DOI 10.37882/2223-2966.2025.10.17

СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ АРХИТЕКТУР GAMEOBJECT И ECS ПРИ МОДЕЛИРОВАНИИ ТОЛПЫ ИЗ N АГЕНТОВ В UNITY

COMPARISON OF THE PERFORMANCE OF GAMEOBJECT AND ECS ARCHITECTURES IN MODELING A CROWD OF N AGENTS IN UNITY

N. Ishchenko

Summary. The paper presents a comparative performance analysis of two approaches in Unity: the classical GameObject/MonoBehaviour architecture and the new Entity Component System (ECS). For validation, a crowd simulation based on the well-known Boids algorithm, widely used for modeling collective behavior, was implemented under both paradigms. Experiments were conducted on two hardware platforms: a PC (Intel Core i5-12400F, NVIDIA RTX 3070) and a MacBook Pro (Apple M4). For each implementation, performance metrics were gathered, including FPS, frame time, memory usage, and the number of draw calls. The collected data made it possible to analyze differences in scalability and efficiency between the architectures. The results demonstrated that ECS achieves significantly higher frame rates and stability as the number of agents increases, which confirms its efficiency for tasks requiring real-time performance and large-scale simulations.

Keywords: Unity, Data-Oriented Tech Stack, DOTS, GameObject, Entity Component System, parallel computing, performance, rendering, GPU instancing, optimization.

Ищенко Никита Николаевич
Южно-Российский государственный политехнический университет им. М.И. Платова, г. Новочеркасск
rektorat@npi-tu.ru

Аннотация. В статье проводится сравнительный анализ производительности двух подходов в Unity: классической архитектуры GameObject (игровой объект в Unity)/MonoBehaviour и новой компонентно-сущностной системы (Entity Component System, ECS). Для проверки реализована симуляция толпы агентов на основе алгоритма «Boids», широко применяемого для моделирования коллективного поведения. Эксперименты проведены на двух аппаратных платформах: ПК (Intel Core i5-12400F, NVIDIA RTX 3070) и ноутбуке MacBook Pro (Apple M4). Для каждой реализации собирались метрики FPS, время кадра, использование памяти и количество вызовов отрисовки. Полученные данные позволяют проанализировать различия в масштабируемости и эффективности архитектур. Результаты показали, что ECS обеспечивает существенно более высокую кадровую частоту и стабильность работы при увеличении числа агентов, что подтверждает его эффективность для задач, требующих высокой производительности в условиях реального времени и масштабируемых симуляций.

Ключевые слова: Unity, Data-Oriented Tech Stack, DOTS, GameObject, Entity Component System, параллельные вычисления, производительность, рендеринг, GPU instancing, оптимизация.

Введение

Unity традиционно использует архитектуру GameObject/MonoBehaviour, которая удобна, но накладывает ограничения на производительность [10]. Традиционная объектно-ориентированная архитектура игровых движков имеет ограничения по масштабируемости [5, с. 228]. Технология DOTS (Data-Oriented Tech Stack — технологический стек, ориентированный на данные) и архитектура ECS призваны устранить эти проблемы: данные отделяются от логики, структуры оптимизируются для кэша, а вычисления параллелизируются с помощью модели параллельного выполнения задач Unity (Job System). Предыдущие исследования показывают, что такой подход значительно повышает эффективность при массовых вычислениях [11, с. 45]. В качестве объекта исследования выбран алгоритм Boids, так как он отражает реальные вычислительные нагрузки при большом числе агентов и поэтому хорошо подходит для демонстрации преимуществ ECS. Таким образом, статья соответствует направлению «информатика и инженерия программного обеспечения» в рамках ВАК, поскольку исследуются архитектуры программирования и их влияние на производительность. Настоящая работа направлена на экспериментальное сравнение двух подходов в идентичных условиях с целью определить, какой из них обеспечивает более высокую производительность.

Материалы и методы

Были реализованы две версии симуляции Boids: на GameObject/MonoBehaviour и на ECS/DOTS. Обе используют одинаковые правила поведения (separation, alignment, cohesion), единые параметры агентов и условия окружения (ограниченный объём пространства с wrap-границей (режим обёртывания границ)). В GameObject-версии каждый агент представлен объ-

ектом с MonoBehaviour, выполняющим перебор всех соседей ($O(N^2)$) на основном потоке. В ECS-версии каждый агент является сущностью с компонентами данных, а вычисления выполняются в параллельных задачах (Jobs) с использованием Burst-компиляции. Для визуализации в ECS применялся GPU instancing.

Производительность оценивалась с помощью AutoSceneProfiler, собирающего статистику FPS, времени кадра, использования памяти и draw calls за 30 секунд работы сцены. Тестировались наборы из 100, 500, 1000, 2000, 4000, 6000 и 10000 агентов. Эксперименты проводились на ПК с RTX 3070 и MacBook Pro с Apple M4 при отключённом VSync и одинаковых настройках качества.

Таким образом, методика задаёт условия для дальнейшего сопоставления с существующими исследованиями и позволяет корректно оценить их результаты.

Литературный обзор

Алгоритм Boids (К. Рейнольдс, 1987) является классическим методом имитации коллективного поведения [1, с. 25]. Агентные модели активно применяются в навигации и анализе толп [2, с. 42–45; 3, с. 7280–7282], а также при анализе поведения толп в эвакуационных сценариях [4, с. 487–490]. В отечественной литературе многоагентные системы подробно рассматриваются в работе Принева и соавт., где отмечаются современные подходы к моделированию и проектированию [12, с. 170]. Кроме того, актуальные материалы о DOTS доступны в официальной документации Unity [9]. Традиционная объектно-ориентированная архитектура игровых движков описана в ряде работ [5, с. 228], но её масштабируемость ограничена. Современные исследования указывают на преимущества дата-ориентированных методов и ECS [6, с. 233; 7, с. 45; 8, с. 65]. В российском контексте также появляется всё больше исследований по ECS: например, работа Докучаева и соавт. демонстрирует рост производительности и гибкости при использовании данного подхода [13, с. 60]. Для обоснования также используются материалы Unity Technologies о DOTS, исследования по параллельным вычислениям в игровых движках [10], и практические обзоры производительности ECS в массовых симуляциях [11, с. 47]. Кроме того, в исследовании Скрябиной приведён сравнительный анализ методов оптимизации, включая параллельное выполнение задач и асинхронную загрузку, что также подтверждает значимость подобных подходов для игровой индустрии [14, с. 3]. При этом прямых сравнений GameObject и ECS для моделирования больших толп немного, что определяет актуальность настоящего исследования. Таким образом, выявленный пробел в литературе напрямую определяет задачи данной работы — проведение экспериментального сравнения двух подходов в идентичных условиях.

Результаты

На обеих платформах выявлены одинаковые закономерности (см. табл. 1 и табл. 2). При числе агентов до 500 разница между подходами невелика. Однако далее GameObject стремительно теряет производительность: при 2000 объектах FPS падает до однозначных значений, а при 10000 — практически до нуля. При этом время кадра возрастает до сотен и тысяч миллисекунд. В то же время ECS демонстрирует плавное снижение FPS и сохраняет интерактивность: около 50–65 FPS на MacBook Pro и более 50 FPS на RTX 3070 при 10000 агентов. Среднее время кадра у ECS даже при максимальной нагрузке не превышает 20 мс, что соответствует требованиям реального времени. Кроме того, ECS удерживает стабильный уровень потребления памяти и GC Alloc, а число draw calls остаётся на уровне десятков, тогда как у GameObject оно растёт до десятков тысяч.

Обсуждение результатов

Классическая архитектура GameObject упирается в главный поток: именно CPU становится узким местом уже при нескольких сотнях агентов [10]. Из-за отсутствия параллелизма и высокой стоимости управления объектами время обработки резко возрастает, а FPS падает. В реализации ECS нагрузка распределяется между ядрами за счёт Jobs и Burst-компиляции, что позволяет обрабатывать сотни тысяч обновлений в секунду [11, с. 46]. Память в DOTS расходуется более предсказуемо: из-за предварительного резервирования объём стабилен [13, с. 61], тогда как у GameObject он растёт вместе с числом агентов. По части рендеринга ECS использует GPU instancing, благодаря чему количество draw calls не зависит от N [5, с. 230], в отличие от GameObject, где каждая сущность генерирует отдельный вызов отрисовки. Платформенные различия проявились в том, что на MacBook Pro (M4) GameObject дольше сохраняет работоспособность за счёт более оптимизированного CPU [12, с. 175], но при больших N преимущество ECS становится очевидным и на Mac, и на ПК. В целом ECS продемонстрировал лучшую масштабируемость и устойчивость по всем ключевым метрикам [14, с. 4].

Использование CPU в классическом подходе оказалось узким местом: основное время тратится на главный поток, в то время как ECS распределяет нагрузку между ядрами [7, с. 47]. Пропускная способность ECS достигала сотен тысяч обновлений агентов в секунду, что в десятки и сотни раз выше, чем у GameObject [8, с. 68]. Память в DOTS используется более эффективно при больших N: начальные затраты выше из-за резервирования, но рост потребления ниже [13, с. 61]. Сборщик мусора в ECS практически не задействован, тогда как у GameObject объём аллокаций растёт с увеличением N [14, с. 3]. По части графики ECS благодаря instancing требует на порядки

Таблица 1.

Результаты измерений производительности симуляции поведения агентов (Apple MacBook Pro; чип M4; 16 ГБ ОЗУ; macOS; разрешение экрана 3024×1964; VSync отключён)

Метод реализации	Количество агентов, N	Средний FPS	Минимальный FPS	Время кадра, мс	Используемая RAM, МБ	GC Alloc, КБ/кадр	Количество draw calls, шт.
GameObject	100	1012,986	222,367	1,09	261,911	1,377	127,039
GameObject	500	79,58	73,019	12,606	262,607	1,34	671,677
GameObject	1000	20,316	18,601	49,401	264,207	1,402	1261,883
GameObject	2000	5,561	5,238	179,981	266,546	1,604	2472,677
GameObject	4000	1,454	1,363	688,702	271,642	2,585	5019,455
GameObject	6000	0,654	0,633	1529,876	276,304	4,071	8023,85
GameObject	10000	0,243	0,241	4123,66	286,404	8,368	12403,625
DOTS	100	1057,685	478,581	1,066	301,029	1,375	23,999
DOTS	500	1043,104	461,114	1,116	301,302	1,376	23,999
DOTS	1000	916,818	437,453	1,275	301,452	1,384	23,999
DOTS	2000	668,092	376,383	1,559	302,805	1,401	23,999
DOTS	4000	267,833	175,1	3,848	303,299	1,411	24
DOTS	6000	147,405	109,195	6,946	304,646	1,382	25
DOTS	10000	65,778	53,765	15,37	307,193	1,345	26,345

Таблица 2.

Результаты измерений производительности симуляции (ПК: Intel Core i5-12400F; NVIDIA GeForce RTX 3070; 16 ГБ ОЗУ; Windows 11; разрешение 2560×1080; VSync отключён)

Метод реализации	Количество агентов, N	Средний FPS	Минимальный FPS	Время кадра, мс	Используемая RAM, МБ	GC Alloc, КБ/кадр	Количество draw calls, шт.
GameObject	100	810,625	452,407	1,254	161,299	0,739	142,781
GameObject	500	69,269	56,307	14,532	162,574	0,719	576,128
GameObject	1000	18,791	16,663	53,379	164,507	0,712	1276,322
GameObject	2000	4,957	4,562	201,88	168,28	0,671	2308,638
GameObject	4000	1,293	1,201	774,418	173,725	0,615	4801,744
GameObject	6000	0,586	0,562	1708,763	178,118	0,667	6897,667
GameObject	10000	0,211	0,209	4733,688	187,009	0	14354,857
DOTS	100	1686,491	1271,655	0,596	203,892	0,756	22,996
DOTS	500	1632,921	1270,952	0,615	204,138	0,742	23,994
DOTS	1000	1422,447	892,926	0,717	204,34	0,724	27,206
DOTS	2000	805,312	406,828	1,31	205,81	0,721	31,357
DOTS	4000	244,231	135,066	4,346	206,259	0,723	44,149
DOTS	6000	127,231	75,404	8,336	207,466	0,722	51,975
DOTS	10000	51,642	35,013	20,194	209,302	0,721	73,893

меньше draw calls (табл. 1–2), разгружая CPU и перенося работу на GPU.

В совокупности ECS демонстрирует лучшее масштабирование, стабильность и эффективность ресурсов по всем ключевым метрикам. Исходный код реализованных сценариев (GameObject и ECS) доступен в GitHub-репозитории: https://github.com/DragonAirDragon/ScientificPaper (дата обращения: 01.09.2025).

Заключение

Данная работа была направлена на проведение экспериментального сравнения двух подходов в Unity — классического GameObject/MonoBehaviour и архитектуры ECS/DOTS — в идентичных условиях моделирования толпы агентов. Исследование подтвердило значительное превосходство ECS при увеличении числа агентов: начиная примерно с тысячи объектов, ECS обеспечивает кратный прирост по FPS и стабильности. Преимущества достигаются за счет параллелизма, эффективного хранения данных и оптимизации рендеринга. Эти результаты получены на двух различных аппаратных платформах (ПК и Mac), что подтверждает универсальность подхода. Научная новизна исследования заключается в том, что впервые проведен сравнительный эксперимент по применению ECS и классического подхода на примере алгоритма Boids с реальными измерениями производительности на современном оборудовании.

Таким образом, поставленная цель — определить, какой из подходов обеспечивает более высокую производительность, — достигнута. ECS рекомендуется для задач, связанных с симуляцией большого числа объектов в реальном времени.

ЛИТЕРАТУРА

1. Рейнольдс К.W. Стаи, стада и школы: распределённая модель поведения // SIGGRAPH'87 (Computer Graphics). — 1987. — Т. 21, № 4. — С. 25–34. URL: https://www.red3d.com/cwr/papers/1987/SIGGRAPH87.pdf (дата обращения: 04.09.2025).
2. Эпштейн Дж. М., Акселл Р.Л. Создание искусственных обществ: социальная наука «снизу вверх». — Кембридж, Массачусетс: MIT Press, 1996. — 243 с.
3. Бонабо Э. Агентное моделирование: методы и техники для имитации человеческих систем // Proceedings of the National Academy of Sciences of the USA. — 2002. — Т. 99, Suppl. 3. — С. 7280–7287.
4. Хельбинг Д., Фаркаш И., Вичек Т. Имитация динамических особенностей паники при эвакуации // Nature. — 2000. — Т. 407. — С. 487–490.
5. Асадуззаман А., Ли Х.Ю. GPU-вычисления для повышения производительности игровых движков // Journal of Engineering and Technological Sciences. — 2014. — Т. 46, № 2. — С. 226–243.
6. Сюй Чж. Параллельные технологии и развитие в игровой индустрии // Data Analysis and Machine Learning (DAML). — Сучжоу, 2024. — С. 232–238.
7. Холл Д.М. Проектирование игрового движка на ECS. — ВКР (B.S. project), Калифорнийский политехнический университет, Сан-Луис-Обиспо, 2014. — 65 с.
8. Харрис С.М. Реализация и анализ архитектуры Entity Component System. — Магистерская диссертация, Калифорнийский политехнический университет, Сан-Луис-Обиспо, 2022. — 82 с.
9. Unity. Entities package documentation [Электронный ресурс]. — 2024. URL: https://docs.unity3d.com/Packages/com.unity.entities@0.17/manual/index.html (дата обращения: 01.09.2025).
10. Unity. ECS (Entity Component System) for Unity [Электронный ресурс]. — 2024. URL: https://unity.com/ecs (дата обращения: 06.09.2025).
11. Боруфка Р. Набор тестов производительности для Unity DOTS. — Диссертация, Карлов университет, Прага, 2023. — 102 с. URL: https://dspace.cuni.cz/handle/20.500.11956/116800 (дата обращения: 02.09.2025).
12. Принев М.А., Леденева Т.М., Гаршина В.В. Многоагентные системы: обзор современных подходов к моделированию и проектированию (Часть 2) // Вестник ВГУ. Серия: Системный анализ и информационные технологии. — 2024. — № 4. — С. 167–190. URL: https://journals.vsu.ru/sait/article/view/12690 (дата обращения: 05.09.2025).
13. Докучаев В.А., Маклачкова В.В., Удалов И.Д. Применение Entity Component System при создании игр // Экономика и качество систем связи. — 2025. — № 1. — С. 57–66. URL: https://journal-ekss.ru/wp-content/uploads/2025/02/57-66.pdf (дата обращения: 05.09.2025).
14. Скрябина С.М. Сравнительный анализ различных методов оптимизации и их влияние на качество игрового процесса // Вестник науки. — 2024. — № 3. — С. 1–8. URL: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-razlichnyh-metodov-optimizatsii-i-ih-vliyanie-na-kachestvo-igrovogo-protsessa (дата обращения: 05.09.2025).

Журнал «Современная наука: актуальные проблемы теории и практики»