Testing Optimization:

Solving the Lifeguard Problem with Discrete and Continuous Methodologies

Humberto Barreto and Ryne Weppler

Summer 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 drowning victim in the ocean. 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 illustrate how we created this game using Visual Basic for Applications and in a future paper will discuss the actual empirical results of our experiment. We then offer a discussion of potential results 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 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 $\frac{\sin\theta_1}{\sin\theta_2} = \frac{V_1}{V_2}$, where v_i is the velocity of travel in the i'th medium and θ_i 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.

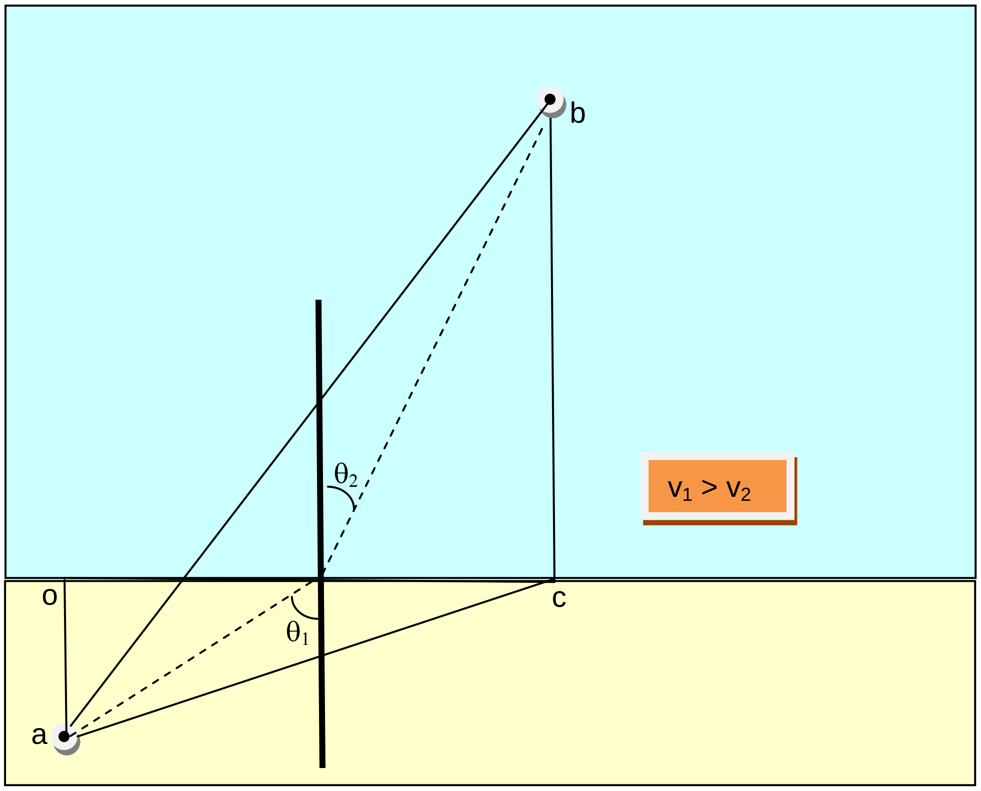


Figure 1: Understanding the Lifeguard Problem

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 $v_1 = v_2$. 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 of the problem. 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.

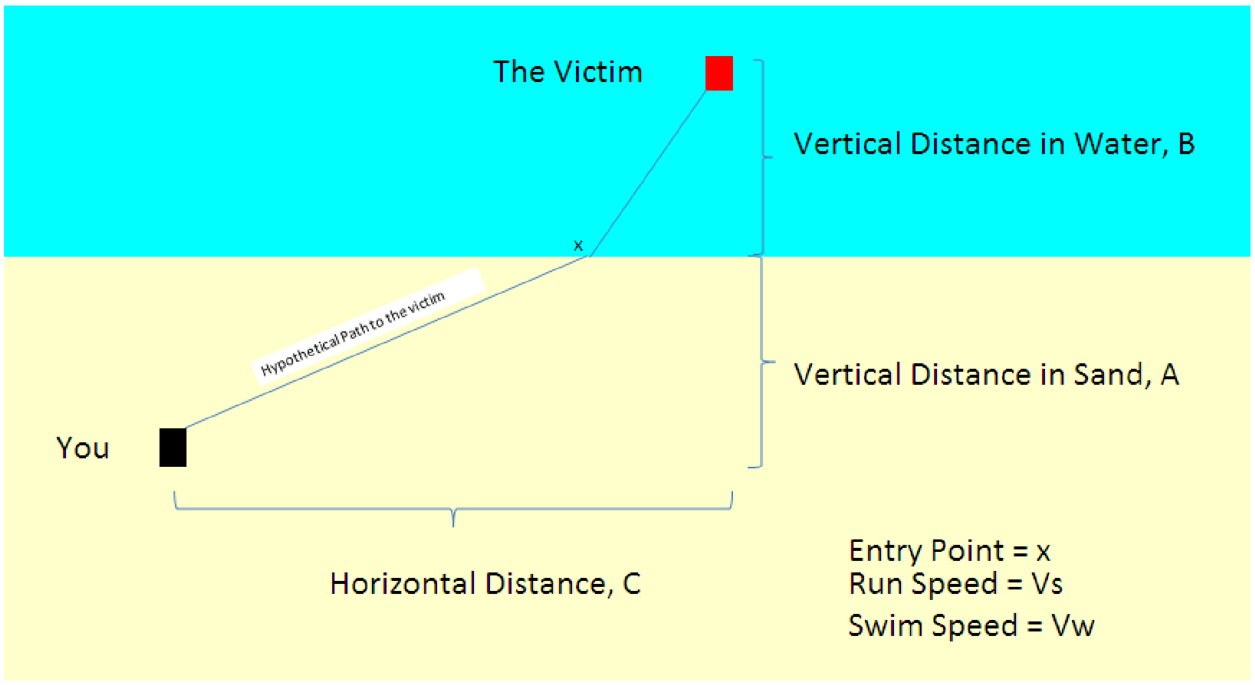

The Victim
Vertical Distance in Water, B
x
Hypothetical Path to the victim
Vertical Distance in Sand, A
You
Horizontal Distance, C
Entry Point = x
Run Speed = Vs
Swim Speed = Vw

Figure 2: The Lifeguard Problem

Time spent running on the sand can be calculated by:

$$T_S = \frac{\sqrt{a^2 + x^2}}{Vs}$$

Similarly, time spent in water can be calculated by:

$$T_W = \frac{\sqrt{(c-x)^2 + b^2}}{Vw}$$

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

$$\min_{\mathrm{x}} \mathrm{T} = \frac{\sqrt{a^2 + x^2}}{Vs} + \frac{\sqrt{(c-x)^2 + b^2}}{Vw}$$

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:

$$\frac{dT}{dX} = \frac{x}{Vs\sqrt{a^2 + x^2}} + \frac{x-c}{Vw\sqrt{(c-x)^2 + b^2}}$$

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] 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

[1] The next step from the above equation involves squaring both sides and rearranging terms, the algebra of which leads us to the equation: $x^2 Vw^2[(c-x)^2 + b^2] = (c-x)^2 Vs^2(a^2 + x^2)$.

an initial *x* value, which can be chosen arbitrarily.[2] From the initial *x*, we create a recursive sequence defined by:$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. 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.

[2] The algorithm will work more quickly if the initial x value is close to the root, but the process will work regardless.

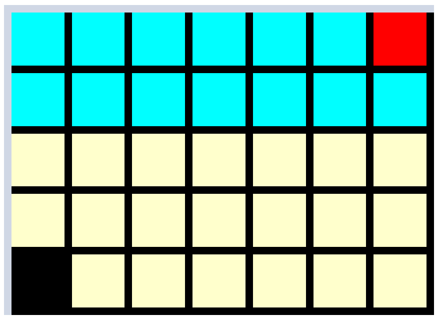


Figure 3: The Discrete Version

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 is based on the lifeguard moving 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.

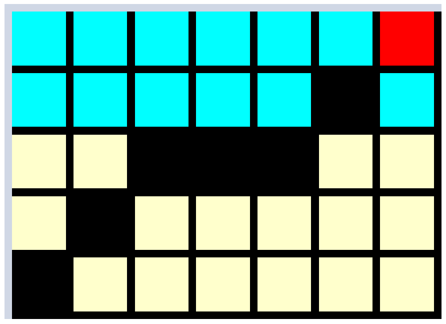


Figure 4: A Discrete Path

The optimalxdiscrete function computes the time based on least water, then uses a loop to take one step back and compare that time to the previous best time. The loop continues running as long as the total time is falling and stops once total time rises. Figure 5 shows how the function would work for a problem with a = 0, b = 5, c = 20, v_s = 1.2, v_w = 1. The optimaldiscretex function starts at x = 20, then steps back in unit decrements until 14, when total time rises and we know we have found the optimal solution at 15.

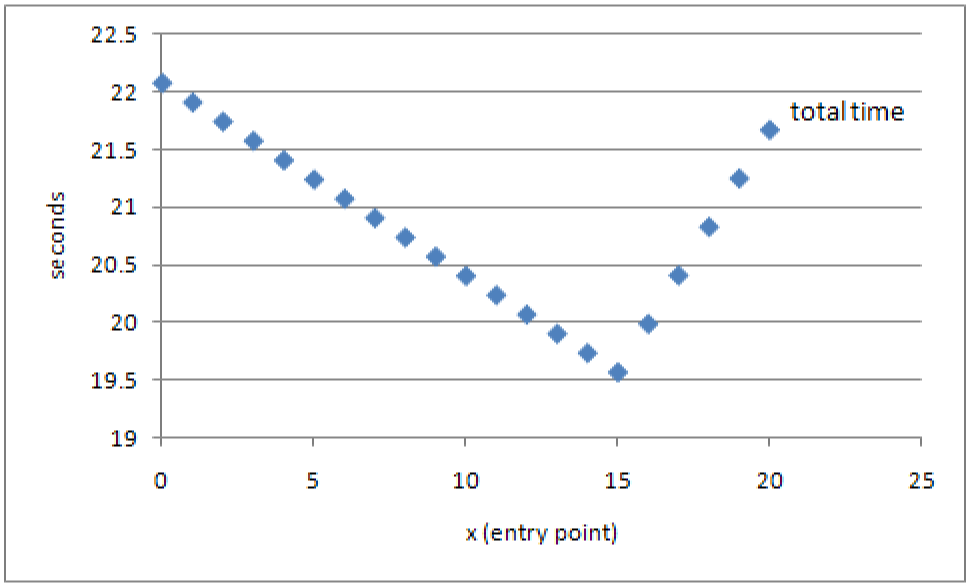


Figure 5: Total time given entry point.

As Figure 5 shows, the discretetime function allows us to graph the distribution of time as a function of entry point in the discrete case. It will also let us calculate the time it takes a subject to play the game in Excel and reach the victim without technology errors, such as mistaken clicks or system delays, by outputting the time assuming constant motion. In other words, when we calculate the participant's time to reach the victim, we will simply use the entry point and the other parameters listed above and place them in the total time equation:

$$T = \frac{\sqrt{a^2 + x^2}}{Vs} + \frac{\sqrt{(c - x)^2 + b^2}}{Vw}$$

Assuming constant speed seems reasonable, since one would expect a lifeguard to be running and swimming at all times, without hesitating or deliberating over the best path while the victim struggles to survive.

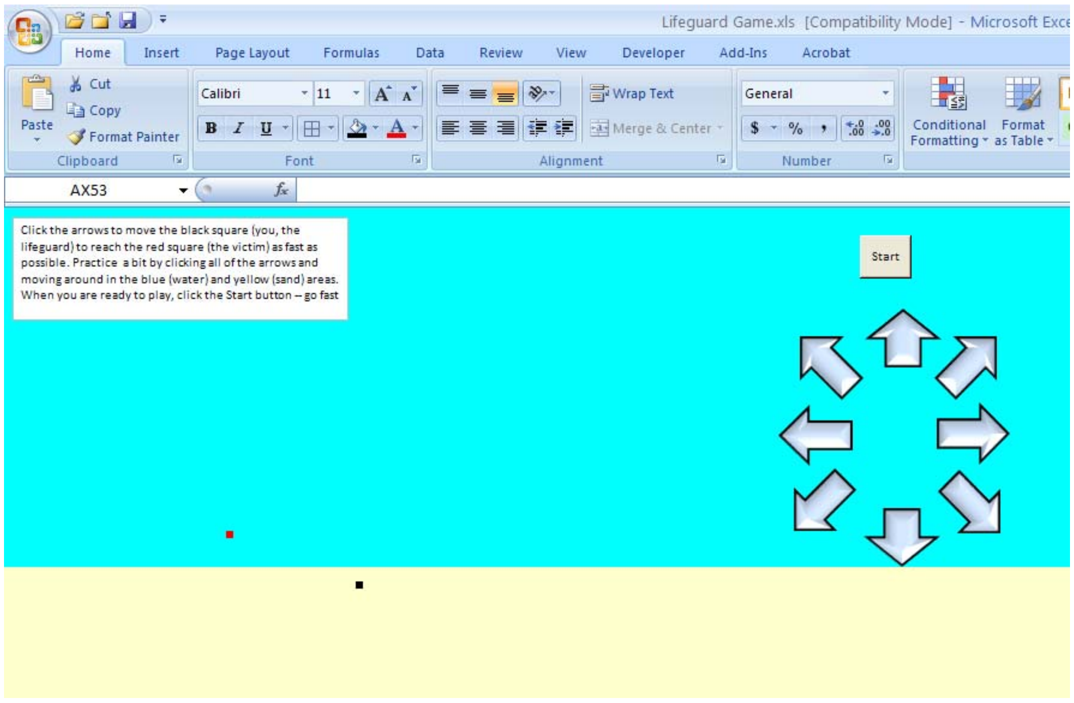
3.4.2 *Discrete Time Solutions*

Unlike the continuous version of the problem, where optimal *x* rises as running speed goes up (given $v_s > v_w$), ceteris paribus, the discrete version's optimal solution is a corner solution (least water) for $v_s/v_w > 2.4142$. 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 $v_s/v_w < 2.4142$, 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.



Click the arrows to move the black square (you, the lifeguard) to reach the red square (the victim) as fast as possible. Practice a bit by clicking all of the arrows and moving around in the blue (water) and yellow (sand) areas. When you are ready to play, click the Start button – go fast

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 (v_s and v_w), 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 twenty 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.

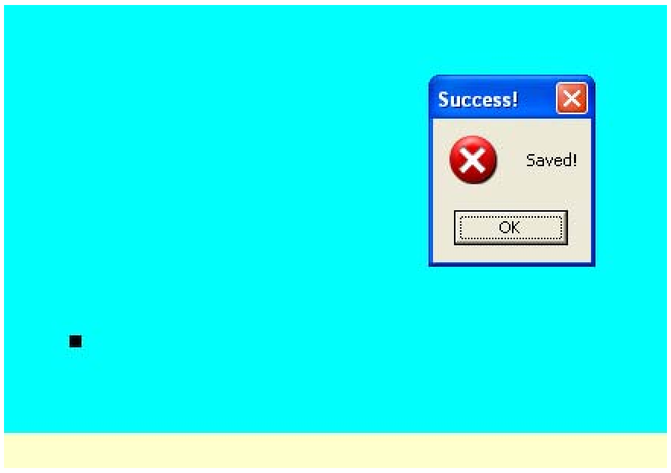


Figure 7: Lifeguard reaches the victim.

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.

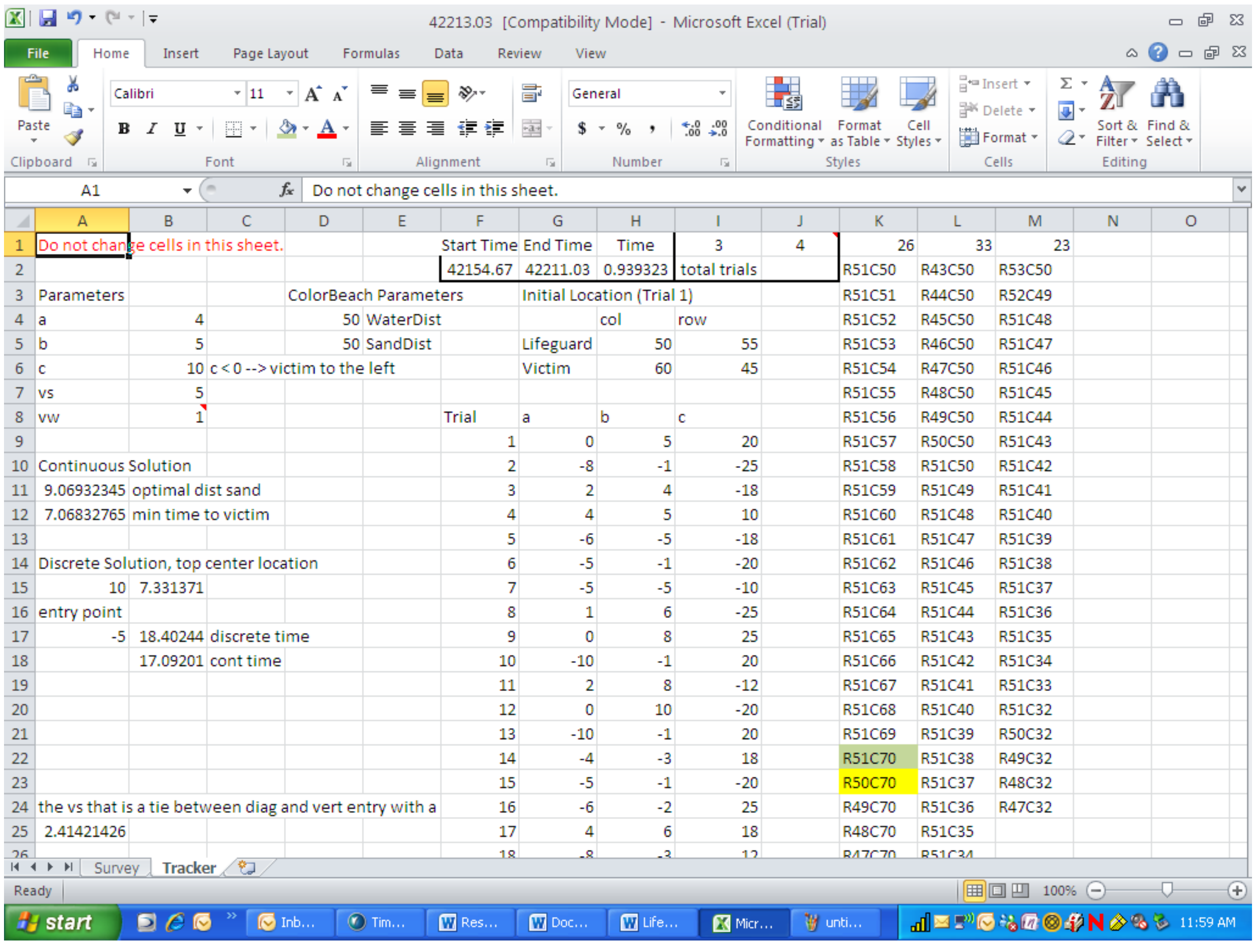
Testing for a difference in behavior depending on the order of the media is tantamount to a test of loss aversion. The discovery that many people seem to behave differently toward offers of expected gain versus expected loss is one of the earliest findings of behavioral and experimental economics and 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 entry point for a given trial. The default setup creates a beach that is fifty rows in height. Thus, if the lifeguard starts on the sand, the entry point is the point at which The "Tracker" sheet reads a row of fifty. The entry point is the column number of the previous move. Figure 8 shows a sample screenshot of the "Tracker" worksheet after the user has played through three trials of the game.



42213.03 [Compatibility Mode] - Microsoft Excel (Trial)

A1 — Do not change cells in this sheet.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Do not change cells in this sheet.					Start Time	End Time	Time	3	4	26	33	23		
2						42154.67	42211.03	0.939323	total trials		R51C50	R43C50	R53C50		
3	Parameters			ColorBeach Parameters			Initial Location (Trial 1)				R51C51	R44C50	R52C49		
4	a	4		50	WaterDist			col	row		R51C52	R45C50	R51C48		
5	b	5		50	SandDist		Lifeguard	50	55		R51C53	R46C50	R51C47		
6	c	10	c < 0 --> victim to the left				Victim	60	45		R51C54	R47C50	R51C46		
7	vs	5									R51C55	R48C50	R51C45		
8	vw	1				Trial	a	b	c		R51C56	R49C50	R51C44		
9						1	0	5	20		R51C57	R50C50	R51C43		
10	Continuous Solution					2	-8	-1	-25		R51C58	R51C50	R51C42		
11	9.06932345	optimal dist sand				3	2	4	-18		R51C59	R51C49	R51C41		
12	7.06832765	min time to victim				4	4	5	10		R51C60	R51C48	R51C40		
13						5	-6	-5	-18		R51C61	R51C47	R51C39		
14	Discrete Solution, top center location					6	-5	-1	-20		R51C62	R51C46	R51C38		
15	10	7.331371				7	-5	-5	-10		R51C63	R51C45	R51C37		
16	entry point					8	1	6	-25		R51C64	R51C44	R51C36		
17	-5	18.40244	discrete time			9	0	8	25		R51C65	R51C43	R51C35		
18		17.09201	cont time			10	-10	-1	20		R51C66	R51C42	R51C34		
19						11	2	8	-12		R51C67	R51C41	R51C33		
20						12	0	10	-20		R51C68	R51C40	R51C32		
21						13	-10	-1	20		R51C69	R51C39	R50C32		
22						14	-4	-3	18		R51C70	R51C38	R49C32		
23						15	-5	-1	-20		R50C70	R51C37	R48C32		
24	the vs that is a tie between diag and vert entry with a					16	-6	-2	25		R49C70	R51C36	R47C32		
25	2.41421426					17	4	6	18		R48C70	R51C35			
26						18	-8	-3	12		R47C70	R51C34			

Survey | Tracker

Ready

Figure 8: A sample "Tracker" sheet.

For example, in column K of Figure 8, we see the starting address for the user was row fifty-one and column fifty. The user enters the water where row fifty-one becomes row fifty, the cells highlighted in green and yellow. The column listing of the green cell reflects the entry point. Looking at the list of parameters for trial #1 (listed on row nine columns F:I), we see the beach length (c) is twenty. Furthermore, the user went from column fifty to column seventy before entering the water, thus the user travelled twenty columns, or the full width of the beach, before jumping in the water. Consequently, the entry point for this particular round is twenty. The experimenter can continue calculating results in a similar fashion.

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 twenty rounds of the game and his or her performance is recorded for the experimenter's analysis. Of particular interest to the researcher is the entry point into the water (the choice variable *x*) because, as mentioned previously, this decision is the only one which the user has control of and which determines how quickly the lifeguard reaches the victim.

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.

After playing the game and having a discussion with the participants, we dismissed the pilot group and examined the results, which appear mostly positive. The completion times varied from 349 seconds to 555 seconds, with five of the six times falling in 400 seconds or less.

It is interesting to analyze the 555 second outlier, who later reported not even noticing a difference in running and swimming speeds in the game. At this moment, it is difficult to ascertain how much of this inefficiency stemmed from an inability to optimize and how much was due to a misunderstanding of the game. In fact, in the first two trials, this participant took the path of maximum time (most water). Generally speaking, her strategy appeared to be to cover the vertical distance first, and then move horizontally to the victim, which systematically produced poor performances. The other participants seemed to pick up on the fact that being in the water was very costly. Indeed, one participant mentioned that her strategy eventually became: avoid the water.

Though a pilot test, we ran the program in the same fashion as the actual experiment to be done in the fall of 2010.

4.2.2 *The Experiment (to be done Fall 2010)*

Having successfully created an Excel implementation of the lifeguard problem, we will advertise and recruit students from DePauw University to come participate in our experiment. Individuals will come to a computer lab at the same time, where they will receive and sign an Informed Consent form. We will then give instructions on how to play the game and allow 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 is over, individuals will compete against one another, seeking to complete their set of twenty trials before everyone else. Three gift cards will be awarded probabilistically according to the performance of the participants. In other words, the top performer will have a much better chance of winning a gift card than the worst performer. After completing the game, a survey will be administered asking individuals to report their strategies, perceived math ability, handedness, and demographic information. Upon completion, participants will be compensated, debriefed, and dismissed from the lab. The process should take roughly thirty minutes. We hope to conduct multiple rounds in this fashion.

4.3 *Evaluation Metrics*

Once we have the subject's chosen entry point for a particular trial, we need to know how well the individual performs. Since we wrote the OptimalXDiscrete and DiscreteTime functions, this process becomes straightforward. Our primary method of evaluating performance is the following quotient, which we will calculate for each subject for each trial of the experiment:

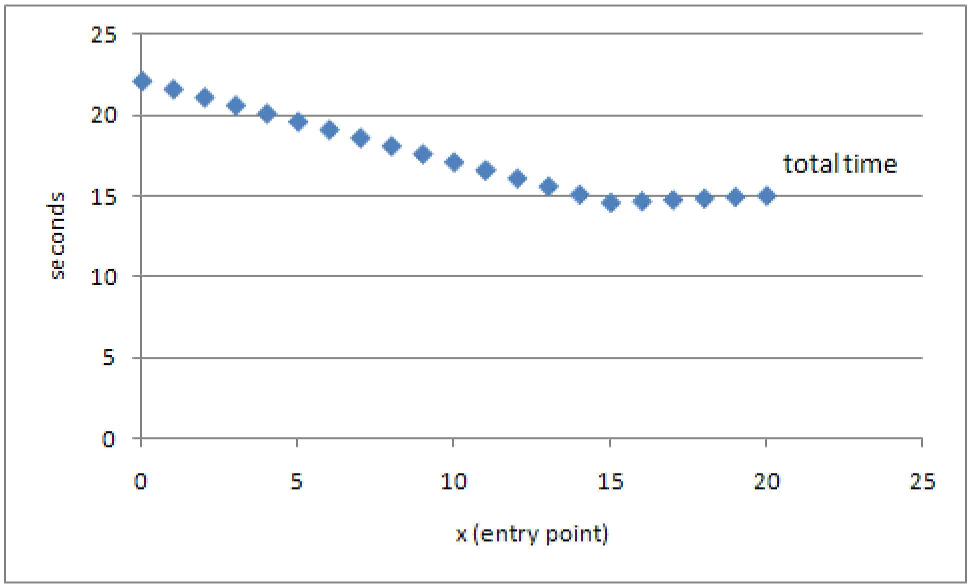
$$Percentage\ Inefficiency = \frac{User'sTime - OptimalTime}{MaxTime}$$

To calculate the user's time, we decided to find the entry point and solve for the time, rather than actually time the user. Meanwhile, participants will be 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. Thus, the first term in the numerator is the output of the DiscreteTime function with the entry point that the user chose.

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 will either be the path of least distance or least water, depending on the ratio of run speed to swim speed. In our trials the path of least distance is the slowest path because it involves more swimming, which is very time consuming relative to running. We calculate both paths with the DiscreteTime function and pick the larger of the two times for the denominator of our metric. The distribution of time as a function of entry point for a sample trial is illustrated in Figure 9, where you may see that the path which involves the most swimming (entry point 0) results in the longest time.

a	0
b	5
c	20
vs	2
vw	1



.

Figure 9: A sample trial.

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 $\frac{OptimalTime}{User'sTime}$ have a tendency to inflate as the time it takes to complete the trial increases. The minimum time to reach the 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.

To gain a better understanding of this metric, take the example above with the parameter values given in Figure 9. In this situation, the optimal entry point is x = 15 (a path that allows the user to swim diagonally to the victim)[3]. If a user chose a suboptimal path of running along the beach for thirteen cells before entering the water (x = 13), his or her resulting time, as calculated by the *discretetime* function would be 15.57107. This time would be the first term in our metric. Next, we calculate the optimal time by using the *optimalxdiscrete* function, and find the optimal time to be 14.57107. This time would be the second term in our metric, the difference of the two terms composing the numerator of percentage inefficiency: $15.57107 - 14.57107 = 1$. Finally, we compute the denominator of the metric, the resulting time of the most inefficient path. As previously mentioned, this time will either come from the least water path or the least distance path that involves the most water. In our case, the former has a resulting time of 15, while the latter has a resulting time of 22.07107. Thus, the maximum time is 22.07107. Putting these terms together, we solve for percentage inefficiency:

$$Percentage\ Inefficiency = \frac{15.57107 - 14.57107}{22.07107} = 0.0453.$$

In other words, the participant was 4.53% inefficient in this trial. Continuing in this manner, we will calculate the metric for the participants in our study this fall.

Evaluation of Results

Once we conduct the experiments and gather data, we will analyze three dimensions of the subjects' performance: initial ability, learning, and survey results.

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

[3] Note that since $v_s/v_w < 2.4142$, the optimal path is not the path of least water.

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.

Though not the primary focus of our study, we will examine the survey data from our participants to see if any trends emerge. For example, individuals who claim to be good at math may perform better at this task because they see some of the underlying mathematics that make up this problem. Alternatively, they could perform worse because they overanalyze what is meant to be an intuitive situation. With information about the participants, we can create models, which will predict how well an individual will do based on his or her background and survey information. We will achieve this through ordinary least squares regression or more sophisticated statistical techniques of analyzing time series data. It could be that only certain types of people optimize or perform well on this task, and having these statistical tools at our side will allow us to pick up on this nuance.

Discussion

We mention above 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, partially because future research can build upon what we are doing now. 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. Plus, having the victim drift with the current would make the scenario even more realistic. 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. Finally, the decision to give the user feedback on his or her performance is still undecided. After each round, we could let the participant know his or her inefficiency score, or we could leave them in the dark throughout the process. The former method could contribute more strongly to learning but could also be an unrealistic addition, as lifeguards are probably not told how inefficient they are when they save a drowning victim.

With these considerations in mind, we anxiously await the first round of our experiment. Designing the game and considering the problem fully turned out to be an impressive task, which is a credit to the intrinsically puzzling nature of the problem. We were amazed as we repeatedly found new, interesting aspects of the problem. On that same note, we still do not know what results to expect. Rather than worry about the mystery, however, we feel our research question remains an interesting one because the answer is not obvious. We would not be surprised to find evidence for or against optimization. If our results show that individuals do not optimize, what becomes of the theories and models, which presuppose this condition? If they do appear to optimize, we have only opened the door to test how widely-applicable this optimizing behavior is. Modeling other behavioral optimization problems, such as finding the cheapest gasoline, given the distance you are from the gas station and your car's gas mileage, can test how robust this behavior is. For now, however, we maintain excitement and intellectual curiosity toward the current experiment, which we will begin to run this fall.

Appendix of Visual Basic Functions

I. 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


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
```

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

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
```

II. 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
```