An Agent-Based Model for the Optimization of Pokémon Points in Providence Pokémon Po

Gabe Akers, Craig Fouts, Carly Middleton

January 4, 2021

Summary

Providence Pokémon Po is developing a game based on Niantic's Pokémon Go! with many of the same features. The goal is to develop a method of traversing the game's coordinate grid to maximize the number of points collected in a 12 hour period. Our team approached this problem by statistically modeling the spawn times, point value spawn frequencies, and coordinate weights based on the Pokémon spawn data recorded over the last 42 days. We then developed a Python simulation of the game in which to test our hypothesis. Our models revealed areas of the grid which displayed higher-value Pokémon activity on average. This, in addition to the limited lifetime of each Pokémon, inspired our final approach: remain in higher priority areas with our radius of focus limited to two streets over. This ensures that we maximize our chances of encountering Pokémon, while remaining focused on those which are reachable before they despawn.

1 Introduction

The Providence Pokémon Po app has so far been given many behaviors by its developers, those behaviors by which we treated as rules to create our model by. These behaviors include: Pokémon spawn in random locations approximately every 30 minutes, remain on a map for exactly 15 minutes, and then disappear. The map consists of a 4 x 4 mile map of a downtown area, whereby the player can move between 100 nodes on a 10 x 10 grid within the map. The downtown area is assumed to be populated with buildings; for this reason the player is assumed to move forward, backward, left, or right along streets but not diagonally through buildings. Finally, each Pokémon has an associated point value ranging from 1 to 20 points; rare Pokémon have high point values and common Pokémon have low point values.

It is of interest to us be able to determine an optimal way to collect as many points as possible in a 12 hour period of gameplay. In order to do this, we employed some basic concepts of agent-based modeling and statistical testing. We created an agent-based model in Python which simulates many behaviors of the Providence Pokémon Po app including agent (or player) mobility, Pokémon point values, and spawning

behavior of Pokémon. We incorporated data from `Providence_Pokemon.xls` into our model in order to simulate the conditions of the actual app, and then we evaluated the efficiency of our model by comparing the amount of points collected by our agent to an estimated amount of possible points to be potentially collected from four Pokémon hotspots we identified using `Providence_Pokemon.xls`. We concluded that our agent was, on average, able to collect about 17.17 points every 12 runs, which is 73.56% of potential Pokémon points.

2 Model and Methods

2.1 Creating the Model

To allow for stronger visualization and to provide a framework for testing our algorithm we created a simulation of the problem. The simulation can be downloaded from GitHub [1] where a more detailed description of the exact inner workings of the code can be found. The simulation was written in Python and is playable by humans as well as by a programmable agent. The human controlled aspect of it was used to develop the algorithm for control of the agent, and after the algorithm was developed the agent was programmed to follow it. The number of points the agent was able to collect was used to evaluate the algorithm's effectiveness.

The simulation uses a one-to-one correspondence between seconds and minutes as its scale. Meaning that one second in the simulation is equivalent to one minute in "real life." All of the Pokémon are spawned on average every 30 seconds and disappear after exactly 15 seconds. The simulation is set to run for 720 seconds after which the agent's score is recorded.

The gridlines in the simulation are 45 pixels apart which corresponds to .4 miles in "real life" as described in the problem statement, and the grid is considered a graph for the purposes of the simulation. From the literature we found that humans move on average 3 miles per hour [2] so it should take a human 8 minutes to traverse .4 miles or to make one "hop" i.e. move from the current vertex to the next. Since the agent traverses the graph in steps, and minutes in real life are in one-to-one correspondence with seconds in the simulation we allow the agent to hop once every 8 seconds.

To simplify the model we don't consider Pokémon spawning anywhere except at vertices of the graph which corresponds to intersections of streets in the problem statement. The Pokémon spawn time is determined by a Gaussian distribution with $\mu = 30.2$ and $\sigma = 9.2$. The Pokémon point values are determined by an exponential distribution with $\lambda = 1/4.6685$. The Pokémon spawn locations are determined by sampling with replacement from the vertices weighted by a weighting factor specific to each vertex of the graph.

To determine the distribution of the Pokémon spawn location we fit a Gaussian distribution to the time between Pokémon spawns using Python. The time between Pokémon spawns was plotted (Figure 1) and it clearly has the form of a Gaussian distribution so using Python the mean and standard deviation were found. We tested this with the Lilliefors test for normality at a 0.05 significance level. A p-value of 1.6237 x 10^{-05} was found and so there is not sufficient evidence to reject the null hypothesis

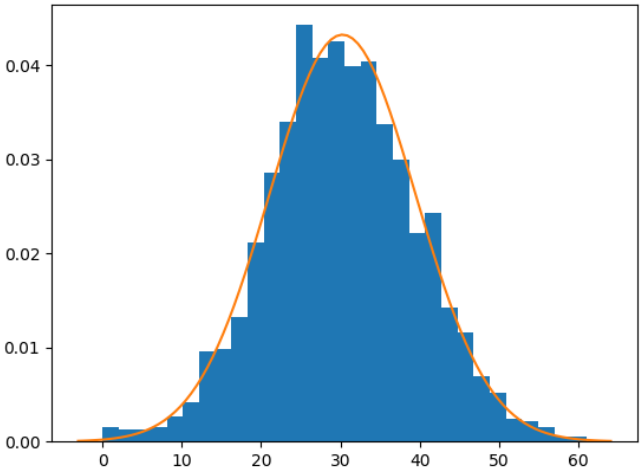

Figure 1: Frequencies of time in minutes between Pokémon spawns in `Providence_Pokemon.xls`, and the fitted Gaussian curve for these frequencies.

that the data is normally distributed with mean and variance estimated from the data itself. The details can be found in `modelling_exploration.py` in the linked code and in the accompanying Jupyter notebook.

Data suggests that the frequency at which the Pokémon spawn is dependent on point value; lower level Pokémon tend to spawn more frequently than higher level Pokémon, as seen in Figure 2. We observed that the distribution of the Pokémon points appeared to be exponential (Figure 2) and so we fit it to find the parameters. We took data from `Providence_Pokemon.xls` and fit an exponential function to it which continuously approximates the frequency of each point value. With this exponential approximation, we were able to accurately reflect each point value's occurrence frequency in our Python simulation. We found that the equation best matching the data provided was:

$$y = 410.9192e^{-0.2142t}$$

Solving this for lambda gives us

$$\lambda = 1/4.6685.$$

As before we consider the Lilliefors test at a 0.05 significance level and find a p-value of 0.01 (the lower bound when testing for exponential distribution) and thus fail to reject the null hypothesis that the distribution is exponential.

Finally, to determine the weights on the nodes we first plotted the data as a histogram in a 3D space. The x and y coordinates of the spawn determined the location and height was the number of Pokémon that spawned at that location over the course of the time period described by the data (Figure 3). This clearly shows that some

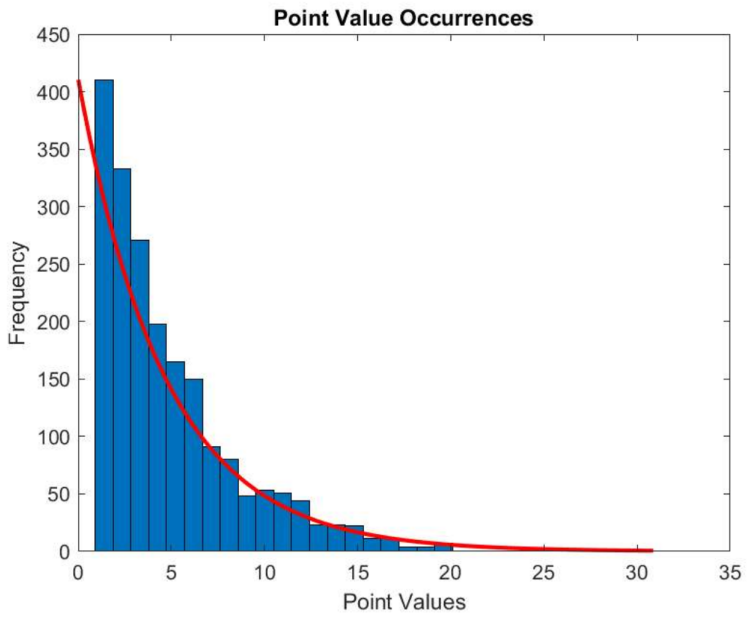

Figure 2: Frequencies at which each Pokémon point value occurs throughout `Providence_Pokemon.xls`, and the fitted exponential curve for these frequencies.

locations were favored over others. To obtain the weights for the Pokémon spawns we simply divided the number of Pokémon seen at a location by the total number of Pokémon seen over the period. These weights correspond to the probability that a given node will be chosen as the spawn point.

For our algorithm we decided that a simple approach would be best given time constraints. Based on the data we have, in particular the data about where Pokémon are most likely to spawn and a heuristic, we decided to have our agent patrol the four vertices $H1, H2, H3$ and $H4$. These correspond to (1,7), (2,7), (3,7), and (4,7) in the weight matrix and are the highest weighted vertices in their respective rows. In the simulation the agent is placed initially on a random vertex, or "node" as we call it, and first paths to the node "hotspot" described. While patrolling this hotspot the agent searches up to two nodes away for Pokémon. If a Pokémon is spotted it is targeted. If the previous target was a Pokémon then we compare the point values of the two. For a new Pokémon to be an eligible candidate it must be both of higher value and no farther than the currently targeted Pokémon. The decision to search two "hops" away was made because the Pokémon de-spawn every 15 minutes, but it takes 8 minutes by our estimate of human walking speed to make one hop (move from one vertex to the next). Therefore two is the maximum number of hops away from the current position that can be reached before a Pokémon on the targeted node would disappear. Not switching targets unless both the conditions described are met is due to a similar line of reasoning. If a Pokémon is currently targeted and the agent has made one hop towards it then spots another of higher point value two hops away, the edge of the agent's vision, then switching targets would mean both time is being wasted and potentially the points

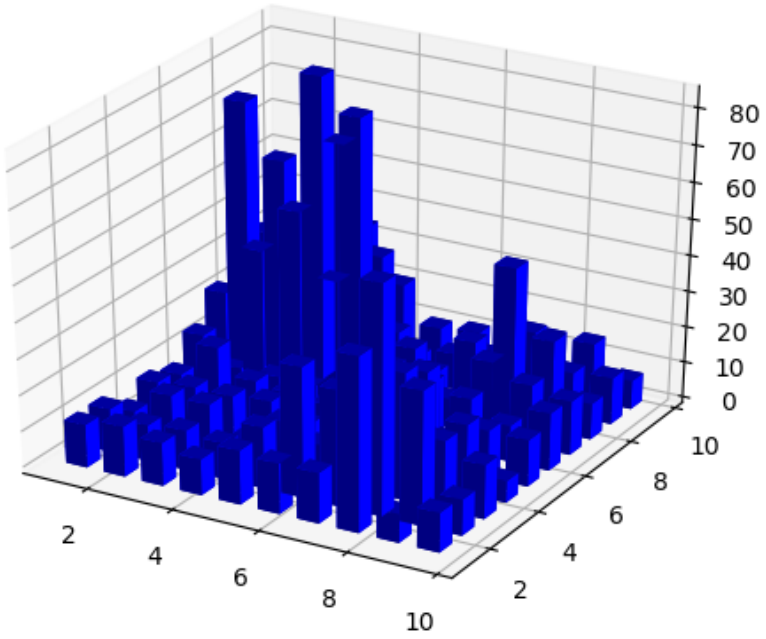

Figure 3: Number of Pokémon (Z axis), regardless of point value, that spawned at each (x,y) coordinate over the course of the time period in `Providence_Pokemon.xls`. X coordinates appear on the left axis and Y coordinates appear on the right axis.

from both could be lost. The agent has no information regarding the spawn time of the Pokémon it sees so we simply assume that all Pokémon within the vision of the agent are reachable in time, but in fact many are not. This is because it takes 8 minutes to hop from node to node, but Pokémon despawn after 15 minutes. Therefore only Pokémon that spawned while the agent was moving from node to node are reachable. In particular at least 12.5% of the agent's hop wait time must be completed before the Pokémon spawns two hops away if it is to be reachable. The agent also does not have knowledge of the Pokémon de-spawning prior to arriving at its location. The explanation for this is that the Pokémon "calls out" when it appears on the grid and only Pokémon within the agent's vision are audible. Since the Pokémon doesn't call out again the agent has no way to verify that a Pokémon is still at its previous location until it arrives at that location. Upon arriving at the targeted location the agent returns to the hotspot and continues patrolling.

2.2 Model Results

With our algorithm implemented the agent running our simulation was able to achieve an average of 17.17 points over 12 runs. Suppose a Pokémon appears every 30 minutes and it's worth 4.6685 points. The hotspot the agent patrols then has a per node value of $4.6685 * w(x, y)$ where w is the weight function at that node. In particular $H1, H2, H3$, and $H4$ have weights of 0.024012, 0.031516, 0.017509, and 0.015508 respectively. Thus the hotspot the agent patrols has a per node value of 0.1121, 0.1471, 0.08174, and 0.07234 respectively. Meaning that every 30 minutes if the agent is on either $H1, H2, H3$, or $H4$ we expect the score to increase by the amount described. Continuing in this manner we see that the regional value, the sum of all node values within two hops of the hotspot, is 1.16304. Since a Pokémon appears every 30 minutes by assumption then 24 Pokémon should appear on the board in 720 minutes (12 hours). The total sum of the weights in the region within two hops of the hotspot path is 0.2491. This means we should expect $24 * 0.2491 = 5.9784 \approx 5$ Pokémon to spawn within the region within a 12 hour period. So we should expect to be able to earn $5 * 4.6685 = 23.3425$ points within a given session on average by patrolling that region. Based on this metric our agent was able to achieve 73.56% of the expected score. This missing 26.44% can be accounted for partially by errors in the simulation itself, the fact that not all Pokémon that spawn within two hops are always reachable, and the fact that the expected points that can be accumulated in the region does not take into account the agent's position (i.e. if the agent is at node $H1$ and a Pokémon spawns within the region at $G4$ it simply can not be reached and in fact will not even be in view of the agent until it moves closer). The final source of error is due to the stochastic nature of the system, but we should expect that in the limit this error would vanish.

2.3 Investigating Data from `Providence_Pokemon.xls`

Before creating the model, it was of interest to be able to visualize any potential spawning patterns of the Pokémon. Such a visualization is displayed in Figure 4. The x and y axes of this cube represent the x and y coordinates of the downtown area

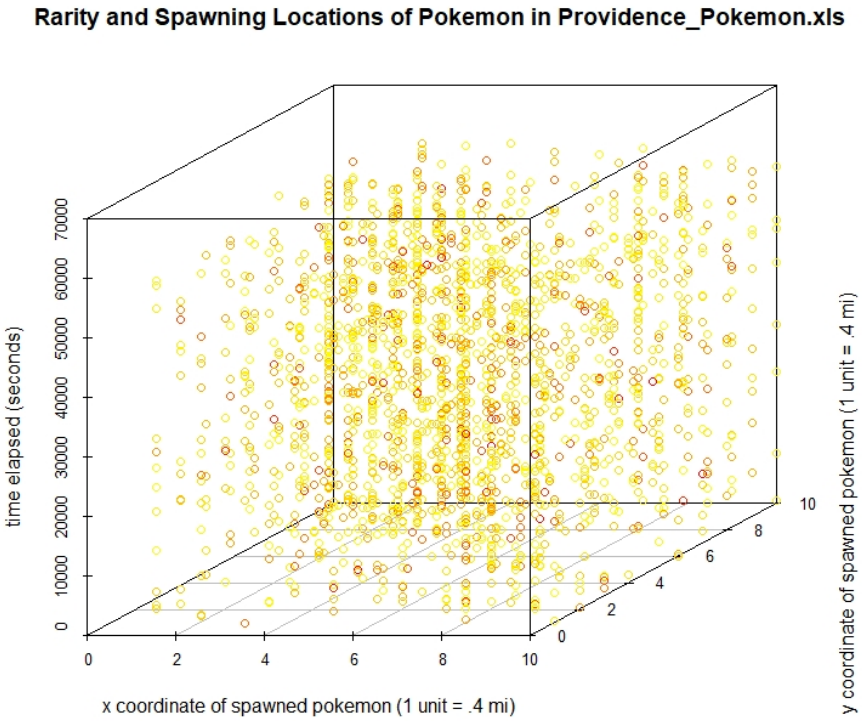

Figure 4: Spawning locations of Pokémon in `Providence_Pokemon.xls`. Rarity is described along a continuous color gradient spanning red to yellow. Red ("rare") Pokémon have the highest associated point values, and yellow ("common") Pokémon have the lowest associated point values.

the player walks in. The z axis represents the time t in seconds at which a spawning occurred in `Providence_Pokemon.xls`, and each circle within the cube represents one Pokémon which spawned.

This visualization represents spawnings only and not the 15 minute lifespan and subsequent disappearance of the Pokémon. Additionally, a color scheme was applied by which Pokémon associated with the highest point values ("rarest" Pokémon) are red, and Pokémon associated with the lowest point values ("common" Pokémon) are yellow. The color scale uses a continuous gradient – in effect, Pokémon appear with a wide range of colors between red and yellow depending on their point value association ("rarity").

Figure 5 displays the same x and y axes, but its z axis graphs the variable Hotspot Potential. This variable is a measure of the total points earned by the (x,y) coordinate throughout `Providence_Pokemon.xls`. It was not directly utilized in the model, but was helpful in visualizing the `Providence_Pokemon.xls` data.

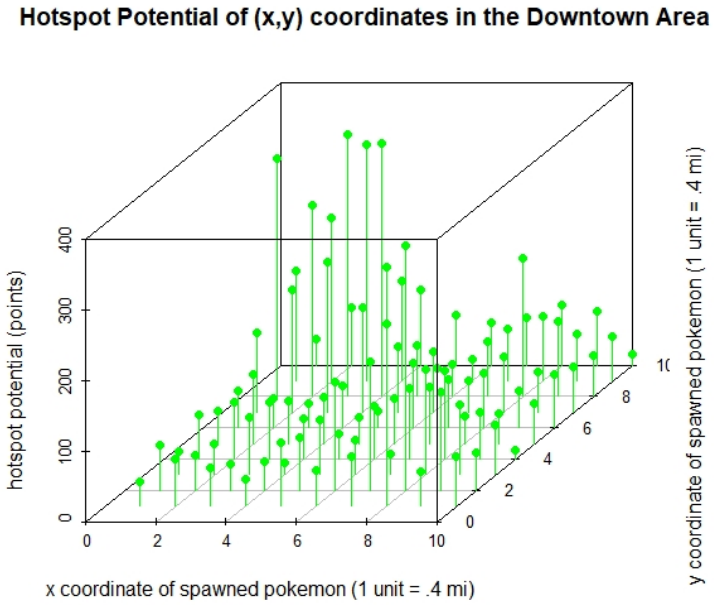

Figure 5: Hotspot Potential (total points earned by each (x,y) coordinate throughout `Providence_Pokemon.xls`

3 Robustness and Future Work

3.1 Summary

Over a period of 12 hours we were tasked with gathering as many points as possible. The 10 x 10 graph over a 4 x 4 mile city center had Pokémon appearing on average every 30 minutes and disappearing after exactly 15. We chose to model the spawn times and point distributions with Gaussian(30.2, 9.2) and Exponential(1/4.6685) distributions respectively. In our model Pokémon could not appear anywhere except at vertices in the graph and the locations they appeared at was determined by choosing with replacement from all available vertices weighted by a factor determined from the data specific to the vertex in question. Using these distributions we created a simulation of the problem and developed an algorithm described in detail above that was able to achieve 73.56% of the expected score in the simulation.

3.2 Conclusions

Our agent achieved a score of 73.56%, by our metric, using the algorithm we described. The authors are not aware of any prior work on the subject and so without anything to compare it to we would evaluate this as "pretty good but room for improvement." 73.56% of the expected score seems to be rather high for such a simple algorithm and suggests that our algorithm is headed in the correct direction.

3.3 Limitations/ Future Work

There were a number of factors limiting the success of our algorithm including, but not limited to, errors in the simulation and time constraints on the authors. The errors in the simulation are primarily a result of inefficiencies in the calculations being performed. Variable frame times as a result of these inefficiencies made timing slightly inaccurate which resulted in certain actions being taken by the agent at particular times when they should not have been and vice versa. This is most probably the largest source of error in the model and resulted in the highest score loss. In the future more development time could be allocated to rectify this. There are many easy optimizations that could be implemented that would greatly bolster the performance of the simulation and would certainly cause the agent to achieve a higher average score. Another source of error in the simulation was the pathfinding. We implemented a very basic naive form of pathfinding that consists of moving along the x axis until the agent is the same column as the target and then moving along the y axis to the target. In the future Djkstra's algorithm for instance could be used as a substitute to allow for more efficient pathfinding and therefore time saved moving from point to point.

The time constraints on the authors of this paper also certainly resulted in a lower score being achieved by the agent. With only a few days to work on the problem the depth we were able to consider some of the details of the problem at was limited. In particular we would have liked to have been able to devote more time to developing the algorithm. The algorithm devised is very simple, bordering on too simple, and given more time we could iterate on the current version and increase its complexity thereby almost certainly improving its performance. For instance, we could model the de-spawn time of the Pokémon as a function of when they were first seen by the agent. If the agent could calculate some probability that the seen Pokémon disappears before it can be reached then it could make a more informed decision about whether or not the Pokémon should be targeted or passed over. There are also opportunities to consider things like whether certain nodes have more high value Pokémon appearing on them than others and if we could optimize to account for this. In particular, future work should certainly consider reinforcement learning as a promising direction to explore for this problem.

References

[1] GitHub https://github.com/GabrielAkers/Math4194-PokemonPo.

[2]