# Стратегический отчет: Архитектурная миграция систем визуализации от Unity uGUI к CSS Layout Engine

## 1. Введение: Фундаментальный сдвиг парадигмы рендеринга

Переход квалифицированного инженера Unity в экосистему Enterprise Web Development требует не просто изучения нового синтаксиса, а глубокой реструктуризации ментальных моделей, касающихся управления экранным пространством и жизненным циклом рендеринга. Настоящий отчет представляет собой исчерпывающее теоретическое руководство к Четвертому дню интенсивного плана обучения, сфокусированное на деконструкции систем верстки веба через призму архитектуры игровых движков. Мы не будем ограничиваться поверхностными сравнениями; наша цель — проанализировать алгоритмическую сложность, управление памятью и потоки исполнения, лежащие в основе CSS Grid и Flexbox, сопоставляя их с механизмами Unity UI (uGUI) и UI Toolkit.1

В основе Unity лежит императивная модель **Immediate Mode** (в контексте OnGUI) или гибридная модель **Retained Mode** (в uGUI), где разработчик имеет прямой доступ к трансформам (RectTransform) и обязан явно управлять перестроением геометрии. Современный веб, в частности React в связке с CSS, представляет собой чистый **Retained Mode**, где разработчик описывает *желаемое состояние* (декларация), а браузерный движок (Blink, Gecko, WebKit) берет на себя роль «черного ящика», решающего систему ограничений (Constraint Solver) для вычисления финальной геометрии. Понимание того, как работает этот «черный ящик», критически важно для предотвращения проблем с производительностью, известных как Layout Thrashing, которые по своей природе идентичны дорогим вызовам Canvas.BuildBatch в Unity.2

Этот документ предназначен для сохранения в корпоративной базе знаний (Google Drive) и служит эталонным руководством по архитектуре фронтенда для специалистов с бэкграундом в компьютерной графике и C#.

## 2. Теория Пространства: Двумерные системы ограничений

Центральной задачей четвертого дня обучения является реализация сетки календаря. В Unity для решения подобных задач традиционно используется компонент GridLayoutGroup. В веб-разработке стандартом де-факто стал **CSS Grid Layout Module**. Несмотря на внешнее сходство — оба инструмента разбивают пространство на ячейки — их внутренняя логика и алгоритмическая база различаются кардинально.

### 2.1. Алгоритмическая природа распределения пространства

#### Unity GridLayoutGroup: Итеративная детерминированность

Компонент GridLayoutGroup в Unity представляет собой относительно примитивный контроллер раскладки. Его алгоритм работает за один проход и обладает линейной сложностью $O(n)$, где $n$ — количество дочерних элементов. Логика работы строится на жестких ограничениях:

1. Считываются глобальные параметры Cell Size (размер ячейки) и Spacing (отступы).
2. Вычисляется доступная ширина контейнера (RectTransform).
3. Определяется количество колонок на основе свойства Constraint (например, Fixed Column Count).
4. Производится итерация по списку дочерних трансформов, где каждому элементу принудительно назначаются координаты и размеры.4

Критическая особенность архитектуры Unity заключается в том, что GridLayoutGroup по умолчанию **игнорирует** внутренние предпочтения дочерних элементов. Если один из элементов списка содержит компонент LayoutElement с заданным preferredWidth, группа раскладки проигнорирует это значение в угоду глобальному Cell Size. Это архитектурное решение продиктовано необходимостью оптимизации: одинаковый размер ячеек позволяет эффективнее использовать батчинг (batching) и упрощает математику перестроения.4 Однако это создает жесткость: создать календарь, где выходные дни визуально шире будних, используя стандартный GridLayoutGroup, невозможно без написания кастомного контроллера раскладки.

#### CSS Grid: Решатель систем ограничений (Constraint Solver)

CSS Grid Layout — это гораздо более сложная система, способная решать системы линейных уравнений для распределения пространства. Браузерный движок выполняет многопроходный алгоритм **Track Sizing Algorithm**, который учитывает как внешние ограничения контейнера, так и внутренние потребности содержимого (Intrinsic Sizing).6

Алгоритм работы CSS Grid можно упрощенно представить следующим образом:

1. **Инициализация треков:** Все строки и колонки инициализируются с минимальными и максимальными функциями размера (например, minmax(auto, 1fr)).
2. **Разрешение внутреннего размера (Intrinsic Sizing):** Движок рекурсивно опрашивает содержимое ячеек. Это аналог вызова метода CalculateLayoutInputHorizontal в Unity, но в CSS это происходит для обоих измерений одновременно. Браузер вычисляет min-content (минимальный размер, при котором контент не переполняется) и max-content (размер контента без переносов строк).8
3. **Распределение свободного пространства:** Если контейнер имеет фиксированную ширину, вычисляется дельта свободного пространства.
4. **Разрешение гибких треков (fr):** Свободное пространство распределяется между треками, использующими единицу измерения fr (fraction), пропорционально их коэффициентам.10

Эта фундаментальная разница позволяет CSS Grid создавать адаптивные интерфейсы, которые в Unity потребовали бы сложной вложенности компонентов HorizontalLayoutGroup и VerticalLayoutGroup или написания сотен строк кода в Update().11

### 2.2. Единица fr против LayoutElement.flexibleWidth

В Unity механизм распределения "оставшегося" места реализуется через свойство flexibleWidth компонента LayoutElement. Это значение представляет собой весовой коэффициент. Если у трех элементов flexibleWidth равен 1, 2 и 1 соответственно, то доступное пространство будет разделено в пропорции 25% / 50% / 25%.12

Математически модель Unity для расчета ширины элемента выглядит так:

$$Width\_{child} = MinWidth + (FreeSpace \times \frac{FlexibleWidth\_{child}}{\sum FlexibleWidth\_{siblings}})$$

В CSS Grid единица fr работает схожим образом, но применяется не к самому элементу, а к определению трека (колонки или строки) в родительском контейнере.

$$TrackSize = BaseSize + (FreeSpace \times \frac{FlexFactor\_{track}}{\sum FlexFactors})$$

**Архитектурное различие в «Владении» (Ownership):**

* **Unity (Child-driven):** Дочерний объект «кричит» родителю: «Я хочу быть гибким!». Это реализуется через компонент LayoutElement на префабе. Логика размазана по иерархии.
* **CSS Grid (Parent-driven):** Родительский контейнер диктует: «У меня есть структура из 7 колонок, каждая занимает 1 долю пространства». Свойство grid-template-columns: repeat(7, 1fr) централизует логику раскладки в одном месте (в CSS классе контейнера).6

Для реализации календаря подход CSS Grid (Parent-driven) является архитектурно более чистым. Вместо того чтобы настраивать каждый DayCell префаб, вы определяете сетку один раз в компоненте MonthView. Это соответствует принципу «Single Source of Truth» (Единый источник истины).

| **Характеристика** | **Unity LayoutElement.flexibleWidth** | **CSS Grid fr unit** |
| --- | --- | --- |
| **Место определения** | На дочернем компоненте (Prefab) | На родительском контейнере (CSS Rule) |
| **Ось воздействия** | Только одна ось (зависит от Layout Group) | Любая ось (Grid Track definition) |
| **Базовый размер** | minWidth / preferredWidth | Функция minmax() или контент |
| **Сложность расчета** | Низкая (линейная интерполяция) | Высокая (решение системы ограничений) |

### 2.3. Имплицитные сетки (Implicit Grids) и Генерация контента

Unity-разработчики привыкли к **явному инстанцированию** (Explicit Instantiation). Чтобы создать календарь на 30 дней, необходимо запустить цикл for и создать 30 экземпляров GameObject. Если добавить 31-й элемент, GridLayoutGroup просто поместит его в следующую доступную позицию согласно правилу Start Axis.4

CSS Grid вводит мощное понятие **Имплицитной сетки (Implicit Grid)**. Вы можете определить явные колонки (grid-template-columns), но не определять строки. По мере добавления элементов в DOM (например, через .map() в React), Grid автоматически создает новые строки (треки) для размещения контента. Это поведение контролируется свойствами grid-auto-rows и grid-auto-flow.15

Аналогия с Unity: Это похоже на GridLayoutGroup с настройкой Constraint: Fixed Column Count, но с одним критическим отличием: в CSS вы можете задать паттерн размеров для этих автоматически создаваемых строк.

Например, правило grid-auto-rows: minmax(100px, auto) гарантирует, что любая новая неделя в календаре будет иметь высоту минимум 100 пикселей, но автоматически расширится, если список событий внутри дня превысит этот размер. В Unity достижение такого поведения («расширение по контенту») требует комбинации ContentSizeFitter и LayoutGroup — пары, известной своей нестабильностью и склонностью к созданию бесконечных циклов перестроения (rebuild loops), если не настроить привязки (Anchors) идеально точно.17

## 3. Одномерные потоки: Flexbox против Linear Layout Groups

В то время как CSS Grid управляет макро-структурой (сетка месяца), **Flexbox** идеально подходит для микро-структуры (содержимое ячейки дня: цифра даты, маркеры событий, иконки статуса). В терминологии Unity это прямые аналоги HorizontalLayoutGroup и VerticalLayoutGroup.

### 3.1. Главная и Поперечная оси (Main & Cross Axis)

Flexbox оперирует понятиями осей, направление которых зависит от свойства flex-direction.

* flex-direction: row концептуально эквивалентен Unity HorizontalLayoutGroup.
* flex-direction: column концептуально эквивалентен Unity VerticalLayoutGroup.19

Однако, механизмы выравнивания в CSS значительно превосходят возможности Unity. В Unity свойство Child Alignment (например, UpperCenter, MiddleRight) жестко связывает выравнивание по обеим осям в один Enum.20 CSS Flexbox декомпозирует это управление:

1. **justify-content (Главная ось):** Управляет распределением пространства вдоль потока. Значения вроде space-between (равномерное распределение с прижатием к краям) или space-around (распределение с равными отступами) не имеют прямых аналогов в Unity. Чтобы реализовать space-between в Unity, разработчику приходится вставлять невидимые «распорки» (Spacer GameObjects) с компонентом LayoutElement и flexibleWidth > 0 между видимыми элементами.21
2. **align-items (Поперечная ось):** Управляет выравниванием перпендикулярно потоку. Это аналог вертикальной составляющей Child Alignment в горизонтальной группе.

Для календаря это означает, что вы можете легко разместить число даты в левом верхнем углу, а список событий — в нижней части ячейки, используя flex-direction: column и justify-content: space-between, без создания сложной иерархии вложенных объектов.

### 3.2. Проблема вложенности и Layout Thrashing

Распространенным анти-паттерном в Unity является глубокая вложенность Layout Groups (Вертикальная группа, содержащая Горизонтальные, которые содержат Вертикальные). Каждый уровень вложенности экспоненциально увеличивает сложность пересчета макета. Когда лист иерархии (leaf node) помечается как «грязный» (dirty), система LayoutRebuilder должна пройти вверх по иерархии до корня, а затем пересчитать все вниз.18 Это одна из главных причин падения FPS в нагруженных UI интерфейсах Unity.5

В веб-разработке глубокая вложенность Flexbox-контейнеров также имеет свою цену, но современные браузерные движки (LayoutNG в Chrome) используют агрессивное кэширование макетов (Layout Caching) и границы изоляции (CSS Containment). Тем не менее, правило «уплощения иерархии» (Flatten the hierarchy) остается актуальным. Переход на CSS Grid для основного контейнера позволяет избежать создания десятков промежуточных div-оберток, которые были бы необходимы для симуляции сетки на Flexbox или вложенных Layout Groups в Unity.22

## 4. Инкапсуляция и Модульность: CSS Modules против Prefabs

Миграция с Unity на React требует переосмысления того, как инкапсулируются визуальные свойства компонентов. В Unity **Prefab** является единицей инкапсуляции, объединяющей структуру (GameObject hierarchy), логику (MonoBehaviour) и стиль (настройки в Inspector, такие как Color, Font, Sprite). Изменение префаба автоматически обновляет все его экземпляры, реализуя мощную модель наследования.24

В традиционной веб-разработке CSS имеет глобальную область видимости. Класс .button, определенный в компоненте календаря, может случайно изменить стиль кнопки в футере сайта. Это эквивалентно тому, как если бы все ваши сериализованные переменные в Unity были объявлены как public static, что неизбежно привело бы к загрязнению пространства имен и непредсказуемым коллизиям.

**CSS Modules** решают эту проблему, принудительно обеспечивая локальную область видимости (Local Scoping), что концептуально аналогично private сериализованным полям в Префабе.26

### 4.1. Механизм хеширования как аналог InstanceID

При использовании CSS Modules (например, файл DayCell.module.css), система сборки (Vite/Webpack) на этапе компиляции переименовывает классы, добавляя к ним уникальный хеш.

* **Исходный код:** .dayCell { background: red; }
* **Результат компиляции:** .DayCell\_dayCell\_\_x7z91 { background: red; }

Этот уникальный суффикс (x7z91) действует как **GUID** или **InstanceID** в Unity.28 Он гарантирует, что визуальные свойства, определенные для компонента DayCell, никогда не «протекут» в компонент MonthHeader, даже если разработчик использует одинаковые имена классов (например, .container или .text). Это создает герметичную капсулу стиля, превращая CSS класс в аналог **Prefab Variant**, где область действия изменений строго ограничена конкретным компонентом.

### 4.2. Композиция против Наследования (Composition over Inheritance)

Unity предлагает механизм **Prefab Variants**, позволяющий создавать иерархию наследования префабов (например, BaseEnemy -> FireEnemy), переопределяя конкретные свойства.24 В React и CSS Modules принята парадигма **Композиции**.

Вместо создания «Варианта» CSS файла, React-разработчик использует утилитарные классы или динамическое присвоение классов на основе пропсов (Props).

JavaScript

// React компонент  
<div className={`${styles.cell} ${props.isWeekend? styles.weekend : ''}`}>

В Unity ближайшим аналогом этого подхода является **Паттерн Декоратора** или добавление нескольких небольших MonoBehaviour на один GameObject для компоновки сложного поведения, вместо использования глубокой иерархии наследования классов. Такой подход позволяет визуальному состоянию быть чистой функцией от данных (UI = f(State)), что идеально согласуется с декларативной природой React.30

## 5. Физика Рендеринга: Pixel Pipeline и Оптимизация

Для разработчика игр производительность измеряется во времени кадра (**Frame Time**) и количестве вызовов отрисовки (**Draw Calls**). В веб-разработке метрики иные, но физический смысл процессов идентичен: **Layout Time** (верстка), **Paint Time** (отрисовка) и **Composite Time** (композитинг). Понимание корреляции между этими процессами в Unity и браузере — ключ к созданию высокопроизводительных интерфейсов без «фризов» (jank).

### 5.1. Система «Грязных флагов»: Reflow против Canvas Rebuild

#### Unity: The Canvas Rebuild

Когда свойство UI элемента в Unity изменяется (например, text.text = "Hello"), он помечает себя как «грязный» (SetDirty).

1. **Layout Rebuild:** Класс LayoutRebuilder пересчитывает размеры и позиции элемента и всех его родительских Layout Groups.
2. **Graphic Rebuild:** Компоненты Graphic (Text, Image) регенерируют свои меши (вершины, треугольники, UV-координаты).
3. **Batch Building:** Система Canvas объединяет эти меши в один или несколько батчей для отправки на GPU (Canvas.BuildBatch).2

Этот процесс является синхронным и выполняется в главном потоке (Main Thread). Если в кадре изменяется много элементов, время выполнения Canvas.BuildBatch резко возрастает, вызывая просадку FPS. Именно поэтому в Unity критически важно разделять статические и динамические элементы по разным Canvas (Split Canvases).32

#### Browser: Reflow (Layout)

В браузере аналогичный процесс называется **Reflow** (или Layout в терминах Chrome). Изменение геометрического свойства (такого как width, margin, font-size, grid-template-columns) инвалидирует макет элемента, а также, потенциально, его детей и родителей.34 Браузер обязан пересчитать Дерево Макета (Layout Tree) — геометрическую структуру страницы. Это прямой эквивалент вызова LayoutRebuilder.Rebuild() в Unity.

Однако, в отличие от Unity, где перестроение происходит надежно в конце кадра, современные браузеры пытаются быть «ленивыми» (lazy evaluation). Они ставят изменения в очередь и выполняют их пакетно перед отрисовкой следующего кадра.

### 5.2. Layout Thrashing: Синхронная катастрофа

Критическая ловушка производительности, существующая в обеих системах — это принудительный вызов пересчета макета в середине кадра.

* **Unity:** Вызов Canvas.ForceUpdateCanvases() заставляет систему немедленно перестроить геометрию UI. Если сделать это внутри цикла foreach, производительность упадет катастрофически.35
* **Browser:** Если ваш JavaScript код запрашивает геометрическое свойство (например, element.offsetHeight или getBoundingClientRect()) *сразу после* того, как он изменил стиль элемента, браузер вынужден прервать JS-исполнение и **немедленно** запустить синхронный Reflow, чтобы вернуть корректное значение.3

**Сценарий:** Автоматическое изменение размера ячеек календаря.

* *Плохой паттерн:* Записать текст в ячейку -> Прочитать её новую высоту -> Изменить высоту контейнера -> Записать текст в следующую ячейку.
* *Результат:* Это вызывает **Layout Thrashing** (в вебе) или множественные **Canvas Rebuilds** (в Unity) за один кадр.
* *Решение:* Пакетная обработка. Сначала выполнить все записи (изменения DOM / State), и только потом позволить движку один раз пересчитать макет. В React это происходит автоматически благодаря механизму Virtual DOM и фазе Reconciliation, которая накапливает изменения перед применением к реальному DOM.1

### 5.3. Repaint против Draw Calls

Изменение чисто визуальных свойств, не влияющих на геометрию (например, color, background-color, opacity), вызывает **Repaint**, пропуская дорогую стадию Layout (Reflow).37 В Unity это аналогично изменению свойства материала или цвета вершин (vertex color), что может потребовать пересборки батча (UpdateBatches), но не требует пересчета RectTransform (UpdateRectTransform).2

Использование CSS свойства transform: translate() переводит элемент в отдельный **Compositor Layer** (обрабатываемый GPU), полностью пропуская этапы Reflow и Repaint на CPU. Это веб-эквивалент использования Sprite в мировом пространстве (World Space), который не вызывает перестроение UI Canvas, а просто обновляет матрицу трансформации на видеокарте.38 Для анимации ховеров в календаре (например, увеличение ячейки при наведении) следует использовать именно transform, а не изменение width/height.

## 6. Адаптивная Архитектура: Алгоритмы Масштабирования

Unity-разработчики полагаются на компонент CanvasScaler для адаптации UI под различные разрешения экранов. Веб использует **Media Queries** и **Fluid Units** (гибкие единицы измерения).

### 6.1. CanvasScaler против Viewport Units

CanvasScaler в режиме Scale With Screen Size применяет глобальный коэффициент масштабирования ко всему корню UI, основываясь на референсном разрешении (Reference Resolution).39 По сути, это зум камеры.

Веб-единицы вьюпорта (vw - viewport width, vh - viewport height) позволяют отдельным элементам масштабироваться относительно экрана *независимо* от глобального зума.

* 100vw = 100% ширины экрана (Unity: AnchorMin.x=0, AnchorMax.x=1).
* 50vh = 50% высоты экрана.

Единица fr в CSS Grid еще мощнее, чем масштабирование в Unity, так как она распределяет пространство *после* учета фиксированных элементов. Unity CanvasScaler не может легко выразить правило: "Оставь боковую панель 200px фиксированной, а контент масштабируй на остаток". В Unity это требует сложной комбинации якорей (Anchors) и Layout Groups. CSS Grid решает это нативно: grid-template-columns: 200px 1fr.10

### 6.2. Media Queries против Программной Верстки

Для кардинального изменения макета на мобильных устройствах (например, переключение с таблицы на список):

* **Unity:** Требует написания C# скрипта, который в Update() или OnRectTransformDimensionsChange проверяет Screen.width, и императивно включает/выключает Layout Groups или меняет родителей объектов.41 Это императивный подход, склонный к ошибкам.
* **CSS:** **Media Queries** (@media (max-width: 600px)) позволяют декларативно менять правила отображения прямо в таблице стилей.

.calendar { grid-template-columns: repeat(7, 1fr); }

@media (max-width: 600px) {

.calendar { grid-template-columns: 1fr; } /\* Переключение в режим списка \*/

}

```

Это архитектурное разделение «Логики» (JS) и «Правил представления» (CSS) является фундаментом стабильности веб-приложений, предотвращая создание «спагетти-кода», характерного для Unity UI контроллеров, вручную манипулирующих RectTransform.43

### 6.3. Жидкая типографика: clamp() против Best Fit

В Unity опция Best Fit на компоненте Text является ресурсоемкой, так как она генерирует различные атласы шрифтов для разных размеров текста, увеличивая потребление памяти (Texture Memory) и нагрузку на CPU при генерации вершин.5

CSS функция clamp(min, preferred, max) позволяет реализовать жидкое масштабирование, которое подчиняется строгим математическим ограничениям и обрабатывается растеризатором браузера максимально эффективно.

font-size: clamp(12px, 2vw, 24px);

Эта одна строка заменяет сложные скрипты интерполяции размера шрифта в зависимости от DPI экрана, которые пришлось бы писать в Unity.44

## 7. Ось Z и Сортировка: Stacking Contexts против Sorting Groups

Последняя, и часто самая запутанная концепция для Unity-разработчиков, переходящих в веб — это управление глубиной (Depth Sorting).

### 7.1. Глобальный Z против Контекста Наложения (Stacking Context)

В Unity 2D свойство SortingOrder (int) является глобальным. Объект с Order 10 всегда рендерится поверх объекта с Order 5, независимо от их иерархии (если не используется компонент SortingGroup).46

В CSS свойство z-index не является глобальным. Оно работает относительно текущего Контекста Наложения (Stacking Context).

Контекст наложения создается специфическими свойствами, такими как opacity < 1, transform, filter, или явным z-index (если position не static).48

Архитектурная Ловушка:

Если у родительского div установлен z-index: 1 (создан контекст), то его дочерний элемент с z-index: 9999 никогда не сможет перекрыть соседа родителя, у которого z-index: 2. Дочерний элемент навсегда "заперт" внутри слоя своего родителя.

Полная Аналогия с Unity:

Это работает ровно так же, как компонент Sorting Group в Unity.48 Когда вы добавляете Sorting Group на GameObject, все его дочерние спрайты сортируются вместе как единый слой. Внутренние SortingOrder детей разрешаются локально, а результирующее "сплющенное" изображение сортируется относительно мира, используя SortingOrder самой группы.

* **CSS Stacking Context** = **Unity Sorting Group**.
* **CSS z-index** = **Unity Order in Layer** (локальный внутри группы).

Понимание этой аналогии предотвращает часы отладки, когда установка z-index: 9999 не выводит модальное окно поверх остального контента, потому что оно находится внутри контейнера с transform (который неявно создал Stacking Context).48

## 8. Практическое применение: Архитектура Календаря

Синтезируя вышеизложенную теорию для выполнения задачи Четвертого дня (Frontend микросервиса Календаря), мы формулируем следующие архитектурные директивы:

1. **Выбор Контейнера:** Использовать **CSS Grid** для компонента DaysGrid.
   * *Обоснование:* В отличие от GridLayoutGroup в Unity, использование grid-template-columns: repeat(7, 1fr) гарантирует создание семи колонок равной ширины, которые реактивно адаптируются под ширину экрана, не требуя фиксированного размера ячейки в пикселях. Это обеспечивает корректный рендеринг на любых устройствах.
2. **Позиционирование:** Использовать grid-column-start для смещения первого дня месяца.
   * *Аналогия:* В Unity для того, чтобы первое число месяца начиналось со среды, пришлось бы инстанцировать пустые "ghost" GameObjects в начале списка. В CSS Grid мы просто применяем стиль к первому дочернему элементу: grid-column-start: 3 (для среды). Это устраняет необходимость в создании лишних DOM-узлов, снижая потребление памяти.52
3. **Инкапсуляция:** Применять **CSS Modules** (Calendar.module.css).
   * *Обоснование:* Это удерживает имена классов, такие как .day, в локальной области видимости префаба Календаря, предотвращая конфликты стилей с другими частями Enterprise-приложения (например, с глобальными стилями кнопок).
4. **Производительность:** Минимизировать изменения state, вызывающие Layout Thrashing.
   * Для анимации наведения (hover) на ячейку дня использовать CSS свойство transform: scale(), чтобы задействовать GPU Compositor и избежать запуска тяжелого процесса Reflow (аналога Unity Rebuild) в главном потоке.

## 9. Глубокий анализ алгоритмов обновления (Update Loops)

Для написания действительно производительных приложений необходимо понимать, *когда* и *как* происходит перемещение пикселей. Проведем сравнительный анализ циклов обновления обоих движков.

### 9.1. Unity: Анализ исходного кода LayoutRebuilder

Система UI в Unity базируется на интерфейсе ICanvasElement. Такие компоненты, как Text, Image и LayoutGroup, реализуют этот интерфейс.

Когда изменяется свойство, влияющее на макет (например, rectTransform.sizeDelta), компонент вызывает метод LayoutRebuilder.MarkLayoutForRebuild(rectTransform).18

Этот вызов добавляет компонент в список List<ICanvasElement> внутри реестра CanvasUpdateRegistry.

**Последовательность перестроения (в конце кадра):**

1. **Layout Pass:** Unity сортирует "грязные" элементы макета по глубине иерархии (родители обрабатываются перед детьми или наоборот, в зависимости от типа группы).
2. **CalculateLayoutInputHorizontal/Vertical:** Система итерирует по элементам и запрашивает их minWidth, preferredWidth и т.д..55
3. **SetLayoutHorizontal/Vertical:** Контроллеры раскладки (например, VerticalLayoutGroup) применяют математику и модифицируют RectTransform своих детей.
4. **Graphic Update:** После того как RectTransforms зафиксированы, компоненты Graphic (Text/Image) регенерируют свои вершинные массивы (Mesh) под новые размеры.

**Архитектурная слабость:** Весь этот процесс происходит в **Главном потоке C#**. Сложный UI с глубоко вложенными Layout Groups может легко потреблять 5-10 мс кадрового бюджета только на итерации GetComponent и сортировку списков.18

### 9.2. Браузер: Архитектура Reflow (Layout)

Браузерный движок компоновки (например, LayoutNG в Blink) значительно сложнее, так как он должен обрабатывать обтекание текста (floats), фрагментацию (разбивку на страницы при печати) и двунаправленный текст, что Unity обычно игнорирует.

Система «Грязных битов» (Dirty Bit System):

Как и в Unity, браузеры используют систему грязных флагов. Модификация класса (element.classList.add('active')) помечает узел DOM как "dirty".

Но в отличие от Unity, где перестроение надежно происходит в конце цикла Update (фаза Canvas Update), браузер пытается быть ленивым. Он не пересчитывает макет немедленно. Он ждет.

Ловушка «Forced Synchronous Layout»:

Если ваш JavaScript код запрашивает геометрическое значение до того, как браузер естественным образом обновился:

JavaScript

div.classList.add('expanded'); // Запись: Пометили как грязный  
console.log(div.offsetHeight); // Чтение: ТРЕБУЕТ немедленного ответа

Браузер вынужден приостановить выполнение JS, запустить весь алгоритм компоновки (Layout Pass) немедленно, чтобы вычислить offsetHeight, и только затем возобновить JS.3

Если это происходит внутри цикла (например, итерация по дням календаря для выравнивания их высоты), это вызывает Layout Thrashing, что может "заморозить" UI на сотни миллисекунд. В Unity существует точный эквивалент этой ошибки: вызов Canvas.ForceUpdateCanvases() внутри цикла foreach.35

**Выводы по производительности:**

* **Unity:** Оптимизация достигается через разделение Canvas (Split Canvases), что уменьшает область перестроения (dirty scope).32
* **React:** Оптимизация достигается через пакетную обработку обновлений состояния (React делает это автоматически) и чтение свойств макета только тогда, когда это абсолютно необходимо (например, внутри хука useLayoutEffect), что позволяет избегать принудительных синхронных пересчетов.

## 10. Продвинутая Архитектура Стилизации: Design Tokens и ScriptableObjects

В крупных проектах на Unity для хранения конфигурационных данных (цветов, шрифтов, настроек игры) используются ScriptableObjects. Это позволяет дизайнерам настраивать внешний вид игры («скины»), не затрагивая код. В экосистеме веба этот паттерн реализуется с помощью **Design Tokens** (обычно через CSS Variables / Custom Properties).

### 10.1. Единый Источник Истины (Single Source of Truth)

Unity:

Вы создаете ThemeData ScriptableObject, содержащий поле public Color PrimaryColor;.

UI Префабы ссылаются на этот объект. Изменение цвета в ассете ScriptableObject обновляет вид всей игры (иногда требуя перезапуска или скрипта OnValidate).

CSS:

Вы определяете CSS Custom Properties (Переменные) на корневом уровне :root:

CSS

:root {  
 --primary-color: #ff0000;  
 --spacing-unit: 8px;  
}

Компоненты используют эти токены: background-color: var(--primary-color);.

Это ссылка времени выполнения (runtime-linked reference), в точности как ссылка на поле ScriptableObject. Изменение значения --primary-color в корне (например, при переключении со Светлой темы на Темную) мгновенно перерисовывает все приложение без перезагрузки страницы и без выполнения JS-логики компонентов.56

### 10.2. React Context против Static Singletons

Чтобы передавать темы вниз по иерархии, Unity-разработчики часто используют Синглтон GameManager.Instance.Theme.

React предлагает Context API (ThemeContext). Однако, для чисто визуальной стилизации CSS Переменные превосходят Context API, так как они обходят цикл рендеринга React (Render Cycle) полностью. Композитор браузера обрабатывает изменение цвета нативно, делая «смену скина» практически бесплатной с точки зрения CPU, тогда как обновление React Context может вызвать ре-рендер всего дерева компонентов.58

Это подтверждает преимущество **Retained Mode**: мы обновляем *определение* состояния (переменную CSS), и движок сам распространяет это изменение, вместо того чтобы наш код итерировал и обновлял каждый пиксель вручную.

## 11. Заключение: Ментальная карта Старшего Инженера

Данный отчет деконструировал переход от Unity UI к React/CSS через призму архитектурной эквивалентности. «Четвертый день» плана обучения — это не просто изучение синтаксиса сетки; это процесс перевода понятий **Constraint Solver** (Grid), **Layout Controller** (Flexbox) и **Render Pipeline** (Reflow/Repaint) на язык их аналогов из игровых движков.

**Сводная таблица теоретических переводов:**

| **Концепция Unity (C#)** | **Концепция Web (CSS/JS)** | **Архитектурное следствие** |
| --- | --- | --- |
| **GridLayoutGroup** | **CSS Grid** | Grid решает ограничения по 2 осям одновременно; поддерживает неявное создание треков (implicit sizing), в отличие от жестких ячеек Unity. |
| **Horizontal/Vertical Layout** | **Flexbox** | Flexbox разделяет выравнивание по главной и поперечной осям и нативно поддерживает перенос строк (wrapping). |
| **LayoutElement.flexibleWidth** | **Fr Unit / Flex-grow** | Оба распределяют свободное пространство, но CSS позволяет *контейнеру* определять это для ребенка (Grid), централизуя логику. |
| **Prefab Variant** | **CSS Module** | Инкапсулирует область видимости стилей, предотвращая глобальное загрязнение пространства имен (Side Effects). |
| **Canvas.BuildBatch** | **Reflow (Layout)** | Оба процесса — дорогие геометрические вычисления. Необходимо избегать их синхронного вызова (Thrashing) внутри циклов. |
| **Sorting Group** | **Stacking Context** | Изолирует сортировку по глубине. Локализует влияние z-index / OrderInLayer в рамках конкретного поддерева. |
| **CanvasScaler** | **Viewport Units / Media Queries** | Веб масштабирует элементы индивидуально или меняет топологию макета на основе правил, вместо глобального зума всего Canvas. |

Интернализировав эти соответствия, Unity-разработчик может перестать бороться с «поточной» (Flow) моделью браузера и начать использовать её как высокооптимизированный, нативный движок компоновки, который автоматизирует большую часть ручных вычислений, необходимых при разработке игр. Реализация Календаря станет полигоном для закрепления этих концепций: использование Grid для макро-структуры, Flexbox для контента ячеек и CSS Modules для инкапсуляции «Префаба Дня».

#### Источники

1. Интенсив React для Backend Разработчика неделя 10
2. Unity UI Profiling: How Dare You Break My Batches? - The Gamedev Guru, дата последнего обращения: декабря 6, 2025, <https://thegamedev.guru/unity-ui/profiling-canvas-rebuilds/>
3. What forces layout/reflow. The comprehensive list. - GitHub Gist, дата последнего обращения: декабря 6, 2025, <https://gist.github.com/paulirish/5d52fb081b3570c81e3a>
4. Grid Layout Group | Unity UI | 1.0.0, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-GridLayoutGroup.html>
5. Optimizing Unity UI - Unity Learn, дата последнего обращения: декабря 6, 2025, <https://learn.unity.com/tutorial/optimizing-unity-ui>
6. CSS Grid Layout Module Level 2 - W3C, дата последнего обращения: декабря 6, 2025, <https://www.w3.org/TR/css-grid-2/>
7. Performance analysis of Grid Layout – make everything intensely, дата последнего обращения: декабря 6, 2025, <https://blogs.igalia.com/jfernandez/2015/06/24/performance-on-grid-layout/>
8. Grid Tidbits part 3: grid track sizing - Emil Björklund, дата последнего обращения: декабря 6, 2025, <https://thatemil.com/blog/2015/08/24/grid-tidbits-3-track-sizing/>
9. Understanding min-content, max-content, and fit-content in CSS - LogRocket Blog, дата последнего обращения: декабря 6, 2025, <https://blog.logrocket.com/understanding-min-content-max-content-fit-content-css/>
10. Basic concepts of grid layout - CSS - MDN Web Docs, дата последнего обращения: декабря 6, 2025, <https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Grid_layout/Basic_concepts>
11. UGUI Layout Performance Considerations - Unity Learn, дата последнего обращения: декабря 6, 2025, <https://learn.unity.com/tutorial/ugui-layout-performance-considerations-1>
12. Anyone understand what "FlexibleWidth" and "FlexibleHeight" properties on ILayoutElement are? : r/Unity3D - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/Unity3D/comments/3iu5c5/anyone_understand_what_flexiblewidth_and/>
13. Layout Element | Unity UI | 1.0.0, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-LayoutElement.html>
14. Auto Layout | Unity UI | 1.0.0, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UIAutoLayout.html>
15. CSS Grid Layout Guide, дата последнего обращения: декабря 6, 2025, <https://css-tricks.com/snippets/css/complete-guide-grid/>
16. Auto-placement in grid layout - CSS - MDN Web Docs - Mozilla, дата последнего обращения: декабря 6, 2025, <https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Grid_layout/Auto-placement>
17. Making UI elements fit the size of their content | Unity UI | 1.0.0, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/HOWTO-UIFitContentSize.html>
18. Optimizing UI Performance in Unity: Deep Dive into LayoutElement and LayoutGroup Components | by magic Hung, дата последнего обращения: декабря 6, 2025, <https://llmagicll.medium.com/optimizing-ui-performance-in-unity-deep-dive-into-layoutelement-and-layoutgroup-components-b6a575187ee4>
19. Vertical Layout Group | Unity UI | 1.0.0, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-VerticalLayoutGroup.html>
20. Horizontal, Vertical, and Grid Layout Groups | by Felix M Quinones - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@felix.m.quinones777/horizontal-vertical-and-grid-layout-groups-886b311c34eb>
21. CSS Layouts: Flexbox vs G | R | I | D | by Maurice Roy - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@mautayro/css-layouts-flexbox-vs-g-r-i-d-7845d39eef2f>
22. Relationship of grid layout to other layout methods - CSS - MDN Web Docs, дата последнего обращения: декабря 6, 2025, <https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Grid_layout/Relationship_with_other_layout_methods>
23. Unity UI performance optimization tips, дата последнего обращения: декабря 6, 2025, <https://unity.com/how-to/unity-ui-optimization-tips>
24. Prefabs vs Prefab Variants in Unity - Extra Ordinary, the Series, дата последнего обращения: декабря 6, 2025, <https://extra-ordinary.tv/2021/07/03/prefabs-vs-prefab-variants-in-unity/>
25. Unity Prefabs Explained: Nested Prefabs and Prefab Variants - YouTube, дата последнего обращения: декабря 6, 2025, <https://www.youtube.com/watch?v=PSKKQKr2bnA>
26. The evolution of scalable CSS, Part 5: Styles Encapsulation - Andrei Pfeiffer, дата последнего обращения: декабря 6, 2025, <https://andreipfeiffer.dev/blog/2022/scalable-css-evolution/part5-styles-encapsulation>
27. Encapsulating styles with CSS Modules | by Emmanuel Pilande - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@epilande/working-with-css-modules-a6b8aad37f3a>
28. From CSS Modules to Styled Components - for React Developers | by Renato Guimaraes Nunes | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@renatognunes/styled-components-or-css-modules-for-react-developers-4eae5bf5cf25>
29. Create variations of prefabs - Unity - Manual, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/6000.2/Documentation/Manual/PrefabVariants.html>
30. React custom compnent css modules vs global css - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/72989889/react-custom-compnent-css-modules-vs-global-css>
31. Encapsulated CSS: The Key to Composable Layouts - Non-traditional Dev, дата последнего обращения: декабря 6, 2025, <https://non-traditional.dev/encapsulated-css-the-key-to-composable-layouts-94f11c177cc1>
32. Split canvas for dynamic objects - Unity Support, дата последнего обращения: декабря 6, 2025, <https://support.unity.com/hc/en-us/articles/115000355466-Split-canvas-for-dynamic-objects>
33. Optimizing Performance in Scenes with Moving Canvasses : r/Unity3D - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/Unity3D/comments/g3ywsk/optimizing_performance_in_scenes_with_moving/>
34. What are Reflow and Repaint? How to optimize? - ExplainThis, дата последнего обращения: декабря 6, 2025, <https://www.explainthis.io/en/swe/repaint-and-reflow>
35. Scripting API: UI.LayoutRebuilder.ForceRebuildLayoutImmediate - Unity - Manual, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.LayoutRebuilder.ForceRebuildLayoutImmediate.html>
36. How To Fix Forced Reflows And Layout Thrashing - DebugBear, дата последнего обращения: декабря 6, 2025, <https://www.debugbear.com/blog/forced-reflows>
37. What's the difference between reflow and repaint? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/2549296/whats-the-difference-between-reflow-and-repaint>
38. Understanding Stacking Contexts - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@svijayaraagavan/understanding-stacking-contexts-b85b227b2fe9>
39. Canvas Scaler | Unity UI | 1.0.0, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-CanvasScaler.html>
40. Canvas Scaler, Explained | Unity Tutorial - YouTube, дата последнего обращения: декабря 6, 2025, <https://www.youtube.com/watch?v=be4gDI8RZ2U>
41. is responsive design or media queries not a thing with game ui? : r/gamedev - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/gamedev/comments/yzghmu/web_designer_interested_in_game_ui_is_responsive/>
42. Does Unity UI Toolkit support CSS media queries? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/73819165/does-unity-ui-toolkit-support-css-media-queries>
43. What do you prefer Styled Components or CSS modules? : r/webdev - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/webdev/comments/sxjqoj/what_do_you_prefer_styled_components_or_css/>
44. Fluid typography with CSS clamp - Piccalilli, дата последнего обращения: декабря 6, 2025, <https://piccalil.li/blog/fluid-typography-with-css-clamp/>
45. Linearly Scale font-size with CSS clamp() Based on the Viewport, дата последнего обращения: декабря 6, 2025, <https://css-tricks.com/linearly-scale-font-size-with-css-clamp-based-on-the-viewport/>
46. Render order according to hierarchy in Unity - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/62300426/render-order-according-to-hierarchy-in-unity>
47. Controlling Unity's rendering order - why and how | Toca Boca Tech Blog - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/toca-boca-tech-blog/controlling-unitys-rendering-order-why-and-how-3c2bb1da8e78>
48. Stacking context - CSS - MDN Web Docs, дата последнего обращения: декабря 6, 2025, <https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Positioned_layout/Stacking_context>
49. css - Can someone explain stacking contexts? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/19796687/can-someone-explain-stacking-contexts>
50. Scripting API: SortingGroup - Unity - Manual, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/6000.2/Documentation/ScriptReference/Rendering.SortingGroup.html>
51. The Z-Index Problem. Why Stacking Contexts Are a CSS… | by Shubham Sharma | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@ss-tech/the-z-index-problem-91226fb74955>
52. grid-column-start - CSS - MDN Web Docs - Mozilla, дата последнего обращения: декабря 6, 2025, <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Properties/grid-column-start>
53. A Calendar in Three Lines of CSS, дата последнего обращения: декабря 6, 2025, <https://css-tricks.com/a-calendar-in-three-lines-of-css/>
54. Scripting API: LayoutRebuilder - Unity - Manual, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/2018.1/Documentation/ScriptReference/UI.LayoutRebuilder.html>
55. Decomposing the Unity UI Automatic Layout System including Vertical, Horizontal, and Grid Layout Groups - Thor Projects, дата последнего обращения: декабря 6, 2025, <https://thorprojects.com/2020/08/25/decomposing-the-unity-ui-automatic-layout-system-including-vertical-horizontal-and-grid-layout-groups/>
56. Design Token-Based UI Architecture - Martin Fowler, дата последнего обращения: декабря 6, 2025, <https://martinfowler.com/articles/design-token-based-ui-architecture.html>
57. Unity Architecture: Scriptable Object Pattern | by Simon Nordon - Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@simon.nordon/unity-architecture-scriptable-object-pattern-0a6c25b2d741>
58. Do react update the whole component when state change or just specific element? - Reddit, дата последнего обращения: декабря 6, 2025, <https://www.reddit.com/r/reactjs/comments/18f5l2b/do_react_update_the_whole_component_when_state/>