Will Solow and Skye Rhomberg

CS346 – Final Project

Writeup on Wildfire Simulations

To tackle simulating a wildfire, we chose to start small and build up, adding complexities to the model with each iteration of the code. As such, this writeup will be presented in group of exercises, explaining the assumptions made for each addition to the model.

1.

To start simulating a basic wild fire, we followed the book's model with parameters for the probability of a tree at a cell, the probability that the tree is burning, and the probability that the tree is immune to fire at a give time step. Additionally, the cell being empty, having a tree, or a burning tree was simulated using three different states. To ensure consistent results, the seed of 1 was used throughout all simulations.

We start the simulation by initializing the forest using the number of rows and columns (i.e. length and width). We create a random grid the size of our forest with doubles in the range (0,1). Then, we create a boolean array of 0's and 1's for if each cell has a tree. If the index in the random array is less than the product of the tree probability and burning probability, we then assign a 1 into another boolean array of 0's and 1's for the burning trees. By summing these two arrays up, we get an array of 0's, 1's and 2's to simulate our empty cells, trees, and burning trees.

We assume that the boundary of the forest is absorbent (i.e. the boundary is 0/empty) so that the fire cannot spread beyond those boundaries. This makes sense as in the real world there is an edge to the forest and so the fire generally cannot spread past that boundary as it would run out of things to burn on. Additionally, we use the Von Neumann neighborhood as we felt that it better represented tree proximity than a Moore neighborhood.

Our spread function controls the vast majority of the logic within the simulation. Instead of going through each cell and getting the neighbors and evaluating neighbor like the book suggests, we instead opted to create four offset arrays that represent the forest. Using the extended forest with the padded boundary, all the northern neighbors consist of removing the east-most and west-most columns from the extended grid and taking off the bottom row as well. This gives us an array the size of the initial forest and when we repeat this for the east, west, and south neighbors, we sum these four arrays to get the values of the neighbors.

Note that this is not enough to evaluate if the cell has a burning neighbor as 4 non-burning tree neighbors could evaluate to 2 burning neighbors and 2 empty cells. Thus, the arrays we created evaluate to 1 only if the neighbor is burning. By summing these, we only account for the burning neighbors and set ourselves up nicely for when we add complexities to how a tree catches fire.

As every burning tree burns to empty, the only remaining trees will be those that are trees at the current time step. Then, to find the cells that aren't immune to burning, we use a logical operator

on a forest sized random array to return an array of 1's and 0's. We then point-wise multiply this by a logical operator for if the burning neighbors array is greater than 0. This gives us an array of all the cells that can burn and are not immune. To wrap this up, our forest is equal to the remaining trees pointwise multiplied by the burning cells that are not immune plus the array of remaining trees. Thus, any tree that is now burning sums to 2, any unchanged tree stays at 1, and the burnt trees or empty cells go to empty.

We then graph this using some code very similar to the show_CA_list.m file demonstrated in class. We create our own three color colormap of brown, green, and orange for empty, tree, and burning tree. Then, by passing this function the list of forest arrays we are able to visualize the spread of the fire across multiple time steps.

2.

In the second iteration of our simulation, we add the complexities for tree growth and lightning strikes, as well as change in the model that trees would be more likely to catch on fire if more of their neighbors were on fire. Tree growth is .01 which gave a reasonable rate of reproduction for the rate that the fire spread and the lightning strike probability was .0001, as suggested in the book. The only changes necessary for this simulation were in the spread function.

As noted in 1., by summing the Von Neumann neighbors to obtain a sum of which ones are burning, we are able to multiply the immunity linearly such that if all 4 neighbors are burning there is a 100% chance that the inner tree will catch on fire. While in real life this might not be true, it seemed like a reasonable simplifying assumption in the model as a fully surrounded tree is very likely to start burning. This was done in the nonimmune_cells array calculation by increasing the threshold of the nonimmune probability linearly from its base value of 1-prob_immune, when no neighbors were burning to 1 when all four neighbors are burning.

Then, to add lightning strikes, we created another random array that has a logical operator to evaluate it to 1 if the cell was struck by lightning. As lightning cannot strike an already burning tree, the nonimmune burning cells was a pointwise product of the non-immune cells and the burning neighbors or the lightning strikes arrays.

Finally, to account for new tree growth, we employed a similar strategy as above by creating a random array and boolean evaluating it to less than prob_grow. The sum of all these arrays gives us the new forest array which is then returned to the main simulation loop.

3.

In the third iteration of the model we added wind and global rainfall. We chose to represent wind as a two-dimensional vector. We originally considered implementing wind using polar coordinates for strength and direction, however this was not giving us the results we wanted as non-cardinal direction wind created undesired effects. As such, we stuck with the vector-based model. To implement this, we added a second set of summations to the north, south, east and west partial grids using the wind direction as a x-y vector with x affecting the east-west spread and y affecting the north-south spread. A northern wind with vector (0, -1), meaning that the wind was blowing from the north with the vector facing south, meant that a southern burning

neighbor would have no impact on the probability that the current cell would burn, while the northern neighbor would have double the impact.

This was immediately noticeable in the simulation as a full-strength southern wind forced all the fires north right away with no southern spread. Obviously, a full-strength wind is not entirely realistic so any wind strength less than 1 will result in some spread in the opposite direction.

For this, we considered implementing Moore neighborhoods to simulate wind coming from the northeast, southeast, northwest, and southwest. However, it did not provide anything more meaningful to the model than the Von Neumann neighborhoods did, so we chose to not overly complicate the model in this regard. Moreover, as fire spreading is not exactly the same as diffusion, it made more sense to consider just the four direct neighbors which have more of a surface to spread the fire to. Additionally, we noticed that Moore neighborhoods caused the fire to spread in a more square-like fashion as opposed the more realistic circular/diamond fashion of the Von Neumann neighborhoods.

In addition to wind direction, we also added rainfall. The rainfall can be programmed into an array to last a certain period of steps. We thought about adding random rainfall, however we decided that giving the user more control over when the rain appears made more sense in the simulation. We decided to make rainfall global as we figured that a forest fire is much smaller than a weather pattern, and any rain would likely affect the entire area. As such, rainfall affects the entire forest uniformly by increasing the immunity rate of trees uniformly. This affect was toggled on and off in the simulation loop and made note of in another array to use when graphing to notify the user that rain was occurring.

The affects of rainfall were immediately noticeable, and the fires spread much slower and were sparser when the rain was on. Once the rain stopped the fires immediately ramped back up. While this is not entirely realistic, as we are simulating without a great degree of time accuracy, it is not unreasonable to assume that the ground dries in between time steps so the fires continue to burn with their normal rigor as opposed to a slow ramp up that we might see in continuous time in the real world.

4.

In our final iteration of the simulation, we implemented tree age. Trees have four ages, sapling; young; old; and ancient. To implement this, we had to drastically overhaul our simulation. Instead of simulating the forest with one array, we chose to simulate it with two. We had an array of 0’s and 1’s for which cells were currently burning and an array that stored the trees with their respective ages. To correctly calculate the burning neighbors in our offset grid described previously, we changed the extended forest grid to an extended burning grid and discarded the extended forest grid. This actually made the logic slightly simpler as we could directly sum the neighbors we created instead of testing if they had a value that matched burning.

This decision changed our init_forest function as we added a new constant called forest_age. The forest age simulates how old the forest is under the assumption that an older forest will have more ancient trees. With a value of forest_age of 1, every tree that is generated will be ancient

while with a value of 0 every tree will be a sapling. We operated under the assumption that each age of tree was exponentially less probable than its younger counterparts, and so we generated a random grid to test for all of these. Thus, we summed them to overlay the values into our forest with values 0 through 4. The burning was calculated the same way as in previous version and was returned in its own array.

As mentioned previously, the spread function changed drastically as well with the burning neighbors being calculated directly from the burning array. We decided that an ancient tree (with a value of 4) would burn in 4 steps while a sapling with a value of 1 would burn in 1 step. This makes our code immediately extendable if someone wants to complicate the simulation by adding more values of trees. Additionally, a tree cannot burn to 0 and grow a new sapling in the same time step, so which cells grow trees is calculated based off of which cells are currently empty within the time step.

The following logic gets slightly complicated as we need to ensure that a burning tree stays burning, but that when a tree burns down the burning array is updated to no longer have a 1 representing burning in that cell. We start by calculating the cells that can burn by creating a random array, dividing it by the oldest tree (4), and then pointwise multiplying it by the forest array. This ensures that the random array is in the range 0-1 while also ensuring that ancient trees are linearly less likely to burn than their younger counterparts. We then check if this new value is less than the immunity of the cell which is calculated off of the burning neighbors. If the cell has 4 burning neighbors, there is still a 100% chance that it will burn, regardless of the age of the tree. We were okay with this simplification as, regardless of age, it feels like that much fire should cause the tree to burn. Last, to ensure that only non-burning cells were in this calculation, we pointwise multiplied by the negation of the burning array.

To nicely implement the trees burning based on the assumption that an ancient tree burns for 4 steps, we track the tree burning down as it becomes younger by subtracting the burning array from the forest array. This means that if an ancient tree catches on fire it will burn to a sapling before becoming an empty cell in terms of the data in the array. While this is a little counter intuitive, as we are keeping track of which cells are burning, we will always know if a tree is burning and so it will not impact the other parts of our simulation.

Thus, the new burning cells was able to be calculated as before and the final burning array was the sum of the new burning cells and the burning array at time step i, pointwise multiplied by the burnt forest array. This meant that if any saplings burned to empty, the value in the burning array would be updated as that cell is no longer on fire.

To go along with this, we modified our tree growth with a function called rate_of_growth. We created function under the assumption that the older a tree got, the longer it would take to grow to the next size up. This was capped at a value of 4 so that an ancient tree would have a 0 percent likelihood of growing older. As it does not make sense than a burning tree can grow, this was pointwise multiplied by the burning array and then by a logical operator for the forest not being empty. This meant that the cells returned that were 1 would only be the cells where new growth occurred. Thus, we were able to nicely sum the burnt forest (current trees with some that burned

to a younger age) with the tree growth (only non-non burning trees that grew) with the new trees (empty cells that grew a tree). This completed the model and allowed us to fully simulate forest age.

To visualize this, we created a new script called show_forest_age. This script took in the forest list and burning list. We made a new color map with darker greens representing older trees. Any burning tree was the same color orange. To normalize our graph in the range [0,5] to account for the empty cells, burning cells, and 4 tree ages, we had some logic to combine the burning array with the forest array. This gave us a really nice view that was relatively easy on the eye while still showing how the forest burnt.

5. Code validation

To validate our code, we mainly focused on the logic in the spread function as it was by far the most complex part of the model. Rainfall was the easiest to validate as the effect was very visual. We were able to see how the spread of the fire dampened during the timesteps that rainfall was on. Wind direction was likewise easy to validate, and with a strong wind we were able to see that the fire spread in the direction of the wind. This was done by carefully keeping track of where the fires were at a certain time step and making sure that they were not spreading backwards when they should not be.

To track the burning of the cell, watched for when a tree caught on fire and then made sure that it stayed on fire for the correct amount of time steps before eventually turning to an empty cell. Additionally, we tracked that older trees seemed to be less likely to catch fire which was in line with our model. We had some mistakes with the immunity of trees along the way and in fixing it, we are confident that we have arrived at the correct logic.

5. Hypothesis testing

To look at the effectiveness of our model at predicting outcomes, we created a few hypotheses to test. Unless otherwise specified, rainfall and wind were assumed to not have an affect on the model. Clearly, in a more complicated simulation, these would always want to be present in some form, but for our purposes of model testing we do not always need to account for every variable as we are dealing with a simplified system. Also, we start with constants prob_tree = .6, prob_grow = .02, forest_growth = .1, forest_age = .5, prob_burning = .0005, and prob_lightning = .0001;

Hypothesis 1: An older forest will be less susceptible to burning down completely than a younger forest. To test this, we tested with forest_age values of 0, .5 and 1 to get the following:

Young forest:

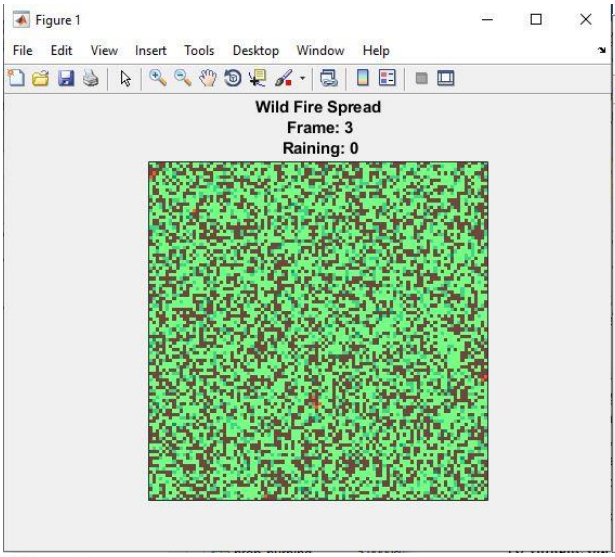


Figure 1
File Edit View Insert Tools Desktop Window Help
Wild Fire Spread
Frame: 3
Raining: 0

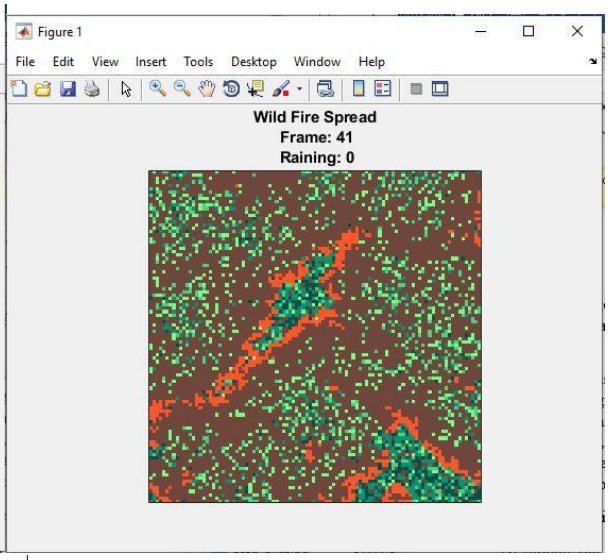


Figure 1
File Edit View Insert Tools Desktop Window Help
Wild Fire Spread
Frame: 41
Raining: 0

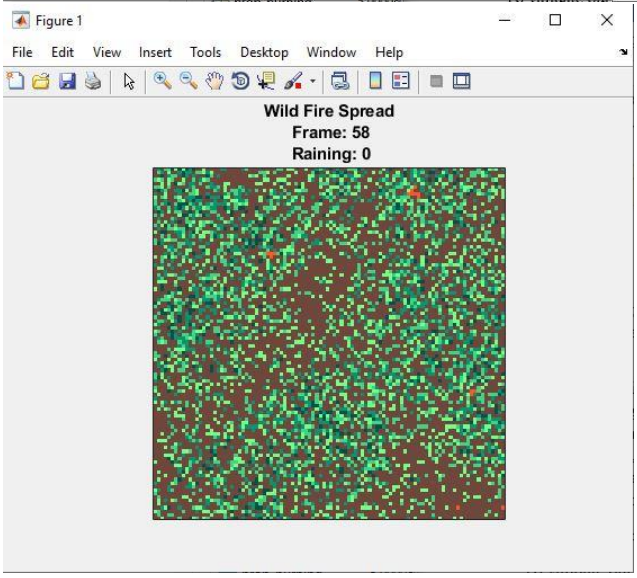
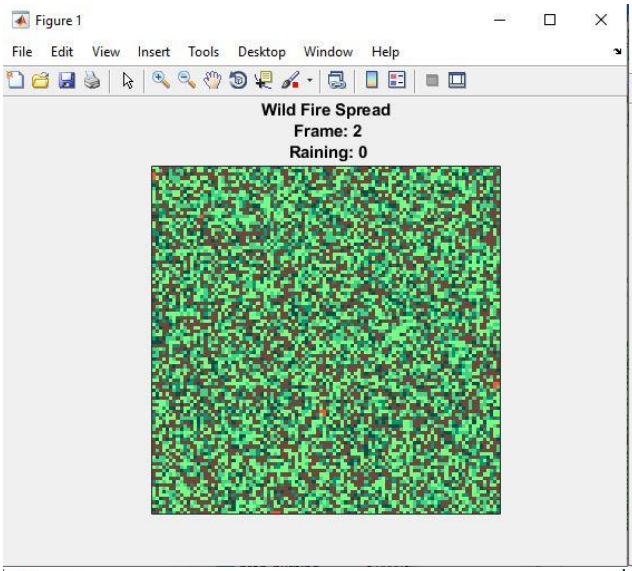
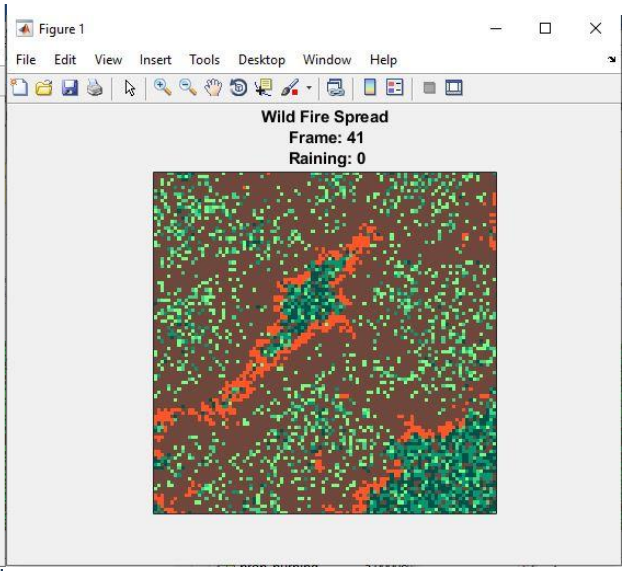
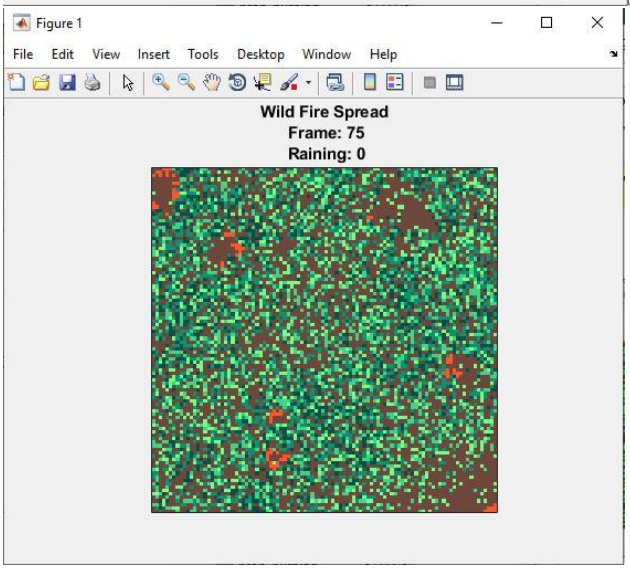


Figure 1
File Edit View Insert Tools Desktop Window Help
Wild Fire Spread
Frame: 58
Raining: 0

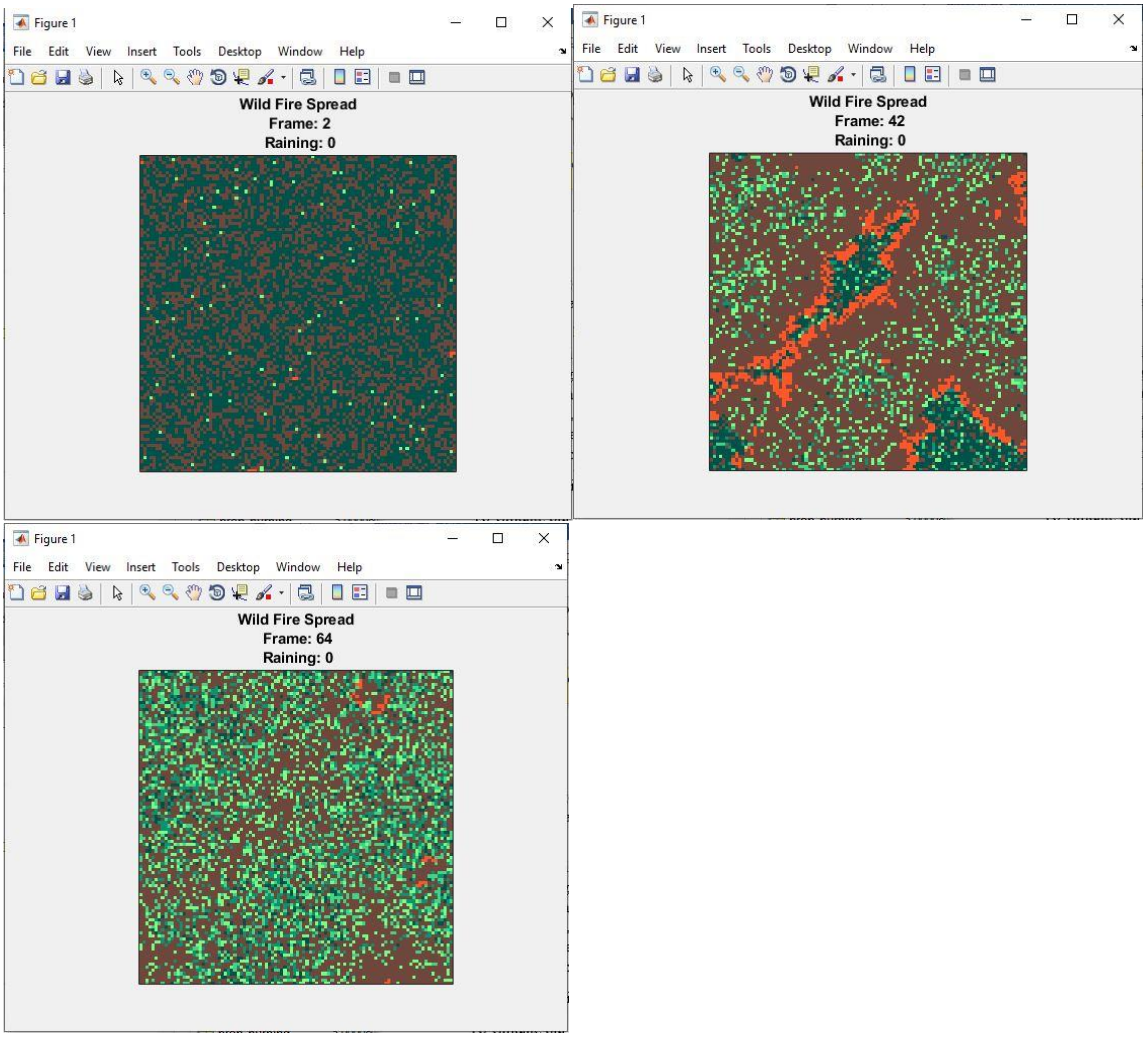
Old forest:







Ancient forest:

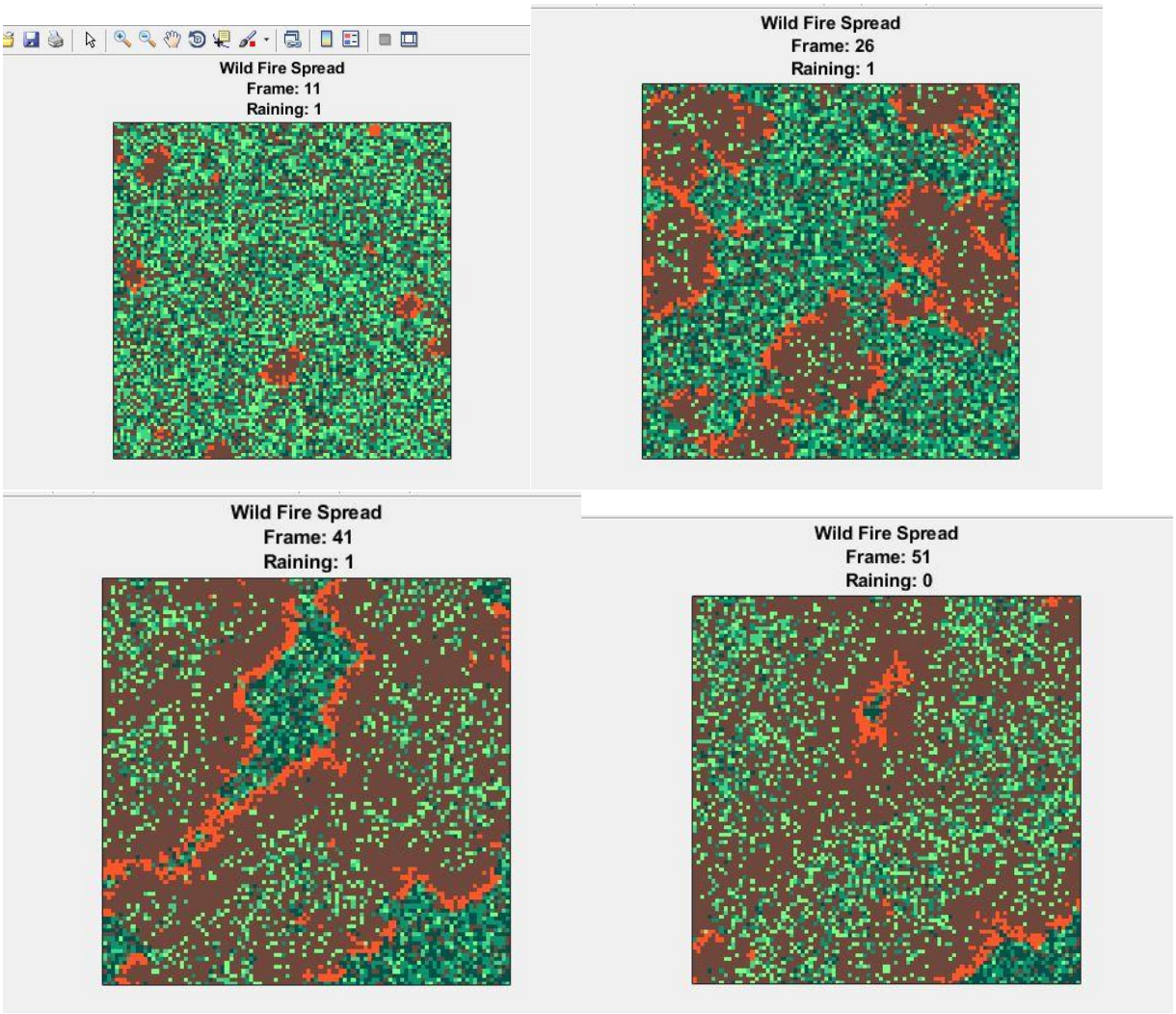


In these 3 simulations, we have the starting frame (or close to it, with some new growth), then a snapshot at about 40 frames, and then a snapshot after the large south east fire dies out at the bottom right of the map. Interestingly, the middle-aged forest took the longest to burn out and was also more densely populated at the point than either the young or old forests. As expected, the young forest was suffering the most devastation part way through the simulation, however it was interesting to see that the middle-aged forest actually was suffering less devastation in the south east quadrant than the ancient forest. This implies the importance of the presence of younger trees to continually grow. Additionally, as larger trees burn for a longer time, they obstruct the growth of new trees for a longer period so we see that it does make sense to have some underbrush in our forests.

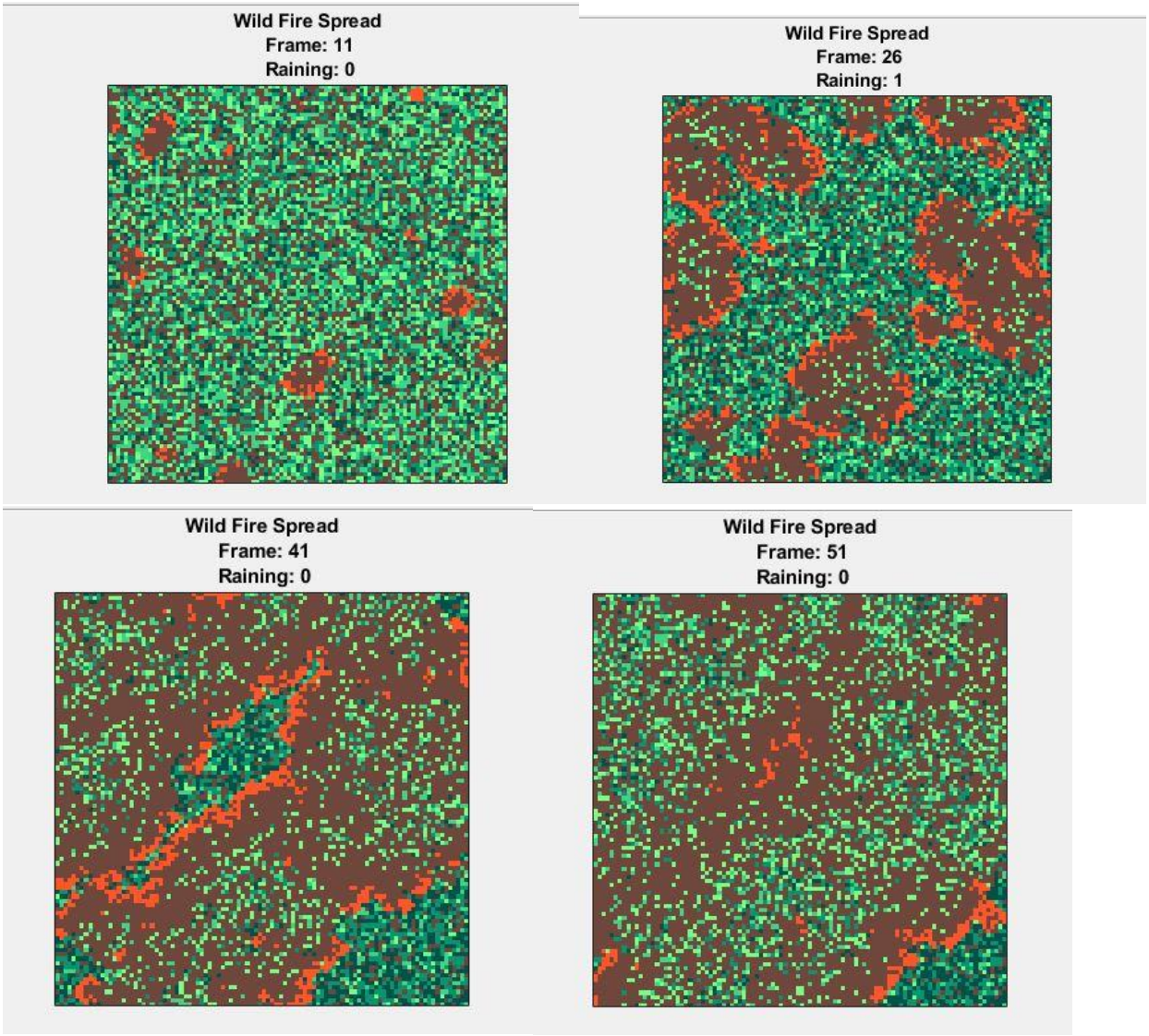
Hypothesis 2: Periodic rainfall for shorter periods of time is more effective at curbing the spread of forest fires than one long rainfall period.

This hypothesis looks at periodic rainfall versus seasonal rainfall, in a sense. To simulate it, we returned forest_age to .5 and then had a rainfall period from time steps 10-15, 25-30, and 40-45 and then a second simulation with rainfall just from 20-35. The results are below:

Period rainfall:



Seasonal rainfall:

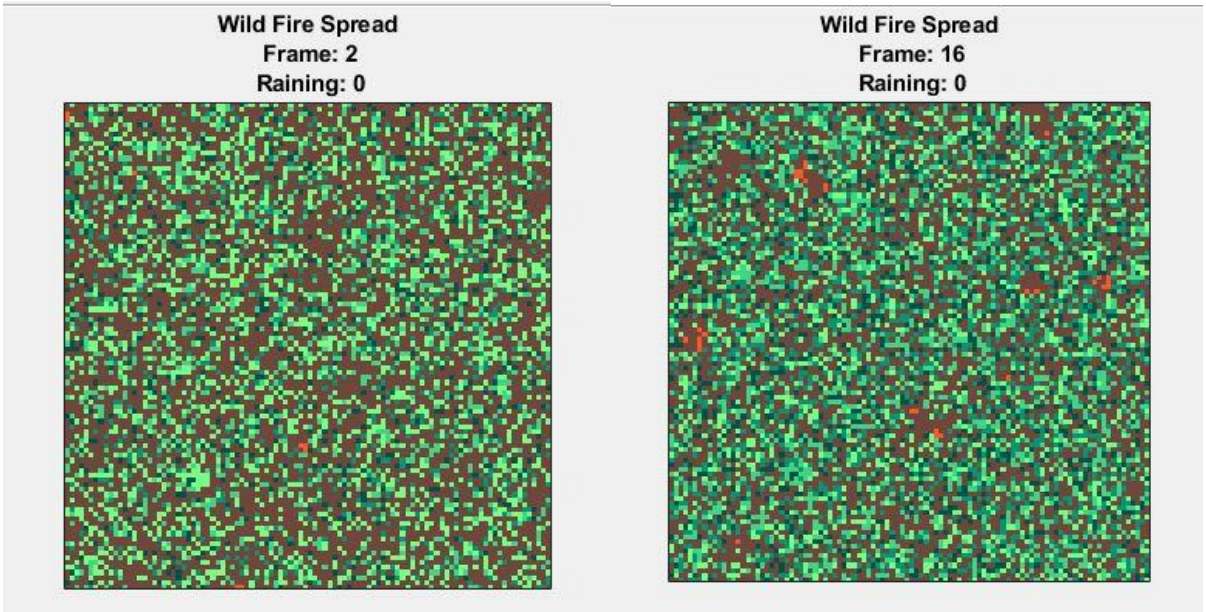


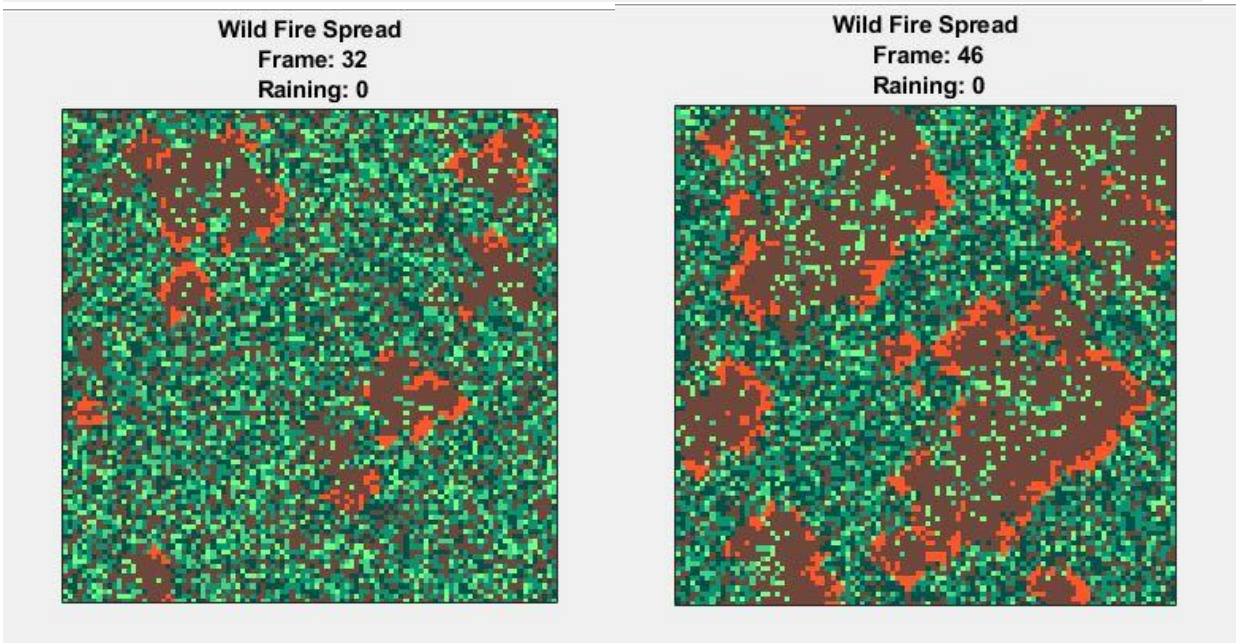
As somewhat expected in this model, periodic rainfall generally made for an increase in survivability. If the seasonal rainfall period were longer, it would have been likely that we saw an increase in survivability of the forest in that simulation. This makes sense as period rainfall continually dampens the spread of the fire while in a seasonal model the fire is left unchecked for longer periods of time. Drawing parallels into the real world, we see that areas with long droughts are more prone to large forest fires than areas with consistent rainfall. This is in line with our simulation (even though the rainfall does not dampen in our simulation as much as the real world) so we can be confident that we are getting reassuring results from our model.

Hypothesis 3: A higher forest density will result in a larger burn that leaves more of the forest barren.

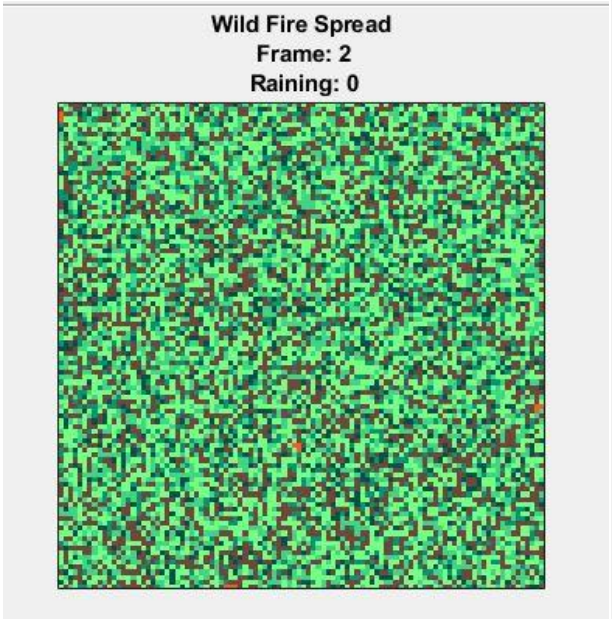
To test this hypothesis, we tested prob tree values at .4, .7 and 1 with the results below:

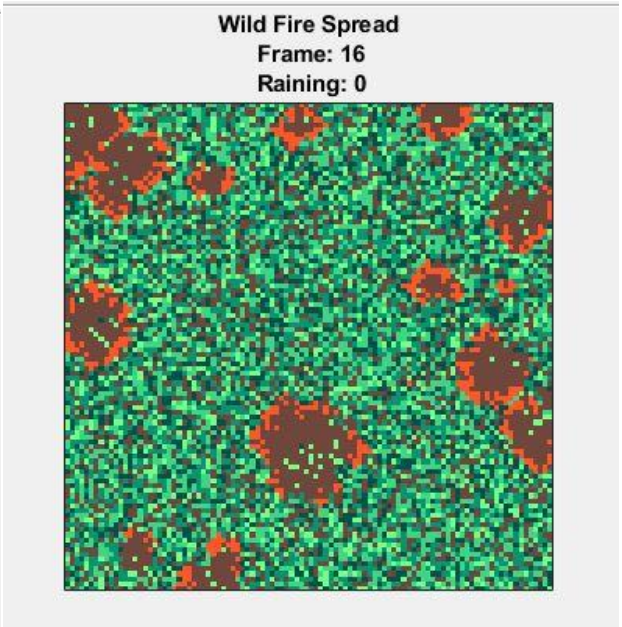
Prob_tree = .4

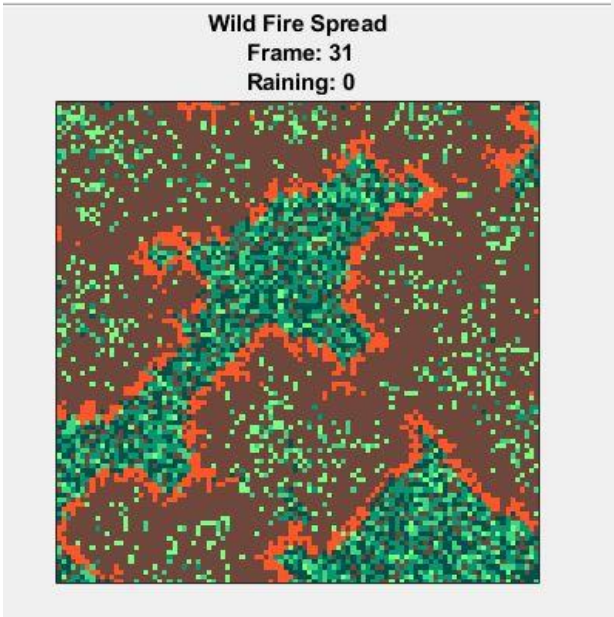


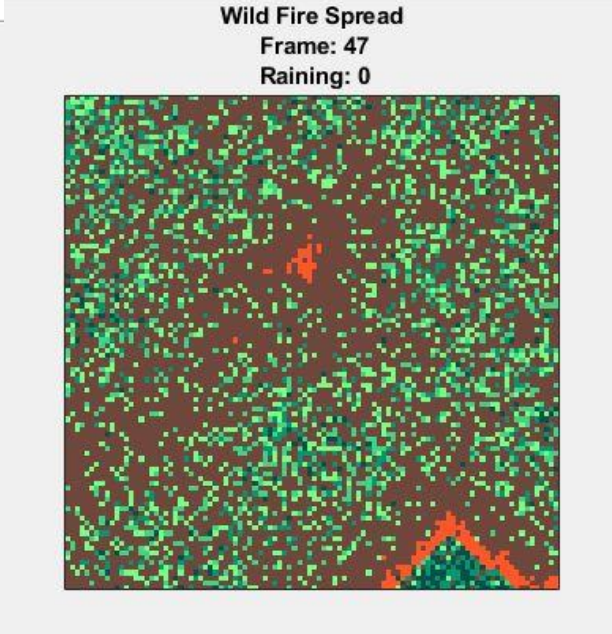


Prob_tree = .7

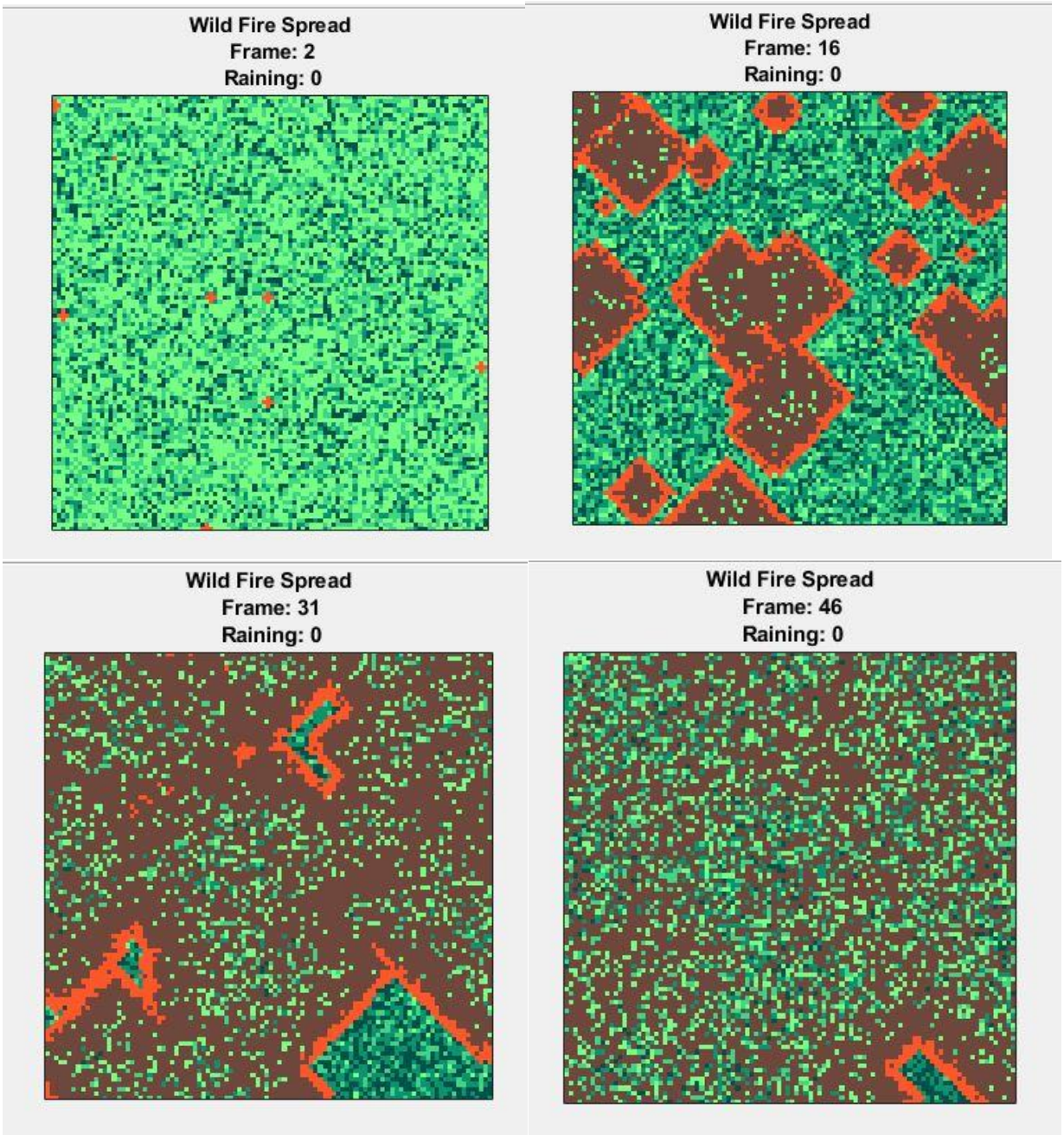

Wild Fire Spread
Frame: 2
Raining: 0


Wild Fire Spread
Frame: 16
Raining: 0


Wild Fire Spread
Frame: 31
Raining: 0


Wild Fire Spread
Frame: 47
Raining: 0

Prob_tree = 1



As expected, a less dense forest was much less susceptible to wide-spread forest fires than denser grids initially. However, as the simulation progressed and saplings grew and immediately caught fire, the fire actually spread farther and wider at the end of the simulation whereas in the denser simulations the forest was actually already recovering. This is interesting as it shows that denser forests have a stronger ability to recover in this simulation. As some trees do survive, it makes sense that more trees present leads to a higher survival rate like we do see in the real world.

Wrapping up, these three hypotheses show some things that can be tested in the model and confirm some things that we know from the real world. Overall, we believe that this is an

effective model to understand some aspects of forest fires and spreads that could be applicable to the real world.