

Modular Encounter System

Designer controlled combat encounters connected to collectible progression and spatially activated gates.

01 SYSTEM OVERVIEW

The player enters an authored arena and activates its encounter trigger. An Arena Manager selects enemy classes from a designer defined pool and spawns them over time at placed Target Points. The encounter ends only after the full spawn quota has been created and every active enemy has been defeated. Completion generates one Memory Fragment at an authored reward position.

The player collects fragments across separate arenas. Each pickup records a unique AreaID in the player inventory. A portal checks the required number of collected identifiers when the player enters its trigger. If the requirement is met, the linked gate moves to its open position. Otherwise, the portal reports the remaining requirement.

Player experience



DESIGN INTENT Make encounter composition editable in the level while keeping spawn rules, kill accounting, and progression logic in reusable Blueprints. The encounter determines what is fought. The level determines where and when the player fights it.

02 SCOPE AND IMPLEMENTATION

Runtime	Authoring model	Combat integration
Unreal Engine 5, Blueprint gameplay	One Arena Manager instance per encounter	Four NPC variants inherit from BP_Energy_Base
Single player character	Designer placed spawn markers and reward point	Shared AIController, health, damage and death events

SYSTEM ARCHITECTURE

Runtime responsibilities

Each class owns one part of the encounter lifecycle. Cross system communication relies on references, events, and a small number of explicit data contracts.

03 BLUEPRINT OWNERSHIP

Blueprint	Primary responsibility	Key interface
BP_ArenaManager	Owns trigger activation, spawn cadence, active enemy counts, encounter completion and the reward spawn.	StartEncounter, TrySpawnEnemy, HandleEnemyDied, CheckArenaCompleted, CompleteArena
BP_Enemy_Base	Shares combat attributes, damage reception, health checks, attack Montages and death behavior across enemy variants.	OnEnemyDied dispatcher
BP_AIC_Enemy	Controls navigation, pursuit and attack requests using aggro and attack ranges.	TryAttack on the controlled enemy
BP_MemoryFragment	Collectible created at RewardPoint after a completed encounter.	AreaID passed on spawn, then AddFragment on pickup
BPC_MemoryInventory	Stores completed identifiers on the player. A Set of Names prevents duplicate credit for the same AreaID.	AddFragment, GetFragmentCount, HasEnoughFragments
BP_PortalTrigger	Evaluates the player's collection objective and routes a successful interaction to linked gates.	RequiredFragments, GatesToOpen, ActivatePortal
BP_MemoryGate	Moves the blocking gate mesh to an authored open position.	OpenGate, OpenOffset, OpenDuration

04 COMMUNICATION CONTRACT

01 Arena to enemy. Spawn a BP_Enemy_Base child class and bind the spawned instance to its OnEnemyDied dispatcher.

02 Enemy to arena. The death event emits the dying enemy reference. The manager updates AliveEnemies and evaluates completion.

03 Arena to reward. CompleteArena spawns one BP_MemoryFragment using RewardPoint world transform and the configured AreaID.

04 Reward to inventory. Pickup sends the fragment AreaID to the player inventory, then removes the collectible from the level.

05 Inventory to portal. The portal queries HasEnoughFragments using RequiredFragments. On success, ActivatePortal calls OpenGate on its gate references.

SEPARATION OF CONCERNS The AIController does not own the arena objective. NPC deaths do not spawn rewards. The pickup does not open gates. These boundaries allow combat and progression to change independently.

LEVEL AUTHORING

Designer facing controls

The system is built for iteration from placed Blueprint instances. Encounter designers do not need to duplicate the spawn logic to create a new combat space.

05 ENCOUNTER SETUP WORKFLOW

01 Place BP_ArenaManager. Position the manager in the level and resize ArenaTrigger to cover the intended entry threshold.

02 Define the reward location. Move RewardPoint to a readable position where the player can locate the fragment after combat.

03 Place spawn locations. Distribute Target Points across navigable ground. Use cover, traversal routes and sightlines to control pressure and readability.

04 Author the enemy composition. Choose the available Paragon child classes, spawn quota, interval and AreaID in the manager instance.

05 Bind progression. Set the portal objective threshold and assign the gate Actors that should open when the collection requirement is met.

06 PRIMARY AUTHORING PARAMETERS

Parameter	Location	Designer decision
TotalEnemiesToSpawn	BP_ArenaManager	Total number of successful enemy spawns required for the encounter.
SpawnInterval	BP_ArenaManager	Time in seconds between spawn attempts. Controls encounter density and pressure.
EnemyTypes	BP_ArenaManager	Array of BP_Energy_Base child classes available to the encounter.
SpawnPoints	BP_ArenaManager	Array of Target Point Actor references placed in the level.
ArenaTrigger	BP_ArenaManager component	Physical boundary that begins the encounter.
RewardPoint	BP_ArenaManager component	World position where the Memory Fragment is generated.
AreaID	BP_ArenaManager / fragment	Unique progression identifier granted by this arena.
RequiredFragments	BP_PortalTrigger	Number of unique collected identifiers required to activate the portal.
GatesToOpen	BP_PortalTrigger	References to gate Actors controlled by the portal.
OpenOffset / OpenDuration	BP_MemoryGate	Direction, travel distance and duration of the gate motion.

Example encounter: Ruins_A

Five enemies spawn at a two second interval using a pool of four NPC variants. Four Target Points distribute pressure across the arena. The manager creates exactly one fragment after the last enemy dies. The sample portal objective is eight unique fragments gathered from separate encounters.

guifamar.com

GAMEPLAY EXECUTION

Encounter state and completion

The encounter uses a small runtime state with deterministic counters. Completion is evaluated from confirmed spawns and confirmed enemy deaths, not from elapsed time.

07 ENCOUNTER LIFECYCLE

Phase	Entry condition	System response
Dormant	HasStarted is false.	ArenaTrigger waits for a valid BP_ThirdPersonCharacter overlap.
Active	Player enters the trigger for the first time.	StartEncounter sets HasStarted, initializes counters and begins the spawn timer.
Spawning	Quota not yet reached.	TrySpawnEnemy selects an authored class and point, spawns an enemy and binds death notification.
Awaiting clear	Spawn quota reached, at least one enemy alive.	No additional enemies are generated. The encounter continues until all active enemies die.
Complete	Spawn quota reached and AliveEnemies is zero.	CompleteArena marks IsCompleted and creates one Memory Fragment.

08 BLUEPRINT LOGIC CONTRACT

On ArenaTrigger overlap:

if player is valid and HasStarted is false:
StartEncounter()

TrySpawnEnemy():

choose one EnemyTypes class and one SpawnPoints actor
spawn using the selected point transform
if spawn succeeded:
increment SpawnedEnemies and AliveEnemies
bind this enemy to HandleEnemyDied
evaluate CheckArenaCompleted()

HandleEnemyDied(enemy):

decrement AliveEnemies once for this enemy
CheckArenaCompleted()

CheckArenaCompleted():

if SpawnedEnemies >= TotalEnemiesToSpawn
and AliveEnemies == 0 and not IsCompleted:
CompleteArena()

09 COUNTING INVARIANTS

Confirmed spawns. Increment SpawnedEnemies and AliveEnemies only when Spawn Actor returns a valid enemy instance.

Confirmed deaths. Bind OnEnemyDied for each spawned enemy. Process each death once so AliveEnemies cannot become negative.

Single reward. Guard CompleteArena with IsCompleted. No fragment is generated from an individual enemy death.

guifamar.com

PROGRESSION DESIGN

From encounter reward to portal unlock

Combat completion produces a physical reward. The player collects it, tracks the objective, then returns to an authored portal location to change the level state.

10 MEMORY FRAGMENT CONTRACT

01 Award. CompleteArena reads RewardPoint world transform and spawns BP_MemoryFragment with its configured ArealD.

02 Collect. The fragment overlap validates the player, obtains BPC_MemoryInventory and calls AddFragment using the fragment instance ArealD.

03 Register. CollectedAreas stores the ArealD in a Set of Names. A repeated identifier does not increase the unique collection count.

04 Consume no resource. Portal qualification checks collection progress but does not remove fragments. The memory objective is proof of completed encounters, not a spendable currency.

11 PORTAL AND GATE CONTRACT

```
On PortalTrigger overlap:
    identify the player and obtain BPC_MemoryInventory
    total = GetFragmentCount()
    if HasEnoughFragments(RequiredFragments):
        ActivatePortal()
        for each valid gate in GatesToOpen:
            OpenGate()
    else:
        ShowMissingMessage(total, RequiredFragments)
```

```
OpenGate():
    move GateMesh from its current relative location
    by OpenOffset over OpenDuration
    disable blocking collision when appropriate
```

12 WHY THE PORTAL IS A SEPARATE INTERACTION

The final kill does not automatically open the route. The reward appears inside the cleared arena, establishing a moment to acknowledge the result. The return to the portal gives the player a deliberate transition from combat to navigation and makes the locked route part of the level structure rather than a UI reward.

The portal can also communicate progress before the objective is complete, such as 3/8 fragments. This keeps the player aware of the requirement without making every individual enemy responsible for progression logic.

CURRENT DATA MODEL The prototype portal checks the total number of distinct ArealDs recorded on the player. It does not yet filter collection by a region or objective group. A separate objective identifier and filtered query would be required for several independent region specific gates.

ENCOUNTER DESIGN PRACTICE

Pacing, spatial composition and validation

The technical system deliberately leaves encounter composition in the hands of the level designer. The same logic can serve a compact duel, a sustained arena or a mixed enemy encounter.

13 LEVEL DESIGN DECISIONS

Approach and commitment. Place `ArenaTrigger` where the player understands the transition into combat. The entry volume should not fire while the player merely passes nearby or approaches from an unintended route.

Pressure curve. `TotalEnemiesToSpawn` and `SpawnInterval` establish a baseline cadence. A shorter interval can create overlapping threats; a longer interval leaves recovery time. Enemy health and attack patterns still determine the real concurrency.

Composition. `EnemyTypes` can combine fast pursuit enemies, high health pressure units and balanced attackers. Their roles should create readable priorities, not just variety of meshes.

Spatial distribution. `SpawnPoints` should use navigable space and avoid unfair creation directly on the player. Positioning controls sightlines, flank pressure, traversal options and time to contact.

Resolution and wayfinding. `RewardPoint` should be visible once combat clears, while the portal should remain legible as the next destination. The unlocked gate provides a physical confirmation of progression.

14 ACCEPTANCE TESTS

Test	Expected result
Trigger reentry	A completed or active encounter cannot start a second time.
Empty or invalid setup	Missing <code>EnemyTypes</code> or <code>SpawnPoints</code> does not silently produce an impossible objective.
Spawn quota	Only successfully created enemies contribute to the required spawn count.
Fast player kills	The arena cannot complete while further enemies remain scheduled to spawn.
Repeated death callback	<code>AliveEnemies</code> cannot be decremented twice for the same enemy.
Reward duplication	One completed arena produces one fragment, even after repeated overlaps.
Identifier uniqueness	Distinct <code>ArealIDs</code> increase progress; duplicate <code>ArealIDs</code> do not.
Portal requirement	Below threshold shows progress; at threshold calls the intended gate <code>Actors</code> .
Gate motion	<code>OpenOffset</code> , motion duration and final collision permit passage.

15 NEXT PRODUCTION PASS

The next implementation pass would add explicit invalid configuration handling, a `SaveGame` path for collected identifiers, richer feedback for spawn telegraphs and portal states, and encounter telemetry. I would track encounter completion time, peak simultaneous enemies, time spent without a reachable enemy and the distance traveled from the final reward to the portal.

Those measurements connect Blueprint behavior to level design decisions. The goal is not only a working spawn system, but a tool that supports repeatable combat pacing and reliable progression across multiple spaces.