Ants

High performance agent-based modeling and visualization with Rust, WebAssembly, and WebGL

Casey Primozic

Overview

1

- https://github.com/ameobea/minutiae
- Originally started as a game engine at the beginning of last summer, but spiraled off into a lot of other things along the way.
- Model of computation similar to Turing Machines, Cellular Automata, Stack Machines, etc.

Agent-Based Modeling

- Immutable State
- Strictly Controlled

Minutiae System Architecture

- Representation of state and logic
- Allows for the creation of “simulations,” “demos,” and interactive applications
- Deterministic + Nondeterministic variants
- Serial / Parallel
- Runs in the browser via WebAssembly/ Asm.JS

Universe

- 2-dimensional array of cells
- Always square (equal height + width)
- Populated with cells and entities at application initialization using a `Generator`

Cells

- Cells
  - Every coordinate of the universe contains a cell
  - Contain generic state
  - Do nothing by themselves
  - Roughly equivalent to a Turing Machine’s tape

Entities

4

- Exist in a single coordinate within the universe
- Have two kinds of state
  - State
  - Mutable State

Entity States

- Non-mutable State
  - Fully generic (only limited by some Rust traits to enable safe program execution)
  - Visible to all other entities in the universe (unless limited by the entity driver explicitly)
  - Well represented by Rust’s fat enums
  - Only changeable by the Engine or Middleware

Mutable Entity State

- Not visible to other entities in the simulation
- Mutable by the entity driver during its evaluation
- Useful for storing stuff like PRNG state
- Generic, but with strict limitations on its contents using Rust state in order to preserve performance

Computation Model

- State (data)
- Logic (rules)

Minutiae Computation Model

- State located in 3 places:
  - Universe/cells
  - Entity state
  - Mutable Entity State
- Logic located in 3 places:
  - Entity Driver
  - Engine
  - Middleware

Entity Driver

5

- Function that takes the universe (all cells and entities) as input and returns an array of actions as output
- Evaluated for every entity in the universe each tick

Actions

- Enum of possibilities
- Creatable by entities during execution of the simulation
- Three varieties
  - Cell Actions
  - Self Actions
  - Entity Actions

Engine

- Defines a set of rules for processing *actions*
- Defined by the user

Engine cont.

- Defines rules for handling the actions created by entities
- Implemented as a (generic) trait

```
pub fn exec_actions(
    universe: &mut Universe<CS, ES, MES, CA, EA>, cell_actions: &[OwnedAction<CS, ES, CA, EA>],
    self_actions: &[OwnedAction<CS, ES, CA, EA>], entity_actions: &[OwnedAction<CS, ES, CA, EA>]
) {
    for cell_action in cell_actions { exec_cell_action(cell_action, &mut universe.cells, &mut universe.entities); }
    for self_action in self_actions { exec_self_action(self_action, &mut universe.entities); }
    for entity_action in entity_actions { exec_entity_action(entity_action, &mut universe.entities); }
}
```

Serial Engine

7

- Executes the entity driver for each entity in the universe one by one
- Stores up all generated actions in an array
- Handles all of the actions one by one, transitioning state sequentially
  - Handles conflicts e.g. (if an action targets an entity that moved or was deleted)

Parallel Engine

- Entity driver for each entity is run in parallel
- Work stealing parallel iterator is used along with synchronization primitives to create a vector of actions to evaluate

Parallel Engine cont.

- Made possible by separating the creation of actions from their evaluation
- The entire universe (all cells and entities) are immutable during this phase
  - If they weren't, there would be the possibility of all kinds of parallel issues to crop up
- Required copious usage of `unsafe` Rust code

Parallel Engine Cont.

- Any possible determinism is lost
  - No guarantee to the ordering of the evaluation of actions
- (Theoretically) linear efficiency gains for the entity driver phase
- Engine’s evaluation of actions is not parallelizable
  - Requires mutable access to the universe and entities

Middleware

- Take all of the bounds off the system
- Allow direct, mutable access to the entire universe before and after each tick
- Allow mutation of cell states, access to universe state for output, etc.

Middleware Examples

- `MinDelay`: If the time between the current frame and the previous frame is less than an interval, sleeps until that interval is reached
- `TracerMiddleware`: Alters the state of the cells under which entities exist to leave a trace of the entity's color that persists after the entity moves. This fades over time.

```
10
11
```

Driver

- Actually executes the simulation
- Calls entity drivers, processes actions, executes middleware
- Can be used to execute in asynchronous environments (Emscripten via browser event loop)

Overview of Simulation Process

- Cells hold data in a 2D array
- Entities live on top of cells and make decisions (generate actions)
- Engine processes actions and mutates cells+entities
- Middleware breaks the rules and allows for side effects
- Driver makes everything happen

Output + Visualization

- A 2D universe is conveniently displayed in a lot of ways
- Can be treated as an array of pixels in an image or video

About Rust

- Low level language, like C/C++
- Enforces memory safety and (almost) always prevents things like use-after-free, dangling pointers, etc.
- Powerful type system with generics, traits, smart pointers, full control of memory.

So what can you do with it?

WebAssembly

- Multi-architecture support for free
- Near native performance (or better) for countless work domains
- Pretty new, only an MVP right now (but very widely supported nonetheless)
- Really, really cool

Rendering to a Canvas or GIF

- Middleware is created to calculate the color of each coordinate in the universe
  - Takes a `calc_color` function which takes a cell state and list of all entities at that cell and returns a color

Try it out yourself, right now!

https://ants.ameo.design

About the Model

- Independently functioning entities (ants) communicate and cooperate to accomplish a goal (collecting food)
- Very limited information available
  - 9 cells adjacent + beneath them
  - Offset from the hive (conditionally - will explain in a bit)

Implementation (pheromones)

Two kinds of pheromones (Cell State):

- “Wandering” trail (looking for food)
- Report trail (bringing back / reporting a found food source)

Implementation (ant logic)

Three different states ants can be in:

- “Wandering” - looking for previously undiscovered food sources
- “Reporting” - Leaving a trail to a known food source
- “Following” - Following a previously known trail to a food source

Cell State

- Blank
  - Pheromone Levels
- Food
- Barrier
- Anthill

Entity State

Ants only know the following information on which they base all of their decisions:

- State of the 9 cells adjacent to and beneath them
- Conditionally, their offset (in cells) from the anthill
- As limited as possible

World Generation

- Very simple worldgen
- All entities spawn on the same cell on frame 1
- Barriers and food patches generated according to configuration values

Real Ants

- Pretty smart tbh
- Counting steps
- Visual memory
- Teach each other
  - Very complicated chemical messaging systems

Goal

- Exhibit the “ant-colony optimization”
  - Swarm Intelligence
  - **Optimization**, solving NP-hard problems
  - Gradient Descent + Machine Learning
- Finding the optimal parameters for this model would be a great use of that

Results

- Mostly successful!
- Swarm Intelligence is clearly exhibited
- Ants make use of information gathered by other ants indirectly by reading pheromones, contributing to it in the process

Results cont.

- Several inefficiencies and issues
  - Ants get “stuck” a lot, especially on their trip back to the anthill
  - Naïve pathing algorithm
  - Somewhat clunky and abstruse UI

Future Work

- My very own personal Internet Ant colony
- Economic aspects
  - Birth of ants, regrowth of food
- Better / more elaborate worldgen
- More advanced chemical messaging systems

Open Source is Nice

Check out the source code!

https://ameo.link/ants

Thanks for Listening!