AgentLang

FACULTY OF MATHEMATICS AND PHYSICS
Charles University

Programming Language for Agent-based Modeling

Author: Tomáš Boďa | **Supervisor:** Mgr. Tomáš Petříček, Ph.D. | **2024**

Introduction

AgentLang is a programming language designed for modeling agent-based simulations. It provides a straightforward structure, where the user defines an agent model and a set of properties representing the behaviour of the agent.

Agent-based Modeling

Agent-based modeling is a simulation technique, where a system is modeled using a set of fundamental units of a system called agents. Each agent individually assesses the current situation and makes decisions based on a set of defined rules.

Primary Goals

Due to the increasing popularity of the agent-based modeling technique in many scientific fields, there is a need for a unified framework usable by people of all scientific fields, regardless of their technical proficiency. For these reasons, the AgentLang framework aims to provide:

- unified language with simple and straightforward syntax and structure
- only the necessary, essential core library of features
- an alternative way of modeling using a spreadsheet representation

Global Variables
We define a set of **global** constants that can be shared among all agents.

Agent Model
We define an **agent model** of type **boid** and spawn 100 of these agents.

Agent Constants
We define a set of **constant** properties for this agent - properties that have a constant value during the course of the simulation.

Agent Properties
We define a set of **properties** for this agent, properties that are recalculated each step of the simulation for each agent.

Retrieving Agents
We **retrieve** the array of agents of type **boid** and their values using the **agent(boid)** function.

Filtering Agents
We **filter** the array of agents of type **boid** based on some expression and retrieve the filtered array of agents.

```
# Boids Algorithm for Bird Flocking

define visual_range = 100;
define avoid_range = 20;
define centering_factor = 0.0005;
define avoid_factor = 0.005;
define matching_factor = 0.05;

define s_max = 8;
define s_min = 6;

agent boid 100 {

    const w = 10;
    const h = 10;

    const x_init = random(100, width() - 100);
    const y_init = random(100, height() - 100);
    const x_vel_init = choice(-2, 2);
    const y_vel_init = choice(-2, 2);

    property x: x_init = (x + x_vel_limit) % width();
    property y: y_init = (y + y_vel_limit) % height();

    property x_vel: x_vel_init = x_vel + x_separation + x_alignment + x_cohesion;
    property y_vel: y_vel_init = y_vel + y_separation + y_alignment + y_cohesion;

    property s_sqrt = sqrt(x_vel * x_vel + y_vel * y_vel);
    property s = if s_sqrt == 0 then 1 else s_sqrt;

    property vrc = count(boids_vr);

    property x_separation = sum(boids_ar | b -> x - b.x) * avoid_factor;
    property y_separation = sum(boids_ar | b -> y - b.y) * avoid_factor;

    property x_alignment = (sum(boids_vr | b -> b.x_vel) / vrc - x_vel) * matching_factor;
    property y_alignment = (sum(boids_vr | b -> b.y_vel) / vrc - y_vel) * matching_factor;

    property x_cohesion = (sum(boids_vr | b -> b.x) / vrc - x) * centering_factor;
    property y_cohesion = (sum(boids_vr | b -> b.y) / vrc - y) * centering_factor;

    property boids = agents(boid);

    property boids_ar = filter(boids | b -> dist(b.x, b.y, x, y) < avoid_range);
    property boids_vr = filter(boids | b -> dist(b.x, b.y, x, y) < visual_range);

    property x_vel_limit = ……;
    property y_vel_limit = ……;
}
```

Technical Details

AgentLang is an interpreted programming language. The source code is parsed using a pushdown automaton, which produces a semantic representation of the program in form of an abstract syntax tree (AST). The interpreter is a tree-walk interpreter that traverses the AST and interprets the code node by node.

Interesting Concepts

The primary structure of an AgentLang model is an agent. Its behaviour is modelled using a **set of property declarations**, each assigned an inline expression upon which it is recalculated. This structure is simple to understand as well as can easily be mapped to the columns of the spreadsheet interface. Moreover, interactions between agents are realized using the built-in **set comprehension** structure, which takes an array of agents as the parameter, followed the name of the agent instance that is currently operated on and finally an expression based on which the set comprehension either finds, filters or sums the agent instances.

Conclusion

The project features a fully usable programming language and a web-based interface with a code editor, a spreadsheet editor and visualisation module. The project fulfilled all of the stated goals and criteria.

Acknowledgements

I would like to thank my supervisor Mgr. Tomáš Petříček, Ph.D., whose guidance, patience and expertise have been invaluable, shaped this thesis and contributed significantly to its completion.

AgentLang

Tomáš Boďa

github.com/TomasBoda/agent-lang