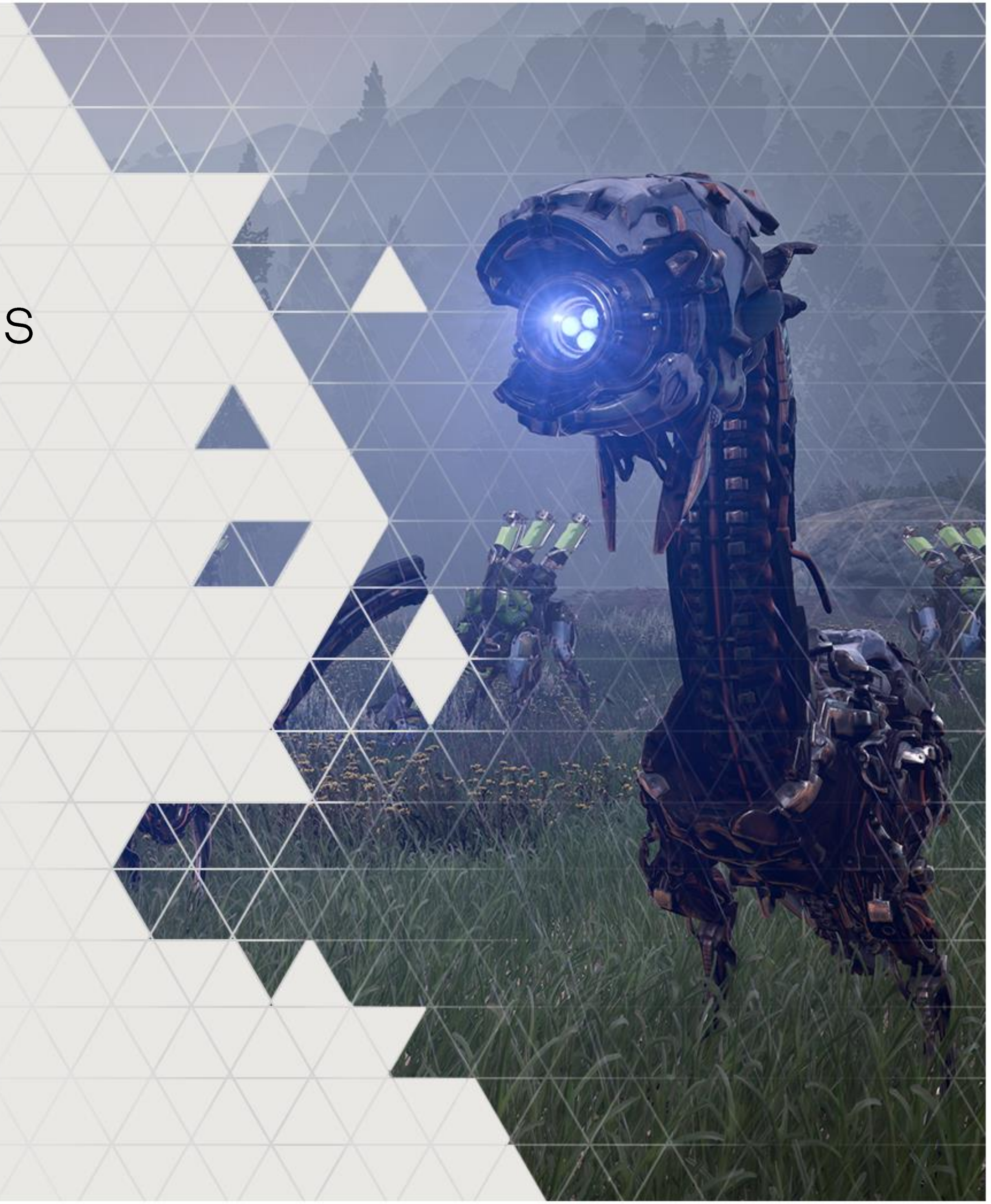
The AI of Horizon Zero Dawn

Horizon
ZERO DAWN
COPYRIGHT © 2017 GUERR[illegible]

Machine Ecology

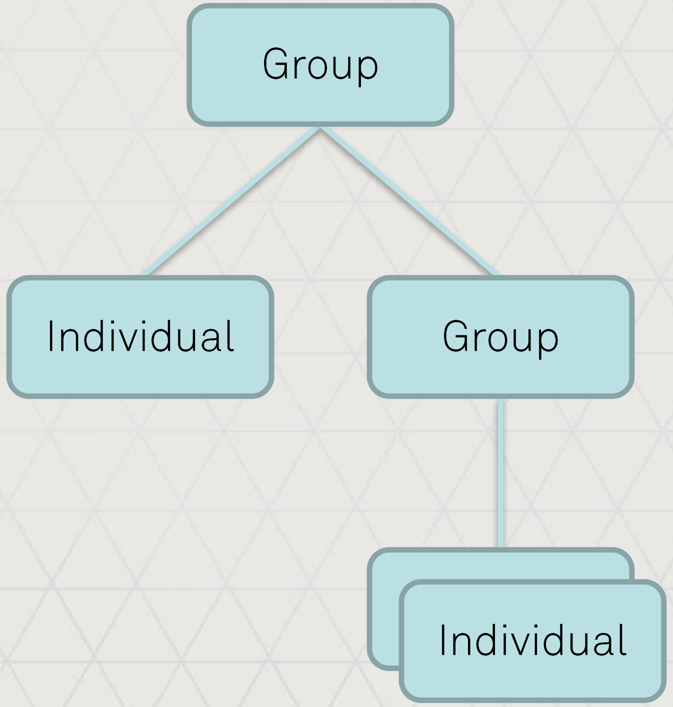
- Machines coexist in mixed groups
- Each class serves a function
  - Acquisition
  - Transport
  - Scavengers
  - Reconnaissance
  - Combat
- Composition affects behavior



HORIZON
ZERO DAWN™

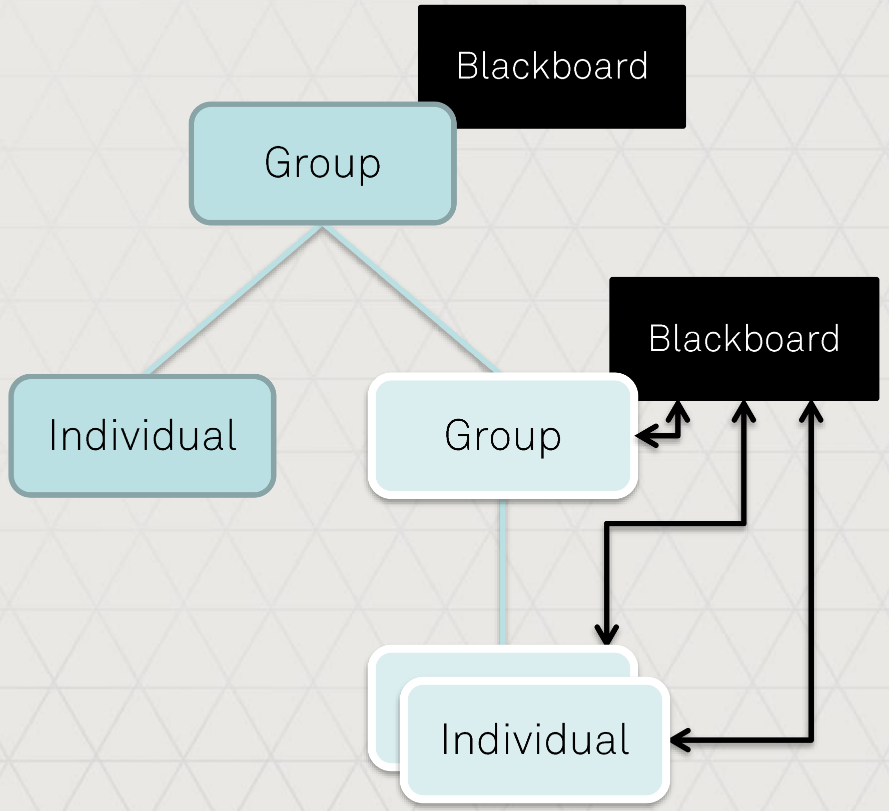
Agent Hierarchy

- Machine herds organized hierarchically
  - Previously: strategic, tactical, individual
  - Now: flexible, dynamic hierarchy
- Individual agent
  - Controls a character
  - Observes world through perception model
- Group agent
  - No physical manifestation
  - Coordinates agents



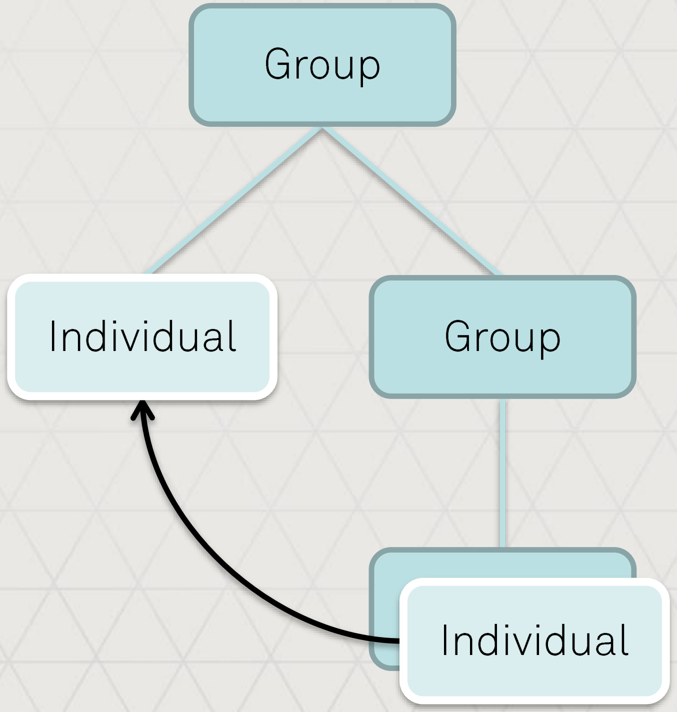
Agent Hierarchy

- Machine herds organized hierarchically
  - Previously: strategic, tactical, individual
  - Now: flexible, dynamic hierarchy
- Individual agent
  - Controls a character
  - Observes world through perception model
- Group agent
  - No physical manifestation
  - Coordinates agents
- Communication
  - Blackboard



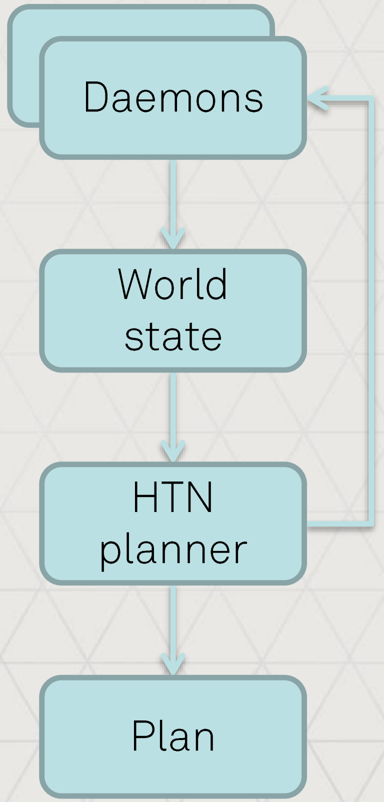
Agent Hierarchy

- Machine herds organized hierarchically
  - Previously: strategic, tactical, individual
  - Now: flexible, dynamic hierarchy
- Individual agent
  - Controls a character
  - Observes world through perception model
- Group agent
  - No physical manifestation
  - Coordinates agents
- Communication
  - Blackboard
  - Direct messaging



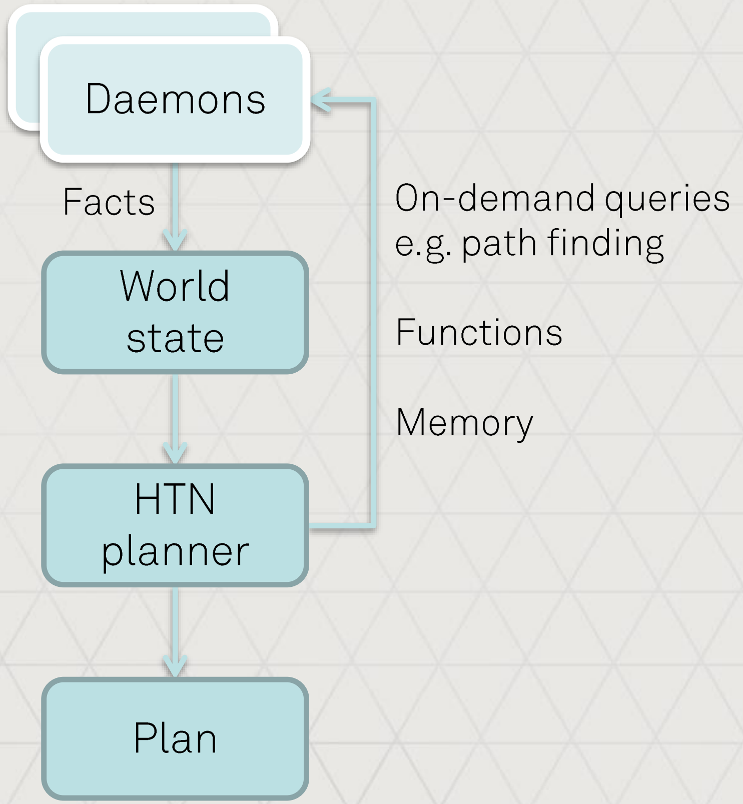
The Agent

- Groups and individuals use same core decision process



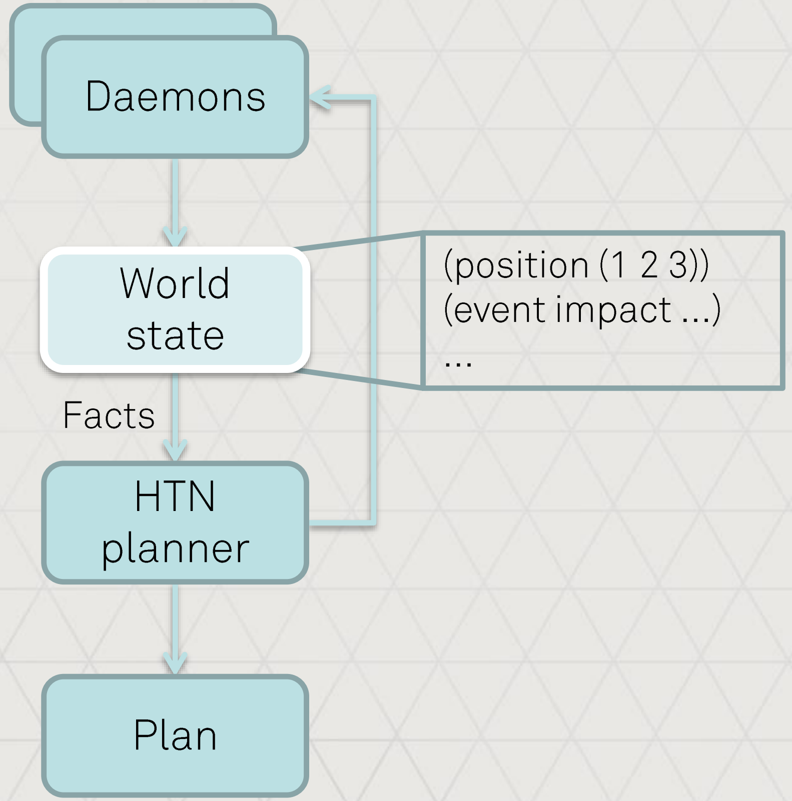
The Agent

- Groups and individuals use same core decision process
- Daemons provide knowledge, e.g.
  - Individual: body state, perception
  - Group: hierarchy
  - Common: messaging, memory
- On-demand queries, such as path finding
  - Parameters only apparent during planning



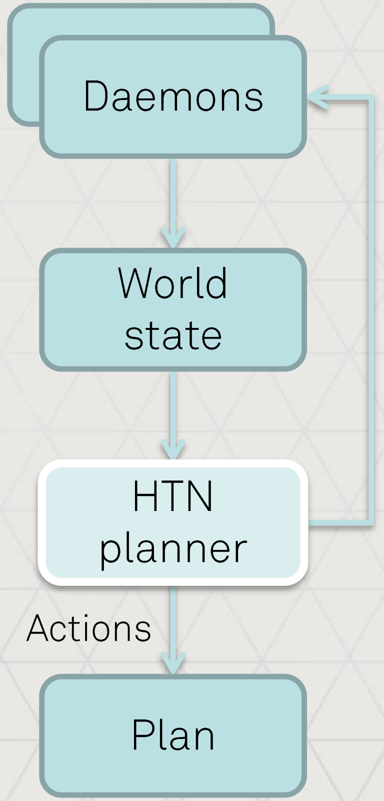
The Agent

- Groups and individuals use same core decision process
- Daemons provide knowledge
- World state consists of facts
  - Flexible data representation
  - Tuple of variable type elements, e.g.
    - Identifier
    - Integer, floating point
    - Object pointer
    - Lists, can be nested



The Agent

- Groups and individuals use same core decision process
- Daemons provide knowledge
- World state consists of facts
- Hierarchical Task Network makes a plan
  - Solves a problem by hierarchical decomposition of an abstract task into concrete actions

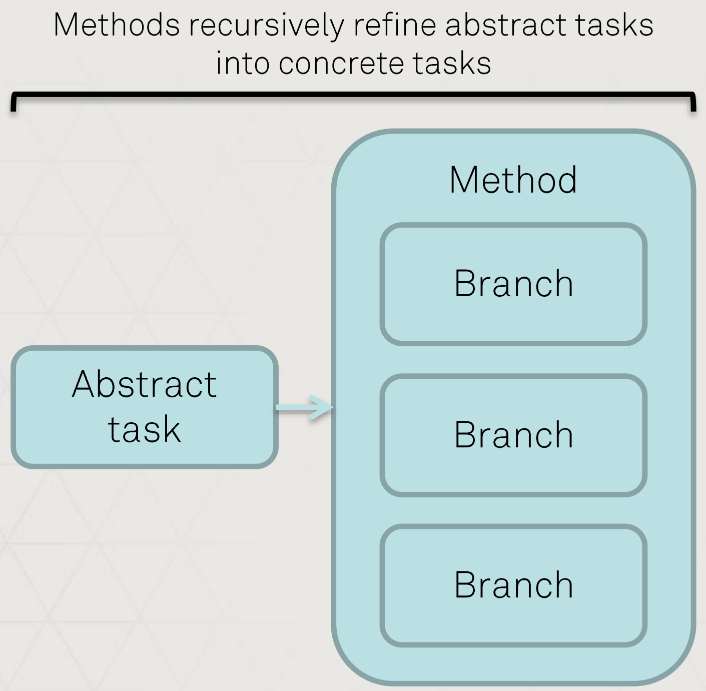


HTN in brief

This is a brief overview of the Hierarchical Task Network planning process. The planner solves a problem by hierarchical decomposition of abstract tasks into concrete actions.

The planner is given an abstract task: something that is not complete and descriptive enough to act upon. As a toy example, consider the task: “eat fruit”, where “fruit” is a parameter of the abstract task “eat”. The planner takes this abstract task and recursively breaks this down into smaller sub-problems until finally you are left with a concrete plan of actions.

What we call the planning domain is a collection of methods for breaking down these tasks. A method consists of a number of possible replacements for a task, called branches. The branches are considered in-order from top to bottom. Our example may have branches for eating something in our possession, getting it from our house or buying it in a store.


Methods recursively refine abstract tasks into concrete tasks
Method
Branch
Branch
Branch
Abstract task

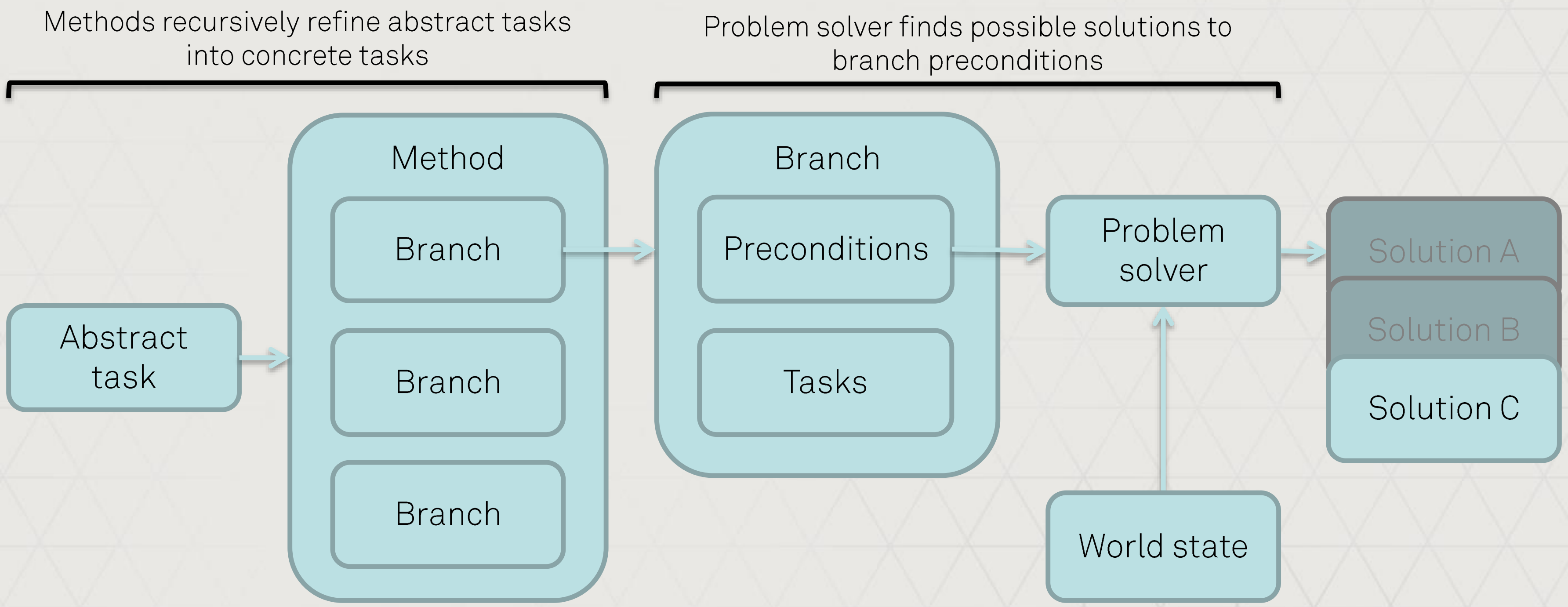
$\underbrace{\text{(eat fruit)}}_{\uparrow}$ a parameter

Method: (eat ?type) ← a variable
Branches:
  In possession
  Get from house
  Buy in store

HTN in brief

Each branch consists of preconditions and a set of replacement tasks. These replacement tasks can be abstract or concrete. The preconditions are stated declaratively. For example our 'in possession' branch preconditions may state: I have an item X, such that X is edible and X is of the given type. This example only uses an and statement, but we allow for nested and / or / not statements.

The preconditions are evaluated by running a problem solver. The solver searches for facts in the world state that match the terms of the preconditions. As it is searching it will assign values to variables. If the solver fails to find a matching fact for a term it will backtrack and try a different substitution for the previous term of the precondition. In our example the solver may first attempt our wallet and find that it is not edible. Next it could consider our sandwich, but that is not a fruit. Finally it may settle on a banana, which is both edible and a fruit.



Methods recursively refine abstract tasks into concrete tasks
Problem solver finds possible solutions to branch preconditions
Abstract task
Method
Branch
Branch
Branch
Branch
Preconditions
Tasks
Problem solver
World state
Solution A
Solution B
Solution C

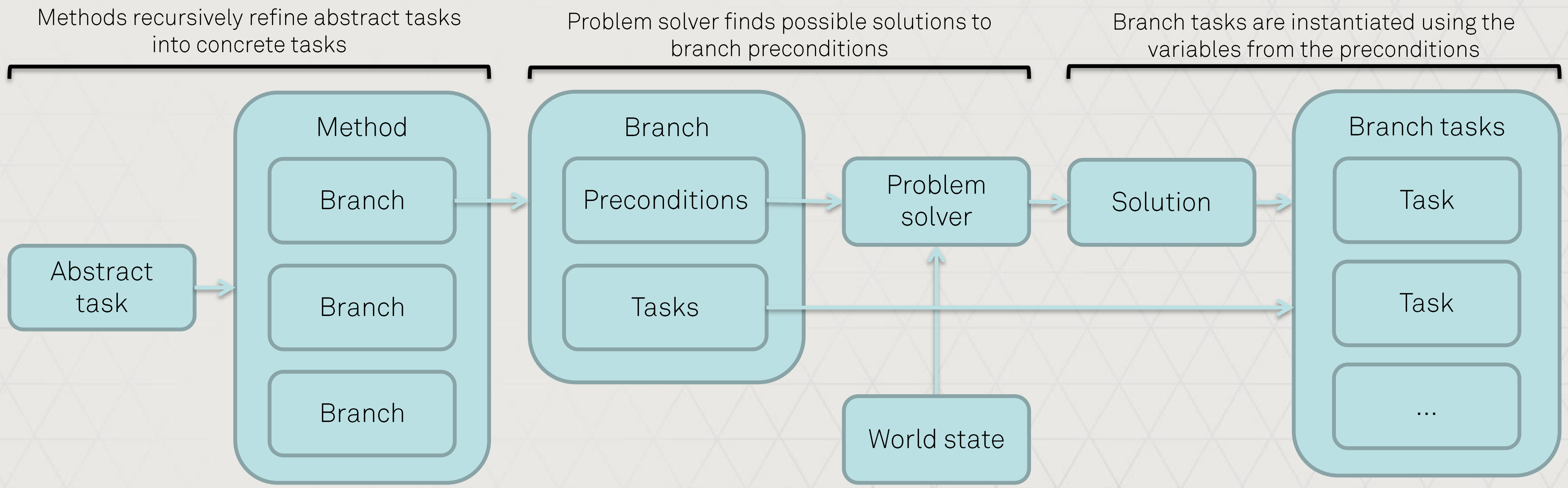
(eat fruit)	Method: (eat ?type)	Branch: in possession	Solution A	Solution B	Solution C
	Branches:	Preconditions:			
	In possession	(item ?x)	(item wallet)	(item sandwich)	(item banana)
	Get from house	(edible ?x)	~~(edible wallet)~~	(edible sandwich)	(edible banana)
	Buy in store	(type ?x ?type)	(type wallet fruit)	~~(type sandwich fruit)~~	(type banana fruit)

HTN in brief

The planner then replaces the original abstract task with the set of tasks of the branch. These tasks can reference the variables of the preconditions. In our example this could be: take the banana, prepare it for eating and finally eat it.

The exclamation mark indicates that this is a concrete task. The preparation task is abstract and requires further refinement. In case of our banana it will need to be peeled, but a sandwich would need to be unwrapped.

In this case we found a solution and the process will continue recursively, but if no valid solution is found the planner will try the different branches. If those also fail and there is no valid replacement for a task, the planner will backtrack the recursion to where that task was added and try a different solution there.


Methods recursively refine abstract tasks into concrete tasks
Problem solver finds possible solutions to branch preconditions
Branch tasks are instantiated using the variables from the preconditions
Abstract task
Method
Branch
Branch
Branch
Branch
Preconditions
Tasks
Problem solver
World state
Solution
Branch tasks
Task
Task
...

(eat fruit)

Method: (eat ?type)
Branches:
- In possession
- Get from house
- Buy in store

Branch: in possession
Preconditions:
- (item ?x)
- (edible ?x)
- (type ?x ?type)

Solution

- (item banana)
- (edible banana)
- (type banana fruit)

concrete task

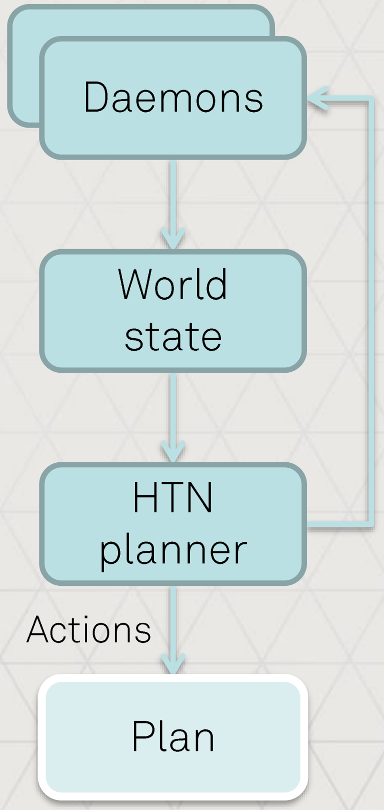
abstract task

Replacement tasks:

- (!take banana)
- (prepare banana)
- (!eat banana)

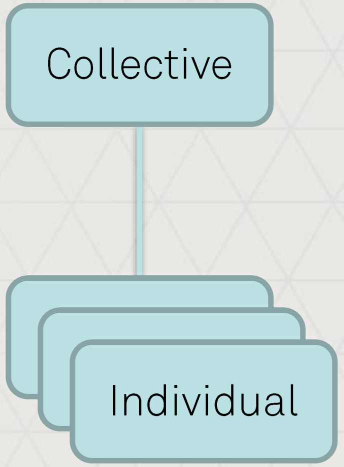
The Agent

- Groups and individuals use same core decision process
- Daemons provide knowledge
- World state consists of facts
- Hierarchical Task Network makes a plan
- Execute sequence of actions
  - Individuals: move, look, speak, etc.
  - Groups: communicate, make subgroup, etc.



The Collective

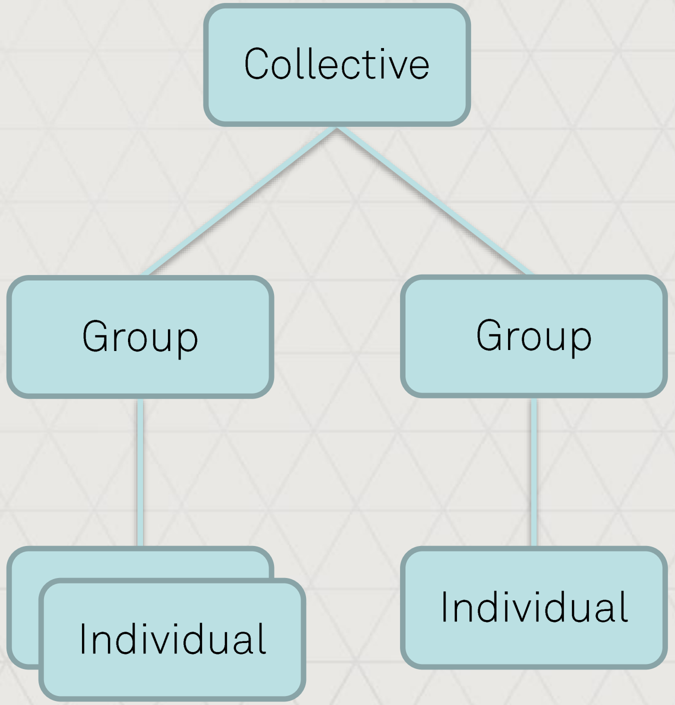
- Collective agent is the root group
- All new individuals added to it
- Creates groups for compatible individuals
- Individuals provide passport
  - Opaque tuple
  - Equality determines compatibility



Passports:
Individual A : (machine ...)
Individual B : (machine ...)
Individual C : (hacked machine ...)

The Collective

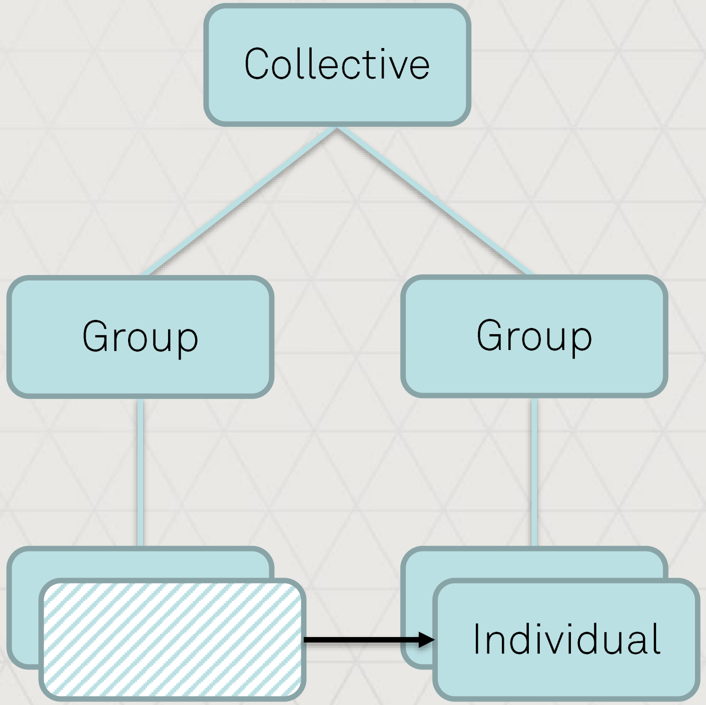
- Collective agent is the root group
- All new individuals added to it
- Creates groups for compatible individuals
- Individuals provide passport
  - Opaque tuple
  - Equality determines compatibility
- Creates main groups
- Assigns individuals



Passports:
Individual A : (machine ...)
Individual B : (machine ...)
Individual C : (hacked machine ...)

The Collective

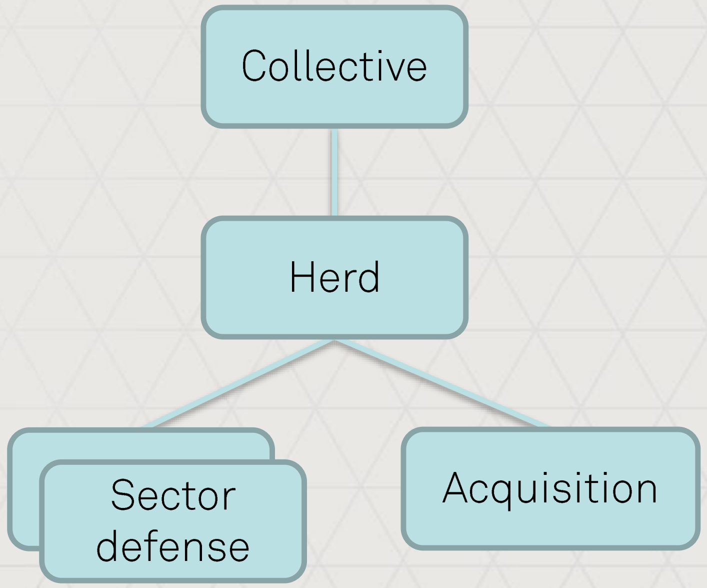
- Collective agent is the root group
- All new individuals added to it
- Creates groups for compatible individuals
- Individuals provide passport
  - Opaque tuple
  - Equality determines compatibility
- Creates main groups
- Assigns individuals
- Reassigns if passport changes



Passports:
Individual A : (machine ...)
Individual B : (hacked machine ...)
Individual C : (hacked machine ...)

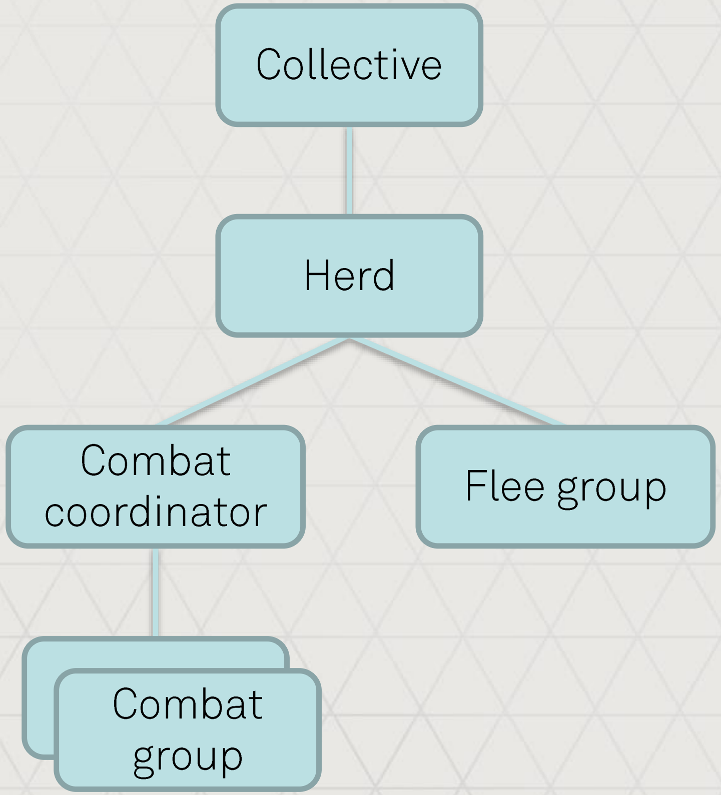
A Relaxed Herd

- Mix of acquisition and combat machines
- Acquisition group will take center
- Defense split in sections around outside
- Patrol paths are generated
  - Avoid passing through stealth vegetation
  - But pass nearby, allow player opportunity to hack or trap
- Systemic solutions help fill a large open world



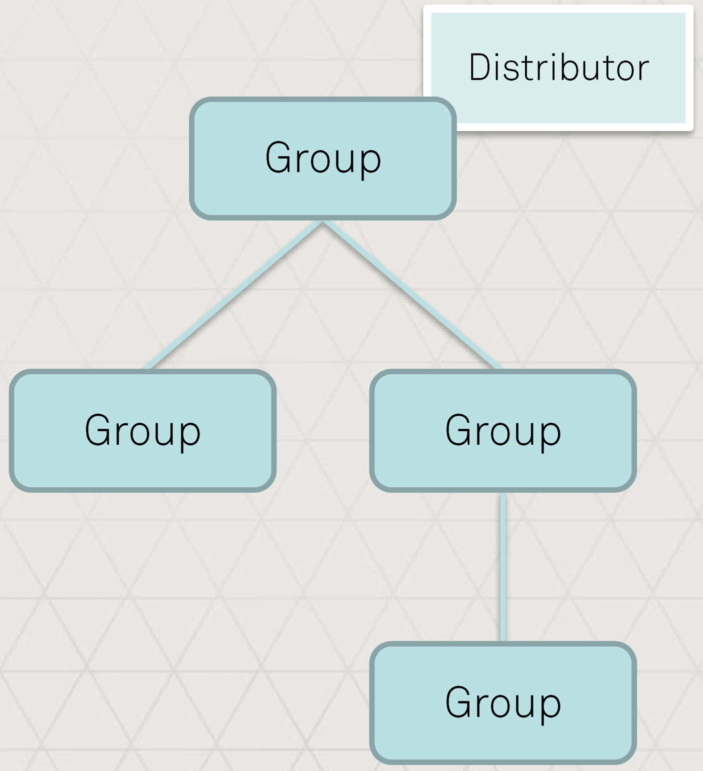
Going To Combat

- Individuals are eyes and ears of group
- In principle limited world knowledge
  - Take out machine before it alerts the group
  - Don't know what you can't see
  - Not strictly enforced if it benefits gameplay
- When alerted create specialized groups
  - Acquisition machines flee
  - Combat group per enemy
- Specialized groups limit complexity
- Multiple roles exist within group



Role Distribution

- Role defined by identifier and context
  - Examples: attacker, scavenger, searcher, etc.
  - Context is a tuple, contents specific to role
- Roles are requested by groups
- Optional distributor component handles assignment
  - Assigns roles for itself and recursively
- Goal: find the right individual for the job
  - Distribute combat machines over enemies
  - Who will scavenge, who will patrol
  - For some roles many factors weigh in



Role Distribution Process

- All groups request roles with priority and limit
- For all roles in order of priority
  - Query all candidate individual's utility for that role
    - Solved by individual's HTN domain
    - Proximity, type of machine, knowledge of target, being attacked by target
  - Assign best candidate

Priority	Limit	Role	Context
→ 0.9	1	attacker	player ...
0.8	1	attacker	enemy_a ...
0.6	inf	combatant	enemy_a ...
0.6	inf	combatant	player ...
...			

Individual	Utility	Reasons
Watcher A	0.5	pretty close, but not in view
Watcher B	-1	do not know this target
→ Ravager	0.7	in view, bigger machine
...		

Role Distribution Process

- All groups request roles with priority and limit
- For all roles in order of priority
  - Query all candidate individual's utility for that role
    - Solved by individual's HTN domain
    - Proximity, type of machine, knowledge of target, being attacked by target
  - Assign best candidate
  - If limit reached: next role

Priority	Limit	Role	Context
~~0.9~~	~~1~~	~~attacker~~	~~player ...~~
0.8	1	attacker	enemy_a ...
0.6	inf	combatant	enemy_a ...
0.6	inf	combatant	player ...
...			

Individual	Utility	Reasons
Watcher A	0.4	quite close
Watcher B	0.6	quite close, angry at target
Watcher C	-1	knocked down
...		

Role Distribution Process

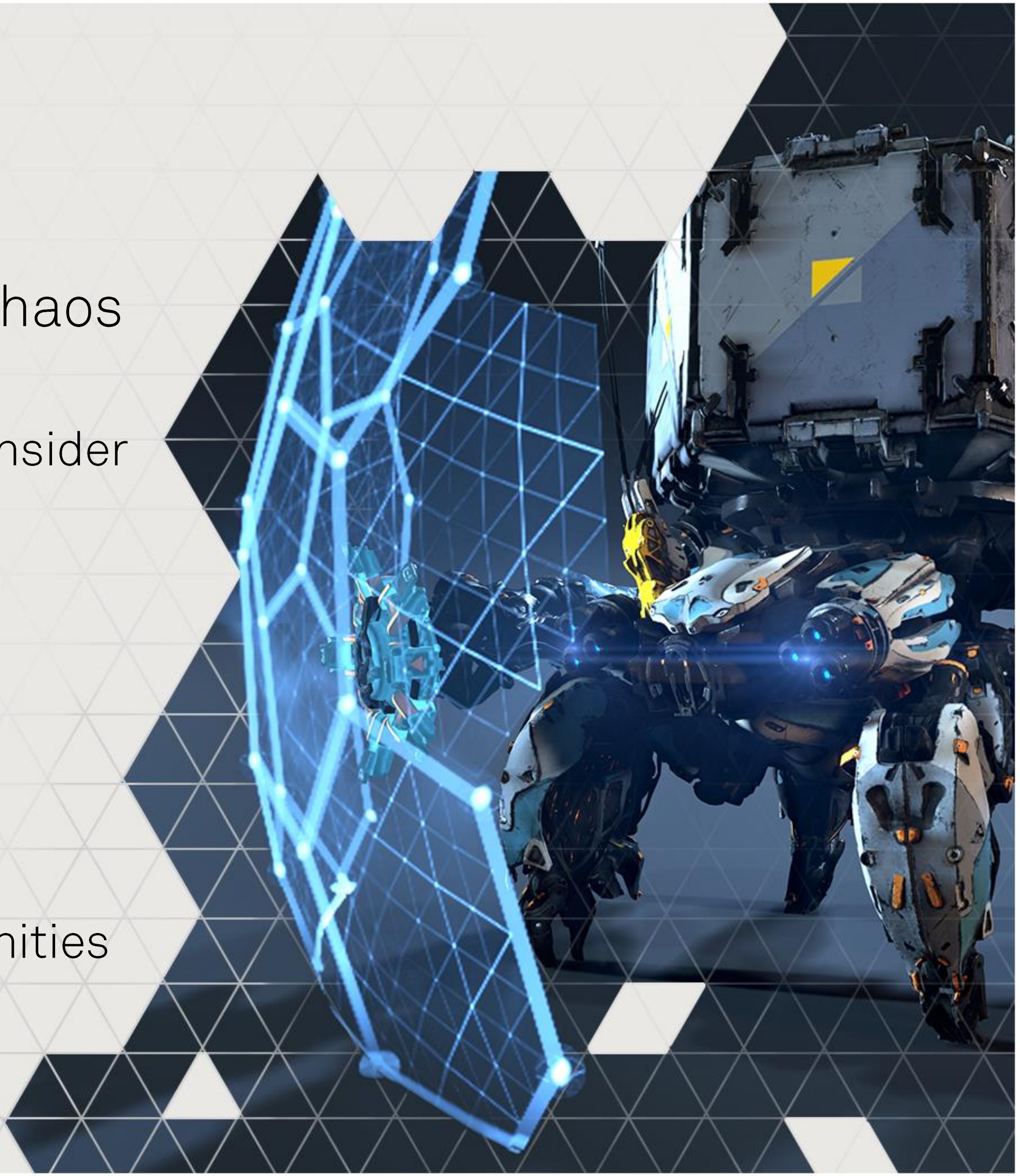
- All groups request roles with priority and limit
- For all roles in order of priority
  - Query all candidate individual's utility for that role
    - Solved by individual's HTN domain
    - Proximity, type of machine, knowledge of target, being attacked by target
  - Assign best candidate
  - If limit reached: next role
  - If not: group updates priority
    - Useful for spreading over multiple groups

Priority	Limit	Role	Context
~~0.9~~	~~1~~	~~attacker~~	~~player ...~~
~~0.8~~	~~1~~	~~attacker~~	~~enemy_a ...~~
~~0.6~~ 0.5	inf	combatant	enemy_a ...
0.6	inf	combatant	player ...
...			

Individual	Utility	Reasons
Watcher A	0.4	quite close
Watcher C	0.7	very close
Grazer	-1	not a combat machine
...		

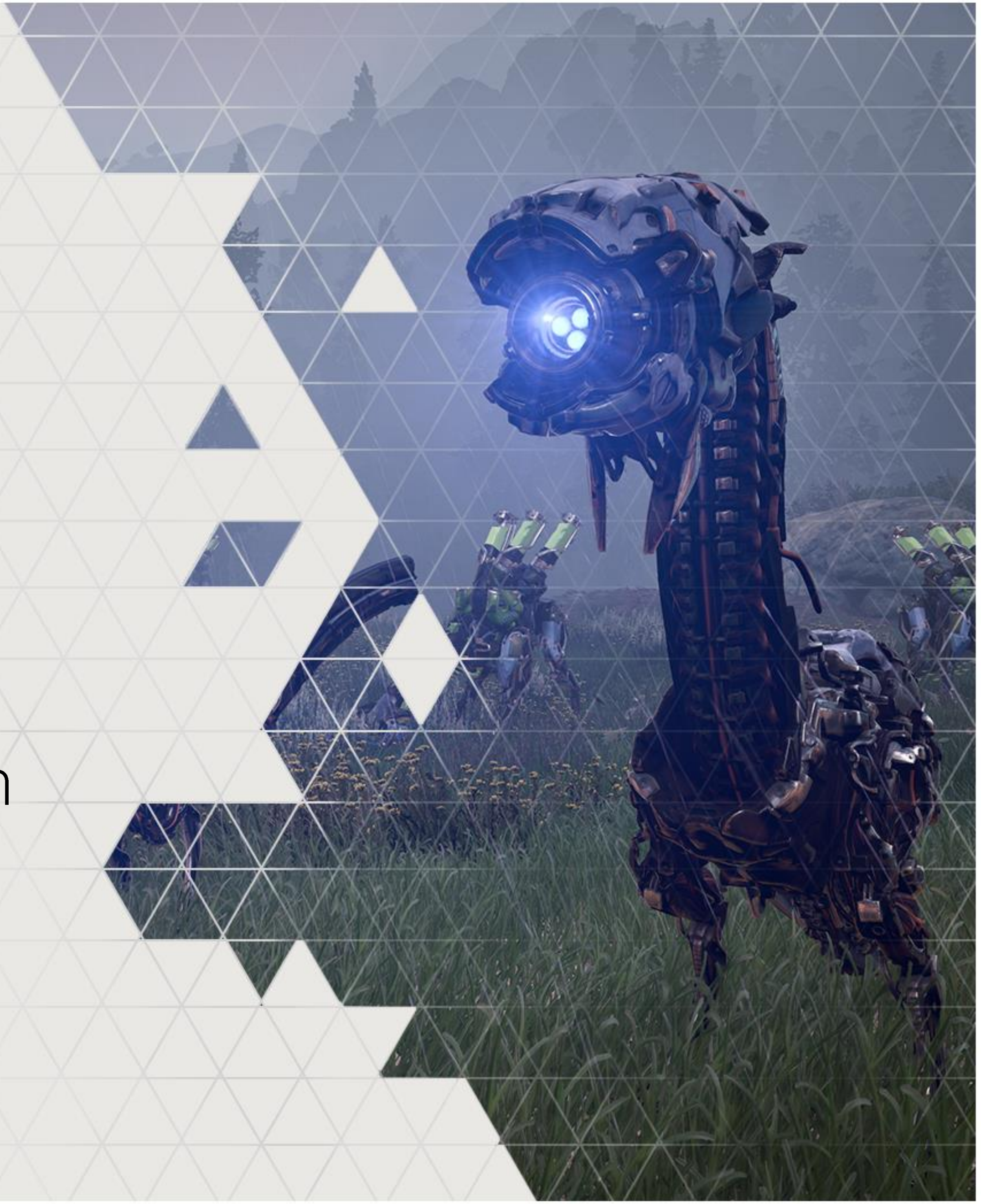
Choreographing Combat

- Role distribution used to minimize chaos
  - Spread machines over targets
  - Choose a fair attacker for the player, consider
    - Machine awareness and state
    - Player awareness
    - Proximity and aggressiveness
    - Damage received and dealt
    - Etc.
- Attack selection also utility based
  - Integrated into HTN preconditions
  - Aims for variety, challenge and opportunities for the player



Perception Model

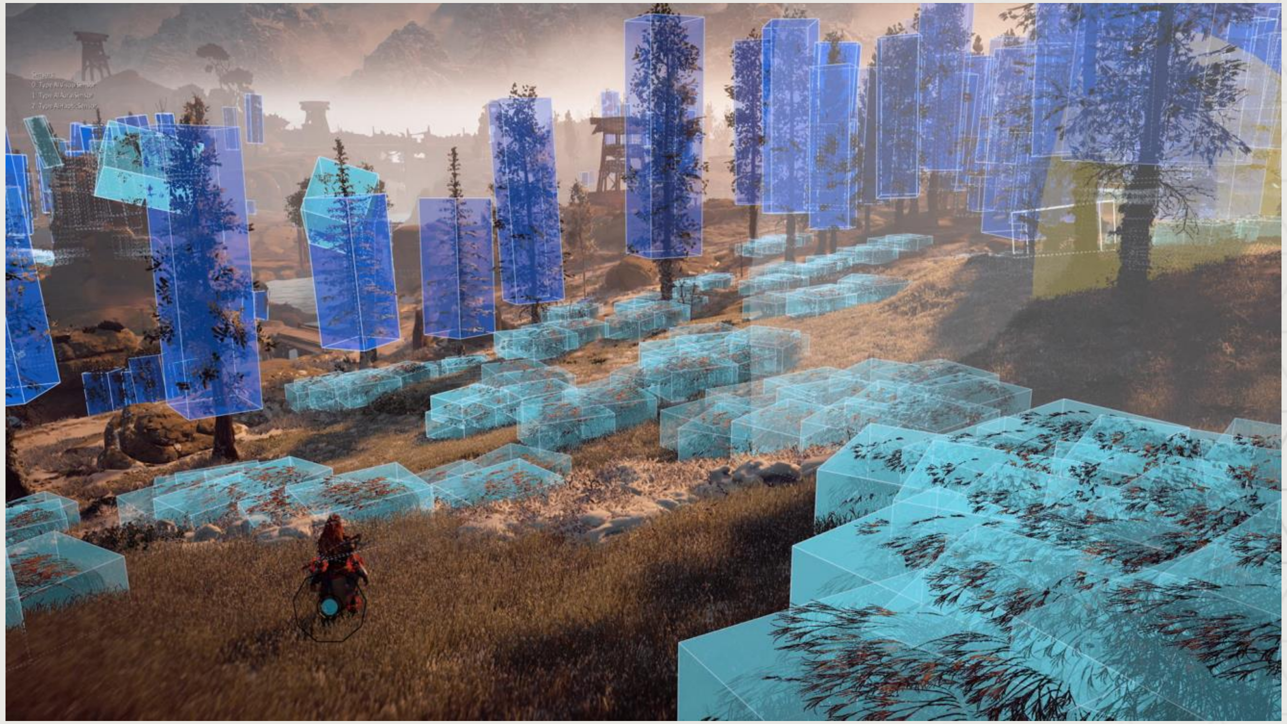
- Individuals have sensors
- Observable objects have stimuli
- Sensor and stimuli types
  - Common: visual, aural, touch
  - Special: smell, radar, proximity
- Observation strength depends on
  - Sensor sensitivity
  - Stimulus size / loudness

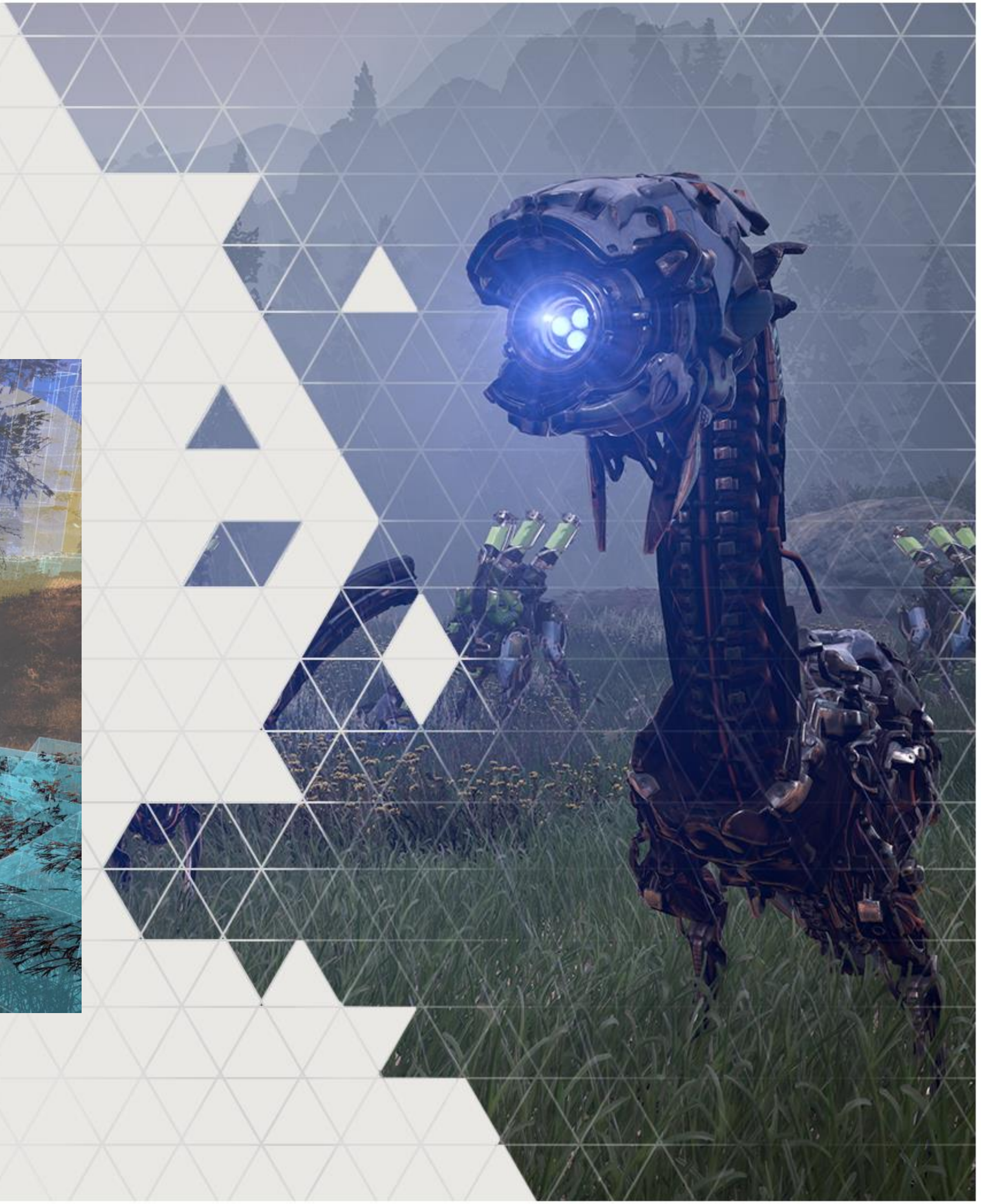


Stimuli

- Stimuli carry information packets, e.g.
  - Enemy description
  - Arrow impact / whizzing by
  - Carcass
- Handler interprets information
  - Configurable per individual
  - Rabbit could infer less than human
  - Low observation strength can reduce information

Visual perception





Navigation

- Previous offline process not desirable for open world workflow
  - Need traversal between all areas
  - More designers and artists working on same content
  - Offline navigation graph would often be out of date
  - Also: cumbersome to deal with dynamic terrain
- Large variety agent sizes and capabilities to consider
  - From human size to big as a house
  - Breaking through trees and rocks
  - Some machines can also swim
  - Some machines can also fly
- Solution: runtime generation of multiple navigation meshes

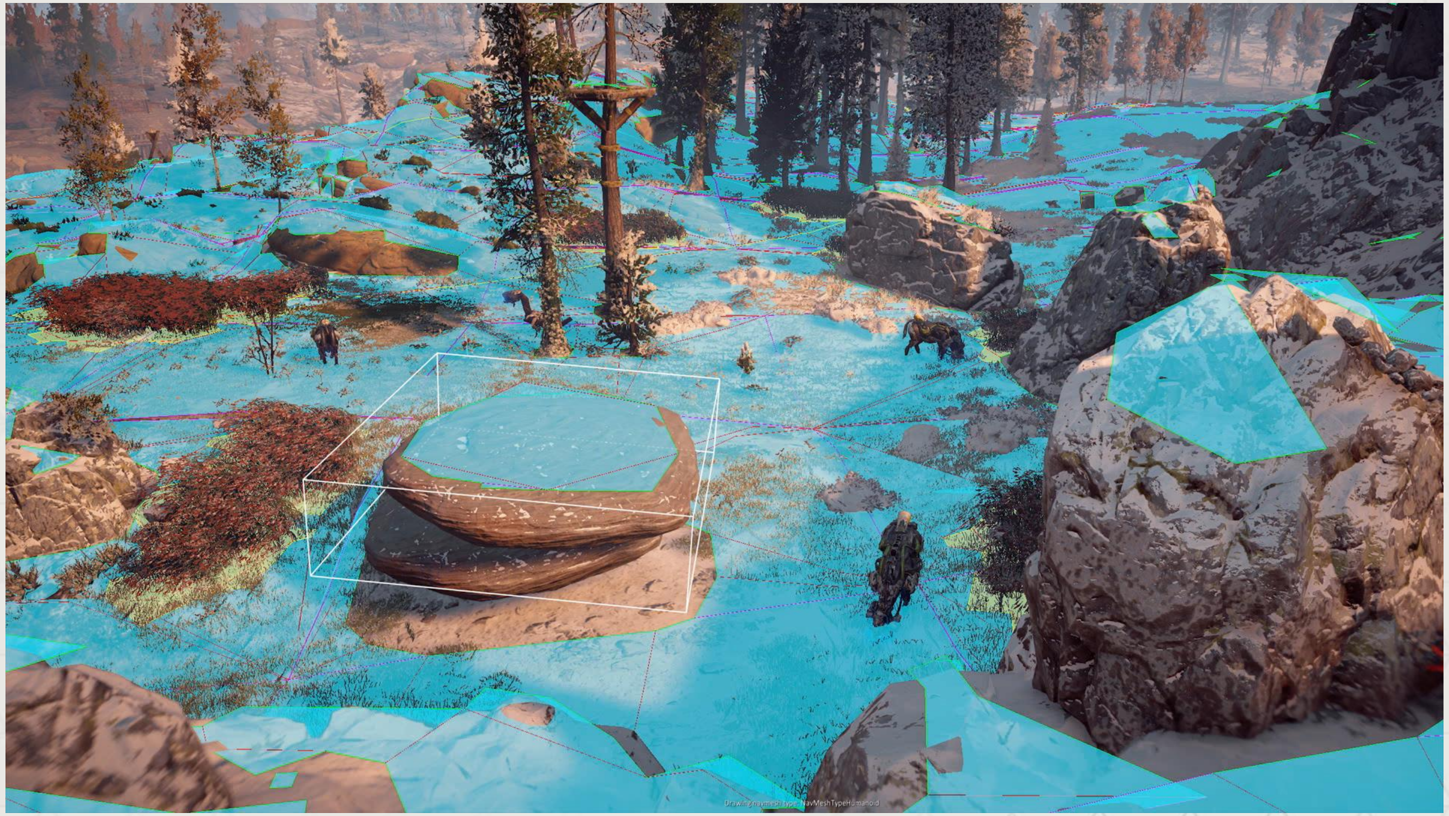
Navigation

- Runtime generation of navigation mesh
  - Fast, iterative designer workflow: never out of date
  - Only depends on collision geometry
  - Available anywhere, generated in activity bubble
  - Dynamic environments require no additional effort
- Updates when collision changes
- Six different meshes
  - Small / Medium / Large / Extra large machine
  - Special cases: player mount and swimming machine

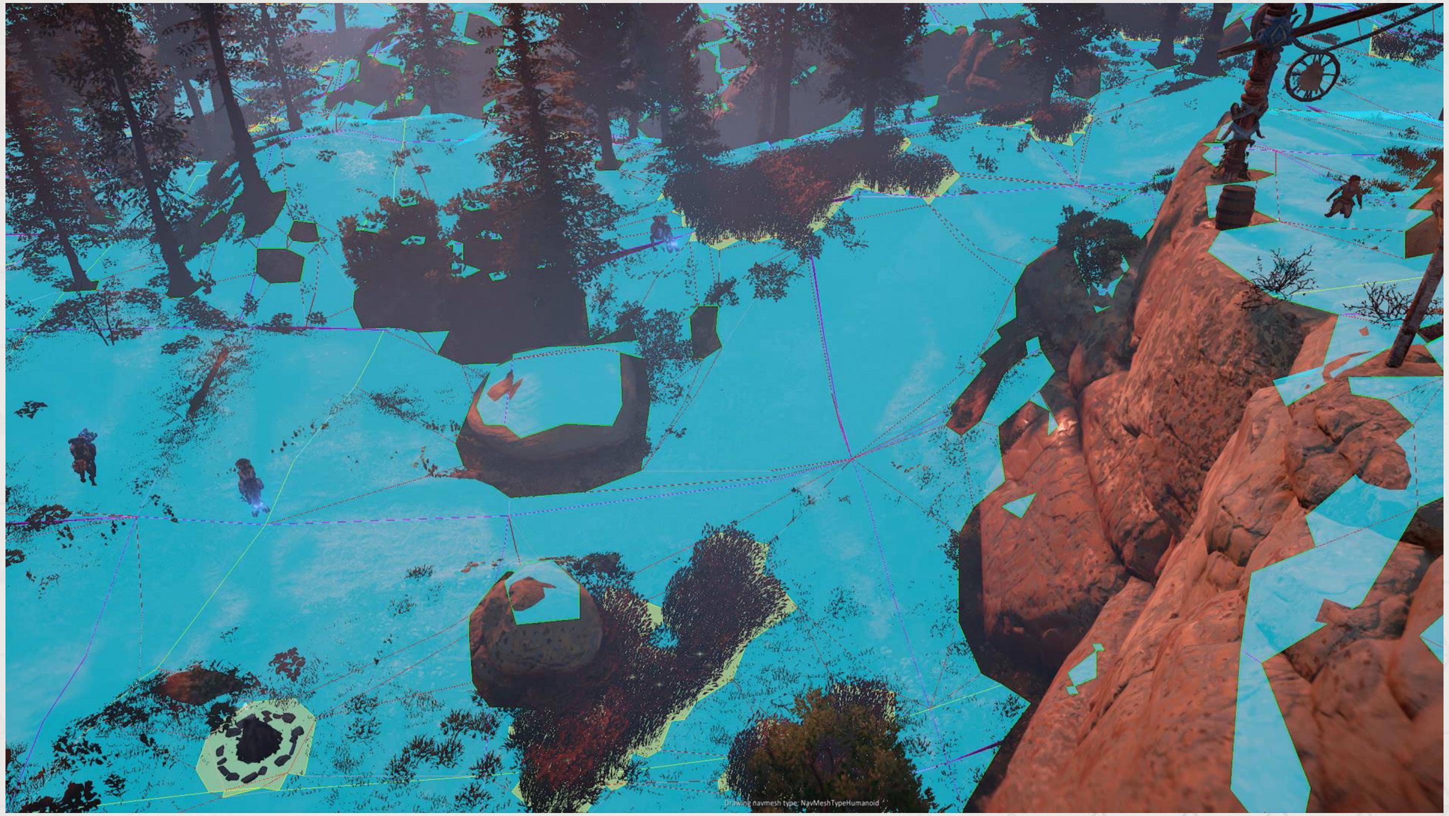
Areas and Obstacles

- Hulls represented on navigation mesh
- Relevance depends on query cost function
- Obstacles
  - Hard or soft blocking : impassable or undesirable
  - Depends on machine type and state
    - Don't break through trees unless large and angry enough
- Restrictions
  - Confine to activity bubble or designed hull
- Stealth vegetation
  - Avoid unless in combat
- Danger areas
  - Avoid path of machine attacks

Navigation


Drawing navmesh type: NavMeshTypeHumanoid

Navigation



The team:

Jose Antonio Escribano Ayllon
Julian Berteling
Ivaylo Ivanov
Wouter Josemans
Hylke Kleve
Robert Morcus
Marco Pisanu
Carles Ros Martínez
Pieter van de Kerkhof
Tim Verweij

Contact: arjen.beij@guerrilla-games.com

HORIZON ZERO DAWN™