CS221 Project Proposal
Catch'em all: A Pokémons Go Playing AI

SUNet ID: yulelee
Name: Yue Li

Background

Pokémons Go is a popular game for which you need to get out of your house and explore the outside. There are many components within the game, however, in this project, I'll focus on the most attractive part to me: catching Pokémons. The basic rule is: Pokémons are spawned at different places in the real world, if you are near enough to them, they will show up on your screen, so that you can catch them. As a result, the No.1 question of all the players: where are the Pokémons! Also, to give the players a little bit of hint, the game provides a feature called "sightings", which is essentially a radar system: if a Pokémon is at somewhere near the player, it will show up on the radar, and will keep showing up as long as the player is walking towards the correct direction, until the player actually find it; on the other hand, it would disappear if the player goes to the wrong way.

Goal

In this project, I want to implement an AI agent that could use information provided by the game (radar system as described above), and try to catch as many Pokémons as it can.

Data and Preprocess

The dataset being used is "SF Bay Area Pokemon Go Spawns"[1], which contains the spawning information of more than 300k Pokémons. The information includes the ID (from 1 to 151) and the name of the Pokémon, the position (longitude and latitude) of the spawn, as well as the appearing and diasppearing time. With this dataset, I'm able to build a "frequency model" and a "score model" based on basic statistics (rarer Pokémons has higher scores), and a "neighbor model" using the Naive Bayes method, exploring the spatial and temporal similarity of the Pokémon spawns.

Simulation

First part of the simulation is a Pokémon selector: when selecting Pokémons, it first refers to the "frequency model" and generate one initial Pokémon, and then, based on the "neighbor model" and this initial Pokémon, it would further select a few neighbor Pokémons.

The second part is a game board, because the real world is frustratingly huge, I choose to simulate the game on a $n \times n$ board. The major task of the board is to spawn the Pokémons generated by the Pokémon selector. Right now I've implemented two types of board, first one is "random board", which spawn the initial Pokémon at a random place and put all the neighboring Pokémons somewhere near the initial spawn. The second board is "regional board", which follows the same procedure as the "random board", but provides the extra feature that each type of Pokémon has a spawn center. This is more similar to the real game, where the spawns are mostly random, but you are more likely to find a certain type of Pokémon within a certain region. Furthermore, all of the Pokémons can only exist for a random amount of time, which is predetermined when it is spawned.

The simulation is proceeded by iterations. At each iteration, the agent can move one step on the board (go "Left", "Right", "Up" or "Down"). The board would reward the agent with some scores if the agent catches any Pokémons (if there are Pokémons currently at the agent's new position). The board also spawns a few new Pokémons and delete the Pokémons whose time is up. At last, the board gathers the radar information for the agent, which simply contains the IDs of the nearby Pokémons, no score, no position, and no actual distance.

[1] https://www.kaggle.com/kveykva/sf-bay-area-pokemon-go-spawns

Input and Output

For this game, no special input is needed, the game could just start (agent being put to the center of the board). We also don't have an end state, a player can just play as long as he wants to continue. The game will be forced to end after a certain amount of iterations, and the output is the total amount of rewards (scores), as well as the total number of Pokémons the players has caught, the more the better! Also, to make the results more comparable, the output could also be defined as the percentage of the scores being rewarded to the agent, out of the total scores of all the Pokémons ever being spawned onto the board.

Oracle Agent

The oracle agent has access to all of the information of the board, it knows the exact positions of all the Pokémons and their disappearing time, as well as all of the scores. Based on those information, it constructs a graph connecting all of the Pokémons: each Pokémon is represented by a node, and an edge from one node to the other means it's possible for the player to catch this Pokémon first and then go to the next one before the next one disappears, and the weight on each edge is defined to be the reward at the end node of this edge. Now the problem is transformed into finding the longest path on this graph, start from the agent's position.

Since this is still not trivial, I deleted all the "possible edges" in the graph, and only leave the "guaranteed edges", which is defined by the edges representing that you are guaranteed to be able to catch the second Pokémon after catching the first one, not just possibly. As a result, it can be proved that now the graph is reduced to a DAG, and the longest path can be found by dynamic programming following a topology sort.

Baseline Agent

The baseline agent only knows about its own position and what is on the radar. Here the game is defined to be a MDP problem. States contain the current position of the agent, and the list of Pokémon IDs in radar. Action includes the four directions the agent can choose (as long as it doesn't fall off the board). No end state is needed and the game would terminate after a certain iterations of simulation.

Q-learning algorithm with function approximation is applied, the feature extractor is fairly straight forward, which transform the current position and the Pokémon IDs in radar (with their rankings) into indicator features.

Current results and Discussion

With 30000 iterations, on the "regional board"(50×50): The oracle agent can get an average of more than 50% of the total scores and catch 20% of the total Pokémons being spawned. Considering that some Pokémons are spawned too far away from the agent and it's essentially impossible for the agent to catch them, those numbers should be satisfying. Also, the fact that the percentage of scores is higher than the percentage of the number of Pokémons, means that the agent is actively giving higher priorities to the Pokémons with higher scores.

However, the baseline agent doesn't perform well, it can only catch 0.1% of the total scores, and 0.1% of the total number of Pokémons. In fact, its performance is at the same level of a random agent. On the one hand, this baseline is not very successful, but on the other hand, I do have lots of space to improve!

Analyzing the current result, I think the major challenge for Q-learning is firstly, the board is always changing, even though Q-learning can adapt to new situations, it cannot adapt that fast; and second, the reward is not effective, it may take many many "correct" moves for the agent to finally get the reward.

Next step, I will try to further optimize the baseline, for example, let the agent remember the last radar information, even though the official reward is offered by the board (for catching Pokémons), the agent could give itself some partial credits for keeping the Pokémons within the radar (not losing track of them). And also, maybe I need to introduce deep learning. All in all, the plan is to stick with the reinforcement learning but also keep an open mind for other methods that might be more suitable for this problem.