Dan Morris
Partners : Neel Joshi, Soren Spies, Mike Fried
CS148
5-9-99

FINAL PROJECT REPORT :

2-ROBOT SUPER KILLER DEATH TAG

When the lights went down and the power started, it was every robot for himself. Killer[1] and Armageddon fought a fierce battle to the finish, leaving both of our heroes gasping for power… now, with their charred, broken remains returned to the Lego Lab, all that the world has to represent this colossal confrontation is a videotape[2]… and this document, which chronicles the days leading up to the grand final battle.

I. THE STORY :

In the year 2025, the last surviving human[3] was eradicated from the planet Earth, leaving a world dominated entirely by very small Lego robots with large pieces of wood on their heads. With the common enemy eliminated, a contest for power developed within the LegoBot community; two groups found themselves competing for precious resources. *The Ronald McDonalds*[4] represented all that was good and happy in the Lego world… *The Hamburglers*[4] represented the dark side of robotics : thievery, deceit, plunder, etc. Because neither side possessed any armaments (only fluorescent light bulbs and an account at Radio Shack), it was decided that the two factions would agree on a set of rules and send their best warriors into a competition to determine who would rule Earth. Killer represented *The Ronald McDonalds*, and Armageddon stood brave for *The Hamburglers*. The rules of the game were simple… the technical report that follows will contain all these rules and more.[5]

II. THE RULES :

At the time this project was presented, the rules of *2-RSKDT* were as follows :

- One robot was designated "it" – the "hunter" – at power-up time; the other robot was designated "not it" – the "prey"
- The hunter aimed to locate and collide with the prey; the prey aimed to avoid the hunter
- A round ended when either of two conditions was met :
  - A collision occurred, *as determined by the prey*
    - Following a collision, the robots moved away from each other during an "inter-round" phase, to avoid rapid "tagbacks" and make the game look more interesting
  - A pre-designated number of clock cycles passed, *as determined by the hunter* (considered a "score" for the prey)
    - Following a "timeout", both robots paused for a few moments, so an observed could immediately tell that a timeout had occurred
- At the beginning of the new round "Hunter" and "prey" roles were reversed.

[1] It was stated in a previously submitted document that "Killer" was actually short for "Jeremiah Kill-Kill-Killer Johnson". After further review, Killer's full name has been truncated to "Jeremiah Kill-Killer Johnson", because he didn't really do all that much killing.
[2] Also now available for viewing on the comfort of your own desktop… you can find this rare but captivating footage in an inconveniently large file at "http://techhouse.brown.edu/vivek/148.mov"
[3] The "last surviving human", believe it or not, was Strom Thurmond
[4] Please note that the terms "Ronald" and "Hamburgler" are not related to the pop icons "Ronald McDonald" and "The Hamburgler", as the use of such trademarks would be in blatant violation of copyright laws.
[5] A careful reader will note that this story is actually never concluded.

III. This isn't quite what was proposed… and it didn't work quite right on demo day…

I will now attempt to summarize the state of the project at demonstration time, being honest but not self-deprecating, and list the items in the original proposal that were never implemented. Specific hardware issues will be discussed later.

At demonstration time, these were the <u>key functional features in our project</u>

- The hunter was in fact able to pursue the prey, and the prey was able to deliberately flee.
- The robots could signal time-outs and collisions to each other, and react according to the above rules.
- The robots were able to move away from each other between rounds, and switch roles after a round, maintaining appropriate internal representations of the state of the game.

These aspects of the project were functional, and represent the most fundamental aspects of the above rules. Hence we were able to run games in which one robot chased and tagged the other, followed by a reversal of roles and a second successful tag.

With that said, we still had to resort to a video tape at demonstration time, and we never observed long and exciting games. Following are the <u>primary shortcomings in our project</u> that prevented an ideal and reliable game of tag :

- The robots had quite a bit of trouble finding each other at long ranges, due to problems with lighting that will be discussed below. Consequently, once the robots were separated by a large distance, the game tended not to recover.
- The signals that were used to communicate changes in the current state of the game (discussed in detail below) were not perfectly reliable, either due to interference or to hardware flakiness, and a lost signal basically meant the end of the current game. However, missed signals were rare, and the radio hardware *really did work* in general.
- Collisions were hard on the robots, and bump sensors began to beak or lose sensitivity over the course of many trial games, leading to collisions that did not register properly.
- The physical changes to our robots that were required for all of the project's hardware added a significant amount of weight, which slowed the robots down and slowly wore down the gears and axles. After running for twenty-four hours straight before our demonstration, our robots were not in top mechanical shape.

Furthermore, the original proposal differed significantly in one aspect of the rules: originally, the "prey" robot was trying to reach a "safe zone" while avoiding the "hunter". The safe zone was also to be marked by a light, and the robots would differentiate between "the enemy" and the safe zone based on some quality of the light, either color or direction of polarization. The reasons this item was omitted will be outlined later.

Despite all of the project's shortcomings (welcome to robotics), we were happy that we had achieved baseline functionality, and reduced the problem to what we considered hardware tweaking (which we worked at for many painful hours). We considered this an ambitious project, and learned above all else that two robots present engineers with n^2 problems, where n is the **huge** number of problems generally encountered in working with any robot. The normal frustrations of dealing with flaky hardware were magnified tremendously.

But the real academic focus of this project was on communication and coordination between multiple robots. It is thus from this perspective that I will approach my discussion of all the hardware we used and/or tried for this project.

IV. AT LEAST OUR ROBOTS WERE NEVER LONELY... ALWAYS SOMEONE AROUND TO TALK TO...

The discussion of hardware will be broken into the three primary methods by which the robots interacted: light sensation, mechanical contact, and radio transmission.

IVA. Light sensation :

In order to follow or avoid "the enemy", each robot had to have some way of localizing its opponent. The most accessible method of localization was a combination of light sources and light sensors. This required that we equip each robot with a light source bright enough to guide the opponent robot. Initially, we also hoped to have a local DC supply driving the lights, which would obviate the need to carry AC power cords around behind the robots. The need to support large light bulbs and high-current DC batteries was a major driving force behind the major structural modification we made to each robot : the addition of a large plywood octagon[6], mounted securely on top of each robot (bracketed into the Lego infrastructure).

Unfortunately, powering bright lights on DC batteries proved to be more of a challenge than we expected. Incandescent lights draw a tremendous amount of current, and have a nasty tendency to melt any hardware that is mounted near them (e.g. Polaroid). Hence we opted to use fluorescent lights, which are bright, cool, and energy-efficient. Unfortunately, they also require a huge current burst at startup-time. We experimented with circuits to deliver such a current burst (which are normally mounted within fluorescent light fixtures), but were unsuccessful. Hence, in the end, we opted for AC-powered fluorescent bulbs, and settled for carrying AC power cords around behind the robots.

As mentioned earlier, we were also originally planning on using light polarizers to allow the robots to differentiate between "opponent" and "safe zone." Hence we purchased a large quantity of Polaroid, and mounted pairs of light sensors that were covered with perpendicularly-oriented Polaroid. And, initially, we had some successful runs in which the robot was able to selectively attract itself to light of one polarization state or the other, depending on which set of light sensors were active.

Unfortunately, using polarized light proved to be impossible in the context of *2-RSKDT*. As light passed through the polarizer that covered the light source, then through the polarizer that covered the light sensor, it was dimmed so much as to reduce the differential between same-polarization light sensors (left and right, for example_ to a range smaller than the inherent noise in the sensors. Furthermore, in order to achiever a differential between alternate polarization states, polarizers had to be precisely perpendicular – a condition that virtually never occurs when Lego robots are in motion. We briefly experimented with using colored light filters instead of polarizers, but had even less success – quality monochrome filters are extremely expensive and severely dim the passing light.

Hence we opted to use a single fluorescent bulb on each robot, with unpolarized light sensors guiding a robot toward or away from its opponent. This still left several challenges. Initially, our robots were equipped with only two front-mounted light sensors, and we intended to use an algorithm similar to the "Love" code employed earlier in the semester (ipsilateral inhibitory sensor-motor connections). However, this was not useful for helping a robot evade an opponent that was chasing it from behind; light sensors pointed away from the light source do not provide useful information. Hence we added rear-facing light sensors to each robot, and devised a system for arbitrating between sensor pairs. If a robot was trying to seek a light source (that is, if the robot was "it"), it would spin in place until its front sensors were receiving more light than its back sensors, then would run a modified "Love" program. The "not it" robot would take the same approach to orienting its back sensors toward the light source, and would then drive its motors based on a contralateral-excitatory pattern of sensor-motor connections (which allowed it to run away).

[6] Though one might describe these platforms as "large plywood octagons", we prefer the following term :

"bigass-pieces-o-plywood"

This approach was generally effective; difficulties arose with LARGE inherent differences between front and rear sensor pairs, requiring that we multiply various sensors by adjustment constants to allow comparison of front and back sensor totals. This approach also yielded a unique behavior, apparent in several places on our video: when a robot was "confused" – that is, when it had more or less lost its light source – it tended to spin in place until the opponent approached it. This is actually a desirable behavior in the context of the game; if you can't figure out where your opponent is, it's probably best to scan all directions until you find him.

The final issue we faced in using light sensation was a consequence of the brightness of our fluorescent bulbs : robots had a strong tendency to find and react to their own reflections off of couches, humans, the floor, etc. This would have been partially resolved had we been able to construct an all-black environment, but it was not possible for us to cover everything in the Techhouse lounge or in Lubrano with black sheets. Hence we opted to flatten the lights a bit by using our defunct and abandoned cylinders of Polaroid. The video does reveal one or two instances in which a robot clearly followed its own reflection off of a couch, but the addition of a Polaroid screen over both lights greatly reduced this problem. Of course, this solution came at the expense of some brightness and hence some range (range at which the robots could effectively react to each other's lights).

IVA. Mechanical Contact :

The ability for one robot to detect and signal a "tag" is central to the rules of 2-RSKDT. As specified in the rules above, the game's software left tag detection to the "prey" robot – that is, the goal of the "hunter" was to contact the "prey" with enough force to cause tag recognition.

In its final state, our "tag-sensation" system consisted of a metal ring (cut from thin sheet metal), approximately ½" tall, encircling the plywood platform on each robot. The ring rested on curled ends of eight vertically mounted bump sensors (one at each vertex of the octagonal platform), and was glued at its contact points. The bump sensors were wired in series[7] onto a breadboard (also mounted on the robot's wooden platform), and were connected to the processor as a single bump sensor. Thus when any of the sensors was tripped, the robot registered a tag (not directionally specific) and reacted accordingly (sent a signal to the other robot and began to prepare for the next round).

This system was effective for the head-on collisions that we generally observed (after the "hunter" backed the "prey" into a corner), but may not have been sensitive enough for less forceful "on-the-run" tags. The metal ring was relatively rigid and required a reasonable amount of force before it could be deformed sufficiently to trip one of the bump sensors. For this reason, it was generally difficult to predict which sensor would actually be switched first when a collision took place. This prompted our non-specific treatment of the bump sensors (that is, it would have been virtually impossible to tell from which direction the "attacking" robot had actually initiated the collision).

The structure we chose for tag sensation was something of a compromise; we wanted a system that would be sensitive enough to detect collision with an opponent, but not so sensitive that it tripped in the normal (and somewhat unstable) course of the robot's motion. For example, our initial system involved plastic knives, glued to each bump sensor. However, this structure was too flexible, and collisions often went unnoticed. At the other extreme, we also tried a heavier system of metal, which weighed down the bump sensor arms enough to signal false collisions.

Another significant limitation on our system of tag detection was the limited number of functional bump sensors that we had access to. One who has not taken CS148 might assume that switches either work or don't work, but after sorting through thirty or so to find our favorite sixteen, we found a wide array of approaches that bump sensors can take to malfunctioning.

A final point about bump sensation and mechanical structure... one of our goals in using the plywood platforms was to allow flexibility in our structure without making significant changes to the Lego architecture of the robots. As an example, we were able to leave in place the front-mounted bump sensors that we had used earlier in the course for

[7] Mounting the bump sensors in series rather than in parallel, though counterintuitive, was necessary because the bump sensors used actually *open* the circuit when contacted.

object avoidance. And, in a somewhat legitimate stroke of "hacking the environment", we mandated that all walls which the robots would encounter be high enough to trip the front-mounted bump sensors, but not high enough to trip the top-mounted sensors that were used for tag detection. This is how the robots distinguished between colliding with a wall and colliding with "the enemy". Collisions detected by the front-mounted sensors *always* caused the robot to back up and turn away from the offending object, regardless of the current behavior or game state (It's a subsumption architecture! Get it?).

IVC. Radio Communication :

Up to this point, I have simply asserted that robots signaled game events to each other. You, the curious reader, may ask what wizardry we employed to allow Lego robots to communicate as such. As it happens, we did in fact cast powerful magic spells on our robots that gave them life, allowing them to simply call each other on cellular phones.

Unfortunately, our robots soon began using their cellular phones to make prank calls and make long-distance calls, so we opted for a more traditional system. In particular, we equipped each robot with a transmitter and receiver from a Home Depot remote doorbell kit, with each transmitter coded to match the receiver on the opposing robot.

Sending signals was accomplished by activating the "button" component of the doorbell system. A 5V-switched mechanical relay was wired into the circuitry of the button, such that a signal voltage would connect the button's internal battery and initiate transmission of a radio signal. This 5V signal was provided by an LED output port on the robot's board (motor ports were not appropriate, because the pulsed nature of the voltage supplied by the motor ports made the relay unhappy). Sending a single signal required that we activate the appropriate port for approximately 40 clock cycles, to allow time for the relay to switch and more or less guarantee transmission.

The receiver component of each robot's doorbell system had its ground wired to the board's ground, and its power wired halfway into the board's battery system (because it demanded 3V, while the board uses 6V). Because the circuits had common grounds, an incoming signal could be detected by monitoring the voltage on a single signal wire, which connected the receiver to one of the board's analog input ports. A signal reliably changed the value at this port from 4 to approximately 130.

So, during execution, the "prey" robot sent a signal any time a collision was detected, and the "hunter" robot sent a signal when a round timed out. Incredibly enough, this was the only portion of our project on which I don't have any failures to report. Except of course for the cellular phone idea...

V. At least the software was as sketchy as the hardware...

Rex code has been attached as APPENDIX A. Essentially, the software existed to manage a finite state machine that allowed each robot to be in the appropriate state relative to the other robot at all times. Of course, missed signals disrupted the current state and caused everything to go ill, but the inputs were calibrated such that this only occurred when we were demonstrating the project in front of the entire department.

To summarize the 9 pages of 8-point type that direct the finite state machine, a single master state variable "dansmom" represented the "it" or "not it" state, and switched at the conditions mandated by the rules of the game. State variables were also used to represent the states between rounds. The use of single boolean variables for all possible behaviors simplified the actual calculation of motor values to a large conditional statement.

It was also necessary to use register variables for all input events; radio signals and collisions were of unpredictable duration, requiring that we use only *changes* in input state to direct state changes.

Both robots ran more or less identical code, although all constants were adjusted for the sensors and motors of each robot. Note that two Rex "makem" statements appear in our code. This allows separate constants for each of two output files, one of which was intended for each robot. This also explicitly specifies that one robot will always be "it" at startup.

VI. Group Dynamics...

Because you asked, work on this project (beyond the whole-group tasks of construction, frustration, and staying up really late) was divided approximately as follows :

- Soren “Da Radio Pimp” Spies was solely responsible for the hardware implementation of radio communication, and thus deserves credit for taking charge of the single most functional portion of our project (which almost anyone would have suggested was the least likely to work).

- Mike “The Camera Man” Fried was responsible for some of the structural modifications made to the robots (and for the brilliant camera work displayed on our presentation video).

- Neel and I worked almost entirely in cooperation on writing all of the code required for the project (a large whiteboard was consumed by our large FSM diagram), and also talked about women a lot[8].

VII. Lilliputians...

No Lilliputians or other especially small individuals were harmed in the construction of our robots.

VIII. Special Thanks...

Special thanks to all of the following for their unique contributions :

- Professors Rob Netzer and William Patterson, for hardware advice

- The good folks at Stop and Shop, for manufacturing a Triscuit-like product cheap enough for us to live on for a 40-hour period.

- Neel’s mom... yeah...

- Metallica, Guns and Roses, Ozzy Osbourne, and Megadeth, for their contributions to our presentation’s soundtrack. Metal is always a crowd-pleaser.

[8] The aforementioned discussions, though entirely unrelated to CS148, culminated in the following document, the “daughter” project of 2-RSKDT: “http://techhouse.brown.edu/dmorris/th/discourse”

APPENDIX A: REX CODE FOR 2-RSKDT

```
(in-package :cl-user)
(use-package :rex)

(load "/course/cs148/lib/6.270-rex-defs.lisp")

(setf *c-dir* "/u/nsj/course/cs148/FinalProj/")
;(setq *M_MAX* !15)
;(setq *M_MIN* !8)
;(setq *TURN_TIME* !30)
;(setq INITIALDANSMOMVAL init-dan)

;;Set default values
(defun lego-defaults (outp)
  (== (lego-out-motor0 outp) !0)
;  (== (lego-out-motor1 outp) !0)
  (== (lego-out-motor2 outp) !0))
;  (== (lego-out-motor3 outp) !0))
  ;(== (lego-out-led0 outp) !'0b)
  ;(== (lego-out-led1 outp) !'0b)
  ;(== (lego-out-lcd-int0 outp) !122)
  ;(== (lego-out-lcd-int1 outp) !122)
  ;(== (lego-out-lcd-int2 outp) !188)
  ;(== (lego-out-lcd-int3 outp) !119))


;;Define a struct for a bundle of inpus and outputs
(define-rex-struct wire-bundle
  ((left-motor int) (right-motor int)
   (l-light-fv int) (r-light-fv int)
   (l-light-fh int)(r-light-fh int)
   (l-light-b int) (r-light-b int)
   (dip0 bool) (dip1 bool) (dip2 bool) (dip3 bool)
   (low-light int) (med-light int) (high-light int)
   (bumpR bool) (bumpL bool) (ring bool) (clock int) (clock-count-l int)
   (clock-count-r int) (it bool) (frob int)
   (lfgatring bool)(lbgatring bool)(lastring bool)
   (fgtblast bool)(turningleft bool)(neelsrobot bool)
   (M_MAX int)(M_MIN int)

   (dansmom bool)(runaway bool)(pause bool)


))

;;Fixed devide function
(defun rdivm (n d)
  (ifm (equalm (signm n) (signm d))
       (divm (absm n) (absm d))
       (unary-minusm (divm (absm n) (absm d)))))

;;Clip motor values, so that motors aren't driven at illegal values
(defun clip-motor-val (motor-val)
  (ifm (<m motor-val !5)
       !0
       (ifm (>m motor-val !15)
            !15
            motor-val)))

(defun avoid-pair (motor-pair clock clock-count-l clock-count-r frob)
  (ifm (andm (notm (equalm clock-count-l !0))
             (<m (minusm clock clock-count-l)
                 (plusm (timesm !2 (divm frob !3))
                        (modm clock-count-l !10))))
       (list !-13 !-13)

       (ifm (andm (notm (equalm clock-count-r !0))
                  (<m (minusm clock clock-count-r)
                      (plusm (timesm !2 (divm frob !3))
                          (modm clock-count-r !10))))
            (list !-13 !-13)
            (ifm (andm (notm (equalm clock-count-l !0))
                       (<m (minusm clock clock-count-l)
                           (plusm frob (modm clock-count-l !10))))

                 (list !13 !-13)
                 (ifm (andm (notm (equalm clock-count-r !0))
                            (<m (minusm clock clock-count-r)
                                (plusm frob
                                       (modm clock-count-r !10))))
                      (list !-13 !13)
                      motor-pair)))))

(defun scale-motor-maze-orbit (X_L X_R bundle)
  (list (minusm (first (avoid-pair (scale-motor-pair
                                   (wire-bundle-low-light bundle)
                                   (wire-bundle-med-light bundle)
                                   (wire-bundle-high-light bundle)
                                   X_L
                                   X_R
                                   (wire-bundle-M_MIN bundle)
                                   (wire-bundle-M_MAX bundle))
                                  (wire-bundle-clock bundle)
                                  (wire-bundle-clock-count-l bundle)
                                  (wire-bundle-clock-count-r bundle)
                                  (wire-bundle-frob bundle))) !1)
        (minusm (second (avoid-pair (scale-motor-pair
                                    (wire-bundle-low-light bundle)
                                    (wire-bundle-med-light bundle)
                                    (wire-bundle-high-light bundle)
                                    X_L
                                    X_R
                                    (wire-bundle-M_MIN bundle)
                                    (wire-bundle-M_MAX bundle))
                                   (wire-bundle-clock bundle)
                                   (wire-bundle-clock-count-l bundle)
                                   (wire-bundle-clock-count-r bundle)
                                   (wire-bundle-frob bundle))) !1)))


(defun scale-motor-maze-love (X_L X_R bundle)
  (list (plusm !1  (first (avoid-pair (scale-motor-pair-love
                                      (wire-bundle-low-light bundle)
                                      (wire-bundle-med-light bundle)
                                      (wire-bundle-high-light bundle)
                                      X_L
                                      X_R
                                      (wire-bundle-M_MIN bundle)
                                      (wire-bundle-M_MAX bundle))
                                     (wire-bundle-clock bundle)
                                     (wire-bundle-clock-count-l bundle)
                                     (wire-bundle-clock-count-r bundle)
                                     (wire-bundle-frob bundle))))
        (plusm !1  (second (avoid-pair (scale-motor-pair-love
                                       (wire-bundle-low-light bundle)
                                       (wire-bundle-med-light bundle)
                                       (wire-bundle-high-light bundle)
                                       X_L
                                       X_R
                                       (wire-bundle-M_MIN bundle)
                                       (wire-bundle-M_MAX bundle))
                                      (wire-bundle-clock bundle)
                                      (wire-bundle-clock-count-l bundle)
                                      (wire-bundle-clock-count-r bundle)
                                      (wire-bundle-frob bundle))))))

(defun neg (motor-pair)
  (list (minusm !0 (first motor-pair))
        (minusm !0 (second motor-pair))))
```

```
(defun scale-motor-maze-orbit-neg
  (L M H X_L X_R clock clock-count-l clock-count-r frob)
  (neg (scale-motor-maze-orbit
       L M H X_L X_R clock clock-count-l clock-count-r frob)))

;;Scale both motors toghther and return toghther
(defun scale-motor-pair (L M H X_L X_R M_MIN M_MAX)
  (ifm (andm (<m X_L L) (<m X_R L))
       (list !0 !0) ;if both motors are less that L, stop
       (list (scale-motor L M H X_L M_MIN M_MAX)
             (scale-motor L M H X_R M_MIN M_MAX))))

;;Combine inhibitory and excitatory functions
(defun scale-motor (L M H X M_MIN M_MAX)
  (clip-motor-val (plusm
                   (scale-motor-in L M H X M_MIN M_MAX)
                   (scale-motor-ex L M H X M_MIN M_MAX))))

;;Defines the function to calculate the inhibitory line
(defun scale-motor-in (L M H X M_MIN M_MAX)
  (ifm (>=m X M)
       !0
       ;y = b + x/s
       (plusm (b-in L M M_MIN M_MAX)
              (rdivm X (inv-slope-in L M M_MIN M_MAX)))))

;;Defines the function to calculate the excitatory line
(defun scale-motor-ex (L M H X M_MIN M_MAX)
  (ifm (<m X M)
       !0
       ;y = b + x/s
       (plusm (b-ex L M M_MIN M_MAX)
              (rdivm X (inv-slope-ex L M M_MIN M_MAX)))))

;;Clip motor values, so that motors aren't driven at illegal values
(defun clip-motor-val (motor-val)
  (ifm (<m motor-val !5)
       !0
       (ifm (>m motor-val !15)
            !15
            motor-val)))

;;Finds the 1/slope of the inhibitory line
(defun inv-slope-in (L M M_MIN M_MAX)
  ;(M-L)/(M_MIN-M_MAX)=s
  (rdivm (minusm M L) (minusm M_MIN M_MAX)))

;;Returns the 1/slope of the excitatory line as the negative of
;;the inhibitory
(defun inv-slope-ex (L M M_MIN M_MAX)
  ;-s
  (minusm !0 (inv-slope-in L M M_MIN M_MAX)))

;;Finds the vertical shift, b, for y = mx +b for the inhibitory line
(defun b-in (L M M_MIN M_MAX)
  ;M-MIN - (M/s)
  (minusm M_MIN (rdivm M (inv-slope-in L M M_MIN M_MAX))))

;;Finds the vertical shift, b, for y = mx +b for the excitatory line
(defun b-ex (L M M_MIN M_MAX)
  ;M-MAX - (M/-s)
  (minusm M_MIN (rdivm M (inv-slope-ex L M M_MIN M_MAX))))

;;Love motor function

(defun scale-motor-love (L M H X M_MIN M_MAX)
  (clip-motor-val (ifm (<m X L)
                       !15
                       (ifm (>=m X H)
                            !0
                            (plusm M_MAX
                                 (rdivm
                                  (timesm (minusm M_MAX M_MIN)
                                          (minusm L X))
                                  (minusm H L)))))))

;;Scale both motors toghther and return toghther
(defun scale-motor-pair-love (L M H X_L X_R M_MIN M_MAX)
  (ifm (andm (<m X_L L) (<m X_R L))
       (list !0 !0)
       (list (scale-motor-love L M H X_L M_MIN M_MAX)
             (scale-motor-love L M H X_R M_MIN M_MAX))))


;;Define wires and allow for dip switch calibration
(defun setmotorsm (bundle)

  (some* ((pause bool)(dansmom bool)(runaway bool))

        (== pause (wire-bundle-pause bundle))
        (== dansmom (wire-bundle-dansmom bundle))
        (== runaway (wire-bundle-runaway bundle))

        (== (wire-bundle-low-light bundle) !5)
        ;;(init-next 30 (ifm (wire-bundle-dip0 bundle)
        ;;             (wire-bundle-left-light-front bundle)
        ;;      (wire-bundle-low-light bundle))))

        (== (wire-bundle-med-light bundle) !50)
        ;;(init-next 150 (ifm (wire-bundle-dip1 bundle)
        ;;              (wire-bundle-left-light-front bundle)
        ;;       (wire-bundle-med-light bundle))))

        (== (wire-bundle-high-light bundle) !150)
        ;;(init-next 210 (ifm (wire-bundle-dip2 bundle)
        ;;              (wire-bundle-left-light-front bundle)
        ;;       (wire-bundle-high-light bundle))

        (== [(wire-bundle-left-motor bundle)(wire-bundle-right-motor bundle)]

            ; if I'm paused
            (ifm pause
                 (list !0 !0)
                 ; if I'm not running away and I'm not paused and I'm it,
                 ;go after the other robot
                 (ifm (andm dansmom (notm runaway))

                      ; if front is greater than back
                      (ifm (>m (plusm (wire-bundle-l-light-fv bundle)
                                      (wire-bundle-r-light-fv bundle))
                              (plusm (wire-bundle-l-light-b bundle)
                                      (wire-bundle-r-light-b bundle)))
                                      ; if we're real close to the light
                           (ifm (andm (>m (wire-bundle-l-light-fv bundle) !75)
                                      (>m (wire-bundle-r-light-fv bundle) !75))
                                      ; run at the light
                               ;(list !15 !15)
                               (list (plusm !4 (first (scale-motor-maze-love
                                                       (wire-bundle-l-light-fv bundle)
                                                     (wire-bundle-r-light-fv bundle)
                                                     bundle)))
                                     (plusm !4 (second (scale-motor-maze-love
                                                        (wire-bundle-l-light-fv bundle)
                                                        (wire-bundle-r-light-fv bundle)
                                                        bundle))))

                                      ; if front is greater but we're far away, play love
                               (scale-motor-maze-love
                                (wire-bundle-l-light-fv bundle)
                                (wire-bundle-r-light-fv bundle)
                                bundle))
```

```
                                       ; if front is not greater, spin around
                                       ; We need to change this so it turns in an intelligent dir
                            (list !-13 !13))

                        ; otherwise I must either be not it or running away, so I should run away

                                       ; if front is greater than back
                        (ifm (>m (plusm (wire-bundle-l-light-fv bundle)
                                       (wire-bundle-r-light-fv bundle))
                                (plusm (wire-bundle-l-light-b bundle)
                                       (wire-bundle-r-light-b bundle)))
                                       ; spin but be smart
                            (list !-13 !13)

                                       ; if back is greater, run away
                            (scale-motor-maze-orbit
                             (wire-bundle-l-light-b bundle)
                             (wire-bundle-r-light-b bundle)
                             bundle)))))))


;;Map wires to inputs and outputs
(defun mazem (inputs)
  (some* (
         (outputs lego-out)
         (bundle wire-bundle)

         (dansmom bool)(pause bool)(runaway bool)

         (justgotbumped bool)(justbecameit bool)(lastdansmom bool)
         (sendsignal bool)(receivesignal bool)(lastsendsignal bool)(lastreceivesignal bool)
         (justgotsignal bool)(timeout bool)(pausetimeout bool)
         (runawaydone bool)

         (dip0 bool)(dip1 bool)(dip2 bool)(dip3 bool)

         (saved-clock int)(start-signal int)(clock int)
         )

        ; shortened variable names
        (== dansmom (wire-bundle-dansmom bundle))
        (== pause (wire-bundle-pause bundle))
        (== runaway (wire-bundle-runaway bundle))
        (== clock (wire-bundle-clock bundle))
        (== dip0 (wire-bundle-dip0 bundle))
        (== dip1 (wire-bundle-dip1 bundle))
        (== dip2 (wire-bundle-dip2 bundle))
        (== dip3 (wire-bundle-dip3 bundle))


       ;;; Don't forget to set the defaults
        (lego-defaults outputs)


        ; hard connections to inputs
        (== (wire-bundle-l-light-fv bundle)
            (lego-in-ana26 inputs))
        (== (wire-bundle-l-light-fh bundle)
            (lego-in-ana27 inputs))

        (== (wire-bundle-r-light-fv bundle)
            (lego-in-ana22 inputs))
        (== (wire-bundle-r-light-fh bundle)
            (lego-in-ana23 inputs))

        (== (wire-bundle-l-light-b bundle)
            (rdivm (timesm (lego-in-ana18 inputs) NUMER) DENOM))
        (== (wire-bundle-r-light-b bundle)
            (rdivm (timesm (lego-in-ana19 inputs) NUMER) DENOM))

        ;(== (wire-bundle-left-motor bundle)
        ;    (ifm dip3 !0
        ;    (lego-out-motor1 outputs)))
        ;(== (wire-bundle-right-motor bundle)
        ;    (ifm dip3 !0
        ;    (lego-out-motor3 outputs)))

        (== (lego-out-motor1 outputs) (ifm dip3 !0 (wire-bundle-left-motor bundle)))
        (== (lego-out-motor3 outputs) (ifm dip3 !0 (wire-bundle-right-motor bundle)))

        (== (wire-bundle-dip0 bundle)
            (lego-in-dip0 inputs))
        (== (wire-bundle-dip1 bundle)
            (lego-in-dip1 inputs))
        (== (wire-bundle-dip2 bundle)
            (lego-in-dip2 inputs))
        (== (wire-bundle-dip3 bundle)
            (lego-in-dip3 inputs))

        (== (wire-bundle-ring bundle)
            (lego-in-dig5 inputs))

        (== (wire-bundle-bumpR bundle)
            (lego-in-dig6 inputs))
        (== (wire-bundle-bumpL bundle)
            (lego-in-dig7 inputs))

        (== (wire-bundle-clock bundle)
            (lego-in-cycle inputs))


        ; motor control
        (== (wire-bundle-clock-count-l bundle)
            (init-next 0 (ifm (wire-bundle-bumpL bundle)
                              (wire-bundle-clock bundle)
                              (wire-bundle-clock-count-l bundle))))

        (== (wire-bundle-clock-count-r bundle)
            (init-next 0 (ifm (wire-bundle-bumpR bundle)
                              (wire-bundle-clock bundle)
                              (wire-bundle-clock-count-r bundle))))

        (setmotorsm bundle)

        ;(== (wire-bundle-neelsrobot bundle) (wire-bundle-dip3 bundle))
        (== (wire-bundle-neelsrobot bundle) !'1b)

        (== (wire-bundle-M_MAX bundle) (ifm (wire-bundle-neelsrobot bundle)
                                         !18 !18))

        (== (wire-bundle-M_MIN bundle) (ifm (wire-bundle-neelsrobot bundle)
                                         !8 !8))

         (== (wire-bundle-frob bundle) !26)
;(lego-in-frob inputs))
             ;(init-next 0 (ifm (wire-bundle-dip2 bundle)
       ;                      (lego-in-frob inputs)
       ;                      (wire-bundle-frob bundle))))


        ; "it" in the old sense
        (== (wire-bundle-it bundle)
            (init-next '0b (ifm (andm (wire-bundle-ring bundle)
                                 (andm
                                  (notm (wire-bundle-lastring bundle))
                                  (wire-bundle-it bundle)))
                              !'0b
                              (ifm (andm (wire-bundle-ring bundle)
                                         (andm
                                          (notm (wire-bundle-lastring bundle))
                                          (notm (wire-bundle-it bundle))))
                                  !'1b
```

```
                          (wire-bundle-it bundle)))))


(== (lego-out-lcd-int0 outputs)
    (ifm (andm (notm dip0) (notm dip1))
         (wire-bundle-left-motor bundle)
         (ifm (andm dip0 (notm dip1))
              (wire-bundle-l-light-fv bundle)
              (ifm (andm (notm dip0) dip1)
                   (ifm dansmom !30 !0)
                   clock))))

(== (lego-out-lcd-int1 outputs)
    (ifm (andm (notm dip0) (notm dip1))
         (wire-bundle-right-motor bundle)
         (ifm (andm dip0 (notm dip1))
              (wire-bundle-r-light-fv bundle)
              (ifm (andm (notm dip0) dip1)
                   (ifm pause !30 !0)
                   saved-clock))))

(== (lego-out-lcd-int2 outputs)
    (ifm (andm (notm dip0) (notm dip1))
         (lego-in-ana16 inputs)
         (ifm (andm dip0 (notm dip1))
              (wire-bundle-l-light-b bundle)
              (ifm (andm (notm dip0) dip1)
                   (ifm runaway !30 !0)
                   start-signal))))

(== (lego-out-lcd-int3 outputs)
    (ifm (andm (notm dip0) (notm dip1))
         (ifm receivesignal !30 !0)
         (ifm (andm dip0 (notm dip1))
              (wire-bundle-r-light-b bundle)
              (ifm (andm (notm dip0) dip1)
                   (ifm justgotsignal !30 !1)
                   !44))))


 (== (lego-out-led1 outputs) sendsignal)

 (== (lego-out-led0 outputs)
     (orm (orm (wire-bundle-bumpR bundle) (wire-bundle-bumpL bundle))
          (wire-bundle-ring bundle)))


 ; supplementary state variables
;(== receivesignal (ifm (>m (lego-in-ana16 inputs) !110) !'1b !'0b))
 (== receivesignal (orm (lego-in-dig4 inputs) (>m (lego-in-ana16 inputs) !80 )))

(== (wire-bundle-lastring bundle)
    (init-next '0b (init-next '0b (wire-bundle-ring bundle))))

(== justgotbumped (init-next '0b (ifm (andm (wire-bundle-ring bundle)
                                            (notm (wire-bundle-lastring bundle)))
                                      !'1b
                                      !'0b)))


(== lastdansmom (init-next '0b (ifm dansmom !'1b !'0b)))

(== justbecameit (init-next '0b (ifm (andm dansmom
                                           (notm lastdansmom))
                                     !'1b
                                     !'0b)))

(== lastsendsignal (init-next '0b (ifm sendsignal !'1b !'0b)))
```

```
        (== lastreceivesignal (init-next '0b (ifm receivesignal !'1b !'0b)))


        (== justgotsignal (init-next '0b (ifm (andm receivesignal
                                                    (notm lastreceivesignal))
                                              !'1b
                                              !'0b)))

        (== timeout (>m (minusm clock saved-clock) !500))
        (== pausetimeout (>m (minusm clock saved-clock) !600))
        (== runawaydone (>m (minusm clock saved-clock) !70))


        ; game state variables
        (== pause (init-next '0b (ifm (notm pause)
                                 (ifm (orm (andm (andm (notm runaway) (notm dansmom))
                                 justgotsignal)
                                          (andm (andm (notm runaway) dansmom) timeout)) !'1b
                                          !'0b)
                                 (ifm (orm (andm (notm dansmom) justgotsignal)
                                          (andm dansmom pausetimeout)) !'0b !'1b))))

        (== runaway (init-next '0b (ifm (notm runaway)
                                   (ifm (orm (andm (andm (notm pause) dansmom) justgotsignal)
                                            (andm (andm (notm pause) (notm dansmom)
                                    justgotbumped)))
                                       !'1b !'0b)
                                   (ifm (orm (andm (notm dansmom) justgotsignal)
                                            (andm dansmom runawaydone))
                                       !'0b !'1b))))

        (== saved-clock
        ;    (init-next 0 (ifm (orm (orm runaway (notm dansmom))
       ;                          (orm justbecameit justgotsignal))
       ;                     clock
       ;                     saved-clock)))
            (init-next 0 (ifm (orm (notm dansmom) (andm dansmom justgotsignal))
                                    clock saved-clock)))

        (== start-signal (init-next 0 (ifm sendsignal start-signal clock)))

        (== sendsignal (init-next '0b (ifm (notm sendsignal)
                       (ifm (orm (orm (andm (andm (notm runaway) (notm dansmom)) justgotbumped)
                                      (andm (andm dansmom runaway) runawaydone))
                        (orm (andm (andm dansmom (andm (notm pause) (notm runaway)) timeout))
                                      (andm (andm dansmom (notm runaway)) pausetimeout)))
                                              !'1b !'0b)
                                         (ifm (>m (minusm clock start-signal) !40) !'0b !'1b))))

        (== dansmom (init-next INITIALDANSMOMVAL (ifm (notm dansmom)
                                    (ifm (orm (andm pause justgotsignal)
                                              (andm runaway justgotsignal)) !'1b !'0b)
                                    (ifm (orm (andm (notm runaway) pausetimeout)
                                             (andm runaway runawaydone)) !'0b !'1b))))
        outputs))

(defun maze* ()
       (out (mazem (in lego-in))))

(setq INITIALDANSMOMVAL '0b)
(setq NUMER !2)
(setq DENOM !1)
(setq BACKADD !15)
(makem (maze*) :module 'bytch :ic t)
(setq INITIALDANSMOMVAL '1b)
(setq NUMER !6)
(setq DENOM !5)
(setq BACKADD !15)
(makem (maze*) :module 'danbytch :ic t)
```