CS 354R Ants Final Report

Group Members: Emma Simon, Ellen Agarwal

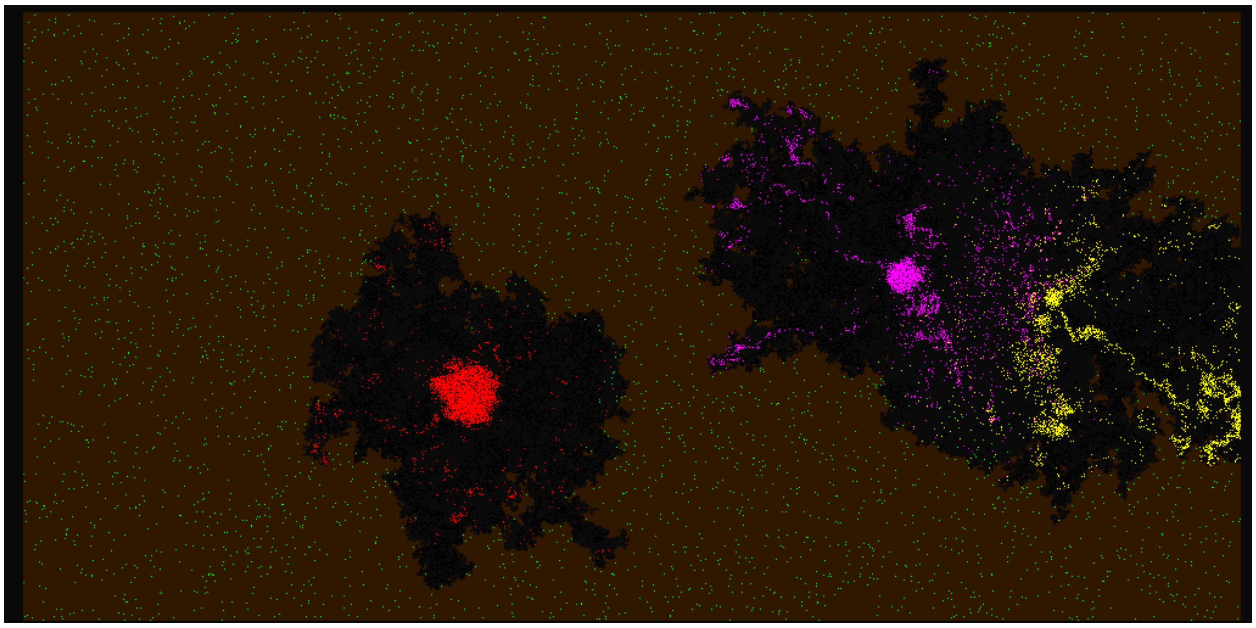
Final Features

We have a fully implemented Ant colony simulator with scent tracking, A* pathfinding to food sources, and a general ant state machine for controlling individual ant's behaviors. Additionally, we have implemented random generation of dirt in the simulation, as well as the ability for ants to dig through (albeit at a slower pace than normal traversal through an open cell). The ant colony behavior is also fully parameterizable through sliders and options we have added to the side of the simulator window so the user can adjust things like scent radius, simulation speed, and food sources.

Our reach goal of physical dirt simulation was not something we were able to implement in time. However, we did end up implementing scent trails, which was extremely complex and difficult to get working without causing infinite loops in our code. This was not an originally planned feature, but now having added it, it gives our ant farm simulation a much more realistic depiction of ant colony behavior.

The entire project was written in Rust and the graphics functionality of the project was implemented using egui.

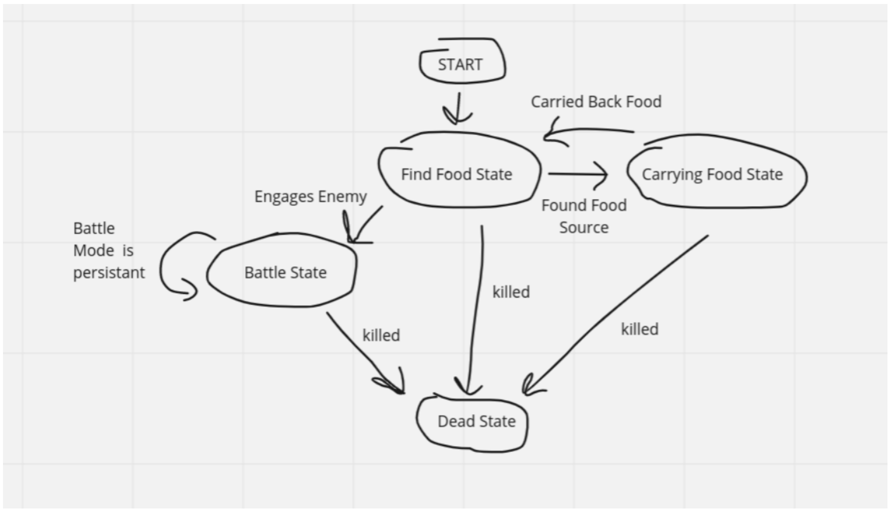
Simulation Results

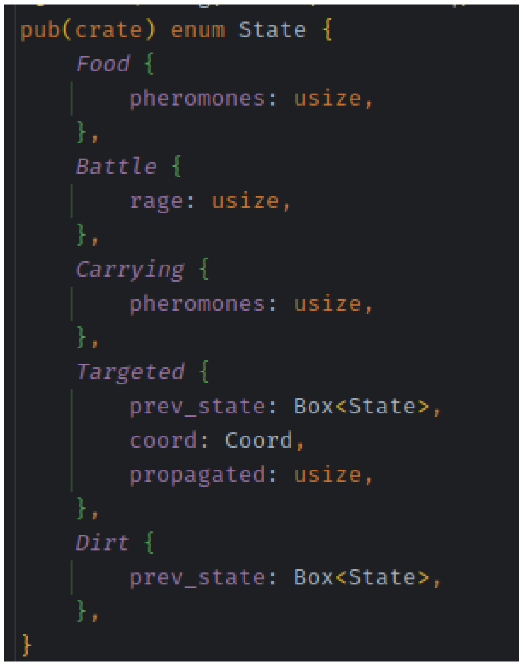


In this screenshot, three ant colonies of different colors are present. Green pixels represent food and brown pixels represent dirt. This simulation is around 2 or 3 minutes in. If you examine closely, you can see the ants forming “trails” into and out of their hives (the large clumps in the center). This behavior was achieved using scent trails as markers to allow the ants to more effectively communicate with each other. The pink and yellow colonies can also be seen fighting each other to the right (although they are currently in a stalemate).

State Machine

All ants start in find food state and progressively move to other states as they encounter stimuli in their environment. When an ant encounters a food source (or another ant carrying food back that wants to pass off food), they enter carrying state. When an ant deposits food it is carrying (or passes it off to another ant), it returns to find food state. If an ant is killed, they enter death state. The ant hive produces new ants over time as food is brought into it. One change that has been made to the state machine since our presentation is that battle mode is no longer permanent and instead has a cool off period of a couple turns before the ant is reverted back to their old state.

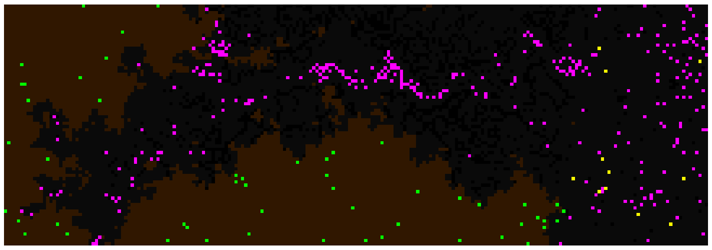




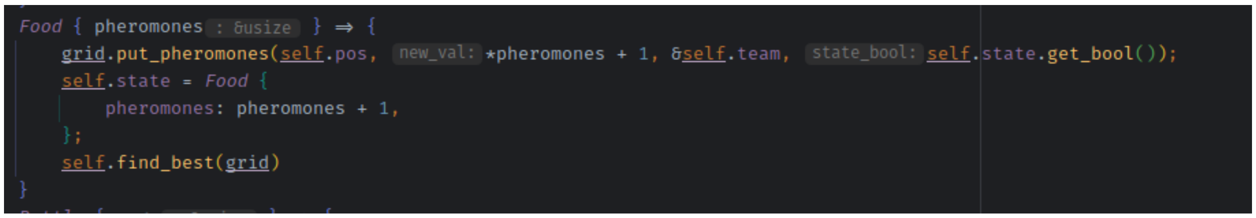
```rust
pub(crate) enum State {
    Food {
        pheromones: usize,
    },
    Battle {
        rage: usize,
    },
    Carrying {
        pheromones: usize,
    },
    Targeted {
        prev_state: Box<State>,
        coord: Coord,
        propagated: usize,
    },
    Dirt {
        prev_state: Box<State>,
    },
}
```

Adding Scent Trails

Per Dr. Abraham’s suggestion during our presentation, our final product now has fully functional scent trails implemented. This gives the ant colonies much more realistic behavior in terms of their pathing. Prior to implementing scent trails the ants would mostly wander around aimlessly without much coordination. After implementing scent trails, the ants tunnel through the dirt in coordination, as shown below.



Scent Trails were implemented using our pheromone signaling system. When an ant leaves a grid cell, it leaves behind its pheromones (or scent), with metadata about its path. This helps guide other ants that enter that cell later to keep all of the ants on target to their destination (whether it be food source, colony center, etc). For instance, when the ants are in food state, they leave behind a signal on each cell stating the number of steps they had taken before reaching that cell (from the center of the hive). This effectively functions as a way of telling distance from the hive for each cell. The ants trying to path to their food source use this data in a very similar way to djikstra’s shortest path algorithm.

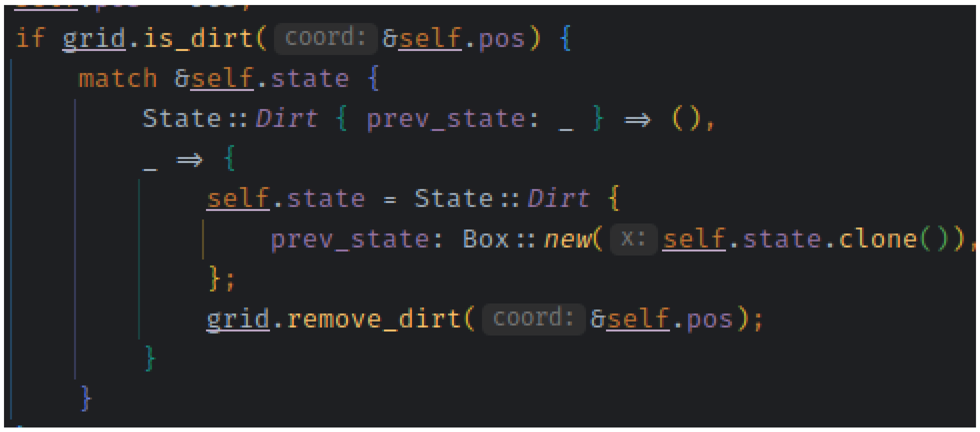


```
Food { pheromones : &usize } => {
    grid.put_pheromones(self.pos, new_val: *pheromones + 1, &self.team, state_bool: self.state.get_bool());
    self.state = Food {
        pheromones: pheromones + 1,
    };
    self.find_best(grid)
}
```

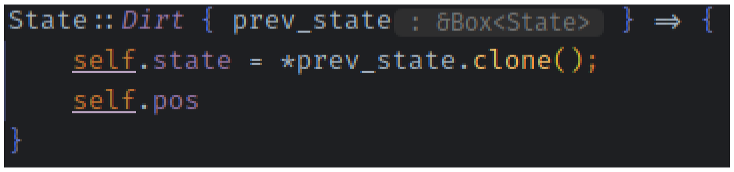
Dirt Generation and Digging as part of the State Machine

Dirt generation ended up being fairly simple. Initially we used a noise algorithm to determine where dirt was placed and where empty space was on the initial grid, but we found it looked better if we made the entire grid dirt and then randomly chose hive locations / food source locations and undid the dirt placement in those locations after the fact.

Digging through dirt is no different for ants than normal movement, except that it takes an extra turn (implemented through a “Dirt/Digging” state in the ant state machine).

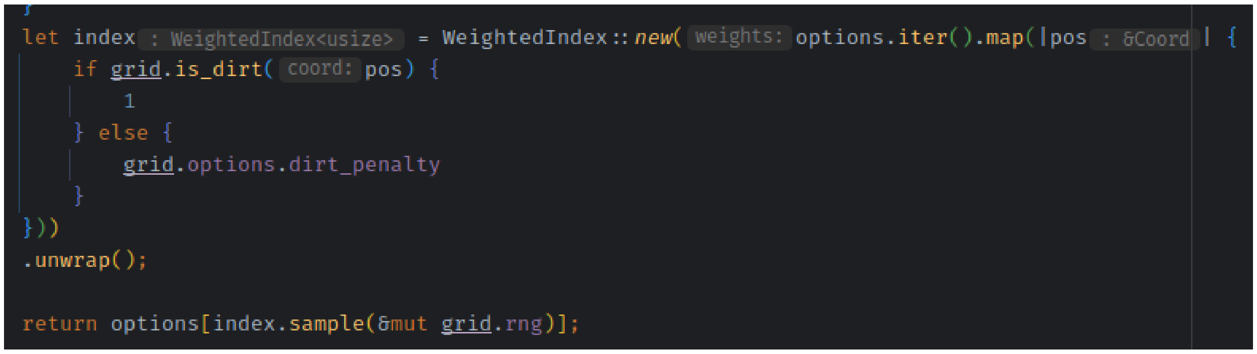


```
if grid.is_dirt( coord: &self.pos) {
    match &self.state {
        State::Dirt { prev_state: _ } => (),
        _ => {
            self.state = State::Dirt {
                prev_state: Box::new( x: self.state.clone()),
            };
            grid.remove_dirt( coord: &self.pos);
        }
    }
```



```
State::Dirt { prev_state : &Box<State> } => {
    self.state = *prev_state.clone();
    self.pos
}
```

Additionally, we heavily weigh against ants digging through dirt in order to force them to adhere to each other's paths more closely (which makes sense from a logical standpoint, it’s better to follow the already carved tunnel rather than make a new one).



```
let index : WeightedIndex<usize> = WeightedIndex::new( weights: options.iter().map(|pos : &Coord| {
    if grid.is_dirt( coord: pos) {
        1
    } else {
        grid.options.dirt_penalty
    }
}))
.unwrap();

return options[index.sample(&mut grid.rng)];
```

References

Ant Pheromones Background:

https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/trail-pheromone

Pathfinding Background (specifically Dijkstra and A* are the most important here):

https://en.wikipedia.org/wiki/Pathfinding

Finite State Machines Background:

https://statecharts.dev/what-is-a-state-machine.html

Rust Documentation:

https://doc.rust-lang.org/book/

Egui Rust Documentation in Specific:

https://docs.rs/egui/latest/egui/