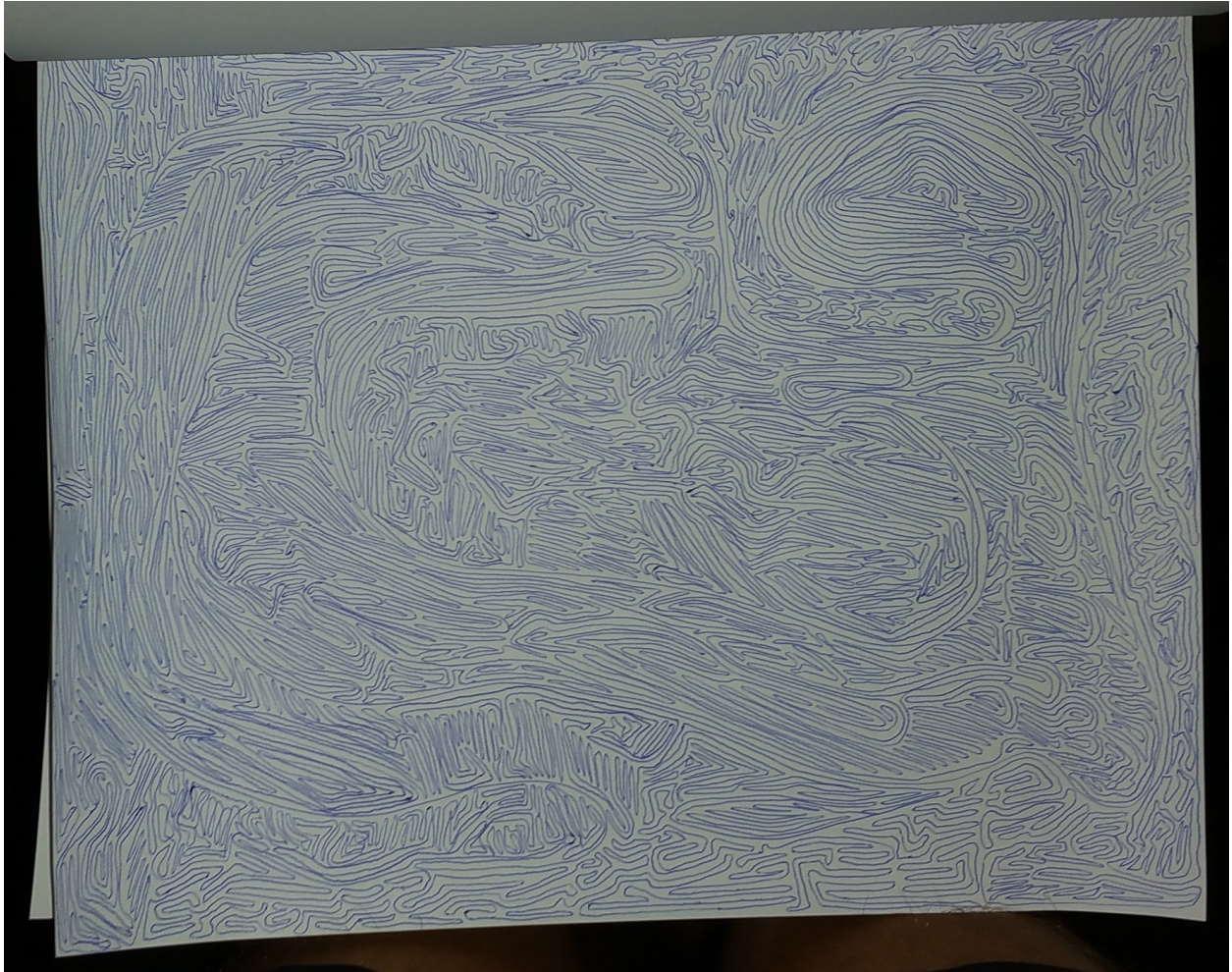


**Wavy Line Search Problem
CS 221 Project Final Report
Fall 2017**

Mateo Garcia



Task Definition

My system generates a "wavy line." It draws a single line through a grid of points, with some element of randomness. The objective is to fill as much of the grid as possible with this line.

Infrastructure

I chose to use Processing in Python for my project.

Approach

- **Task definition:** What does your system do (what is its input and output)? What real-world problem does this system try to solve? Make sure that the **scope** of the project is not too narrow or broad. For example, building a system to answer any natural language question is too broad, whereas answering short factoid questions about movies is more reasonable. This is probably the most important part of the project, but is also the part that you will not get practice doing from homeworks.

The first task you might come up with is to apply binary classification on some standard dataset. This is probably not enough. If you are thinking in terms of binary classification, you are probably thinking too narrowly about the task. For example, when recommending news articles, you might not want to make predictions for individual articles, but might benefit from choosing a diverse set of articles.

An important part of defining the task is the **evaluation**. In other words, how will you measure success of your system? For this, you need to obtain a reasonably sized **dataset** of example input-output pairs, either from existing sources, or collecting one from scratch. A natural evaluation metric is accuracy, but it could be memory or running time. How big the dataset is depends on your task at hand.

- **Infrastructure:** In order to do something interesting, you have to set up the infrastructure. For machine learning tasks, this involves collecting data (either by scraping, using crowdsourcing, or hand labeling). For game-based tasks, this involves building the game engine/simulator. While infrastructure is necessary, try not to spend too much time on it. You can sometimes take existing datasets or modify existing simulators to save time, but if you want to solve a task you care about, this is not always an option. Note that if you download existing datasets which are already preprocessed (e.g., Kaggle), then you will be expected to do more with the project.
- **Approach:** Identify the challenges of building the system and the phenomena in the data that you're trying to capture. How should you model the task (e.g., using search, machine learning, logic, etc.)? There will be many ways to do this, but you should pick one or two and explain how the methods address the challenges as well as any pros and cons. What algorithms are appropriate for handling the models that you came up with, and what are the tradeoffs between accuracy and efficiency? Are there any implementation choices specific to your problem? Besides your primary approach(es), you should have two types

of other ones, **baselines** and **oracles**. These are really important as they tell you how significant the problem you're solving is. Baselines are simple algorithms, which might include predicting the majority label, using a small set of hand-crafted rules, training a simple classifier, etc. Baselines are meant to be extremely simple, but you might be surprised at how effective they are. Oracles are algorithms that "cheat" and look at the correct answer or involve humans. If your problem has two components (identifying entities and classifying them), an oracle uses the correct answer for one of the components so that we can see how well the other can do. Baselines give lower bounds and oracles give upper bounds on your performance. If this gap is too small, then you probably don't have an interesting enough task.

- **Literature review:** Have there been other attempts to build such a system? Compare and contrast your approach with existing work, citing the relevant papers. The comparison should be more than just high-level descriptions. You should try to fit your work and other work into the same framework. Are the two approaches complementary, orthogonal, or contradictory?
- **Error analysis:** Design a few experiments to show the properties (both pros and cons) of your system. For example, if your system is supposed to deal with graphs with lots of cycles, then construct both examples with lots of cycles and ones without to test your hypothesis. Each experiment should ask a concise question, such as: *Do we need to model the interactions between the ghosts in Pac-Man?* or *How well does the system scale up to large datasets?* Analyze the data and show either graphs or tables to illustrate your point. What's the take-away message? Were there any surprises?