# Финальный технический план проекта (2D RTS на Unity)

## Архитектурные модули

* Domain (Доменная логика): Содержит основные игровые модели и правила – например, сущности юнитов, зданий, экономики, инфекции и пр., а также логику их взаимодействия. В Domain определяются правила боя, экономики (производство ресурсов, ограничения по ресурсам и т.д.), спред инфекции, и прочие механики, независимые от движка. Все вычисления (например, разрешение столкновений юнитов, рост/смертность населения, распространение инфекции) инкапсулированы здесь в виде Plain C# Object моделей. Пример: класс EconomyManager в Domain рассчитывает суточный тик производства ресурсов[[1]](file://file_0000000049b062438e06b107048784b7#:~:text=1,%D0%B2%20%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9%20%D0%B4%D0%B5%D0%BD%D1%8C%20%D0%B2%20%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE), не зная о UI, а класс CombatSimulator рассчитывает исход сражения на основе статов юнитов.
* Data (Данные и конфигурация): Отвечает за хранение и загрузку конфигураций и сохранений. Статические данные (баланс) хранятся в ScriptableObject ассетах (например, типы ресурсов, параметры зданий, таблицы заражения) – только для чтения, не мутируются на runtime[[2]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,%D0%BF%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC%D0%B0%20%E2%80%94%20%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D1%8C%20%D0%B2%20%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%BC). При старте игры конфиги SO копируются в обычные объекты (POCO), и игра оперирует ими, обеспечивая иммутабельность исходных ассетов. Сохранения хранят состояние мира (включая все активные и агрегированные сектора) в формате JSON (с версией для миграции). Пример: GameConfig.asset – ScriptableObject со списком типов ресурсов и их свойств, SaveData – структура для сериализации текущего состояния экономики, логистики, позиций юнитов и пр.
* Application (Приложение): Слой координации, связывает Domain, AI, инфраструктуру и презентацию. Здесь располагается EventBus (централизованная шина событий) для реагирования на игровые события асинхронно и декуплинга модулей (отделяет момент отправки события от момента обработки[[3]](https://gameprogrammingpatterns.com/event-queue.html#:~:text=Decouple%20when%20a%20message%20or,from%20when%20it%20is%20processed)[[4]](https://gameprogrammingpatterns.com/event-queue.html#:~:text=Most%20games%20aren%E2%80%99t%20event,that%20want%20to%20stay%20decoupled)). Application реализует основной игровой цикл: фиксированный тик симуляции, последовательный вызов subsystems (например, Economy -> Infection -> AI -> и т.д.) и затем рассылку событий через EventBus (например, событие о постройке здания или заражении юнита)[[5]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,tick). Также здесь находятся контроллеры случаев использования – например, сервис команд игрока: получение команды от UI, валидация и передача в Domain (паттерн Command). Пример: класс GameLoopController в каждом FixedUpdate вызывает SimulationTick(deltaTime), который обновляет экономику, инфекцию, AI и др., после чего вызывает eventBus.Flush() для обработки накопленных событий[[6]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%BF%D0%BE%D0%BA%D0%B0%D0%B7%D1%8B%D0%B2%D0%B0%D0%B9%20%D0%B2%20%D1%83%D0%B3%D0%BB%D1%83%20UI,%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B8%D1%82%D1%8C%20VSync%2F%D0%B2%D1%8B%D0%BA%D0%BB%D1%8E%D1%87%D0%B8%D1%82%D1%8C%29%20%E2%80%94%20%D1%82%D0%B8%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F).
* AI (Искусственный интеллект): Отдельный модуль, реализующий поведение NPC и врагов. В проекте комбинируется Utility AI, GOAP и Behavior Trees[[7]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%98%D0%98%3A%20Utility%20AI%20%2B%20GOAP,Behavior%20Trees). Utility AI оценивает цели (например, захват базы, отступление) по скорам полезности, GOAP (Goal-Oriented Action Planning) планирует последовательность действий для выбранной цели, а Behavior Trees управляют помегнтовым исполнением действий (патрулирование, атака, строительство). AI-планирование учитывает стратегические данные (например, распределение ресурсов, состояние фронта) и умеет работать как с детализированной информацией видимых секторов, так и с агрегированными данными вне обзора (см. раздел про упрощённую симуляцию). Пример: AIPlanner генерирует план нападения, опираясь на агрегированные данные о слабой защите в соседнем секторе, а при входе в сектор в обзор – переключается на точное управление юнитами через BT. Для исключения тяжелых расчётов в каждом кадре, AI может обновлять оценки целей раз в N тиков и ограничивать частоту принятия решений для поведения низкого уровня[[8]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=match%20at%20L138%201,%EE%88%80cite%EE%88%82turn2search5%EE%88%81).
* Presentation (Презентация): Отвечает за визуализацию и ввод. Состоит из MonoBehaviour-скриптов на сцене, UI экранов и эффектов. Presentation проецирует состояние Domain на экран: спрайты юнитов, анимации, эффекты боя, интерфейс игрока. Вся логика Presentation пассивна: она подписывается на события EventBus или наблюдает за моделями Domain. Например, UnitView : MonoBehaviour слушает события о перемещении юнита или изменении здоровья и воспроизводит анимацию. UI включает экран экономики/отчёта (показывает баланс ресурсов и статистику производства[[9]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20,%D0%B2%20%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%20%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8%2C%20%D0%BE%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C%20%D0%B7%D0%B0%D0%B4%D0%B0%D1%87)), панели приказов логистики (переброска юнитов, ETA)[[10]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%92%D0%B2%D0%B5%D1%81%D1%82%D0%B8%20,%D1%81%20%D0%BF%D0%BE%D0%B2%D1%8B%D1%88%D0%B5%D0%BD%D0%BD%D1%8B%D0%BC%20%D1%80%D0%B0%D1%81%D1%85%D0%BE%D0%B4%D0%BE%D0%BC%20%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D0%BE%D0%B2), постройки фортификаций и т.д. Input (команды игрока) через UI или клики по карте преобразуется в события или команды в Application. Presentation также отвечает за отображение “фронтовой карты” – стратегического обзора всего мира: показываются сектора, уровень заражения, принадлежность территорий, чтобы игрок видел общую ситуацию. Эта карта обновляется из агрегированных данных Domain по секторам (даже если они вне обзора), например, меняет цвет сектора при его захвате врагом вне поля зрения.
* Infrastructure (Инфраструктура): Низкоуровневые детали Unity и вспомогательные системы. Здесь – менеджеры сцены, загрузка данных, файловая система для сохранений, а также интеграция Unity Jobs/ECS для оптимизации. Например, тяжёлые циклы (расчёт pathfinding, массовый апдейт инфекций) выносятся в многопоточные Jobs с Burst-компиляцией для максимального использования 12 потоков CPU AMD Ryzen 9 5900X. Возможно использование Unity ECS (Entity Component System) для отдельных подсистем: например, массовое обновление позиций снарядов или агентов инфекции через ECS- системы с фиксированным шагом[[11]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,%D0%9F%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D1%8F%D0%B9%2C%20%D1%87%D1%82%D0%BE%20%D0%BD%D0%B8%D0%B3%D0%B4%D0%B5). Infrastructure предоставляет сервис Pathfinding: хранит данные сетки пути (grid-graph), кэширует поисковые структуры и может выполнять поиск маршрутов параллельно. Также инфраструктура отвечает за профилирование и сбор метрик – интеграция с Unity Profiler, сбор логов производительности, точек расширения для автотестов (например, режим детальной отладки, когда на N-м тике печатаются ключевые данные для сравнения с эталоном).

## Ключевые подсистемы

### Экономика и ресурсы

Назначение: Моделирование производства и потребления ресурсов, рост населения, снабжение войск. Введены 4 базовых ресурса: трудочасы населения (LaborHours), энергия (Power), материалы (Materials) и еда (Food)[[1]](file://file_0000000049b062438e06b107048784b7#:~:text=1,%D0%B2%20%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9%20%D0%B4%D0%B5%D0%BD%D1%8C%20%D0%B2%20%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE). Каждый день происходит суточный тик экономики – пересчёт всей экономики раз в игровой день[[12]](file://file_0000000049b062438e06b107048784b7#:~:text=%28%D1%83%D0%BD%D0%B8%D1%84%D0%B8%D1%86%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%29.%20,%C2%A79). Правила Domain-уровня не позволяют ресурсам уходить в минус; при дефиците автоматически урезаются низкоприоритетные расходы[[13]](file://file_0000000049b062438e06b107048784b7#:~:text=%2A%2A%D1%88%D1%82%D0%B0%D1%82%D0%BD%D1%83%D1%8E%20%D0%BF%D0%BE%D1%82%D1%80%D0%B5%D0%B1%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%D0%B2%20%D1%82%D1%80%D1%83%D0%B4%D0%B0,%28%D1%81%D0%BC.%20%C2%A710). В экономическую модель входит рост населения (ежедневно на основе базы CivGrowth, добавляется в пул трудочасов) и конверсия в военных через специальные здания (казармы) – определён лимит обучения в день, стоимость в ресурсах и время подготовки[[14]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%92%D0%B2%D0%B5%D1%81%D1%82%D0%B8%20,%D0%B2%20%D1%80%D0%B5%D0%B7%D0%B5%D1%80%D0%B2%2F%D0%B3%D0%B0%D1%80%D0%BD%D0%B8%D0%B7%D0%BE%D0%BD%20%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%20%D0%B7%D0%B0%D0%B2%D0%B5%D1%80%D1%88%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D1%8F). Экономические здания (электростанции, фермы, шахты, склады и т.д.) дают приток ресурсов, потребляют трудочасы и энергию[[15]](file://file_0000000049b062438e06b107048784b7#:~:text=4%29%20%D0%A2%D0%B8%D0%BF%D1%8B%20%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B9%20%28MUST%29%201,%D1%87%D0%B0%D1%81%D1%8B%20%D0%B8%20%D0%B5%D0%B4%D1%83%20%D0%BD%D0%B0%20%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B0%D0%BD%D0%B8%D0%B8). Реализация: В Domain существует модель ResourcePool с текущими запасами и методами TryConsume() и Produce(). Подсистема экономики (например, EconomySystem) каждый суточный тик проходит по списку зданий и юнитов, суммирует производство и потребление, выполняет балансировку (отключение потребителей при нехватке). Кроме того, реализована передача части прироста ресурсов союзному ИИ (% слайдеры для каждого ресурса)[[16]](file://file_0000000049b062438e06b107048784b7#:~:text=3,%28%D0%BF%D1%80%D0%B8%20%D0%BD%D0%B0%D0%BB%D0%B8%D1%87%D0%B8%D0%B8%20%D0%BD%D0%B5%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%BA%D0%B8%D1%85%20%D0%98%D0%98). API пример: EconomyService.DoDailyTick() – применяет прирост/расход ко всем секторам, генерирует события типа ResourceBalanceUpdated.

Примечания производительности: Экономика обновляется сравнительно редко (раз в день игрового времени, что может соответствовать, например, раз в N секунд реального времени). Поэтому нагрузка низкая. Однако, вычисления сводятся в агрегированные формулы (суммы, произведения), что легко распараллеливается: при необходимости можно вынести расчёт экономики на Job, если секторов много. Использование ScriptableObject конфигов ускоряет доступ к постоянным параметрам (например, стоимость строительства, выход ресурсов) – они загружены в память и не требуют парсинга. Важна проверка стабильности: профилировать EconomySystem на 1000+ объектов (юнитов/зданий) и следить, чтобы сборка мусора не давала скачков (например, переиспользовать объекты для транзакций ресурсов). Юнит-тесты проверяют инварианты экономики: недопущение появления/пропажи ресурсов, корректность резервирования и списания[[17]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%2A%2A%D0%9F%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D0%BA%D0%B8%2F%D1%82%D0%B5%D1%81%D1%82%D1%8B%3A%2A%2A%201%29%20Unit,%C2%AB%D0%B4%D0%B5%D0%BD%D1%8C%D0%B3%D0%B8%20%D0%BD%D0%B5%20%D0%B8%D1%81%D1%87%D0%B5%D0%B7%D0%B0%D1%8E%D1%82%2F%D0%BD%D0%B5%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%C2%BB).

### Инфекция

Назначение: Симуляция распространения некой инфекции (например, зомби-вирус) среди населения и территории. Реализована гибридная модель SIR/SEIR (Susceptible, Exposed, Infected, Recovered) в сочетании с клеточным автоматом по сетке мира[[18]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=). Каждая клетка/сектор имеет счётчики населения в разных состояниях (здоровые, заражённые, иммунные), инфекция распространяется с вероятностями, заданными параметрами SIR-модели. Передача может происходить между соседними клетками – тип соседства фиксирован (напр. по 8 направлениям, Moore)[[19]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%98%D0%BD%D1%84%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%20). В Domain модуль InfectionSystem на каждом такте симуляции обновляет состояние: рассчитывает новые заражения, прогресс болезни (перевод I->R), потенциальные мутации. Инфекция влияет на экономику (заражённые не работают, умирающие снижают население) и на военные юниты (может конвертировать части армии во врагов). Реализация: Используется grid из ячеек, соответствующий разбиению карты на сектора. Внутри каждой ячейки – числовые показатели инфекции. В полном (детализированном) секторе инфекция может моделироваться даже поклеточно (на уровне позиций юнитов), а в упрощённом – агрегатно (только проценты заражённых). API пример: InfectionModel.TickCell(cell, dt) – обновляет одну ячейку по дифференциальным уравнениям SIR; InfectionModel.Spread(cell, neighborCell) – передача части заражённых соседней ячейке.

Примечания производительности: Расчёт инфекции потенциально массовый (охватывает всю карту). Для оптимизации: вне сектора игрока можно снижать частоту тиков (например, обновлять инфекцию в дальних регионах раз в несколько игровых часов). Кроме того, сам расчет SIR – простой, может быть выполнен в параллель для множества ячеек. Планируется вынести обновление большого массива клеток инфекции в Job (batch processing) либо ECS System (ECS хорош для однородных расчетов по сетке, поддерживает deterministic physics[[20]](https://docs.unity3d.com/Packages/com.unity.physics@1.0/manual/design.html#:~:text=Design%20philosophy%20,using%20ECS%20best%20practices), что полезно для повторяемости симуляции). Также, для контроля корректности, предусмотрены отладочные визуализации: подсветка уровня заражения на карте, графики S/I/R по времени, и автотесты на корректность параметров модели (сходимость, соответствие ожидаемой эпидемиологической кривой)[[21]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%98%D0%BD%D1%84%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%20).

### AI (логистика, боёвка и поведение)

Назначение: Управление противником (и союзным ИИ, если есть) – от стратегических решений до тактического манёвра юнитов. Стратегический уровень: AI анализирует экономику и фронтовую обстановку – решает, какие сектора атаковать или оборонять, куда отправить ресурсы. Используется Utility AI для выбора целей на основе множества факторов (значимость сектора, слабость обороны игрока, распространение инфекции и т.д.)[[7]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%98%D0%98%3A%20Utility%20AI%20%2B%20GOAP,Behavior%20Trees). Планировщик: после выбора цели AI строит план действий через GOAP – например, цель “захватить сектор X” раскладывается на план: построить армию → отправить армию в X → атаковать и уничтожить противника. Тактический уровень: конкретные юниты управляются через Behavior Trees (или похожий паттерн) – они обеспечивают реактивное поведение: поиск пути, атака ближайших целей, укрытие при низком здоровье и т.п. Behavior Tree узлы могут быть переиспользуемыми задачами (патрулирование области, ремонт построек, строительство фортификаций). Реализация: Модуль AI взаимодействует с Domain через команды и события – например, планировщик издаёт команду на производство юнитов, подписывается на событие UnitReady (когда производство завершено) и затем издаёт команду на марш-бросок. Локальный AI юнита – компонент UnitAI : MonoBehaviour, который получает команду “атаковать позицию” и реализует её через FSM или Behavior Tree. Логистика: отдельная подсистема, помогающая AI (и игроку) в перемещении войск. Используются логистические узлы и коридоры: узлы – ключевые точки (базы, города, станции), соединённые коридорами (дороги, ж/д)[[10]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%92%D0%B2%D0%B5%D1%81%D1%82%D0%B8%20,%D1%81%20%D0%BF%D0%BE%D0%B2%D1%8B%D1%88%D0%B5%D0%BD%D0%BD%D1%8B%D0%BC%20%D1%80%D0%B0%D1%81%D1%85%D0%BE%D0%B4%D0%BE%D0%BC%20%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D0%BE%D0%B2). AI умеет автоматически резервировать транспорт и прокладывать маршрут через узлы (например, отправка подкреплений по железной дороге), рассчитывая время в пути и нагрузку.

Примечания производительности: Основные нагрузки AI – вычисление планов и pathfinding. Планировщик Utility/GOAP работает не каждый кадр; его можно обновлять раз в несколько секунд (с небольшим случайным смещением, чтобы разные ИИ не думают строго синхронно). Behavior Trees для множества юнитов могут выполняться в каждом тик – здесь важно оптимизировать: использовать пулл объектов для BT-нод, ограничивать глубину дерева, а для простых действий (движение по маршруту) – полагаться на готовый путь от Pathfinding-системы. Pathfinding: реализован на основе сетки (tile graph) с алгоритмом A, с возможным подключением оптимизаций JPS (Jump Point Search) или иерархического поиска HPA для масштабирования[[22]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=Pathfinding%20%D0%BD%D0%B0%20%D1%81%D0%B5%D1%82%D0%BA%D0%B5%20%28A,JPS%20%E2%86%92%20HPA). Расчёт путей – кандидат для вынесения в многопоточные Jobs; при большом числе запросов может быть реализован менеджер путей, распределяющий вычисления по кадрам (например, не более N узлов A\* за кадр в главном потоке, либо полные расчёты в фоновых потоках с последующим возвратом результата). Важна детерминированность\*\* AI при прочих равных условиях – для отладки и честности геймплея: при фиксированном состоянии мира и входных данных решения AI должны повторяться. Поэтому используются фиксированные случайные сиды для рандомизации действий и избегаются плавающие точки в логике Utility (нормализованные оценки хранятся с ограниченной точностью).

### Фортификации

Назначение: Постройка оборонительных сооружений на карте. Разделены на временные (например, окопы, мешки с песком) и постоянные (ДОТы, бункеры, рвы) типы[[23]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B8%D1%82%D1%8C%20%D0%BD%D0%B0%20,%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F%20%D1%87%D0%B8%D1%81%D0%BB%D0%BE%D0%BC). У каждой фортификации определены стоимость (в трудочасах, материалах, иногда энергии), время возведения, прочность и уровень защиты, а также требования обслуживания (возможно, постепенная деградация)[[23]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B8%D1%82%D1%8C%20%D0%BD%D0%B0%20,%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F%20%D1%87%D0%B8%D1%81%D0%BB%D0%BE%D0%BC). Игрок через UI строительных инструментов может начертить линию траншеи или указать точки для постройки башен и т.д. (есть инструменты линейного строительства и узловых точек[[24]](file://file_0000000049b062438e06b107048784b7#:~:text=3.%20%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B%20%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%B0%3A%20,%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F%20%D1%87%D0%B8%D1%81%D0%BB%D0%BE%D0%BC%20%D0%B1%D1%80%D0%B8%D0%B3%D0%B0%D0%B4%2F%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0)). Постройка выполняется инженерными юнитами; скорость зависит от их количества и эффективности. Domain-слой фортификаций взаимодействует с экономикой (списание ресурсов на строительство) и с логистикой/юнитами (прикрепляет задачи к инженерным подразделениям). Реализация: Каждая фортификация – объект с состоянием (строится/готова/повреждена). В Domain существует FortificationSystem, который обрабатывает команды строительства: создает запись о новом объекте, помечает координаты, снижает ресурсы, и через EventBus оповещает Presentation (для отображения прогресса на карте). Инженерный юнит, получив задачу, периодически отправляет команду «строить Х», и FortificationSystem прогрессирует постройку. Готовые фортификации интегрируются в систему боя (дают бонус защиты союзникам внутри, задерживают заражённых и т.д.).

Примечания производительности: Количество фортификаций обычно не огромное (десятки-сотни), поэтому их влияние на производительность невелико. Однако, важно оптимизировать отрисовку длинных объектов (например, траншеи) – возможно, схлопывая сегменты в единый Mesh или используя SpriteShape для гладких линий. Физически фортификации могут представляться коллайдерами в Unity (для блокирования пути врагам), которые активны только в пределах активного сектора. Для упрощённой симуляции вне обзора нет необходимости эмулировать процесс строительства в реальном времени – можно сразу по истечении рассчитанного времени пометить фортификацию построенной. Поэтому FortificationSystem при работе в агрегированном секторе просто увеличивает таймер постройки и завершает её мгновенно, как только таймер вышел (генерируя событие о завершении без подробностей).

### Pathfinding (Поиск пути)

Назначение: Построение маршрутов для юнитов по сетке карты с учётом препятствий (террейн, здания, фортификации). Базовый алгоритм – A по тайл-сетке[[22]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=Pathfinding%20%D0%BD%D0%B0%20%D1%81%D0%B5%D1%82%D0%BA%D0%B5%20%28A,JPS%20%E2%86%92%20HPA). Для ускорения на большом числе узлов рассматриваются улучшения: Jump Point Search (пропуск промежуточных узлов на прямых участках) и Hierarchical Pathfinding A (разбиение карты на кластеры с предварительным расчётом межкластерных путей)[[22]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=Pathfinding%20%D0%BD%D0%B0%20%D1%81%D0%B5%D1%82%D0%BA%D0%B5%20%28A,JPS%20%E2%86%92%20HPA). Pathfinding работает в связке с логистикой: учитывает скорость типов юнитов, загруженность дорог (например, может увеличивать вес ребра дороги при избытке юнитов на ней). Реализация: В Infrastructure модуль PathfindingService предоставляет асинхронный метод FindPath(start, goal, unitType). В режиме полной симуляции (сектор в поле зрения) результат – список точек маршрута, которые передаются юниту. В упрощённом режиме (вне обзора) детализация пути не нужна – используется агрегированное расстояние: например, время пути между узлами логистики по заранее просчитанному графу. PathfindingService может иметь два режима: детальный (для локальных движений в активном секторе) и стратегический (для межсекторных перемещений). API пример: PathfindingService.RequestPath(startCell, goalCell, callback) – добавляет задачу на поиск пути; если используется многопоточная реализация, задача помещается в очередь, результат потом приходит в callback в основном потоке.

Примечания производительности: Поиск пути – одна из самых затратных операций, особенно если много юнитов двигается одновременно. Поэтому важна кеширование и повторное использование путей: PathfindingService хранит последние рассчитанные маршруты, чтобы повторно использовать их для других юнитов (особенно для стратегических перемещений между одними и теми же узлами логистики). Также разумно ограничить частоту перестроения пути – юнит не будет перезапускать A каждый тик, а только если отклонение от пути велико или появился новый препятствие. Unity Jobs могут параллелить несколько поисков сразу, т.к. поиск по независимым графам (или разным целям) хорошо распараллеливается. На целевой платформе (5900X) можно одновременно считать десятки путей. В идеале, профилировать время выполнения FindPath при различных размерах карты и применять HPA для очень больших карт, чтобы уменьшить поиск до нескольких промежуточных точек. Автотесты для pathfinding должны проверять корректность построенного маршрута (нет прохода через стены), а нагрузочные тесты – время вычисления 1000 маршрутов и отсутствие существенных фризов в кадре.

### События и команды (EventBus)

Назначение: Асинхронное взаимодействие между системами и обработка действий игрока. EventBus – глобальная шина, куда доменные подсистемы публикуют события (например, UnitDied, BuildingComplete, SectorLost). Другие системы подписываются на интересующие их события: UI – на обновление экономики (чтобы отрисовать новый баланс ресурсов), AI – на событие смерти юнита (чтобы, например, пересчитать силы), система сохранений – на событие "игра окончена" (для автосейва) и т.д. Паттерн «event queue» гарантирует, что отправка события отделена от немедленной обработки[[3]](https://gameprogrammingpatterns.com/event-queue.html#:~:text=Decouple%20when%20a%20message%20or,from%20when%20it%20is%20processed) – это важно для стабильности симуляции (например, чтобы экономика могла за один тик произвести ресурсы без немедленного расходования этими же системами до завершения расчёта цикла). В конце тика Application вызывает EventBus.Flush(), и все события обрабатываются в определённом порядке. Кроме событий, существует система Command для запросов на действия: команды от игрока (построить здание, переместить войска) или высокоуровневые команды AI. Они проходят через валидацию (например, проверка ресурсов) и затем вызывают соответствующие методы Domain. Реализация: EventBus – синглтон (или передаётся через зависимости), имеющий методы Publish(event) и Subscribe(eventType, handler). Команды представлены либо объектами (паттерн Command), либо простыми вызовами сервисов Domain (например, UnitService.OrderMove(unitId, targetPos)).

Примечания производительности: События – легковесные объекты (часто struct), их генерация не должна создавать мусора. Желательно использовать object pool для повторно возникающих событий (например, deaths, hits). Отлаживать EventBus нужно на предмет потенциальных массовых рассылок: например, убийство 100 юнитов за тик приведёт к 100 событиям UnitDied – нужно убедиться, что обработчики таких событий работают быстро и не вызывают каскада дорогостоящих операций. Также важно избегать «подписок-призраков» – отписывать слушатели при удалении объектов, иначе возможны утечки памяти или NullRef. В контексте упрощённой симуляции (см. ниже) EventBus также выступает связующим звеном: события, произошедшие вне поля зрения, могут ставиться в очередь и обрабатываться, когда игрок их «раскроет» (например, событие SectorInfected могло произойти в фоновой симуляции – UI получит его сразу, чтобы обновить стратегическую карту, даже если игрок не видел сам процесс заражения).

### Сохранение игры

Назначение: Позволить сохранять и загружать состояние мира, включая все модули. Структура сохранения: JSON-файл, содержащий версии и разделы: экономика (запасы ресурсов, состояние зданий, прогресс строительства), население, инфекция (статистика SIR по секторам), позиции юнитов и их состояния, фортификации, состояния AI (возможно, только минимум – цели AI, т.к. планы можно перестроить при загрузке), а также данные упрощённой симуляции (агрегированные модели секторов вне обзора). Реализация: Слой Data имеет класс SaveManager с методами SaveGame(file) и LoadGame(file). Сохранение запрашивается через EventBus (например, при выходе или по таймеру автосейва). Иммутабельность конфигов: важно, что конфигурационные ScriptableObject не сериализуются в сейв – их загружают из ассетов, а сейв хранит только изменяющееся состояние. Перед сохранением система убеждается, что все данные приведены к сериализуемым формам (например, словари заменены на списки пар, ссылки на объекты – на идентификаторы)[[25]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=1%29%20,%D0%BA%D0%BE%D1%80%D1%80%D0%B5%D0%BA%D1%82%D0%BD%D0%BE%20%D0%B2%D0%BE%D1%81%D1%81%D1%82%D0%B0%D0%BD%D0%B0%D0%B2%D0%BB%D0%B8%D0%B2%D0%B0%D0%B5%D1%82%20%D0%BF%D0%BE%D0%B4%D1%82%D0%B8%D0%BF%D1%8B). Версионирование: в JSON включается поле версии, при загрузке SaveManager выполняет миграцию старых версий (через переключатель или апдейтер). Пример содержимого: раздел “population” с численностью и приростом[[26]](file://file_0000000049b062438e06b107048784b7#:~:text=%7D%2C%20,60), “buildings” – список строений с их прогрессом[[27]](file://file_0000000049b062438e06b107048784b7#:~:text=,), “logistics” – узлы и ссылки (для восстановления графа путей)[[28]](file://file_0000000049b062438e06b107048784b7#:~:text=,6%7D%5D).

Примечания производительности и надежности: Сохранение может быть объёмным, поэтому выполняется либо в отдельном потоке (через Async I/O) чтобы не мешать кадру, либо разбивается на части (например, сначала собрать JSON в памяти, а запись на диск – асинхронно). На платформе с 32 ГБ RAM объём памяти не критичен, но важно не допустить утечки – после сохранения чистить временные структуры. Автотесты для сохранений: сравнение состояния до сохранения и после загрузки (глубокое сравнение основных подсистем), тесты на совместимость версий (эмуляция загрузки старых сейвов), а также нагрузочные тесты (сохранение в стресс-тесте, например, на максимальном количестве объектов, чтобы проверить время выполнения и корректность).

### Пользовательский интерфейс (UI)

Назначение: Обеспечивает управление и отображение информации игроку. Включает: стратегический экран/“фронтовую” карту, показывающий целиком ситуацию по секторам (под контролем игрока, врага, уровень заражения и т.п.); экраны экономики – отчёты о дневном балансе ресурсов, узких местах, настройках передачи ресурсов ИИ[[9]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20,%D0%B2%20%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%20%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8%2C%20%D0%BE%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C%20%D0%B7%D0%B0%D0%B4%D0%B0%D1%87); панели строительства и логистики – где игрок выдаёт приказы переброски (выбирает узел отправки и прибытия, приоритет)[[10]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%92%D0%B2%D0%B5%D1%81%D1%82%D0%B8%20,%D1%81%20%D0%BF%D0%BE%D0%B2%D1%8B%D1%88%D0%B5%D0%BD%D0%BD%D1%8B%D0%BC%20%D1%80%D0%B0%D1%81%D1%85%D0%BE%D0%B4%D0%BE%D0%BC%20%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D0%BE%D0%B2), заказывает строительство зданий или фортификаций (с отображением требуемых ресурсов и времени). Боевой UI: панель отрядов, здоровье юнитов, мини-карта. UI реализован на Canvas + Unity UI Toolkit (при необходимости для сложных элементов). UI элементы взаимодействуют с Application через события/команды: например, нажатие кнопки “Построить ферму” вызывает команду строительства, а результат (появление здания) приходит как событие, на которое UI реагирует (обновляет список построек). Реализация особенностей: фронтовая карта получает данные от Domain о каждом секторе (с периодическим опросом или через события типа SectorStatsUpdated). Она не отображает реальных моделей юнитов вне обзора, вместо этого – значки или агрегированные индикаторы (например, “отряд врага (~50 ед.)” без детализации). UI также предоставляет отладочные панели (в Dev-режиме): например, график производительности по тикам, режим слоёв отрисовки (экономика, инфекция, пути) для тестирования.

Примечания производительности: UI должен обновляться с использованием unscaled time для анимаций, чтобы пауза игры не останавливала анимации интерфейса[[29]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,%D1%81%D1%87%D1%91%D1%82%D1%87%D0%B8%D0%BA%20%C2%AB%D0%BA%D0%B0%D0%B4%D1%80%D0%BE%D0%B2%20UI%C2%BB%20%E2%80%94%20%D0%B8%D0%B4%D1%91%D1%82). На высокой нагрузке убедиться, что обновление UI не потребляет много CPU: использовать батчи рендеринга, избегать часто изменяемых Layout Group при больших списках (например, список юнитов лучше виртуализировать или обновлять диффом). Для профилирования: Unity Profiler (категории UI) и UI Toolkit Debugger помогут найти узкие места. UI не должна зависеть от частого опроса Domain – лучше пуш-модель (событие изменилось -> UI обновился). При тестировании важно проверить, что UI корректно реагирует на события даже из фоновой симуляции (например, если база уничтожена вне экрана, значок на фронтовой карте меняется).

### Производительность и оптимизация

Общие подходы: Проект изначально проектируется с расчётом на высокую нагрузку (масштабные сражения, множество агентов инфекции, обширная экономика) на целевом железе (12-ядерный CPU, 32 GB RAM, 12 GB VRAM). Поэтому применены следующие решения: - Фиксированный тик симуляции: Используется собственный цикл с аккумулированием времени или Unity ECS FixedStepSimulationSystemGroup[[11]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,%D0%9F%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D1%8F%D0%B9%2C%20%D1%87%D1%82%D0%BE%20%D0%BD%D0%B8%D0%B3%D0%B4%D0%B5), что обеспечивает детерминированность шага и стабильность темпа симуляции (независимо от просадок FPS). Это упрощает отладку и делает результаты повторяемыми (важно для возврата с упрощённой симуляции – см. далее). - Разделение на активные/пассивные сектора: (подробно в следующем разделе) Ограничивает количество одновременно полносимулируемых объектов. Это главный приём оптимизации, позволяющий в большой игре не обрабатывать каждый объект каждый кадр. - Многопоточность: Критические по производительности системы (pathfinding, расчёт инфекции, может быть боевая логика) работают через C# Jobs, позволяя задействовать все ядра CPU. Burst-компиляция ускоряет математику в этих Jobs. Также, Background WorkerThreads могут использоваться для несрочных задач – сохранение на диск, генерация больших текстур карт и т.п. - Object Pooling: Активно используется пул объектов для юнитов, снарядов, эффектов – чтобы при массовом создании/уничтожении (например, взрыв породил десятки частиц или одноразовых объектов) не вызывались лишние аллокации. MonoBehaviour-объекты для юнитов берутся из пула при входе сектора в видимость, и возвращаются в пул при выгрузке. - LOD для графики и логики: Графические LOD – при отдалении камеры юниты могут отображаться упрощённо (меньше детализации спрайта или отключение эффектов). Логические LOD – собственно, упрощённая симуляция вне обзора (пониженная частота тиков, агрегированные расчёты)[[30]](https://gamedev.stackexchange.com/questions/3090/elegant-way-to-simulate-large-amounts-of-entities-within-a-game-world#:~:text=The%20solution%20was%20to%20use,update%20to%20detailed%2C%20regular%20updates). Также, для очень далёких областей можно вообще "заморозить" симуляцию до возвращения внимания, если это не критично для геймплея. - Память: 32 ГБ RAM достаточно, но следим за утечками. Особое внимание – на крупные структуры: граф путей, массивы клеток инфекции, историю событий для сейва. Используем Addressables или подгрузку по требованию для редких данных (например, спрайты юнитов могут подгружаться пакетом при начале боя и выгружаться после, чтобы не висеть в VRAM если бой далеко). - Профилирование: Регулярно проводятся профили в сценариях worst-case (максимум юнитов на экране, вспышка инфекции на всю карту, одновременные бои). Используется Unity Profiler (CPU, Rendering, Memory модули) и Profile Analyzers для сравнения результатов до/после оптимизаций. Для удобства, встроены маркеры профилировщика вокруг крупных операций (например, Job расчёта инфекции помечен Begin/EndSample, чтобы увидеть время). - Автотесты производительности: Пишутся на основе PerformanceTest Framework (если доступно) – сценарии: 1000 тиков симуляции с максимальной нагрузкой, замер среднего времени тика и отклонений. Цель – убедиться, что средний тик укладывается, например, в 16мс (60 FPS) при целевой нагрузке. Также тестируются переходы сектора в/из видимости на предмет отсутствия фризов.

## Упрощённая симуляция «жизни» вне обзора игрока

Задача: Реализовать механизм масштабирования симуляции мира, при котором полная детализация рассчитывается только в зоне, близкой к игроку, а на удалённых участках – упрощённая, агрегированная симуляция. Это позволит поддерживать ощущение живого мира (экономика, бои, инфекция продолжаются повсеместно), не расходуя ресурсы на тщательную покадровую симуляцию всего мира.

### Разделение мира на сектора (ячейки карты)

Игровая карта делится на крупные сектора (квадраты) фиксированного размера – например, 256x256 тайлов каждый (конкретный размер зависит от масштаба мира). Сектора образуют сетку. Активный сектор – сектор, в котором находится камера игрока (поле зрения). Соседние сектора – все клетки, смежные по стороне или углу с активным (то есть 8 соседей). Эти области считаются зоной повышенной детализации. Все сектора, находящиеся дальше одной клетки от поля зрения, считаются дальними (off-screen) и обрабатываются упрощённо. Таким образом, в каждый момент максимум 9 секторов (активный + окружение) работают в полном режиме, а остальные – в агрегированном.

Полная симуляция (FullSim): В активных и соседних секторах игра работает на полную: каждый кадр/тик обновляются все юниты, физика, ИИ, анимации и т.д. Это гарантирует, что перед глазами игрока все происходит с полной достоверностью.

Упрощённая симуляция (SimplifiedSim): В далеких секторах время течёт более крупными шагами и без деталей: - Сниженная частота тиков: вместо, скажем, 60 тиков в секунду, дальний сектор может обновляться 1 раз в секунду (или реже, настраивается). Проще говоря, мы имитируем крупные шаги времени – за один упрощённый тик может пройти сразу несколько секунд/минут игрового времени. - Агрегированная боевая модель: Вместо покадрового просчёта каждого выстрела и перемещения, бои рассчитываются статистически. Например, если в секторе присутствуют две враждующие группировки, каждый упрощённый тик мы вычисляем потери с каждой стороны на основе суммарной мощи и численности. Используются формулы вроде законов Ланчестера или упрощённые вероятностные модели: столкновение статов → потери/эффекты. Например, сравниваем соотношение сил A и B, генерируем случайную вариацию (с фиксированным сидом для детерминизма) – результат: у A погибло X юнитов, у B – Y юнитов[[30]](https://gamedev.stackexchange.com/questions/3090/elegant-way-to-simulate-large-amounts-of-entities-within-a-game-world#:~:text=The%20solution%20was%20to%20use,update%20to%20detailed%2C%20regular%20updates). Таким образом, длительное сражение “схлопывается” в серии резких шагов, но сохраняет примерно тот же исход, что и подробная симуляция. - Производство и строительство вне экрана: продолжается по цифровой модели. Здания в далеком секторе все еще производят ресурсы, но без создания игровых объектов – просто увеличивают счётчики ресурсов сектора раз в упрощённый тик. Строительство происходит с учетом времени: например, ферма, требующая 3 дня, в упрощённой симуляции просто имеет таймер – каждые N упрощённых тиков уменьшаем оставшееся время строительства, и по истечении – помечаем здание построенным (даже если не наблюдали процесс анимации). Ничего не отрисовывается: нет спавна рабочих юнитов, моделируется только конечный результат (ресурсы списаны, через 3 дня объект добавлен). - Логистика: перемещения юнитов между дальними секторами рассчитываются по времени фактически телепортацией на крупные расстояния. Например, если приказано перебросить армию через 3 сектора, в упрощённой симуляции можно сразу задать, что она прибудет через 5 минут (сохраняя таймер). Юниты не будут покадрово перемещаться – их позиция обновится при входе в ближнюю зону или по завершении таймера. - Инфекция: в дальних секторах моделируется на уровне суммарных цифр населения. Например, вместо заражения каждого человека, мы используем уравнения SIR с большим шагом – сразу рассчитываем, что за час заразилось 30 новых, 5 умерло, 10 выздоровело. Пространственное распространение в упрощёненном режиме тоже может быть огрублено: например, заражение перекидывается на соседний сектор только раз в сутки и на основе среднего уровня заражённости. Таким образом, инфекция “ползет” не непрерывно, а скачкообразно, но с сохранением общего тренда.

Данные сектора вне обзора: Для каждого неактивного сектора хранится агрегированная модель состояния. Это может быть структура SectorState с полями: - troopCounts – словарь {фракция: примерно сколько юнитов}, - resourceStockpiles – ресурсы, накопленные в секторе, - productionRates – скорости производства ресурсов, - infectionLevel – процент/стадия заражения населения, - buildingsInProgress – список строек с оставшимся временем, - ongoingBattles – упрощённое представление боёв (например, параметры противоборствующих сторон), - и т.д.

Эти данные обновляются дискретно каждый упрощённый тик. Важно: Каждая такая симуляция должна быть детерминированной при одних и тех же начальных условиях. Это значит, что использование случайности должно основываться на сохранённом seed (например, привязанном к координатам сектора и глобальному времени), чтобы при повторном расчёте результаты не “скакали”. В SectorState можно хранить randomSeed для генератора случайных событий (вспышка инфекции, результат боя), чтобы после загрузки игры или повторного входа в сектор получить те же результаты.

### Алгоритм перехода от упрощённой к полной симуляции

Когда сектор, ранее находившийся в упрощённой симуляции, попадает в поле зрения игрока (или в соседний с активным), необходимо плавно переключиться на полную симуляцию, не создавая нелогичных скачков. Алгоритм перехода: 1. Остановка упрощённого тика: как только сектор должен активироваться, его фоновые обновления приостанавливаются. Например, мы видим, что камера подошла к границе – соседний сектор будет в обзоре через секунду; система заранее останавливает дальнейшие агрегированные обновления этого сектора, зафиксировав его SectorState. 2. Генерация конкретного состояния: на основе агрегированных данных создаётся детализированная сцена. Это включает: - Спавн юнитов: Для каждой группы в troopCounts создаются реальные объекты юнитов (MonoBehaviour или ECS-entity). Их количество округляется до ближайшего целого (если агрегат хранил нецелые). Расположение: генерируется разумно – например, случайно по сектору, но с сохранением общей группировки (можно использовать алгоритм раскладки: войска обороны – вокруг базы, атакующие – на границе сектора, и т.п., исходя из контекста ongoingBattles). Уровень здоровья юнитов может быть снижен согласно потерям, понесённым в агрегатном бою. - События боя: Если ongoingBattles в секторе указывал, что шёл бой, то при активации можно создать ситуацию “две армии уже вступили в бой”. Например, разместить их на небольшом расстоянии друг от друга и сразу инициировать бой между ними. Таким образом, игрок, придя, увидит продолжающееся сражение, которое соответствует тому, что фоново считалось (с уже понесёнными потерями). - Обновление построек: Все строения, которые были построены или разрушены в упрощённом режиме, синхронно создаются или удаляются. Если строительство было частично завершено, в полном режиме мы можем создать незавершенное здание с соответствующим процентом прочности или стадии (например, недостроенный каркас). В Unity это может быть реализовано через несколько стадий префаба (фундамент, полуготовое, готовое). - Инфекция и население: Создаются объекты населения/гражданских, если они симулируются явно (возможно нет, если они абстрактны). Уровень заражения сектора переносится на отдельные юниты: например, если было 50% заражено, можно считать, что половина гражданских появляются как инфицированные NPC (или сам сектор имеет параметр “заражённость” для графических эффектов). Также применять эффекты – если в агрегате была вспышка (например, +100 заражённых за время вне экрана), при активации можно запустить соответствующую анимацию или просто отразить численно. - Логистика и транспорт: Все отложенные прибытия войск в сектор проверяются. Если по таймеру должно было прийти подкрепление через 2 минуты, то в полном режиме можно либо сразу создать их на краю карты с приказом идти в пункт назначения (если как раз “сейчас” время прибытия), либо запустить таймер в реальном времени на эти 2 минуты и по истечении – заспавнить (имитируя, что они пришли из-за края карты). 3. Синхронизация времени: убедиться, что локальное время сектора скорректировано. Например, если упрощённый тик сектора обновлялся раз в час, и сектор не был в фокусе 5 часов, то при активации мы “догнали” время в агрегате. Далее сектор переходит на мелкие тики, синхронные с глобальным игровым временем. Возможен короткий переходный период: первый секунды после активации могут быть потрачены на интерполяцию состояний. Например, если в агрегированном состоянии солдаты “должны” были находиться уже у середины сектора, мы можем при спавне сразу поместить их ближе к цели, чтобы не было рассинхрона во времени пути. 4. Рассылка событий и уведомлений: когда сектор становится активным, могут генерироваться события: SectorBecameVisible, DetailedSimStart(sectorId) – это сигнал для других систем. Например, EventBus уведомляет UI/Map, что теперь сектор в фокусе – можно, например, показать детальные иконки на миникарте вместо абстрактного значка. Также уведомляется AI: если ранее AI видел сектор агрегировано, теперь у него есть точные данные – AI может пересчитать планы с учётом точного состава сил.

Обратный процесс – деактивация сектора (уход из обзора) – тоже обрабатывается: 1. Все актуальные объекты (юниты, эффекты) сектора удаляются из сцены или отключаются и возвращаются в пул, чтобы не потреблять ресурсы. 2. Их состояние сворачивается обратно в SectorState. Например, если в момент ухода в секторе осталось 10 юнитов A и 5 юнитов B воюющих, мы заносим эти цифры в агрегат, помечаем бой продолжающимся. 3. Желательно сохранить достаточную информацию, чтобы при последующем возвращении воспроизвести по возможности ту же ситуацию. Например, фиксируем, что бой идет уже X минут, что такие-то здания горят и будут разрушены через Y минут если не починить, и т.п. 4. Запускается упрощённый симулятор для этого сектора (с учётом нового состояния) – либо сразу, либо со следующего условного “крупного” шага.

### Ограничения и детали реализации

Иммутабельность и консистентность: При переключении туды/сюды важно избегать «рассыпания» симуляции. Это означает, что состояние мира должно остаться консистентным и не зависеть от того, смотрел ли на него игрок. Мы обеспечиваем это детерминизмом и хранением полного состояния: - Immutable модели: Агрегированное состояние сектора (SectorState) можно хранить как отдельную структуру, которая не изменяется из других систем напрямую, а только через специальный симулятор. Например, SectorState может быть реализован как иммутабельный снимок, который при каждом упрощённом тике заменяется новым (чтобы не было половинчатого обновления). Это упрощает откат или проверку – в любой момент SectorState либо до, либо после тик-шага, но не в процессе. - Граница кэширования: Нужно чётко определить границы, где данные должны храниться между режимами. Например, при деактивации сектора – сохраняем последние подробные объекты в агрегат. При активации – должны быть готовы мгновенно породить детали. Чтобы не делать это синхронно в один кадр (что может вызвать фриз), можно подготовить кэш перехода: за несколько секунд до входа в обзор начать создавать объекты в неактивном состоянии. Unity позволяет создать объекты вне экрана (деактивированными) заранее. Например, если игрок быстро скроллит, мы можем предвосхитить, что соседний сектор возможно понадобится, и подгрузить его содержимое заранее (заготовить пул юнитов нужного типа, прогрузить текстуры). Граница кэширования может быть как раз соседние сектора: они всегда подгружены наперёд. Дальние же не имеют объектов, только данные. - Триггеры событий: События, происходящие в упрощённой симуляции, должны учитываться. Однако, часть событий имеет смысл генерировать только при детализации. Например, событие UnitDied – в далеком секторе не будет вызываться для каждой смерти (их там может быть десятки за шаг), вместо этого можно генерировать агрегированное событие BattleResults с итоговыми потерями. Но если игрок потом входит и видит тела – ему не нужны ретроспективные события о каждой смерти, достаточно того, что бой был. Поэтому некоторые события «приглушаются» вне обзора. С другой стороны, глобально важные – всё равно генерируются: напр. уничтожение базы игрока в off-screen секторе сразу вызывает PlayerBaseLost событие, даже если игрок этого не видел непосредственно. - Сохранение состояния: В файле сохранения храним все SectorState для упрощённых секторов. Также храним, какие секторы активны. При загрузке, те что неактивны, запускаются сразу в упрощённом режиме (можно даже не создавать их объекты, а просто планировать события, например, если по сохранению через 2 часа должен пасть сектор – после загрузки запланировать это через (2 часа - уже прошедшее время)). Сохранение/загрузка должны быть аккуратны: например, сохранить randomSeed секторов, чтобы продолжить ту же детерминированную последовательность.

### Влияние на EventBus, AI и карту фронта

EventBus: нужно учитывать, что события из фоновой симуляции могут поступать реже и более агрегировано. Реализация: может быть два канала событий – локальные события (для конкретных секторов, детальные) и глобальные события (агрегированные). Например, вместо множества UnitDied в секторе X (который off-screen) отправляется одно событие SectorBattleUpdate(X, потериA, потериB). Подписчики AI или UI, которые отслеживают ход войны, обрабатывают его, обновляя состояние фронтовой карты. Когда сектор становится видимым, EventBus может переключить тип событий: теперь снова идут индивидуальные UnitDied, BuildingComplete и т.д. Для подписчиков это прозрачно, если они изначально умеют обрабатывать оба режима. Важный момент – не задваивать события: если база была разрушена вне экрана и это отразилось в событии SectorLost, то при входе в сектор мы не должны снова генерировать BuildingDestroyed (уже было учтено). Нужно либо помечать такие события как уже обработанные, либо подавлять их генерацию при переходе.

Система сохранений: При сохранении игры в середине упрощённой симуляции все агрегированные данные уже находятся в SectorState и будут записаны. Но нужно также сохранить, что, например, в секторе шёл бой с такими-то участниками. Чтобы при загрузке и немедленном открытии сектора не получить “статичных” мирных юнитов вместо ожидающегося боя. То есть, часть агрегированной динамики должна сохраниться: либо в виде событий (в сейве хранить «pending events» типа “бой в секторе N продолжается”), либо просто через состояние (например, флаг sectorState.isBattleOngoing = true, и списки участников). Это позволит правильно инициализировать полный режим после загрузки.

AI-планирование: AI должен уметь принимать решения, опираясь на неполные данные. Обычно AI «знает» о враге только то, что видит игрок (fog of war). Но если AI контролирует дальние сектора (например, вражеская база вне обзора), то для него эти сектора всё равно симулируются, просто с пониженной точностью. Стратегический AI будет работать с агрегированными данными: например, решает атаковать сектор игрока, зная, что у него там ~50 солдат (агрегировано). Он может отправить 60 своих – и эта отправка происходит через упрощённую логистику (например, отметится, что через 10 минут прибудет армия). AI также получает информацию с той же периодичностью: если в off-screen секторе произошли изменения (потери, строительство турелей), AI узнает об этом через глобальные события или через прямой доступ к SectorState (если считаем, что ИИ “всеведущ” для своих территорий). Главное – планы AI должны быть стабильны: если AI решил что-то, уход игрока не должен сбросить планы. К примеру, AI осадил город игрока, игрок ушёл – бой продолжился упрощённо. AI не должен внезапно передумать, получив слегка другие цифры – для этого можно вводить гистерезис: AI решения не пересчитывает каждую секунду в off-screen, а держится курса, пока не произойдёт крупное событие. На тактическом уровне, когда сектор активен, AI переходит к детальному управлению (исполняет Behavior Tree для каждого юнита). При деактивации AI может освобождать тактические контроллеры и вернуть управление стратегическому симулятору.

“Фронтовая” карта: Эта стратегическая карта напрямую питается от упрощённой симуляции. Каждому сектору на карте соответствуют агрегированные данные, и цвет/значки на карте обновляются соответственно: - Если сектор вне обзора, карта показывает его состояние из SectorState: напр., численность сил врага (можно округленно, “??” если информация неизвестна игроку), уровень заражения (например, индикатор 30%), наличие боёв (иконка перекрещенных мечей, если идет бой). - Если сектор в поле зрения, карта может показать более точную информацию или отметить, что игрок “наблюдает” там. Переход сектора в полный режим должен плавно отразиться: например, значок “бой” может замениться конкретными юнитами на тактическом мини-экране. Важно: фронтовая карта – инструмент игрока, поэтому она должна быть согласованной с реальностью. Упрощённая симуляция обеспечивает, что даже если игрок не смотрит, на карте все равно будет отражаться приближение фронта (например, красная зона инфекции распространяется). При возврате в сектор никаких ретконавтов: состояние на карте соответствует тому, что игрок увидит глазами.

### Стабильность и детерминированность симуляции

Чтобы мир вел себя правдоподобно, необходимо, чтобы при повторном входе в сектор состояние не “случайно” отличалось. Для этого: - Все генераторы случайности в упрощённой симуляции используют фиксированные seeds, как упомянуто. Можно даже привязать seed к глобальному времени: напр., каждый день для боёв генерируется одна случайная серия, чтобы при сохранении/загрузке в середине боя результаты не менялись. - Агрегированные формулы отлаживаются против полного симулятора: проводится тест, когда тот же сценарий прогоняется двумя способами – (A) полностью, (B) выключив сектор на час и вернув. Результаты (потери, оставшиеся ресурсы) должны совпадать или быть очень близкими. Если расхождения сильны, корректируем модели. Возможно, придется хранить дополнительные данные, чтобы при детализации скорректировать. Например, упрощённый бой не учитывал положение юнитов, и при активации они начнут стрелять сразу – это может удваивать потери по сравнению с расчетом. Решение: при активации временно уменьшить меткость или дать им время на сближение, чтобы компенсировать. - Идемпотентность переходов: многократное включение/выключение сектора за короткое время не должно сильно влиять на результат. Если игрок дергает камеру туда-сюда, мы должны предотвращать эксплойты (например, “пока не смотрю – мои войска не умирают” или наоборот). Для этого можно вводить задержку на переключение: сектор считается активным, если он был в зоне обзора более X секунд, и деактивируется с небольшим лагом. Это сгладит частые переключения. Также, при быстром уходе и возврате, можно частично сохранить детальные объекты (не уничтожать их мгновенно, а через некоторое время) – чтобы если вернулись, не создавать заново. - Согласованность времени: фиксированный глобальный тик помогает: даже если сектор обновляется раз в секунду, он делает это по меткам глобального времени. Значит, если игрок вернулся ровно на 12:00, а на 11:59 там был последний апдейт, то между 11:59 и 12:00 может быть короткий “догоняющий” период, но потом полная симуляция идет синхронно с остальными. - Тестирование: Понадобятся тщательные тесты: сценарии, где игрок покидает бой и возвращается к его окончанию – убедиться, что исход логичен и воспроизводим. Тесты, где строится здание: начать стройку, уйти, вернуться после предполагаемого завершения – здание должно стоять готовое. На автоматическом уровне можно скриптом: сравнить мир, эмулированный полностью 10 минут, и мир, где 5 мин был off-screen, 5 мин on-screen – разницы нет в ключевых метриках.

### Профилирование и автотесты поведения упрощённой симуляции

Профилирование: Необходимо измерять выгоду и накладные расходы введения SimplifiedSim. С помощью Unity Profiler оцениваем: - Время обновления 1 сектора FullSim vs десятка секторов SimplifiedSim в сумме. Ожидание: упрощённый на порядки дешевле. Если выяснится, что агрегированные расчёты тоже тяжелы (например, огромные формулы или много аллокаций), их оптимизируем (кэшируем результаты, упрощаем модель). - Спайки при переходе: включаем Deep Profile или инструмент измерения времени при активации сектора. Проверяем, нет ли больших фризов при спавне объектов. Если есть, внедряем поэтапную подгрузку: как говорилось, создавать объекты постепенно. Можно распределить создание юнитов на несколько кадров (например, 100 юнитов спавнить по 20 в кадр). Также использовать профилировщик памяти – убедиться, что переход не приводит к неожиданному всплеску (например, повторная загрузка тех же текстур – это нужно предотвратить, держа их в памяти). - Тест производительности на масштабируемость: увеличить размер активной зоны (например, 5х5 секторов full) и проверить, держит ли движок нагрузку. Цель – выявить, что даже без упрощения всё ок на небольших областях, а с упрощением – линейное масштабирование по количеству секторов.

Автотесты и QA: Для такого механизма будут написаны специальные сценарии тестирования: - Consistency Test: как описано, сравнение полного vs смешанного симулирования. Это можно автоматизировать: запустить игру в режиме “всё активно” на X минут, сохранить state. Потом перезапустить, но отключать сектор (телепортировать камеру прочь) на некоторое время, потом возвращать – и снова сохранить state. Сравнить ключевые показатели (ресурсы, кто контролирует сектор, численность юнитов, уровень инфекции). Разрешены небольшие отклонения, но не драматические. Эти тесты должны проходить перед релизом каждого баланса изменения. - Edge Case Test: ситуации, когда на границе видимости происходят события. Например, юнит пересекает границу активного сектора: он не должен исчезнуть или подвиснуть. Пишем тест: отправить юнита из видимого сектора в соседний невидимый и обратно – убедиться, что при выходе он корректно передан в агрегат (возможно, мы мгновенно переключаем его под управление SimplifiedSim), а при возвращении – вновь появился. - Determinism Test: запускаем один и тот же сценарий несколько раз (с фиксированным seed) и проверяем, что статистика по окончании совпадает. Особенно касается боёв: 10 авто-битв оффскрин с одних условий дают один и тот же результат. - Performance Test: скрипт, который перемещает камеру по маршруту, охватывающему множество секторов, в течении, например, часа игрового времени. В конце смотрим лог профилировщика или метрики: средний FPS, максимальные задержки на переходах. Эти тесты могут запускаться на разных конфигурациях (PC, target hardware) чтобы убедиться, что нет проблем. В частности, на заявленном железе 5900X + 32GB + 12GB GPU ожидается, что фриз при самом тяжёлом переключении (например, город с 200 юнитами становится видимым) не превышает допустимого (например, <50 мс). Если больше – оптимизировать (предзагрузка, уменьшение объектов и т.д.).

В итоге, данный механизм упрощённой симуляции интегрирован во все слои архитектуры (Domain: логика LOD, Data: хранение состояния, Application: переключение режимов, AI: адаптация к агрегатам, Presentation: стратегическая карта и отсутствие визуала оффскрин, Infrastructure: много потоков для бэкграунда). Он не отключаем – это фундаментальный элемент оптимизации под масштаб. На целевой платформе система обеспечивает правдоподобную активность мира и одновременно высокую производительность, создавая иллюзию, что “жизнь продолжается” повсюду, хотя на самом деле вычислительные ресурсы тратятся только там, где это нужно[[30]](https://gamedev.stackexchange.com/questions/3090/elegant-way-to-simulate-large-amounts-of-entities-within-a-game-world#:~:text=The%20solution%20was%20to%20use,update%20to%20detailed%2C%20regular%20updates).

[[1]](file://file_0000000049b062438e06b107048784b7" \l ":~:text=1,%D0%B2%20%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9%20%D0%B4%D0%B5%D0%BD%D1%8C%20%D0%B2%20%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE) [[9]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20,%D0%B2%20%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%20%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8%2C%20%D0%BE%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C%20%D0%B7%D0%B0%D0%B4%D0%B0%D1%87) [[10]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%92%D0%B2%D0%B5%D1%81%D1%82%D0%B8%20,%D1%81%20%D0%BF%D0%BE%D0%B2%D1%8B%D1%88%D0%B5%D0%BD%D0%BD%D1%8B%D0%BC%20%D1%80%D0%B0%D1%81%D1%85%D0%BE%D0%B4%D0%BE%D0%BC%20%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D0%BE%D0%B2) [[12]](file://file_0000000049b062438e06b107048784b7#:~:text=%28%D1%83%D0%BD%D0%B8%D1%84%D0%B8%D1%86%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%29.%20,%C2%A79) [[13]](file://file_0000000049b062438e06b107048784b7#:~:text=%2A%2A%D1%88%D1%82%D0%B0%D1%82%D0%BD%D1%83%D1%8E%20%D0%BF%D0%BE%D1%82%D1%80%D0%B5%D0%B1%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%D0%B2%20%D1%82%D1%80%D1%83%D0%B4%D0%B0,%28%D1%81%D0%BC.%20%C2%A710) [[14]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%92%D0%B2%D0%B5%D1%81%D1%82%D0%B8%20,%D0%B2%20%D1%80%D0%B5%D0%B7%D0%B5%D1%80%D0%B2%2F%D0%B3%D0%B0%D1%80%D0%BD%D0%B8%D0%B7%D0%BE%D0%BD%20%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%20%D0%B7%D0%B0%D0%B2%D0%B5%D1%80%D1%88%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) [[15]](file://file_0000000049b062438e06b107048784b7#:~:text=4%29%20%D0%A2%D0%B8%D0%BF%D1%8B%20%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B9%20%28MUST%29%201,%D1%87%D0%B0%D1%81%D1%8B%20%D0%B8%20%D0%B5%D0%B4%D1%83%20%D0%BD%D0%B0%20%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B0%D0%BD%D0%B8%D0%B8) [[16]](file://file_0000000049b062438e06b107048784b7#:~:text=3,%28%D0%BF%D1%80%D0%B8%20%D0%BD%D0%B0%D0%BB%D0%B8%D1%87%D0%B8%D0%B8%20%D0%BD%D0%B5%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%BA%D0%B8%D1%85%20%D0%98%D0%98) [[23]](file://file_0000000049b062438e06b107048784b7#:~:text=1.%20%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B8%D1%82%D1%8C%20%D0%BD%D0%B0%20,%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F%20%D1%87%D0%B8%D1%81%D0%BB%D0%BE%D0%BC) [[24]](file://file_0000000049b062438e06b107048784b7#:~:text=3.%20%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B%20%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%B0%3A%20,%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%8F%D0%B5%D1%82%D1%81%D1%8F%20%D1%87%D0%B8%D1%81%D0%BB%D0%BE%D0%BC%20%D0%B1%D1%80%D0%B8%D0%B3%D0%B0%D0%B4%2F%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0) [[26]](file://file_0000000049b062438e06b107048784b7#:~:text=%7D%2C%20,60) [[27]](file://file_0000000049b062438e06b107048784b7#:~:text=,) [[28]](file://file_0000000049b062438e06b107048784b7#:~:text=,6%7D%5D) document-4.txt

<file://file_0000000049b062438e06b107048784b7>

[[2]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,%D0%BF%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC%D0%B0%20%E2%80%94%20%D0%BF%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D1%8C%20%D0%B2%20%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%BC) [[5]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,tick) [[6]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%BF%D0%BE%D0%BA%D0%B0%D0%B7%D1%8B%D0%B2%D0%B0%D0%B9%20%D0%B2%20%D1%83%D0%B3%D0%BB%D1%83%20UI,%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B8%D1%82%D1%8C%20VSync%2F%D0%B2%D1%8B%D0%BA%D0%BB%D1%8E%D1%87%D0%B8%D1%82%D1%8C%29%20%E2%80%94%20%D1%82%D0%B8%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F) [[7]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%98%D0%98%3A%20Utility%20AI%20%2B%20GOAP,Behavior%20Trees) [[8]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=match%20at%20L138%201,%EE%88%80cite%EE%88%82turn2search5%EE%88%81) [[11]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,%D0%9F%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D1%8F%D0%B9%2C%20%D1%87%D1%82%D0%BE%20%D0%BD%D0%B8%D0%B3%D0%B4%D0%B5) [[17]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%2A%2A%D0%9F%D1%80%D0%BE%D0%B2%D0%B5%D1%80%D0%BA%D0%B8%2F%D1%82%D0%B5%D1%81%D1%82%D1%8B%3A%2A%2A%201%29%20Unit,%C2%AB%D0%B4%D0%B5%D0%BD%D1%8C%D0%B3%D0%B8%20%D0%BD%D0%B5%20%D0%B8%D1%81%D1%87%D0%B5%D0%B7%D0%B0%D1%8E%D1%82%2F%D0%BD%D0%B5%20%D0%BF%D0%BE%D1%8F%D0%B2%D0%BB%D1%8F%D1%8E%D1%82%D1%81%D1%8F%C2%BB) [[18]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=) [[19]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%98%D0%BD%D1%84%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%20) [[21]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=%D0%98%D0%BD%D1%84%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%20) [[22]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=Pathfinding%20%D0%BD%D0%B0%20%D1%81%D0%B5%D1%82%D0%BA%D0%B5%20%28A,JPS%20%E2%86%92%20HPA) [[25]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=1%29%20,%D0%BA%D0%BE%D1%80%D1%80%D0%B5%D0%BA%D1%82%D0%BD%D0%BE%20%D0%B2%D0%BE%D1%81%D1%81%D1%82%D0%B0%D0%BD%D0%B0%D0%B2%D0%BB%D0%B8%D0%B2%D0%B0%D0%B5%D1%82%20%D0%BF%D0%BE%D0%B4%D1%82%D0%B8%D0%BF%D1%8B) [[29]](file://file_000000001f0461f4847b90af8dfa8700#:~:text=,%D1%81%D1%87%D1%91%D1%82%D1%87%D0%B8%D0%BA%20%C2%AB%D0%BA%D0%B0%D0%B4%D1%80%D0%BE%D0%B2%20UI%C2%BB%20%E2%80%94%20%D0%B8%D0%B4%D1%91%D1%82) document-3.txt

<file://file_000000001f0461f4847b90af8dfa8700>

[[3]](https://gameprogrammingpatterns.com/event-queue.html#:~:text=Decouple%20when%20a%20message%20or,from%20when%20it%20is%20processed) [[4]](https://gameprogrammingpatterns.com/event-queue.html#:~:text=Most%20games%20aren%E2%80%99t%20event,that%20want%20to%20stay%20decoupled) Event Queue · Decoupling Patterns · Game Programming Patterns

<https://gameprogrammingpatterns.com/event-queue.html>

[[20]](https://docs.unity3d.com/Packages/com.unity.physics@1.0/manual/design.html#:~:text=Design%20philosophy%20,using%20ECS%20best%20practices) Design philosophy | Unity Physics | 1.0.16 - Unity - Manual

<https://docs.unity3d.com/Packages/com.unity.physics@1.0/manual/design.html>

[[30]](https://gamedev.stackexchange.com/questions/3090/elegant-way-to-simulate-large-amounts-of-entities-within-a-game-world#:~:text=The%20solution%20was%20to%20use,update%20to%20detailed%2C%20regular%20updates) architecture - Elegant way to simulate large amounts of entities within a game world - Game Development Stack Exchange

<https://gamedev.stackexchange.com/questions/3090/elegant-way-to-simulate-large-amounts-of-entities-within-a-game-world>