

# Lightweight Simulation Scripting with Proto

Jacob Beal, Kyle Usbeck, Brian Krisler

Raytheon BBN Technologies

[kusbeck@bbn.com](mailto:kusbeck@bbn.com)

Spatial Computing Workshop @ AAMAS 2012

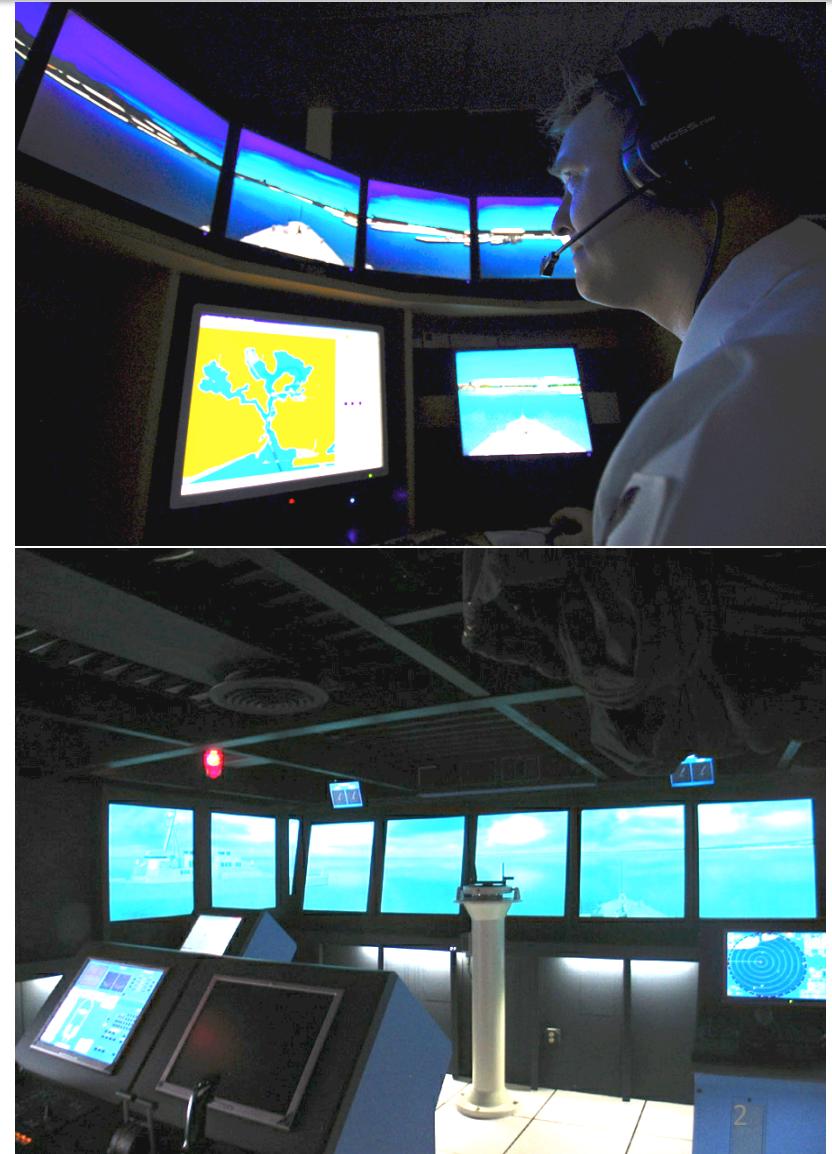
Work partially sponsored by DARPA; the views and conclusions contained in this document are those of the authors and not DARPA or the U.S. Government.



# Serious Games

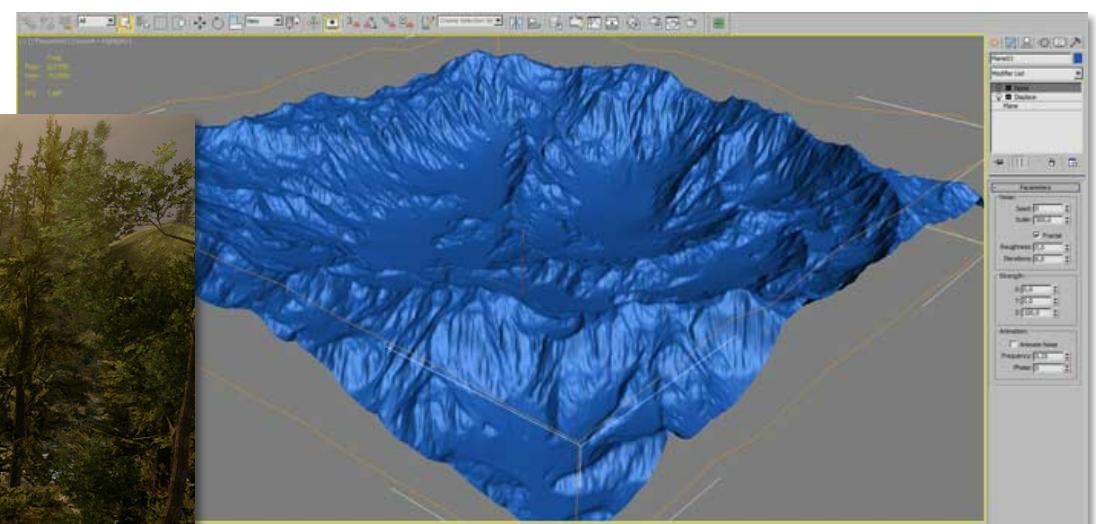
**Raytheon  
BBN Technologies**

- 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

**Raytheon**  
**BBN Technologies**



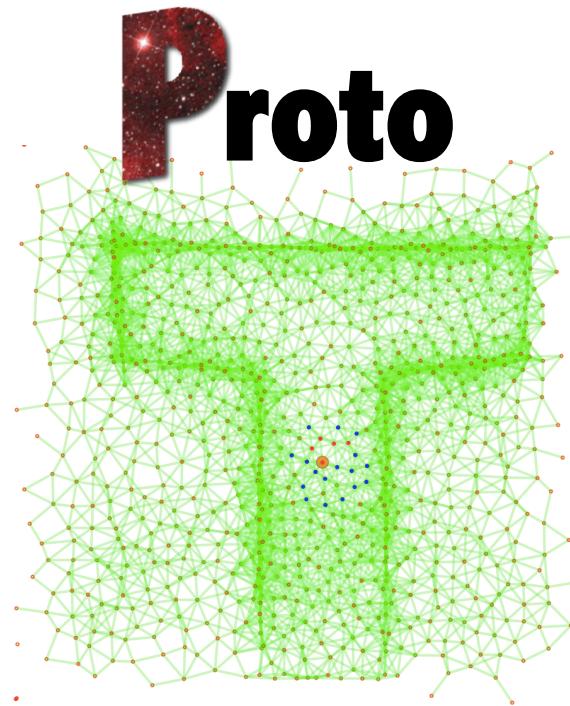
Shibuya Crossing, Tokyo

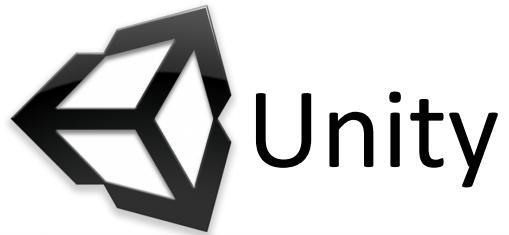
<http://www.youtube.com/watch?v=P5vuWJft32g>

# Solution

**Raytheon  
BBN Technologies**

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



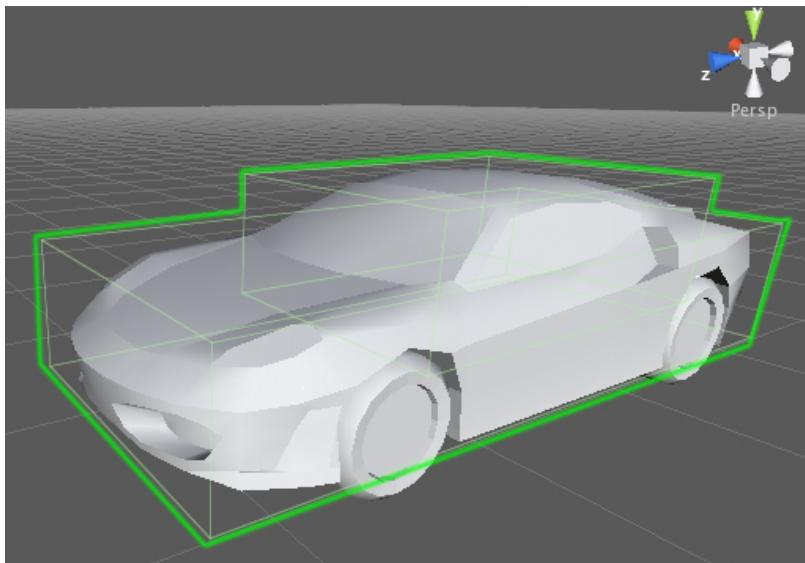


**Raytheon  
BBN Technologies**

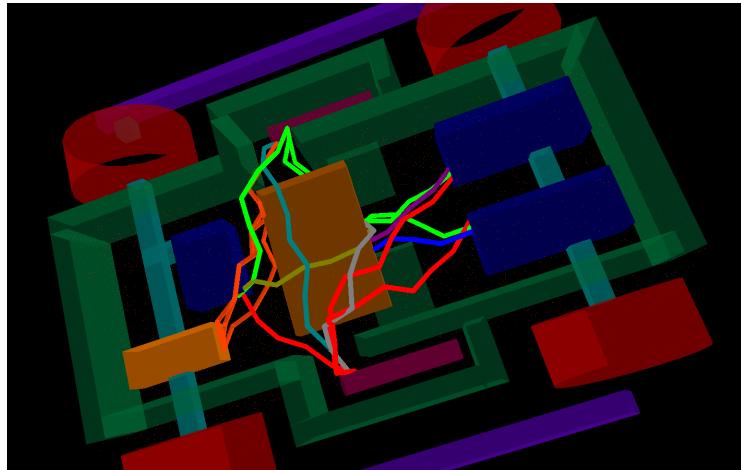
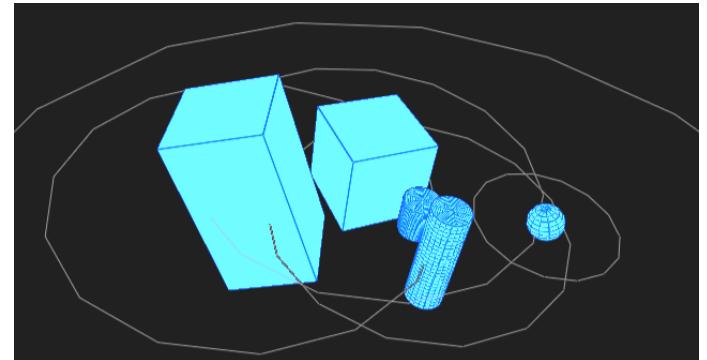
- What is Unity?
- Why Unity?
  - Realistic physics simulator
  - Simple/Realistic terrain modeling
  - Online market for “assets”

The screenshot shows the Unity Asset Store homepage with a prominent banner for 'ASSET STORE MADNESS'. The banner features the text 'SAVE BIG ON TOP PACKAGES' and 'ASSET STORE MADNESS' in large, bold letters. Below the banner, there's a section titled 'Accelerate your development' with a brief description of what the Asset Store offers. To the right, there's a sidebar for 'Asset Store Madness' (valid until 23rd June) listing several items with their names, ratings, and prices:

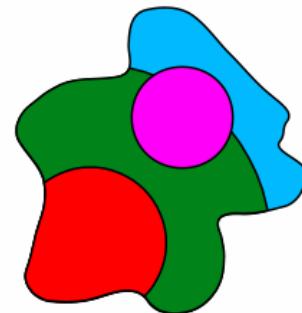
Item	Rating	Original Price	Sale Price
Antares Universe (VIZIO)	★★★★★	\$150.00	\$75.00
iGUI	★★★★★	\$95.00	\$45.00
Bitmap2Material	★★★★★	\$150.00	\$75.00
Smooth Moves	★★★★★	\$75.00	\$35.00
NGUI: Next-Gen UI	★★★★★	\$95.00	\$45.00



- What is Proto?
- Why Proto?
  - Global-to-local compiler
  - Extensible VM / Simulator Design

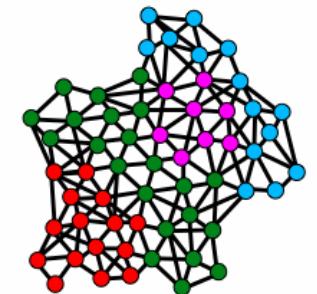


Continuous Specification



approximate

Discrete Implementation



# Approach

---

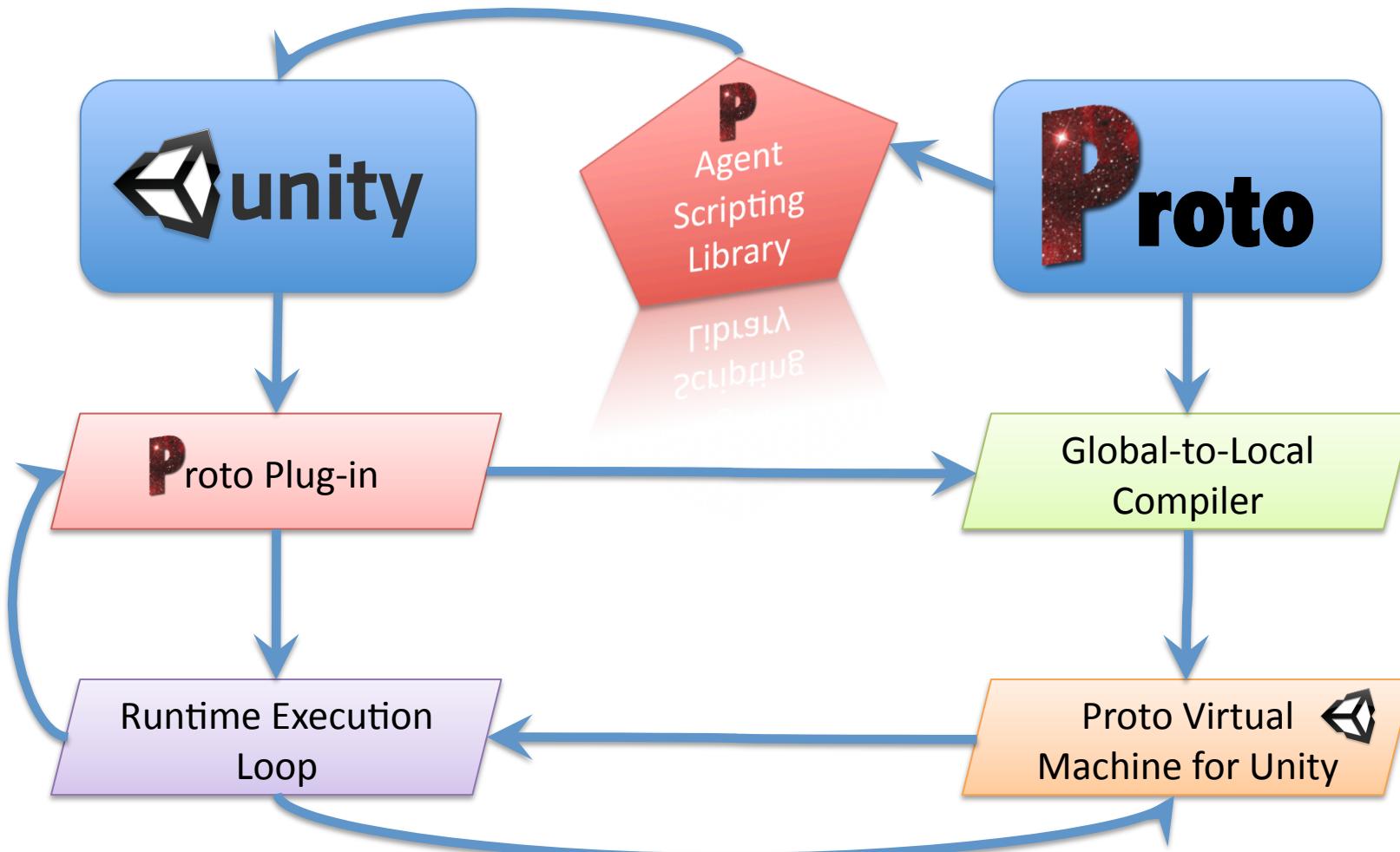
**Raytheon  
BBN Technologies**

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

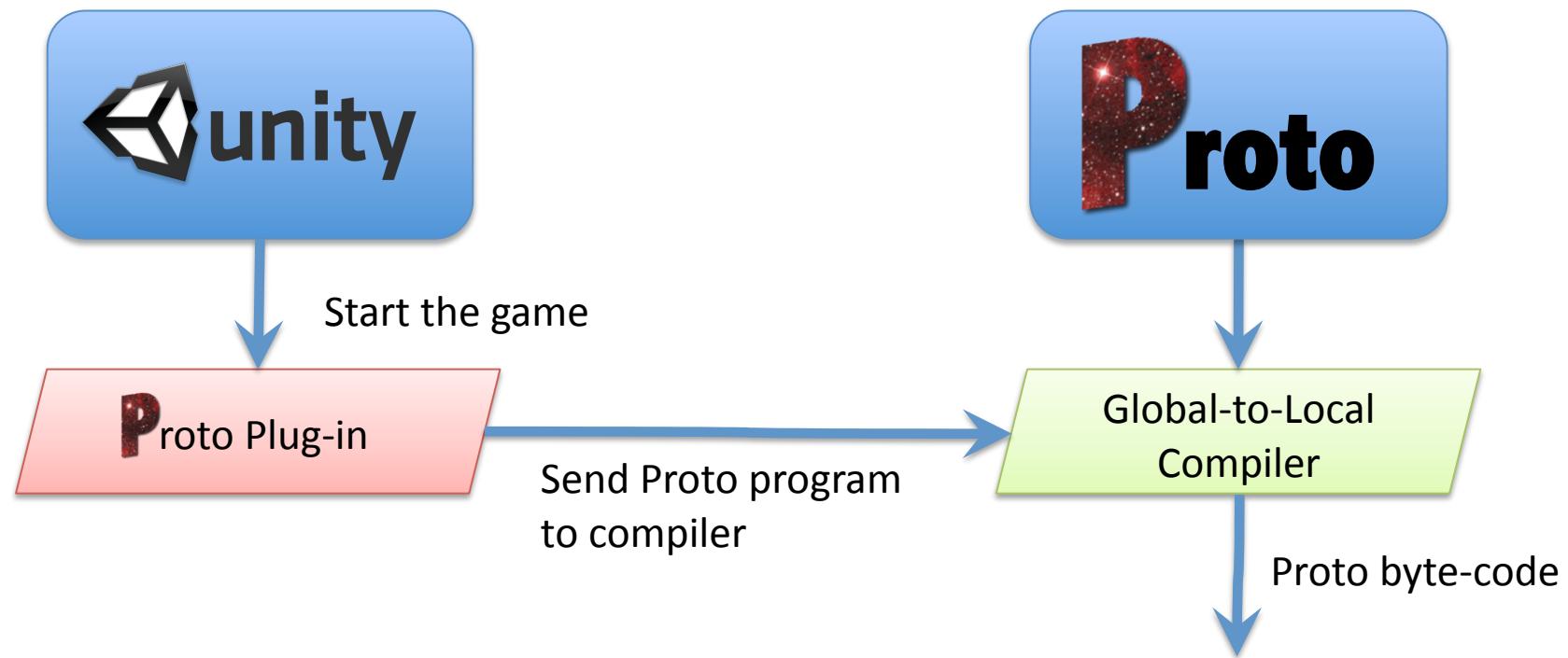


# Architecture

**Raytheon  
BBN Technologies**



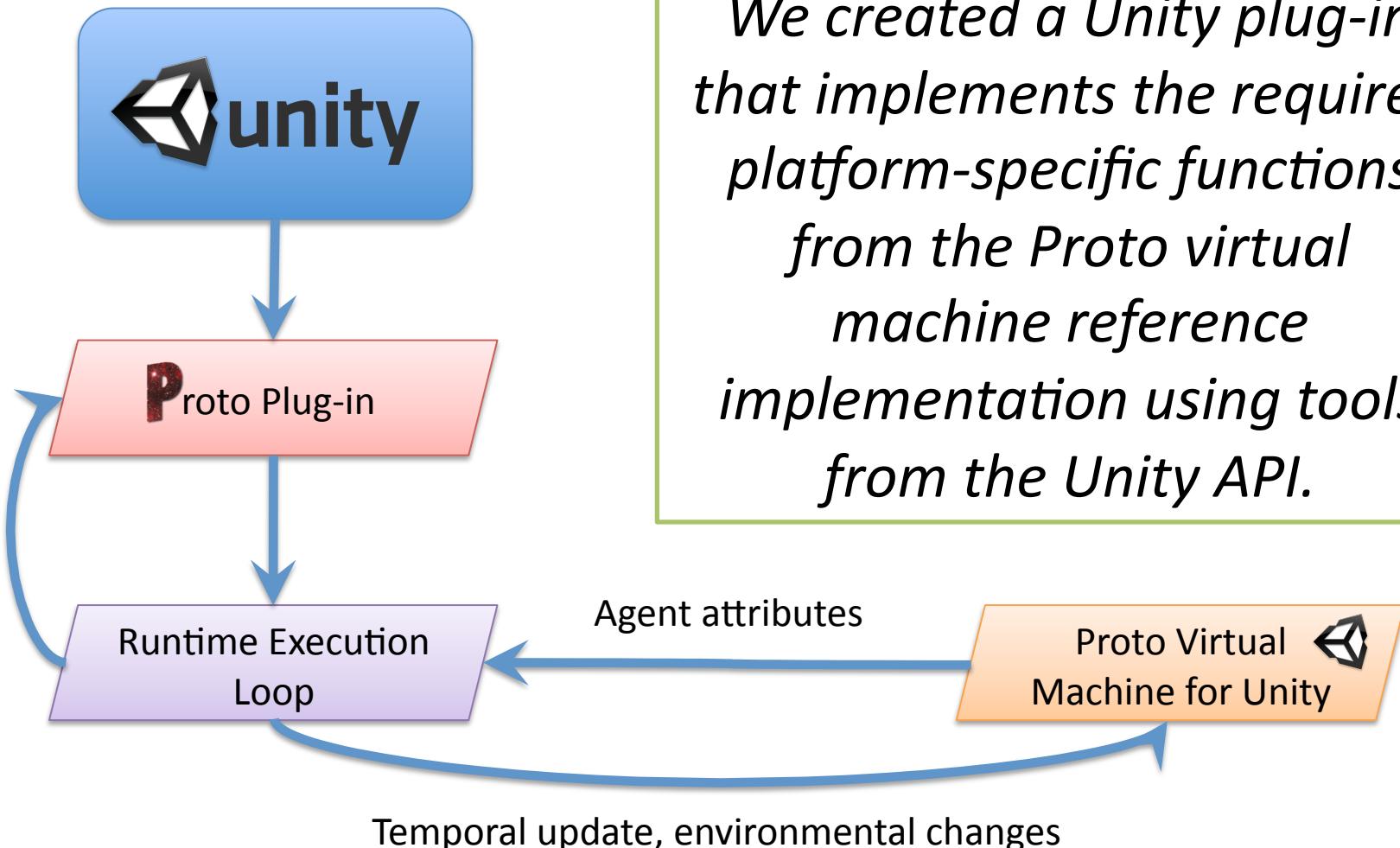
# Invoking the Proto Compiler



*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).*

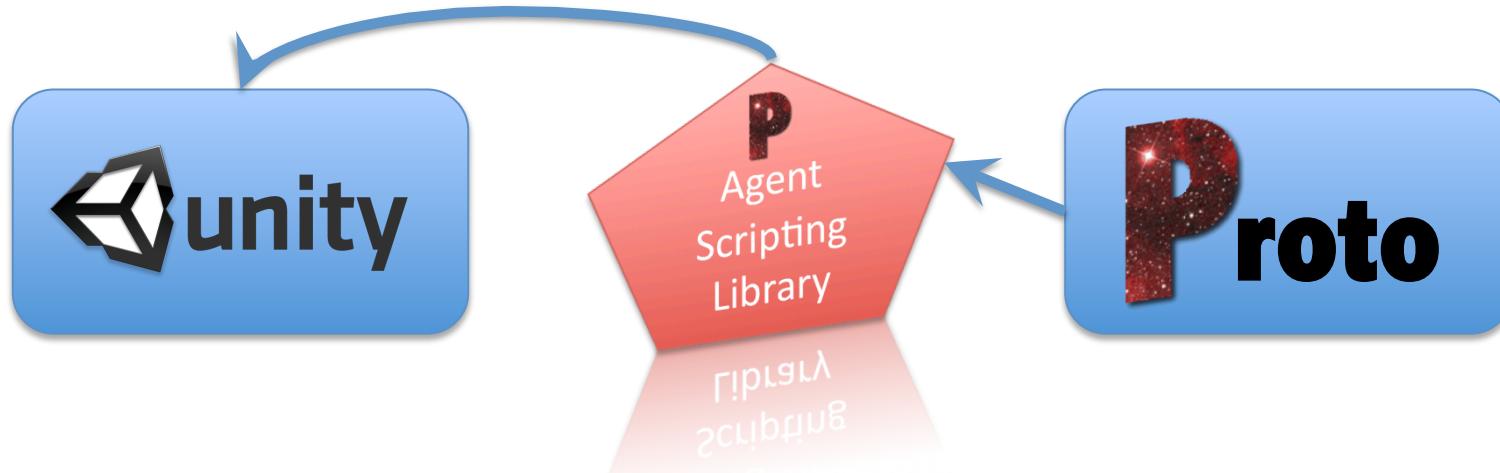
# A Proto VM Implementation for Unity

**Raytheon  
BBN Technologies**



# Agent Scripting Library

**Raytheon  
BBN Technologies**



*We created an agent scripting library that extends the Proto language with group behavior primitives and imperative-style macros.*

# Group Behavior Primitives

**Raytheon**  
**BBN Technologies**



Random Walk



Flock / Flock-to



Cluster-by



Toward



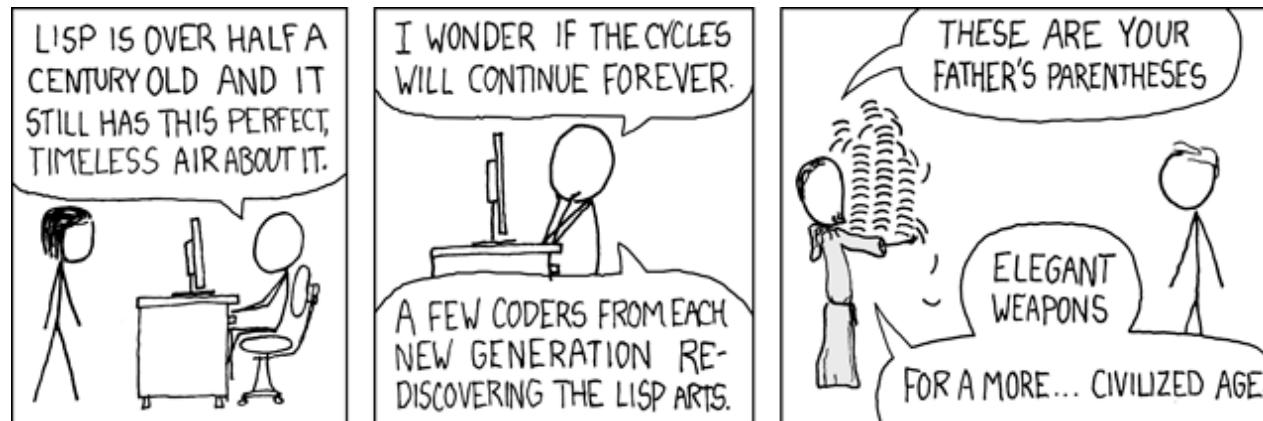
Disperse / Scatter

# Imperative-Style Agent Scripting

---

**Raytheon**  
**BBN Technologies**

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



# 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)
      (scatter (away-from red-team))) ;; when Red is near...
    (default (tup 0 0)))))
```

# Agent Scripting Library

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

```
(where TEST BEHAVIOR)
```

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

```
(on-trigger TRIGGER BEHAVIOR)
```

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

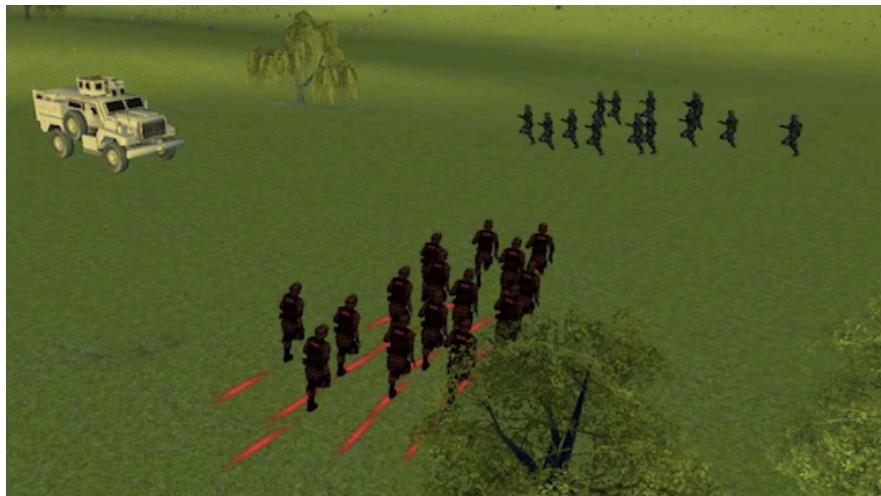
Functional composition  
still applies!

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)   ;; when Red is near...
      (scatter (away-from red-team))) ;; flee from Red!
    (default (tup 0 0))))))

```



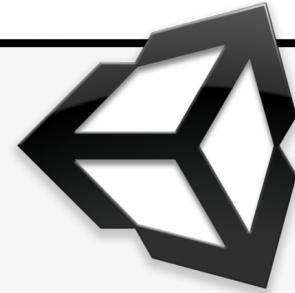
# Example: Deploy



```
(def deploy (squadID)
  (sequence
    (stage leave-vehicle           ;; First stage:
           (flock (tup -1 0 0))   ;; move left...
           (timeout 20)           ;; ... for twenty seconds.
    (stage group-by-squad         ;; Second stage:
           (cluster-by squadID) ;; group into squads...
           (timeout 50)           ;; ... for fifty seconds.
    (stage deploy-to-destination ;; Third stage:
      (group-case             ;; Each squad goes to a different location:
        (behavior-of (= squadID 0) ;; First squad ...
          (flock-to (tup 50 100)) ;; ... goes to (50, 100)
        (behavior-of (= squadID 1) ;; Second squad ...
          (flock-to (tup -200 0)) ;; ... goes to (-200, 0)
        (behavior-of (= squadID 2) ;; Third squad ...
          (flock-to (tup -100 -100)) ;; ... goes to (-100, -100)
        (default (tup 0 0)))))))
      ongoing                      ;; Sequence doesn't end or repeat
    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))))))))
      (normalize
        (+ dir (mux (> (vdot d d) 0) d v)))))))
```



```
var Controller : GameObject;

private var initied = false;
private var minVelocity : float;
private var maxVelocity : float;
private var randomness : float;
private var chsee : GameObject;

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

function boidSteering () {
    while(true) {
        if (initied) {
            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);
    }
}

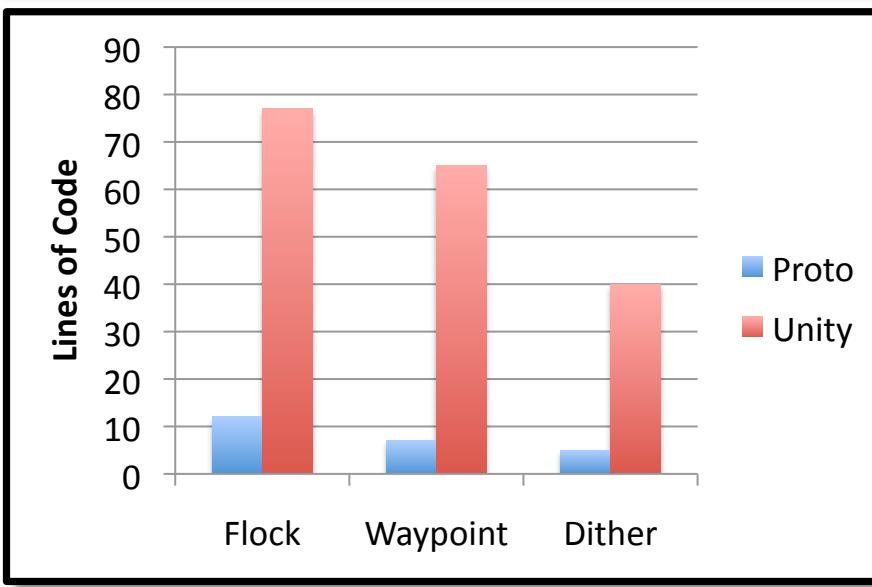
function calc () {
    var randomize = Vector3((Random.value * 2) -1, (Random.value * 2) -1, (Random.value * 2) -1);
    randomize.Normalize();

    flockCenter = Controller.GetComponent("Boid Controller").flockCenter;
    flockVelocity = Controller.GetComponent("Boid Controller").flockVelocity;
    follow = chsee.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("Boid Controller").minVelocity;
    maxVelocity = Controller.GetComponent("Boid Controller").maxVelocity;
    randomness = Controller.GetComponent("Boid Controller").randomness;
}
```



# 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

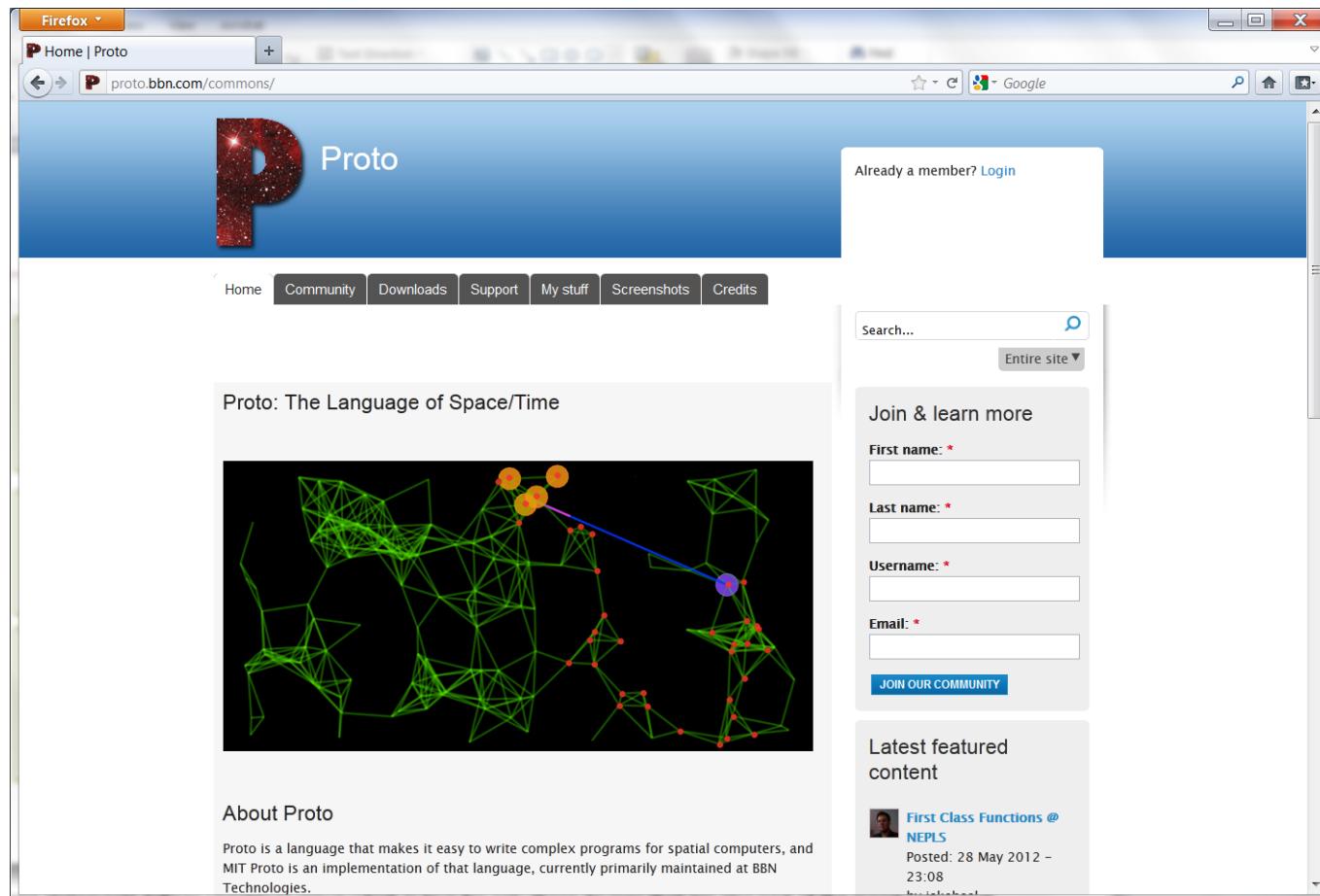
# Future Work

---

- 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

**Raytheon  
BBN Technologies**



*<http://proto.bbn.com>*