The Overseer-Overlord Paradigm: Distributed Swarm Intelligence and Hierarchical Process Management in Screeps AI Systems

The architectural evolution of Screeps artificial intelligence has moved decisively away from decentralized, role-based finite state machines toward centralized, hierarchical governance models. At the forefront of this shift is the "Overmind" framework, which implements a sophisticated Overseer-Overlord paradigm.[1] This model abstracts colonial operations into a structure analogous to an operating system's process management, where specialized controllers (Overlords) orchestrate unit actions under the supervision of a colony-wide executive (the Overseer).[1] By decoupling high-level strategic objectives from low-level unit execution, the paradigm enables a level of multi-room coordination and computational efficiency that is unattainable through traditional state-based approaches.[4]

Structural Hierarchy and the Evolution of Governance

The Overmind architecture is organized as a nested hierarchy, moving from global concerns down to specific environmental interactions. At the apex is the Overmind singleton, a global object that serves as the repository for all colonies, networks, and game-wide intelligence.[1] Below this apex, the system is subdivided into three primary structural layers: the Colony, the HiveCluster, and the Directive.[1]

Functional Layering within the Colony

The Colony acts as the fundamental unit of organizational sovereignty, grouping multiple rooms—the core room and its associated remote outposts—into a single managed entity.[1] Within each Colony, structural logic is bifurcated into HiveClusters and Directives. HiveClusters represent the "organs" of the colony, grouping together stationary structures with related functionality, such as a Hatchery or a Command Center.[1] Conversely, Directives function as conditional attachment points for logic, typically wrapping around flags to trigger specific behaviors in response to environmental stimuli.[1]

The Overseer-Overlord paradigm is the mechanism through which these structural components are animated. While HiveClusters and Directives provide the "where" and "what" of a colonial objective, Overlords provide the "how" by orchestrating the creeps—internally referred to as Zerg—necessary to fulfill those objectives.[1]

Component	Hierarchy Level	Primary Responsibility	Analogous System
Overmind	Global	Global state management, inter-colony coordination, and singleton maintenance.	Kernel / Hypervisor
Colony	Regional	Sovereignty over a set of rooms, resource pooling, and high-level goal setting.	Virtual Machine / OS Instance
Overseer	Colonial	Monitoring for anomalous conditions, placing directives, and managing the process queue.	Process Scheduler / Executive
Overlord	Functional	Task implementation, creep population management, and role-based unit micro-management .	Thread / Application Process
HiveCluster	Structural	Management of stationary assets and structural logic (Hatchery, Links, Labs).	Hardware Drivers / Peripherals
Directive	Conditional	Event-based logic triggers and geographic	Interrupt / Event Listener

		markers for dynamic tasking.	

The Overseer: The Executive Engine of Colonial Adaptation

In the Overmind paradigm, the Overseer serves as the localized executive that governs the instantiation and execution of all colonial processes.[1] Its primary function is twofold: environmental monitoring (stimulus detection) and process prioritization (management of Overlords).[1]

Stimulus Response and Directive Placement

The Overseer continuously examines the state of the rooms under its jurisdiction to detect anomalous conditions. These conditions are categorized as external threats (invasions), internal failures (colony crashes), or strategic opportunities (new expansion sites).[1] Upon detecting such a condition, the Overseer places a Directive—a flag-based object that persists in the game world and serves as a hook for specialized logic.[1]

Directives are classified by a standardized color-coding system, allowing the Overseer to quickly filter and respond to different categories of events.[1]

Directive Category	Primary Color	Secondary Color Example	Logic Trigger
Offensive Combat	Red	Red/Red (Attack), Red/Purple (Dismantle)	Instantiates high-priority CombatOverlords for sieging.
Defensive Combat	Blue	Blue/Blue (Guard), Blue/Purple (Invasion Defense)	Triggers reactive defense protocols against player or NPC threats.
Logistics/Transport	Yellow	Yellow/Yellow (Haul), Yellow/Blue (Link)	Estantiates transport and distribution processes.

Terminal State	Brown	Brown/Red (Abandon), Brown/Yellow (Rebuild)	Manages terminal resource evacuation or emergency rebuilding.

When a Directive is placed, the Overseer immediately instantiates the associated Overlord, bypassing the need to wait for the next full architectural rebuild.[6] This allows the colony to respond to threats, such as a nuke or a sudden breach, with near-instantaneous latency.[6]

The Process Priority Queue

Every Overlord, upon instantiation, registers itself with the Overseer's priority queue.[1] This priority system is the cornerstone of the paradigm's efficiency, ensuring that limited resources—energy and spawn time—are allocated to the most critical tasks first.[1] The Overseer runs all registered Overlords in descending order of priority, ensuring that a colony defense or a hatchery refilling process is executed before non-critical tasks like controller upgrading or room scouting.[1]

The priority calculation is often dynamic, influenced by the current state of the colony. For instance, the priority of a WorkerOverlord might increase significantly if the colony's ramparts are below a critical threshold, while the priority of an UpgradingOverlord might drop to zero if the storage energy levels fall below a specific safety margin.[6]

The Overlord: Process-Based Abstraction of Creep Logic

If the Overseer is the scheduler, the Overlord is the process. An Overlord is a generalization of a set of related tasks within a colony, such as mining a specific source, maintaining a bunker, or conducting a long-range raid.[3] It replaces the traditional model of individual creeps "knowing" their roles with a model where a centralized controller "directs" a pool of assigned units.[1]

Population Management and the Wishlist System

Overlords are responsible for maintaining their own workforce. Instead of individual units checking if replacements are needed, the Overlord maintains a "wishlist" of required creeps based on its current objectives.[3] This wishlist is communicated to the HiveCluster (specifically the Hatchery) through the requestCreep() and wishlist() methods.[3]

The request logic utilizes CreepSetup objects, which define the ideal body composition for a given role. This allows Overlords to adapt the size and capabilities of their units to the colony's

current energy capacity and GCL (Global Control Level).[3] To ensure 100% uptime, Overlords utilize a lifetimeFilter() and prespawn logic:

$$T_{prespawn} = T_{spawn} + T_{travel}$$

where T_{spawn} is the number of ticks required to spawn the creep (3 ticks per body part) and T_{travel} is the estimated path length to the task site.[3] By requesting a replacement $T_{prespawn}$ ticks before the current creep expires, the Overlord ensures seamless transitions in critical roles like mining and guarding.[6]

Unit Orchestration and Task Delegation

Once creeps are assigned to an Overlord, they are wrapped in a specialized class called "Zerg".[1] The Overlord manages these Zerg through a centralized logic loop, typically implemented in the run() phase. Instead of each creep running its own if/else state machine, the Overlord scans through its assigned Zerg and assigns tasks to those that are idle.[1]

This centralized delegation allows for complex coordination that is impossible in role-based systems. For example, a CombatOverlord can ensure that a healer and a tank move in unison, or a TransportOverlord can implement a preference-based matching system to optimize the flow of energy from multiple sources to multiple sinks.[1]

Overlord Implementation	Target Object	Primary Task
QueenOverlord	Hatchery	Refilling spawns and extensions; essential for colony survival.
MiningOverlord	Source	Constant energy extraction from a source.
HaulingOverlord	LogisticsNetwork	Moving energy from containers/links to storage or spawns.
UpgradingOverlord	Controller	Consuming surplus energy to increase GCL.
WorkerOverlord	ConstructionSites	Building and repairing structures within the

		colony.
CombatOverlord	Directive	Conducting offensive or defensive military operations.

Lifecycles and Phases: Optimizing for Performance and Persistence

The Overseer-Overlord paradigm relies on a strictly defined lifecycle that balances the need for complex object-oriented logic with the constraints of the Screeps CPU environment. The Overmind architecture executes this lifecycle in three major phases each tick: build(), init(), and run().[1]

The Build vs. Refresh Duality

A significant innovation in the paradigm is the distinction between the full instantiation phase and the "soft update" phase. Because re-instantiating thousands of class instances every tick is prohibitively expensive for the V8 engine's garbage collector, the Overmind uses a persistent architecture.[5]

1. **The build() Phase:** This phase recursively instantiates all classes (Colonies, HiveClusters, Overlords, Directives). It is only run every N ticks (typically $N = 20$) to minimize the overhead of constructor calls and object creation.[5]
2. **The refresh() Phase:** On all other ticks, the system runs a refresh() phase. This phase keeps the existing script class instances alive and simply updates their internal references to game objects (like RoomObjects or Structures) that have changed between ticks.[5]

This duality reduces the bot's overall CPU cost by over 40% by drastically reducing the frequency and complexity of garbage collection cycles.[5] Persistence is maintained on the global heap, allowing the bot to keep complex internal states—such as tracking enemy movements or logistics flow—without the need for expensive Memory serialization.[5]

Execution Flow within the Tick

Following the structural update (either build or refresh), the system proceeds to the logic phases:

- **init() Phase:** Overlords and HiveClusters register their requirements for the current tick. This includes creep spawning requests, logistics network requests (inputs/outputs), and military squadron assembly.[1]

- **run() Phase:** The execution of state-changing actions. Overlords command their Zerg, the Overseer monitors for new stimuli and adjusts directives, and inter-colony transfers are initiated.[1]

This phased approach prevents common race conditions, such as a creep attempting to withdraw energy from a container before a hauler has deposited it, by ensuring all requests are registered before any actions are executed.[1]

Combat Mechanics: The CombatOverlord and Swarm Logic

The Overmind bot's military prowess is rooted in the specialized CombatOverlord class, which extends the base Overlord to support tactical maneuvers and squad-based coordination.[3] Unlike economic Overlords, which focus on persistent throughput, CombatOverlords are typically conditional, instantiated by the Overseer in response to specific directives like DirectiveGuard or DirectiveInvasion.[1]

Swarm Formations and Synchronization

A key refinement in the paradigm is the SwarmOverlord, which manages groups of Zerg as a single unit. This is critical for overcoming the inherent defensive advantages in Screeps, where rampart repair and tower bursts make uncoordinated attacks ineffective.[6]

SwarmOverlords implement a swarmWishlist() method that ensures all members of a squad are spawned synchronously. This prevents "trickling" where attackers enter a room one by one and are easily picked off by towers.[6] Once assembled at a staging point, the swarm moves in formation, utilizing methods like Swarm.pivot() to rotate the entire group in place without breaking the formation's spatial integrity.[6]

Combat Mechanism	Implementation	Tactical Advantage
Pre-healing	CombatZerg wrapper	Healers predict damage and heal the target creep in the same tick the damage occurs.
Ranged Kiting	CombatOverlord logic	Ranged attackers maintain a distance of 3 to maximize damage while staying out of melee range.

Tower Draining	CombatOverlord logic	Creeps with high TOUGH parts enter and exit room boundaries to deplete tower energy.
Coordinated Breach	SwarmOverlord	Multiple ATTACK or RANGED_ATTACK creeps target the same rampart tile simultaneously to out-damage repairers.

Tactical Adaptation to Defense

The Overseer-Overlord paradigm allows for high-level tactical changes based on the defender's response. For instance, if an Overseer detects that a room is heavily fortified with high-level ramparts, it may switch from a SimpleAttackOverlord to a DismantleOverlord or a SiegeOverlord that utilizes boosted creeps.[6] The cost-effectiveness of rampart repair (100 hits per energy) means that offensive logic must be precisely timed and highly coordinated to be successful.[11]

Logistics and Resource Management: The LogisticsNetwork

Resource distribution within the Overseer-Overlord paradigm is handled by a specialized LogisticsNetwork, which decouples the *supply* of energy (Sources, Containers) from the *demand* (Spawns, Towers, Labs).[1] This system is managed primarily by TransportOverlords and HaulingOverlords.[3]

Request-Driven Resource Flow

The LogisticsNetwork operates on a request-based system. Any HiveCluster or Overlord requiring resources (e.g., a Tower needing energy) generates a requestInput, while any source of resources (e.g., a MiningSite) generates a requestOutput.[6] These requests are prioritized by the Overseer and assigned to transporters.

To optimize CPU and avoid redundancy, the network uses a predicted amount calculation:

$$A_{predicted} = A_{current} + \sum A_{assigned}$$

where $A_{current}$ is the amount of resource currently in the target and $\sum A_{assigned}$ is the

sum of all resources currently being carried by transporters already assigned to that target.[6] This prevents over-supplying a structure and wasting hauler capacity. At higher RCLs, the system further optimizes by using stationary managers (managed by QueenOverlord) at the bunker's central anchor point to handle local distribution, minimizing movement costs.[6]

Inter-Colony and Market Integration

Beyond the localized colony, the Overmind uses a TerminalNetwork to equalize resources across the entire empire. This network monitors the levels of base minerals and energy in every colony terminal and automatically schedules transfers to balance supply.[6] This global coordination is handled by the Strategist and ExpansionPlanner modules, which look at the total state of the bot's assets to make high-level economic decisions.[6]

Global Strategy: The Strategist and ExpansionPlanner

While the Overseer handles the tactical needs of a colony, the Strategist module manages the macro-level expansion of the swarm. It is responsible for choosing the next room to colonize based on numerical scores generated by the ExpansionPlanner.[6]

Room Scoring and Expansion Criteria

The ExpansionPlanner evaluates potential rooms based on a variety of metrics, including source proximity, terrain density (for bunker placement), and distance from existing colonies.[6]

Expansion Metric	Weight	Rationale
Source Proximity	High	Minimizes hauling distance and increases energy throughput.
Terrain "Bunker" Score	High	Determines if the room can support a compact, efficient RCL8 bunker layout.
Distance to Colony	Medium	Ensures the new expansion can be supported by "incubation" (sending workers and energy from a parent colony).

Mineral Type	Medium	Prioritizes rooms with minerals not currently owned by the TerminalNetwork.

The Strategist uses these scores to determine when and where to place a DirectiveClaim, which then instantiates the Overlords necessary to bootstrap a new colony.[6] This allows the bot to grow entirely autonomously, moving from a single room to a multi-shard empire without human intervention.[4]

The Assimilator: Collective Intelligence and Verification

A unique and controversial feature of the Overmind architecture is the Assimilator, which allows different players running the same codebase to cooperate as a single entity.[4] This "hivemind" capability facilitates the sharing of creeps and resources and allows for coordinated military responses across player accounts.[2]

Verification and Code Integrity

To ensure that assimilated players are not running modified versions of the codebase (which could be used to siphon resources without contributing), the Assimilator implements a strict verification protocol.[5]

1. **Checksum Generation:** The @assimilationLocked decorator registers critical parts of the script with the Assimilator, which generates a sha256 checksum of the code.[5]
2. **Heartbeat Transfers:** Every 1,000 ticks, an assimilated player sends a small amount of energy (100 units) to the master terminal. The transaction description contains the current codebase hash.[5]
3. **Master Ledger:** The Overmind compares these hashes against a master ledger of clearance codes to determine which players are trusted and eligible for cooperative directives.[5]

This system creates a verifiable "automated union" where players can mutually benefit from the collective strength of the swarm while maintaining the security of the overall network.[2]

Performance Engineering: Beyond the Heap

While the Overseer-Overlord paradigm is conceptually powerful, its practical success depends on extreme CPU optimization. The architecture employs several techniques to minimize the "management tax" of its hierarchical structure.[5]

The Caching Module and Intent Grouping

The Overmind uses a custom $ caching module to memoize the results of expensive operations. For example, $.refreshRoom() updates the cached lists of structures and resources in a room without re-scanning the entire room object every tick.[5] Furthermore, the system groups "intents"—state-changing actions like move, transfer, or harvest—to maximize the efficiency of the 0.2 CPU cost per successful intent.[13]

Optimization Technique	Implementation	CPU Impact
Memory Restoration	Restoring parsed memory between ticks.	Eliminates JSON.parse overhead.
$ Caching	Memoizing structural and environmental data.	Reduces room.find and structural scan costs.
Intent Grouping	Only calling transfer when a creep is full.	Minimizes the 0.2 intent tax.
Bucket Management	Suspending operations at low bucket levels.	Avoids the "limbo state" of high overhead with no action.

CPU Bucket Limiting

To prevent a colony from collapsing due to CPU exhaustion, the architecture implements a sophisticated limiter. Reaching a critical bucket threshold (typically below 1,000) suspends all non-essential Overlords, prioritizing only the QueenOverlord and basic defense until the bucket has replenished.[5] This ensures that the colony remains "alive" during periods of server-side lag or intense combat without incurring the extra memory parsing costs of a failed execution cycle.[6]

Conclusions: The Paradigm as a Distributed Operating System

The Overseer-Overlord paradigm represents the pinnacle of programmatic strategy in Screeps. By moving beyond simple unit-level logic and embracing a hierarchical, process-oriented architecture, the Overmind framework effectively creates a distributed operating system for a multi-room swarm.[1]

The paradigm's strength lies in its abstraction layers: the Overseer handles the "when" of adaptation, the Overlord handles the "how" of execution, and the Zerg wrapper handles the "micro" of unit movement and interaction.[1] This allows the bot to scale to hundreds of rooms and thousands of units with high efficiency and strategic flexibility.[2] While it introduces significant complexity and requires rigorous performance engineering to manage the overhead of its class-based architecture, the results—autonomous expansion, coordinated swarm combat, and inter-player cooperation—demonstrate that hierarchical governance is the most effective model for achieving high-level swarm intelligence in competitive programming environments.

Works cited

1. Screeps #1: Overlord overload - Ben Bartlett, accessed February 21, 2026, https://bencbartlett.com/blog/screeps-1-overlord-overload/
2. Overmind - Ben Bartlett, accessed February 21, 2026, https://bencbartlett.com/projects/overmind/
3. Overlord | Overmind - Ben Bartlett, accessed February 21, 2026, https://bencbartlett.com/overmind-docs/classes/overlord.html
4. bencbartlett/Overmind: AI for Screeps, a multiplayer programming strategy game - GitHub, accessed February 21, 2026, https://github.com/bencbartlett/Overmind
5. Screeps #6: Verifiably refreshed | Ben Bartlett, accessed February 21, 2026, https://bencbartlett.com/blog/screeps-6-verifiably-refreshed/
6. Releases · bencbartlett/Overmind - GitHub, accessed February 21, 2026, https://github.com/bencbartlett/Overmind/releases
7. Setting creep spawn que priority order | Screeps Forum, accessed February 21, 2026, https://screeps.com/forum/topic/2418/setting-creep-spawn-que-priority-order
8. Screeps #18: Spawn Uptime - Field Journal, accessed February 21, 2026, https://jonwinsley.com/notes/screeps-spawn-uptime
9. Overmind - Ben Bartlett, accessed February 21, 2026, https://bencbartlett.com/overmind-docs/
10. Remove the CPU cost of having memory and scale available memory akin to CPU | Screeps Forum, accessed February 21, 2026, https://screeps.com/forum/topic/760/remove-the-cpu-cost-of-having-memory-and-scale-available-memory-akin-to-cpu
11. Encouraging more combat at high GCL | Screeps Forum, accessed February 21, 2026, https://screeps.com/forum/topic/2809/encouraging-more-combat-at-high-gcl
12. Screeps #5: Evolution | Ben Bartlett, accessed February 21, 2026, https://bencbartlett.com/blog/screeps-5-evolution/
13. Looking For Efficiency Tips : r/screeps - Reddit, accessed February 21, 2026, https://www.reddit.com/r/screeps/comments/ocdfg8/looking_for_efficiency_tips/
14. Screeps performance concerns - Reddit, accessed February 21, 2026, https://www.reddit.com/r/screeps/comments/iax8c1/screeps_performance_concerns/

15. Need some help improving my CPU usage | Screeps Forum, accessed February 21, 2026, https://screeps.com/forum/topic/2381/need-some-help-improving-my-cpu-usage