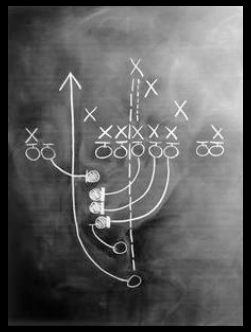
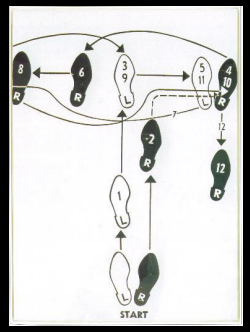
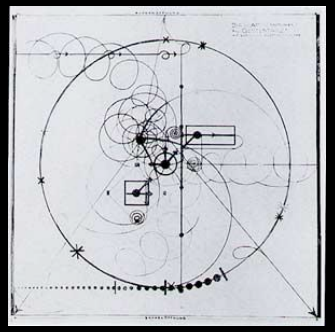
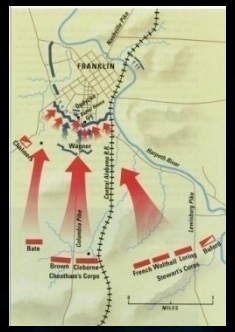
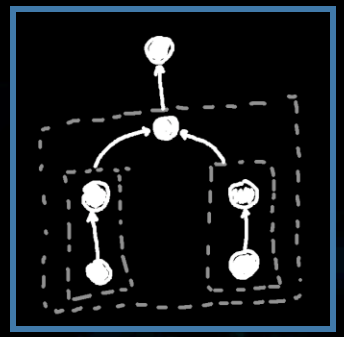
Building a Better Battle

The Halo 3 AI Objectives System



Damián Isla

Bungie Studios

BUNGIE

NON FACETE NOBIS CALCITRARE VESTRVM

Building A Better Battle

Designer tools

AI is an integral part of it

An interesting *Next-Gen* problem

BUNGIE

NON FACETE NOBIS CALCITRARE VESTRVM

“Big Battle” Technology

Precombat

Combat dialogue

Ambient sound

Scalable perception

Flocking

Encounter logic

Effects

Targeting groups

In-game cinematics

Scalable AI

Mission dialogue

“Big Battle” Technology

Activities

Combat dialogue

Ambient sound

Scalable perception

Flocking

Encounter logic

Effects

Targeting groups

In-game cinematics

Scalable AI

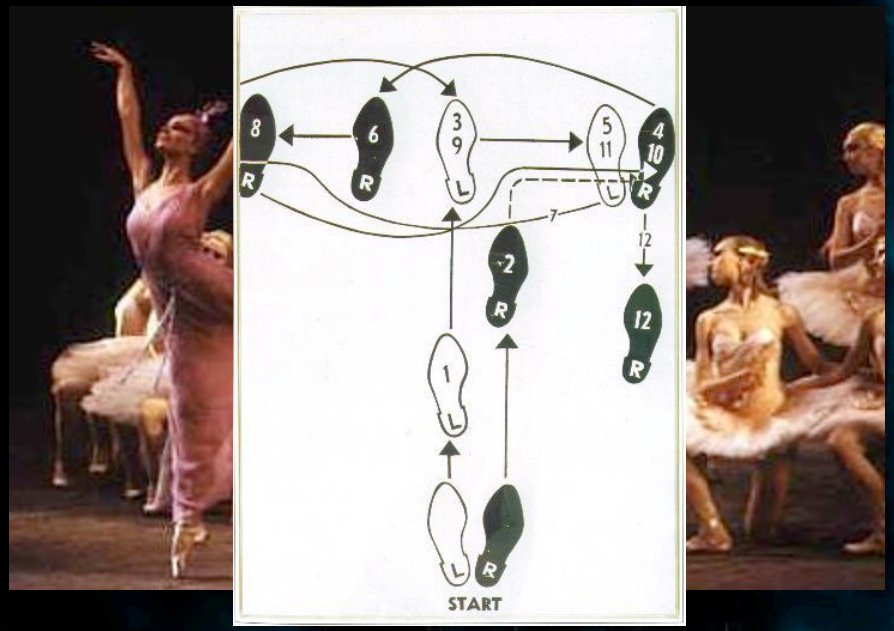
Mission dialogue

NON FACETE NOBIS CALCITRARE VESTRVM

BUNGIE

Encounter Design

- Encounters are *systems*
- Lots of guys
- Lots of things to do
- The system reacts in interesting ways
- The system collapses in interesting ways



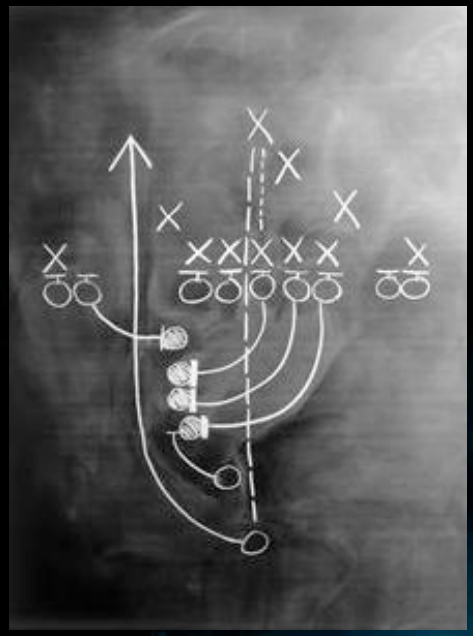
An encounter is a complicated dance with lots of dancers

How is this dance choreographed?

Choreography 101

- The dance is about the illusion of strategic intelligence
- Strategy is environment- story- and pacing-dependent

Designer provides the strategic intelligence



AI acts smart within the confines of the plan provided by the designer

The Canonical Encounter

Two-stage fallback

- Enemies occupy a territory
- Pushed to *"fallback"* point
- Pushed to *"last-stand"* point
- Player "breaks" them
- Player finishes them off

... plus a little "spice"

- snipers
- turrets
- dropships

Task

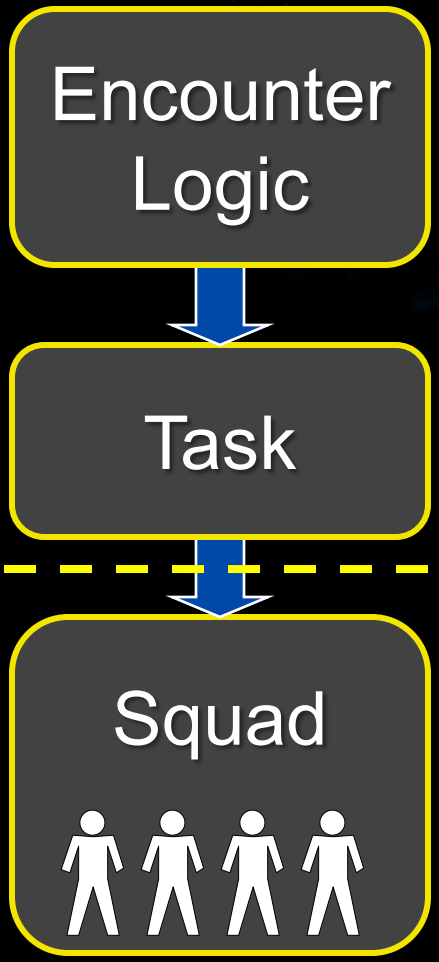
The *mission designers'* language for telling the AI what it should be doing

Halo:

- Territory
- Behavior
  - aggressiveness
  - rules of engagement
  - player following

Changing task moves AI around the encounter space

The Control Stack

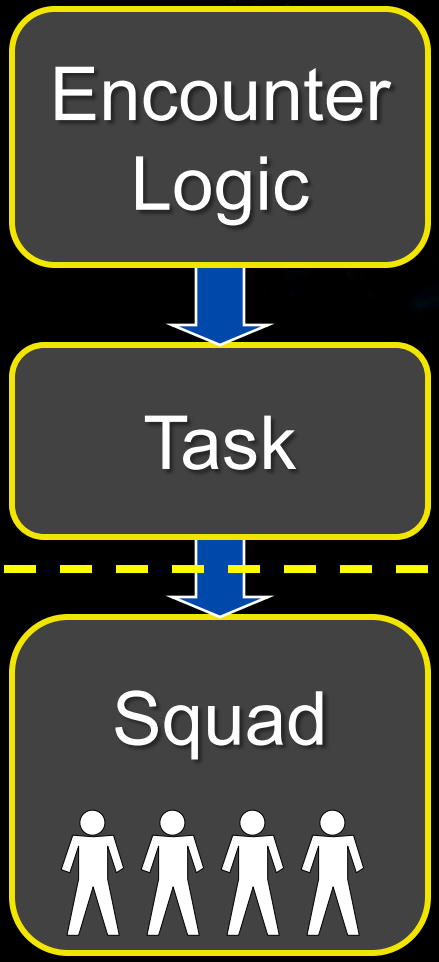


Mission-designers script sequence of tasks

Mission designers

AI engineers, AI designers Within the task, the AI behaves autonomously

The Control Stack



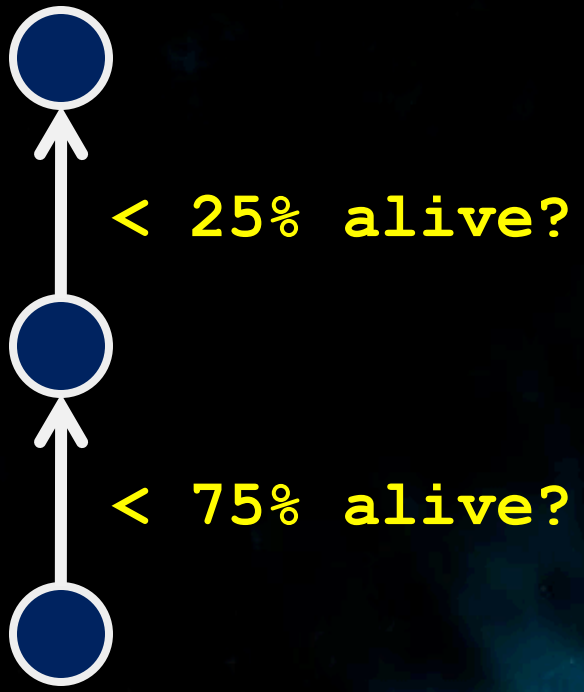
Mission-designers script sequence of tasks

~~Within the task, the AI behaves autonomously~~

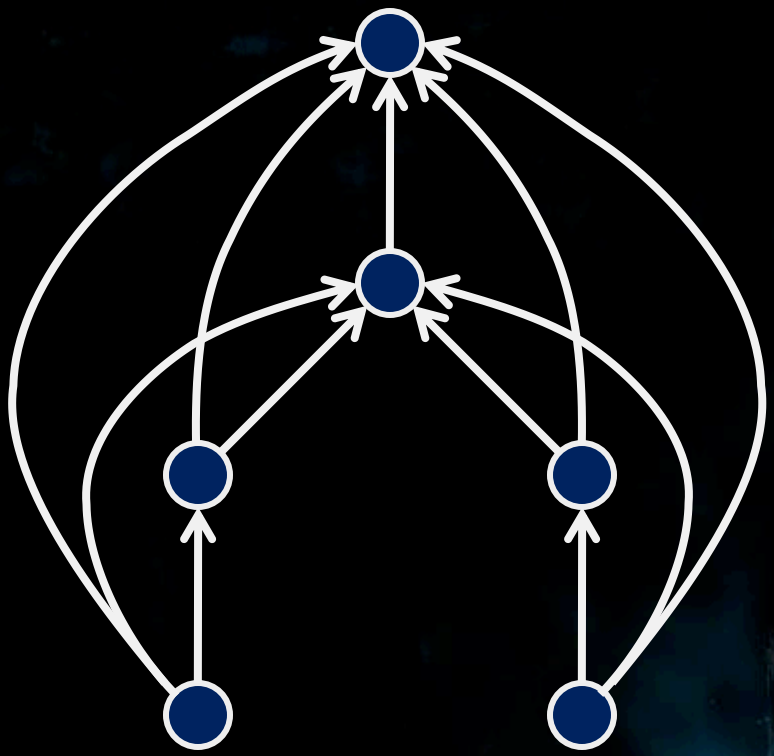
Halo 2: The Imperative Method

The Imperative Method

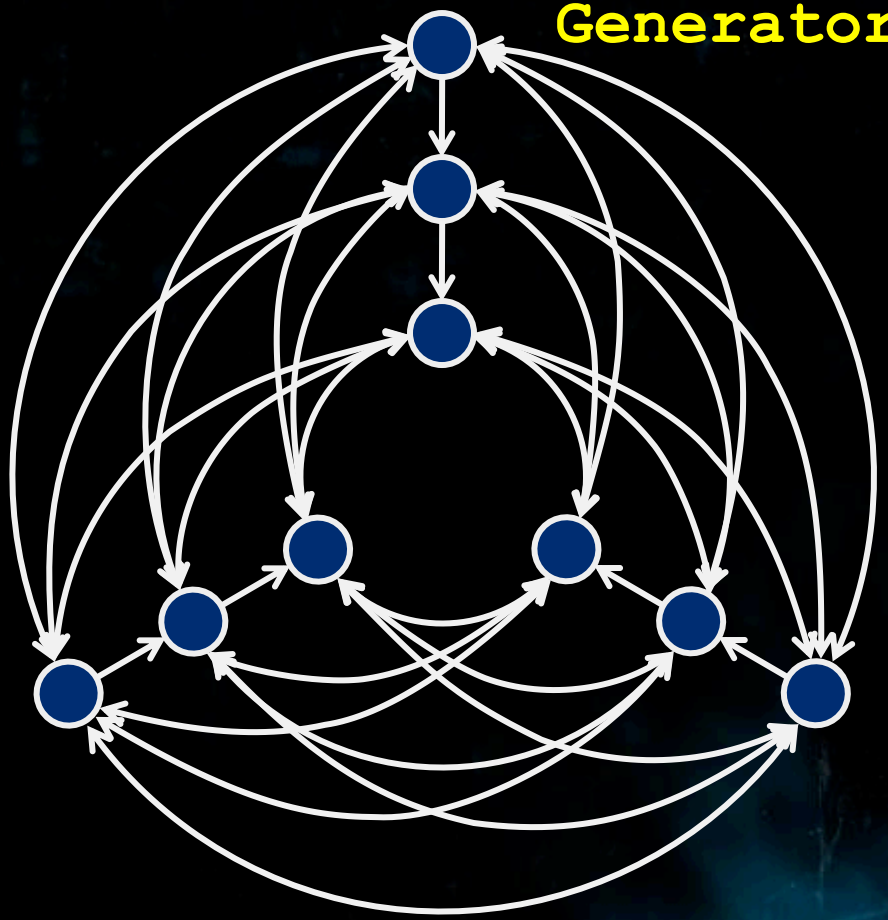
Give the designers an FSM construction tool



Problems with the Imperative Method



Problems with the Imperative Method



Explicit transitions → n^2 complexity

Problems with the Imperative Method

For Halo 3:

- Larger encounters
- More characters
- More open spaces
- More avenues of attack

Halo 3: The Declarative Method

BUNGIE

The Declarative Method

The new approach:

Enumerate “tasks that need doing” in the environment

Let the system figure out who should perform them

The Declarative Method

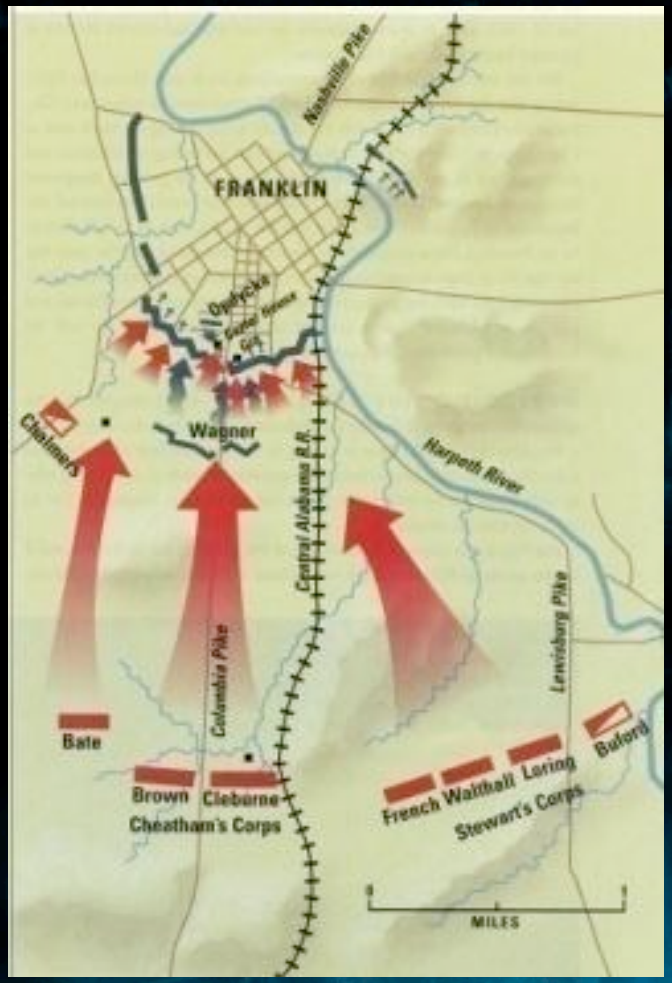
Not without precedent

Similar to “affordances”

The Declarative Method

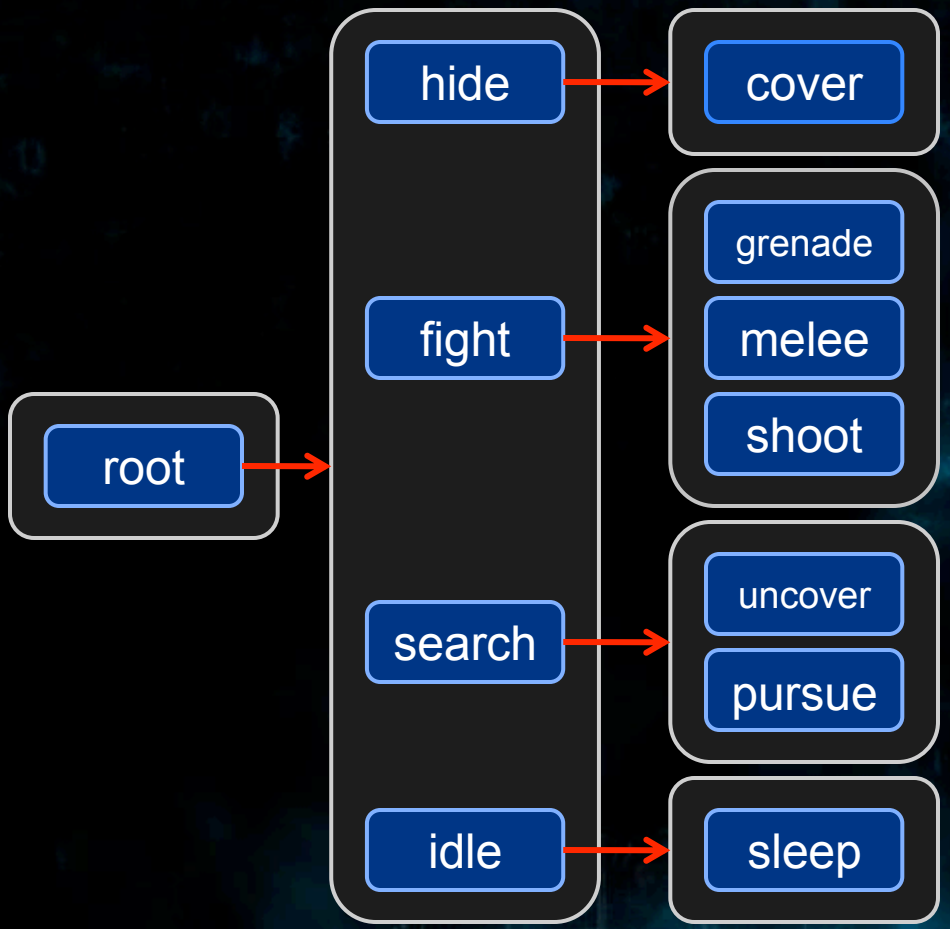
Tasks have *structure*

- Relative priorities
  - "The *most important* thing is to guard the door, but if you can, also guard the hallway"
- Are made up of sub-tasks
  - "Guarding the hallway means guarding the front, the middle and the rear of the hallway."



Behavior Trees

(*Handling Complexity in the Halo 2 AI*, GDC 2005)



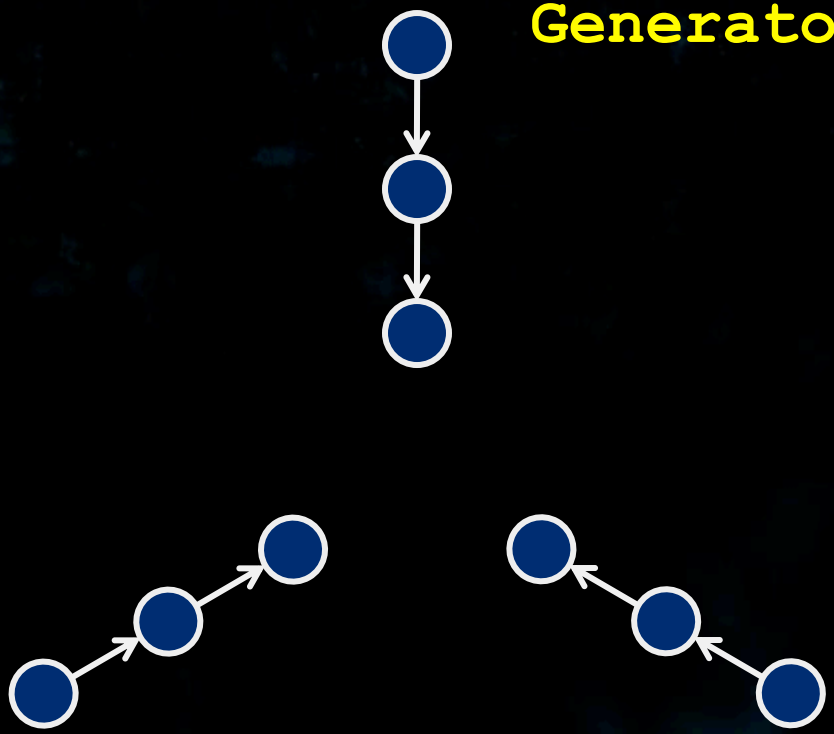
Takeaways:

1. Prioritized-list decision scheme
2. Behaviors are self-describing

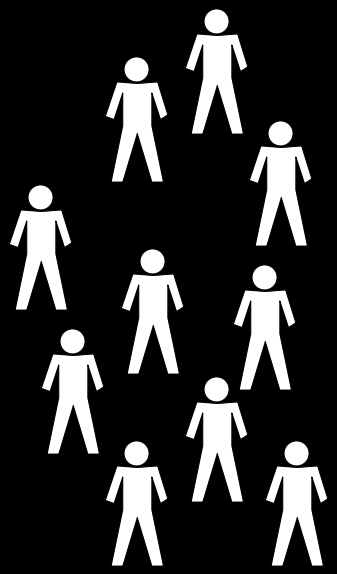
We are not making a *single* choice.

We are finding a *distribution* across *all* choices.

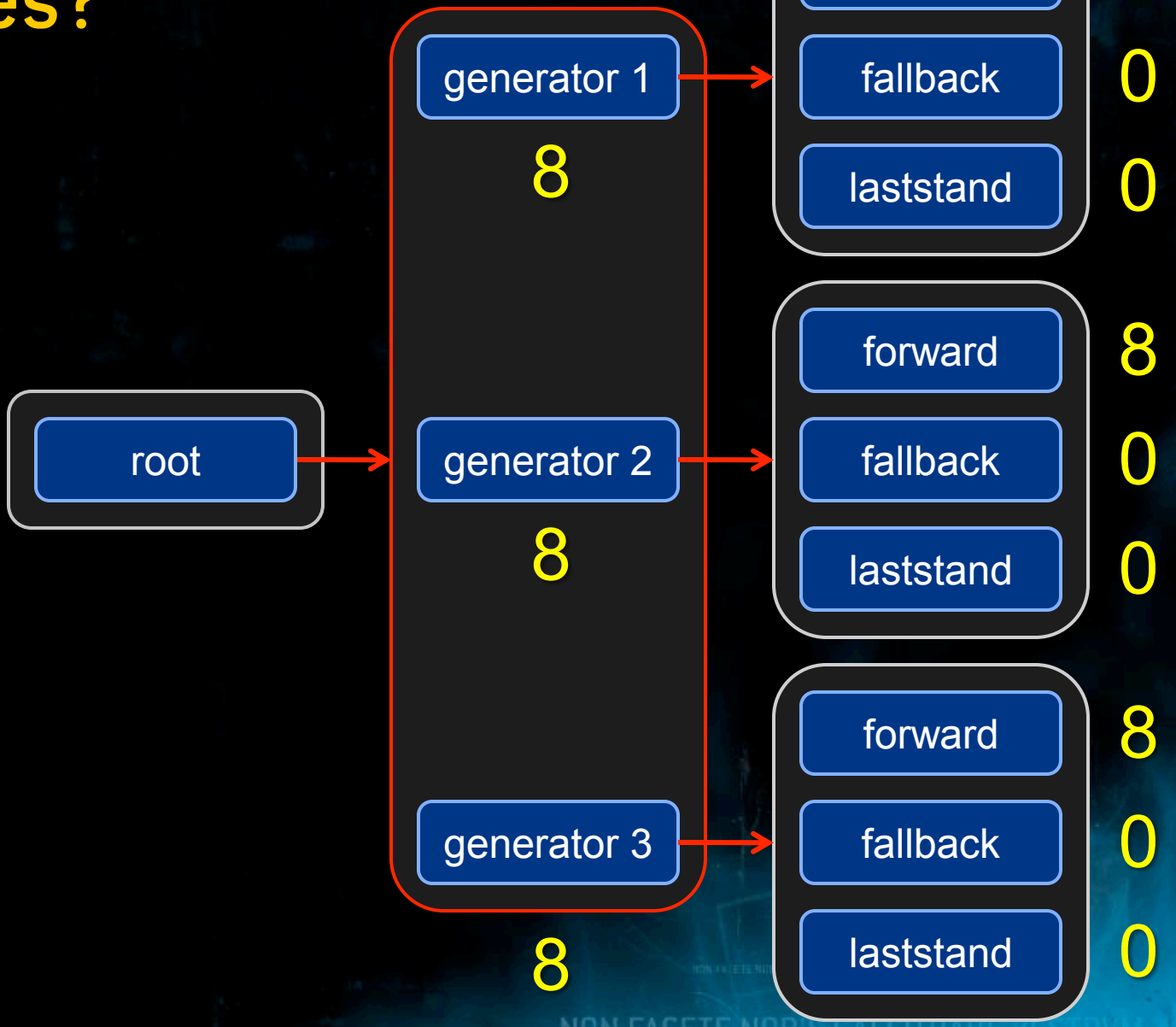
Task Trees?



Task Trees?



24 guys



Halo 3 AI Objectives System

The structure:

- A Tree of Prioritized *Tasks*
- Tasks are self-describing
  - priority
  - activation script-fragments
  - capacities

The Algorithm:

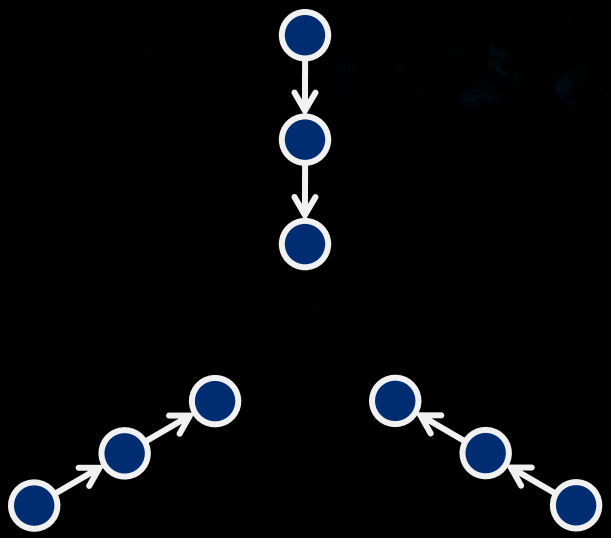
- Pour squads in at the top
- Allow them to filter down to the most important tasks to be filling RIGHT NOW

Basically, it's a plinko machine.

The *Dynamic* Plinko Machine

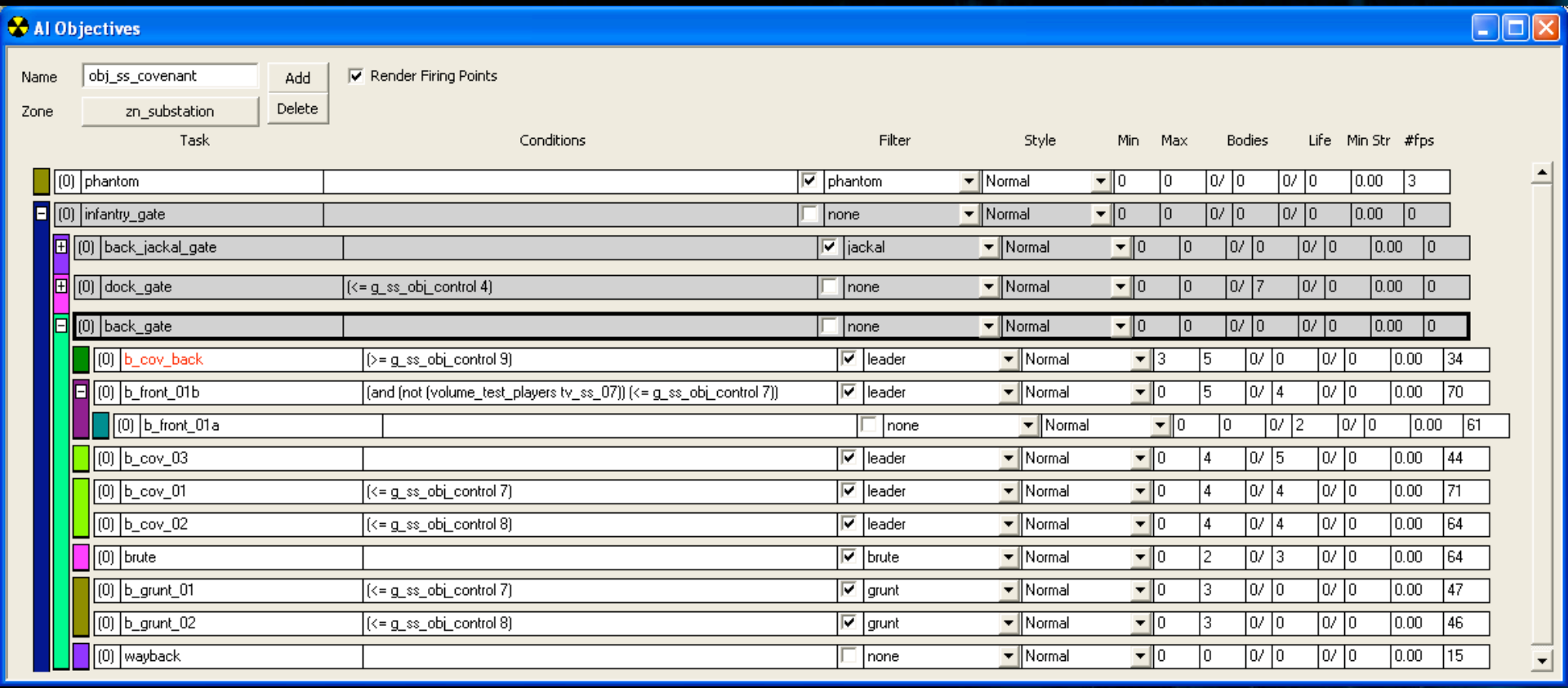
- Tasks turn themselves on and off
- Squads pulled UP, on activation of a higher-priority task
- Squads pushed DOWN, on deactivation of the task they're in

3 Generators Revisited



- root
  - g1_group g1 alive max 10
    - g1 forward >75%
    - g1 fallback >50%
    - g1 laststand
  - g2_group g2 alive max 10
    - g2 forward >75%
    - g2 fallback >50%
    - g2 laststand
  - g3_group
    - g3 forward >75%
    - g3 fallback >50%
    - g3 laststand

Designer UI

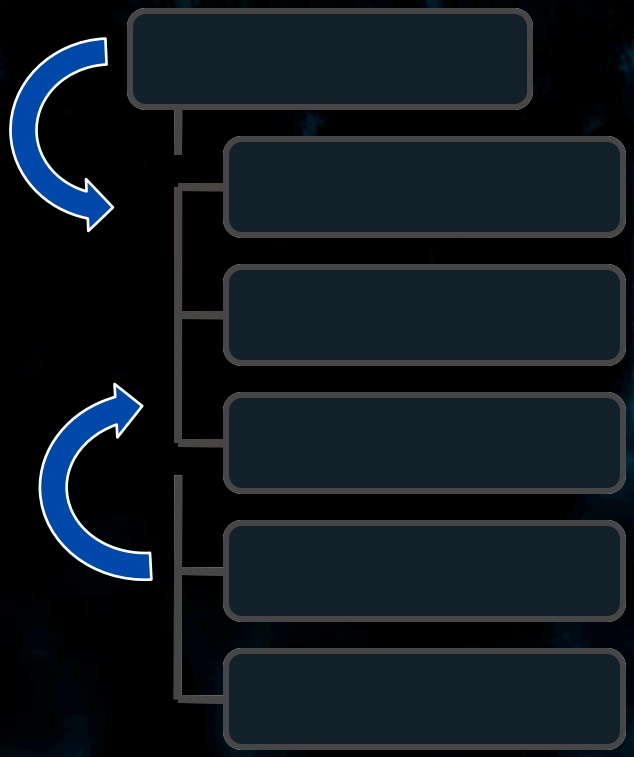


- Integration with HaloScript
- Run-time feedback

The Algorithm

The Algorithm

- Consider a subtree fragment
- Determine which children are active
  - Squads in inactive tasks assigned back up to parent
- Consider top priority group
- Collect squads to attempt to distribute
  - Squads currently in parent
  - Squads in lower-priority tasks
- Distribute Squads
- Recurse for children in top priority-group
- Iterate to next “priority group”



Squad Distribution

Formally, we have

- set *S* of *n* squads
- set *T* of *m* tasks

Now, find a mapping $F(S) \rightarrow T$

Two parts:

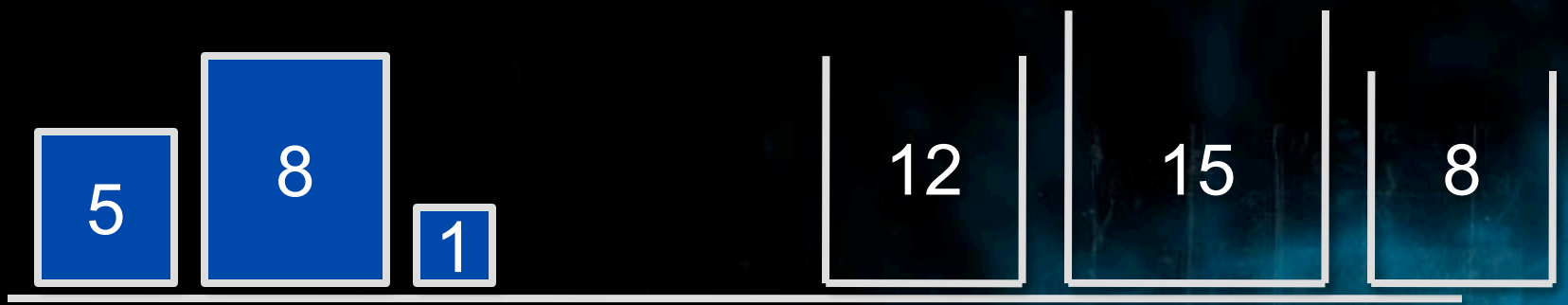
1. Respect Task-Capacity Constraints
2. Minimize cost function *H(F)*

Squad Distribution

1. Respect Task-Capacity Constraints

 # guys assigned to task t ≤ capacity(t)

... but remember, we're bucketing by squads.



This is called ***bin-packing***. And it's NP-Hard.

Squad Distribution

1. Respect Task-Capacity Constraints

Fortunately

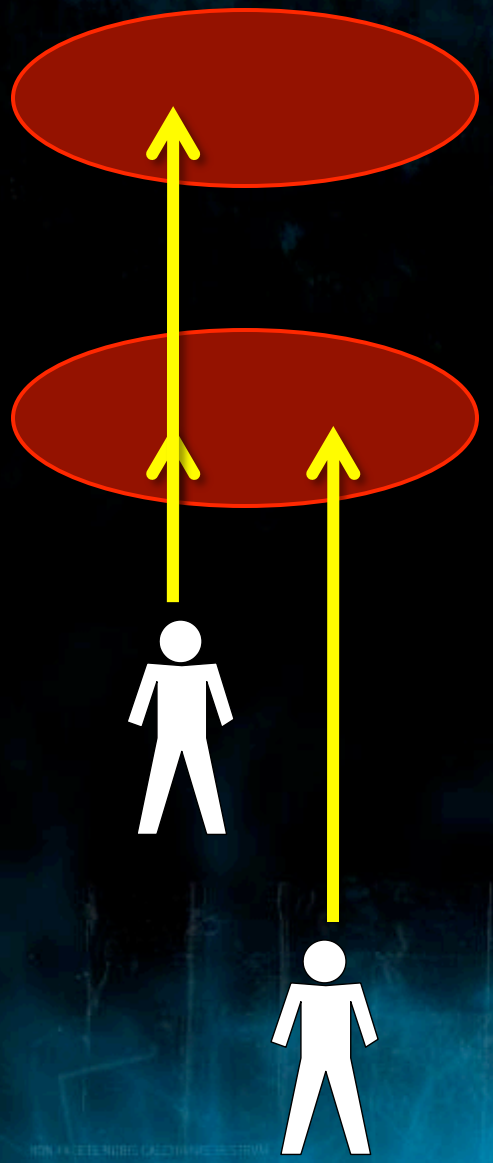
a) there's always Wikipedia

b) we can live with sub-optimal

c) we're optimizing not for *m*, but for ***H(F)***

Squad Distribution

2. Minimize cost function *H(F)*



Why a cost function?

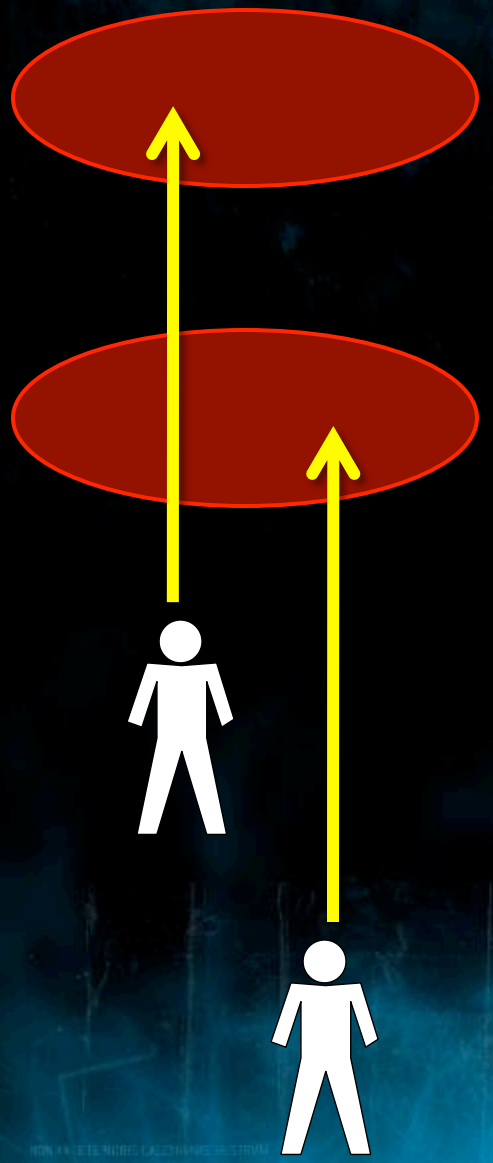
- Gives us a basis for choosing one distribution over another
- Weigh different concerns
  - *don't want* to travel far
  - *want* to act coordinated
  - *want* to balance the tree
  - *want* to get near to the player

Squad Distribution

2. Minimize cost function *H(F)*

DANGER: AI can look really stupid with wrong H(f)

OPPORTUNITY: Designer has abdicated his decision-making authority



Squad Distribution

2. Minimize cost function *H(F)*

A class of cost functions:

We use

A Greedy Approach

```
while (S is not empty)

  find pair (s,t) that give the minimum
  H(s,t) for all S x T (where adding s to t
  would not exceed t's capacity)

  if (s,t)
     assign(s, t)
     capacity(t) = capacity(t) - size(s)
     S = S - s
  else
     end
```

A note on Perf

Our algorithm may be $O(n^2 m)$, but we are redeemed by the fact that n and m are small

Other perf measures

- Cache H(s,t) results
- Timeslice entire trees ← Halo3
- Timeslice nodes within trees

Refinements

Filters

Particular tasks only available to particular *kinds* of guys

E.g.

- Must be of character type X
- Must be in vehicles
- Must NOT be in vehicles
- Snipers

"Filters"

- Specify *occupation* conditions (as opposed to *activation* conditions)
- "Trivially" implemented as an inf return value from H(s, t)
- Helpful for the "spice"

Further Task Refinements

Activation behavior

- Latch on
- Latch off / exhaustion

Exhaustion behavior

- Death count
- Living count

Assignment behavior

- One-time assignment

All of these were designer requests

Case Studies

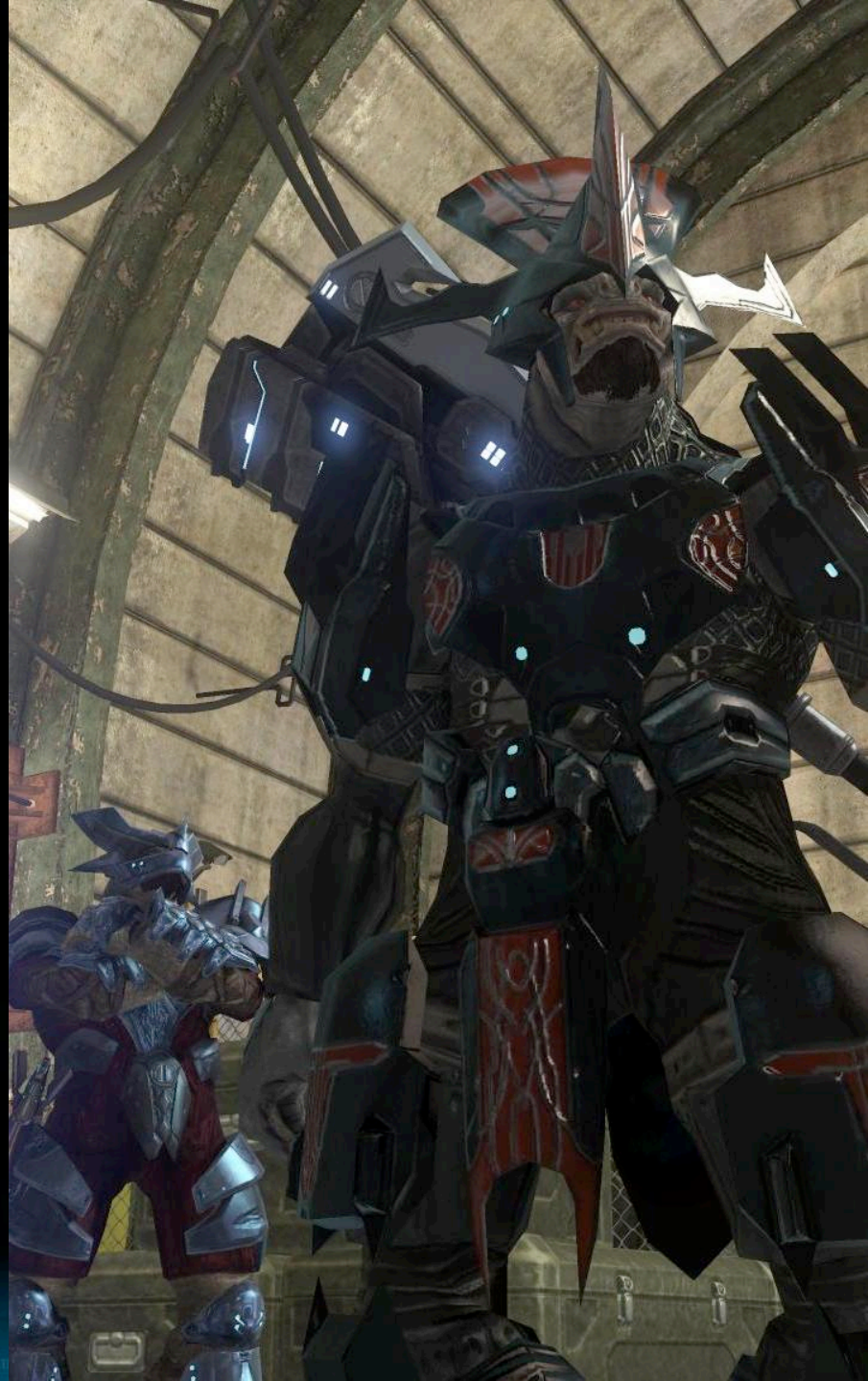
Case Study #1: Leadership

Want to have leaders and followers

- Brute and three grunts
- Brute Chieftan and brute pack

Gameplay

- Leaders provide structure to encounter
- Leader death “breaks” followers



Case Study #1: Leadership

Two Parts:

1. Leadership-based filters
   - Core task: “leader” filter
   - Peripheral tasks: “NO leader” filter

2. Task “broken” state
   - Task does not allow redistribution in or out while broken
   - NPCs have “broken” behaviors

Case Study #2: Player pickup

Vehicle encounters are not fun without a vehicle

Gameplay

- When the player needs a vehicle, allies go pick him up

Case Study #2: Player pickup

Implementation: one dedicated player-pickup task per encounter

Four parts:

1. vehicle filter
2. player_needs_vehicle() script function
3. “follow player” task option
4. driver player_pickup behavior

And that’s it!

Demo
(Max Dyckhoff, everybody)

Summaries

Badness Summary

- Requires designer training
- Sometimes awkward relationship between scripting system and Objectives
- Tying together allied and enemy “fronts” was complicated.
- The squad wasn’t always the best level at which to do the bucketing
  - e.g. give a guy a sniper rifle … shouldn’t he then be allowed to occupy a “sniper” task?

Technique Summary

- Declarative approaches are great
  - less direct control, more manageability
- Hierarchies are great
  - more modular
  - better scalability
- Self-describing tasks makes this whole thing O(n) complexity rather than O(n^2) (conceptually)

Production Summary

- The Goal: provide a powerful tool for designers to control strategy-level decision-making for a large group of characters
- Flexible enough to incorporate plenty of designer-requested features / modifications
- Great for Prototyping
  - became much more complicated as we neared shippable encounter state
- One-stop-shop for encounter construction
- Design of the system driven from the UI outwards

Summary Summary

Not a problem isolated to Halo

As number of NPCs grows, these kinds of techniques will become more and more important

All you need ...

... is $H(s,t)$