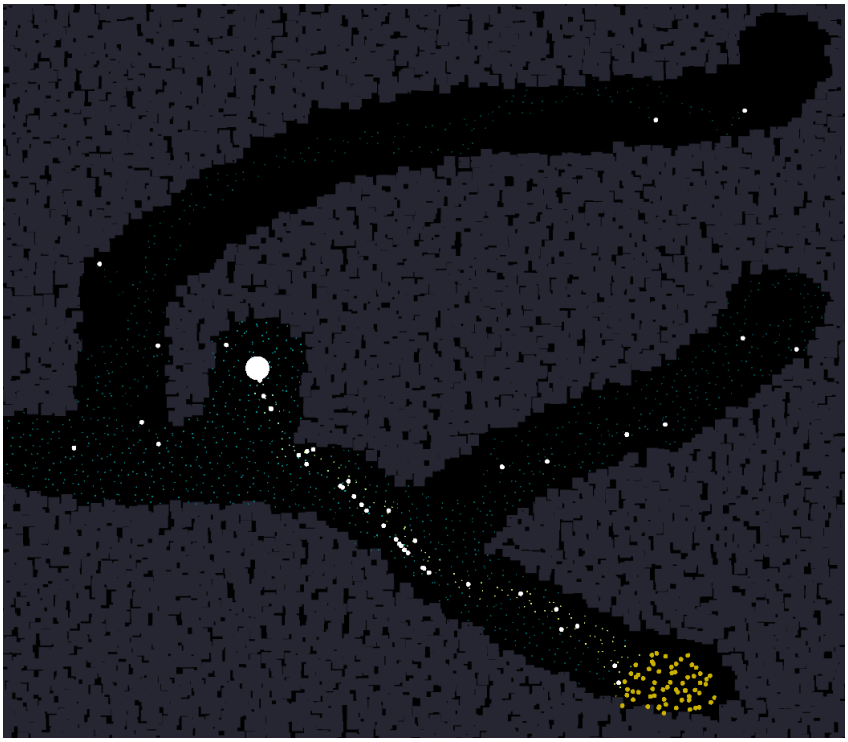
CS202

Cross-Platform Development

Prototype Report

Ant Lab, Manny Kwong 2021



Submitted to

YOOBEE COLLEGES

1 Weekly Progress

1.1 Week 1

Failed Parallelization:

Looked into implementations of simulations with shaders using textures as data buffers and the WebGL examples from lectures.The goal was to use WebGL to compute the behaviour of each ant in parallel with the GPU. I managed to get ants on the screen and the drawing onto a texture to work. However I struggled to get the ants to wander around randomly properly. At the end of the week, behind schedule and with little in the way of progress I decided to fall back to having the ant behaviour dealt with by javascript (CPU) and having the WebGL only deal with the graphics.

1.2 Week 2

Ant movement and drawing:

Getting the ants to wander around randomly and then converge to a point was straightforward with normal javascript. Implementation of the WebGL is heavily borrowed from the first 2 weeks of lecture material, drawing many points and interacting with them. The intention was to quickly get something on the screen and replace the points with textured triangles if I had time.

Simulation space:

The simulation space consists of an array of cells. Each cell could hold an obstacle, food, homing pheromone strength and food pheromone strength. These values were passed onto the shader as a 4 dimensional vertex. From these values the shader determined size, color and shape of the point drawn so these objects could be distinguished from one another. The default square shape of the point was turned to a circle by discarding fragments with coordinates larger than some length from the center. Each ant reads the cell that they currently are positioned on and behave according to its contents.

UI:
A rough UI was made to switch between drawing obstacles and food and to input simulation parameters such as the number of active ants and the rate of evaporation of pheromones.

1.3 Week 3

Ant homing pheromone trail behaviour:
Having the ants lay homing pheromone and detect food in nearby cells was straightforward. When food is found, the ants action goes from finding food to returning home with the food. Having the ants follow the homing trail proved challenging.

As the pheromone strength decreases over time it creates a gradient with the highest strength being at the food and the lowest at home. At first, I tried to use this to determine the direction of the trail but the ants would frequently get lost when the path back was too long as the trail would disappear before home. If trail evaporation was too slow the trail becomes too long and complicated causing the ants to get lost.

Next I tried giving the pheromones a direction by having the cells store the pheromones as a vector and having the ant choose the direction of the highest strength pheromone. With this the ants followed the trail closely, however trails that crossed each other or looped back onto itself would sometimes cause the ants to be stuck in a loop (death spiral). Furthermore the trails did not converge to an optimal path in a timely manner.

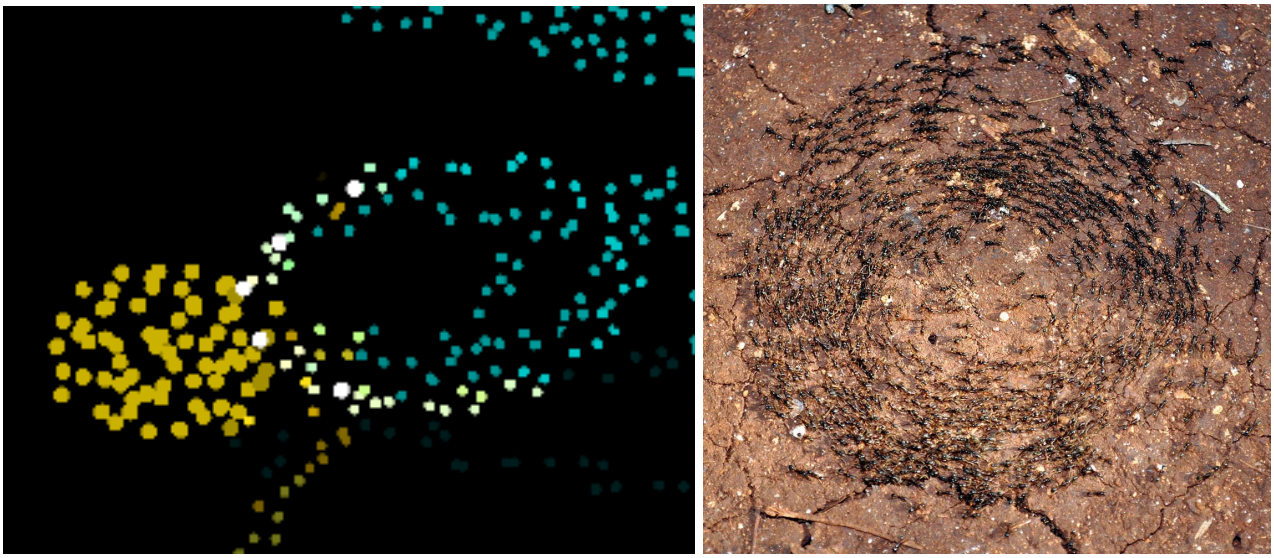


Fig 1 Ant death spiral in the simulation and in real life

Ant obstacle behaviour:
Detecting obstacles was straightforward, differing from detecting food by only reading the cells in front of the ant. This was done testing if the dot product of the ants direction and the cells direction wrt the ant is larger than 0. Initially I had the ants change its direction to the opposite direction of the cell but this would cause the ants to sometimes be stuck bouncing between 2 nearby obstacles. They also had trouble traversing through tunnels with sharp turns. To solve this I calculated the direction perpendicular to the cell direction that required the least turning for the ant. This improved the ants ability to find food but would sometimes cause them to get stuck in adjacent obstacle cells. This was somewhat improved by having the ant bounce back after colliding with an obstacle but this prevented them from getting through narrow tunnels. In the end I fudged it by having the ants respawn at home when they get stuck for too long.

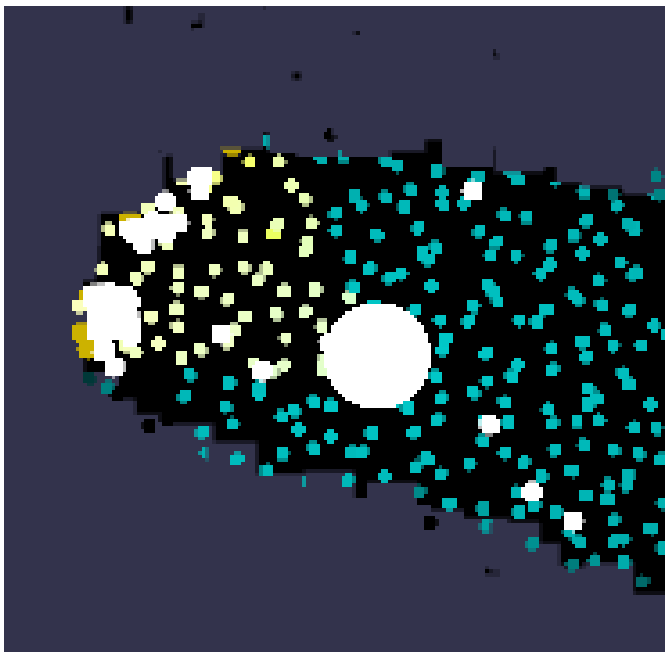


Fig 2 Ants getting stuck on obstacles

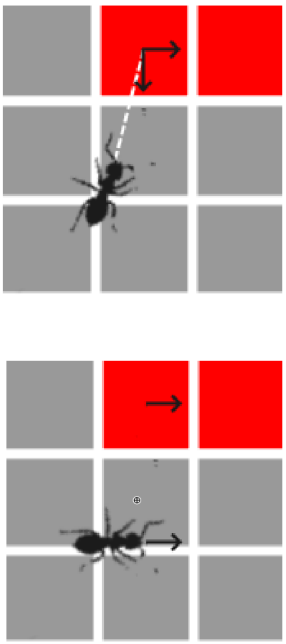


Fig 3 Ant choosing its next direction

1.4 Week 4

Interactions:
Picking up, throwing and squishing ants was straightforward and was a welcome break from the path optimization problem.

Path optimization:
The thesis An Ant Colony Simulator by Delan Ren 2012 (https://mro.massey.ac.nz/bitstream/handle/10179/3758/02_whole.pdf) he mentions solving some of the trail following problems by setting a limit to the strength of the pheromones per cell.

I also came across a video of a similar project C++ Ants Simulation by Pezzza's Work 6/4/2021 (https://www.youtube.com/watch?v=emRXBr5JvoY) which was vastly superior to mine in finding optimal paths. Its solution to the trail following problem was to have the ant decrease the strength of the pheromone strength it laid on the cell over time to create a gradient that gave the trail its direction. It also had a liberty coefficient that controlled how strictly the ant followed its path which allowed the ant to stumble into better paths.

Adopting these solutions reduced the frequency of ants getting stuck at trail intersections and improved optimal path convergence.

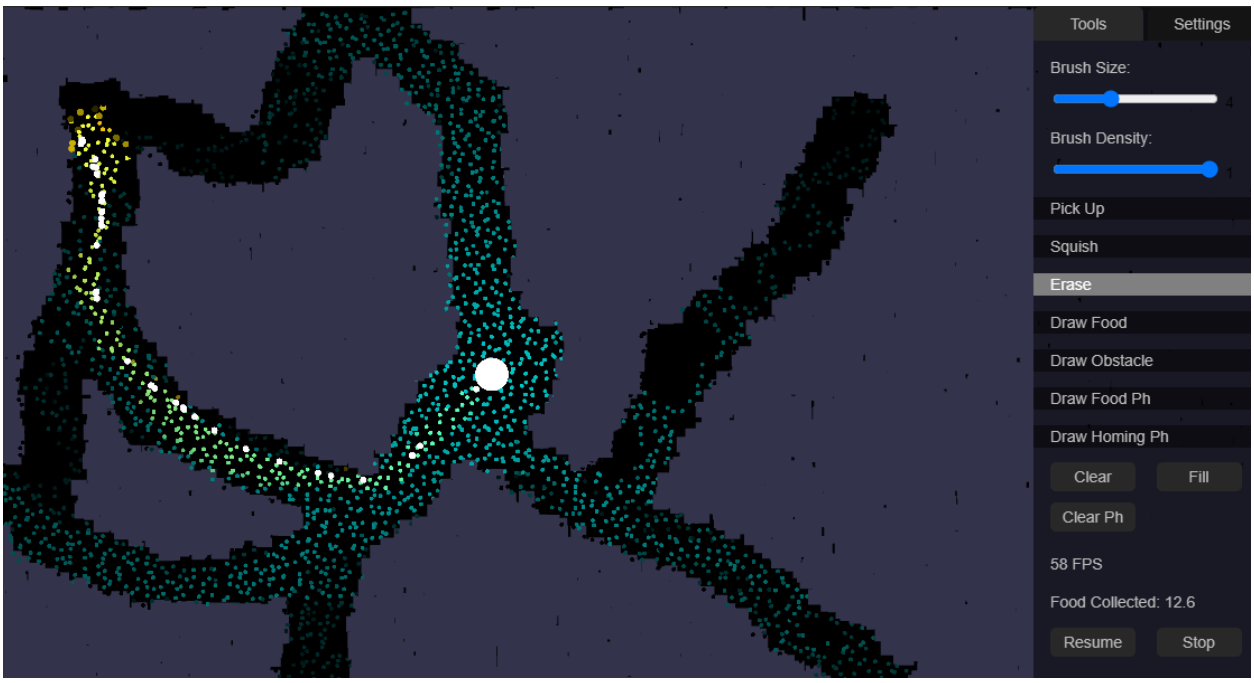


Fig 4 Ants finding the shortest path to the food source (yellow). The homing pheromone (green) strength deposited decreases further along the ants journey giving the trail a direction towards home.

Deployment:
With everything working offline in the browser I moved onto deploying the app online with firebase. Initially it did not load the javascript files apart from main.js. Having changed the first letter of each javascript to lowercase, the javascript files were present but its contents were replaced with html. I then moved all the classes into main.js and everything functioned correctly. Not satisfied with this I tried moving all the javascript files (including the main.js) out of the /src folder and into the same folder as the index and changed the script reference location in the index accordingly. This caused the javascript files to load correctly but with errors at line 1 for some of the files. After deleting the empty line(s) at the top of the scripts the app worked as it did offline.

Mobile responsiveness:
The app did not take to portrait mode well, with the menu taking up too much space and the canvas element not resizing to match the mobile's screen as it did on desktop. Running out of time, I opted to just rotate the UI 90 degrees and switch the x and y vertex coordinates when the orientation of the media device is portrait.

To allow dragging on the touch screen to draw obstacles, food, throw ants etc. a touch listener was added with e.preventDefault() to stop the app from scrolling instead. Ants now also pop up above their positions while being picked up as to not be covered by the users finger.

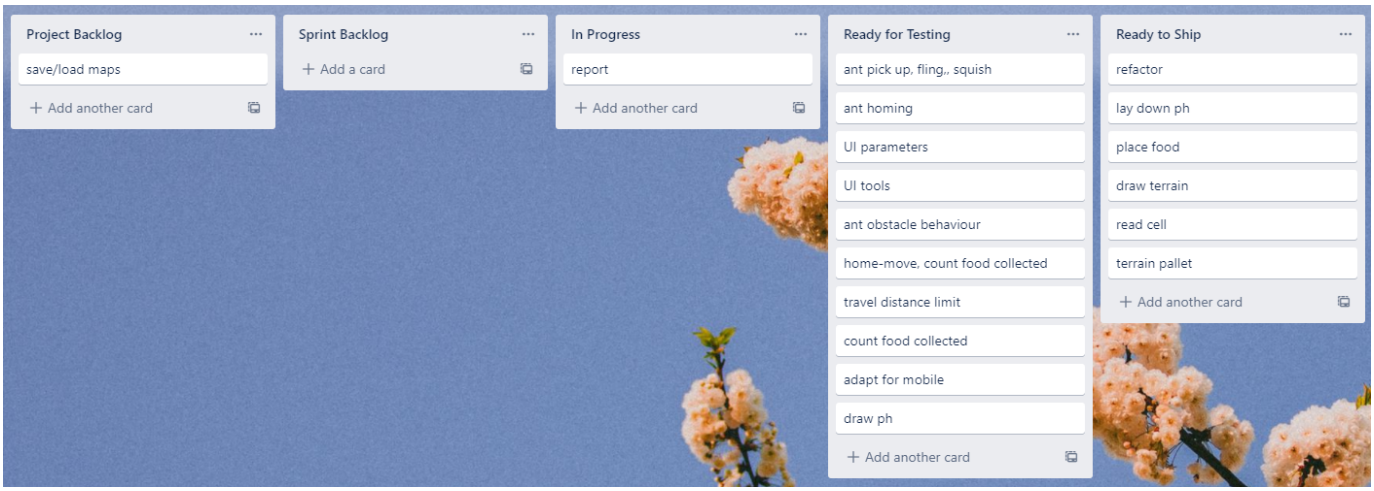
Unsolved problems:
A problem remains in that the app starts out zoomed in on mobile. In the proposal I had included the feature of saving and loading the map (simulation space). I had planned to let the user save their maps by converting the cells to a .json file but ran out Otherwise it works more or less as intended. As the project went on it was more and more apparent that the simulation speed was too slow and features too limited for it to be used practically.

2 Deliverables

2.1 Release

A link to the app is provided at the bottom of my portfolio site at https://mansinh-d25ff.web.app/

2.2 Kanban Board



2.3 Git Repository

The source files can be found at https://github.com/mansinh/BCS-202 under the Ant Lab folder.

3 Conclusion

3.1 Summary

The project ran behind schedule due to unfamiliarity with computation using WebGL leading me to abandon this approach. The simulation does not run fast enough and the map generation tools are a bit too clumsy for the app to be used in any practical application. Most of the proposed features were implemented with the big exception being saving and loading maps. The app is more or less bug free on desktop browsers but has trouble adapting to mobile screens but is more or less usable on mobile. It was, however, enjoyable to draw maps for the ants to solve and to fiddle with the simulation parameters for finding the optimal path during testing. In this sense this app is more a web toy than anything else.

3.2 Future Work

If I was to make this application more practical I would:

- Discard free drawing on cell based maps and go for drawing lines on a grid instead to produce graphs that can be saved and loaded.
- Implement ant parallelization with WebGL.
- Add a profiler and tables for graphing performance and results.
- Add multiple goals/food source groups such that an ant must collect some of each group before returning home so problems like the travelling salesman can be set up.

Alternatively I could develop this toy into a god game like Populous. To do this would probably mean replacing points primitives with animated sprites and/or models which would be very challenging for me at this point. With this in view I would probably be better off using a framework more suited to web game development such as Babylon.js or Three.js.

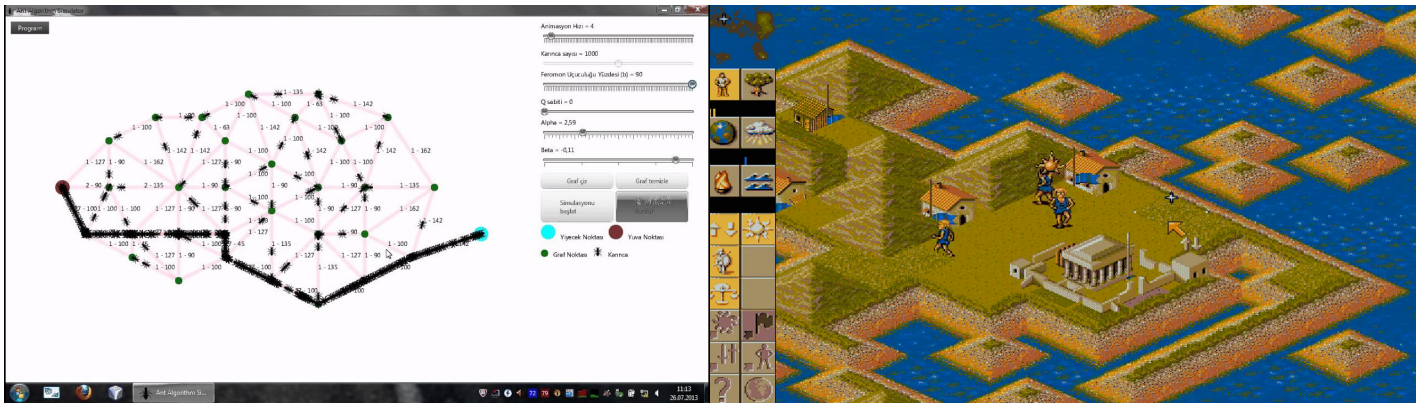


Fig 6 Node/graph base ant algorithm simulator by Numan Karaaslan (left) and god game Populous (right)