Lightweight Simulation Scripting with Proto

Jacob Beal, Kyle Usbeck, Brian Krisler
Raytheon BBN Technologies
kusbeck@bbn.com
Spatial Computing Workshop @ AAMAS 2012

Work partially sponsored by DARPA; the views and conclusions contained in this document are those of the authors and not DARPA or the U.S. Government.
Serious Games

• Training
  – Reduce classroom lecture
  – Promote *active* learning

• US Navy VESSEL trainer
Game Engines

• Simplify creating complex, realistic simulations
• De-couples agent and terrain modeling and visualization (e.g., rendering, lighting, geotypical terrain)
Problem

- Every game engine has a scripting API
- APIs allow control of all objects in the game
- Game Engines are limited in their support for quickly and easily scripting behaviors of large groups of autonomous agents
- Multi-Agent System (MAS) toolkits and simulators lack realism and features for spatial-aggregate programming
Spatial-Aggregate Programming

Shibuya Crossing, Tokyo

http://www.youtube.com/watch?v=P5vuWJft32g
Solution

• Combine modern game engine with **spatial** approach to scalable multi-agent behavioral scripting
Unity

• What is Unity?
• Why Unity?
  – Realistic physics simulator
  – Simple/Realistic terrain modeling
  – Online market for “assets”
• What is Proto?
• Why Proto?
  – Global-to-local compiler
  – Extensible VM / Simulator Design
Approach

• Proto’s global-to-local compiler & VM
• Unity’s simulation environment
• Novel agent scripting library:
  – Group behavior primitives
  – Imperative-style scripting
Architecture

- Unity
- Proto Plug-in
- Runtime Execution Loop
- Global-to-Local Compiler
- Proto Virtual Machine for Unity
- Proto Compiler
- Agent Scripting Library
We designed a Unity plug-in for Proto that invokes Proto’s compiler, which in-turn creates byte-code to be executed by the virtual machine(s).
We created a Unity plug-in that implements the required platform-specific functions from the Proto virtual machine reference implementation using tools from the Unity API.
We created an agent scripting library that extends the Proto language with group behavior primitives and imperative-style macros.
Group Behavior Primitives

Random Walk  Flock / Flock-to  Cluster-by

Toward  Disperse / Scatter
Imperative-Style Agent Scripting

• Proto is a pure-functional language based on LISP.
• Doesn’t map well to the typical agent scripting user’s imperative approach.

LISP is over half a century old and it still has this perfect, timeless air about it.

I wonder if the cycles will continue forever.

These are your father’s parentheses.

A few coders from each new generation re-discovering the Lisp arts.

Elegant weapons for a more... civilized age.
Imperative-Style Agent Scripting

• Macro functionality added to Proto
• Added macros to make Proto read more sequentially, event-driven, and/or behaviorally

```
(def red-advance (red-team blue-team)
  (group-case
    (behavior-of red-team ;; Red team behavior:
      (where in-group
        (flock-to (tup 0 0))) ;; go to Blue starting location
    (behavior-of blue-team ;; Blue team behavior:
      (on-trigger (can-see red-team) ;; when Red is near...
        (scatter (away-from red-team))) ;; flee from Red!
      (default (tup 0 0)))))))
```
Agent Scripting Library

(group-case
  (behavior-of MEMBERSHIP-TEST BEHAVIOR
  (behavior-of MEMBERSHIP-TEST BEHAVIOR ...
  (default BEHAVIOR)...))

(priority-list
  (priority NAME TEST BEHAVIOR
  (priority NAME TEST BEHAVIOR ...
...))))

(when TEST BEHAVIOR)

(on-trigger TRIGGER BEHAVIOR)

(sequence
  ([stage | group-stage] NAME ACTION TERMINATION
  ([stage | group-stage] NAME ACTION TERMINATION ...
  [end-sequence | repeat])...))

Functional composition still applies!

Just a sampler... More to come!
Example: **Advance & Flee!**

```
(def red-advance (red-team blue-team)
  (group-case
    (behavior-of red-team ;; Red team behavior:
      (where in-group
        (flock-to (tup 0 0))) ;; go to Blue starting location
    (behavior-of blue-team ;; Blue team behavior:
      (on-trigger (can-see red-team)
        (scatter (away-from red-team))) ;; when Red is near...
      (default (tup 0 0))))))
```
Example: Deploy

(def deploy (squadID))
(sequence
(stage leave-vehicle
 (flock (tup -1 0 0))
 (timeout 20))
;; First stage:
;; move left...
;; ... for twenty seconds.

(stage group-by-squad
 (cluster-by squadID)
 (timeout 50))
;; Second stage:
;; group into squads...
;; ... for fifty seconds.

(stage deploy-to-destination
 (group-case
  (behavior-of (= squadID 0)
   (flock-to (tup 50 100))
   ;; First squad ...
   ;; ... goes to (50, 100)
  (behavior-of (= squadID 1)
   (flock-to (tup -200 0))
   ;; Second squad ...
   ;; ... goes to (-200, 0)
  (behavior-of (= squadID 2)
   (flock-to (tup -100 -100))
   ;; Third squad ...
   ;; ... goes to (-100, -100)
  (default (tup 0 0)))))
;; Sequence doesn’t end or repeat

ongoing

end-sequence))))
Code Comparison

(def flock (dir)
  (rep v
    (tup 0 0 0)
    (let ((d (normalize
             (int-hood
               (if (< (nbr-range) 5)
                   (* -1 (normalize (nbr-vec))))
               (if (> (nbr-range) 10)
                   (* 0.2 (normalize (nbr-vec))))
               (normalize (nbr v))))))
     (+ dir (mux (> (vdot d d) 0) d v)))))

var Controller : GameObject;
var var inited = false;
private var minVelocity : float;
private var maxVelocity : float;
private var randomness : float;
private var chance : GameObject;

function Start () {
  StartCoroutine("boldSteering");
}

function boldSteering () {
  while(true) {
    if (inited) {
      rigidbody.velocity = rigidbody.velocity + calc() * Time.deltaTime;
      // enforce minimum and maximum speeds for the boids
      var speed = rigidbody.velocity.magnitude;
      if (speed > maxVelocity) {
        rigidbody.velocity = rigidbody.velocity.normalized * maxVelocity;
      } else if (speed < minVelocity) {
        rigidbody.velocity = rigidbody.velocity.normalized * minVelocity;
      }
    }
    waitTime = Random.Range(0.3, 0.5);
    yield WaitForSeconds(waitTime);
  }
}

function calc () {
  var randomize = Vector3((Random.value + 2) -1, (Random.value + 2) -1, (Random.value + 2) -1);
  randomize.Normalize();
  flockCenter = Controller.GetComponent("Bold Controller").flockCenter;
  flockVelocity = Controller.GetComponent("Bold Controller").flockVelocity;
  follow = chase.transform.localPosition;
  flockCenter = flockCenter - transform.localPosition;
  flockVelocity = flockVelocity - rigidbody.velocity;
  follow = follow - transform.localPosition;
  return ((flockCenter + flockVelocity + follow*2 + randomize*randomness));
}

function setController (theController : GameObject) {
  Controller = theController;
  minVelocity = Controller.GetComponent("Bold Controller").minVelocity;
  maxVelocity = Controller.GetComponent("Bold Controller").maxVelocity;
  ...}
Benefits

• Scalable
  – Supports large numbers of agents
  – Scripts remain constant with dynamic numbers of agents

• Lightweight
  – Small memory and CPU profile

• Realistic movement – agents are affected by their environment (e.g., collision, gravity, etc.)

• Robust to behavioral changes – both during programming and during game-play
Future Work

• Proto Plug-ins for Unity-specific operators / controls
  – Line-of-sight (including terrain obstacles)
  – Operator feedback (e.g., “Agent can’t run at 5 mph in that direction because it would be up a hill.”)

• Adding to group behavior primitives and agent scripting library
Join the Proto Community

http://proto.bbn.com