ChatGPT

Swarm Behaviors: From Fish Schooling to Auto-Shooter Arenas

1. Fish/Swarm Behavior Theories

- **Cohesion–Alignment–Separation (Boids Rules):** Individuals follow simple local rules – they steer toward the group's center, match neighbors' direction, and avoid crowding [1]. These decentralized interactions yield emergent cohesive movement without any leader, as seen in fish schools or bird flocks [2] [3].
- **Predator Confusion & Risk Dilution:** Large schools gain safety in numbers. Synchronized movements and sheer density confuse predators and make it hard to single out prey [4]. Each individual's risk of being caught is diluted by the presence of many others ("oddity effect" – any prey that stands out is targeted first [5]). This collective defense mechanism is a core reason fish evolved tight shoaling behavior.
- **Flash Expansion (Scatter Burst):** Upon sensing a threat, a school may suddenly explode outward in all directions [6]. This **flash expansion** rapidly increases the group's volume, creating chaos and distance to evade a lunging predator. It's a split-second evasive maneuver where unified form gives way to maximal dispersion, hoping the predator strikes empty water.
- **Fountain Effect:** Similar to flash expansion, but here prey bolt *away from the predator's incoming path*. Fish near an approaching predator accelerate in the opposite direction, creating a "fountain" spray pattern opening away from the threat [7]. This clears an escape corridor and can divert the predator through the middle of where the school was – catching nothing but water.
- **Vacuole Formation:** Schooling fish often form an empty **vacuole** (hole) around a predator that dives into the school [8]. The school momentarily creates a donut-shaped formation with the predator in the center and no prey in its immediate reach. This coordinated spacing keeps just out of the predator's strike range, then the school can close the gap once the predator exits.
- **Split and Reunion Tactic:** Another evasive strategy is for the school to **split into multiple groups** when attacked, then later reform. A predator rushing a dense school might find the school cleaving into two swarms that veer apart (sometimes called *"split and scatter"*). Once the threat passes, the groups merge back together [9] [10]. This confusion tactic can cause the predator to pick the wrong sub-group or lose momentum.
- **Compaction (Bait Ball):** In some cases, prey do the opposite of scattering – they **pack tightly into a ball**. Fish like sardines herd into a spinning bait ball when threatened, presenting a unified whirling mass [11]. This **compaction** makes it hard for a predator to target one individual and can intimidate or bewilder attackers. It's risky (a well-aimed predator can still bite into the ball), but as a short-term defense it leverages collective presence to confuse (and sometimes even physically deter) the predator.

Real-World Swarm Behavior Insights

Schooling fish are a prime example of **swarm intelligence** – complex group behavior arising from simple rules. Craig Reynolds' classic Boids model showed that cohesion, alignment, and separation rules produce lifelike flocking and schooling in simulations [1]. Each “boid” (bird-oid) steers based on local neighbors, not global instructions, yet the whole group moves as an organized unit [2]. This decentralized coordination is observed in nature: fish adjust to neighbors’ positions via vision and lateral line sensing, maintaining optimal spacing and direction without any leader [3].

Schooling provides many survival benefits. A large, synchronized school creates a **predator confusion effect** – the visual of dozens of fish moving in unison confounds predators’ targeting [4]. Experiments show predators have lower attack success on tightly schooling prey, as the swarm’s rapid collective maneuvers make it difficult to focus on one fish [4]. Additionally, the **dilution of risk** means each individual is less likely to be the unlucky victim in a big crowd [4]. Biologists note that fish actively prefer larger shoals for this reason. Any individual that looks different (color, size) becomes the “odd one out” and gets picked off first [5] – hence natural selection often favors uniform appearance in schooling species.

When a predator does strike, schools have evolved remarkable evasive maneuvers. A **flash expansion** is a near-instantaneous explosion of the school in all directions [6]. High-speed video of fish attacked by fast predators (like marlin or tuna) shows the school expanding like a burst bubble, leaving a predator lunging into a void. A related response is the **fountain effect**, where fish fling themselves away from the predator’s approach path [7]. If a predator charges the center of the school, the nearest fish shoot outwards radially, creating a gap – akin to a parted curtain – so the attacker passes through empty space. The school often *reforms behind* the predator once it passes (sometimes called the **“fountain maneuver”** as fish stream around and behind the predator).

Sometimes, instead of scattering, fish tighten up. **Compaction** into a bait ball is typically seen when multiple predators are around or escape routes are limited [11]. By bunching up, the fish present a whirling sphere that’s visually and sensorily overwhelming – all those bodies and flashing scales can confuse predators (and also make it harder for a single strike to catch more than a mouthful). However, this is usually a last resort when escape seems unlikely, since it’s a “all eggs in one basket” defense. Dolphins and sharks will often team up to keep a bait ball corralled. For the prey, constant movement within the ball is key – if they stay dynamic, predators have trouble zeroing in.

Another advanced tactic is **split and reunion**. Researchers using simulations have identified patterns where a school under attack will split into two or more groups moving in different directions [9]. This forces the predator to pick one target group. Once the threat is gone, the groups often loop back and merge again, reforming the school (hence *“reunion”*) [9]. Real-world observations of fish (and other swarm animals like krill or birds) confirm that splitting is a common anti-predator response, especially against fast-swimming predators that charge through the middle of a school. Splitting reduces casualties – a predator can only chase one mini-school at a time – and the unpredictability of which way the groups go adds to the confusion.

Underpinning all these behaviors is the idea of **self-organized criticality** at the “edge of chaos” [12]. Fish schools can rapidly transition from an ordered cruising formation to seemingly chaotic dispersion and back to order, all through local interactions. The unpredictability of the group’s exact movements over time (yet maintaining cohesion) is an emergent property also sought in artificial life simulations [13]. For instance, a school might generally swim east, but exactly where individuals will be a minute later is hard to predict – this gives predators a tough time. Yet the school doesn’t disintegrate completely; cohesion reasserts after evasive maneuvers, showing a resilient **collective behavior** that is neither static nor random chaos [14].

These natural swarm behaviors have clear parallels in gaming and robotics (via swarm algorithms). The Boids algorithm itself became a staple for simulating flocks in film and games (from Batman's bat swarm to schooling fish in Finding Nemo). Importantly, boid-like agents are computationally efficient when using spatial partitioning to limit neighbor checks [15], allowing real-time simulation of hundreds or thousands of agents – an approach that game developers have adopted for large crowds or enemy hordes. The lessons from fish swarms – like maintaining spacing, reacting only to local neighbors, and coordinating evasion – provide a rich toolbox for designing game AI that **feels lifelike and unpredictable yet coordinated**.

2. Auto-Shooter Enemy Behavior Theories

- **Zerg Rush Horde (Direct Pursuit):** The most common pattern in "bullet heaven" auto-shooters is sheer volume: dozens of weak enemies **beeline straight toward the player**. This overwhelms by numbers and forces crowd-control movement. Games like *Vampire Survivors* famously throw hordes that simply **march at the player's position** [16], relying on quantity over complex AI. The challenge emerges from managing and kiting an ever-growing mob.
- **Flocking/Clustering Swarm AI:** More advanced hordes use flocking rules so enemies don't bunch up too uniformly. For example, a group of monsters might maintain spacing and move as clusters, creating **dense packs with small voids** the player can slip through [17]. This boid-like approach (seen in some Unity prototypes) makes enemy masses feel organic – swarms undulate and avoid all converging on one point, instead semi-encircling the player in arcs or waves.
- **Encirclement Patterns:** Some enemy waves deliberately **spawn in formations around the player**. A classic example is the ring of enemies that appears to trap the player (e.g. the "flower wall" event in Vampire Survivors spawns a circle of stationary plants around you [18]). This forces the player to break through a gap before the ring closes. Encirclement creates a sudden positioning puzzle: you're surrounded unless you find or create an opening [18].
- **Leader–Follower Packs:** Enemies can exhibit pseudo-social behavior by forming squads. A small group might **follow a designated leader** unit, sticking in formation until engaging. For instance, an enemy "pack" could roam together behind a tough leader, then disperse and attack individually once within range [19]. This mimics wolf-pack or fish-school dynamics and introduces target prioritization (take out the leader to disrupt the formation).
- **Orbital Strafing and Ranged Orbits:** Rather than bee-lining, some enemies **circle the player at a fixed radius**. They maintain distance while possibly shooting projectiles or waiting for openings. This circling behavior (seen in certain arena shooters) creates a moving hazard ring around the player. For example, a "swarmer" enemy type in one game tracks an *ideal position on a circle around the player*, causing a group of them to orbit when in range [20]. The player must dash out or pick them off to break the ring.
- **Coordinated Charge (Staggered Dashes):** Enemies with charge attacks can be coordinated so that **only a few charge at once**, rather than all at the same time. This staggers the threat and keeps the player on their toes continuously. In *Jetboard Joust*, the developer gave "swarmer" enemies a **ram attack** but controlled it via a "baton" token – only the enemy holding the baton would lunge at high speed, then pass the baton [21]. This results in alternating dive-bombs instead of an overwhelming simultaneous rush.
- **Flanking and Pincer Movements:** Smart enemy AI can detect if they're all coming from one side and split to **attack from multiple directions**. For instance, if the player is keeping all zombies in front, a subset might peel off and loop around behind cover or via the arena's wraparound edges to flank. The Jetboard Joust devlog describes a "split AI" where swarm enemies, if too clustered on one

side, will **wrap around to the other side** of the arena to pincer the player [22]. This prevents a simple kite-in-one-direction strategy and forces the player to manage threats on several fronts.

Examples in Auto-Shooter Games

The modern *auto-shooter* or "horde survival" genre (e.g. Vampire Survivors, Brotato, Soulstone Survivors, etc.) leans on enemy behaviors that create constant pressure while remaining relatively simple individually. The baseline behavior is indeed the **zerg rush**: endless weak foes homing toward the player. In Vampire Survivors, "enemies… attack the player's character simply by approaching them" [16] – there's no complex pathfinding, just a straightforward pursuit. This works because when hundreds of such agents spawn from all sides, the aggregate forms a living maze that the player must navigate. The design intentionally evokes the *feeling of being encircled by a horde* even if each enemy is dumb. At higher difficulties or later minutes, the screen fills with monsters such that **movement becomes the only way to survive**, weaving through any gaps before they close.

Despite the simplicity of individual AI, developers have found ways to make the swarm behavior more interesting. A prominent technique is using **flocking algorithms (boids)** to guide hordes. One Unity project demonstrated enemies grouped into swarms with a boids algorithm, which created **"dense clusters with voids players can move through"** [17]. Instead of a uniform blob, the enemies naturally form eddies and streams. This emergent spacing is reminiscent of fish schools – the "voids" between clusters are analogous to the holes predators exploit in bait balls [17]. For the player, it means there's a dynamic pathfinding element: the horde isn't an impenetrable wall but has weaknesses and gaps if you watch their flow. Vampire Survivors itself doesn't explicitly implement boid behaviors (enemies just converge), but many VS-like indie projects experiment with these algorithms to achieve massive enemy counts without clumping. The result is often a more **organic swarm** that "breathes" – tightening and loosening around the player – which can feel more like battling a living horde with its own group dynamics.

Encircling maneuvers are used sparingly as spice in these games. The Vampire Survivors "flower wall" event is a great example: a ring of immobile enemies spawns around the player at a set radius, slowly closing in [18]. It's essentially a timed cage. The only way out is to break through one side of the circle before it fully encloses you (the plants eventually vanish, but linger long enough to panic an unprepared player). This design was directly inspired by classic shooter moments where enemies surround the player. It teaches the player that sometimes standing still is death – you must carve an exit route. The **"traps with circles of enemies"** became a noted feature of VS's design [18], and many other horde survivors now include periodic surround events or boss patterns that confine the arena. It's an artificial recreation of the **vacuole effect** in nature – but in reverse: here the *enemies* form the ring around the prey (player), whereas in fish schools the prey forms a ring around the predator. Either way, it's a deliberate positional challenge.

Different enemy *roles* often exhibit distinct movement AI. **Ranged enemies** in these games commonly keep a distance – for example, in Brotato some enemies fire projectiles and try to stay at mid-range, strafing around the player. This is a basic form of **orbital behavior**. They are coded to maintain a certain radius and line of sight, effectively circling if the player moves. Some games explicitly implement orbiters: the Jetboard Joust "swarmer" enemy we discussed is designed to circle the player by assigning each enemy a position on an imaginary ring around the player [20]. As the AI constantly updates the target positions (rotating them), the enemies end up **spiraling around**. This is a more controlled version of what some top-down action games do with melee enemies that strafe to surround the player before attacking (seen in certain beat 'em ups and ARPGs). The effect is to apply pressure from all directions; the player can't just point one way and shoot, because the enemies aren't lining up in a single file.

To keep such circling enemies from being too static, developers add burst attacks. A common pattern is the **telegraphed charge**: an enemy circles or approaches slowly, then occasionally darts in a straight line towards the player (often with increased damage or speed). *20 Minutes Till Dawn* (another shooter) has enemies that pause then dash at you, for instance. Jetboard Joust's solution was elegant – the **baton system for charges** [21] . By limiting how many enemies can charge at once, it prevents the scenario where dozens of foes all dash together (which would be almost impossible to dodge and also wasteful as many would overshoot). Instead, you get a **rhythmic series of lunges**, one or two at a time, much like how wolves or velociraptors in fiction take turns attacking a cornered prey. This feels more *intentional* and less chaotic, and the player can manage it by timing dodges or focusing down the charging enemy before the next one comes.

Smarter group tactics like **flanking** are starting to appear in advanced games, though in the pure "survivors" subgenre most enemies are still relatively mindless. However, we can draw from other top-down shooters or even FPS AI for inspiration. In Jetboard Joust, when the developer noticed all swarmers clumping on one side of the player, he implemented a split behavior: a subset would intentionally go the opposite way around the game world (that game has wraparound edges) to attack from behind [22] . This is essentially a **pincer maneuver** – very similar to how real predators might work or how military AI in RTS games might flank. In an open arena with no walls, doing this requires either a wraparound world or just coordinating spawns on multiple sides. Some VS-like games achieve a similar effect by spawning new waves from the opposite side to where the player is kiting. The end result is the player suddenly finds enemies coming from the front and back, forcing a quick change in strategy. **Surround AI** (where enemies attempt to position around the player rather than all in front) is a bit more complex to code but can be approximated by simple rules (e.g., if an enemy is too close to directly behind another enemy relative to the player, it tries to sidestep to get a flank angle). Even without true tactical AI, designers can script events or diverse spawn points to simulate a coordinated attack from multiple sides, which keeps the player moving dynamically rather than circle-strafing in one direction endlessly.

In summary, auto-shooter enemies often borrow swarm concepts in a simplified form: huge numbers for confusion/dilution, occasional **encirclement** to trap the player (mirroring predator–prey ring dynamics), **spacing behaviors** to avoid overly dense clusters (making the horde flow around the player like a school of fish around a diver), and **diverse attack patterns** (chargers, orbiters, leaders) to introduce the need for positional responses. These behaviors not only make gameplay more engaging but also tap into something viscerally satisfying – it feels *alive*. Players often describe the hordes in Vampire Survivors or similar games as a "living wave" or "swarm", indicating that even without complex pathfinding, the collective movement of enemies triggers our instinctual recognition of swarm behavior. Leveraging those patterns, inspired by nature, leads to enemy mobs that are both fun to mow down *and* compelling to dodge.

3. Arena Mechanic Concepts (Swarm/Movement Focused)

- **Arena 1 – Gentle Swarm Gauntlet:** An introductory arena featuring a slow, steady trickle of weak foes from all sides, teaching the player to **keep moving and kite**. Enemies spawn in a loose "ring" formation but with large gaps, mirroring a dispersed fish school. The player practices weaving through openings in the encroaching circle rather than getting surrounded. (This leverages basic cohesion/separation: monsters don't all stack up, leaving lanes to slip through.)

- **Arena 1 – Predator Chase Tutorial:** One durable enemy (or a small pack) fixates on the player with moderate speed – essentially a “predator” to outrun. The arena itself might be free of obstacles, emphasizing **movement and spacing**. This scenario encourages circle-strafing and retreating in a loop, akin to a shark chasing a fish where the fish must turn frequently to avoid a straight-line catch. It introduces the idea of not letting an enemy touch you by maintaining distance (fundamental in swarm games).
- **Later Arena – Encirclement Escape Challenge:** A mid-game arena spawns a **ring of enemies that slowly closes in**, echoing the vacuole or surround tactics from nature. For example, stationary turrets or slow golems spawn in a circle and constrict the playable area. The twist: one section of the ring has a weaker enemy or a gap – a **designed exit**. The player must break through at that point before being enclosed (teaching target prioritization and timing). This replicates the “trapped by swarm” feeling and trains quick **route finding under pressure** [18] .
- **Swarm Split & Flank Arena:** An arena with partitions or a wraparound layout that causes the enemy horde to **split into two prongs** (e.g. two separate entrances funnel enemies into the area). The player faces a coordinated two-front assault (left/right or top/bottom). This concept mimics fish splitting and a pincer attack [22] – the player can’t just run one direction; they must dodge between two converging swarms. Mechanically, this could use two spawners timing waves so the second group hits shortly after the first, simulating intelligent flanking.
- **Flocking Formation Event:** A special wave where a large cluster of enemies moves together in a **flocking formation**, maintaining a shape (line, V-shape, or ball). For example, a “flock” of bat enemies enters in a V-formation (like geese or tuna fish school) and then breaks apart upon taking damage. During the formation phase they present a unified front (possibly absorbing frontal damage), so the player must hit from the sides or disperse them. This showcases emergent behavior: before breaking, they avoid overlapping and steer as one unit (implementable via boids steering around a moving invisible leader).
- **Charger and Void Dance Arena:** An arena emphasizing **high-speed evasion**: it features periodically spawning charger enemies (e.g. bulls or charging rams) that come in from edges in straight lines. In between chargers, small swarms wander in semi-random (boid) patterns, creating moving “obstacles”. The player’s movement challenge is two-fold: dodge the fast linear chargers (like avoiding lunging predators) and weave through the wandering clusters (taking advantage of the voids they leave [17]). This arena highlights quick reflexes and micro-positioning, essentially a “dance” with the swarm – step aside at the right moment, then slip through gaps before they close.
- **Bait Ball Boss or Core Mechanic:** Introduce an enemy formation that **behaves like a bait ball** – e.g. a swarm of minion enemies that tightly orbit a powerful leader or energy core. This swarm compaction moves as one unit shielding the core (similar to how smaller fish cluster around a big threat). The player must either break the formation (perhaps by thinning the edges with AOE attacks) or use a special ability to disperse them (analogous to how predators try to break a bait ball by slicing through it). This mechanic would stress area damage and positioning to hit the “core” once the protective ring opens. It’s a direct translation of fish defensive schooling into a game scenario (many weak units creating a big HP shield).

Implementing Swarm Mechanics in Three.js Arenas

Designing these arenas and behaviors in a Three.js (r134) powered game entails combining **algorithmic AI** with clever use of the engine’s capabilities. The centralized `gameState` can maintain global control flags (like “formation active” or target positions for flocks) while individual enemies are still updated each frame, ideally by referencing shared data to stay in sync. **Object pooling** will be essential – these ideas involve lots of entities spawning/despawning (hordes, projectiles, etc.), so reusing enemy objects from a pool prevents garbage collection spikes and keeps performance smooth.

For **Arena 1's gentle swarm**, implementation is straightforward: spawn enemies at the arena edges with a simple seek behavior towards the player. To mimic a loose ring formation, spawners can be placed roughly evenly around the circumference and enemies have slight lateral dispersion in their movement (randomized approach angle) so they don't all converge on the exact same point. No complex AI is needed here – volume and positioning do the work. The gameState could have a parameter for "spawn rate" and "spawn ring radius" that increases gradually, so the player experiences an encroaching circle. This can be tuned so that at first the gaps are huge (easy escape), but by the end of the arena the circle nearly closes (teaching the consequence of not thinning the horde).

The **predator chase** in Arena 1 (single enemy focus) can be handled by a basic pathfinding or straight-line pursuit component on one enemy entity. For instance, a fast enemy continuously updates its velocity toward the player's position each tick. In Three.js, you'd update the enemy's mesh position based on this velocity. The trick is ensuring the player can outrun or outmaneuver it – so either cap the enemy speed slightly below the player's or have it make wide turns (perhaps add a small inertia or turning speed limit). This can be done by not instantly rotating the enemy toward the player but using a lerp or steering force. `gameState` might hold the player's current position, and each frame the chaser adjusts direction a bit toward that, creating a *pursuit curve* rather than a perfect beeline (which makes it easier to juke). This behavior introduces the concept of kiting in code as well: the enemy is effectively implementing a pursuit algorithm (like "seek" in steering behaviors), which is one of the simpler AI behaviors to code.

Moving to the **encirclement escape challenge**, we'd script a specific event: e.g., at time X or wave Y, pause other spawns and generate a ring of enemies around the player's current position. In Three.js coordinates, if the player is at (Px, Pz), spawn, say, 12 enemies at radius R evenly spaced by angle (i.e., positions (Px + R*cosθ, Pz* + *R*sinθ) for θ in 30° increments). These enemies could either be stationary (forming a literal wall) or slowly move inward (each has a target point slightly inside from its spawn, maybe converging toward the player's position). A simple approach is to give each of these enemies a velocity vector pointing toward the center of the ring. If stationary, they just act as obstacles that eventually time-out. If moving, they create a closing wall – you might implement that by reducing the radius R over time (e.g., each enemy's target radius shrinks a little every second). To telegraph the **gap**, you could spawn one of them with significantly lower HP or make one segment an actual gap with no enemy at spawn – the player will naturally head for the weak point. From a coding perspective, this is more of a level-scripted pattern than emergent AI. It's easily controlled via gameState flags: set `gameState.encircled = true` and perhaps store the positions of ring enemies in an array if needed (though standard collision detection might suffice). After either a duration or once the player breaks out, you flip the flag and resume normal spawns.

The **swarm split & flank arena** leverages layout. If your game supports obstacles or walls (even just invisibly partitioning the arena), you can funnel enemies into separate streams. For example, imagine a central obstacle so enemies have to go left or right around it to reach the player. Many will naturally split. To ensure a split, you could spawn two groups at opposite ends of the arena. One implementation trick: maintain two target points in gameState, one slightly to the player's left and one to the right. Assign half the enemies to target the left point, half to the right. This will cause the horde to bifurcate and attempt a pincer. If using a wraparound world, you can exploit coordinates (some enemies interpret the shortest path which might be "the other way around" due to wraparound). Essentially, the AI here can still be simple seek behavior, but by manipulating *what* they seek (not always the exact player position but an offset of it), you achieve a flank. The Jetboard Joust devlog's approach was to have a condition if too many enemies have similar angle to player, mark some to go the opposite way [22] . We can mimic that: each frame, compute the average direction of all enemies relative to player. If that average direction is, say, within a 90° cone, then for a random subset of enemies, invert their direction vector (e.g., add 180° to their movement angle). In a centralized gameState system, you could have a routine that monitors enemy distribution and issues a "flank order" to some when things get too one-sided. This adds a layer of dynamic difficulty – if the player keeps all enemies on one side, the game responds by sending some around back. Pooled entities can handle this fine; you're just updating their velocities on the fly.

Now, the **flocking formation event** is a chance to use true boids or a simplified version. Suppose you want a formation of enemies moving as a unit. You could designate one enemy as the notional "leader" (or have an invisible leader object) that follows a path. All formation members then try to maintain a relative offset or follow distance from that leader. A simple formation like a line or V can be done by assigning each enemy a formation index and a desired offset position from leader (for a V, leader in front, others at diagonal offsets). They then steer not directly toward the player but toward their formation position relative to the leader. This can be done in gameState by storing an array of offsets for the formation and updating each enemy's target. Another method is full boids: enable alignment, cohesion, separation behaviors among that specific group. Three.js doesn't have anything special for this; you'd implement by iterating through that group's entities each frame and applying forces. For example, for each enemy in the flock, find neighbors (could limit to just that flock group for simplicity), compute average heading (alignment), center (cohesion), and separation vector, then adjust velocity. The outcome is they'll flow together. During the formation event, maybe also have them ignore the player and just move in some pattern until "shot at", then break and switch to normal homing AI. This gives a dramatic effect: the player sees a disciplined formation approach (perhaps like a squadron of flying enemies in a pattern, a nod to classic shmups), then when the formation "breaks", it becomes a regular chaotic melee. All of this is doable with moderate math. **Spatial hashing** or grid can optimize neighbor queries if boids are many, but for a limited formation, simple loops are fine.

The **charger & void dance arena** mixes two enemy types and relies on contrast. Implementation-wise, you'd have one enemy prefab for "charger" with a behavior: on spawn, pick a random straight-line trajectory across the arena (maybe aimed at the player's current position or just randomly across the arena), accelerate to high speed, and despawn or slow down after passing a certain distance. This is easy to implement by giving it a fixed velocity vector and no homing – essentially a "dumb projectile" enemy. Because they move fast, using Three.js, you might update their position in larger increments or even use a tween for a smooth effect. Meanwhile, the wandering clusters can use a boid-lite AI: give them a small random wandering force plus a very mild attraction to the player (so they meander generally toward the player but not directly). This creates moving obstacles that aren't laser-focused on the player – the **voids** in their spacing will shift unpredictably. The player has to navigate these while also reacting to the occasional charger zooming through. Balancing this in gameState means possibly staggering the spawns: e.g., every 10 seconds spawn a wave of 3 chargers that streak by, while continuously spawning 20 slow swarmers that wander. You might use object pooling to recycle the charger enemies quickly as they exit one side and reappear later. A global timer or state machine can orchestrate these alternating patterns.

Finally, the **bait ball boss** or core-swarm concept is an exciting AI challenge. One approach: have a central "boss" entity that doesn't move much, and a swarm of minions that orbit it in a sphere or circle. You can achieve the orbit by assigning each minion an angular position around the boss and update their angle each frame (e.g., theta += ω * dt for each, with ω maybe varying slightly to not be too uniform). In Three.js coordinates, position each minion at `boss.position + radius * (cosθ, sinθ)` around it (assuming

2D plane). This creates a ring of bodyguards. To make them also react to the player, you might allow the ring to **flex or expand** when the player gets too close (like a school expanding). For instance, if the player comes within a certain range, temporarily increase the orbit radius or let some minions break formation to attack. After the attack, they return to orbit (reform the bait ball). This behavior closely mirrors how real bait balls sometimes send a few individuals out as sacrificial distractors (or simply how the ball loosens when threatened and then reforms). Technically, you'd coordinate this via gameState flags: `bossPhase = "defense"` means all minions maintain orbit; `bossPhase = "attack"` means allow minions to chase player briefly. The transition could be time-based or health-based (e.g., every 30% boss HP lost, the swarm "panics" and scatters outward = flash expansion, then reforms). Using pooled entities for the minions is crucial because the boss might spawn many over time (if the boss can regenerate them). You might even recycle minions: when one is destroyed, spawn a new one from the pool after a delay to simulate continuous reinforcement until the core is killed.

Throughout these implementations, a **centralized gameState** can manage high-level patterns (when to encircle, when to spawn waves, global flock toggles) while each enemy's internal logic handles the frame-to-frame movement. This separation is valuable: complex swarm behaviors often emerge from simple individual rules, but orchestrating special events (like a perfect ring spawn or synchronous charge) is easier from a top-down control. A centralized system can temporarily override individual AI (e.g., "all enemies, align into formation now") and then release control back to local steering. By pooling entities, you also ensure that these dramatic swarm flourishes don't introduce performance hiccups – you can spawn 100 orbiting minions or 50 flanking zombies without a spike, because they're recycled objects with preallocated Three.js meshes and geometries.

In summary, the above arena concepts translate real-world swarm tactics into gameplay by either **scripted formation events** (encirclements, bait ball) or by applying **swarm algorithms** (flocking, separation, coordinated charges) to enemy AI. With Three.js, one must handle all the movement logic in JavaScript, but the engine capably renders thousands of simple Sprites/meshes if the logic is efficient. Utilizing boids algorithms can give lifelike motion to hundreds of enemies at low CPU cost, especially if combined with spatial hashing to limit neighbor calculations [15]. The result should be arenas that *feel* alive with swarming enemies: sometimes clustering, sometimes scattering, sometimes surrounding the player – a dynamic challenge that evolves as the player progresses. By grounding these mechanics in established swarm behavior theories (whether fish-inspired or via known game AI patterns), we ensure that the gameplay not only is fun but also intuitively resonates with the player's understanding of group behavior, making the challenges **engaging, readable, and excitingly emergent**.

Figure: A school of surgeonfish maintaining cohesive formation. Such real-world schooling principles (alignment, spacing, coordinated turns) inspire algorithms for enemy swarms in games [3] [6]*. In gameplay, this translates to groups of enemies that flow around obstacles and the player, rather than acting as independent random units.*

---

[1] Creating realistic Swarms - Boid Movement
https://www.gamedeveloper.com/design/creating-realistic-swarms---boid-movement

[2] [12] [13] [14] [15] Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)
https://www.red3d.com/cwr/boids/

[3] [4] [6] [7] [8] [10] Fish schooling | Swarm Intelligence and Robotics Class Notes
https://fiveable.me/swarm-intelligence-and-robotics/unit-2/fish-schooling/study-guide/JDfDKMIP44U95ddb

[5] Shoaling and schooling - Wikipedia
https://en.wikipedia.org/wiki/Shoaling_and_schooling

[9] [2210.03989] A Stochastic Differential Equation Model for Predator-Avoidance Fish Schooling
https://arxiv.org/abs/2210.03989

[11] [PDF] 15_schooling & migration.pdf - CSUN
http://www.csun.edu/~msteele/classes/Ich530/lectures/15_schooling%20&%20migration.pdf

[16] Enemies | Vampire Survivors Wiki - Fandom
https://vampire-survivors.fandom.com/wiki/Enemies

[17] Mutiplayer - players vs 10k hordes of enemies using Unity DOTS + Reactor Mutliplayer Engine : r/Unity3D
https://www.reddit.com/r/Unity3D/comments/1qjvu5m/mutiplayer_players_vs_10k_hordes_of_enemies_using/

[18] Vampire Survivors screenshots - MobyGames
https://www.mobygames.com/game/178788/vampire-survivors/screenshots/

19 20 21 22 Jetboard Joust DevLog #109 – Making New Enemies pt 1 « The BitBull Blog
https://blog.bitbull.uk/2020/01/13/jetboard-joust-devlog-109-making-new-enemies-pt-1/