Final Project

MASL
Multi-Agent Simulation Language

Jiatian Li   jl3930
Wei Wang    ww2315
Chong Zhang cz2276
Dale Zhao   dz2242
MASL OVERVIEW

WHAT & WHY
Motivation

The Agent-Based Model (ABM)

- A system where the interactions between autonomous agents (individuals) are simulated
- Global patterns and effects of such interactions as a whole can be observed and assessed
- Example: Game of Life (as a cellular automaton), Boids, Heatbugs
- Applications: Physical world reality simulation, cryptology, etc.
Motivation

Examples of cellular automata

- Conway’s Game of Life
- Heatbugs
Motivation

MASL – Multi-Agent Simulation Language

- Facilitate building ABMs without having to start from scratch or engaging complex domain toolkits
- Particularly, we focus on developing cellular automata.
Features of MASL

- Imperative programming language
- Static and strong typing system
- Functions as first class objects
- Compound types supported: objects and lists
- Objects as state machines
- Simple simulation environment
Features of MASL

Why state machines?

- Each individual in the system will act according to its observation of local environment as well as its inner state. State machines are a perfect model for this.

What is a simulation?

- In a simulation, individuals will update themselves (take actions) and visually illustrated. All these individuals will be represented using objects and stored in lists for the simulation environment to step through.
A SHORT TUTORIAL ON MASL
Basic Data Types & Lists

Basic Data Types

- Integer (32-bit)  
  ```
  int i = 19;
  ```

- Double (64-bit)  
  ```
  double pi = 31.4e-1;
  ```

- Char  
  ```
  char c = 'a';
  ```

- Boolean  
  ```
  bool flag = true;
  ```

Lists

- Defining a list  
  ```
  [int] fib = [int] {1, 1, 2, 3, 5, 8};
  ```

- A string is essentially a list of char elements:  
  ```
  [char] str = "hello world";
  ```
Functions as First Class Objects

Functions in MASL can be stored in variables, and used like a variable.

```plaintext
int max(int a, int b) {
    if (a > b) {
        return a;
    }
    return b;
}

fun ((int, int):int) f = max;
```
Objects as State Machines

An class consists of

- Any number of statements that defines members of its instances and does initialization upon instantiation (equivalent to a constructor), and

- Any number of states.

class Guard {
    state Defend {
        if(enemySighted()) this->Attack;
    }

    state Attack {
        if(!enemyEliminated()) shot();
        else this->Defend;
    }

    bool enemySighted() { /*...*/ }
    bool enemyEliminated() { /*...*/ }
}

An object is an instance of a class.

Class Guard g = class Guard();
if(g@Attack) { /*...*/ }
More on Lists

Lists are able to accommodate elements of any data types.

```java
class Programmer
```

```java
team = /*...*/;
```

```java
[[double]] matrix = {
    [double] { 1, 0, 0}
    [double] { 0, 1, 0}
    [double] { 0, 0, 1}
};
```

A for-loop using list iterator: Equivalent to:

```java
for (int n : list) {
    sum = sum + n;
}
```

```java
for (int i = 0; i < list.size(); i = i + 1) {
    sum = sum + list:i;
}
```

Functions can be applied to elements of a list.

```java
int n = list:.count(fun (int n):bool { return n > 3; });
```
A MASL program is essentially a simulation. Currently we only support the simulation of cellular machines.

class Cell {
   /* ... */
}

[class Cell] container;

/* Fill in the container. */

// Set the attributes of the simulation environment.
cellSize = 10;
nx = 100;
ny = 100;
interval = 100;

run(container);
```c
// Greatest Common Divider

int gcd(int a, int b) {
    if (b == 0) {
        return a;
    } else {
        return gcd(b, a % b);
    }
}

// Filtering a list

bool isEvenNum(int num) {
    return (num%2 == 0);
}

[int] list = [int]{1, 2, 3, 4, 5, 6};
[int] evenList = list:.filter(isEvenNum);
for(int i : evenList) {
    printInt(i);
}
```
DEVELOPING
MASL
Compiler Implementation

- Scanner recognizes the tokens
- Parser checks the syntax correctness of the token strings building up the program
- AST is generated after parsing
- Check the semantic correctness of the program
- Translate MASL into Java source, and then compile it into Java bytecode
Java Classes for Runtime Support

- **MaslList** Base class of all MASL list types.
- **MaslFunction** Base class of all MASL function types.
- **MaslClass** Base class of all MASL class types.
- **MaslSimulation** Base class of MASL simulation environment.
Unit Tests for Individual Features
COLLABORATION

- A repository on GitHub was established for the collaboration of this project.
- Establish code framework and module-wide interfaces first, then divide the work and develop in parallel.
- Exchange ideas in group meetings or communicate with instant messaging tools while coding.
- Each member is responsible for an individual part and has good knowledge about others’ work.
PROJECT PLAN

- Start simple. Start early.
- Experiment with code while designing the language.
- Interfaces between modules should be well defined from the beginning.
- Perform unit tests frequently and thoroughly.
- Expect failure to implement some features...