left, into the same hex-tile, resolves in one time step.



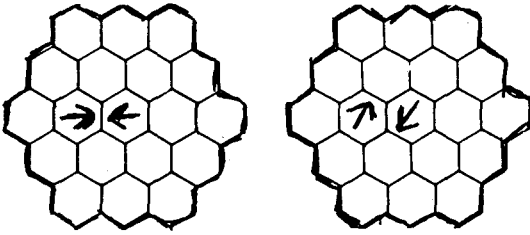
2. One machine moving to the right and one moving down-left, into the same hex-tile, resolves in one time step.



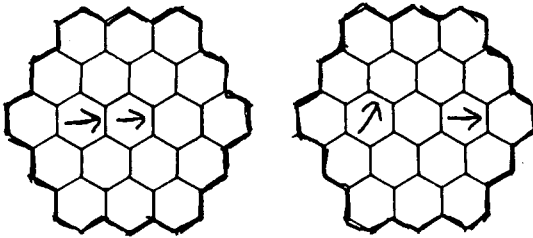
3. One machine moving to the right and one moving down-right, into the same hex-tile, resolves in two time steps.



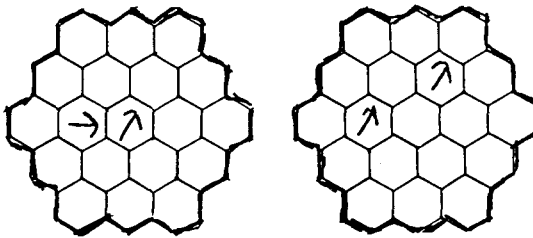
4. One machine moving to the right and one moving to the left, with no tiles in between them, resolves in one time step.



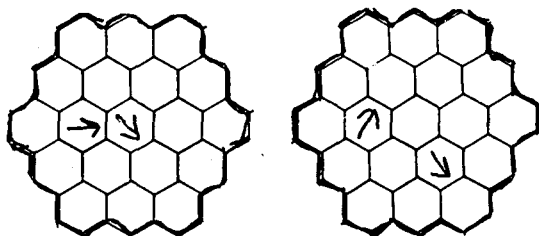
5. Two machines moving to the right, with one attempting to move into the hex-tile the other currently occupies, resolves in one time step.



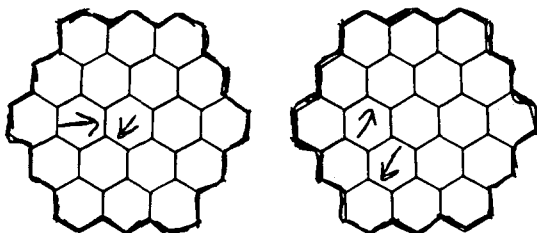
6. One machine moving to the right, attempting to move into the hex-tile currently occupied by a machine moving up-right, resolves in one time step.



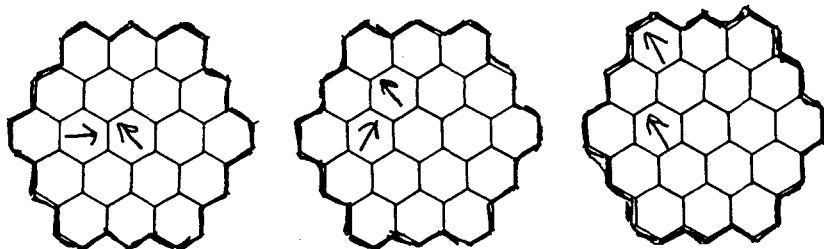
7. One machine moving to the right, attempting to move into the hex-tile currently occupied by a machine moving down-right, resolves in one time step.



8. One machine moving to the right, attempting to move into the hex-tile currently occupied by a machine moving down-left, resolves in one time step.



9. One machine moving to the right, attempting to move into the hex-tile currently occupied by a machine moving up-left, resolves in two time steps.



Model 2B: A mobile automaton on the plane tiled by hexagons, incorporating collision avoidance and primitive communication.

Notation:

p^t : position of machine at time t ; 6 bit string of 1's and 0's representing relative coordinates so that 000000 is the machine's current relative position at the beginning of each clock cycle

s^t : internal state at time t ; consists of:

σ_D^t : internal directional state at time t ,

σ_C^t : internal collisional state at time t ,

σ_E^t : emitter state at time t ,

σ_R^t : receiver state at time t .

$D = 6$ bit code with 5 zeros and 1 one. The position of the one corresponds to the direction indicated, in the following order:

down-right, right, up-right, up-left, left, down-left.

$C = \{\Lambda, \#, \Omega, \natural\}$, where Λ is the default state, $\#$ is the collision alert state, Ω is the following state, and \natural is the following-collision-alert state (last two used only by receiver machines).

$E = \{first, second, off\}$, emits on first and second. (Receiver machines permanently in *off* state.)

$R = \{1, 0\}$, 1 indicates paused, 0 indicates unpaused. (Emitter machines permanently unpaused.)

Let e_x denote a six-bit string with 1 in the x^{th} position, 0 elsewhere.

Let $e_{n,x}$ denote a six-place string with n in the x^{th} position, 0 elsewhere.

Let $e_{x,y}$ denote a six-bit string with 1 in the x^{th} and y^{th} positions, 0 elsewhere.

Let $e_{n,x,m,y}$ denote a six-place string with n in the x^{th} position, m in the y^{th} position, 0 elsewhere.

$N(\alpha) = \{\alpha\} \cup \{\alpha + e_x : 1 \leq x \leq 6\}$ is the neighborhood of hex-tile α which includes α and the 6 hex-tiles surrounding it.

$s(\alpha)$: symbol set written on hex-tile α : (s_D, s_E)

s_D : direction and pausing status of machine occupying α , 0000000 if unoccupied, 0101010 if wall.

s_E : emitter neighborhood code, or 000000 if no active emitter is within range.

Note that addition of bit strings satisfies the following relations:

1. $100100 \equiv 000000$
2. $010010 \equiv 000000$
3. $001001 \equiv 000000$

Execution Steps:

1. Read $s(p^t)$ and change state σ and timer value (if receiver machine) according to rules below.
2. Write: $s(p^t)_D = 0000000$.
3. Move:

$$p^{t+1} = \begin{cases} p^t & \text{if } \sigma_C^{t+1} == \# \text{ or } \downarrow \text{ or if } \sigma_R == \text{paused} \\ \sigma_D^{t+1} & \text{otherwise} \end{cases}$$

4. Write: $s(p^{t+1}) = \sigma_D^{t+1}$ adjoined to σ_R^{t+1} , and emit if $\sigma_E = on$.

State Change Rules:

For Emitter Machines

$$\sigma_E^{t+1} = \begin{cases} \textit{first} & \text{if } \sigma_E^t == \textit{off} \\ \textit{second} & \text{if } \sigma_E^t == \textit{first} \\ \textit{off} & \text{if } \sigma_E^t == \textit{second}. \end{cases}$$

$$(\sigma_D^{t+1}, \sigma_C^{t+1}) = \begin{cases} (s(p^t)_D, \Lambda) & \text{if } s_D(s_D(p^t)) == \Lambda \text{ and for } n \in N(s(p^t)_D) \text{ such that} \\ & n \neq s(p)_D, n \neq p^t, \text{ seventh bit of } s(n)_D \neq 1, \\ & \text{it is the case that } s(n)_D + p(n)_D \neq s(p^t)_D \\ (e_{x+3(\textit{mod}6)}, \#) & \text{if } s(s(p^t)_D)_D == 0101010 \\ & \text{where } s(p^t)_D = e_x. \\ (e_{x+1(\textit{mod}6)}, \#) & \text{otherwise; notation as above.} \end{cases}$$

Codes Written to Each Hex-tile in the Emit Neighborhood

Note that the emitter's direction of movement is indicated by e_x , and all addition is modulo six.

1. Write e_x to hex-tiles $e_{n \cdot x}$, $1 \leq n \leq 4$.
2. Write e_{x+n} to hex-tiles e_{x+n} , $e_{2 \cdot (x+n)}$, $1 \leq n \leq 5$.
3. Write $e_{x,(x+1)}$ (resp., $e_{x,(x-1)}$) to hex-tiles $e_{n \cdot x, m \cdot (x+1)}$ (resp., $e_{n \cdot x, m \cdot (x-1)}$), $\{n, m : 2 \leq n + m \leq 4\}$.
4. Write $e_{x+n, x+n+1}$ to hex-tiles $e_{x+n, x+n+1}$, $1 \leq n \leq 4$.

For Receiver Machines

Receiver machines have an internal timer, ε^t , which has an integer maximum value, ε_{max} .

$$\varepsilon^{t+1} = \begin{cases} \varepsilon_{max} & \text{if } s(p)_E \neq 000000 \\ \varepsilon^t - 1 & \text{if } \varepsilon^t > 0 \text{ and } s(p)_E = 000000 \\ 0 & \text{otherwise} \end{cases}$$

$$\sigma_R^{t+1} = \begin{cases} \text{paused} & \text{if } \varepsilon^t > 0 \text{ and } \sigma_C^t == \Lambda \text{ or } \# \\ \text{unpaused} & \text{otherwise.} \end{cases}$$

If $\sigma_C^t == \Lambda$ or $\#$, then:

$$(\sigma_D^{t+1}, \sigma_C^{t+1}) = \begin{cases} (e_y, \Omega) & \text{if direction of emitter has been calculated} \\ & \text{(see below), where } e_y \text{ is the direction of} \\ & \text{emitter movement.} \\ (s(p^t)_D, \Lambda) & \text{if } s_D(s_D(p^t)) == \Lambda \text{ and for } n \in N(s(p^t)_D) \\ & \text{such that } n \neq s(p)_D, n \neq p^t, \text{ seventh bit of } s(n)_D \neq \\ & \text{it is the case that } s(n)_D + p(n)_D \neq s(p^t)_D \\ (e_{x+3(mod6)}, \#) & \text{if } s(s(p^t)_D)_D == 0101010 \\ & \text{where } s(p^t)_D = e_x. \\ (e_{x+1(mod6)}, \#) & \text{otherwise; notation as above.} \end{cases}$$

If $\sigma_C^t == \Omega$ or \natural , then:

$$(\sigma_D^{t+1}, \sigma_C^{t+1}) = \begin{cases} (s(p^t)_D, \Lambda) & \text{if } \varepsilon^t == 0 \text{ and } s_D(s_D(p^t)) == \Lambda, \\ & \text{and for } n \in N(s(p^t)_D) \text{ such that} \\ & n \neq s(p)_D, n \neq p^t, \text{ seventh bit of } s(n)_D \neq 1, \\ & \text{it is the case that } s(n)_D + p(n)_D \neq s(p^t)_D \\ (s(p^t)_D, \Omega) & \text{if } \varepsilon^t > 0 \text{ and } s_D(s_D(p^t)) == \Lambda, \\ & \text{and for } n \in N(s(p^t)_D) \text{ such that} \\ & n \neq s(p)_D, n \neq p^t, \text{ seventh bit of } s(n)_D \neq 1, \\ & \text{it is the case that } s(n)_D + p(n)_D \neq s(p^t)_D \\ (e_{x+3(mod6)}, \natural) & \text{otherwise, where } s(p^t)_D = e_x. \end{cases}$$

Calculating the Emitter's Direction of Movement

1. If first string received has one entry set to one, that is, it is e_x for some x , then:
 - (a) If second string received = e_x : then emitter is moving in direction indicated by e_x , on a collision course with receiver.
 - (b) If second string received = $e_{x,y}$: then emitter is moving in direction indicated by e_{x+y-1} , alongside receiver.
 - (c) If second string received = 000000: then emitter is moving away or is *off*, wait for more information.
 - i. If third string received = e_x : as above.
 - ii. If third string received = $e_{x,y}$: as above.
 - iii. If third string received = 000000: then emitter is moving away; direction unclear.
2. If first string received has two entries set to one, that is, it is $e_{x,y}$ for some x, y , then:
 - (a) If second string received = $e_{x,y}$: then emitter is moving alongside receiver, direction indicated by either e_x or e_y ; wait for more information (see below).
 - (b) If second string received = e_x [or e_y]: then emitter is moving in direction indicated by e_y [or e_x], alongside receiver.
 - (c) If second string received = 000000: then emitter is moving away or is *off*, wait for more information.
 - i. If third string received = $e_{x,y}$: as above.
 - ii. If third string received = e_x [or e_y]: as above.
 - iii. If third string received = e_z , $z \neq x$, $z \neq y$: if $x - z = 2$, then the emitter is moving in the direction indicated by e_x . If $y - z = 2$, then the emitter is moving in the direction indicated by e_y , in both cases alongside the receiver.
 - iv. If third string received = $e_{x,z}$ [or $e_{y,z}$]: then emitter is moving in direction indicated by e_y [or e_x], alongside receiver.
 - v. If third string received = 000000: then emitter is moving away; direction unclear.

Figure Legends

Figure 1. Course of a stable “pretzel path” of one machine on a 4×5 space.

Figure 2. Resolution of an East-West and a North-South head-on collision between two machines on a 5×5 space, resulting in machines following along same path.

Figure 3. Steps preceding an East-West head-on collision on a 5×5 space. This generalizes to an $m \times n$ space as explained in the text.

Figure 4. The 5×5 centered and two 3×3 directional emitting neighborhoods.

Figure 5. Interpretation of the location string 210000.

Figure 6. Emit neighborhood with signal bitstrings for an emitter moving to the right.

Figure 7. The ten hex-tiles in the emit neighborhood which are used for following.

Figure 8. Collision with wall by a swarm of following receiver machines. Note that the lead machine becomes the trailing machine, and so an emitter *leading* receivers would become an emitter *following* receivers.

Figure 9. Neighborhood of tubulin dimers in a microtubule strand. After [12].

Figure 1.

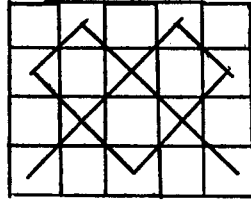


Figure 2.

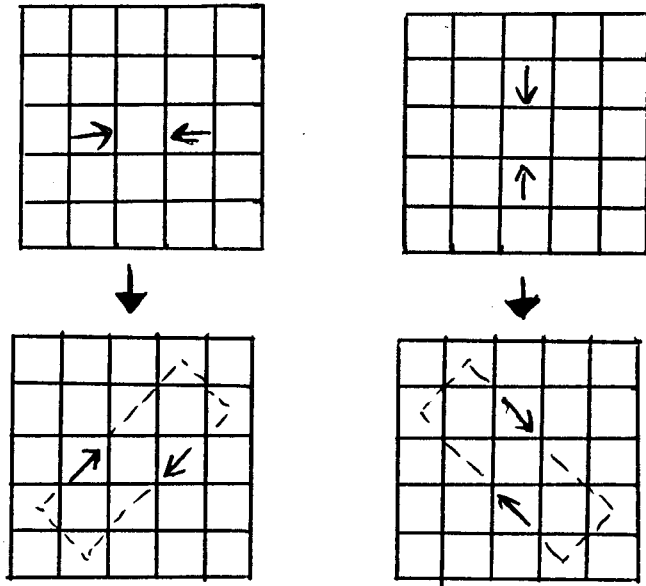


Figure 3.

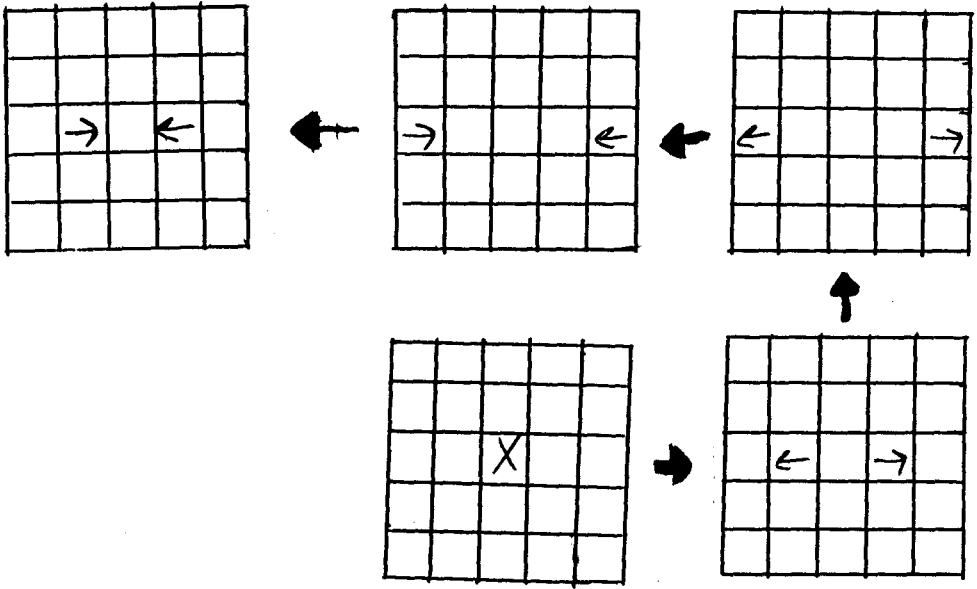


Figure 4.

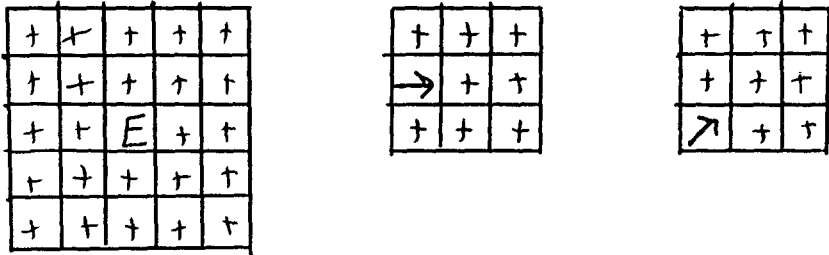


Figure 5.

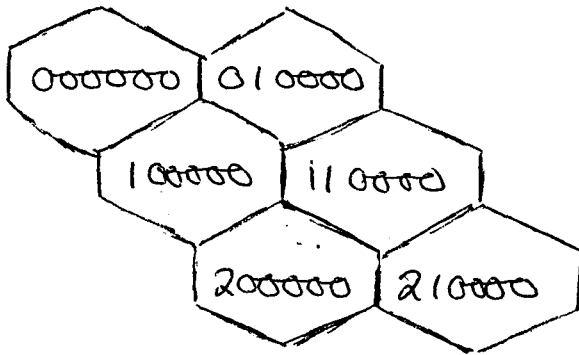


Figure 6.

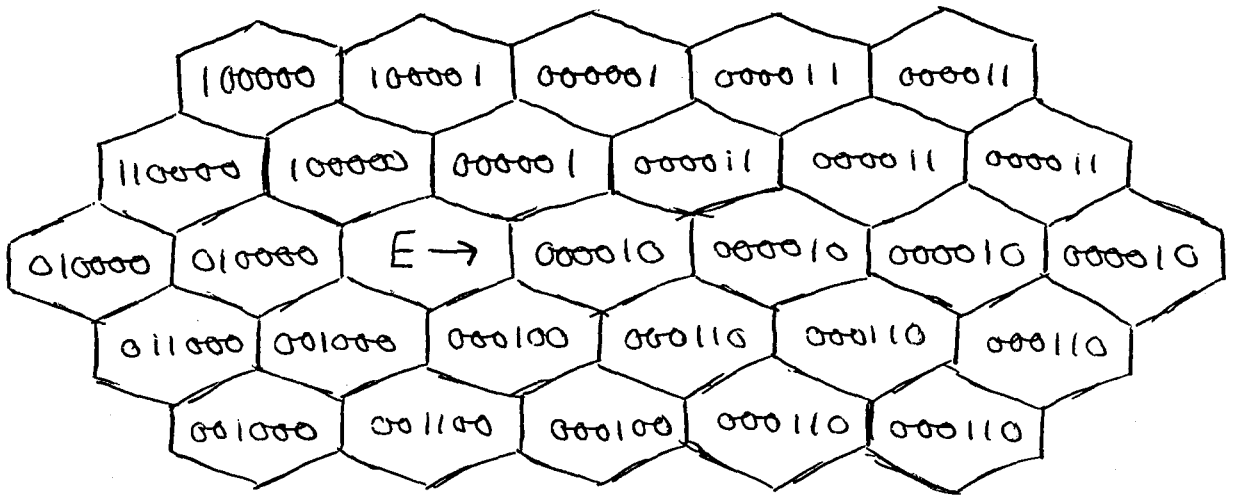


Figure 7.

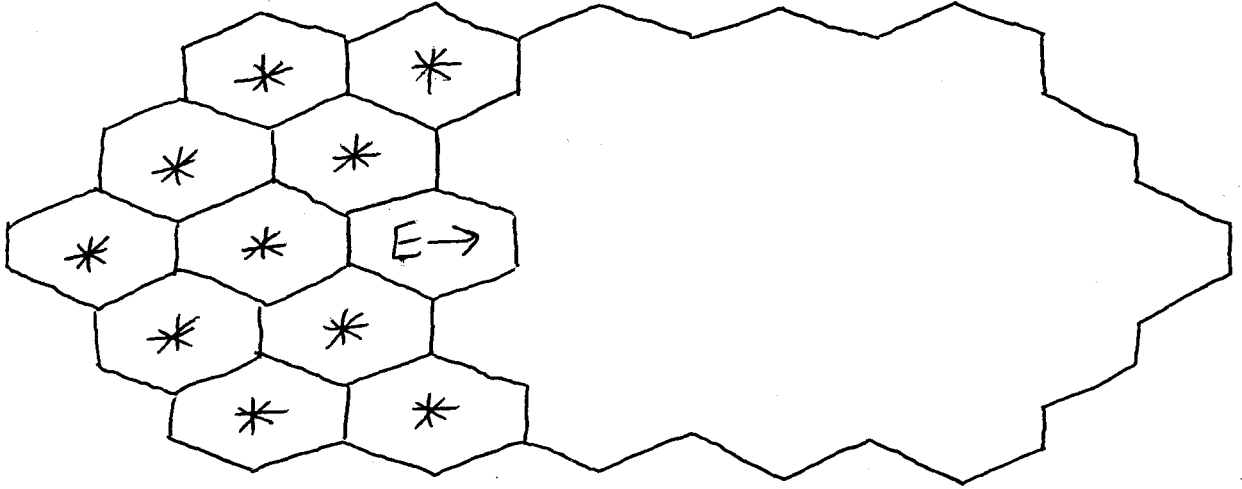


Figure 8.

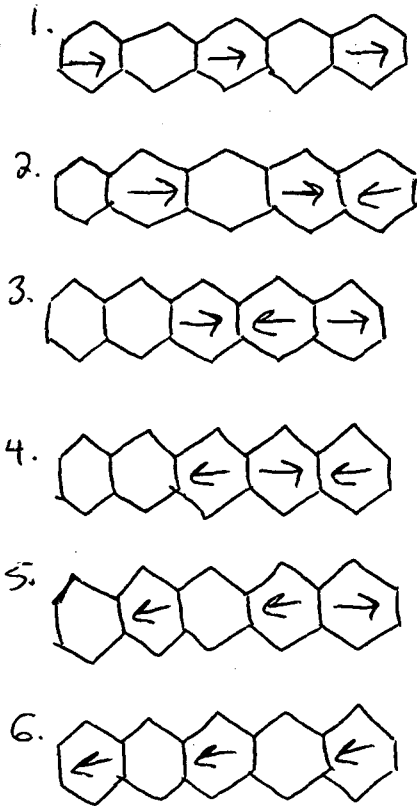


Figure 9.

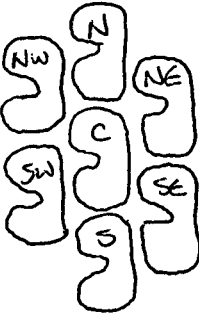


Table 1. Emitter direction, receiver point of entrance into emit neighborhood, and length of time receiver pauses, showing no correlation among the three.

Emitter is at $(0, 0)$, moving in direction shown in column **E**. Receiver enters at square with coordinates shown in the third column and with initial direction shown in column **R**. ϵ_{max} is the maximum value for the receiver's internal timer.

Column A: Emitter final direction.

Column B: Receiver final direction.

Column C: Number of time steps receiver is frozen.

Starting	directions	Rec. start. pos.	Entry	on	.	Entry	on	+
E	R		A	B	C	A	B	C
E	N	(2, -2)	E	N	$\epsilon_{max}+3$	E	N	$\epsilon_{max}+4$
E	N	(2, -1)	NE	N	$\epsilon_{max}+3$	E	N	$\epsilon_{max}+4$
E	N	(2, 1)	E	N	$\epsilon_{max}+3$	E	N	$\epsilon_{max}+4$
E	N	(2, 2)	E	N	0	E	N	$\epsilon_{max}+4$
E	N	(1, -2)	E	N	$\epsilon_{max}+2$	E	N	$\epsilon_{max}+3$
E	N	(0, -2)	E	N	$\epsilon_{max}+1$	E	N	$\epsilon_{max}+2$
E	N	(-1, -2)	E	N	ϵ_{max}	E	N	$\epsilon_{max}+1$
E	N	(-2, -2)	E	N	0	E	N	ϵ_{max}
E	N	(2, 0)	E	N	$\epsilon_{max}+3$	NE	N	$\epsilon_{max}+4$
E	W	(1, -2)	E	W	$\epsilon_{max}+3$	E	W	$\epsilon_{max}+3$
E	W	(1, -1)	E	W	$\epsilon_{max}+1$	E	W	$\epsilon_{max}+3$
E	W	(1, 0)	NE	SW	$\epsilon_{max}+1$	NE	W	$\epsilon_{max}+3$
E	W	(1, 1)	E	W	$\epsilon_{max}+2$	E	W	$\epsilon_{max}+3$
E	W	(1, 2)	E	W	$\epsilon_{max}+1$	E	W	$\epsilon_{max}+3$
E	W	(2, -2)	E	W	$\epsilon_{max}+1$	E	W	$\epsilon_{max}+4$
E	W	(2, -1)	E	W	$\epsilon_{max}+2$	E	W	$\epsilon_{max}+4$
E	W	(2, 0)	NE	SW	$\epsilon_{max}+2$	NE	W	$\epsilon_{max}+4$
E	W	(2, 1)	E	W	$\epsilon_{max}+2$	E	W	$\epsilon_{max}+4$
E	W	(2, 2)	E	W	$\epsilon_{max}+2$	E	W	$\epsilon_{max}+4$