COMS W4115 Programming Languages and Translators

Sim2D Project Proposal

David Suess
dcs2136@columbia.edu

June 4, 2008

Introduction

Sim2D is a language designed for simulation of the movement of objects on a 2 dimensional map. A programmer may define objects and their behavior. The interpreter will present the simulation on a 2 dimensional map for observation of the interaction of the objects. Possible applications of this language would be to simulate the traffic of air, land and sea vehicles, or the interactions of animals in an environment. The language will be simplified and interpreted for quick development and debugging.

Design

Simulations generally consist of a loop in which operations are performed. Therefore, to simplify the language, the loop for this language will be implied. A user will construct stationary *landmarks* and mobile *rover* objects as desired. For stationary objects, a user must define a position and any number of user customizable fields. For mobile objects, a user must define an origin, destination, and speed. The simulation will automatically move the mobile objects toward their destination. A user can then construct rules to govern changes to the movement of the objects. Any number of user customizable fields can be defined for a *rover* or *landmark* for use in custom behavior of the objects. A 2 dimensional map will be rendered utilizing java swing. The objects will be represented with different color dots. Several buttons will be provided to control the simulation speed and to load and reload programs.

Language Description

The language will look familiar to a C programmer, but it is simplified for what is needed for a 2D simulation. There are no provisions for user defined loops, or pointers. Function definition is limited to the capability of the *rule* keyword. The only basic data types are integer and boolean. The *rover* and *landmark* objects are predefined types that have built in variables that are required for the simulation such as x and y coordinates. A program will generally consist of one or more *create rover* and *create landmark* statements followed by one or more *create rule* statements to define custom behavior. *create* statements are

interpreted immediately and only once unless they exist within a rule. *rule* statements are executed at every iteration of the simulation loop.

Keywords

create	Create a landmark, rover or rule
destroy	Destroy a landmark or rover
reset	Reset a rover to it's initial state
rover	A movable object
origin	A rover field defining the start location of a movable object
destination	A rover field defining the end location of a movable object
speed	A rover field defining the speed of a movable object
landmark	An immovable object
x	The x coordinate of the position of an object
y	The y coordinate of the position of an object
visible	A Boolean field indicating that the object is displayed
custom	Prefix for creating user defined fields
rule	A procedure that is run every iteration of the loop
distance_to	A function that returns the distance to another object
bearing_to	A function that returns the relative bearing to another object
if	Conditional for comparison of objects or their fields
else	Conditional for comparison of objects or their fields
and	Logical operator for conditional expressions
or	Logical operator for conditional expressions
true	Value for Boolean types
false	Value for Boolean types

Operators

.	Dereference fields of a landmark or rover
= !=	Rover or landmark object comparison
= < > <= >= !=	Integer comparison
//	Indicates that the rest of the line is comments

Testing

The interpreter will have a debug option that will inhibit the graphical display and instead output the x and y position of all objects at each time step. Comparison

of these coordinates from different runs of the program will certify deterministic results. Various basic benchmark programs will be written to test all language features. A script will be written to run all benchmarks and diff the results against expected results.

Example Code For Air Traffic Control Simulation

```
// This program will simulate 4 aircraft on approach to an airport.
// The airport will only allow one aircraft within it's airspace.
// Once an aircraft has reached the airport, it is deleted and another
// aircraft may land.
create landmark KPHL {
        x 250;
        y 250;
        visible true;
        custom landing_aircraft none;

}
create landmark westentry {
        x 0;
        y 250;
        visible false;
}
create landmark eastentry {
        x 500;
        y 250;
        visible false;
}
create landmark northentry {
        x 250;
        y 500;
        visible false;
}
create landmark southentry {
        x 250;
        y 0;
        visible false;
}

create rover united421 {
        origin westentry;
        destination KPHL;
        speed 1;
}
create rover american331 {
```

```
        origin eastentry;
        destination KPHL;
        speed 1;
}
create rover southwest65 {
        origin northentry;
        destination KPHL;
        speed 2;
}
create rover american12 {
        destination southentry;
        destination KPHL;
        speed 1;
}


// air traffic control rules for the KPHL airport
create rule KPHL {
        // rover here refers to any rover
        if (rover.destination = KPHL) and (rover.distance_to(KPHL) = 100) {
                // check if aircraft just landed
                if (KPHL.landing_aircraft != none) {
                        if (KPHL.landing_aircraft.distance_to(KPHL) = 0)
                                KPHL.landing_aircraft = none;
                        }
                }
                if (KPHL.landing_aircraft != rover)
                        and (KPHL.landing_aircraft != none) {
                        rover.speed 0; // simulate a circling aircraft
                } else {
                        KPHL.landing_aircraft rover;
                }
        }
}

// rule for all rovers
// remove aircraft from simulation when they reach their destination
create rule rover {
        if (rover.distance_to(rover.destination) = 0) {
                destroy rover;
        }
}
```

Example Code For Dog and Cat Chase Simulation

```
// This program will simulate a dog chasing a cat.  The dog will make a
// bee line for the cat.  The cat will run in the opposite direction when the
// dog enters a certain distance.
create landmark dogstart {
        x 50;
        y 50;
        visible false;
}
create landmark catstart {
        x 150;
        y 150;
        visible false;
}

create rover dog {
        origin dogstart;
        destination cat;
        speed 4;
}
create rover cat {
        origin catstart;
        destination none;
        speed 3;
}

create rule cat {
        if (cat.distance_to(dog) < 20) {
                // cat will run from dog in opposite direction
                cat.destination(cat.bearing_to(dog) + 180, 20);
        }
}

create rule dog {
        if (dog.distance_to(cat) = 0) {
                destroy cat;
        }
}
```