Behavior Trees: Three Ways of Cultivating Game AI
BEHAVIOR TREES APPLIED!

- Halo 3 & ODST
- [PROTOTYPE]
- Spore
- GTA: Chinatown Wars
- The Bourne Conspiracy
- SWAT 4, Bioshock
- Dark Sector
- ...
FEATyRES

• BTs are a framework for game AI.

• BTs model character behaviors extremely well.

• BTs are simple, yet extremely customizable.
OVERVIEW

Behavior

Behavior

Behavior

Behavior

Behavior

Action

Action

Action

Action
AGENDA

1) Building Blocks
2) Design Patterns
3) Script Integration
4) Debugging
5) Discussion
NODE TYPES

• Priority
  – Child nodes are evaluated in order until one validates

• Sequential
  – First child is validated and executed
  – When it is finished, the next one is validated

• Stochastic
  – All children are validated
  – A random node is selected among the valid ones
BEHAVIOR TREE UPDATE

Root

Combat

- Melee
  - Flee
  - Attack
- Weapon 1
- Weapon 2
- Weapon 3

Ranged

Idle

- Use Computer
- Patrol
EVENT-DRIVEN BEHAVIORS

- Stimulus types
  - Disabled by event
  - Autodisabled
DYNAMIC BEHAVIORS

• Dynamic behaviors support
  – Level specific content
    • Patrols
    • Initial setups
    • Story driven events
  – DLC

• Behaviors are added to actors in the level (enticers)
  – When a NPC uses the actor, it attaches the behavior to the tree
DYNAMIC BEHAVIORS

- Root
  - Combat
    - Melee
      - Flee
    - Attack
    - Weapon 1
    - Weapon 2
    - Weapon 3
  - Ranged
  - Idle
    - Use Computer
    - Patrol
DYNAMIC BEHAVIORS

- **Validate**
  - Look for enticers

- **Update**
  1. Move to enticer
  2. Wait for other NPCs
  3. Subscribe
     - Attach new behavior to the tree
  4. Wait for behavior to finish
  5. Unsubscribe
     - Remove behavior from the tree
IN THE NEXT 10 MINUTES, FIND OUT...

What’s the biggest problem developers face working with behavior trees and scaling up?
IN THE NEXT 10 MINUTES, FIND OUT...

When should you build your BT like a HFSM, and what happens if you do?
RunToTarget

BLACKBOARD

COVER

MOVING

STANDING

CROUCHING
BT DESIGN TAKE AWAY

• Decouple your BT from the problem at hand, for example using a blackboard.

• Build purposeful behaviors as sequences of short goal-directed actions.

• Be careful with “state-like” behaviors that keep running.

• Leverage the power of the tree search!
Behavior Trees Part 3, Michael

SCRIPT INTEGRATION
SCRIPT INTEGRATION

• Behavior trees are all about flexibility
  – Selector choice!
  – Reuseable goals!
• Rapid iteration is a key goal
  – Bigger games, more actors, individualized behaviors
  – Need to quickly change in response to prototyping and playtest
• Separate the algorithm from the behaviors
FLEXIBILITY FROM SCRIPT

• Lua at BHG, but could be any language
• What you’ll need:
  – Scripting language integration
    • Calling script from code and vice versa
• Really nice to have:
  – Designers comfortable with scripting
    • You will need support time (more on that later)
  – Script debugging
What behaviors want to run?

Type/precondition results

Calls to on_exit, on_enter, behavior

Gather

Run

C++

Lua
CREATING A BEHAVIOR SCRIPT

• Behaviors have a common structure
  – Precondition
  – Behavior

• Optional components
  – Type (priority, sequential, random)
  – on_enter, on_exit
  – Whatever else you decide your behaviors need

• In Lua, these can be known function names in a table
BEHAVIOR SCRIPT

```
follow_basic = {
    precondition =
        function(object)
            local formation_leader = behavior.get_my_follow_object(object)
            return formation_leader ~= 0 and ACTOR.is_farther_than(object, formation_leader, exposed.follow_far_dist)
        end,
    behavior =
        function(object)
            ACTOR.move_to_form_position(object)
            ACTOR.update_leashing_home_location(object)
        end,
    on_enter =
        function(object)
            local formation_leader = behavior.get_my_follow_object(object)
            if not ACTOR.is_inFormation(object) then
                ACTOR.add_to_Formation(object, formation_leader, exposed.follow Formation_line)
            end
            ACTOR.look_at_set_target_object(object, "GLANCE", get_player(), "Head")
        end,
    on_exit =
        function(object)
            ACTOR.look_at_clear_target(object)
        end,
}
```
CREATING A TREE WITH BEHAVIOR SCRIPTS

• First implementation: Scripts that create behavior trees
• Lua functions to add, remove, insert behaviors from an existing tree
  – add_behavior(tree, behavior)
• Great flexibility, but hard to conceptualize
  – Creating trees in script was difficult to grasp
  – Especially when trying to reuse trees you didn’t write
BEHAVIOR TREE TOOL

• External .NET app to manage trees and behaviors
• Easy to create new behaviors or reuse existing ones
• Statistics on commonly used behaviors
• Search for behavior/tree by name or usage
BEHAVIOR TREE TOOL
BENEFITS OF USING SCRIPT

• Designers write behaviors so you don’t have to
  – Currently 63 unique behaviors in our game
    • I wrote 7
  – Lots of time back for other tasks
• Faster implementation and iteration
  – No rebuilding code
  – Can reload scripts while the game runs
    – Need prep for this; flush behaviors, cached names, pointers?
COMMON QUESTIONS

• Performance-related
  – “Isn’t scripting slow?”
  – “How do you stay under CPU budget?”

• Behavior creation-related
  – “Are designers scripting well?”
  – “What if my designers aren’t scripters?”
KEEPING SCRIPT FAST

• **Don’t let it be slow!**
  – BHG limits lua to integer math
  – Prevent mid-frame garbage collection

• **Limit scripting to where it makes sense**
  – AI loop is not in script
  – No trig in script!
  – Anything “complicated enough” done in code

• **Could put behaviors to code for performance**
  – …but maybe not
DESIGNER SCRIPTING

• Good enough is great!
• Does take code support time
  – ~10% of my time debugging for designers
  – ~10% on function requests (trig, &c.)
• Watch for things that should be done in code
• Strength in speed! Don’t stifle creativity
• Plan on reviewing trees and behaviors periodically
DEBUGGING
# DEBUGGING

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Start</th>
<th>Finish</th>
<th>Think Result</th>
<th>Debug String</th>
</tr>
</thead>
<tbody>
<tr>
<td>TakeCover</td>
<td>238</td>
<td>238</td>
<td>Thinking</td>
<td>Shooting</td>
</tr>
<tr>
<td>GetInRange</td>
<td>237</td>
<td>238</td>
<td>Success</td>
<td>Completed</td>
</tr>
<tr>
<td>Shoot</td>
<td>206</td>
<td>207</td>
<td></td>
<td>Completed</td>
</tr>
<tr>
<td>GetInRange</td>
<td>202</td>
<td>203</td>
<td></td>
<td>Completed</td>
</tr>
<tr>
<td>Shoot</td>
<td>201</td>
<td>202</td>
<td></td>
<td>Completed</td>
</tr>
<tr>
<td>PlayAnimation</td>
<td>201</td>
<td>202</td>
<td></td>
<td>Completed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>121</td>
<td></td>
<td>Completed</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>121</td>
<td></td>
<td>Completed</td>
</tr>
</tbody>
</table>

- RenID: 669820585
- Current Behavior: Shoot
- Debug String: Not ready to shoot
- Debug String: Not ready to shoot

- RenId: 573115146
- Current Behavior: TakeCover
- Debug String: Not ready to shoot
- Debug String: Not ready to shoot

- RenId: 571014990
- Current Behavior: Shoot
- Debug String: Not ready to shoot
- Debug String: Not ready to shoot

- RenId: 563917748
- Current Behavior: Shoot
- Debug String: Not ready to shoot
- Debug String: Not ready to shoot
DEBUGGING
Behavior Trees: Three Ways of Cultivating Game AI