ПРИМЕНЕНИЕ ENTITY COMPONENT SYSTEM ПРИ СОЗДАНИИ ИГР

***В.А. Докучаев**, д.т.н., профессор, Московский технический университет связи и информатики, v.a.dokuchaev@mtuci.ru;*
***В.В. Маклачкова**, Московский технический университет связи и информатики, v.v.maklachkova@mtuci.ru;*
***И.Д. Удалов**, Московский технический университет связи и информатики, igor.udalov.95@mail.ru.*

УДК 004.9

Аннотация. В данной статье рассматривается применение архитектурного паттерна *Entity Component System (ECS)* в контексте безопасной разработки компьютерных игр. Проводится сравнительный анализ *ECS* с традиционными подходами объектно-ориентированного программирования (ООП). Детально обсуждаются преимущества, которые *ECS* предоставляет в игровой разработке, включая повышение производительности, гибкости, масштабируемости и безопасности персональных данных участников игры. В качестве практического примера рассматривается фреймворк *Unity DOTS (Data-Oriented Technology Stack)*, построенный на принципах *ECS*.

Ключевые слова: *Entity Component System*; *ECS*; объектно-ориентированное программирование; ООП; игровая разработка; *Unity DOTS*; *data-oriented design*; производительность; гибкость; композиция; безопасность персональных данных.

APPLICATION OF ENTITY COMPONENT SYSTEM IN GAME DEVELOPMENT

***V.A. Dokuchaev,** Doctor of Technical Sciences, Professor, Moscow Technical University of Communications and Informatics;*
***V.V. Maklachkova,** Moscow Technical University of Communications and Informatics;*
***I.D. Udalov,** Moscow Technical University of Communications and Informatics.*

Annotation. The article discusses the application of the Entity Component System (*ECS*) architectural pattern in the context of secure computer game development. *ECS* is comparatively analysed with traditional object-oriented programming (*OOP*) approaches. The benefits that *ECS* provides in game development are discussed in detail, including improved performance, flexibility, scalability, and data security. As a practical example, the Unity *DOTS* (Data-Oriented Technology Stack) framework, built on *ECS* principles, is discussed.

***Keywords**:* Entity Component System; *ECS*; object-oriented programming; *OOP*; game development; Unity *DOTS*; data-oriented design; performance; flexibility; composition; personal data security.

Введение

Современная разработка игр характеризуется постоянно растущими требованиями к качеству, детализации и масштабу игровых миров. Обработка сложных взаимодействий между тысячами объектов в реальном времени ставит перед разработчиками непростые задачи в плане оптимизации производительности и обеспечения стабильной работы. Традиционные подходы объектно-ориентированного программирования (ООП), несмотря на свою широкую распространенность и удобство, в ряде сценариев начинают демонстрировать ограничения в контексте создания высокопроизводительных игровых приложений.

В ответ на эти вызовы в последние годы набирает популярность архитектурный паттерн *Entity Component System (ECS)*, предлагающий альтернативный взгляд на организацию игровой логики и данных, ориентированный на принципы *data-oriented design* [1].

Цель исследования, результаты которого представлены в статье, заключается в аргументации преимущества применения *Entity Component System* в процессе создания игр с точки зрения обеспечения производительности, гибкости, масштабируемости и безопасности персональных данных пользователей. В рамках исследования проведен сравнительный анализ *ECS* и ООП, выделены ключевые отличия в их подходах к организации кода и данных. Подробно рассмотрены преимущества, которые *ECS* предоставляет разработчикам игр, а также обозначены потенциальные недостатки [2] и потенциальные риски для персональных данных участников игры. Практическая реализация принципов *ECS* проиллюстрирована на примере фреймворка *Unity DOTS*, который активно использует данный архитектурный паттерн. Также в статье затронут вопрос о возможности интеграции и совместного использования парадигм ООП и *ECS* в рамках одного игрового проекта.

Отличие *ECS* от объектно-ориентированного программирования

В основе архитектурного паттерна *ECS* лежит концепция разделения данных и поведения. В отличие от ООП, где данные и методы инкапсулируются внутри объектов, *ECS* предлагает три ключевые сущности для организации игровой логики:

- Сущности *(Entities)*: представляют собой уникальные идентификаторы, обозначающие «вещи» в игре. Сами по себе сущности не содержат никаких данных или логики, являясь по сути «пустыми контейнерами».
- Компоненты *(Components)*: структуры данных, описывающие характеристики сущности. Примерами компонентов могут служить «Позиция», «Скорость», «Здоровье», «Текстура». Компоненты содержат только данные и не включают в себя какую-либо логику.
- Системы *(Systems)*: модули, содержащие логику обработки данных. Системы оперируют сущностями, обладающими определенным набором компонентов. Например, система «Движение» будет обрабатывать все сущности, имеющие компоненты «Позиция» и «Скорость», обновляя их координаты.

Ключевое различие между *ECS* и ООП заключается в способе организации данных и поведения. В ООП поведение и данные связаны воедино в рамках объектов, которые взаимодействуют друг с другом через вызовы методов. Наследование позволяет создавать иерархии объектов, переиспользуя и расширяя функциональность [3].

ECS, напротив, делает акцент на композиции вместо наследования. Сущности формируются динамически путем добавления к ним необходимых компонентов. Например, сущность «Враг» может состоять из компонентов «Позиция», «Скорость», «Здоровье», «Агрессивность». Другая сущность, например, «Предмет», может состоять из компонентов «Позиция», «Текстура», «Интерактивность». Общее поведение реализуется в системах, которые обрабатывают сущности на основе наличия у них определенных компонентов.

В табл. 1 приведены сравнительные характеристики ООП и *ESC*.

Таблица 1.

Характеристика	**Объектно-ориентированное программирование (ООП)**	***Entity Component System (ECS)***
Организация	Объекты с данными и методами (инкапсуляция)	Сущности (*ID*); Компоненты (данные); Системы (логика)
Поведение	Методы внутри объектов	Системы, оперирующие компонентами
Наследование	Ключевой механизм повторного использования кода	Композиция через добавление компонентов
Хранение данных	Распределено по объектам, может быть фрагментировано в памяти	Централизованно в массивах компонентов
Основной фокус	Поведение объектов	Данные и их обработка
Оптимизация	Сложности с оптимизацией и кэшированием	Лучшие возможности для оптимизации и кэширования

Переход к архитектуре *ECS* предоставляет ряд значительных преимуществ для разработки игр.

Одним из ключевых преимуществ *ECS* является возможность оптимизации производительности за счет применения принципов *data-oriented design*. Компоненты одного типа хранятся в непрерывных массивах в памяти, что значительно улучшает кэширование данных и позволяет эффективно использовать *SIMD*-инструкции *(Single Instruction, Multiple Data)* для параллельной обработки. Системы обрабатывают данные последовательно, минимизируя случайный доступ к памяти и снижая нагрузку на процессор. Также *ECS* позволяет легко добавлять и изменять поведение игровых объектов, просто добавляя или удаляя необходимые компоненты. Это делает систему более гибкой и устойчивой к изменениям требований. Новые функциональные возможности могут быть реализованы путем создания новых систем, не затрагивая существующий код [4].

Кроме того, *ECS* использует композицию вместо наследования. Это обеспечивает большую гибкость и позволяет избежать проблем «хрупкого базового класса», характерных для наследования. Игровые объекты могут быть собраны из различных комбинаций компонентов, что позволяет создавать разнообразные сущности без раздувания иерархии классов. Разделение данных и логики при программировании, с использованием *ECS*, упрощает процесс тестирования. Компоненты представляют собой чистые данные, которые легко создавать и проверять. Системы можно тестировать независимо, предоставляя на вход определенный набор компонентов и проверяя корректность результата. Также архитектура *ECS* хорошо подходит для параллельного программирования. Системы могут обрабатывать разные типы компонентов параллельно, а внутри одной системы обработка компонентов одного типа также может быть распараллелена, что особенно важно для многоядерных процессоров.

Плюсы и минусы *Entity Component System*

Использование *ESC* в разработке имеет как положительные, так и отрицательные стороны по сравнению с ООП.

Из положительных моментов можно выделить следующие:

- высокая производительность благодаря *data-oriented design* и оптимизации кэширования;

- гибкость и расширяемость за счет компонентного подхода;
- уменьшение связности между различными частями кода;
- улучшенная тестируемость благодаря разделению данных и логики;
- хорошая поддержка параллельного программирования;
- более чистый и поддерживаемый код за счет разделения ответственности.

Из отрицательного выделим:

- более высокий порог вхождения для разработчиков, привыкших к ООП;
- потенциально большее количество «шаблонного» кода при явном определении компонентов и систем;
- возможная сложность отладки на начальном этапе из-за неявного взаимодействия систем и компонентов;
- необходимость адаптации мышления при переходе от объектно-ориентированного подхода.

Совместное использование ООП и *ECS* в рамках одной игры

Несмотря на значительные преимущества *ECS*, полный отказ от ООП не всегда является необходимым или целесообразным. Вполне возможно и часто эффективно совместное использование ООП и *ECS* в разных частях одной и той же игры, выбирая наиболее подходящую парадигму для решения конкретных задач [5].

ECS особенно хорошо подходит для обработки больших массивов однотипных сущностей, где критически важна производительность, например, для управления сотнями врагов или частиц. В то же время, ООП может оставаться предпочтительным для реализации сложных, уникальных объектов с ярко выраженным поведением и состоянием, таких как пользовательский интерфейс, менеджеры игрового процесса или уникальные игровые персонажи с разветвленной логикой.

Такой гибридный подход позволяет использовать сильные стороны обеих парадигм. *ECS* обеспечивает высокую производительность там, где это необходимо, а ООП предоставляет удобство и интуитивность для задач, где сложность логики важнее массовой обработки данных. Например, основная игровая механика, включающая взаимодействие множества объектов, может быть реализована с использованием *ECS*, в то время как элементы пользовательского интерфейса могут быть созданы с применением традиционных ООП-подходов. Ключевым моментом является грамотное разделение ответственности и проектирование эффективных механизмов взаимодействия между *ECS*-системами и ООП-объектами.

Unity DOTS:* пример реализации *ECS

Unity DOTS представляет собой современную архитектуру *Unity*, основанную на принципах *ECS*. *DOTS* предоставляет инструменты и *API* для разработки игр с акцентом на высокую производительность, используя *data-oriented* подход.

Ключевыми компонентами *Unity DOTS* являются:

- *Entities*: представляют собой сущности, как и в общей концепции *ECS*. В *DOTS* они реализуются через структуры *Entity*.
- *Components*: описывают данные сущностей и реализуются через структуры (*struct*). Для обеспечения максимальной производительности рекомендуется использовать простые типы данных (*blittable types*).

- *Systems*: содержат логику обработки сущностей и реализуются через классы, наследующие от *SystemBase* или *JobComponentSystem*.
- *Jobs*: позволяют выполнять ресурсоемкие задачи в отдельных потоках, что значительно повышает производительность. *Unity DOTS* активно использует *Job System* для параллельной обработки данных.
- *Burst Compiler*: компилирует *C#* код, используемый в системах и джобах, в высокооптимизированный машинный код, что обеспечивает значительное увеличение скорости выполнения.
- *Collections*: предоставляют эффективные структуры данных для работы с сущностями и компонентами, такие как *NativeArray*, *EntityCommandBuffer*.

Проблема безопасности данных пользователей игровых приложений

По мере роста объемов данных, собираемых компаниями-разработчиками компьютерных игр (игровая компания), одной из важнейших и актуальных проблем становится проблема обеспечения безопасности и защиты этих данных. Количество и разнообразие собираемых данных делает их привлекательными целями для киберпреступников [6-8]. Данные крадутся или для получения финансовой выгоды, или просто для совершения вредоносной атаки типа «отказ в обслуживании» (*DDoS*) на игровую платформу или на игроков. Чаще всего киберпреступники осуществляют кражу личных (персональных) данных игроков, например, номера кредитных карт, либо виртуальных товаров.

Проблемы сбора, обработки и защиты персональных данных, безусловно, не являются новыми или уникальными для отрасли игровой индустрии [9]. Однако по мере ее развития роль персональных данных становится все более значимой для игровой аналитики, используемой игровыми компаниями, в том числе для развития бизнеса. В настоящее время игровые компании используют данные для лучшего понимания поведения игроков как на определенной платформе или устройстве, так и между ними. С помощью современных методов анализа данных можно использовать закономерности и корреляции в игровом процессе для получения дополнительных выводов, например, о биометрической личности пользователя, возрасте и поле, эмоциях, навыках, интересах, социально-экономическом статусе, потребительском поведении, чертах личности, состоянии физического и психического здоровья, показателях тела, культурном и географическом происхождении, привычках и т.п. При этом сами субъекты персональных могут даже не подозревать об этом [10]. И по мере того, как становится доступным больше данных и способов их обработки и анализа, риски персональных данных будут продолжать увеличиваться, поскольку игровые компании все больше вовлекаются в продажу, покупку, торговлю или хранение цифровых активов пользователей (игроков) [7, 8, 11]. Нарушение безопасности данных не только наносит серьезный ущерб самим субъектам персональных данных, но и игровым компаниям, причем ущерб может быть нанесен как напрямую, так и опосредованно, неся за собой финансовые, репутационные и др. потери, в том числе связанные с нарушениями требований Федерального закона «О персональных данных» от 27.07.2006 № 152-ФЗ (далее – ФЗ-152) [12, 13].

В этих условиях перед игровыми компаниями стоит задача обеспечения безопасной разработки самих игровых платформ, защиты своей ИТ-инфраструктуры разработки, а также обеспечение безопасной обработки персональных (личных) данных пользователей в соответствии с ФЗ-152 и другими нормативными актами, регулирующими область персональных данных [14, 15]. Только так игровая компания может гарантировать, что ее игры и платформы являются безопасной средой.

Под безопасной разработкой будем понимать не только интеграцию безопасности во все этапы жизненного цикла разработки игрового программного обеспечения, но и применения при создании продукта программных инструментов, обеспечивающих безопасность данных.

Безопасность в *Unity Entity Component System (ECS)* при компилировании кода

Переход к архитектуре *DOTS* в *Unity*, включающей *ECS*, принес не только значительные улучшения в производительности, но и новые принципы, касающиеся безопасности данных. Традиционные подходы, полагавшиеся на защиту объектов и управление состоянием на уровне отдельных экземпляров, в *ECS* уступают место строгим правилам доступа к данным и явной работе с памятью. Понимание этих принципов критически важно для написания стабильного и предсказуемого кода.

ECS стремится обеспечить безопасность данных на нескольких уровнях, минимизируя возможности для возникновения гонок данных и других ошибок, связанных с конкурентным доступом. Это достигается за счет комбинации ограничений компилятора и системы *Job System*, которая управляет выполнением задач в параллельных потоках.

Разделение данных и логики

Как было написано выше, ключевой принцип *ECS* заключается в разделении данных (компоненты) от логики (системы). Системы работают с массивами компонентов, что позволяет эффективно обрабатывать большие объемы данных. Этот принцип является краеугольным камнем архитектуры *ECS* и играет фундаментальную роль в обеспечении безопасности данных. В традиционном объектно-ориентированном программировании данные и поведение часто тесно связаны внутри объектов. В *ECS*, напротив, данные становятся пассивными, представляясь в виде простых структур, называемых компонентами. Компоненты описывают, что представляет собой сущность (например, позицию, скорость, здоровье), но не как она ведет себя. Логика же концентрируется в системах, которые отвечают за обработку этих данных. Системы итерируются по массивам однотипных компонентов, применяя к ним определенные преобразования и алгоритмы. Такое четкое разделение имеет несколько важных преимуществ для безопасности:

- уменьшение вероятности случайного изменения данных;
- улучшенная предсказуемость;
- оптимизация для работы с большими объемами данных;
- упрощение отладки и тестирования.

В целом, разделение данных и логики в *ECS* создает более строгую и контролируемую среду для работы с данными, что значительно повышает безопасность и надежность приложения.

Использование *Job System*

Большинство вычислительно-интенсивных задач в *ECS* выполняются через *Job System*. Это позволяет распределить нагрузку на несколько ядер процессора, но требует соблюдения определенных правил для обеспечения безопасности. *Job System* является ключевым механизмом параллелизации в *Unity DOTS* и играет решающую роль в достижении высокой производительности. Однако работа с параллельными потоками по своей природе подвержена рискам, таким как «гонки

данных» (*data races*), когда несколько потоков пытаются одновременно изменить одни и те же данные, что приводит к непредсказуемым результатам. *Job System* спроектирована таким образом, чтобы минимизировать эти риски, но требует от разработчика соблюдения определенных правил:

- Автоматическое управление зависимостями: *Job System* позволяет явно указывать зависимости между задачами (*jobs*). Это гарантирует, что задача начнет выполняться только после завершения тех задач, от которых она зависит.
- Контроль доступа к данным в задачах: при создании задачи разработчик должен явно указать, какие компоненты будут читаться (*ReadOnly*) и какие будут записываться. *Job System* использует эту информацию для предотвращения конфликтов.
- Безопасная работа с нативными контейнерами: *Job System* предоставляет механизмы для безопасной передачи и использования нативных контейнеров (например, *NativeArray*) между задачами. Это включает в себя проверку границ и предотвращение доступа к освобожденной памяти.

Таким образом, *Job System* не только позволяет эффективно использовать многоядерные процессоры, но и предоставляет структурированный и безопасный способ организации параллельных вычислений, снижая вероятность возникновения гонок данных и других ошибок, связанных с многопоточностью.

Явное управление *Native Containers*

ECS активно использует нативные контейнеры (например, *NativeArray*, *NativeList*, *NativeHashMap*), выделенные в неуправляемой памяти. Разработчик несет ответственность за их явное выделение и освобождение (через метод *Dispose()*), чтобы избежать утечек памяти. Использование нативных контейнеров является ключевым фактором производительности в *Unity DOTS*. Поскольку контейнеры выделяются в неуправляемой памяти, это позволяет избежать накладных расходов, связанных с автоматической сборкой мусора в управляемой памяти. Однако это также возлагает на разработчика ответственность за явное управление их жизненным циклом, чтобы предотвратить утечки памяти и другие проблемы:

- Явное выделение: разработчик должен явно создавать экземпляры нативных контейнеров, указывая необходимый размер и способ выделения памяти (*Allocator*). Это дает полный контроль над тем, сколько памяти выделяется и когда.
- Явное освобождение (*Dispose*()): после того, как нативный контейнер больше не нужен, разработчик обязан вызвать метод *Dispose*() для освобождения выделенной памяти. Если это не сделать, память, занимаемая контейнером, останется зарезервированной и не будет доступна для повторного использования, что приведет к утечке памяти.
- Проверка на освобождение: перед использованием нативного контейнера, особенно в многопоточной среде, важно убедиться, что он не был случайно освобожден в другом месте.
- Безопасная передача между задачами: при передаче нативных контейнеров между задачами *Job System* необходимо соблюдать определенные правила, чтобы избежать ситуаций, когда несколько задач пытаются одновременно освободить один и тот же контейнер или получить доступ к уже освобожденной памяти.

Несоблюдение правил управления нативными контейнерами является одной из наиболее распространенных причин утечек памяти и других ошибок в проектах, использующих *Unity DOTS.*

Пример использования *Unity DOTS*

Рассмотрим пример реализации системы движения игровых объектов в *Unity DOTS*. В традиционном *Unity* для перемещения каждого игрового объекта потребовался бы отдельный *GameObject* со скриптом, содержащим логику движения.

В *Unity DOTS* подход иной. Каждый игровой объект представлен сущностью. Информация о положении и скорости объекта хранится в компонентах «Позиция» и «Скорость», которые прикреплены к этой сущности. Логика перемещения реализуется в системе «Движение». Эта система сканирует все сущности, у которых есть компоненты «Позиция» и «Скорость». Затем система «Движение» берет данные из этих компонентов и обновляет позицию каждой соответствующей сущности на основе ее скорости и прошедшего времени [16].

Главное преимущество здесь заключается в массовой обработке данных. Компоненты «Позиция» всех двигающихся объектов хранятся рядом в памяти, что позволяет процессору эффективно загружать и обрабатывать их. Система «Движение» может быть распараллелена на несколько потоков, используя *Jobs*, что позволяет задействовать все ядра процессора для одновременного обновления позиций множества объектов. Таким образом, *Unity DOTS* позволяет эффективно управлять движением большого количества игровых объектов, значительно повышая производительность по сравнению с традиционным подходом.

Заключение

Entity Component System представляет собой мощный архитектурный паттерн, предлагающий эффективный подход к разработке производительных и гибких игровых приложений. Отход от традиционной парадигмы ООП в сторону композиции через компоненты позволяет создавать более масштабируемые, тестируемые и оптимизированные игровые системы.

Unity DOTS, как наглядный пример реализации *ECS*, демонстрирует практическую пользу данной архитектуры, предоставляя разработчикам инструменты для создания сложных игровых миров с большим количеством взаимодействующих объектов. Несмотря на определенный порог вхождения, преимущества *ECS* в плане производительности, гибкости и поддержки параллелизма делают его все более привлекательным для разработчиков, стремящихся создавать современные, требовательные и безопасные игры. Возможность совместного использования *ECS* и ООП открывает новые горизонты для разработки, позволяя сочетать сильные стороны обеих парадигм. Дальнейшее развитие *ECS* и его интеграция в различные игровые движки, вероятно, будут играть ключевую роль в эволюции игровой разработки.

Литература

1. Ernest Adams., Joris Dormans. Game Mechanics: Advanced Game Design, 2012. URL: https://typeset.io/papers/game-mechanics-advanced-game-design-23pl62mlvp (дата обращения – ноябрь 2024).
2. Eberly David H. «3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics» Morgan Kaufmann, 2006 URL: https://www.academia.edu/26649713/3D_game_engine_design_a_practical_approach_to_real_time_computer_graphics_second_edition (дата обращения – ноябрь 2024).
3. Zubek Robert. «Game Physics Engine Development» CRC Press, 2018.

4. Brown Adam. «Entity-Component Systems and C#» Apress, 2019.
5. Аскерли М.В. Ключ к эффективной разработке: архитектура ECS в сравнении // Вестник науки, 2024. – № 5 (74). URL: https://cyberleninka.ru/article/n/klyuch-k-effektivnoy-razrabotke-arhitektura-ecs-v-sravnenii (дата обращения – декабрь 2024).
6. Докучаев В.А. Влияние новых информационно-коммуникационных технологий на конфиденциальность персональных данных // Актуальные проблемы и перспективы развития экономики: Труды XXIII Международной научно-практической конференции, Симферополь - Гурзуф, 17-19 октября 2024 года. – Симферополь: ИП Зуева Т.В., 2024. – С. 12-15. – EDN LLFRKS.
7. Статьев В.Ю. Информационная безопасность на пространстве «Больших данных» // T-Comm: Телекоммуникации и транспорт, 2022. – Т. 16. – № 4. – С. 21-28. DOI: 10.36724/2072-8735-2022-16-4-21-28. – EDN IXUYWS.
8. Dokuchaev V.A. Data subject as augmented reality // Synchroinfo Journal, 2020. – V. 6. – № 1. – P. 11-15. – DOI: 10.36724/2664-066X-2020-6-1-11-15. – EDN ULPVZC.
9. Докучаев В.А., Владимирова К.С., Маклачкова В.В., Статьев В.Ю. Аудит информационных рисков в процессе обработки персональных данных // Технологии информационного общества: Материалы XIII Международной отраслевой научно-технической конференции, Москва, 20-21 марта 2019 года. Том 2. – Москва: ООО «Издательский дом Медиа паблишер», 2019. – С. 34-36. – EDN XNJJOM.
10. Докучаев В.А., Маклачкова В.В., Сундатов В.О. Подходы к защите персональных данных на просторе «Больших данных» // Теория и практика экономики и предпринимательства: Труды XX Международной научно-практической конференции, Симферополь - Гурзуф, 20-22 апреля 2023 года. Под редакцией Н.В. Апатовой. – Симферополь: Крымский федеральный университет им. В.И. Вернадского, 2023. – С. 34-36. – EDN BUQVMG.
11. Dokuchaev V.A. Classification of personal data security threats in information systems // T-Comm, 2020. – V. 14. – № 1. – P. 56-60. DOI: 10.36724/2072-8735-2020-14-1-56-60. – EDN QOGYHH.
12. Докучаев В. А. Цифровизация субъекта персональных данных // T-Comm: Телекоммуникации и транспорт, 2020. – Т. 14. – № 6. – С. 27-32. DOI: 10.36724/2072-8735-2020-14-6-27-32. – EDN XVWYJP.
13. Неманова В.И., Вакурин И.С., Маклачкова В.В., Гадасин Д.В. Определение экономической эффективности затрат предприятия на защиту персональных данных // DSPA: Вопросы применения цифровой обработки сигналов, 2024. – Т. 14. – № 3. – С. 37-45. – EDN UIELEK.
14. Докучаев В.А. Постановка задачи оценки качества информации при обработке персональных данных в информационных системах мультиоблачной архитектуры // Теория и практика экономики и предпринимательства: Труды XX Международной научно-практической конференции, Симферополь - Гурзуф, 20-22 апреля 2023 года. Под редакцией Н.В. Апатовой. – Симферополь: Крымский федеральный университет им. В.И. Вернадского, 2023. – С. 37-39. – EDN JACYHT.
15. Докучаев В.А. Постановка задачи оценки качества информации при обработке персональных данных в информационных системах мультиоблачной архитектуры // Теория и практика экономики и предпринимательства: Труды XX Международной научно-практической конференции, Симферополь - Гурзуф, 20-22 апреля 2023 года. Под редакцией Н.В. Апатовой. – Симферополь: Крымский федеральный университет им. В.И. Вернадского, 2023. – С. 37-39. – EDN JACYHT.
16. Entity systems are the future of MMOG development. URL: https://t-machine.org/index.php/2007/11/11/entity-systems-are-the-future-ofmmog-development-part-2/comment-page-1/ (дата обращения - ноябрь 2024).