Developing the Optimal Algorithm for Providence Pokémon Po (BMCM Problem 2)

Stephen Leung, Timothy Sudijono, Harrison Xu

November 6, 2016

Non-Technical Summary

Everyone loves when their favorite game is hit with mathematical analysis. Consider Pokémon Po, Providence's version of the famous (or infamous) app Pokémon Go. Even for players who somehow shun the idea of fusing mathematical analysis with fun, strategies behind Pokémon Go are inherently mathematical and are linked to an interdiscplinary field known as Operations Research. This field answers questions that are salient to Pokémon Go: how does a player catch the most Pokémon while walking the shortest possible distance? How does a player catch the *rarest* Pokémon in the shortest amount of time and walking? More subtly, how does the way in which Pokémon spawn affect these strategies?

To study these questions, we simplified the game, beginning with a standard set of assumptions which we arrived at given a data set over the last 42 days. First, we looked at a model of our surroundings: we restricted the playing of the game to Providence's downtown (assumed to be a 4 mile by 4 mile region), which we modeled as a 10 by 10 grid. Assuming that the grid models the cityscape, a player can only walk along the edges in the grid; we also made assumptions including a 100% catch rate, the average walking speed, and no traffic that would slow down the player's movement.

To begin creating methods to find, on average, the highest amount of points gained in a 12 hour period, we looked at how the Pokémon were spawning. We came to several conclusions, the most important ones being that there were "hot spots" where Pokémon appeared very often, that rarer Pokémon or more common Pokémon do not spawn only in certain locations, and that the times between consecutive Pokémon spawns are not completely random.

Combining these findings with a player's intuition that hot spots are key to catching the most Pokémon, the methods in our paper are primarily fixed. We first offer two simple strategies in which the player stays strictly within the area where most Pokémon spawn, and another where the player also catches Pokémon immediately adjacent to the player on the map. Our third and more complex method allowed the player to catch any Pokémon that were accessible to the player while simultaneously trying to stay near the main hot spots.

To actually check these findings, it would be impractical to ask a large number of players to conduct these strategies and report the results. Instead, we used a computer simulation to run large numbers of these trials, known as Monte Carlo Methods, and found the average number of points gained using each of these strategies. Our findings were that the first "stay in the same spot" method yielded on average about **5** points over 12 hours, the second method yielded **20** points over 12 hours, and our third method yielded the most points, **36** over 12 hours.

It's then clear then that you should adopt our third strategy in playing this game, given the data. You should stay in the largest hot spot, and catch all the Pokémon that are in reachable distance of you; if this leads you away from the hot spot, always try to go back. However, Pokémon take precedence in this case over returning home. Don't worry: the chance that the Pokémon lead you completely away from the hot spot is low. Good luck Catching Them All, and consider Operations Research again the next time you want to win at a game!

I. Introduction

Pokémon Po is the Providence knockoff version of Niantic's record-breaking app Pokémon Go. While among the most popular games of the year, a mathematical analysis has not been conducted on the optimal method of catching Pokémon. Given data over the past 42 days about Pokémon spawns in the city, how can we develop an algorithm to collect the most number of points? Using a player's intuition, it would be better to stay relatively near areas where Pokémon are common, and stay away from places in which Pokémon are sparse. Hence, we implemented models that agreed with this intuition.

Seeing as our only way to get around is walking, we modeled the player's speed as the average walking rate, and quickly realized that the player could move around only two blocks on the city grid in 15 minutes - the time in which a Pokémon disappears. Even under the assumption that no areas of the city were congested and no traffic affected the player, mobility was still one of the key factors in developing our approach.

Although we did a literature search on related problems, we did not use any results and instead developed our own analysis. We did discover, however, that the problem is a version of Online Vehicle Routing with Time Windows, with our constraints being a 1 vehicle fleet and immediacy of time windows: they begin as soon as knowledge of the task is communicated to the player.

To actually begin developing an algorithm, we had to collect information about the Pokémon sightings, and in particular, their distributions. Our analysis follows in the assumptions section, but we provided visualizations for several salient features of Pokémon spawns: the time differences between consecutive spawns, the frequency of different locations as Pokémon spawns, the frequencies of point values for the Pokémon, and the interactions between these features.

In particular, had the Pokémon spawn locations followed a uniform density, an optimal approach would have been to stay near the center to try to minimize the distance between the player and possible Pokémon spawns. What we visualized, however, was a grid that certainly had a large number of 'hot spots,' a feature that influenced our model significantly. We then used our conclusions from the data over the 42-day sample set as representative of Pokémon sightings in the future, and base our model off these findings.

In this report, we developed three increasingly effective approaches to achieve maximum Pokémon point value, in terms of expectation: if we implement our strategy over a large number of 12-hour games, how many points will the player net on average? To model this expectation directly, we utilized a robust Monte Carlo implementation to simulate our algorithm over a large number of games.

II. Assumptions

To begin with our model, we establish necessary terminology that we will use to examine the provided data.

Definition 1: *Let the* **map** *or* **grid** *be a* 10×10 *matrix encoding the map of the city; that is, if a Pokémon spawns at the location* $(1,1)$*, we consider its spawn location to be the entry* $(1,1)$ *of the matrix. The player can only inhabit one of these grid squares at a time and can only move to a vertically or horizontally adjacent entry in the matrix; no diagonal movement is permitted.*

To analyze the set of data, we define terminology relating the sightings of Pokémon.

Definition 2: *Let* S *be the set of all Pokémon sightings in the data. Regard each* $s \in S$ *as a vector in* $\mathbb{R}^4$ *containing four pieces of information: the* x*-coordinate of the Pokémon's spawn location, the* y*-coordinate, the point value of this Pokémon, and the Pokémon's spawn time.*

For each $s \in S$, consider the following functions:

- $Loc(s)$, which outputs the location of the Pokémon in planar (x, y) coordinate form.
- $Val(s)$, which evaluates the Pokémon's point value,
- $Time(s)$, which returns the time at which the Pokémon spawned,

We assume that the player's walking speed in the game is a typical walking speed of 3.2 miles per hour. Since the grid is 4 miles by 4 miles, the player traverses roughly one grid edge of length 0.4 miles every 7.5 minutes. Given that the player is located in a downtown area with buildings and residents, the player is not allowed to cut diagonally through the grid. We therefore analyze the space using a taxicab metric.

Pokémon Spawn Times

To model the average spawn time per Pokémon, we analyze the differences in spawn times between consecutive Pokémon given the data for the problem. A visualization of the set of time differences gives the obvious similarity to a normal distribution, whose parameters we estimate in the following calculations. Denote Random Variable X to describe the spawn times of consecutive Pokémon. The bias-corrected estimator for Variance gives

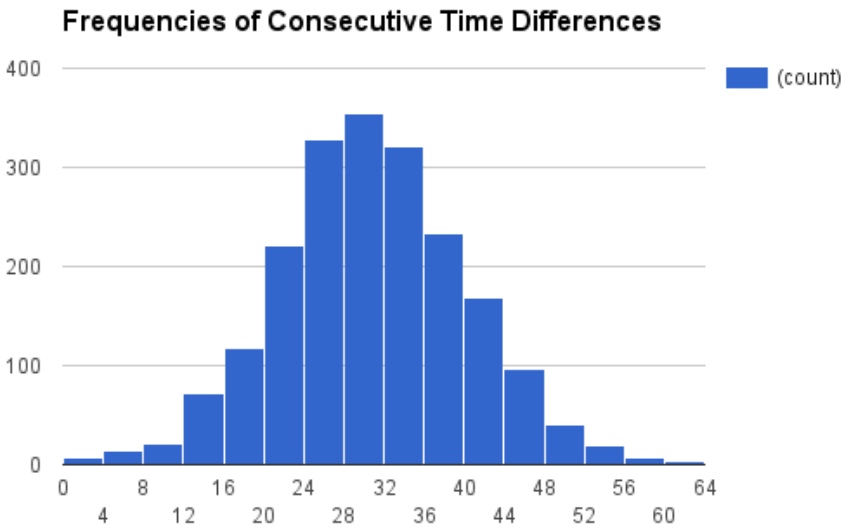
$$\hat{Var}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = 84.64$$

And the common unbiased estimator for the mean has

$$\hat{E}(X) = \frac{1}{n}\sum_{i=1}^{n} x_i = 30.27$$

With which we estimate the distribution of spawn times to be $N(30.27, 9.2)$ (a Normal with $\mu = 30.27$ and $\sigma = 9.2$).

A Kolmogorov-Smirnov goodness-of-fit test affirms this conclusion with 0.95 confidence (See appendix B), and our visualization of the data is given below. A discussion concerning the correctness of this test is included in the appendix.

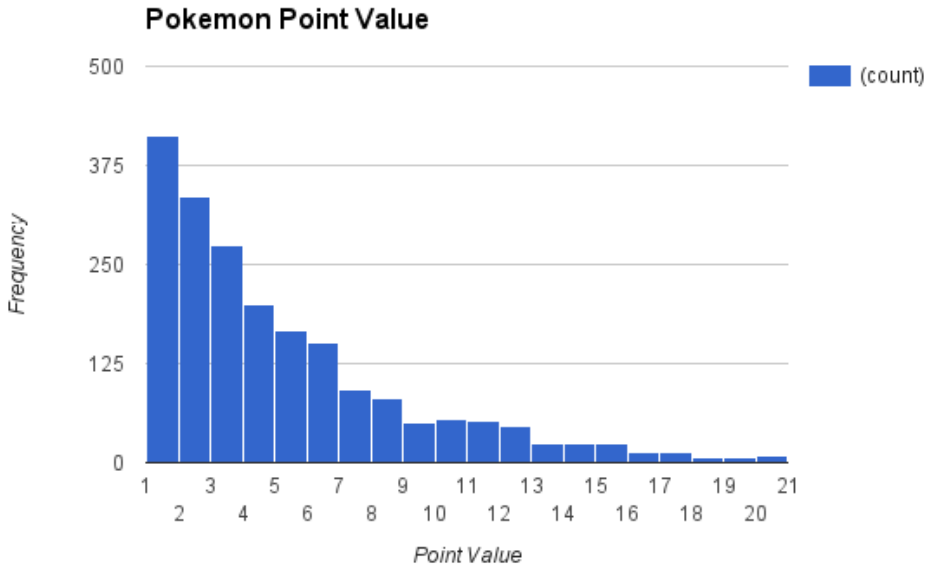


Distribution of spawned Pokémon values

It is immediately evident that there exists a trend correlating Pokémon rarity with their respective values. After visualizing the given data, as displayed below, it is reasonable to conclude that the distribution of point values is exponential. Letting Random Variable Y denote the values of spawned Pokémon, we estimate rate parameter λ of the exponential distribution as follows:

$$\hat{E}(Y) = \frac{1}{n}\sum_{j=1}^{n} y_i = 4.67 \implies \lambda \approx \frac{1}{\hat{E}(Y)} = 0.21411$$

This estimation of Y is affirmed through the Lilliefors test for exponentiality under 0.95 confidence, given by an implementation in Matlab. Our visualization is seen below, and a discussion of the Lilliefors test is given in the appendix.



Given the above estimations and assumptions, it becomes critical to correlate grid locations with Pokémon spawn frequencies and, more importantly, with *valuable* Pokémon spawn frequencies. A visualization of Pokémon spawn distributions is given below, in a 2-D histogram - a frequency *heatmap* of sorts.

To properly distinguish between spawn frequencies of differently valued Pokémon, three additional histograms, each corresponding to three level strata (1-5, 6-12, 13-20, respectively), are visualized below. We are concerned primarily with the potential correlation between Pokémon *values* and the *locations* in which they spawn - is it the case that higher-valued Pokémon prefer some locations to others? Any indication of such preferences may drastically alter the model.

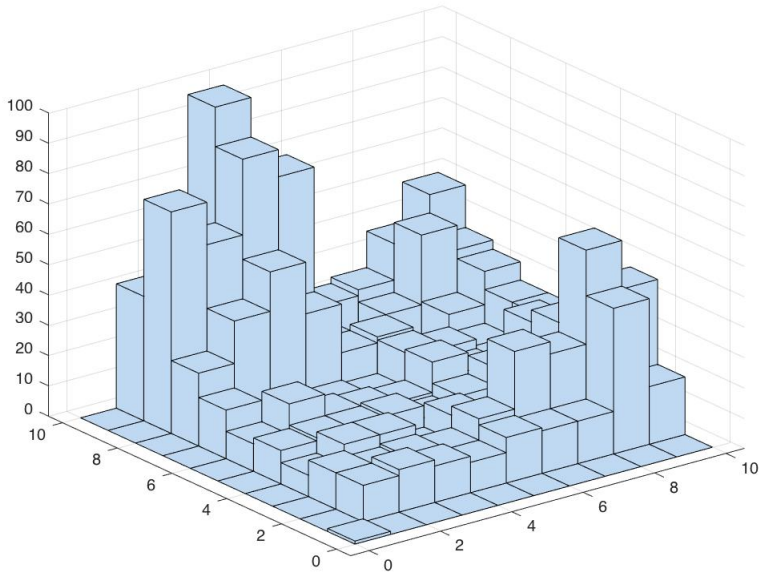


Figure 1: Location frequency of all Pokémon

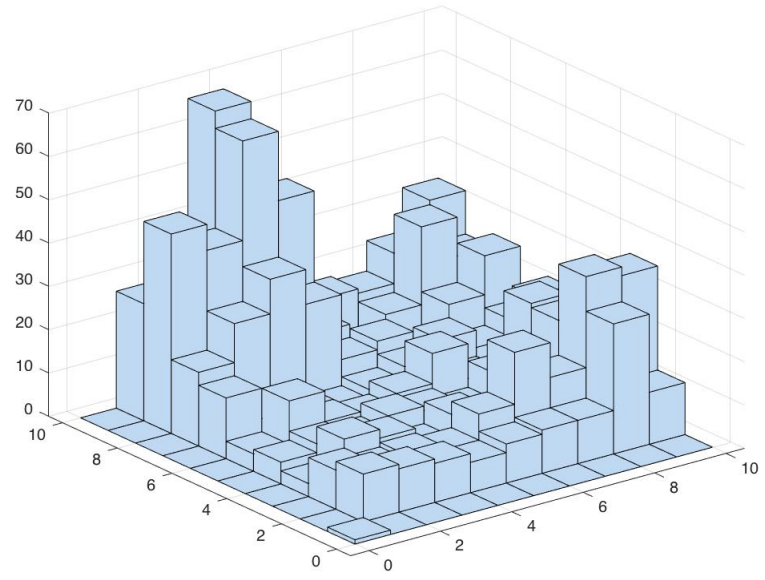


Figure 2: Location frequency of all low-value Pokémon (levels 1-5)

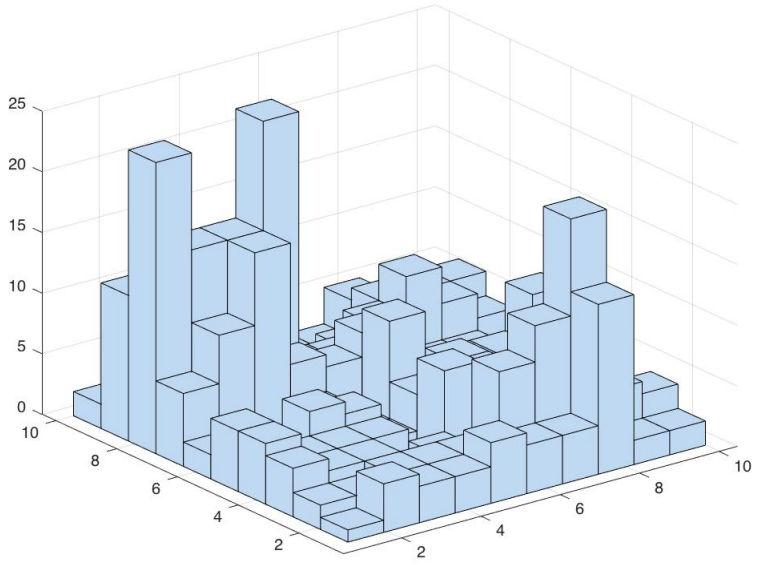


Figure 2: Location frequency of all mid-value Pokémon (levels 6-12)

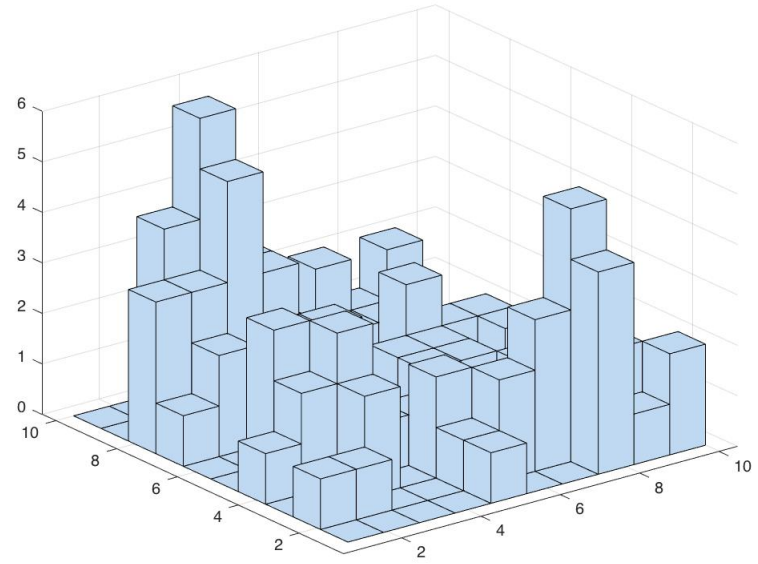


Figure 2: Location frequency of all high-value Pokémon (levels 13-20)

Empirical data suggests, quite plausibly, that there is little to no significant correlation between spawn location preference and the point value of the Pokémon. In other words, frequency distributions of Pokémon are largely unaffected by their point value. We will work henceforth with this assumption - that Pokémon point values and spawn locations are independent of each other.

We can summarize our assumptions in the following list:

- **Assumption I** : The player moves at average walking speed, translating to 1 grid edge in 7.5 minutes without variation - it is difficult to maintain anything higher than walking speed consistently for 12 hours.
- **Assumption II** : The consecutive spawn times for Pokémon are distributed according to $N(30.27, 9.2^2)$.
- **Assumption III** : The point values for Pokémon are distributed according to Exp(4.67), where 4.67 is the mean.
- **Assumption IV** : The spawn times for Pokémon and their point values are independent.
- **Assumption V** : The locations for Pokémon spawns and their point values are also independent.
- **Assumption VI** : We catch every Pokémon we encounter.
- **Assumption VII** : The distribution of spawn locations equals the distribution observed in the sample data.

Some immediate limitations of these assumptions are as follows: it may not be true that we catch every Pokémon we encounter: higher value Pokémon should normally have lower catch rates. Further, the locations for Pokémon spawns and their point values are not necessarily independent; we offer an alternate solution sketch given that this assumption is false. We also assumed that no traffic - motor or pedestrian - affects the player, and that Pokémon only disappear given the 15 minute time limit, and not by other means (such as other players catching it).

III. Three approaches, and Monte-Carlo estimates

The following section details three approaches - one static, and the others dynamic - to the point-maximization problem. The most important factor in developing these approaches is the *mobility* of the player. Under our assumption that the player takes 7.5 minutes to move one tile, the player can only reach Pokémon a distance of two tiles away from the player before it disappears. Intuitively, then, restricting the player to the grid locations of highest Pokémon point density will enable the player to collect a comparatively high percentage of Pokémon, and a high percentage of the total points, that spawn within this area. Thus, our methods are mostly fixed - the player will stay in regions of high value and ignore Pokémon far away from these regions for several reasons: the first and most obvious being that Pokémon cannot be reached past two grid edges, and the second being that moving away from the regions of high value will result in more valuable missed opportunities. Again, we are extremely limited by traveling speed.

To quantify areas of high value, we used our location frequency histogram for spawn locations to recognize that there was a clear region where Pokémon spawned the most, in the region surrounding the grid box $(3, 8)$. It's important to note **Assumption V**, that the Pokémon's spawn location and point value are independent; we do not have to analyze if the high value Pokémon are concentrated away from the most frequent spawn regions.

As a discussion, we introduce three models of increasing efficacy based on weaknesses from

the previous model. We begin with the naive strategy, restricting oneself to the square of highest frequency and collecting all the Pokémon that appear at the grid location. We then progress to strategies that offer more mobility - strategies based on the observation that the expected values of missed opportunities are low.

A. Approach 0

Clearly, the most naive method in our family of fixed strategies is the "stand at the optimal square strategy" where player stands on the grid square with highest frequency of observations and collects all points that fall direcly in the square. To model this, we implemented a Monte Carlo simulation playing the game $n = 1000$ times. To model a game, we first calculated the number of Pokémon that would spawn in a game; we drew samples from $N(30.27, 9.2^2)$ until the sum of these samples was over 720 minutes, or the 12 hour time period was over. For each of these Pokémon, we assign a point value distributed according to the exponential distribution we had established. We then sum up the total values of points of every Pokémon that lands in our optimal square. To model a Pokémon spawning in the optimal square, we check if a sample from $U([0, 1])$ is less than the frequency value at $(3, 8)$.

For each game we play, we define p_n as the number of points scored on game n. We then define our simple Monte Carlo Estimator to be

$$\frac{1}{n}\sum_{i=1}^{n} p_n,$$

where $n = 1000$. Our code is attached in the appendix, and we get a result of **4.86**. Immediately, we can suggest multiple features for improvement: why should the player constrain himself to the most optimal square O? If Pokémon spawn in adjacent squares, which happens with relatively high frequency, the player should go and collect the Pokémon. This leads us to Approach 1.

B. Approach 1

In Approach 1, the player moves between an optimal group of five squares; we can describe this group as O itself and the squares of taxicab distance 1 unit away from O. Whenever a Pokémon appears in this cross-shaped region, the player will always be within a distance of 2 from the Pokémon and thus will always have the the option of moving to catch this Pokémon. In this model, the player moves to catch the Pokémon and then immediately returns to the starting square; any other Pokémon outside the cross region will be ignored. An important factor that we can remove from our analysis is the chance that a Pokémon will appear within the cross before the player has moved back to O. We show that this probability is negligible, allowing us to simplify our Monte Carlo simulation without compromising its accuracy.

Once a Pokémon appears in an adjacent tile, the player takes 7.5 minutes to collect it, and 7.5 minutes to come back. For the player to miss a Pokémon in one of the tiles adjacent to O, that

new Pokémon must spawn exactly when the player is collecting the old Pokémon in the opposite tile. This is because Pokémon only last 15 minutes, and the time it takes to move two grid edges by the player is exactly 15 minutes. The configuration is shown below:

Player, Old Poké	O	New Poké

Notice that for this to happen, the spawn time between these two Pokémon must be less than 15. Then, the probability is given by

$$\mathbb{P}(\text{ Consecutive spawn time is less than 7.5 }) \cdot \mathbb{P}(\text{ new Pokémon spawns in opposite square}).$$

This is conditional on which square the player is adjacent to O, but each frequency value for the tiles adjacent to O are roughly 3%. Then we can calculate this probability to be

$$\mathbb{P}(X \leq 7.5) \cdot .03,$$

where $X \sim N(30.27, 9.2^2)$. Translating these into z-scores, we see that the probability is now equal to

$$\mathbb{P}(Z \leq \frac{30.27 - 7.5}{9.2}) \cdot .03 = .0067 \cdot .03 \approx 2.01 \times 10^{-4},$$

which is a negligible event.

To show that the case where another Pokémon spawns within the cross before the player has moved back to O is negligible, we repeat the above argument. The probability in this case is given by

$$\mathbb{P}(\text{ Consecutive spawn time is less than 15 }) \cdot \mathbb{P}(\text{ new Pokémon spawns in region}),$$

and we can then calculate this probability to be

$$\mathbb{P}(Z \leq \frac{30.27 - 15}{9.2}) \cdot \sum \text{frequencies of squares in this region} = .0559 \cdot .1655 = .00925,$$

which is also negligible with roughly a 1% chance.

We can model the expected number of points gained by using a similar Monte Carlo analysis as above; all that differs in Approach 1 is the region we are taking into consideration. Nothing else changes, since we regard the chance of another Pokémon spawning within the cross as negligible. Our code is attached in the appendix, and we have a result of **20.0**.

C. Approach 2

Even with the improvements of Approach 1 over Approach 0, there are still inherent flaws to consider:
Even if we do not remain within a stationary square, we still constrict ourselves to a limited search

region. As lucrative as this region may be, this method cannot possibly be optimal - we potentially miss very high-valued Pokémon which may only slightly exceed our search boundaries. How do we rectify this?

Approach 2 continues to iterate on a proven region of success - we do not destroy the foundation we have created. We construct a **range - limited search algorithm** that has no restrictions beyond those imposed by the 10x10 playing grid. This algorithm recognizes the "hotspots" introduced in prior models as lucrative, and will capitalize on this fact whilst actively searching for new targets potentially beyond the hotspot. Critical to the success of this algorithm is that Pokémon are not static, spawn randomly, and have a deterministic expiration timer of 15 minutes before disappearing. We do not want to path towards targets that are beyond feasible walking distance.

Our algorithm is initialized at the optimal grid point $(3, 8)$ and performs actions determined by discrete time steps of 7.5 minutes. In each iteration, it searches for potential lucrative targets within striking distance. If these do not exist, or are not within distance, we begin pathing back towards the high-frequency spawn zones while searching for new targets after each action. If such Pokémon are within reach, we path towards them. If, under the rare circumstance that a higher-valued Pokémon spawns that is *simultaneously* within reach, we "forget" the prior Pokémon and immediately retarget to the higher-valued Pokémon.

As before, we rely on the normality of Pokémon spawn frequencies, and the exponentiality of their point distributions, which the Kolmogorov-Smirnov and Lillefors tests (see Appendix B) affirm within 0.95 confidence. Under such assumptions, our Monte Carlo estimates over 10,000 trials give a much improved estimate of **35.9544** points accrued per game.

There is an immediate and obvious optimization to the algorithm that can be made - we do not consider the obvious "value" disparities between each grid point. Even intuitively speaking, it is preferred to stray to the middle than near the edges, as we have greater pathing freedom, **even if** Pokémon spawn at marginally higher frequencies near the edges. It would be preferable, when considering pathing options (moving towards Pokémon targets or back to preferred hotspots) of equal distance, to path through an area of where Pokémon spawn frequencies are maximized on grids within reachable distance, **at every step** of the path. If we were to conduct the factorial-time generation of all possible paths, and to take only the path with maximized "value", our algorithm could potentially see an increase in average points per game. However, this is at the severe cost of a massive increase in computation time. Furthermore, it is not necessarily the case that our [3,8] centered hotspot is ideal regardless of our current location - it may be the case that we are so far from [3,8] that it is optimal to path towards a more accessible hotspot. That said, we draw attention to the fact that spawn frequencies beyond our initial hotspot area diminish extremely quickly, such that it is unlikely to path very far beyond $(3, 8)$ before returning. The net gains of such a modification would be marginal at best.

D. *Results*

To reiterate, we developed models of increasing strength and complexity, and tested them with Monte Carlo Methods attached in Appendices $\mathbf{C}, \mathbf{D}$. Approach 0 returned an expected **4.86** points, Approach 1 returned **20.0**, and Approach 2 gave the best result of **35.95** points. It is our recommendation that the player adopt the strategy described in Approach 2.

IV. Model Analysis

A. *Advantages*

The greatest strength of our approach is the reliability of our results, along with the simplicity of our method for collecting the most amount of points. The strategies described in Approach $0, 1, 2$ are intuitive and simple to implement, as clearly seen. Our model, however, gives highly accurate results with respect to our assumptions. Our use of Monte Carlo to simulate several hundred runs of Pokémon Po gives an accurate expected number of points over a twelve hour period.

B. *Drawbacks*

The greatest drawback of our approach is specificity, and its emphasis on heuristics. For example, the approach would likely be less than optimal had the Pokémon spawn distribution been uniform. Because our method relies on the existence of localized 'hot spots,' if the Pokémon were generated uniformly on the grid, there would not have been a spot of maximal frequency. The model's reliance on heuristics also hurts its efficacy in more general situations. Suppose that there were two separate peaks of high frequency: how should the model decide which peak to choose? Our model could have been improved by giving some algorithm to find the optimal starting point, or optimal square in terms of frequency, given any probability distribution function for the Pokémon.

Another aspect that could have been improved was rigor; in the literature surrounding a related problem, Online Vehicle Routing, the term "competitive ratio" is used to describe the ratio between the worst case solution and the expected case. We could have offered bounds for the competitive ratio, in an attempt to prove the efficacy of our method in relation to established results or other methods.

With respect to other assumptions in the model, the key **Assumption V** postulated that Pokémon locations and their point values were uncorrelated. While this allowed us to model each strategy using Monte Carlo methods, it might not necessarily be the most accurate assumption, given the histograms for low, middle, and high valued Pokémon. For example, the high valued Pokémon in particular seem to be a lot more uniformly distributed than the low and middle valued Pokémon. Even though these effects are small, they could affect the veracity of our results significantly.

Below, we briefly explore a method that takes into account correlation between these two factors. Mimicking the idea of areas of high value on the grid, we defined the concept of a Point Density

Matrix:

Definition 3: *A **Point Density Matrix** of the set of Pokémon values is the* 10×10 *grid of real numbers, where the entry* (i, j) *is defined as*

$$\frac{\sum_{s \in S} Val(s) \mathbb{1}\big(Loc(s) = (i, j)\big)}{\sum_{s \in S} Val(s)},$$

or the sum of the point values of all Pokémon appearing in that square, divided by the total number of points observed.

We can interpret the Point Density Matrix as the proportion of total point values concentrated at each square on the grid; intuitively, this assigns a measure of how 'profitable' each square is. Moreover, the matrix is effective because it captures the interaction of the distribution of the point values and the distribution of locations of the Pokémon: independence is not assumed between these two factors.

We reinterpreted the above approaches using the Point Density Matrix. Out of the total points observed in a 12 hour period, we can use the Point Density Matrix to find the proportion of total points observed in a given square of the grid, or more generally, a given region on the grid. (Reiterating, we have defined the total points observed as $\sum_{s \in S} Val(s)$). To clarify the methods used in the reinterpreted models, consider Approach 0 again.

C. Reinterpreted Approach 0

Recall that the strategy in Approach 0 was to remain stationary on the square with most frequency, which in this case was $O = (3, 8)$. Standing at O would net the total number of points of all Pokémon that spawned there; because we had assumed that point values of Pokémon were independent from their locations, there would be no advantage of finding another square in which had a higher chance of spawning more valuable Pokémon.

In the reinterpreted approach, we calculated the total number of points observed in a 12 hour period, on average. This is given by

$$\mathbb{E}[X] \cdot \mathbb{E}[Y]$$

where X is the random variable representing number of Pokémon seen in the 12 period, and Y is the random variable for the point value of a Pokémon. Note that we have this formula under **Assumption III**, that the times at which Pokémon spawn are independent from their point values.

Calculating $E[Y]$ is a simple task: under the assumption that Y follows an exponential distribution with some given parameters, we know that $E[Y] = 4.67$ is the mean of this distribution. Calculating $E[X]$ is a more interesting task, however: we must find the expected number of samples needed, drawn from $N(\mu, \sigma)$, such that their sum is greater than 720 minutes, or 12 hours. We

defer this work to the appendix, and use the approximation $E[X] = 23.3$ here. Combining these two facts gives us an expected $23.3 \cdot 4.67 = 108.811$ points seen in a 12 hour period; now multiplying by the entry $(3, 8)$ on the point density matrix will give the expected number of points that occur on O, the optimal square. This point density value is observed to be 3.974%, and so the average number of points we will collect in 12 hours under Approach 0 is

$$108.811 \cdot .03974 = \boxed{4.32}.$$

An identical analysis with Approach 1 gives us 108.811 points seen throughout the period, and the percentage of points that will occur in the cross region defined in the Approaches Section is seen to be 16.55%, the sum of the entries $(3, 8), (3, 7), (3, 9), (2, 7), (2, 8)$ in the point density matrix. Another quick calculation gives us

$$108.811 \cdot 16.55 = \boxed{18.0}.$$

Note that both of these calculations give lower estimates for expected scores than our Monte Carlo methods.

V. Conclusion

When faced with the daunting task of finding the optimal algorithm for catching Pokémon, it is essential to first establish a list of assumptions as a sort of anchor. Obviously, we must first assume that the player will catch every Pokémon he encounters - hopefully those Dratinis don't run away! The first assumption was one of common sense, that a player whose only method of transportation is walking will not be able to consistently maintain speeds above 3.2 miles per hour, or about 7.5 minutes per edge. Using the data given, we determined that the spawn times for the Pokémon follow a $N(30.27, 9.2^2)$ normal distribution and that the point values of the Pokémon follow a Exp(4.67) distribution where 4.67 is the mean, using statistical tests in Appendix B. We hypothesized that the spawn times and the point values are independent and that the spawn locations and the point values are independent. Finally, we assume that the actual distribution of spawn locations follows the distribution given to us in the data. From this set of assumptions, we are able to generate three working approaches of increasing strength and complexity.

The first approach, an approach of such naivete that we call it Approach 0, is mindless but intuitive. Using the location frequency of the Pokémon spawns in the sample data, we determine that the square of highest frequency is the grid box $(3, 8)$. Thus, the player stands directly inside this box and refuses to move, catching only the Pokémon that spawn exactly where he sits. We are able to simulate 12 hours of play by drawing from our hypothesized distributions for spawn time and strength and by using the location frequency in the sample data to calculate the probability of any given Pokémon spawning in the grid box $(3, 8)$. Then, we use a Monte Carlo estimator on 1000 sample runs to reach our estimated point value, a measly 4.86. Not too bad, considering no

physical movement is required at all.

In our next approach, the player choose an optimal group of five squares (in the shape of a cross) and moves between these five squares. Using our assumption of walking speed, the player will always be at a maximum 15 minutes away from any other square in the grid. In this approach, the player has the mental capacity to move to another square in the grid to catch any Pokémon that spawn there, but then immediately moves back to his original position in the center of the cross. Now that the player is engaging in physical movement, we must take into account any Pokémon that he may miss. However, we calculate that the probability of there emerging a Pokémon inside the cross that the player is unable to catch is so low that the effects on the expected value will be negligible; we can then avoid altogether the cases where the player has to reroute to catch another Pokémon within the cross region. This makes our algorithms much simpler: using the Monte Carlo analysis we used in our previous approach, we reached an estimated point value of 20.0.

Our third and most lucrative approach follows a sort of range-limited search algorithm. We gift our Pokémon player with the power of choice, the ability to decide if a Pokémon is within walking range (two grid lengths). If there are no Pokémon within walking range, the player paths towards the high frequency hotspots. If there exist Pokémon within walking range, like any normal functioning Pokémon player, the player will go catch the Pokémon. Using the same Monte Carlo analysis as the other two approaches, this approach achieves an estimated point value of 35.954.

However, the increasing complexity of this approach leads to many drawbacks. With this increased movement, pathing becomes a major issue. There are multiple ways to get from one grid point to another, and every grid point has a different frequency of Pokémon spawning. Although it is possible to generate all possible paths and determine the optimal one (based on our location frequency analysis), this would only marginally increase the expected value of points per game at the large cost of both the runtime of the algorithm and our time coding the algorithm. There are also many other possibilities, such as a trail of Pokémon leading the player away from the high frequency area to the point where it is optimal to stay at another high frequency area. These cases are extremely unlikely, given the low frequency of Pokémon spawn and the superiority of the starting grid square (in terms of spawn frequency). Indeed, we can use an argument similar to the one we used in our section on Approach 1. We must accept that any modifications to our algorithm would create only marginal changes in expectation, and thus are not worth implementing.

Our methods are simple, intuitive, and reliable. However, they lack in rigor and are based heavily on our assumptions. We fail to use any sort of competitive ratio, something that is essential in similar problems. Furthermore, our approach relies heavily on the existence of hot spots, so much so that without a designated best hot spot, it is difficult for our approaches to work effectively. Although we previously assumed that point value and location were independent, there is not enough evidence to solidify our claim. Thus, we provide an alternate approach which assumes otherwise. We define a Point Density Matrix as the proportion of total point values concentrated on each square of the grid. This creates an intrinsic value of each grid square that is slightly different from the location frequency. Using this point density matrix to reevaluate our approaches, we

estimate the total number of points observed in a 12 hour period, and use the Point Density Matrix to find how many points are distributed within the optimal square and the cross region of Methods 0 and 1; in this case, we get slightly lower estimates than those of the Monte Carlo approach.

Given the unpredictable nature of Pokémon spawns, it is difficult to come up with an optimal approach of catching Pokémon. However, our approach effectively uses a range-limited search algorithm to locate the player in a high frequency area while simultaneously allowing him to catch any Pokémon that appear nearby. Although this approach is not void of weaknesses, it is clearly the optimal algorithm given the unpredictability of Pokémon spawns.

Appendix A.

Here we attach the frequency and point density matrices (each given in percentages) that we used for our analysis.

Point Density Matrix:

0.3642	0.6855	0.3535	0.6748	0.4820	0.5677	1.1997	3.6097	1.6602	0.3963
0.7177	0.5356	0.4713	0.8569	1.0497	0.4499	1.8530	2.9027	2.4636	0.8783
0.5784	0.4070	0.8783	0.8569	0.6427	0.3642	2.2708	3.9739	3.5775	1.4781
0.4070	0.4392	0.1821	0.6213	0.6962	0.6320	1.5853	3.8346	1.5103	1.1461
0.9640	0.8033	0.8355	0.3856	0.3963	0.2464	0.9854	0.7819	0.1821	0.7498
0.5463	1.6388	0.5356	0.8141	0.6748	0.9854	0.9212	0.4820	0.3213	0.6427
0.7605	1.9602	0.3213	1.0711	0.8462	0.7284	0.4713	0.8355	0.7819	0.7177
2.7742	3.7168	1.6067	1.3389	0.4177	0.8248	0.8355	2.0994	0.9747	0.9212
0.5141	1.4996	1.0604	0.7177	0.4606	0.5677	0.6105	1.1461	0.7069	0.8141
0.7605	0.5677	0.7498	0.1285	0.6105	0.8033	0.6855	0.6213	0.6641	0.1607

Frequency map:

0.6003	0.6003	0.3502	0.5503	0.4502	0.8004	1.2006	3.6518	1.6508	0.4002
0.7004	0.5003	0.7004	0.6003	0.9505	0.4502	1.9010	2.9515	2.0010	0.8004
0.6003	0.5503	0.7004	0.6003	0.6003	0.4502	2.5513	4.2021	3.1016	1.7509
0.5003	0.5003	0.3002	0.6503	0.5503	0.4502	1.7009	3.7519	1.6508	1.0505
0.7504	0.8504	0.7004	0.4502	0.5003	0.4002	0.9005	0.9005	0.3502	0.5503
0.7004	1.8009	0.5003	0.6503	1.0005	0.9505	1.0005	0.4502	0.5003	0.6003
0.7004	1.6008	0.4002	0.9505	0.7004	0.8504	0.4002	1.0005	0.6003	0.6003
2.4012	3.1516	1.7509	1.5508	0.4002	0.7504	1.0505	2.1511	1.0505	0.7504
0.4002	1.8509	0.9505	0.9505	0.6503	0.5003	0.8504	1.2506	0.6003	0.8004
0.5503	0.5003	0.7504	0.3002	0.6503	0.8004	0.7504	0.5003	0.6503	0.4002

Appendix B.

Calculating the average number of Pokémon : Our task was to find the expected number of samples needed, drawn from $N(\mu, \sigma)$, such that their sum is greater than 720 minutes, or 12 hours. Remember that **Assumption II** stated that the consecutive times for Pokémon spawns were normally distributed, according to $N(30.27, 9.2^2)$. We can then write this expectation as

$$\mathbb{E}[X] = \sum_{n=1}^{\infty} n \cdot \mathbb{P}\big(\sum_{i=1}^{n-1} s_i < 720\big) \cdot \mathbb{P}\big(\sum_{i=1}^{n} s_i \geq 720\big),$$

where the s_i denote samples from the above normal distribution. However, note that the sum of n independent, identically distributed normal variables is itself normally distributed, using the well-known fact that if $X \sim N(\mu_1, \sigma_1^2)$ and $Y \sim N(\mu_2, \sigma_2^2)$ then $X + Y \sim N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. In this way, the sum of n samples from $N(30.27, 9.2^2)$ is distributed according to $N(30.27n, 9.2^2 n)$. Therefore the above expectation can be rewritten as:

$$= \sum_{n=1}^{\infty} n \cdot \mathbb{P}(x_{n-1} < 720)\mathbb{P}(s \geq 720 - x_{n-1}).$$

where $x_n \sim N(30.27n, 84.64n)$. This is difficult to solve analytically, so we utilize Monte Carlo to solve for the $E[X]$. The code below already utilizes this implementation, so we borrow the code from the Monte Carlo analysis of our approaches; we obtain $E[X] = 23.3$, which supports an intuitive calculation of $720/30.26 \approx 23.78$.

Kolmogorov-Smirnov Test: To confirm the normality of Pokémon spawn rates, we quantify a distance measure between what is known as the Kolmogorov-Smirnov *empirical distribution function* and the CDF of our reference normal distribution $N(30.9, 9.2)$. Denote the referece CDF $F(x)$, and define the empirical distribution function $F_n(x)$ as follows

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} I_{[-\infty, x]}(X_i)$$

And the distance metric D_n is given by

$$D_n = \sup_x |F_n(x) - F(x)|$$

Which, for our purposes, can be given the reformulation

$$D_n = \max_{1 \leq i \leq N} \left(F(Y_i) - \frac{i-1}{N}, \frac{i}{N} - F(Y_i) \right)$$

Some limitations we consider:

- It is generally more sensitive near the centre of the distribution than at the tails.
- The reference distribution to which we are contrasting must be fully defined and continuous. If certain distribution parameters are unspecified, the critical region of the test will be invalid. We handle this by explicitly estimating μ, σ^2 using a fitted curve beforehand.
- While generally considered to be not as statistically powerful as the Shapiro-Wilk/Anderson-Darling tests, it is sufficient for our purposes.

Under this test, we have

$$H_0 : \text{The frequency samples are distributed normally}$$

$$H_a : \text{Our data do not follow this distribution}$$

Using an α value of 0.05, our critical K-S value $D_{n,\alpha}$ is (from tables) approximately 0.0304, while the calculated D_n statistic is negligible in comparison (0.00927). We conclude, therefore, that Pokémon frequencies are distributed normally with 0.95 confidence.

Lilliefors test for Exponentiality The exponentiality of a distribution can be affirmed, to a certain confidence range, by the Lilliefors test. We begin by applying the transformation

$$Z_i = \frac{Y_i}{Y}$$

For random samples Y_i. For our test statistic, let $S(x)$ denote the empirical distribution function, as before in Kolmogorov-Smirnov determined by random variable Z, and let $F(x)$ denote the CDF of the reference exponential distribution $F(x) = 1 - e^{-x}$ against which we are testing. The test statistic, W, is given by

$$W = \sup_y |F(y) - S(y)|$$

Under this test, we have:
H_0 : The random sample is distributed according to:

$$F'(x) \begin{cases} 1 - e^{-\frac{x}{\alpha}} & x > 0 \\ 0 & x < 0 \end{cases} \qquad \text{for some } \alpha \in \mathbb{R}$$

H_a : Our data do not follow this distribution

Appendix C.

Code : The following is code for the number of Pokémon seen in a 12 hour period, followed by the code for Approaches 0, 1, and 2.

```
% A Monte Carlo Simulation of how many pokemon appear in a span of 12
 hours
% Here we took the running sum of random samples until the sum reached
 over
% 720.
n = 1000;
Matrix = zeros([1,n]);
for i = 1:n
    time = 0;
    numPokemon = 0;
    while time < 720
        time = time + normrnd(30.27,9.2);
        numPokemon = numPokemon + 1;
    end
    % Don't count the last pokemon spwaning past the 12 hour period
    Matrix(i) = numPokemon - 1;
end
mean(Matrix)


ans =

   23.3100
```

```
%MonteCarloPokemon Method0!
filename = '/Users/sianamuljadi/Documents/MATLAB/Pokemon/
PokemonValues.csv';
M = csvread(filename);
Coordinates = M(:,[1,2]);

FreqMap = zeros(10);
for i = 1:1999
    x = Coordinates(i,1);
    y = Coordinates(i,2);
    FreqMap(x,y) = FreqMap(x,y) + 1/1999;
end
%disp(FreqMap);
%sum(sum(FreqMap))

% Strategy is to stay at the point (3,8), collect only pokemon that
 spawn
% there

%See how many pokemon spawn in all, taking normal samples
n = 1000;
Matrix = zeros([1,n]);
for i = 1:n
    time = 0;
    numPokemon = 0;
    while time < 720
        time = time + normrnd(30.27,9.2);
        numPokemon = numPokemon + 1;
    end

    points = 0;
    for j = 1:numPokemon-1
        x = rand(1);
        if x < FreqMap(3,8)
            points = points + ceil(exprnd(4.67));
        end
    end

    Matrix(i) = points;
end
mean(Matrix)


ans =

    4.8640
```

```
%MonteCarloPokemon Method1!
filename = '/Users/sianamuljadi/Documents/MATLAB/Pokemon/
PokemonValues.csv';
M = csvread(filename);
Coordinates = M(:,[1,2]);

FreqMap = zeros(10);
for i = 1:1999
    x = Coordinates(i,1);
    y = Coordinates(i,2);
    FreqMap(x,y) = FreqMap(x,y) + 1/1999;
end
%disp(FreqMap);
%sum(sum(FreqMap))

% Strategy is to stay at the point (3,8), collect only pokemon that
 spawn
% there

%See how many pokemon spawn in all, taking normal samples
n = 5000;
Matrix = zeros([1,n]);
for i = 1:n
    time = 0;
    numPokemon = 0;
    while time < 720
        time = time + normrnd(30.27,9.2);
        numPokemon = numPokemon + 1;
    end

    points = 0;
    for j = 1:numPokemon-1
        x = rand(1);
        if x <
 FreqMap(3,8)+FreqMap(3,7)+FreqMap(3,9)+FreqMap(2,8)+FreqMap(4,8)
            points = points + ceil(exprnd(4.67));
        end
    end

    Matrix(i) = points;
end
mean(Matrix)


ans =

   20.0294
```

Appendix D.

Code : The following code outlines the Monte-Carlo approach used to estimate points-per-game of our search algorithm, approach 2.

```
Model 2 Algorithm - Monte carlo estimates
```

```
% Initialize the frequency estimates for each grid portion
frequency_array = [0.6003    0.6003    0.3502    0.5503    0.4502    0.8004    1.2006    3.6518    1.6508
0.4002;
    0.7004    0.5003    0.7004    0.6003    0.9505    0.4502    1.9010    2.9515    2.0010    0.8004;
    0.6003    0.5503    0.7004    0.6003    0.6003    0.4502    2.5513    4.2021    3.1016    1.7509;
    0.5003    0.5003    0.3002    0.6503    0.5503    0.4502    1.7009    3.7519    1.6508    1.0505;
    0.7504    0.8504    0.7004    0.4502    0.5003    0.4002    0.9005    0.9005    0.3502    0.5503;
    0.7004    1.8009    0.5003    0.6503    1.0005    0.9505    1.0005    0.4502    0.5003    0.6003;
    0.7004    1.6008    0.4002    0.9505    0.7004    0.8504    0.4002    1.0005    0.6003    0.6003;
    2.4012    3.1516    1.7509    1.5508    0.4002    0.7504    1.0505    2.1511    1.0505    0.7504;
    0.4002    1.8509    0.9505    0.9505    0.6503    0.5003    0.8504    1.2506    0.6003    0.8004;
    0.5503    0.5003    0.7504    0.3002    0.6503    0.8004    0.7504    0.5003    0.6503    0.4002];

points_estimate = 0;
% Monte Carlo algorithm estimating the average points accrued for each
% 12-hour iteration of the algorithm

function bool = is_within_reach(x_coord,y_coord)
   if abs(current(1) - x_coord) + abs(current(2) - y_coord) < 2
       bool = 1
   else bool = 0
   end
end

for q = 1:10000

time = 0;
pokenum = 0; % number of pokemon found
spawned = zeros(5,35); % Fifth row is the "caught" boolean flag. If we catch the pokemon we set to 1
while(time <= 720)
    time = time + normrnd(30.92,9.2);
```

```
    pokenum  = pokenum + 1;
    spawned(1,pokenum) = time;
end

time = 0;

% Initialize our pokemon spawns using a hardcoded distribution table
for l = 1:pokenum
    prob = 0;
    randnum = rand*100;
    for i = 1:10
        for j = 1:10
            if (prob < randnum) && (randnum <= prob + frequency_array(i,j));
                spawned(2,l) = i; % x coord
                spawned(3,l) = j; % y coord
                spawned(4,l) = ceil(exprnd(4.67)); % Ceiling of exponential distribution samples to remove
"level 0" outliers
            end
            prob = prob + frequency_array(i,j);
        end
    end
end
current = [3,8];
points = 0;
index = 1;
    for t = 0:7.5:720
```

```
        % Gets which pokemon is currently spawned, places in index.
        for i = 1:pokenum
            if spawned(1,i) <= t && t < spawned(1,i+1)
                index = i;
                break
            end
        end
```

```
if i + 1 <= pokenum
   if spawned(1,i+1) <= t
          if is_within_reach(spawned(2,i+1), spawned(3,i+1)) == 1 && spawned(4,i+1) > spawned(4,i)
            index = index + 1
          end
    end
end

        % check if spawned is within reach
       if abs(current(1) - spawned(2,index)) + abs(current(2) - spawned(3,index)) > 2
                % Go back to starting location with highest mapped values
            if current(1) > 3
                    current(1) = current(1) - 1;
            elseif current(1) < 3
                    current(1) = current(1) + 1;
            elseif current(2) > 8
                    current(2) = current(2) - 1;
            elseif current(2) < 8
                    current(2) = current(2) + 1;
            else
                continue
            end
        else
                % newly spawned is within reach, we move towards it
            if current(1) - spawned(2,index) > 0
                  current(1) = current(1) - 1;
            elseif current(1) - spawned(2,index) < 0
                     current(1) = current(1) + 1;
            elseif current(2) - spawned(3,index) > 0
                    current(2) = current(2) - 1;
            elseif current(2) - spawned(3,index) < 0
                    current(2) = current(2) + 1;
            else
                % Checks if we've already "caught" the pokemon using
```

```
                % boolean flag
                if spawned(5,index) ==   0 ;
                    points = points + spawned(4,index);
                    spawned(5,index) = 1;
                end
             end
       end
    end
    points_estimate = points_estimate + points;
end

points_estimate = points_estimate/10000
```

```
points_estimate =
 35.9544
```