Testing Optimization:

Solving the Lifeguard Problem with Discrete and Continuous Methodologies

Humberto Barreto and Ryne Weppler

Fall 2010

DePauw University

Link to Excel file:

<http://academic.depauw.edu/~hbarreto/working/index.html>

**1. Introduction**

Optimization is a fundamental assumption in economics: companies minimize costs, individuals maximize utility, and firms maximize profits. Economists take for granted that decision-makers act rationally, weighing costs and benefits to find the best solution. While the idea is not incredible, optimization is a rather broad assumption, especially when it forms the foundation for modern economic theory.

We are interested in testing the assumption of optimization. We designed an experiment to examine the ability of individuals to behave rationally and optimize. Specifically, we created a game in Microsoft Excel, which places the user in the shoes of a lifeguard who needs to reach a victim. In this paper, we will discuss the lifeguard problem in more detail, including the underlying mathematics, and how to analyze the test subject’s performance. We also examine the results of our experiment and what they mean for economics and behavioral economics.

**2. Literature Review**

Previous research shows conflicting answers to the question of whether or not people optimize intuitively. Helbing (1996) found that gas-kinetic equations can be used to model vehicular traffic flow. The only fundamental difference in the two models was that in bottleneck situations, gas-kinetic scenarios actually speed up traffic whereas vehicular traffic slows down. This discrepancy is due to the ability of gas-kinetic systems to allocate more resources to bottlenecks in order to speed the process up. The fact that individuals naturally mimic this efficient movement pattern suggests that drivers can optimize and move efficiently, too.

Meanwhile, Braess’s Paradox suggests that traffic does not move with inherent efficiency. The paradox explains that sometimes adding a lane to a busy highway can actually slow down traffic, or alternatively closing a roadway can help traffic flow more efficiently. This counterintuitive reality stems from game theory, in particular the inability to distinguish between an equilibrium solution and the optimal solution.

Outside of the field of traffic analysis, Helbing (2001) offers further support for human optimization when he observes that individuals hiking in forests find a compromise between comfort (well-kept trails) and efficiency (more direct pathways) when asked to quickly reach an end destination. Though there is no specific metric to measure if they achieved optimality, this behavioral tendency to consider multiple dimensions of information when choosing a path does suggest humans are, if nothing else, seeking to maximize utility or minimize discomfort.

On the other hand, it appears that when individuals are faced with a decision between two items, one of which has a higher immediate payoff but whose utility decreases faster than the other item, individuals will behave sub-optimally, over-preferencing the instantly-gratifying item (Herrnstein and Prelec, 1991). This example can be extended to scenarios such as aversion to exercise. Exercising is not as immediately-gratifying as alternative activities, but over time will become more enjoyable as the individual gets in better shape, increases self-esteem, etc. However, many individuals will settle on more sedentary lifestyles because the immediate payoff is greater than the discomfort of exercising. Herrnstein and Prelec (1991) find evidence of melioration (failing to optimize because of a focus on the immediate payoff rather than the globally optimal choice) in computer-based experiments.

Neth, Sims, and Gray (2005) try eliminating melioration by providing participants with efficiency feedback on their performance, under the presumption that this added information will preclude the participants from falling into suboptimal behavioral patterns. On the contrary, however, nineteen of the twenty-two participants deliberately choose outcomes that reflect inefficiency caused by melioration. Thus, regardless of having sufficient information to behave optimally, individuals still make irrational decisions when faced with immediate versus delayed payoffs.

Although the previous studies offer some insight into the nature of human optimization, the results seem at odds with one another. Furthermore, the body of work is too sparse to get a sense of whether or not humans optimize. To offer further evidence and to add a nuance to the question of optimization, we implement the lifeguard problem in our experiment to analyze whether or not people behave optimally.

**3. The Lifeguard Problem**

The lifeguard problem is easy to describe. From an aerial perspective, as the lifeguard you have a horizontal and vertical distance to cover. When moving vertically, however, eventually you will hit the shoreline. When this transition occurs, your speed will decrease if you are coming from the beach because you cannot swim as fast as you can run. Knowing that seconds can mean the difference between life and death, you feel the urge to reach the drowning victim by the quickest path possible. Maybe you try to take the path that is the shortest distance because if you cover less ground, surely you will get there faster. On the other hand, what if swimming takes much, much longer than running? Indeed, maybe you are better suited to run on the beach as much as possible and dive into the water at the last second. Or, the quickest path could be somewhere in between these two extremes. Will you find this shortest path through instinct and intuition alone? Before addressing this question, we analyze the mathematics that explains the lifeguard problem.

3.1 *Snell’s Law, Fermat’s Principle, and Light Propagation*

The lifeguard problem is modeled after optical physics and the propagation of light waves. Snell’s Law and Fermat’s Principle both offer mathematical descriptions of how light waves find the path of least time when travelling through different media (for example, air and water). In Figure 1, the transition point from one medium to the other is that which satisfies the equation , where is the velocity of travel in the i’th medium and is the angle between the line of travel and the normal line (the line which is perpendicular to the interface or the beach in the lifeguard problem) in the i’th medium.

b

c

a

v1 > v2





o

Figure 1: Understanding the Lifeguard Problem

Starting from point a in Figure 1, the dashed line shows the least-time path, which depends on the speeds of travel in the two media. A faster lifeguard would have an optimal path with an entry point closer to point c. The path of least distance (a to b) seems attractive because it economizes on the total distance traveled, but too much time is spent in the water. It is optimal only when *v1* = *v2*. The path of least water (a to c to b) has the virtue of minimizing the time spent in the water, but the increase in total distance traveled is not worth it. Light correctly solves the problem, and we will test whether humans do the same.

Ganem (1998) supposes that human navigation (walking and running) can be modeled with optical principles and uses them to teach Snell’s Law. Ganem’s participants appear to have been exposed to the concept of least-time travel before participating. In our experiment, subjects will not be made explicitly aware of either the problem or the optimizing solution. Our experiment is designed to test how well humans can solve this problem.

*3.2 Continuous and Discrete Versions of the Problem*

In real life, the problem is a continuous one because the individual can change direction at any specific point and take any angled path he or she desires, but implementing the problem in Microsoft Excel required a discrete specification. Since the user may only travel in one-cell increments, the lifeguard can move left, right, up, down, or diagonally at a 45 degree angle. Furthermore, the entry point into the water must be an integer, since the user cannot change direction halfway through the cell. Although the optimal path in the discrete version and the continuous version will be different in most cases, the evaluation of speeds and distance to travel take place in either instance. We will discuss both versions of the problem.

*3.3 Solving the Continuous Problem with Calculus*

Though the lifeguard problem can be solved trigonometrically using Snell’s law, we approached the problem using calculus. When it comes to finding the shortest path, only a few factors really matter: the vertical distance to cover in the sand, the vertical distance in the water to cover, the horizontal distance to cover, your running speed, your swimming speed, and where you choose to enter the water. In equations, these variables will be referred to as a*, b, c, Vs, Vw*, and *x* respectively. Figure 2 illustrates the problem and makes clear that *x*, the entry point, is the endogenous variable: the lifeguard’s path is determined by this choice. With this information defined, we can formulate an objective function, seeking to minimize the time to the victim. For the sake of simplicity, we assume at this time that no riptides or currents are present.

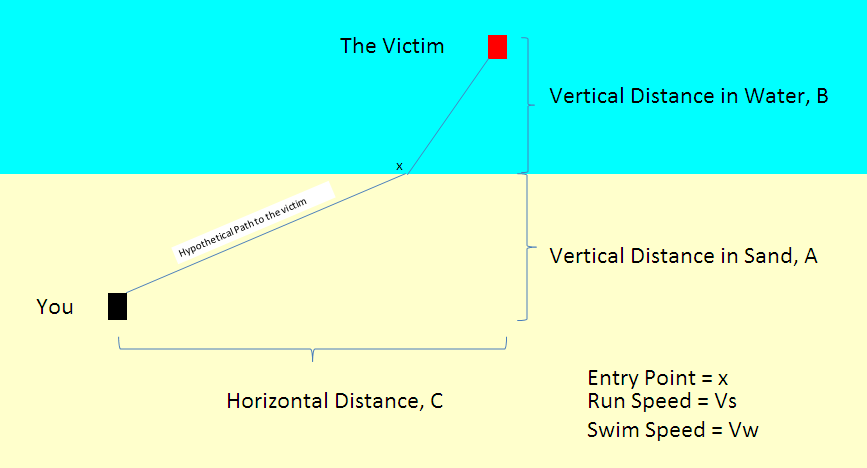


Figure 2: The Lifeguard Problem

Time spent running on the sand can be calculated by:

Similarly, time spent in water can be calculated by:

Combining these two equations will give the total time function, which we may now structure as a minimization problem, denoted as:

To solve this problem, we can take the derivative with respect to the choice variable, and set that derivative equal to zero. The distance variables, a, b, and c, are exogenous, and so are the lifeguard’s running and swimming speeds (assuming that individuals will move as quickly as possible). Thus, the only choice to make is where to enter the water, the variable we denote as *x*. After simplification, we have:

Unfortunately, setting the right-hand side equal to zero and solving for *x*, given the other parameters yields a quartic equation whose roots are available via analytical formula, but the expression is extremely complicated and unwieldy.[[1]](#footnote-1) Consequently, we turned to an algorithmic method to solve for the optimal *x*.

*3.3.1 The Newton-Raphson (Steepest Descent) Algorithm*

We use the Newton-Raphson iterative method to approximate the root of the derivative of the total time function at zero, which will be the optimal entry point. The procedure requires an initial *x* value, which can be chosen arbitrarily.[[2]](#footnote-2) From the initial *x*, we create a recursive sequence defined by: . The resulting limit of the convergent sequence will be the root of the function. Rather than formally prove the limit using analysis, however, we simply set a benchmark in our code that accepts the *x* value when the change in *x* is smaller than 0.000000001. Once this convergence criterion is met, we accept the *x* value as a reasonably close approximation to the exact optimal *x* or the optimal entry point for the lifeguard.

*3.3.2 The OptimalX Function*

We coded the Newton-Raphson method into a function called *OptimalX*. This enables us to use the function in a cell in Excel. Essentially, the algorithm runs through a loop in Visual Basic until the change in *x* is less than our precision benchmark of 0.000000001. It then outputs the optimal entry point into the cell where the user typed the function. We also wrote a function that takes any x value and outputs the time for that value. The full Visual Basic code for these functions can be found in the appendix of this paper.

*3.4 Solving the Discrete Problem*

Since the game we created is a discrete version of the lifeguard problem, we had to create an alternative way of finding the optimal entry point that accounted for the discrete nature of the problem. Figure 3 makes clear how the discrete and continuous versions differ. A straight diagonal line from lifeguard to victim is not available. The lifeguard must move in discrete steps from one square to the next.



Figure 3: The Discrete Version

3.4.1 *The OptimalXDiscrete and DiscreteTime Functions*

We wrote functions called *optimalxdiscrete* and *discretetime* to determine the optimal entry point and minimum time to the victim, given values of a, b, c, Vs, and Vw. The code for both functions is found in the appendix.

The discretetime function computes the time to the victim for any discrete entry point. It assumes the lifeguard will move to diagonal adjacent cells when possible to save time. In Figure 3, if the user chose the fifth cell from the left as the entry point, the code would compute the time according to the path in Figure 4.



Figure 4: A Discrete Path

The optimalxdiscrete function loops through all possible entry points using the discretetime function and reports the entry point with the lowest time. The loop continues running as long as the total time is falling and stops once total time rises.

3.4.2 *Discrete Time Solutions*

Unlike the continuous version of the problem, where optimal *x* rises as running speed goes up (given vs > vw), ceteris paribus, the discrete version’s optimal solution is a corner solution (least water) for vs/vw ≥ 2.4142. We set our running to swimming speed ratio at 5, so all of our discrete version trials yield corner solutions. This will make it easy to see if any of our subjects are optimizers. If vs/vw < 2.4142, the optimal solution will be the path that enters the water at a point where the lifeguard may take a purely diagonal path in the water.

**4. Experimental Method**

*4.1 Task and Apparatus*

Utilizing Visual Basic code, we designed a workbook in Microsoft Excel that allows users to play the lifeguard game, as shown in Figure 6.

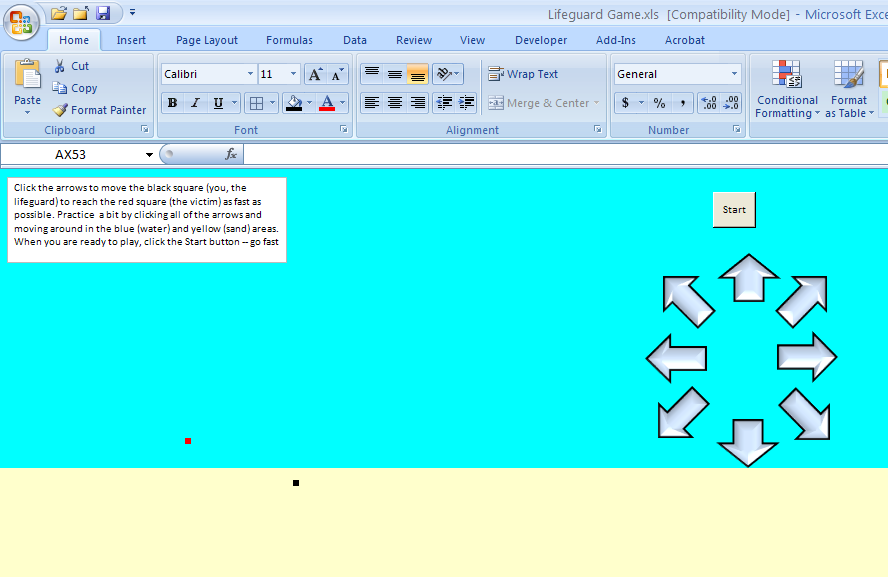


Figure 6: LifeguardGame.xls

The Excel file may be downloaded from <academic.depauw.edu/~hbarreto/working>. On open, you must click Enable Macros to play the game. The user clicks on the arrows to move the lifeguard (the black square) on the spreadsheet.

Coded into this game are momentary delays after clicking, which simulate the time it takes to travel one square cell. The appendix contains all of the game macros. The user can move left, right, up, down, or in a diagonal direction. Diagonal moves, however, take about 1.4 times as long as horizontal and vertical moves because they cover about 1.4 times as much distance. Additionally, moves in the water will have a longer delay. The experimenter has a great deal of control over this workbook. He or she can designate any of the position variables (a, b, and c) and the speed variables (vs and vw), which in turn determine the delay after the user clicks on the arrows.

Upon reaching the victim, the player is informed of his or her success (as shown in Figure 7), and the next trial begins with the lifeguard and the victim in new locations. To track the learning behavior of the user, each participant plays thirty trials of the game, with different positional parameters. We repeat sets of positional configurations (a, b, and c) to test if individuals perform better over time, indicating learning and improving performance.

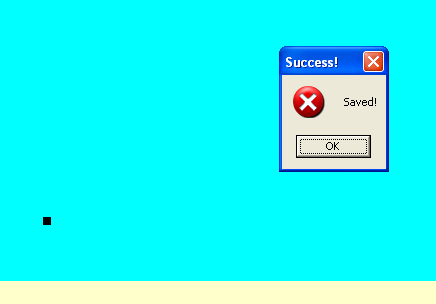


Figure 7: Lifeguard reaches the victim.

4.2 *Loss Aversion*

In half of the trials, the lifeguard starts in the water and has to save a victim on the beach. We are interested in examining if users behave differently when they have to navigate through the water first. It could be that when users feel themselves moving so slowly, they will be motivated to get out of the water as soon as possible. It is also possible that there is no difference between fast/slow or slow/fast and users will play the same way in either setup.

Testing for a difference in behavior depending on the order of the media is tantamount to a test of loss aversion. The discovery that many people seem to behave differently toward offers of expected gain versus expected loss is one of the earliest findings of behavioral and experimental economics and it has been repeated many times. Consider this simple scenario: you receive $1,000 and then you are given the option of an additional $500 or a coin is flipped and you win $1,000 if the coin comes up heads or you get nothing if it’s tails. Most people choose the $500 with certainty. The paradox arises when the same individual is placed in a different situation. The player is given $2,000 and forced to choose between losing $500 or a coin is flipped and $1,000 must be paid if the coin comes up heads or nothing is lost if it’s tails. In tests, many subjects switch and opt for the risky choice instead of losing $500 with certainty. The contradiction lies in the fact that the exact same choice has been offered in both cases: $1,500 with certainty or a 50/50 chance of $2000. The key finding is that an individual is willing to bear more risk to avoid painful losses than when gains are considered. In other words, the way the choice is posed, even though it is the exact same choice, affects the decision. This finding has even been extended to the monkeys, who show the same risk aversion. For more detail on this claim, see Lauri Santos, “A monkey economy as irrational as ours,” TED talk, http://www.ted.com/talks/laurie\_santos.html. The loss aversion example offered above is explained starting at the 10:50 mark.

The internal contradiction expressed in loss aversion experiments, along with a variety of other similar scenarios (e.g., framing effects), have often been found in behavioral economics and used as evidence of less than perfectly rational decision making. It will be interesting to see if our subjects behave differently, even though the problem is formally identical, depending on whether they start in the water or the sand.

*4.3 The “Tracker” Sheet*

Meanwhile, the movements that the user makes are recorded in a hidden worksheet, titled “Tracker,” which the experimenter uses to compile the results. This sheet serves as the control panel for the experimenter. From here, one can modify a number of dimensions in the experiment, including the running speed of the lifeguard, the swimming speed of the lifeguard, the position of the lifeguard relative to the victim, and the number of trials to use before ending the game. Each trial has a different set of positional parameters, which the experimenter can modify in the designated rows. The sheet also reports the optimal entry point and time for the given trial for the continuous and the discrete scenario. Our version of the game only uses the discrete version of the situation; however, having the continuous optimal solution provides confirmation that everything is working properly. The continuous optimal time will always be slightly less than or equal to the discrete time because the set of possible discrete times is a subset of the set of continuous times.

We wrote a macro that reports the cell address after each move. When the user makes a move, the tracker sheet moves down one row and records the user’s new cell address. Upon completion of a trial, the move recorder shifts to the right one column. The resulting output is a column for each trial that shows how the cell addresses change, with the total number of moves made reported at the top. Given this information, the experimenter can calculate the user’s time on each trial by using the homemade function timetracker. This function takes a range and calculates the number of horizontal, vertical, and diagonal moves made by the participant in each medium. Then, using the specified velocities, the function outputs a time that eliminates any hesitations or clicking errors made by the user by using the formula:

.

Thus, we are left with a time that is solely dependent on the decision making of the user. Figure 8 shows the “Tracker” worksheet after the user has played through three trials of the game.

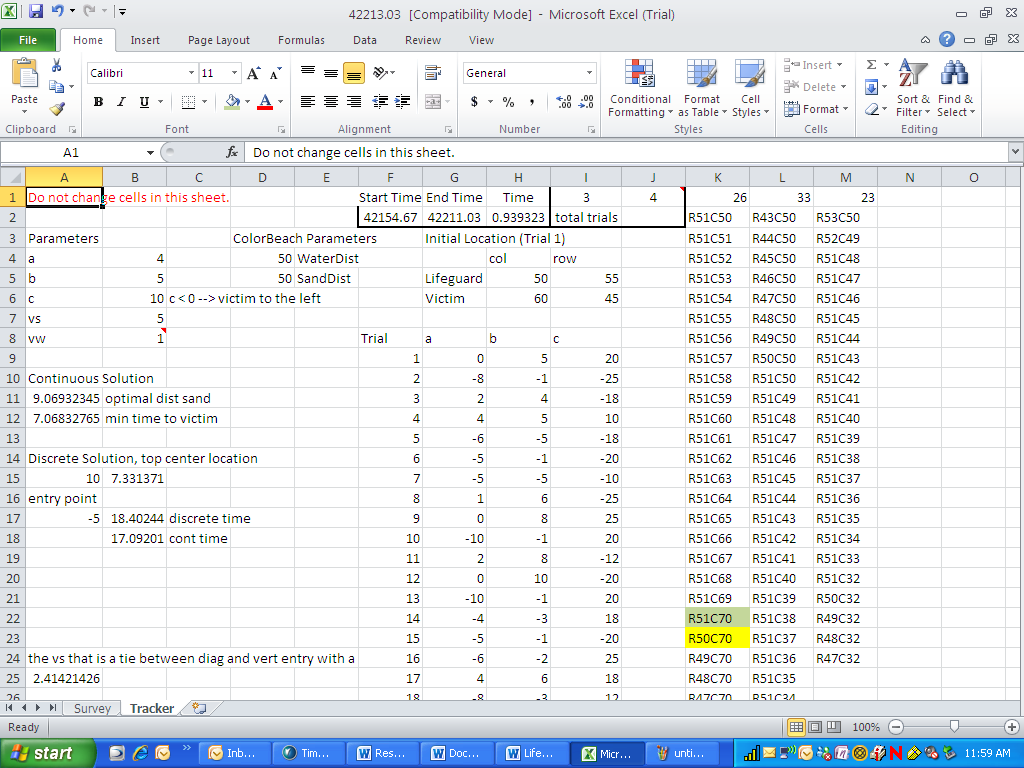


Figure 8: A sample “Tracker” sheet.

To recapitulate, the participant plays a game that simulates a beach rescue, with distance and speed conditions set by the experimenter. The participant plays through thirty rounds of the game and his or her performance is recorded for the experimenter’s analysis. Using the timetracker function, the experimenter can then calculate the time the user would have taken if he or she were moving constantly, without miniscule pauses in between clicks.

4.2 *Procedure*

4.2.1 *Pilot Testing*

Before delving into the official trials of our experiment, we decided to implement a pilot testing program to troubleshoot any programming errors and garner general feedback about the playability and experience of the lifeguard game. We had six individuals come to the lab to play through our twenty trials and discuss their thoughts, including where they thought the game could be improved and where there were strengths we could build upon. We were pleased with how smoothly the game went. Most users reported that the game was fun and that we could easily increase the number of trials to thirty. They also reported that they were engaged and trying to win. We also picked up on some errors in the way our reporting was set up. We had network issues and struggled to compile individual results in a single workbook. Consequently, our old method of putting all of the worksheets into a master file as the users complete the game has been replaced with a method that allows the experimenter to compile the sheets and times into a workbook once everyone has completed the experiment.

4.2.2 *The Experiment*

Having successfully created an Excel implementation of the lifeguard problem, we advertised and recruited students from DePauw University to come participate in our experiment with the promise of a chance to win Wal-Mart gift cards. Individuals came to a computer lab, where they viewed and pledged informed consent. We then gave instructions on how to access the file and how to play the game. We allowed the participants to move around on the sheet to get a feel for the controls and the difference between run and swim speed. After this practice period was over, individuals competed against one another, seeking to complete their set of thirty trials before everyone else. Three gift cards per experimental group were awarded probabilistically according to the performance of the participants. In other words, the top performer had a much better chance of winning a gift card than the worst performer. After completing the game, a survey was administered asking individuals to report their strategies, perceived math ability, handedness, and demographic information. Upon completion, participants were compensated, debriefed, and dismissed from the lab. The process took roughly thirty minutes. We ran three experimental groups in this fashion, with a total of fifty-nine participants.

4.3 *Evaluation Metrics*

Once we had the subject’s trial time, we needed to know how well the individual performed. Since we wrote the OptimalXDiscrete and DiscreteTime functions, this process became straightforward. Our primary method of evaluating performance is the following quotient, which we calculated for each subject for each trial of the experiment:

To calculate the user’s time, we used the timetracker function mentioned earlier, rather than actually time the user. Meanwhile, participants were rewarded according to their actual performance in the lab to keep them engaged and involved in the problem. The decision to not time our participants eliminates time discrepancies that would arise from fast or slow clicking speeds and ability to navigate the interface, variables that we are not interested in since they do not have any relevance to the optimization problem of focus.

The second term in the numerator, OptimalTime, is obtained from the optimalxdiscrete function. Clearly, the lowest value the numerator can have is zero, when the subject chooses the least time entry point.

MaxTime was calculated using a function titled maxtime, which was fundamentally identical to the optimalxdiscrete function; however, instead of finding the lowest time reported by the discretetime function, we report the largest time.

Putting the three components together, we can solve for our metric. The resulting number represents how much the user missed the optimal time by, as a percentage of how much the user could possibly miss by. We use this metric for a couple of reasons. First, simpler metrics such as have a tendency to inflate as the time it takes to complete the trial increases. The minimum time to reach the victim is built into the numerator and the denominator, so an identical distribution of times simply shifted vertically will yield higher scores, ceteris paribus. Also, the spread in the distribution of times is often small, so a score may look very impressive, but it doesn’t account for the fact that the user will get a time close to the optimal no matter what, by the nature of the trial. We measure performance relative to the max time to account for this issue.

**Preliminary Hypotheses or Lack Thereof**

We are interested in innate ability to reach the victim quickly because this result reflects an individual’s ability to optimize instinctively. The practice round should provide them with enough experience to do well on the first trial, if individuals are optimizers. Thus, the first trials will give us initial evidence for or against human rationality.

It is also worth examining the learning patterns of these individuals. Presumably, if they understand the problem from the beginning, learning will be irrelevant and they will perform at a consistently high level. If they struggle at first, however, do the participants improve their performance, or do they continue to make similar mistakes and perform poorly? Sustained suboptimal performance would suggest that people cannot always be expected to act rationally. As stated above, we will also be able to test the condition of whether starting in the sand or the water affects learning patterns or ability. It should not, because the optimal path is the same for a particular positional configuration, regardless of whether the lifeguard is on the sand and the victim in the water or vice versa.

In the framework of loss aversion, transitioning to a new medium is the gamble that the user must consider. Though she may have a vague idea of how fast she can move in each medium, she knows neither velocity for certain and has less awareness of the alternative medium’s velocity. Thus there is a level of uncertainty which makes the choice to transition to a new medium risky. When starting in the water, the lifeguard is considering a gain in speed. Under loss aversion, she will be more willing to take a risk, since she is evaluating a gain. However, when starting on the beach, the individual knows she will move slower in the water, and will avoid this loss. We have no a priori expectations for the results.

**Results**

*Learning and Optimization*

As mentioned previously, we repeated certain parameter sets in order to examine if participants performed better as they progressed through the game, ceteris paribus. Analysis of these duplicate trials is our primary means of examining optimization and learning. The following table breaks down when each parameter set occurred and its corresponding positional parameters:

|  |  |  |
| --- | --- | --- |
|  | Positional Parameters | Trials |
| Duplicate Set 1 | a = 0, b = 5, c = 20 | 1, 20, 27 |
| Duplicate Set 2 | a = -6, b = -2, c = -25 | 2, 16, 24 |
| Duplicate Set 3 | a = -5, b = -1, c = -20 | 6, 15, 29 |
| Duplicate Set 4 | a = 1, b = 6, c = -25 | 8, 13, 22 |

In duplicate set 1, the distribution of times clearly decreases from the first exposure to the second exposure, providing preliminary evidence of learning behavior. This distribution is even more powerful when removing seven apparent non-learners:

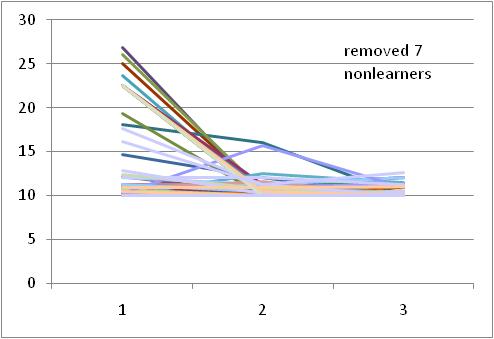


Figure 9: Evidence of Learning in parameter set 1

Even with the inclusion of the nonlearners, a statistically significant difference exists between the means of the first and second exposure, *t =* 3.92, *p <* .001.

The remaining three duplicate sets did not have statistically significant differences in the means; however, the graphs follow a pattern similar to Figure 9. These graphs may be found in the appendix of this paper.

We also utilized f-tests to examine the learning behavior of the participants. First, we ran an unrestricted ordinary least squares regression of percentage inefficiency on twenty-nine trial dummy variables and fifty-eight user dummy variables. This specification treats each user and each trial as unique. Next, we ran a restricted regression that treated trials 1, 20, and 27 as the same, reflecting an assumption that individuals do not learn and will behave the same across the three duplicate trials. Using the same reasoning, we ran restricted regressions of the remaining three duplicate parameter sets. The OLS results were as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Unrestricted | Restriction of Set 1 | Restriction of Set 2 | Restriction of Set 3 | Restriction of Set 4 |
| Intercept | 0.475  (0.078) | 0.475 | 0.475 | 0.475 | 0.475 |
| Trial 1 | 0.166  (0.061) | 0.007 | 0.166 | 0.166 | 0.166 |
| Trial 2 | -0.124  (0.035) | -0.124 | -0.135 | -0.124 | -0.124 |
| Trial 4 | -0.040  (0.039) | -0.040 | -0.040 | -0.040 | -0.040 |
| Trial 5 | -0.087  (0.035) | -0.087 | -0.087 | -0.087 | -0.087 |
| Trial 6 | -0.133  (0.034) | -0.133 | -0.133 | -0.120 | -0.133 |
| Trial 7 | -0.051  (0.037) | -0.051 | -0.051 | -0.051 | -0.051 |
| Trial 8 | -0.032  (0.045) | -0.032 | -0.032 | -0.032 | -0.063 |
| Trial 9 | -0.069  (0.041) | -0.069 | -0.069 | -0.069 | -0.069 |
| Trial 10 | -0.050  (0.035) | -0.050 | -0.050 | -0.050 | -0.050 |
| Trial 11 | 0.006  (0.043) | 0.006 | 0.006 | 0.006 | 0.006 |
| Trial 12 | 0.002  (0.038) | 0.002 | 0.002 | 0.002 | 0.002 |
| Trial 13 | -0.092  (0.036) | -0.092 | -0.092 | -0.092 | EXCLUDED |
| Trial 14 | -0.135  (0.034) | -0.135 | -0.135 | -0.135 | -0.13548 |
| Trial 15 | -0.128  (0.035) | -0.128 | -0.128 | EXCLUDED | -0.12752 |
| Trial 16 | -0.138  (0.034) | -0.138 | EXCLUDED | -0.138 | -0.13801 |
| Trial 17 | -0.003  (0.039) | -0.003 | -0.003 | -0.003 | -0.00296 |
| Trial 18 | -0.010  (0.035) | -0.010 | -0.010 | -0.010 | -0.01029 |
| Trial 19 | 0.000  (0.042) | 0.000 | 0.000 | 0.000 | -0.00016 |
| Trial 20 | -0.084  (0.037) | EXCLUDED | -0.084 | -0.084 | -0.08441 |
| Trial 21 | -0.027  (0.038) | -0.027 | -0.027 | -0.027 | -0.02715 |
| Trial 22 | -0.066  (0.039) | -0.066 | -0.066 | -0.066 | EXCLUDED |
| Trial 23 | -0.059  (0.036) | -0.059 | -0.059 | -0.059 | -0.05901 |
| Trial 24 | -0.144  (0.034) | -0.144 | EXCLUDED | -0.144 | -0.14396 |
| Trial 25 | 0.020  (0.041) | 0.020 | 0.020 | 0.020 | 0.019956 |
| Trial 26 | -0.055  (0.035) | -0.055 | -0.055 | -0.055 | -0.05471 |
| Trial 27 | -0.062  (0.043) | EXCLUDED | -0.062 | -0.062 | -0.06192 |
| Trial 28 | 0.004  (0.047) | 0.004 | 0.004 | 0.004 | 0.003548 |
| Trial 29 | -0.099  (0.038) | -0.099 | -0.099 | EXCLUDED | -0.09876 |
| Trial 30 | 0.051  (0.039) | 0.051 | 0.051 | 0.051 | 0.050947 |
| *N* | 59 | 59 | 59 | 59 | 59 |
| *R*2 | 0.33 | 0.30 | 0.33 | 0.3298 | 0.329 |

Note: Numbers in parentheses are robust standard errors.

We then run f-tests on each of the restricted models with respect to the unrestricted model. Similar to our analysis of means, only the first restriction was formally rejected utilizing an F-test, *F* = 35.23, *p <* 0.001.

*Loss Aversion*

You may have noticed our duplicate parameter sets closely resembled one another in absolute values. This similarity is because duplicate set 1 is a mirror image of duplicate set 3, and duplicate set 2 is a mirror image of duplicate set 4. This specification holds the distance to be run and swum constant, but changes the framework of the problem. Whereas in duplicate set 1, the lifeguard starts on the shore, in duplicate set 3, the lifeguard starts in the water. Since the optimal path is the same in both problems, a significant difference in performance would illustrate loss aversion. We see this statistically significant result in trials 1 / 6, 2 / 8, 13 / 16, and 24 / 22.

We also ran F-tests, which created the restriction that each mirrored couple was identical. We were able to reject this specification in three of the six tests: trials 1 / 6, 2 / 8, and 24 / 22. Thus, we see repeated evidence that individuals perform better when starting in the water, an indication that they were more willing to transition early than when they start on the shore. Individuals are very avoidant of the water when they start in it; however, when they start on the beach, they exhibit less avoidance. In other words, they are less afraid of risking their performance on the relatively slow option of swimming. Since the user moves very quickly on the beach but very slowly in the water, starting in the water gives the user a mental framework of loss and delay, whereas starting on the beach gives the user a mental framework of gain and speed. This willingness to take risks on the water while in a positive mindset of gain validates the presence of loss aversion. Interestingly, loss aversion prompts individuals to behave more efficiently in the case of a loss mindset.

**Discussion**

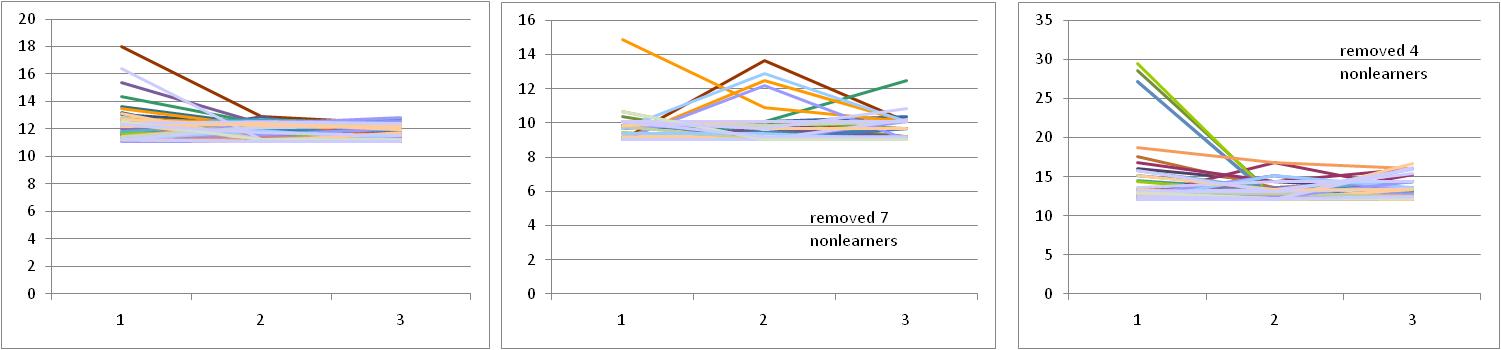
The presence of learning behavior in our participants suggests that initial optimization was not present. While one may argue that participants did not understand the game at first, we should have precluded this effect by giving them a practice period. Although they did not initially optimize, they end up performing very well, as evidenced by the very low inefficiency scores (mean of 0.14). Thus, we conclude that people do not initially optimize but learn quickly and behave close to optimality in a general sense. However, analysis of loss aversion reveals that this improved performance is especially prominent when individuals start in the water, or an environment of loss. Since the mirror image trials were identical in solution, our findings show that people do not behave rationally when starting on the shore. The framework of the problem matters, when mathematically speaking it should not.

We would like to note that the range of times to the victim is somewhat small. In other words, a lifeguard who takes an inefficient path will not be penalized by a large amount of time. The critical reader may question why, practically speaking, it matters if individuals optimize when the spread of times is so small. After all, even if the participant completely misses the problem, the lifeguard will still get to the victim within a few seconds of the optimal. We asked ourselves the same question but hasten to remind you that in a situation such as an ocean rescue, time is critical, and seconds can be the difference between life and death. Thus, the range may be small, but it is certainly not negligible.

We also acknowledge that our experiment is still in its nascence. Modifications could be designed, which would make a more realistic environment. For example, the workbook could be modified in a way that creates a tide in the water. After speaking with lifeguarding schools, it appears that this environmental factor plays a significant role in where the lifeguard chooses to jump in because the lifeguard may use these conditions to his or her advantage. We would also like to continue developing the workbook to make the game more realistic, providing more sensory feedback and more accurately instilling a sense of urgency in the user.

While the field of behavioral economics continues to expand, experiments such as these are challenges to conventional economic thought. Though we see individuals do not actually behave rationally, the difference may be negligible in practical applications, and thus models assuming optimization may remain justified. However, we offer further empirical support for the concept of loss aversion, which has both statistical and apparent practical significance. Thus, in analyses of human behavior, rationality of decision making may be generally assumed, but when changing the framing of the problem, the experimenter must be cognizant of behavioral changes that may occur when they should not from a purely rational perspective.

**Appendix of Supplementary Figures**



**Distribution of trial times for duplicate parameter sets 2, 3, and 4 respectively**

**Appendix of Visual Basic Functions**

1. **Optimization Functions**

**OptimalX**

Function optimalx(a As Range, b As Range, c As Range, Vs As Range, Vw As Range)

'This function takes parameters for the lifeguard problem and outputs the optimal solution

'a = vertical distance the lifeguard is away on the sand from the shoreline

'b = vertical distance the victim is away in the water from the shoreline

'c = total horizontal distance from lifeguard to victim

'vs = velocity on the sand, run speed

'vw = velocity on the water, swim speed

Dim X As Double

Dim PreviousX As Double

Dim I As Long

Dim DeltaX As Double

Dim FX As Double

Dim dFX As Double

DeltaX = 1

X = c.Value / 2 'this is the initial value, X0 (and BTW the best initial, in general, because we don't know where X\* is)

'check to see if victim directly north of lifeguard in which case

'lifeguard should enter water and swim vertically to victim

'FX in code below returns error because slope of function is not defined at zero

If c.Value = 0 Then

optimalx = 0

Exit Function

End If

FX is the objective function and dFX is the derivative with respect to x.

While DeltaX > 0.000000001

PreviousX = X

FX = X \* Vw.Value \* Sqr((c.Value - X) ^ 2 + b.Value ^ 2) - (c.Value - X) \* Vs.Value \* Sqr(a.Value ^ 2 + X ^ 2)

dFX = Vw.Value \* Sqr((c.Value - X) ^ 2 + b.Value ^ 2) - X \* Vw.Value \* ((((c.Value - X) ^ 2 + b.Value ^ 2)) ^ (-1 / 2)) \* (c.Value - X) + Vs.Value \* Sqr(a.Value ^ 2 + X ^ 2) - (c.Value - X) \* Vs.Value \* ((a.Value ^ 2 + X ^ 2) ^ (-1 / 2)) \* X

X = X - FX / dFX

DeltaX = Abs(X - PreviousX)

Wend

'lifeguard in water/victim on sand case

If a.Value < 0 And b.Value < 0 Then

X = c.Value - X

End If

optimalx = X

End Function

**TimetoVictim**

Function TimetoVictim(a As Range, b As Range, c As Range, Vs As Range, Vw As Range, X As Range)

'This function takes parameters for the lifeguard problem and optimalx and outputs minimum time

'a = vertical distance the lifeguard is away on the sand from the shoreline

'b = vertical distance the victim is away in the water from the shoreline

'c = total horizontal distance from lifeguard to victim

'vs = velocity on the sand, run speed

'vw = velocity on the water, swim speed

'x = entry point (can be optimalx computed using the function above)

Dim Reversex As Double

'lifeguard in water/victim on sand case

If a.Value < 0 And b.Value < 0 Then

Reversex = c.Value - X.Value

TimetoVictim = Sqr(b.Value ^ 2 + Reversex ^ 2) / Vs.Value + Sqr((c.Value - Reversex) ^ 2 + a.Value ^ 2) / Vw.Value

Else

TimetoVictim = Sqr(a.Value ^ 2 + X.Value ^ 2) / Vs.Value + Sqr((c.Value - X.Value) ^ 2 + b.Value ^ 2) / Vw.Value

End If

End Function

**OptimalXDiscrete**

Function OptimalXDiscrete(a As Range, b As Range, c As Range, Vs As Range, Vw As Range)

Dim TotalTime As Double

Dim TotalTimeNew As Double

Dim ResultsArray(1 To 2) As Double

TotalTime = DiscreteTime(a, b, c, Vs, Vw, 0)

Dim I As Integer

Dim J As Integer

J = 1

'L in sand & V in water and the reverse, and L left of V

If c.Value >= 0 Then

For I = 1 To c

TotalTimeNew = DiscreteTime(a, b, c, Vs, Vw, J)

J = I + 1

If TotalTimeNew < TotalTime Then

TotalTime = TotalTimeNew

Else

GoTo FoundOptimal1

End If

Next

FoundOptimal1:

ResultsArray(1) = I - 1

ResultsArray(2) = TotalTime

OptimalXDiscrete = ResultsArray

Exit Function

'L in sand & V in water and the reverse, and L right of V

Else 'c.Value < 0

J = -1

For I = -1 To c Step -1

TotalTimeNew = DiscreteTime(a, b, c, Vs, Vw, J)

J = I - 1

If TotalTimeNew < TotalTime Then

TotalTime = TotalTimeNew

Else

GoTo FoundOptimal2

End If

Next

FoundOptimal2:

ResultsArray(1) = I + 1

ResultsArray(2) = TotalTime

OptimalXDiscrete = ResultsArray

Exit Function

End If

End Functio

**DiscreteTime**

Function DiscreteTime(a As Range, b As Range, c As Range, Vs As Range, Vw As Range, EntryPoint As Variant) As Variant

Dim DiagonalMoves As Integer

Dim VerticalMoves As Integer

Dim PreDiscreteTime As Double

Dim HorizontalWater As Integer

Dim NumberDMovesSand As Integer

Dim NumberVMovesSand As Integer

Dim NumberHMovesSand As Integer

If c < 0 And EntryPoint > 0 Then

'MsgBox "The victim is to the left of you, so you need to enter a negative value for entry point.", vbCritical, "Negative C Value Requires Negative Entry Point"

'Exit Function

End If

If a >= 0 Then 'conventional case, lifeguard starts on the sand

'Determines the number of diagonal cells run on sand (up to entry point)

If EntryPoint = a Then

NumberDMovesSand = Abs(a)

NumberVMovesSand = 0

NumberHMovesSand = 0

Else

If Abs(c) > Abs(a) Then

If Abs(EntryPoint) < Abs(a) Then

NumberDMovesSand = Abs(EntryPoint)

NumberVMovesSand = Abs(a) - Abs(EntryPoint)

NumberHMovesSand = 0

Else

NumberDMovesSand = Abs(a)

NumberVMovesSand = 0

NumberHMovesSand = Abs(EntryPoint) - Abs(a)

End If

Else

NumberDMovesSand = c

NumberVMovesSand = Abs(a) - Abs(c)

NumberHMovesSand = 0

End If

End If

'Determines the number of diagonal cells swum in water (past entry point)

If EntryPoint = c Then

DiagonalMoves = 0

Else

If Abs(c - EntryPoint) < Abs(b) Then

DiagonalMoves = Abs(c - EntryPoint)

Else

DiagonalMoves = Abs(b)

End If

End If

'Since a diagonal move yields a move up, this calculates the consequent vertical moves

VerticalMoves = Abs(b) - DiagonalMoves

'Determines if the path will result in horizontal swimming (the case when the entry point

'is early enough that a diagonal path of swimming will not hit the right borderline)

If Abs(c - EntryPoint) < Abs(b) Or c = EntryPoint Then

HorizontalWater = 0

Else

HorizontalWater = Abs(c - EntryPoint) - DiagonalMoves

End If

DiscreteTime = (NumberVMovesSand + NumberHMovesSand) / Vs + NumberDMovesSand \* Sqr(2) / Vs + DiagonalMoves \* Sqr(2) / Vw + VerticalMoves / Vw + HorizontalWater / Vw

Exit Function

Else 'handles lifeguard in the water

'Determines the number of diagonal cells swum (up to entry point)

If -Abs(EntryPoint) = a Then

DiagonalMoves = Abs(a)

HorizontalWater = 0

VerticalMoves = 0

Else

If Abs(c) > Abs(a) Then

If Abs(EntryPoint) < Abs(a) Then

DiagonalMoves = Abs(EntryPoint)

HorizontalWater = 0

VerticalMoves = Abs(a) - Abs(EntryPoint)

Else

DiagonalMoves = Abs(a)

HorizontalWater = Abs(EntryPoint) - Abs(a)

VerticalMoves = 0

End If

Else

DiagonalMoves = Abs(c)

HorizontalWater = 0

VerticalMoves = Abs(a) - Abs(c)

End If

End If

'Determines the number of diagonal cells to run (past entry point)

If EntryPoint = c Then

NumberDMovesSand = 0

Else

If Abs(c - EntryPoint) < Abs(b) Then

NumberDMovesSand = Abs(c - EntryPoint)

Else

NumberDMovesSand = Abs(b)

End If

End If

'Since a diagonal move yields a move down, this calculates the consequent vertical moves

NumberVMovesSand = Abs(b) - NumberDMovesSand

'Determines if the path will result in horizontal running (the case when the entry point

'is early enough that a diagonal path of running will not hit the right borderline)

If Abs(c - EntryPoint) < Abs(b) Or c = EntryPoint Then

NumberHMovesSand = 0

Else

NumberHMovesSand = Abs(c - EntryPoint) - NumberDMovesSand

End If

DiscreteTime = (NumberVMovesSand + NumberHMovesSand) / Vs + NumberDMovesSand \* Sqr(2) / Vs + DiagonalMoves \* Sqr(2) / Vw + VerticalMoves / Vw + HorizontalWater / Vw

Exit Function

End If

End Function

1. **Game Controller Functions**

'Macros below used to set up and play the lifeguard game

'must set in Tracker sheet and used in BarretosPause, ColorBeach, and ClearLifeguard macros

Public WaterDist As Integer

Public SandDist As Integer

Public DistanceTravelled As Double

'set pause lengths

Sub BarretosPause()

' WaterDist not set here because set when called from buttons macro

Dim PauseTime, Start, Finish, TotalTime

If ActiveCell.Row <= WaterDist Then 'in the water

PauseTime = DistanceTravelled

Start = Timer

Do While Timer < Start + PauseTime

DoEvents

Loop

Finish = Timer

TotalTime = Finish - Start

Else 'in the sand

PauseTime = DistanceTravelled / Sheets("Tracker").Cells(7, 2).Value

Start = Timer

Do While Timer < Start + PauseTime

DoEvents

Loop

Finish = Timer

TotalTime = Finish - Start

End If

'track choices made

'move #

Sheets("Tracker").Cells(1, 10 + Sheets("Tracker").Cells(1, 10).Value).Value = Sheets("Tracker").Cells(1, 10 + Sheets("Tracker").Cells(1, 10).Value).Value + 1

'move made

Sheets("Tracker").Cells(1 + Sheets("Tracker").Cells(1, 10 + Sheets("Tracker").Cells(1, 10).Value).Value, 10 + Sheets("Tracker").Cells(1, 10).Value).Value = ActiveCell.Address(ReferenceStyle:=xlR1C1)

End Sub

'move buttons

Sub MoveUp()

Application.Interactive = False 'blocks user input, must be used with DoEvents

WaterDist = Sheets("Tracker").Cells(4, 4).Value

DistanceTravelled = 1

If ActiveCell.Row - 1 = WaterDist Then 'about to enter water from sand

DistanceTravelled = Sheets("tracker").Cells(7, 2).Value \* (0.5 \* DistanceTravelled + 0.5 \* DistanceTravelled / Sheets("Tracker").Cells(7, 2).Value)

End If

BarretosPause

ClearLifeguard

ActiveCell.Offset(-1, 0).Select

Lifeguard

Application.Interactive = True

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then

MsgBox "Saved!", vbCritical, "Success!"

ClearLifeguard

NextTrial

End If

End Sub

Sub MoveRight()

Application.Interactive = False 'blocks user input, must be used with DoEvents

WaterDist = Sheets("Tracker").Cells(4, 4).Value

DistanceTravelled = 1

BarretosPause

ClearLifeguard

ActiveCell.Offset(0, 1).Select

Lifeguard

Application.Interactive = True

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then

MsgBox "Saved!", vbCritical, "Success!"

ClearLifeguard

NextTrial

End If

End Sub

Sub MoveDiagonalUpRt()

Application.Interactive = False 'blocks user input, must be used with DoEvents

WaterDist = Sheets("Tracker").Cells(4, 4).Value

DistanceTravelled = Sqr(2)

If ActiveCell.Row - 1 = WaterDist Then 'about to enter water from sand

DistanceTravelled = Sheets("tracker").Cells(7, 2).Value \* (0.5 \* DistanceTravelled + 0.5 \* DistanceTravelled / Sheets("Tracker").Cells(7, 2).Value)

End If

BarretosPause

ClearLifeguard

ActiveCell.Offset(-1, 1).Select

Lifeguard

Application.Interactive = True

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then

MsgBox "Saved!", vbCritical, "Success!"

ClearLifeguard

NextTrial

End If

End Sub

Sub MoveDiagonalDownRt()

Application.Interactive = False 'blocks user input, must be used with DoEvents

WaterDist = Sheets("Tracker").Cells(4, 4).Value

DistanceTravelled = Sqr(2)

BarretosPause

ClearLifeguard

ActiveCell.Offset(1, 1).Select

Lifeguard

Application.Interactive = True

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then

MsgBox "Saved!", vbCritical, "Success!"

ClearLifeguard

NextTrial

End If

End Sub

Sub MoveDown()

Application.Interactive = False 'blocks user input, must be used with DoEvents

WaterDist = Sheets("Tracker").Cells(4, 4).Value

DistanceTravelled = 1

BarretosPause

ClearLifeguard

ActiveCell.Offset(1, 0).Select

Lifeguard

Application.Interactive = True

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then

MsgBox "Saved!", vbCritical, "Success!"

ClearLifeguard

NextTrial

End If

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then MsgBox "Saved!", vbCritical, "Success!"

End Sub

Sub MoveDiagonalDownLt()

Application.Interactive = False 'blocks user input, must be used with DoEvents

WaterDist = Sheets("Tracker").Cells(4, 4).Value

DistanceTravelled = Sqr(2)

BarretosPause

ClearLifeguard

ActiveCell.Offset(1, -1).Select

Lifeguard

Application.Interactive = True

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then

MsgBox "Saved!", vbCritical, "Success!"

ClearLifeguard

NextTrial

End If

End Sub

Sub MoveLeft()

Application.Interactive = False 'blocks user input, must be used with DoEvents

WaterDist = Sheets("Tracker").Cells(4, 4).Value

DistanceTravelled = 1

BarretosPause

ClearLifeguard

ActiveCell.Offset(0, -1).Select

Lifeguard

Application.Interactive = True

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then

MsgBox "Saved!", vbCritical, "Success!"

ClearLifeguard

NextTrial

End If

End Sub

Sub MoveDiagonalUpLt()

Application.Interactive = False 'blocks user input, must be used with DoEvents

WaterDist = Sheets("Tracker").Cells(4, 4).Value

DistanceTravelled = Sqr(2)

If ActiveCell.Row - 1 = WaterDist Then 'about to enter water from sand

DistanceTravelled = Sheets("tracker").Cells(7, 2).Value \* (0.5 \* DistanceTravelled + 0.5 \* DistanceTravelled / Sheets("Tracker").Cells(7, 2).Value)

End If

BarretosPause

ClearLifeguard

ActiveCell.Offset(-1, -1).Select

Lifeguard

Application.Interactive = True

If Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Address = ActiveCell.Address Then

MsgBox "Saved!", vbCritical, "Success!"

ClearLifeguard

NextTrial

End If

End Sub

'initializes the game

Sub Start()

ActiveSheet.Unprotect

Cells.Clear

ColorBeach

'clear old tracking data

Sheets("Tracker").Range(Sheets("Tracker").Columns(10), Sheets("Tracker").Columns(100)).ClearContents

'Trial 1

Sheets("Tracker").Cells(1, 10).Value = 1

'put down victim

Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Select

ActiveCell.Interior.ColorIndex = 3

'put down lifeguard

Cells(Sheets("Tracker").Cells(5, 9).Value, Sheets("Tracker").Cells(5, 8).Value).Select

Cells.Locked = True

Lifeguard

'put away the Start button

ActiveSheet.Buttons("Button 1").Left = Columns(250).Left

ActiveSheet.Protect , AllowFormattingCells:=True

Sheets("Tracker").Cells(2, 6).Value = Timer

End Sub

Sub reset()

Start

ActiveSheet.Unprotect

'show the Start button

ActiveSheet.Buttons("Button 1").Left = Columns(120).Left

ActiveSheet.Protect , AllowFormattingCells:=True

End Sub

Sub Lifeguard()

ActiveCell.Interior.ColorIndex = 1

End Sub

Sub ClearLifeguard()

WaterDist = Sheets("Tracker").Cells(4, 4).Value

If ActiveCell.Row <= WaterDist Then

ActiveCell.Interior.ColorIndex = 8

Else

ActiveCell.Interior.ColorIndex = 19

End If

End Sub

Sub NextTrial()

ActiveSheet.Unprotect

'update trial number

Sheets("Tracker").Cells(1, 10).Value = Sheets("Tracker").Cells(1, 10).Value + 1

'sheet uses VLOOKUP to use trial # to get parameters for that trial

'check to see if game over

If Sheets("Tracker").Cells(1, 10).Value > Sheets("Tracker").Cells(1, 9).Value Then

Sheets("Tracker").Cells(2, 7).Value = Timer

MsgBox "You must be tired! Nice work. Game over."

Exit Sub

End If

'put down victim

Cells(Sheets("Tracker").Cells(6, 9).Value, Sheets("Tracker").Cells(6, 8).Value).Select

ActiveCell.Interior.ColorIndex = 3

'put down lifeguard

Cells(Sheets("Tracker").Cells(5, 9).Value, Sheets("Tracker").Cells(5, 8).Value).Select

Cells.Locked = True

Lifeguard

ActiveSheet.Protect , AllowFormattingCells:=True

End Sub

Sub ColorBeach()

WaterDist = Sheets("Tracker").Cells(4, 4).Value

SandDist = Sheets("Tracker").Cells(5, 4).Value

For I = 1 To WaterDist

Rows(I).Interior.ColorIndex = 8 'light blue

Next

For I = I To WaterDist + SandDist

Rows(I).Interior.ColorIndex = 19 'light yellow

Next

End Sub

'makes square cells -- 0.1 cm = 3 pixels

'Source: http://blogs.msdn.com/b/excel/archive/2006/09/06/743902.aspx

'visited July 7, 2010

Sub SquareCells()

ActiveSheet.Unprotect

Dim desired As Double, looper As Integer

cm = Application.InputBox("Enter Square Length in Cm", Type:=1)

If cm = False Then Exit Sub

desired = cm \* (0.393700787401575) \* 72

Application.ScreenUpdating = False

For looper = 1 To 200

ActiveSheet.Columns.ColumnWidth = \_

desired \* ActiveSheet.Columns.ColumnWidth / [A1].Width

ActiveSheet.Columns.RowHeight = \_

desired \* ActiveSheet.Columns.RowHeight / [A1].Height

If [A1].Height = [A1].Width Then Exit For

Next

Application.ScreenUpdating = True

End Sub

'makes lifeguard chart axes the same

'Source: Jon Peltier

'http://peltiertech.com/Excel/Charts/SquareGrid.html

'visited July 8, 2010

Sub MakePlotGridSquareOfActiveChart()

'added this line to work on my chart

ActiveSheet.ChartObjects("Chart 9").Activate

With ActiveChart.Axes(xlValue)

.MaximumScaleIsAuto = True

.MinimumScaleIsAuto = True

.MajorUnitIsAuto = True

End With

With ActiveChart.Axes(xlCategory)

.MaximumScaleIsAuto = True

.MinimumScaleIsAuto = True

.MajorUnitIsAuto = True

End With

'end my addition

MakePlotGridSquare ActiveChart, True

Cells(1, 12).Select

End Sub

'not used

Sub MakePlotGridSquareOfAllCharts()

Dim myChartObject As ChartObject

For Each myChartObject In ActiveSheet.ChartObjects

MakePlotGridSquare myChartObject.Chart

Next

End Sub

Sub MakePlotGridSquare(myChart As Chart, Optional bEquiTic As Boolean = False)

Dim plotInHt As Integer, plotInWd As Integer

Dim Ymax As Double, Ymin As Double, Ydel As Double

Dim Xmax As Double, Xmin As Double, Xdel As Double

Dim Ypix As Double, Xpix As Double

With myChart

' get plot size

With .PlotArea

plotInHt = .InsideHeight

plotInWd = .InsideWidth

End With

Do

' Get axis scale parameters and lock scales

With .Axes(xlValue)

Ymax = .MaximumScale

Ymin = .MinimumScale

Ydel = .MajorUnit

.MaximumScaleIsAuto = False

.MinimumScaleIsAuto = False

.MajorUnitIsAuto = False

End With

With .Axes(xlCategory)

Xmax = .MaximumScale

Xmin = .MinimumScale

Xdel = .MajorUnit

.MaximumScaleIsAuto = False

.MinimumScaleIsAuto = False

.MajorUnitIsAuto = False

End With

If bEquiTic Then

' Set tick spacings to same value

Xdel = WorksheetFunction.Max(Xdel, Ydel)

Ydel = Xdel

.Axes(xlCategory).MajorUnit = Xdel

.Axes(xlValue).MajorUnit = Ydel

End If

' Pixels per grid

Ypix = plotInHt \* Ydel / (Ymax - Ymin)

Xpix = plotInWd \* Xdel / (Xmax - Xmin)

' Keep plot size as is, adjust max scales

If Xpix > Ypix Then

.Axes(xlCategory).MaximumScale = plotInWd \* Xdel / Ypix + Xmin

Else

.Axes(xlValue).MaximumScale = plotInHt \* Ydel / Xpix + Ymin

End If

' Repeat if "something" else changed to distort chart axes

' Don't repeat if we're within 1%

Loop While Abs(Log(Xpix / Ypix)) > 0.01

End With

End Sub

'buttons on Testing sheet

Sub BaseGame()

Cells(4, 2).Value = 0

Cells(5, 2).Value = 5

Cells(6, 2).Value = 20

Cells(7, 2).Value = 2.5

Cells(8, 2).Value = 1

MakePlotGridSquareOfActiveChart

End Sub

Sub ClearPossibleMoves()

Range("c31:G38").ClearContents

End Sub

1. The next step from the above equation involves squaring both sides and rearranging terms, the algebra of which leads us to the equation: . [↑](#footnote-ref-1)
2. The algorithm will work more quickly if the initial x value is close to the root, but the process will work regardless. [↑](#footnote-ref-2)