Stopping the Spread: Peach Tree Disease

Group 76: Allison Fister, Fletcher Wells, Mihir Gandhi, Shoale Badr

Git Repository:
https://github.gatech.edu/mgandhi39/Tree-Disease-Spread-Cellular-Automata

Video Submission:
https://youtu.be/WGLxum7YXLg

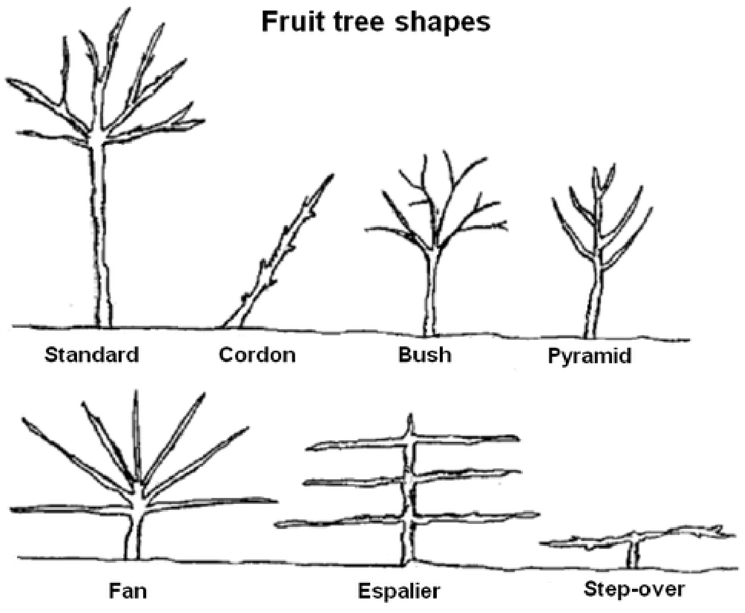
Abstract

We propose a system that models the spread of disease throughout the branches of an individual fruit tree and determines the “optimal” shape to which a tree should be pruned in order to minimize the spread of disease. Our system is unique because of its scope; while many cellular automata (CA) models simulate the spread of disease between distinct individuals, we model infection between different parts of a single entity. This system consists of four major parts: (1) individual tree generation in a grid space, (2) a CA representation to model the spread of disease, (3) visualization of this model, and (4) optimal tree shape analysis. Ultimately, we hope that our model can serve as a novel computational approach to making tree maintenance decisions and can provide general insight into prevention of tree-borne disease.

Project description

Pruning a tree is imperative to maintaining the tree’s health, protecting nearby people and property, and promoting fruit production (Douglas). It is an especially important task for orchard farmers and others who seek to maximize both the quantity and quality of a tree’s output.

There are many known pruning patterns and shapes of fruit trees commonly used in agricultural settings. Below are some notable tree shapes:



Tree Shape	**Description**
Standard	Similar to bush shape, but much taller. Harder to maintain and prune. Bears lots of fruit, but takes a long time to mature.
Cordon	Single stem tree pointing at a 45 degree angle. Bears fruit early. Doesn't bear much fruit.
Bush	Easy to maintain and prune. Often used for apple trees. Bears fruit at a young age.
Pyramid	Similar size as bush. All branches stem from central branch or trunk.
Fan	Has a short trunk with straight branches coming out in a circular pattern.
Espalier	Has a short trunk with three or four horizontal branches on either side.
Step-over	A foot tall espalier tree with two branches.

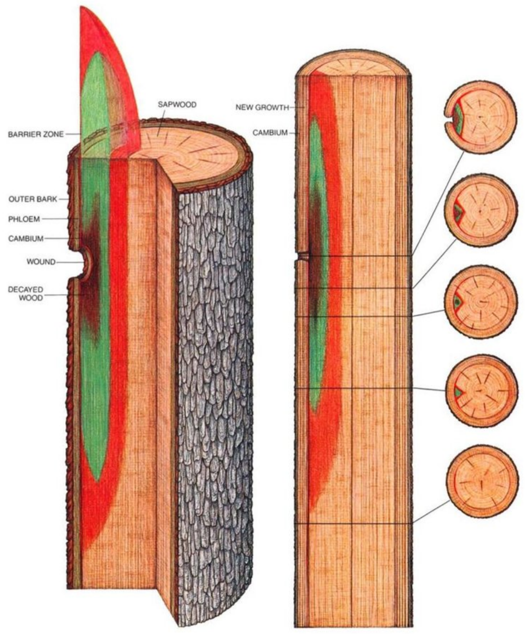
And here are some varieties of disease that threaten fruit trees, specifically peach trees:

Disease	**Description**	**Symptoms**
Blossom Blight	Occurs when a tree is about to bloom. Quickly spreads from blooms to branches and leaves.	Drooping blooms, tan blossoms, or gummy-like branches
Phytophthora Crown Root Rot	Occurs when there is poor drainage in the soil. Begins in the roots.	Reduced tree growth, loss of leaves, death of tree
Armillaria Root Rot	Similar to Phytophthora Crown Root Rot. Spreads through soil. Attacks tree roots. Causes mushrooms to grow at the base of the tree.	Wilting foliage, death of tree

Peach Leaf Curl	Occurs in spring when the weather is cool and damp.	Curling leaves, loss of leaves and fruit
Peach Scab	Likely too late to treat once it appears in the tree.	Makes peach appear to have black scabs
Peach Mosaic Virus	Spread by mites.	Makes leaves appear to have a mosaic pattern
Spider Mites	Feed on tree foliage.	Silk webs spread throughout branches, loss of leaves

We hope that in capturing the spread of disease throughout trees of different shapes, our model can be used to inform pruning decisions by computing the shape that best mitigates the spread of this disease.

We will also attempt to model the immune response of a tree. This immune response is known as the compartmentalization of decay in trees (CODIT). Once a tree cell becomes infected, the tree does not attempt to save the infected cell. Instead, the tree tries to isolate the infected cell from the rest of the healthy cells. It has a rather intricate process to do this. It strengthens cell walls around the decay, then puts up specific barrier walls to prevent further spread, and then finally new growth is able to grow around the barrier walls. This process is what caused knots in wood or large hollowed out trees that are still alive. See the image below to get an idea of what this looks like (Shigo).



Literature review

Huang et. al’s paper (2012) discusses the implementation of cellular automata (CA) to model the spread of pine wilt disease (PWD), simulating the impacts of neighborhood size and deforestation rate on its severity. The authors choose a two-dimensional raster type for the lattice structure (10m x 10m grid), five vegetation types representing cell states, and a transition function using a yearly time-step. Their case study simulating neighborhood size and deforestation rate on the spread of PWD is highly effective, showcasing that PWD’s strength is correlated with the range of the neighborhood and concludes that a deforestation rate of 60% would be effective in stopping the spread of PWD. Although they state that their CA model should be improved in their transition function, we can use a similar approach (in terms of the CA model) at a smaller scale. Instead of neighborhoods of trees, we will examine neighborhoods of individual fruit tree branches using a shorter time-step.

Anderson and Dragicevic (2016) implement a more complex model involving geographic information systems (GIS), multi-criteria evaluation (MCE), and CA to investigate the infestation of the emerald ash borer (EAB), a type of jewel beetle, on ash trees in North America. This was done to overcome the limitations of previous EAB simulations, which use traditional statistical approaches that often do not address the true spatiotemporal complexity of these infestations. Their simulations take into consideration several types of data, both traditional and hypothetical, ranging from tree distribution to weather and meteorological data, resulting in the components of their model composition—tree susceptibility, spatial dynamics, landscape scenarios, and climate scenarios. Their results validate that their model provides a framework for the representation of EAB infestation as a non-linear, complex system, and show that urban landscapes in particular are quite susceptible to infestation. The authors prove this with a lack of data, and their model has the potential to be used in several “what-if" scenarios. Our model is similar; as there are few studies simulating the spread of disease across individual fruit tree branches, we must rely on hypothetical data at this point in time.

Gronewold and Sonnenschein (1997) give an example for the modeling of ecological systems with cellular automata based on the description of cell's behavior by Petri nets. This study was done in order to overcome the disadvantages of individual-oriented models, which are difficult to analyze using mathematical methods and are generally defined in an informal way. Petri nets are a formal modeling technique with event-based, concurrent, asynchronous state changes where graphical symbols are used for the description of states' respective state changes. This allows for the formal description of cellular automata with asynchronous cells' behavior within one synchronous time phase of the automaton (i.e., within one time phase of the whole automaton, many asynchronous `mini-steps' of cells can appear). An obvious advantage of this technique is the graphical representation that allows us to describe systems in a detailed but clear manner. Furthermore, a system modeled by a Petri net has formal semantics such that the interpretation of the system description cannot be ambiguous. Gronewold and Sonnenschein then look for a special description language for ecological systems based on Petri nets so that an easy modeling and quick simulation of ecological systems will be possible. They claim that modelers should be able to define only the behavior of one cell, the way of communication between cells, and the necessary synchronization points. After having made these definitions, the tool should be able to generate the structure of the cellular automaton automatically.

The main difference between our project and the work outlined in these articles is the scope of our simulation. Rather than modeling the spread of disease among individual entities, we are modeling the spread of disease among different parts of a single entity, our entity being a fruit tree.

Conceptual model

The basic conceptual model involves modeling disease spread using a cellular automata system. We plan to procedurally create a bunch of trees, run the CA model on the trees for a certain number of trials and steps, and find the optimal shape of the tree based on a scoring metric, which we defined as the percentage of infected or dead tiles over the total number of tree tiles. This metric can be described by the following equation:

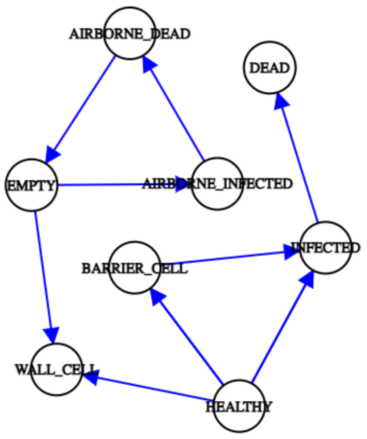
$$score = \frac{(\#\, infected\ tiles) + (\#\, dead\ tiles)}{(\#\, infected\ tiles) + (\#\, dead\ tiles) + (\#\, healthy\ tiles)}$$

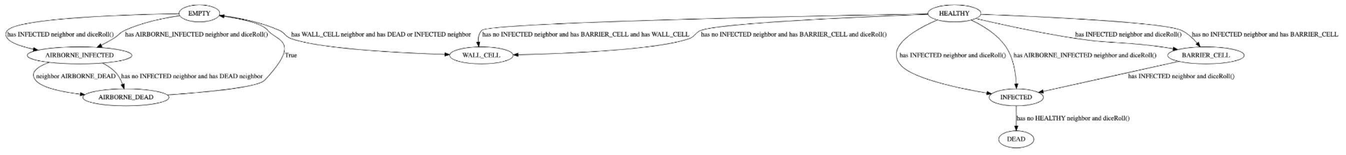
We will represent the tree by a NxN grid, in which each contains an integer representing the state of that cell. The tree can be thought of as the cells in the grid that do not have a state of empty.

Here we are assuming that the tree can be modeled by a 2D drawing of a tree. We further assume that the tree model can be discretized into pixels. We will see that this 2D assumption may affect how closely our CA model resembles how plants fight viruses.

We created our model with eight total states. These states are as follows: [EMPTY, HEALTHY, INFECTED, DEAD, BARRIER_CELL, WALL_CELL, AIRBORNE_INFECTED, AIRBORNE_DEAD]. The states INFECTED and DEAD are meant to model the disease. The state HEALTHY corresponds to healthy tree wood. The states AIRBORNE_INFECTED and AIRBORNE_DEAD are intended to allow the disease to spread to adjacent branches. It can be thought of as the disease being transmitted through the air by some means. The states BARRIER_CELL and WALL_CELL are meant to model the CODIT immune response of the tree, where the BARRIER_CELL acts as a buffer in which disease travels slower, and WALL_CELL acts as a wall to prevent infected cells from spreading entirely.

We created the transition rules to model the spread of decay and the tree's immune response to it to resemble the real life process of CODIT as best we could given the 2D constraints. Two diagrams of these transition rules are listed below as well as the rules.

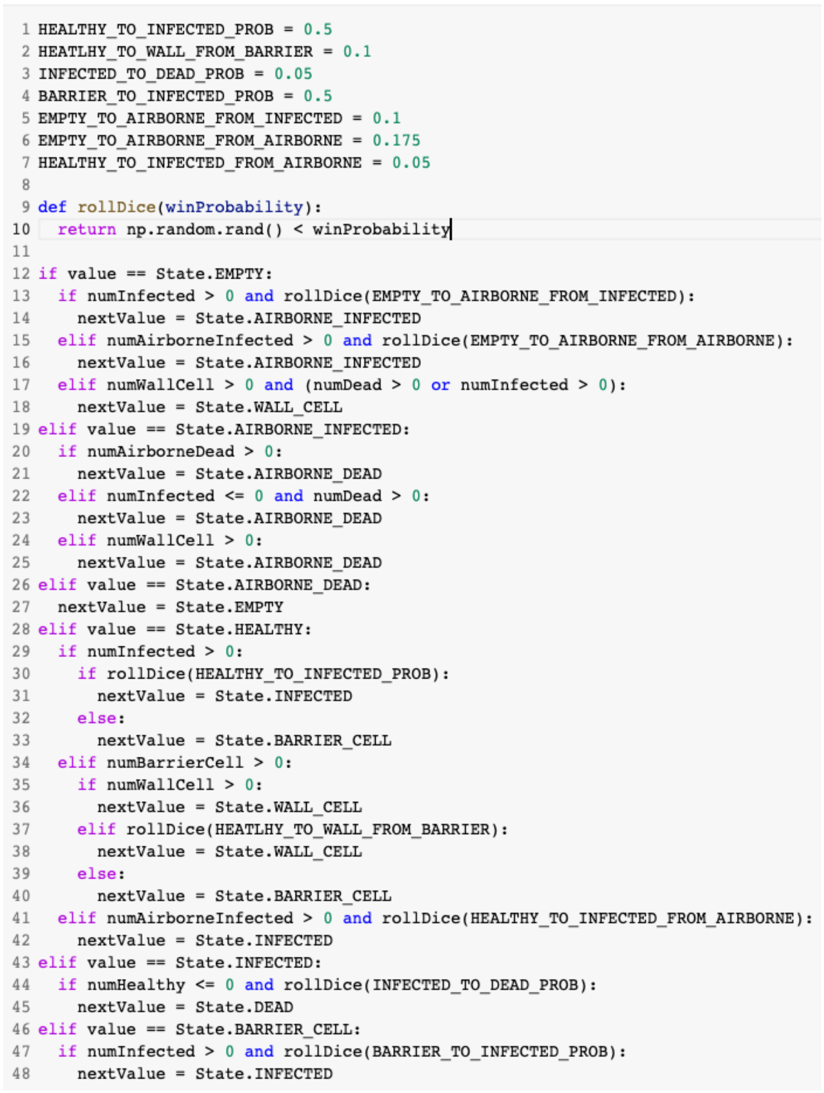



EMPTY
has INFECTED neighbor and diceRoll()
has AIRBORNE_INFECTED neighbor and diceRoll()
has WALL_CELL neighbor and has DEAD or INFECTED neighbor
AIRBORNE_INFECTED
True
neighbor AIRBORNE_DEAD
has no INFECTED neighbor and has DEAD neighbor
AIRBORNE_DEAD
WALL_CELL
HEALTHY
has no INFECTED neighbor and has BARRIER_CELL and has WALL_CELL
has no INFECTED neighbor and has BARRIER_CELL and diceRoll()
has INFECTED neighbor and diceRoll()
has no INFECTED neighbor and has BARRIER_CELL
has INFECTED neighbor and diceRoll()
has AIRBORNE_INFECTED neighbor and diceRoll()
BARRIER_CELL
has INFECTED neighbor and diceRoll()
INFECTED
has no HEALTHY neighbor and diceRoll()
DEAD

- **EMPTY -> AIRBORNE_INFECTED** "when cell has INFECTED neighbor and diceRoll(EMPTY_TO_AIRBORNE_FROM_INFECTED)"
- **EMPTY -> AIRBORNE_INFECTED** "when cell has AIRBORNE_INFECTED neighbor and diceRoll(EMPTY_TO_AIRBORNE_FROM_AIRBORNE)"
- **EMPTY -> WALL_CELL** "when cell has WALL_CELL neighbor and has DEAD or INFECTED neighbor"
- **AIRBORNE_INFECTED -> AIRBORNE_DEAD** "when cell has neighbor AIRBORNE_DEAD"
- **AIRBORNE_INFECTED -> AIRBORNE_DEAD** "when cell has no INFECTED neighbor and has DEAD neighbor"
- **AIRBORNE_DEAD -> EMPTY**
- **HEALTHY -> INFECTED** "when cell has INFECTED neighbor and diceRoll(HEALTHY_TO_INFECTED_PROB)"
- **HEALTHY -> BARRIER_CELL** "when cell has INFECTED neighbor and diceRoll(1 - HEALTHY_TO_INFECTED_PROB)"
- **HEALTHY -> WALL_CELL** "when cell has no INFECTED neighbor and has BARRIER_CELL and has WALL_CELL"
- **HEALTHY -> WALL_CELL** "when cell has no INFECTED neighbor and has BARRIER_CELL and diceRoll(HEATLHY_TO_WALL_FROM_BARRIER)"
- **HEALTHY -> BARRIER_CELL** "when cell has no INFECTED neighbor and has BARRIER_CELL"
- **HEALTHY -> INFECTED** "when cell has AIRBORNE_INFECTED neighbor and diceRoll(HEALTHY_TO_INFECTED_FROM_AIRBORNE)"
- **INFECTED -> DEAD** "when cell has no HEALTHY neighbor and diceRoll(INFECTED_TO_DEAD_PROB)"

- **BARRIER_CELL -> INFECTED** "when cell has INFECTED neighbor and diceRoll(BARRIER_TO_INFECTED_PROB)"

Here is also a code snippet of the rules that get applied to each cell. The variable "value" is the initial value of the cell. The variable "nextValue" is the next value of the cell. The variables with the structure "numX" correspond to the number of cells of state "X" within the current cell's Moore neighborhood.



```python
HEALTHY_TO_INFECTED_PROB = 0.5
HEATLHY_TO_WALL_FROM_BARRIER = 0.1
INFECTED_TO_DEAD_PROB = 0.05
BARRIER_TO_INFECTED_PROB = 0.5
EMPTY_TO_AIRBORNE_FROM_INFECTED = 0.1
EMPTY_TO_AIRBORNE_FROM_AIRBORNE = 0.175
HEALTHY_TO_INFECTED_FROM_AIRBORNE = 0.05

def rollDice(winProbability):
  return np.random.rand() < winProbability

if value == State.EMPTY:
  if numInfected > 0 and rollDice(EMPTY_TO_AIRBORNE_FROM_INFECTED):
    nextValue = State.AIRBORNE_INFECTED
  elif numAirborneInfected > 0 and rollDice(EMPTY_TO_AIRBORNE_FROM_AIRBORNE):
    nextValue = State.AIRBORNE_INFECTED
  elif numWallCell > 0 and (numDead > 0 or numInfected > 0):
    nextValue = State.WALL_CELL
elif value == State.AIRBORNE_INFECTED:
  if numAirborneDead > 0:
    nextValue = State.AIRBORNE_DEAD
  elif numInfected <= 0 and numDead > 0:
    nextValue = State.AIRBORNE_DEAD
  elif numWallCell > 0:
    nextValue = State.AIRBORNE_DEAD
elif value == State.AIRBORNE_DEAD:
  nextValue = State.EMPTY
elif value == State.HEALTHY:
  if numInfected > 0:
    if rollDice(HEALTHY_TO_INFECTED_PROB):
      nextValue = State.INFECTED
    else:
      nextValue = State.BARRIER_CELL
  elif numBarrierCell > 0:
    if numWallCell > 0:
      nextValue = State.WALL_CELL
    elif rollDice(HEATLHY_TO_WALL_FROM_BARRIER):
      nextValue = State.WALL_CELL
    else:
      nextValue = State.BARRIER_CELL
  elif numAirborneInfected > 0 and rollDice(HEALTHY_TO_INFECTED_FROM_AIRBORNE):
    nextValue = State.INFECTED
elif value == State.INFECTED:
  if numHealthy <= 0 and rollDice(INFECTED_TO_DEAD_PROB):
    nextValue = State.DEAD
elif value == State.BARRIER_CELL:
  if numInfected > 0 and rollDice(BARRIER_TO_INFECTED_PROB):
    nextValue = State.INFECTED
```

We modeled many transitions using probabilities. The values of these probabilities are given above and were mostly chosen based on educated guesses, trial and error, and the sparse data we had available regarding how decay spreads in trees. Field research is likely needed to make these values more closely resemble the real life process. For the time being, we decided on a time step of one day.

Simulation

Tree Generation

We use the GenProcTrees package to procedurally generate trees. It makes use of the “Space Colonization Algorithm” to create lifelike trees.

A common way to generate trees is using L-systems. An L-system or Lindenmayer system is a parallel rewriting system and a type of formal grammar. An L-system consists of an alphabet of symbols that can be used to make strings, a collection of production rules that expand each symbol into some larger string of symbols, an initial "axiom" string from which to begin construction, and a mechanism for translating the generated strings into geometric structures. With L-Systems, we can get some interesting looking trees, which typically start from the root, and generate branches based on a set of rules. The Space Colonization algorithm takes a different approach to the problem. Instead you first create the leaves, which serve as attraction points for the branches. Over each iteration, the branches grow closer and closer towards the leaves. In this manner, you end up having tree branches that appear to have grown towards some light source.

The algorithm works as follows:

1. Define an area for the crown of the tree. In our examples, we are using Rectangles, but any shape can be used.
2. Populate the defined area with attraction points (i.e. leaves).
3. Create the trunk of the tree, by adding Branches below the defined area. Keep growing branches upwards until the MaxDistance between a Leaf and a Branch is reached. This MaxDistance is a parameter that defines how far a Leaf can be to attract a Branch. A Branch is not affected by Leaves that are further away than MaxDistance.
4. Process the Leaves by comparing it to all the Branches. Calculate the direction and distance from the Leaf to the Branch. If the distance is smaller than MinDistance, we remove the Leaf for it has been reached. If the distance is greater than MaxDistance, we ignore it, since it is too far. Otherwise, we check if the Branch is the closest Branch to this Leaf. Each Leaf can only affect 1 Branch at a time.
5. Once the closest Branch is determined, we increment the GrowCount of that Branch, and add the direction of the Leaf to the GrowDirection of the Branch. If multiple Leaves attract a branch, then the GrowDirection will be an average of all of the Leaf directions.
6. Now loop through the Branches, and process any Branch with a GrowCount > 0. Divide the GrowDirection by the GrowCount, to get the average direction, then create a new branch with this GrowDirection, linking it to the Branch being processed as its parent. Then reset the GrowCount and GrowDirection of the parent Branch.

7. Repeat from step 4 until there are no Leaves left, or no more Branches are growing.

To generate these trees, we need a set of parameters that define the number of leaves, their placement, and attributes of the branches. These parameters include:

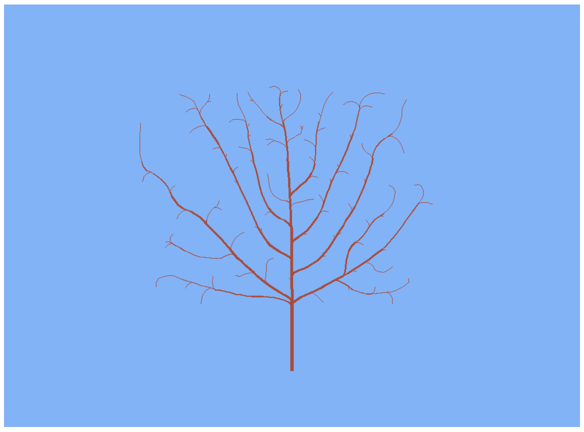
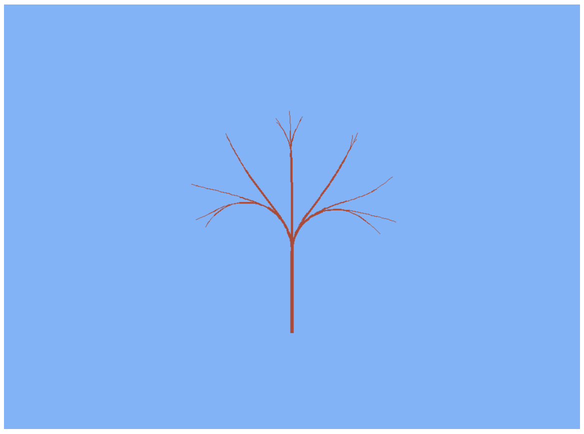
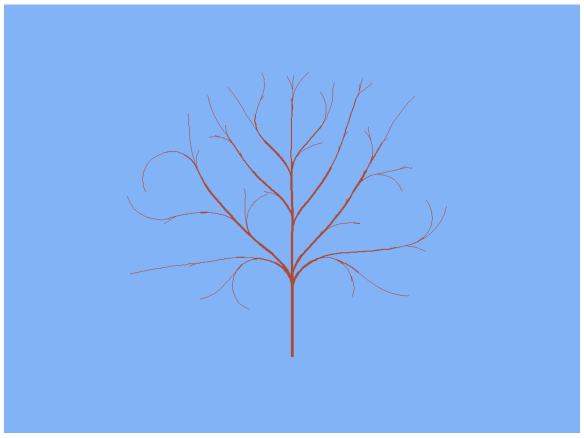
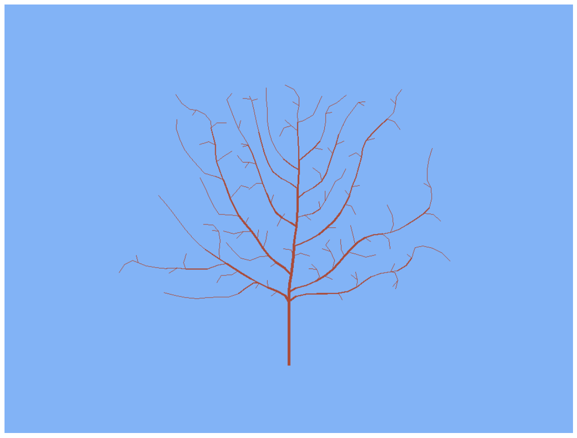
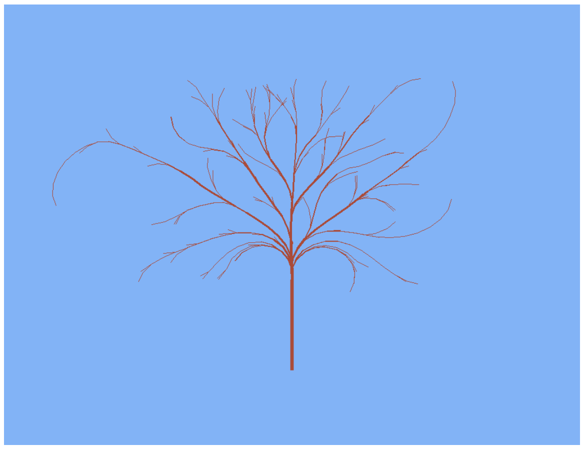
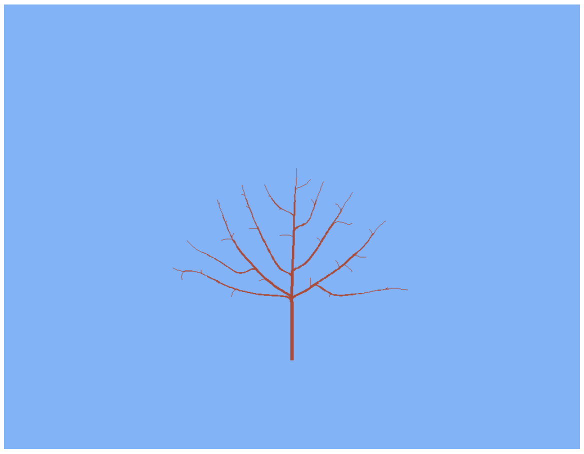
1. min_distances: minimum distance allowed between two leaves
2. max_distances: minimum distance allowed between two leaves
3. branch_lengths: maximum length of the branches
4. turn_factors: how often the branches turn, separates lengthy trees from wiry trees
5. leaf_starts: how far up the stem do leaves start
6. number_of_leaves: total number of leaves in the tree

We use a uniform distribution to generate 6 different possible values for all 6 of these parameters, and find all the possible combinations of these values to get the set of all possible configurations. In total, we have 6^6 = 46,656 sets of parameters to choose from when generating a tree. Following table shows the range and increments of each variable:

Variable	Minimum value	Maximum value	Increment
min_distances	0.02	0.05	0.005
max_distances	0.2	0.5	0.05
branch_lengths	0.01	0.075	0.005
turn_factors	0.06	0.64	0.05
leaf_starts	0.20	0.37	0.03
number_of_leaves	310	400	15

Given *n* number of trees to be generated, we randomly choose n configurations without replacement and use them as parameters to generate the trees. GenProcTrees generates the tree in the form of an image, which we then convert to an array to pass through our CA model. In the initial tree array, 0 refers to an empty cell (or pixel) and 1 refers to a branch.

Some examples of the trees generated:

CA

Using our conceptual model and the parameter combinations outlined above, we created 20 trees of varying shapes, then ran three trials of our CA implementation on each tree. For each trial, we calculated the percentage of the tree that ended up dead or infected after 100 timesteps, then averaged these scores to get an overall score for the tree.

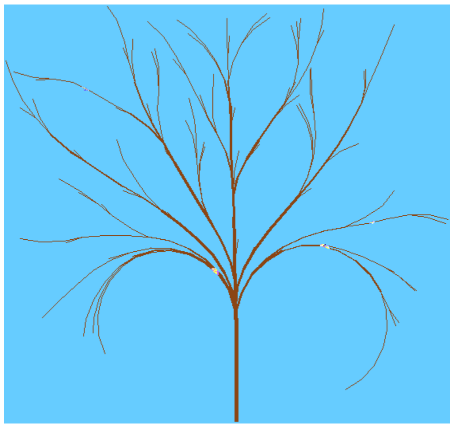
Each tree begins completely healthy, but disease is introduced either at random locations – simulating pest-induced diseases, for example – or at the roots – a starting place for root rot.

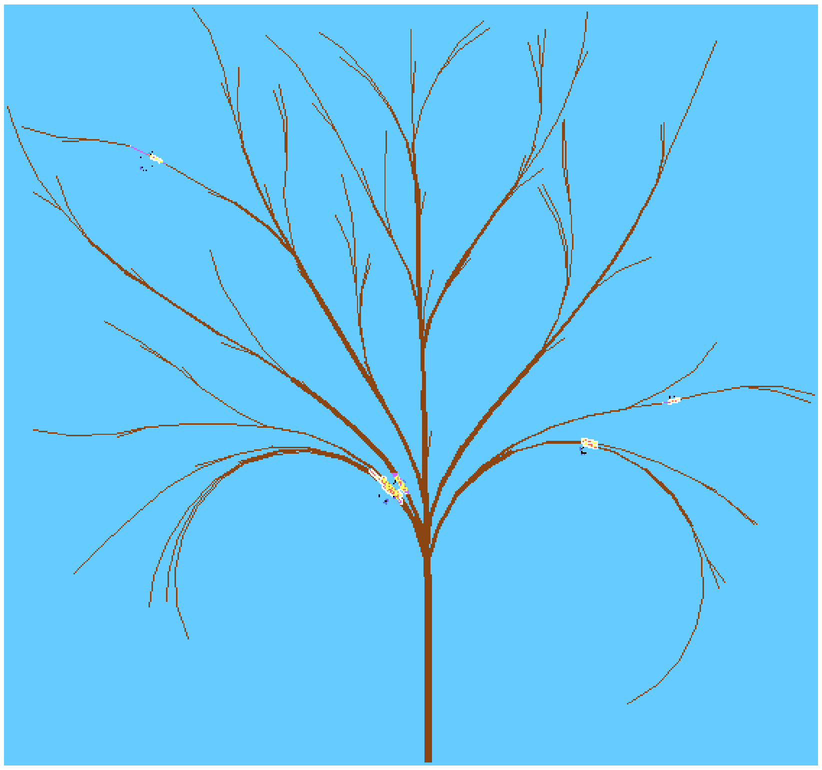
Visualization

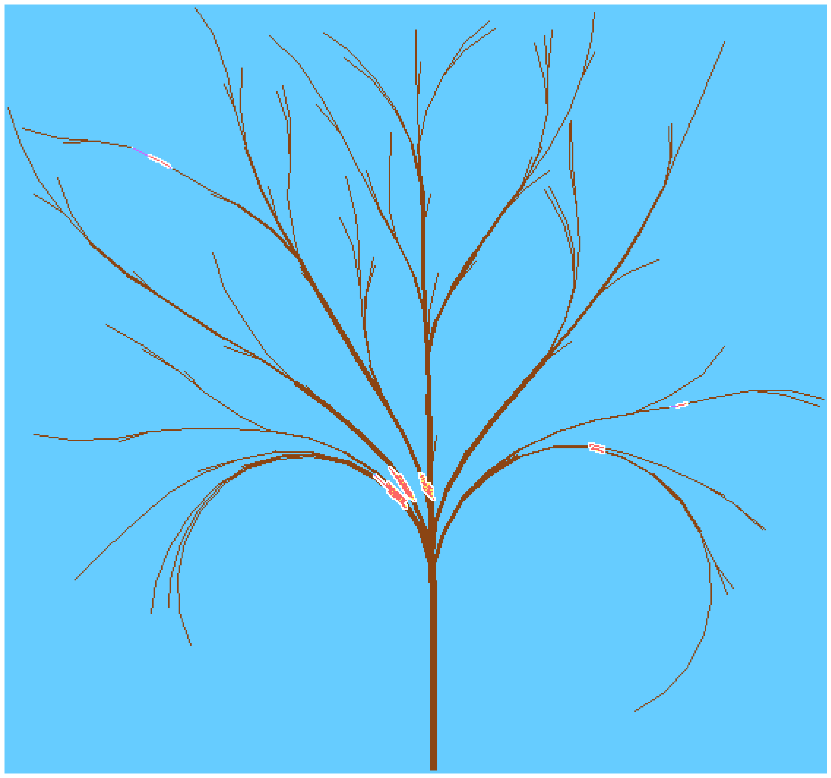
After each step of the simulation, we map the cell values (or pixel values) to the following colors in order to visualize the state of the tree.

Pixel Value	State	Color
0	Empty	Blue
1	Healthy	Brown
2	Infected	Yellow
3	Dead	Red
4	Barrier Cell	Lavender
5	Wall Cell	White
6	Airborne Infected	Orchid
7	Airborne Dead	Black

Below are some examples:







In addition to this, we also create a GIF to summarily visualize the evolution of a tree through different steps in the CA model.

Experimental results and validation

To obtain quantitative results for our model, we run 20 different simulations, where each simulation is run on a different tree shape. The input parameters for tree generation are tweaked in each trial, ensuring that several types of tree growth and branching are accounted for. As can be seen from above, each tree has a different height, width, number of leaves, branch length, etc. For each simulation, the "score" of the tree, or the percent of pixels that were infected by the disease, (as stated in the Conceptual Model section) is calculated and ranked. It can be seen that trees with branches with long branches, large turn factors, and a smaller number of leaves are less susceptible to disease spread than trees with shorter branches, a larger number of leaves, and orthogonal branches. Our full results can be seen in the table below.

Rank	Score	Tree ID	min_distances	max_distances	branch_lengths	turn_factors	leaf_starts	number_of_leaves	Height	Width
1	0.0237295	5	0.03	0.4	0.065	0.61	0.32	310	514	603
2	0.02683912	18	0.02	0.35	0.065	0.61	0.32	355	508	574
3	0.02723403	11	0.02	0.35	0.06	0.36	0.32	310	522	514
4	0.02968247	13	0.03	0.4	0.07	0.56	0.23	340	520	615
5	0.03164093	7	0.03	0.35	0.055	0.56	0.26	340	511	540
6	0.03207973	2	0.04	0.3	0.02	0.31	0.35	385	506	609
7	0.03246206	14	0.035	0.25	0.02	0.31	0.29	370	507	587
8	0.03496937	15	0.025	0.25	0.02	0.21	0.2	355	511	639
9	0.03736436	8	0.03	0.45	0.02	0.11	0.32	385	515	667
10	0.03921376	4	0.045	0.25	0.025	0.41	0.26	340	505	519
11	0.04079978	6	0.045	0.35	0.04	0.36	0.26	340	523	469
12	0.04365584	1	0.03	0.35	0.03	0.21	0.29	370	516	585
13	0.04546537	12	0.045	0.25	0.055	0.31	0.23	340	524	599
14	0.04568995	3	0.045	0.3	0.015	0.61	0.23	340	445	519
15	0.04995473	17	0.02	0.4	0.07	0.36	0.2	370	525	558
16	0.05113221	10	0.02	0.35	0.045	0.21	0.32	340	526	642
17	0.05360277	19	0.045	0.4	0.035	0.41	0.29	310	510	510
18	0.06114744	16	0.02	0.4	0.01	0.36	0.32	340	404	427
19	0.0655192	0	0.045	0.4	0.04	0.11	0.32	355	531	672
20	0.07421116	9	0.045	0.4	0.05	0.26	0.26	325	526	574

Our access to validation data, though, was sparse. As our model is novel- there are no publicly available simulations dealing with disease spread on individual trees- there exist only approximated statistics to compare its results with. The University of California's Integrated Pest Management Program (UC IPMP) states that blight bacteria can spread up to 3 feet from its point of infection on a tree's surface (2019).

Using this statistic, we can validate our study with some clever calculations. We assume that each pixel in each image represents 0.025 feet. To get the surface area (in ft^2) of each tree, we multiply its height by its width (after converting pixels to feet). Then, to get the total area of the tree infected, we multiply the surface area of the tree by its score. From this, we can get the average spread of infection width (average infection size) by dividing by 5 (number of infection sites per tree on average). The workflow for this is as follows:

$(1)\ Tree\ surface\ area\ =\ (height\ *\ 0.025)\ *\ (width\ *\ 0.025)$
$(2)\ Total\ area\ infected\ =\ (Tree\ surface\ area)\ *\ score_{tree}$

$(3)\ Average\ infection\ area\ =\ (Total\ area\ infected)\ /\ 5$

The area per infection for each ranked tree can be seen in the table below.

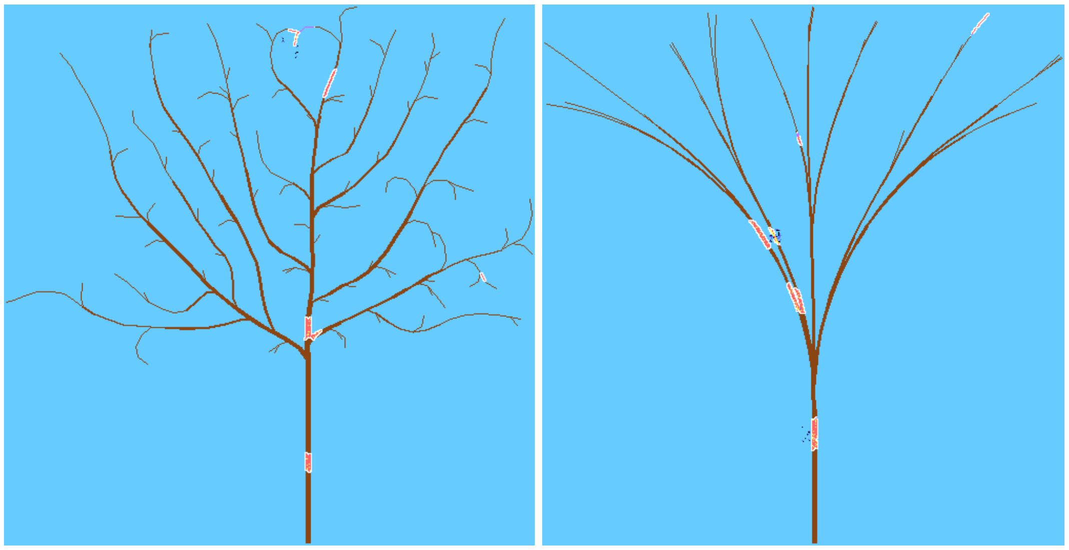
Tree Rank	**Area per infection (ft^2)**
1	0.91934628
2	0.9782591
3	0.91338863
4	1.18655658
5	1.09137463
6	1.23568714
7	1.20762505
8	1.42731408
9	1.60435576
10	1.28471641
11	1.25095693
12	1.64724412
13	1.78381089
14	1.31904032
15	1.82927959
16	2.15836748
17	1.74276011
18	1.31855274
19	2.92241838
20	2.80076634

In 17 of our 20 trial simulations (85%), we observed that infections, on average, spread between 1 and 3 feet from their points of origin. A 99% confidence interval on our data is [1.21, 1.85], indicating that the majority of trees in our system will have average infection sizes of between 1.21 and 1.85 ft^2, which is within the range stated by UC IPMP.

Despite the lack of resources available to validate our system, we *can* say that our simulations were roughly valid as there is a correlation between our hypothetical results and real-life measured data.

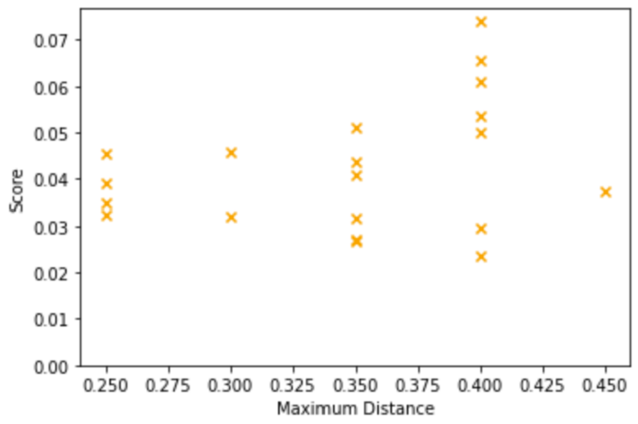
Discussion and conclusions

Based on the scored trees, the most noticeable observation is that trees that have child branches that fork more orthogonally tend to have better scores. For example, in one simulation run, the tree below on the left had the best score of 3.41% infected and the one on the right had the worst score of 7.91% infected.

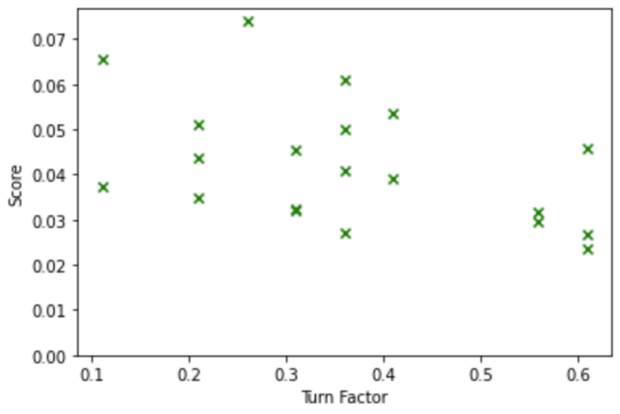


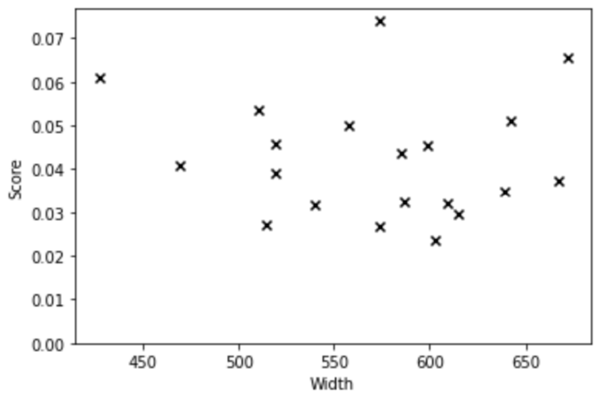
However, the scoring metric may be biased in favor of trees that are larger and start with more "healthy" cells.

We also found that maximizing the distance between branches seems to be ideal, though there is a lot of variance in our results, so further investigation may be required.



Otherwise, looking at the individual impact of tree shape properties was not very informative. Below are Score vs. Property plots for turn factor and tree width respectively.





Evaluating the effectiveness of tree shape turned out to be a quite difficult task, requiring visual comparison in addition to parameter comparison. In future iterations, we would hopefully be able to develop other metrics of scoring, which can be refined as more real data becomes available on fruit tree infection spread.

Future Work

One virtue of our model is that it is easily extensible; simulating other specific diseases requires only a few modifications and a new CA child class. However, many of the diseases that one may want to simulate involve infection of not only the branches, but also the roots and leaves of a tree. To support a wider scope of disease, it would be prudent to add leaves and a root system to our model and visualizations.

References

Anderson, Taylor, and Suzana Dragicevic. “A Geosimulation Approach for Data Scarce Environments: Modeling Dynamics of Forest Insect Infestation across Different Landscapes.” *ISPRS International Journal of Geo-Information*, vol. 5, no. 2, 2016, https://www.mdpi.com/2220-9964/5/2/9.

“Compartmentalization of Decay in Trees.” *Wikipedia*, 10 Mar. 2022, en.wikipedia.org/wiki/Compartmentalization_of_decay_in_trees.

Douglas, Sharon M. “Pruning: An Introduction to Why, How, and When.” *The Connecticut Agricultural Experiment Station*, 2022, https://portal.ct.gov/CAES/Fact-Sheets/Plant-Pathology/Pruning-An-Introduction-to-Why-How-and-When.

“Fruit Tree Forms.” *Wikipedia*, 8 Nov. 2021, en.wikipedia.org/wiki/Fruit_tree_forms. Accessed 30 Mar. 2022.

Gronewold, Anja, and Michael Sonnenschein. “Event-based modelling of ecological systems with asynchronous cellular automata.” *Gale Research*, 1997, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.9347&rep=rep1&type=pdf.

Huang, Mingxiang, et al. “A Cellular Automata Model of Pine Wilt Disease Spread and Its Simulation Tool for Application.” *National Conference on Information Technology and Computer Science*, 2012, https://www.atlantis-press.com/article/3195.pdf.

Shigo, Alex L. “Compartmentalization of Decay in Trees.” *Scientific American. 152 (4): 96-103.*, vol. 152, 1985, pp. 96–103.

Teviotdale, B L. “How to Manage Pests.” *UC IPM Online*, 2019, http://ipm.ucanr.edu/PMG/PESTNOTES/pn7414.html.

“Trichovirus.” *Wikipedia*, 16 May 2021, en.wikipedia.org/wiki/Trichovirus. Accessed 30 Mar. 2022.

University of Georgia CAES. “Diseases - Research | Peaches.” *Peaches.caes.uga.edu*, University of Georgia, peaches.caes.uga.edu/research/diseases.html. Accessed 30 Mar. 2022.

Appendix

Distribution of Labor:

Allison

- CA model
- Optimum tree analysis
- Video Presentation

Fletcher

- CA model
- Optimum tree analysis
- Immune response consideration

Mihir

- Generation of tree shapes
- Visualizing the model
- CA Integration

Shoale

- Generation of tree shapes
- Visualizing the model
- Model Validation