Министерство цифрового развития государственного управления, информационных технологий и связи Республики Татарстан
государственное автономное профессиональное образовательное учреждение
«Международный центр компетенций – Казанский техникум информационных технологий и связи»

Дипломный проект
выполнен и защищен с оценкой
_____
Председатель ГЭК
_____ А.Ф.Хасьянов
«____»_____20____ г

ДОПУСТИТЬ К ЗАЩИТЕ
Заместитель директора по
учебно-методической работе
_____ О. С. Тимофеева

ДИПЛОМНЫЙ ПРОЕКТ

Проектирование и разработка игрового приложения «Кубикс»

Исполнитель,
выпускник группы 421 П _____ И.И.Валиуллин
подпись, дата

Руководитель
преподаватель
ГАПОУ «МЦК – КТИТС» _____ Л.Г.Хрущева
подпись, дата

Казань, 2026г

государственное автономное профессиональное образовательное учреждение
«Международный центр компетенций –
Казанский техникум информационных технологий и связи»

УТВЕРЖДАЮ
Заместитель директора по учебно-методической работе
_____ О.С. Тимофеева
«___» _____ 2026 г.

ЗАДАНИЕ

на дипломный проект

выпускнику Валиуллину Ильдару Ильнуровичу 421П группы

по специальности 09.02.07 Информационные системы и программирование, квалификация: Программист

1 Тема ДП: Проектирование и разработка игрового приложения «Кубикс», утверждена приказом директора техникума от 27.03.26 № 24 - Д
2 Срок сдачи студентом законченного ДП: 13.06.26
3 Исходные данные к ДП: техническое задание
4 Краткое содержание ДП или перечень подлежащих разработке вопросов:
4.1 Требования к оформлению пояснительной записки
Пояснительная записка должна содержать следующие разделы: описание предметной области, анализ предметной области, обоснование выбора средств разработки, проектирование программного продукта, реализация проекта, руководство пользователя, тестирование программного продукта, заключение, список используемых источников, приложения.
4.2 Требования к программному продукту
Разработать 2D-игру-конструктор уровней с клиент-серверной архитектурой. В игровом приложении должна быть реализована регистрация и авторизация пользователей, создание и редактирование уровней, публикация уровней на сервере, прохождение чужих уровней с фиксацией времени, система лайков и оценок, таблица рекордов. Реализовать удобный интерфейс на базе Unity UI, асинхронное получение данных с сервера и обновление контента в реальном времени.
5 Перечень графических материалов/Приложения: код программы.
Приложение А, Приложение Б

План-график выполнения дипломного проекта:

№ п/п	Наименование этапов ДП	Срок выполнения этапов		Подпись руководителя, консультантов
		План	Факт	
1	Создание и анализ предметной области	01.04.2026-05.04.2026		
2	Проектирование системы	06.04.2026-16.04.2026		
3	Разработка и создание компонентов	17.04.2026-12.05.2026		
4	Тестирование и отладка	13.05.2026-20.05.2026		
5	Написание руководства пользователя	21.05.2026-01.06.2026		
6	Подготовка и конечное оформление документации	02.06.2026-08.06.2026		
7	Внесение исправлений и финальная доработка проекта	09.06.2026-13.06.2026		

Рассмотрено на заседании ЦК Программирование
Протокол № 9 от 28.03.26
Председатель ЦК_____ Л.Р. Калинина
подпись *инициалы и фамилия*

Дата выдачи задания: 1.04.26

Руководитель дипломного проекта:

Преподаватель
ГАПОУ «МЦК – КТИТС» _____ Л.Г.Хрущева
подпись, дата *Инициалы и фамилия*

Задание принял к исполнению: _____ И.И.Валиуллин
подпись *Инициалы и фамилия*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы в игровой индустрии набирает популярность концепция пользовательского контента, позволяющая игрокам самостоятельно создавать игровые уровни и делиться ими с сообществом. Такой подход существенно увеличивает срок жизни игры без постоянных обновлений со стороны разработчика, а также формирует активное сообщество вокруг продукта. Платформенные игры с редакторами уровней, такие как Super Mario Maker, Geometry Dash и Levelhead, подтверждают коммерческую и культурную значимость данного направления. Однако среди существующих решений редко встречаются проекты, совмещающие несколько ключевых механик: свободное создание уровней в вертикальной ориентации, зацикленное горизонтальное пространство с телепортацией между левым и правым краями экрана, встроенные инструменты отладки через контрольные точки, а также полноценную систему оценки, лайков и соревновательного прохождения чужих уровней. Большинство аналогов либо ограничивают возможности редактора, либо не предоставляют прозрачной системы проверки проходимости перед публикацией, что приводит к замусориванию каталога непроходимыми уровнями.

Целью дипломной проекта является создание 2D-платформера со встроенным редактором уровней, реализованного на движке Unity, предоставляющего пользователям полный цикл работы с контентом: от проектирования и отладки до публикации, оценки и соревнования. Для достижения поставленной цели проведён анализ предметной области и существующих игр-конструкторов, сформулировано техническое задание, спроектирована схема взаимодействия клиентского приложения и серверной части.

В ходе работы реализовано клиентское приложение на Unity 2D, включающее систему регистрации и входа пользователей, а также панель глобальных настроек. Разработан визуальный редактор уровня с возможностью размещения, перемещения и удаления игровых объектов. Уровни строятся

вертикально — от нижней границы к верхней. Введена уникальная механика горизонтального зацикливания: если персонаж игрока пересекает правую границу экрана, появляется слева, и наоборот, что позволяет создавать нестандартные пространственные конструкции. Для удобства тестирования во время разработки уровня предусмотрена система контрольных точек — игрок может расставлять контрольные точки и при падении или ошибке возвращаться к ближайшей из них. После завершения конструирования доступен режим чистового прохождения без контрольных точек; если уровень признаётся проходимым, автор получает возможность выложить работу на сервер. Помимо этого, в игре реализован каталог уровней, созданных другими пользователями, с механиками лайков и рейтинговой оценки. Отдельным функциональным блоком выступает соревновательный режим, в котором игроки могут состязаться в скорости прохождения каждого уровня с фиксацией лучшего времени.

Практическая значимость работы заключается в создании готового к публикации игрового продукта, который позволяет сообществу игроков самостоятельно генерировать контент, оценивать работы друг друга и соревноваться, не требуя вмешательства разработчика. Разработанная система предоставляет удобный редактор с поддержкой нестандартной механики зацикленного пространства, встроенные инструменты отладки через контрольные точки, механизм модерации проходимости на этапе публикации, а также социальные механики, такие как лайки, оценки, таблицы рекордов. Приложение обладает отзывчивым интерфейсом на базе Unity UI, низким порогом входа для новых пользователей и масштабируемой клиент-серверной архитектурой. Внедрение данного продукта позволит сформировать устойчивое игровое сообщество, увеличить вовлечённость игроков за счёт творческой свободы и соревновательного элемента, а также снизить затраты на создание нового контента силами разработчика.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Описание предметной области

В разрабатываемой игре пользователи выступают одновременно в роли авторов и игроков. Каждый пользователь имеет уникальный идентификатор, логин и пароль, причём логин является уникальным. Пользователь может создавать неограниченное количество собственных уровней, а также проходить, оценивать и оставлять отзывы на уровни, созданные другими игроками. Таким образом, система поддерживает полный цикл жизнедеятельности пользовательского контента: от создания до публикации, оценки и соревнования.

Каждый уровень в системе обладает уникальным идентификатором, названием, автором, датой создания, количеством лайков, средней оценкой и числом прохождений. Публикация уровня на сервере возможна только при условии его проходимости — то есть существования пути от стартовой позиции до финиша. Один пользователь может создавать неограниченное количество уровней, проходя при этом любые этапы редактирования и тестирования.

Конструкция уровня представляет собой двумерное пространство, организованное вертикально: игровые объекты размещаются от нижней границы к верхней. Такая структура позволяет проектировать уровни с постепенным нарастанием сложности, где игрок начинает движение снизу и продвигается к финишу, расположенному выше. Дополнительной особенностью игрового поля является горизонтальная связанность: пространство зациклено таким образом, что перемещение за одну границу экрана приводит к появлению персонажа с противоположной стороны. Данная особенность учитывается при проектировании уровней и проверке их проходимости, однако не является обязательной для всех конструкций и может использоваться автором по желанию для создания нестандартных маршрутов.

В редакторе уровней пользователь может размещать, перемещать, вращать и удалять игровые объекты. Типы объектов включают: платформы, препятствия, стартовую позицию, финиш, а также контрольные точки.

Контрольные точки предназначены для отладки уровня в процессе создания: при падении персонажа или ошибке игрок возвращается к последней активированной контрольной точке, что позволяет поэтапно тестировать сложные участки без необходимости каждый раз проходить уровень с самого начала. При финальной проверке уровня перед публикацией контрольные точки игнорируются.

Для публикации уровня автор обязан пройти его в режиме чистового прохождения без использования контрольных точек. Если уровень успешно пройден от старта до финиша, автор получает возможность опубликовать его на сервере с указанием названия и описания. В случае если уровень оказывается непроходимым, публикация блокируется с выводом соответствующего сообщения, что предотвращает засорение каталога непроходимыми уровнями.

Пользователи могут взаимодействовать с чужими уровнями следующими способами: проходить уровень с фиксацией времени прохождения, ставить лайк, выставлять числовую оценку от 1 до 5. Один пользователь может поставить лайк и оценку конкретному уровню только один раз. При повторном прохождении уровня фиксируется лучшее время пользователя, которое обновляется в таблице рекордов, что стимулирует соревновательный дух и многократное перепрохождение.

Для каждого уровня ведётся таблица лидеров, в которой хранятся: идентификатор пользователя, лучшее время прохождения, дата установления рекорда и количество попыток. В таблице отображаются только лучшие результаты каждого пользователя. Соревновательный режим доступен только для опубликованных уровней. Если пользователь прошёл уровень с результатом, превосходящим его предыдущий личный рекорд, запись в таблице обновляется, а владелец уровня при необходимости может отслеживать активность вокруг своего контента.

В игре также реализованы глобальные настройки, такие как: регулировка громкости звуковых эффектов и фоновой музыки, выбор разрешения экрана, а также переназначение клавиш управления. Настройки сохраняются локально на

устройстве пользователя и сохраняются между сессиями, что обеспечивает комфортную и персонализированную игровую среду.

1.2 Анализ предметной области

Разрабатываемая система представляет собой игровое приложение в жанре 2D-платформера со встроенным редактором уровней. Система обеспечивает полный цикл работы с пользовательским контентом: проектирование уровней, их локальное тестирование, публикацию на сервере, каталогизацию, социальное взаимодействие и соревновательное прохождение с фиксацией рекордов времени.

Ролевая модель и взаимодействие с системой. В разрабатываемой игре выделяется одна основная роль — авторизованный пользователь. Неавторизованные пользователи не имеют доступа к функционалу системы. Авторизованный пользователь может выполнять следующие группы действий.

Группа управления учётной записью: регистрация в системе с созданием уникальной пары логин-пароль; авторизация для получения доступа ко всем функциям; выход из системы.

Группа настроек: регулировка громкости звуковых эффектов и фоновой музыки; выбор разрешения экрана; переназначение клавиш управления. Все настройки сохраняются локально и применяются при последующих запусках.

Группа создания контента: проектирование уровней во встроенном редакторе с возможностью размещения, перемещения, вращения и удаления игровых объектов; расстановка контрольных точек для поэтапной отладки сложных участков; тестовое прохождение уровня с использованием контрольных точек; чистовое прохождение уровня без контрольных точек для проверки проходимости; публикация уровня на сервере.

Группа потребления контента: просмотр каталога уровней, созданных другими пользователями, с возможностью сортировки и фильтрации; прохождение чужого уровня с фиксацией времени; повторное прохождение для улучшения личного рекорда; постановка лайка уровню; оценка уровня по шкале от 1 до 5; просмотр таблицы лидеров.

Функциональные требования. Игровой процесс основан на классической механике 2D-платформера: персонаж может перемещаться влево и вправо по

горизонтальной плоскости, прыгать по платформам, взаимодействовать с объектами уровня. Проверка нахождения персонажа на земле реализуется с помощью лучей, исходящих из нижней части коллайдера персонажа, что обеспечивает точное определение момента, когда можно выполнить прыжок [13].

Камера плавно следует за персонажем, удерживая его в центре области обзора, но с ограничениями по границам уровня. Пространство уровня организовано вертикально: объекты располагаются от нижней границы к верхней, а игрок продвигается снизу вверх к финишу. Уровень может обладать горизонтальной связанностью: при выходе персонажа за левую или правую границу экрана, то появляется с противоположной стороны без изменения вертикальной координаты, что расширяет возможности проектирования маршрутов. Данная особенность не является обязательной и может использоваться автором по своему усмотрению.

Сериализация уровня выполняется в формат JSON. Для каждого сохраняемого объекта фиксируются: тип, координаты X и Y, угол поворота. JSON-представление используется для локального сохранения уровня, а также для отправки на сервер при публикации [14].

Хранение и обработка данных. Для хранения пользовательских данных, уровней, лайков, оценок и рекордов спроектирована реляционная база данных PostgreSQL. Схема базы данных включает следующие основные таблицы.

Таблица users хранит учётные записи пользователей: уникальный идентификатор, логин (уникальный), хешированный пароль, дату регистрации.

Таблица levels хранит опубликованные уровни: идентификатор, название, описание, идентификатор автора, данные уровня в формате JSONB, количество лайков, среднюю оценку, даты создания и обновления. Использование типа JSONB позволяет хранить полную структуру уровня без создания сложных реляционных схем для каждого типа объектов.

Таблица level_likes фиксирует лайки пользователей уровням. Составной уникальный ключ (user_id, level_id) гарантирует, что один пользователь может поставить лайк одному уровню не более одного раза.

Таблица level_ratings хранит числовые оценки пользователей. Аналогично таблице лайков, уникальная пара (user_id, level_id) обеспечивает однократность оценки.

Таблица level_records хранит лучшие времена прохождения уровней пользователями. Уникальная пара (user_id, level_id) гарантирует, что для каждой пары пользователь-уровень сохраняется только один рекорд. При улучшении результата запись обновляется.

Для обеспечения производительности запросов созданы индексы по внешним ключам (author_id, level_id) и по полю времени рекорда для быстрой сортировки в таблицах лидеров.

В системе действуют следующие правила.

Правила публикации уровня: уровень можно опубликовать только тому пользователю, который его создал; перед публикацией автор обязан пройти уровень в режиме чистового прохождения без использования контрольных точек; если уровень пройден успешно, публикация разрешается; в противном случае система блокирует публикацию с сообщением о непроходимости.

Правила оценивания и лайков: один пользователь может поставить лайк конкретному уровню не более одного раза; повторный лайк от того же пользователя игнорируется или отменяет предыдущий; один пользователь может выставить оценку уровню только один раз; повторная оценка заменяет предыдущую; оценка является целым числом в диапазоне от 1 до 5.

Правила рекордов: при первом прохождении уровня пользователем его время фиксируется как рекорд; при повторном прохождении время сравнивается с сохранённым рекордом; если новое время меньше, запись обновляется; если новое время больше или равно, запись остаётся неизменной; таблица лидеров отображает записи, отсортированные по возрастанию времени, с ограничением количества отображаемых позиций.

Правила контрольных точек: при прохождении уровня в режиме отладки после гибели персонаж возрождается на последней активированной контрольной точке; если ни одна контрольная точка не активирована, возрождение происходит на стартовой позиции; при чистовом прохождении контрольные точки игнорируются, гибель персонажа приводит к немедленному завершению попытки.

Сценарии использования. На основе анализа предметной области выделены следующие ключевые сценарии.

Сценарий 1: регистрация и авторизация. Пользователь открывает приложение, заполняет форму регистрации (логин, пароль), система проверяет уникальность логина, создаёт учётную запись и автоматически выполняет вход. При последующих запусках пользователь вводит логин и пароль в форму авторизации, система проверяет соответствие и выдает сессионный токен.

Сценарий 2: создание и публикация уровня. Пользователь открывает редактор, размещает на поле объекты, расставляет контрольные точки по желанию, запускает тестовое прохождение для отладки. После завершения конструирования пользователь запускает чистовой забег. Если уровень пройден успешно, система предлагает ввести название и описание, после чего выполняет публикацию: уровень отправляется на сервер в формате JSON и становится доступным в общем каталоге.

Сценарий 3: прохождение и оценка чужого уровня. Пользователь открывает каталог уровней, просматривает список, применяет сортировку, выбирает интересующий уровень и запускает его. В процессе прохождения система фиксирует время от старта до финиша. После завершения пользователь может повторно пройти уровень для улучшения результата, поставить лайк, выставить оценку или вернуться в каталог.

Сценарий 4: соревновательное прохождение. Пользователь выбирает уровень в каталоге, открывает таблицу рекордов, где отображаются лучшие времена других игроков. Пользователь запускает уровень с целью побить текущий личный рекорд или рекорд лидера таблицы. После улучшения

результата новый рекорд автоматически отправляется на сервер, и позиция пользователя в таблице обновляется.

Нефункциональные требования. Система должна обеспечивать следующие нефункциональные характеристики.

Производительность: время отклика серверного API при получении списка уровней не должно превышать 300 мс при стандартной нагрузке; сериализация и десериализация уровня в JSON должна выполняться на клиенте без заметных задержек для пользователя (менее 100 мс для уровня до 5 МБ).

Надёжность: система должна корректно обрабатывать ошибки сетевого взаимодействия; при потере соединения с сервером во время публикации или отправки рекорда пользователь должен получить понятное сообщение с предложением повторить попытку.

Масштабируемость: архитектура клиент-серверного взаимодействия должна допускать увеличение числа пользователей и уровней без изменения логики клиента.

Безопасность: пароли пользователей хранятся в базе данных только в хешированном виде; все операции, требующие авторизации (публикация, лайки, оценки, рекорды), проверяют валидность сессии; JSON-данные уровней, приходящие от клиента, валидируются на сервере для предотвращения повреждения или некорректной структуры.

Удобство использования: интерфейс редактора является интуитивно понятным и не требует от пользователя изучения сложной документации; размещение объектов выполняется одним кликом; перемещение и удаление — минимальным числом действий; справочные подсказки должны отображаться при наведении на элементы управления.

Таким образом, анализ предметной области позволил сформулировать полный набор функциональных и нефункциональных требований, выявить ключевые сущности и связи между ними, а также определить бизнес-правила, которые реализованы в разрабатываемой системе. Полученные результаты

легли в основу проектирования архитектуры клиентского и серверного приложений, а также схемы базы данных.

1.3 Обзор аналогов

Анализ конкурентной среды представляет собой важный этап разработки проекта, позволяющий оценить текущие рыночные предложения, выявить преимущества и недостатки существующих аналогов. Изучение существующих решений дает возможность определить направления для дифференциации продукта и формирования уникальных характеристик. Полученные данные помогают скорректировать функциональные особенности разработки, адаптировать под актуальные запросы пользователей и выработать эффективную стратегию позиционирования на рынке. Подобный подход способствует созданию конкурентоспособного продукта с четко определенной целевой аудиторией.

В ходе анализа рассмотрены три наиболее популярных представителя жанра игр-конструкторов уровней, а также платформеров с поддержкой пользовательского контента.

Игра «Super Mario Maker» (Nintendo, платформа: Nintendo Switch / Wii U) (Рисунок 1.1).

Рисунок 1.1 Скриншот игры «Super Mario Maker»

Super Mario Maker – игра-конструктор уровней, разработанная компанией Nintendo, позволяющая пользователям создавать собственные уровни в стилистике классических игр про Марио. Игроки могут размещать платформы, врагов, бонусы и декоративные элементы, используя интуитивно понятный редактор с сенсорным управлением. Созданные уровни можно публиковать онлайн, а также проходить уровни других игроков. К преимуществам Super Mario Maker относятся: высокое качество исполнения и полированный геймплей, огромное сообщество игроков, наличие фильтрации уровней по популярности и сложности, интеграция с глобальными рейтингами и системой лайков.

Однако к недостаткам платформы можно отнести: отсутствие механики зацикленного пространства, что ограничивает дизайнерские возможности; отсутствие встроенных инструментов отладки уровня; платформа является эксклюзивом Nintendo, что недоступно на ПК и мобильных устройствах; большинство механик завязано на стилистику вселенной Марио, что затрудняет создание уровней с оригинальной физикой; отсутствие соревновательного режима с таблицами рекордов времени на каждом уровне.

Игра «Geometry Dash» (RobTop Games, платформа: ПК, мобильные устройства) (Рисунок 1.2).

Рисунок 1.2 Скриншот игры «Geometry Dash»

Geometry Dash – ритм-платформер со встроенным редактором уровней, в котором игроки управляют геометрической фигурой, движущейся вперёд, и должны преодолевать препятствия под музыку. Редактор уровней позволяет создавать сложные последовательности препятствий, синхронизированные с музыкальным треком. Пользователи могут публиковать свои уровни, проходить чужие, ставить лайки и оставлять комментарии. К преимуществам Geometry Dash относятся: огромное количество пользовательских уровней, развитая система сложности и рейтинга, наличие системы контрольных точек в редакторе, поддержка нескольких платформ (ПК, iOS, Android), активное сообщество и соревновательные элементы.

Среди минусов выделяются: отсутствие механики горизонтальной телепортации между краями экрана; редактор имеет высокий порог входа из-за сложной системы триггеров и таймингов; оценка проходимости уровня происходит только через публикацию и отзывы сообщества, таблицы рекордов реализованы только для официальных уровней, а не для всех пользовательских.

Игра «Levelhead» (Butterscotch Shenanigans, платформа: ПК, мобильные устройства, консоли) (Рисунок 1.3.3).

Рисунок 1.3 Скриншот игры «Levelhead»

Levelhead – платформер-конструктор уровней, во многом вдохновлённый Super Mario Maker, но с рядом уникальных особенностей. Игроки создают уровни для робота-курьера GR-18, размещая платформы, ловушки, телепорты, подвижные элементы и интерактивные объекты. К преимуществам Levelhead относятся: наличие внутриигровой торговой площадки с экономикой, система заданий для авторов уровней, поддержка сквозных механик с телепортами, возможность публиковать уровни с условной проходимостью, поддержка таблиц лидеров на пользовательских уровнях, наличие контрольных точек в редакторе.

Недостатками платформы являются: отсутствие вертикального построения уровней «снизу вверх»; механика зацикливания пространства не реализована на уровне игрового поля целиком; проверка проходимости уровня полностью лежит на сообществе, автоматическая система отсутствует; небольшая аудитория по сравнению с Mario Maker или Geometry Dash, что сказывается на количестве доступных уровней; система рекордов времени не всегда корректно работает из-за особенностей физики.

Таким образом, в ходе данного этапа проанализированы три популярные игры-конструктора уровней, занимающие лидирующие позиции на рынке пользовательского контента. Ключевым выводом анализа является отсутствие среди существующих решений продукта, который сочетал бы в себе следующие характеристики: вертикальное построение уровней, механику полного горизонтального зацикливания пространства, встроенную автоматическую проверку проходимости уровня перед публикацией с системой контрольных точек для отладки, а также полноценные соревновательные таблицы рекордов для каждого пользовательского уровня. Существующие платформы либо ограничены по доступным платформам, либо не поддерживают уникальные пространственные механики, либо не предоставляют инструментов автоматической верификации проходимости перед публикацией. Это создаёт объективную нишу для разработки специализированного игрового конструктора с заявленным набором функций.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

2.1 ER-диаграмма

Проектирование схемы базы данных выполнено на основе реляционной модели. Структура хранения данных представлена на диаграмме «сущность-связь», которая отображает основные сущности разрабатываемой системы, их атрибуты и связи между ними. Диаграмма построена в соответствии с требованиями, сформулированными в аналитической части, и отражает логику хранения пользователей, уровней, лайков, оценок и рекордов времени.

Центральным узлом схемы является таблица пользователей users. Данная сущность хранит учётные данные каждого зарегистрированного игрока и включает атрибуты: id, login, password, created_at. Таблица users выступает в роли родительской для всех остальных сущностей, поскольку каждый игровой объект привязан к конкретному пользователю.

Основной сущностью, хранящей игровой контент, является таблица levels. Levels содержит атрибуты: id, name, description, author_id, level_data, likes_count, created_at, updated_at. Связь между таблицами users и levels имеет тип «один ко многим»: один пользователь может создать множество уровней, каждый уровень принадлежит ровно одному автору. При удалении пользователя все созданные им уровни автоматически удаляются.

Для реализации механики лайков спроектирована таблица level_likes. Level_likes содержит атрибуты: id, user_id, level_id, liked_at. Составной уникальный ключ user_id и level_id гарантирует, что один пользователь может поставить лайк одному уровню не более одного раза. При удалении пользователя или уровня все связанные записи в level_likes удаляются.

Для хранения числовых оценок уровней создана таблица level_ratings. Её структура включает: id, user_id, level_id, rating, rated_at. Уникальный ключ user_id и level_id обеспечивает однократность оценки: один пользователь может оценить конкретный уровень только один раз. При повторной оценке запись обновляется.

Таблица level_records отвечает за хранение лучших времён прохождения уровней. Её атрибуты: id, user_id, level_id, time_ms, achieved_at. Уникальный ключ user_id и level_id гарантирует, что для каждой пары пользователь-уровень сохраняется только одна запись — наилучшее время. Для повышения производительности создан индекс по полю time_ms, позволяющий быстро сортировать результаты от лучшего времени к худшему.

Сформирована таблица объектов базы данных (Таблица 2.1)

Таблица 2.1 – Объекты базы данных

Объект	Назначение	Основные атрибуты
users	Хранение учётных записей пользователей	id, login, password, created_at
levels	Хранение опубликованных уровней	id, name, description, author_id, level_data (JSONB), likes_count, created_at, updated_at
level_likes	Фиксация лайков пользователей к уровням	id, user_id, level_id, liked_at
level_ratings	Хранение числовых оценок уровней	id, user_id, level_id, rating (1-5), rated_at
level_records	Хранение лучших рекордов времени	id, user_id, level_id, time_ms, achieved_at

Сформирована таблица связей между объектами (Таблица 2.2).

Таблица 2.2 – Связи между объектами

Связь	Тип	Описание
users → levels	1:M	Один пользователь создаёт много уровней, каждый уровень принадлежит одному автору
users → level_likes	1:M	Один пользователь ставит много лайков, каждый лайк принадлежит одному пользователю
levels → level_likes	1:M	Один уровень может получить много лайков, каждый лайк относится к одному уровню
users → level_ratings	1:M	Один пользователь выставляет много оценок, каждая оценка принадлежит одному пользователю
levels → level_ratings	1:M	Один уровень может получить много оценок, каждая оценка относится к одному уровню

users → level_records	1:M	Один пользователь имеет много рекордов, каждый рекорд принадлежит одному пользователю

Продолжение таблицы 2.2

levels → level_records	1:M	Один уровень может содержать много рекордов, каждый рекорд относится к одному уровню

На основе выделенных объектов и связей спроектирована ER-диаграмма базы данных, наглядно отображающая структуру хранения информации в игровом приложении (Рисунок 2.1).

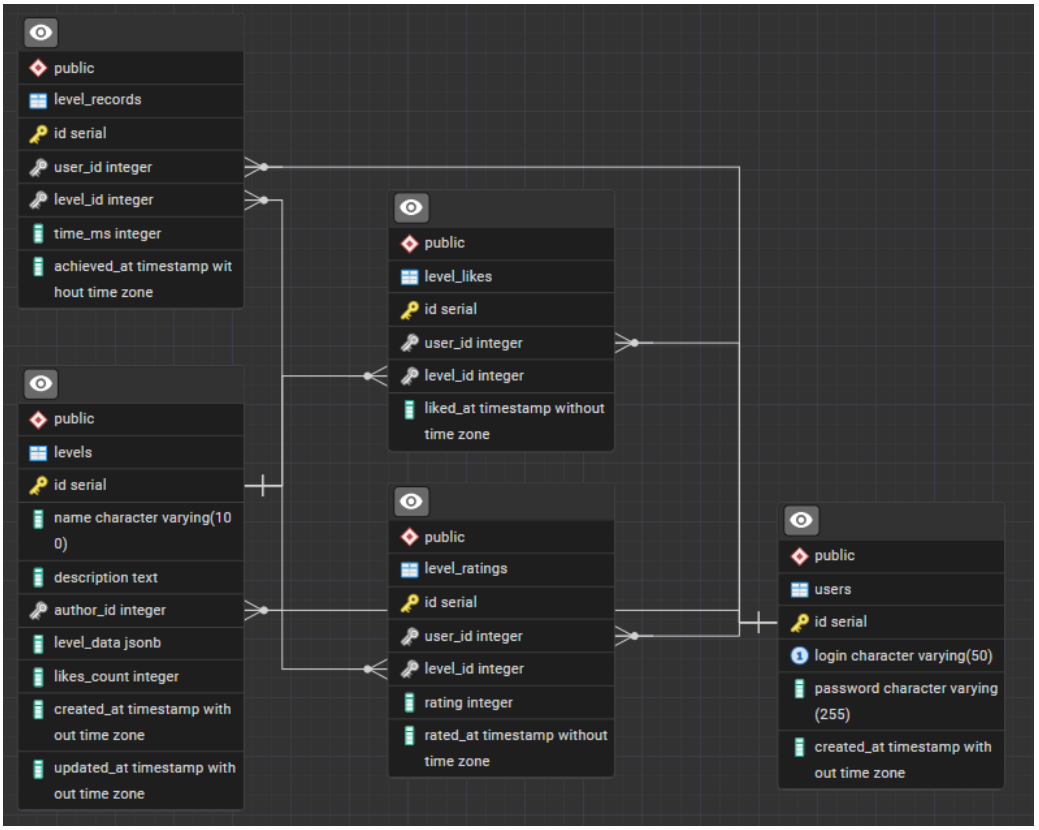

Рисунок 2.1 ER-диаграмма базы данных игрового приложения

Как показано на диаграмме, все периферийные таблицы level_likes, level_ratings, level_records связаны с центральной таблицей levels через внешний ключ level_id, а с таблицей users — через внешний ключ user_id. Такая архитектура обеспечивает строгую ссылочную целостность, исключает избыточность хранимой информации и позволяет эффективно выполнять типовые запросы: получение списка уровней с информацией об авторе, подсчёт среднего рейтинга уровня, формирование таблицы лидеров с сортировкой по

времени. Использование типа JSONB для поля level_data позволяет хранить сложную структуру уровня без создания дополнительных таблиц.

2.2 Диаграмма вариантов использования

Use-Case диаграмма — это тип диаграмм Unified Modeling Language, который отображает функциональные требования к системе с точки зрения взаимодействия с ней внешних пользователей. Диаграмма показывает, какие действия может выполнять пользователь в системе, а также связи между различными сценариями использования [8].

Для разрабатываемой 2D-игры-конструктора уровней построена диаграмма вариантов использования, отражающая весь функционал, доступный конечному пользователю (Рисунок 2.2).

Рисунок 2.2 Диаграмма Use-case

В разрабатываемой системе выделяется один основной актер — Игрок. Игрок представляет собой авторизованного пользователя, который может создавать уровни, проходить чужие уровни, оценивать их и соревноваться за лучшее время прохождения. Неавторизованные пользователи не имеют доступа к функционалу системы.

На основе анализа предметной области и требований к системе выделены следующие варианты использования.

Первой группой функций являются действия, связанные с учётной записью пользователя. Игрок может зарегистрироваться в системе, создав уникальный логин и пароль, а также авторизоваться для получения доступа ко

всем возможностям игры. В любой момент игрок может выйти из игры, завершив текущую сессию.

Следующая группа относится к настройкам игрового процесса. Игрок может изменять настройки игры, включая регулировку громкости звуковых эффектов и музыки, а также переназначение клавиш управления.

Третья группа функций связана с созданием собственного контента. Игрок может создавать уровни с помощью встроенного редактора, размещая платформы, препятствия, собираемые предметы, стартовую позицию и финиш на вертикально ориентированном поле. Для проверки проходимости перед публикацией игрок может пройти уровень в чистовом режиме без контрольных точек, и, если уровень успешно пройден, опубликовать на сервере, сделав доступным для других игроков.

Четвёртая группа функций связана с потреблением контента других пользователей. Игрок может просматривать уровни других игроков в общем каталоге, а также проходить их. После прохождения чужого уровня игрок может оценить уровень по шкале от 1 до 5, добавить уровень в понравившиеся, а также просмотреть таблицу рекордов уровня, где отображаются лучшие времена прохождения всех игроков. Дополнительно доступен соревновательный режим, в котором игрок может проходить уровень с целью улучшить собственное время и подняться выше в таблице лидеров.

Связи между вариантами использования. Варианты использования «Авторизация» и «Регистрация» являются обязательными предшествующими условиями для большинства действий, за исключением «Настройки игры», которая доступна и без авторизации. Вариант использования «Создание уровня» включает в себя «Прохождение созданного уровня» и «Публикацию уровня». Вариант использования «Прохождение чужого уровня» связан с вариантами «Оценка уровня», «Добавление уровня в понравившиеся» и «Соревновательное прохождение», поскольку после завершения уровня игрок может совершить любое из этих действий.

3 РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА

3.1 Обоснование выбора среды разработки

Visual Studio — это интегрированная среда разработки от Microsoft, предоставляющая инструменты для написания, отладки, тестирования и компиляции кода. IDE поддерживает множество языков программирования, включая C#, и обладает расширенными возможностями для работы с проектами различной сложности [4].

Выбор Visual Studio в качестве основной среды разработки обусловлен следующими причинами.

Во-первых, Visual Studio обеспечивает полноценную интеграцию с движком Unity посредством инструмента Visual Studio Tools for Unity. Данный инструмент предоставляет отладку игровой логики непосредственно в процессе выполнения игры внутри Unity, позволяя устанавливать точки останова, отслеживать значения переменных и анализировать стек вызовов в реальном времени. Это существенно ускоряет выявление и исправление ошибок при разработке игровых механик. Без такой интеграции поиск ошибок в сложных сценариях, связанных с физикой персонажа и обработкой столкновений, значительно затруднился.

Во-вторых, Visual Studio обладает мощным редактором кода с поддержкой IntelliSense — системы контекстной подсказки и автодополнения кода. Для языка C# это означает автоматическое распознавание типов, подсказки по параметрам методов и быструю навигацию по классам и файлам проекта. В условиях разработки игрового проекта с множеством взаимодействующих компонентов данная функциональность снижает вероятность синтаксических ошибок и ускоряет написание кода. Кроме того, Visual Studio предоставляет инструменты рефакторинга, позволяющие безопасно переименовывать переменные и методы по всему проекту.

В-третьих, для работы с базой данных на серверной стороне используется Entity Framework Core — объектно-ориентированный ORM от Microsoft. Entity Framework Core позволяет взаимодействовать с PostgreSQL через C#-объекты,

не создавая сложные SQL-запросы вручную. Это упрощает сохранение уровней, лайков, оценок и рекордов, а также ускоряет разработку и поддержку кода. Visual Studio предоставляет встроенные инструменты для работы с Entity Framework Core, включая консоль диспетчера пакетов для создания миграций. Миграции автоматически генерируют SQL-код на основе изменений в C#-моделях, что минимизирует риск ошибок [6].

В-четвёртых, для создания серверного API выбран ASP.NET Core. Этот фреймворк позволяет быстро разработать REST API, через который игровой клиент на Unity отправляет и получает данные: опубликованные уровни, таблицы лидеров, информацию о пользователях. ASP.NET Core обеспечивает высокую производительность, асинхронную обработку запросов и удобную интеграцию с Entity Framework Core. Visual Studio предоставляет шаблоны проектов ASP.NET Core, автоматически создающие базовую структуру веб-приложения с настроенной маршрутизацией и внедрением зависимостей [7].

Вместе Entity Framework Core и ASP.NET Core образуют надёжный фундамент для серверной части игры, предоставляя клиенту готовые API-эндпоинты для всех операций — от регистрации пользователя до публикации уровня и обновления рекорда времени.

В качестве альтернативы рассматривалась среда разработки JetBrains Rider, которая также хорошо интегрируется с Unity. Однако Visual Studio выбрана ввиду наличия полнофункциональной бесплатной версии Community Edition для индивидуальных разработчиков, что делает её доступной при выполнении дипломного проекта без дополнительных лицензионных затрат. Кроме того, Visual Studio является стандартом для разработки на C# в экосистеме .NET, располагает наибольшим количеством обучающих материалов и имеет более широкое распространение в академической среде.

Таким образом, выбор Visual Studio в качестве среды разработки является обоснованным и обеспечивает все необходимые инструменты для эффективного

создания отказоустойчивого и производительного кода 2D-игры-конструктора уровней на языке C# с движком Unity.

3.2 Обоснование выбора языка программирования

C# — объектно-ориентированный язык программирования, разработанный компанией Microsoft. C# сочетает строгую типизацию, высокую производительность и удобство написания кода. C# является основным языком для платформы .NET и игрового движка Unity [2].

Данный выбор обусловлен следующими причинами. Во-первых, C# — родной язык для движка Unity, что обеспечивает полную интеграцию с системой физики, обработкой ввода, графикой и управлением игровыми объектами.

Во-вторых, C# относится к статически типизированным языкам, что позволяет выявлять ошибки на этапе компиляции. Это особенно важно при реализации сложных механик, таких как редактор уровней.

В-третьих, C# обладает встроенными средствами сериализации данных. Преобразование структур в формат JSON позволяет эффективно передавать уровни между клиентом и сервером, а также хранить их в базе данных.

В-четвёртых, C# поддерживает асинхронное программирование. Получение списка уровней, публикация, отправка лайков и оценок, загрузка рекордов — все эти операции не блокируют основной игровой процесс.

В-пятых, C# является кроссплатформенным языком. Совместно с Unity это позволяет компилировать продукт под Windows, Linux, macOS и мобильные платформы, что создаёт потенциал для расширения аудитории.

В качестве альтернатив рассматривались C++, JavaScript с Phaser и Python с Pygame. Ни один из перечисленных языков не обеспечивает столь же сбалансированного сочетания производительности, удобства разработки и интеграции с игровым движком.

Таким образом, выбор языка C# является оптимальным для реализации 2D-платформера с редактором уровней, обеспечивая надёжность, производительность и эффективную разработку в рамках дипломного проекта.

3.3 Обоснование выбора СУБД

Для хранения и обработки данных разрабатываемой игры в качестве системы управления базами данных выбрана PostgreSQL, а в качестве инструмента административного управления — pgAdmin 4 [3].

Выбор PostgreSQL обусловлен следующими причинами. Во-первых, PostgreSQL является объектно-реляционной СУБД с открытым исходным кодом, что исключает лицензионные отчисления и делает её доступной для использования в дипломном проекте. Отсутствие финансовых затрат на инфраструктуру данных позволяет сфокусироваться на функциональной части разработки.

Во-вторых, PostgreSQL обладает поддержкой типа данных JSON и JSONB, что критически важно для разрабатываемой игры. Каждый уровень, созданный пользователем в редакторе, представляет собой набор игровых объектов с координатами, типами и параметрами. Хранение такой структурированной информации в формате JSONB позволяет сохранять уровень как документ, не проектируя сложные реляционные схемы для каждого типа объекта. При этом JSONB обеспечивает индексирование и эффективный поиск по содержимому, что необходимо для фильтрации и сортировки уровней в каталоге [15].

В-третьих, PostgreSQL демонстрирует высокую производительность при работе с большими объёмами данных и поддерживает сложные запросы с соединениями (JOIN). В контексте игры это требуется при формировании таблиц лидеров: запрос объединяет данные из таблиц пользователей (логин) и рекордов времени, сортируя результаты по наименьшему времени прохождения конкретного уровня. Аналогичные соединения необходимы при отображении списка уровней с подтягиванием информации об авторе (логин), количества лайков и среднего рейтинга.

В-четвёртых, PostgreSQL обеспечивает целостность данных на уровне СУБД посредством внешних ключей (FOREIGN KEY) и ограничений (CHECK, UNIQUE). В разработанной схеме базы данных это реализовано через

каскадное удаление: при удалении пользователя автоматически удаляются все уровни, лайки, оценки и рекорды; ограничение уникальности гарантирует, что один пользователь не может поставить более одного лайка или оценки одному и тому же уровню; проверочное ограничение контролирует, что оценка находится в диапазоне от 1 до 5.

В-пятых, PostgreSQL поддерживает транзакционность и соответствие требованиям ACID. Это важно при выполнении операций, которые должны либо полностью выполниться, либо полностью отмениться. Например, при публикации уровня необходимо одновременно сохранить запись в таблице уровней и инициализировать счётчики лайков; в случае сбоя транзакция откатывается, и данные не оказываются в несогласованном состоянии.

В-шестых, для взаимодействия игры с базой данных PostgreSQL используется библиотека Npgsql. Npgsql написана на C# и позволяет выполнять SQL-запросы из игрового приложения. С её помощью игровой клиент может отправлять на сервер рекорды времени, получать таблицы лидеров, а также сохранять опубликованные уровни, лайки и оценки. Благодаря асинхронным методам Npgsql сетевые запросы не блокируют основной игровой процесс [10].

В качестве альтернатив рассматривались MySQL и SQLite. MySQL также является популярной реляционной СУБД с открытым кодом, но её поддержка JSON и JSONB уступает PostgreSQL по функциональности и производительности. SQLite является легковесной встраиваемой СУБД, однако не предназначена для многопользовательского доступа через сеть и не обеспечивает необходимого уровня производительности при одновременной работе большого числа игроков.

Таким образом, выбор PostgreSQL в сочетании с pgAdmin 4 является обоснованным решением для хранения пользовательских данных, уровней, лайков, оценок и рекордов, обеспечивая надёжность, производительность, целостность данных и удобство администрирования.

3.4 Проектирование визуального оформления

Разработка пользовательского интерфейса и визуальной составляющей игрового приложения выполнена с использованием графического редактора Figma. Данный инструмент выбран благодаря его доступности, кроссплатформенности, наличию бесплатного тарифа и широким возможностям для совместной работы над дизайн-проектами [16].

В Figma спроектировано полная визуальная концепция игры, включающая следующие элементы (Рисунок 3.1).

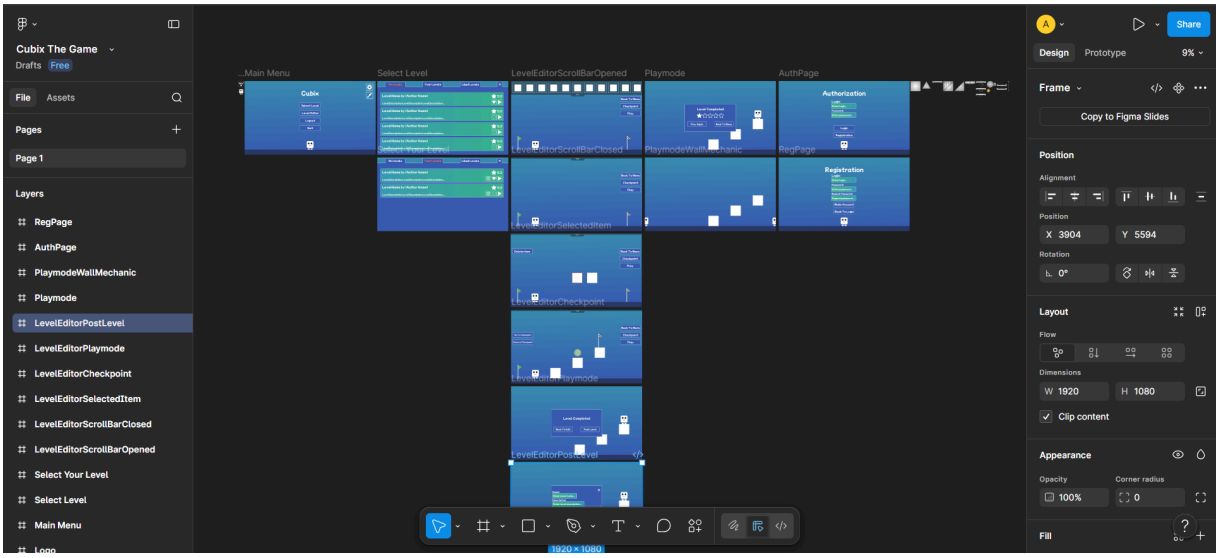

Рисунок 3.1 Примеры визуальной концепции игры в Figma

Разработан логотип приложения в узнаваемом стиле, отражающем жанр платформера. Цветовая гамма сочетает яркие акцентные цвета с нейтральным фоном для хорошей читаемости.

Создан дизайн игровых объектов: платформ, ловушек, стартового и финишного флагов, контрольных точек. Каждый объект имеет узнаваемую форму и цвет.

Разработан дизайн интерфейса редактора уровней. Окно разделено на панель инструментов и рабочую область. Кнопки крупные, снабжены иконками.

Спроектированы экраны приложения: авторизация, регистрация, главное меню, каталог уровней, таблица рекордов, настройки.

Визуальная концепция из Figma взята за основу при реализации интерфейса в Unity UI, что обеспечило целостное восприятие игры [17].

3.5 Программирование серверного приложения

Серверная часть игрового приложения «Кубикс» разработана как RESTful API на базе кроссплатформенной платформы .NET Core с использованием веб-фреймворка ASP.NET Core. Серверное приложение отвечает за обеспечение всей бизнес-логики системы: регистрацию и авторизацию пользователей, публикацию и управление уровнями, обработку лайков, оценок и рекордов времени, а также транзакционное взаимодействие с реляционной базой данных PostgreSQL.

Разработка проекта велась в интегрированной среде Microsoft Visual Studio. Для обеспечения масштабируемости, чистоты кода и соблюдения принципа разделения ответственности структура каталогов серверного приложения разделено на логические слои (Рисунок 3.5.1).

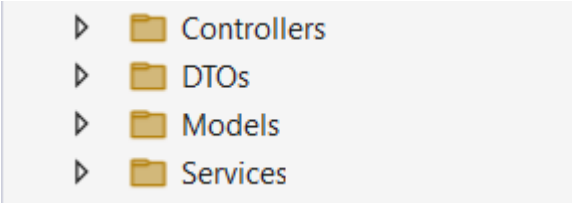

Рисунок 3.2 Структура каталогов серверного приложения

Основные структурные компоненты серверной части включают в себя:

- Controllers — слой контроллеров API, содержащий классы, унаследованные от ControllerBase. Контроллеры описывают конечные точки, принимают входящие HTTP-запросы, вызывают методы бизнес-логики и возвращают клиенту статус-коды ответов вместе с JSON-данными [5]. В проекте реализованы контроллеры: AuthController, LevelsController, RatingsController, RecordsController;

- Models — классы сущностей базы данных, описывающие таблицы: User, Level, LevelLike, LevelRating, LevelRecord. Каждая сущность содержит набор атрибутов, соответствующих полям в PostgreSQL, и навигационные свойства для связей между таблицами;

- DTOs — объекты передачи данных, используемые для валидации параметров и безопасного обмена информацией между клиентом и сервером. К ним относятся RegisterDto, LoginDto, CreateLevelDto, CreateRatingDto, CreateRecordDto и соответствующие Response-объекты;
- Services — слой сервисов бизнес-логики, изолирующий вспомогательные операции от контроллеров. В проекте реализован PasswordService, который выполняет хеширование паролей с использованием алгоритма SHA256 и их верификацию при входе пользователя [19];

Взаимодействие с базой данных реализовано посредством ORM-системы Entity Framework Core с применением подхода Code-First. Описание таблиц происходит через C#-классы, после чего миграционный движок транслирует их в SQL-код. Такой подход позволяет синхронизировать модель данных и структуру базы данных без ручного написания SQL-скриптов.

Реализация аутентификации. Для защиты пользовательских данных в системе реализован механизм хеширования паролей с использованием алгоритма SHA256. При регистрации пароль преобразуется в хеш-строку и сохраняется в базе данных, что исключает хранение паролей в открытом виде. При входе система сравнивает хеш введённого пароля с сохранённым хешем. После успешной аутентификации сервер генерирует временный токен на основе GUID, который клиент использует для авторизации при последующих запросах. Конфигурация сервисов вынесена в главный файл инициализации приложения Program.cs.

API эндпоинты. Разработанное серверное приложение предоставляет следующие группы эндпоинтов:

- AuthController: регистрация и вход пользователей;

- LevelsController: получение списка уровней, получение конкретного уровня, создание, обновление и удаление уровня, а также переключение лайков с автоматическим обновлением счётчика;
- RatingsController: выставление и обновление оценки уровня, получение среднего рейтинга и количества голосов, получение оценки конкретного пользователя;
- RecordsController: сохранение нового рекорда времени с проверкой на улучшение результата, получение таблицы лидеров (топ-10), получение личного рекорда пользователя.

Для аналитики и отображения информации в каталоге реализован рейтинг уровней. Рейтинг вычисляется как среднее арифметическое всех оценок, выставленных пользователями. Каждая оценка представляет собой целое число от 1 до 5. При добавлении новой оценки или изменении существующей среднее арифметическое пересчитывается автоматически.

Для упрощения отладки, документирования API и последующей интеграции с клиентской частью в проект внедрена библиотека Swashbuckle, которая генерирует интерактивную веб-страницу Swagger UI со списком всех эндпоинтов, типов принимаемых объектов и возможных статус-кодов ответов (Рисунок 3.2).

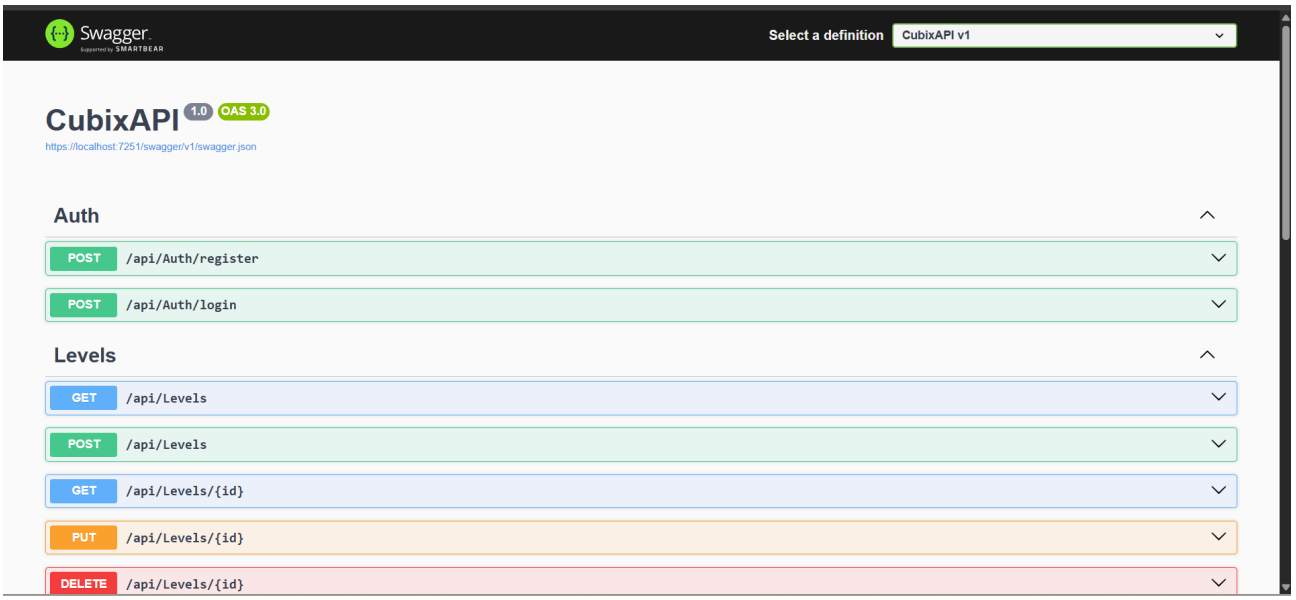

Рисунок 3.2 — Интерактивная веб-страница Swagger UI

Таким образом, разработанное серверное приложение предоставляет надёжную и безопасную платформу для обмена информацией, инкапсулирует всю работу с пользовательскими уровнями, лайками, оценками и рекордами, а также предоставляет стандартизированный программный интерфейс для клиентского приложения на Unity [18].

3.6 Программирование клиентского приложения

Создание проекта выполнено на основе технического задания приведенного в приложении А.

Клиентское приложение разработано на игровом движке Unity 2D с использованием языка C#. Создание проекта выполнено в Unity Hub [1] (Рисунок 3.3).

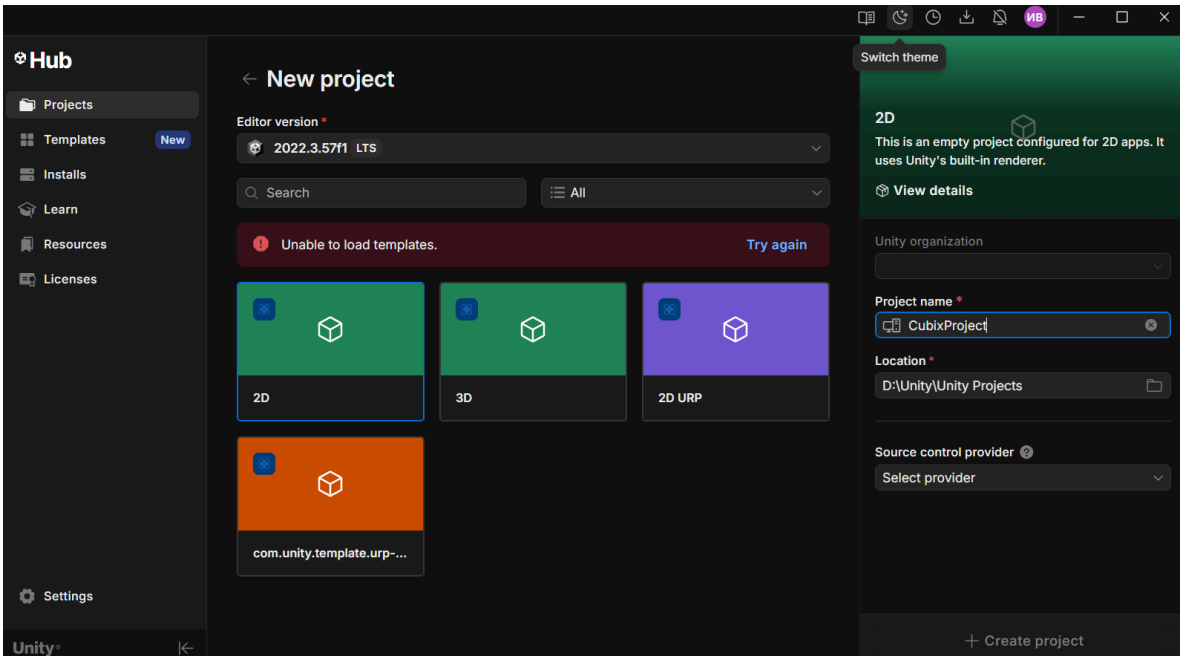

Рисунок 3.3 Создание проекта Unity

Проект имеет следующую структуру каталогов: Animations, Audio, Fonts, Other, PhysMaterial, Prefabs, Scenes, Scripts, Sprites. Такая иерархия обеспечивает удобство разработки и поддержки кода (Рисунок 3.4).

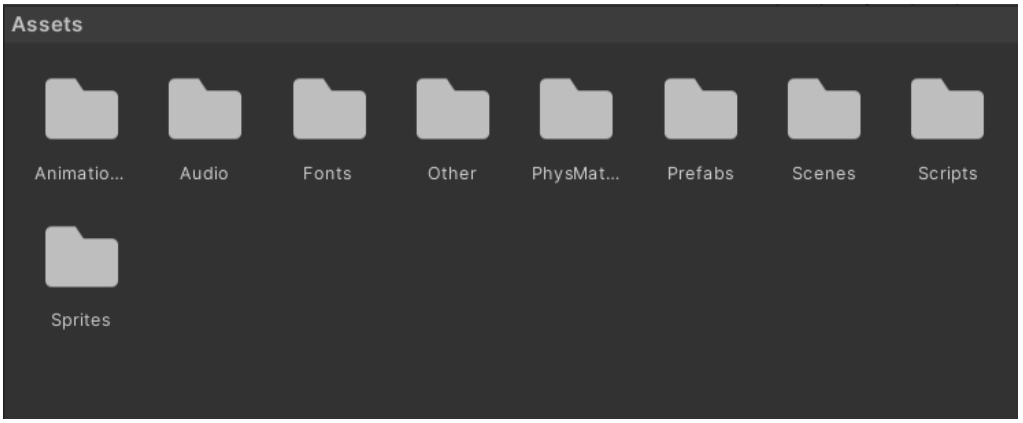

Рисунок 3.4 Создание иерархии проекта

Управление персонажем реализовано в скрипте PlayerScript.cs. Основные механики включают горизонтальное перемещение с заданной скоростью, прыжок с проверкой касания земли, а также телепортацию между границами экрана (Рисунок 3.5).

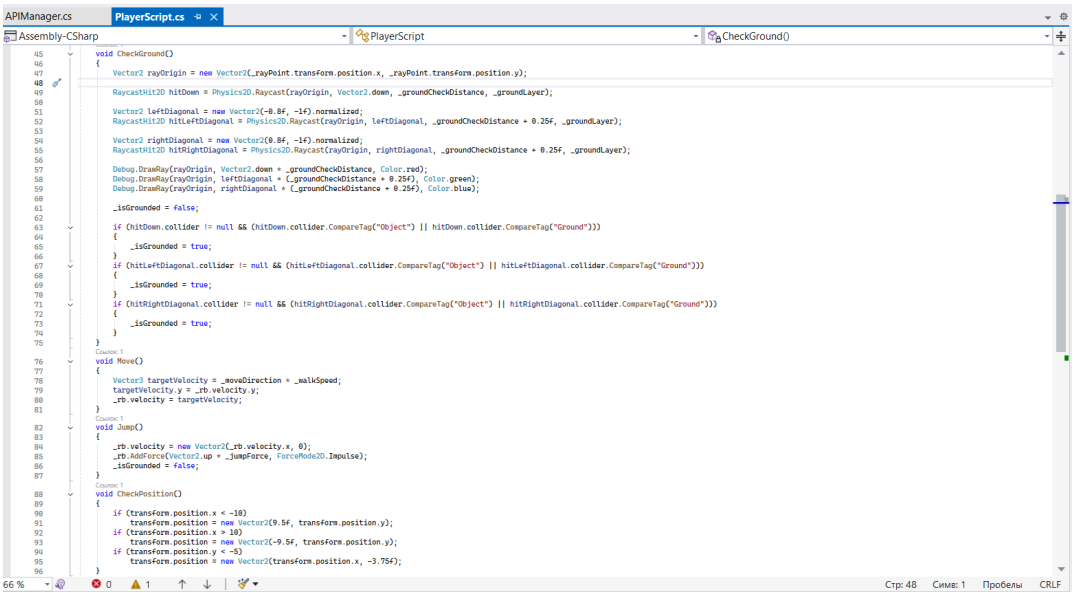

Рисунок 3.5 Скрипт игрока

Скрипт PlayerScript.cs управляет движением, прыжком и телепортацией персонажа. Проверка земли выполняется тремя лучами: прямой вниз и двумя по диагонали влево-вниз и вправо-вниз. Такой подход позволяет точно определять опору даже при движении по наклонным поверхностям или краям платформ. Если любой из лучей касается платформы или объекта, персонажу разрешается прыжок.

Перемещение реализовано в методе FixedUpdate через компонент Rigidbody2D, что обеспечивает корректное взаимодействие с физикой движка. Скорость по горизонтали задаётся отдельным параметром, а вертикальная составляющая сохраняется от текущей физической скорости, что позволяет прыжку прерываться естественным образом. Прыжок выполняется импульсным методом AddForce с предварительным обнулением вертикальной скорости, что гарантирует одинаковую высоту прыжка независимо от направления движения персонажа.

Телепортация реализована на границах экрана: при выходе за левую или правую границу персонаж мгновенно появляется с противоположной стороны без изменения вертикальной координаты. При падении вниз за пределы уровня персонаж возвращается на стартовую позицию. Метод GoToFlag предназначен для программного перемещения игрока к стартовому флагу — это используется при возрождении после гибели персонажа или при активации контрольной точки, позволяя продолжить прохождение с последнего сохранённого места.

Вся работа с серверным API вынесена в отдельный скрипт APIManager.cs, реализующий паттерн Singleton. Скрипт содержит методы для всех типов запросов: регистрация, авторизация, получение списка уровней, создание, обновление и удаление уровня, постановка лайков, выставление оценок и отправка рекордов времени [20] (Рисунок 3.6).

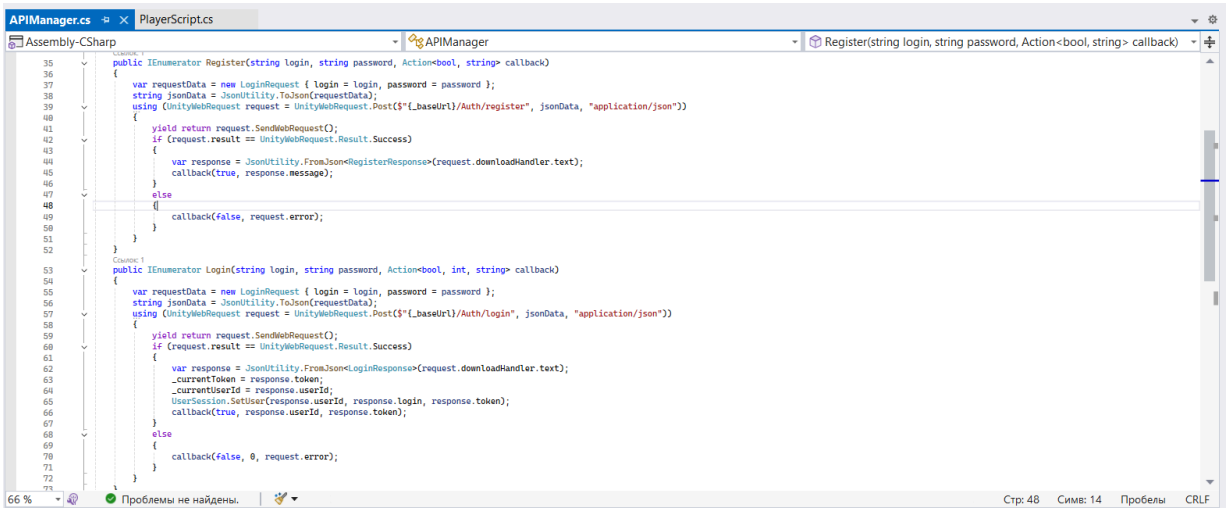

Рисунок 3.6 Скрипт для связи с REST API

Скрипт APIManager.cs выполнен в виде Singleton и содержит все методы для работы с сервером: регистрация, авторизация, создание, обновление и удаление уровней, постановка лайков, выставление оценок и отправка рекордов времени. Все запросы выполняются асинхронно через UnityWebRequest, что не блокирует основной поток игры [11].

Полная версия скрипта APIManager.cs предоставлен в приложении Б.

После успешного входа токен и идентификатор пользователя сохраняются в PlayerPrefs и автоматически восстанавливаются при следующем запуске. Для сериализации JSON используется JsonUtility [12]. Получение списка уровней

выполняется через GetAllLevels, создание нового уровня — через CreateLevel. Отправка рекорда времени — SubmitRecord, получение таблицы лидеров — GetTopRecords.

4 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

4.1 Основные понятия и принципы тестирования

Тестирование программного обеспечения — процесс проверки соответствия продукта функциональным требованиям. Основная цель — выявление ошибок и дефектов до передачи системы пользователю. Тестирование позволяет оценить качество и надёжность продукта [9].

В рамках дипломного проекта использованы следующие виды тестирования:

- модульное тестирование — проверка отдельных компонентов системы (функции, методы, классы) на корректность работы в изоляции от других модулей.
- интеграционное тестирование — проверка взаимодействия между модулями, в частности между клиентской частью на Unity и серверным API.
- функциональное тестирование — проверка соответствия реализованных функций требованиям технического задания.
- регрессионное тестирование — проверка того, что новые изменения не нарушили ранее работающую функциональность.

В работе применяются два основных метода тестирования.

Метод «белого ящика» основан на анализе внутренней структуры исходного кода. Данный метод позволяет выявить скрытые ошибки на ранних этапах разработки.

Метод «чёрного ящика» проверяет поведение системы с точки зрения конечного пользователя без доступа к внутреннему устройству. Этот метод позволяет оценить, насколько система соответствует ожиданиям пользователя.

Также в работе применяется тестирование на основе тест-кейсов — формализованных сценариев, описывающих входные данные, последовательность действий и ожидаемый результат.

Тестирование игрового приложения «Кубикс» охватывает следующие области: корректность работы редактора уровней, игровую физику и управление персонажем, сетевое взаимодействие с сервером.

4.2 Тестирование методом «Черного ящика»

Тестирование методом «чёрного ящика» основано на проверке поведения системы без доступа к внутренней структуре исходного кода. Тестировщик взаимодействует только с пользовательским интерфейсом программы, имитируя реальные сценарии использования. Данный метод позволяет оценить, насколько система соответствует функциональным требованиям и ожиданиям конечного пользователя.

В рамках тестирования игрового приложения «Кубикс» проверены следующие сценарии:

Выполнена проверка успешной регистрации нового пользователя с уникальным логином. Проверена реакция системы на попытку регистрации с уже существующим логином — система выдаёт сообщение об ошибке. Проверен вход зарегистрированного пользователя с корректными и некорректными учётными данными.

Проверена возможность размещения объектов на сцене. Проверено перемещение и удаление уже размещённых объектов. Проверено вращение объектов на 90 градусов. Проверено сохранение уровня в JSON-файл и его загрузка. Проверен тестовый забег без контрольных точек. Проверена публикация уровня после успешного тестового прохождения.

Проверено управление персонажем: ходьба влево и вправо, прыжок. Проверена работа проверки земли — прыжок невозможен в воздухе. Проверена телепортация между границами экрана. Проверена фиксация времени прохождения от старта до финиша.

Проверен просмотр списка уровней других пользователей. Проверена сортировка уровней по различным критериям. Проверено прохождение выбранного уровня.

На основе данных тестов сформирована таблица всех тестов «черного ящика» (Таблица 4.1).

Таблица 4.1 – Результаты тестирования «черного ящика»

№	Тестовый сценарий	Действия	Ожидаемый результат	Фактический результат	Статус
1	Регистрация нового пользователя	Ввести уникальный логин и пароль, нажать «Создать аккаунт»	Сообщение об успешной регистрации, перенаправление на страницу авторизации	Сообщение получено, переход выполнен	Успешно
2	Регистрация с существующим логином	Ввести логин, который уже зарегистрирован	Сообщение об ошибке «Login already exists»	Выведено сообщение об ошибке	Успешно
3	Авторизация с корректными данными	Ввести логин и пароль зарегистрированного пользователя	Вход выполнен, переход в главное меню	Вход выполнен успешно	Успешно
4	Авторизация с неверным паролем	Ввести корректный логин и неверный пароль	Сообщение «Invalid login or password»	Сообщение выведено	Успешно
5	Размещение объекта в редакторе	Выбрать блок на панели, кликнуть правой кнопкой на сцене	Объект появляется в указанном месте	Объект размещён корректно	Успешно
6	Вращение объекта	Выделить объект, нажать кнопку поворота	Объект поворачивается на 90 градусов	Поворот выполнен	Успешно
7	Удаление объекта	Выделить объект, нажать кнопку удаления	Объект исчезает со сцены	Объект удалён	Успешно
8	Сохранение уровня	Зайти в редактор, построить тестовый уровень	Файл PreviousLevel.json создан локально	Файл создан	Успешно
9	Загрузка уровня	Зайти в редактор повторно, в диалоге продолжение редактирования уровня выбрать «Да»	Ранее сохранённый уровень загружен	Уровень загружен корректно	Успешно
10	Публикация проходимого уровня	Пройти уровень в тестовом режиме, нажать «Опубликовать»	Уровень появляется в общем каталоге	Уровень опубликован	Успешно
11	Публикация непроходимого уровня	Создать уровень без финиша,	Тестовый забег невозможен без финиша	Тестовый забег невозможен	Успешно

		попробовать опубликовать			
12	Движение персонажа	Нажать клавиши A/D	Персонаж перемещается в соответствующую сторону	Движение работает	Успешно

Продолжение таблицы 4.1

13	Прыжок на земле	Находясь на платформе, нажать прыжок	Персонаж подпрыгивает	Прыжок выполнен	Успешно
14	Прыжок в воздухе	В прыжке повторно нажать прыжок	Персонаж не прыгает повторно	Прыжок не выполнен	Успешно
15	Телепортация за левый край	Дойти до левой границы экрана	Персонаж появляется справа	Появление на правой стороне	Успешно
16	Телепортация за правый край	Дойти до правой границы экрана	Персонаж появляется слева	Появление на левой стороне	Успешно
17	Постановка лайка	Нажать кнопку лайка на уровне	Счётчик лайков увеличивается на 1	Увеличение подтверждено	Успешно
18	Отмена лайка	Повторно нажать кнопку лайка	Счётчик лайков возвращается к исходному значению	Уменьшение подтверждено	Успешно
19	Выставление оценки	Выбрать 4 звезды, отправить оценку	Средний рейтинг уровня обновлён	Рейтинг обновлён	Успешно
20	Изменение оценки	Выбрать 5 звёзд вместо 4	Оценка обновлена, средний рейтинг пересчитан	Обновление выполнено	Успешно
21	Отправка рекорда времени	Пройти уровень до финиша	Время фиксируется, отправляется на сервер	Рекорд сохранён	Успешно
22	Просмотр таблицы лидеров	Открыть таблицу рекордов уровня	Отображаются лучшие времена всех игроков	Таблица отображается корректно	Успешно

Все проверенные сценарии завершились успешно. Критических ошибок и сбоев в ходе тестирования методом «чёрного ящика» не выявлено.

4.3 Тестирование методом «Test Case»

Тестирование на основе тест-кейсов представляет собой метод верификации программного обеспечения, при котором каждый проверяемый сценарий описывается в виде структурированного алгоритма. Тестовый случай

содержит уникальный идентификатор, цель проверки, последовательность действий, входные данные, ожидаемый и фактический результат.

Тест-кейсы приложения «Кубикс» охватывают ключевые сценарии использования игрового приложения: регистрацию и авторизацию пользователей, создание и публикацию уровней, прохождение уровней с фиксацией рекордов, систему лайков и оценок, а также нестандартные игровые механики, такие как телепортация между границами экрана.

Сформирована таблица с результатами тестирования (Таблица 4.2).

Таблица 4.2 – Тестирование методом «Test Case»

TC -№	Название тест-кейса	Шаги	Ожидаемый результат	Фактический результат	Статус
TC -01	Регистрация нового пользователя	1. Открыть страницу регистрации. 2. Ввести уникальный логин и пароль. 3. Нажать кнопку «Создать аккаунт».	Появляется сообщение об успешной регистрации. Пользователь перенаправляется на страницу авторизации.	Сообщение получено, переход выполнен	Успешно
TC -02	Создание и публикация уровня	1. В редакторе разместить стартовый флаг, платформы и финишный флаг. 2. Запустить чистовой забег. 3. Дойти до финиша. 4. Ввести название и описание. 5. Нажать «Опубликовать».	Уровень сохраняется на сервере и появляется в общем каталоге уровней.	Уровень опубликован, отображается в каталоге	Успешно
TC -03	Постановка лайка и оценки	1. Выбрать любой уровень в каталоге. 2. Нажать кнопку лайка. 3. Выбрать оценку 5 звёзд.	Счётчик лайков увеличивается на 1. Средний рейтинг уровня обновляется.	Лайк и оценка зафиксированы	Успешно
TC -04	Прохождение уровня и установка рекорда	1. Выбрать уровень в каталоге. 2. Пройти уровень от старта до финиша. 3. Завершить уровень.	Время прохождения фиксируется. Рекорд отправляется на сервер. Таблица лидеров обновляется.	Время сохранено, рекорд отображается в таблице	Успешно
TC -05	Телепортация между	1. Запустить любой уровень.	При выходе за левую границу персонаж	Телепортация работает корректно в	Успешно

	границам и экрана	2. Управлять персонажем влево до выхода за левую границу экрана. 3. Управлять персонажем вправо до выхода за правую границу экрана.	появляется справа. При выходе за правую границу — слева. Высота персонажа не меняется.	обе стороны	

Все тест-кейсы успешно пройдены. Критических ошибок не выявлено.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Страница авторизации и регистрации

Запуская игровой проект «Кубикс», пользователя встречает страница авторизации. На странице размещены поля для ввода логина и пароля, а также кнопки «Войти в аккаунт» и «Регистрация». Заполнив поля корректно и нажав «Войти в аккаунт», пользователь перейдет в главное меню игры (Рисунок 5.1).

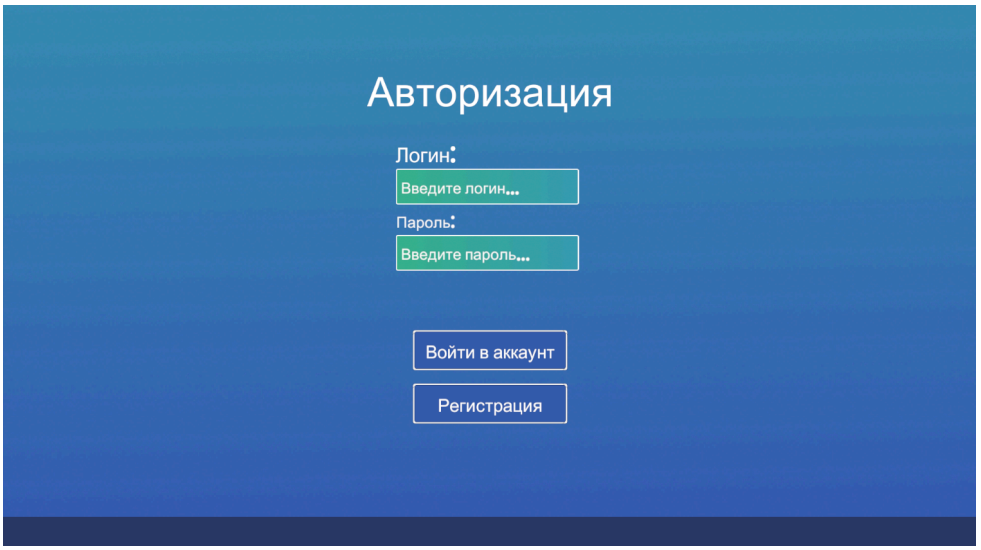

Рисунок 5.1 Страница авторизации

На странице регистрации расположены поля для ввода логина и пароля, кнопка «Создать аккаунт» и кнопка «Вернуться к авторизации» (Рисунок 5.2).

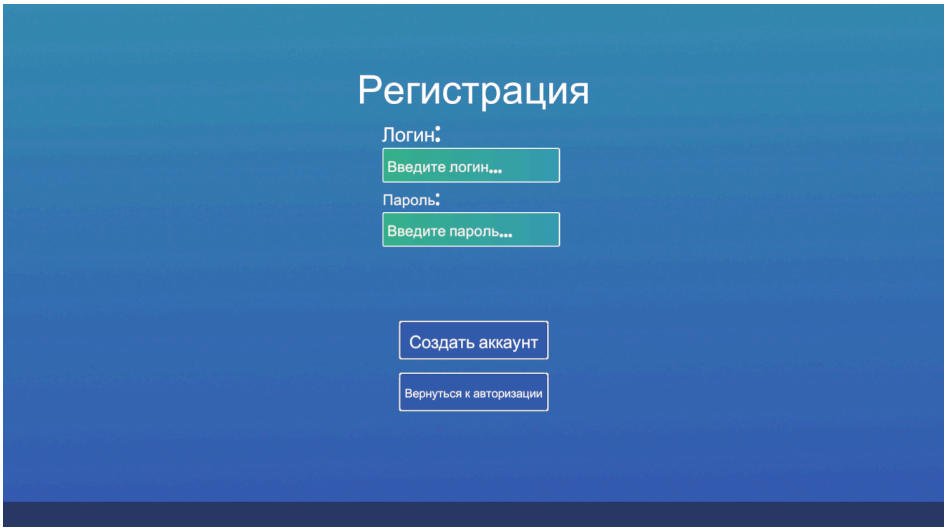

Рисунок 5.2 Страница регистрации

После заполнения полей и нажатия на «Создать аккаунт» пользователь переносится на страницу авторизации для входа в новый аккаунт. Кнопка «Вернуться к авторизации» также возвращает на страницу входа

5.2 Главное меню

Войдя в аккаунт пользователь видит интерфейс главного меню, на котором расположены название игры и кнопки «Выбор уровня», «Редактор уровня», «Выйти из аккаунта», «Выйти из игры» и кнопка настройки в виде иконки шестеренки (Рисунок 5.3).

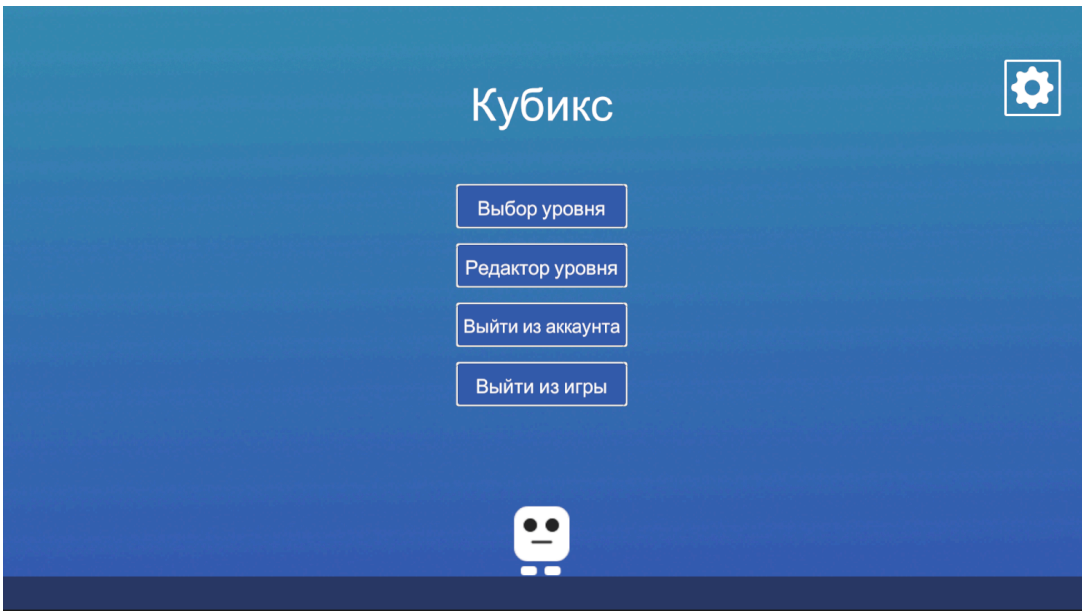

Рисунок 5.3 Главное меню

Нажав на шестеренку, игрок открывает панель настроек громкости звуков и музыки. Для регулировки используются слайдеры. Настройки сохраняются локально и применяются сразу при изменении значений. Кнопка «Закрыть» скрывает панель (Рисунок 5.4).

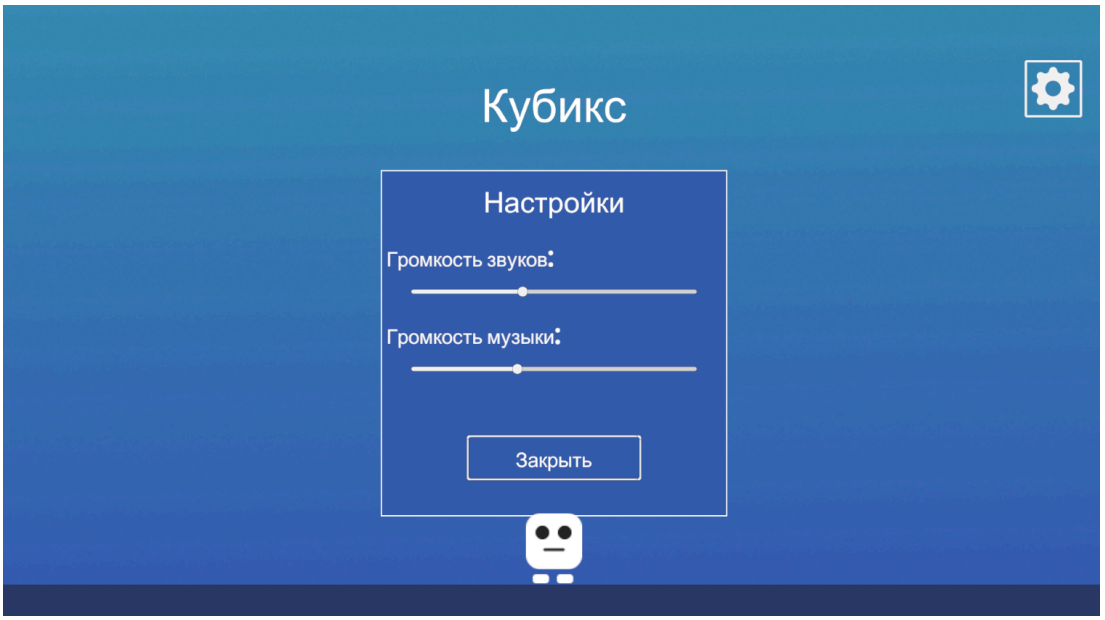

Рисунок 5.4 Панель настроек

Нажав на кнопку «Выбор уровня» открывается каталог уровней. Пользователь видит карточки уровней, на которых отображены: название, автор, описание, рейтинг, кнопки играть, лайк и просмотр таблицы лидеров. Так же есть поиск и сортировка уровней. Есть возможность посмотреть уровни сделанные игроком и уровни, которым пользователь поставил лайк (Рисунок 5.5).

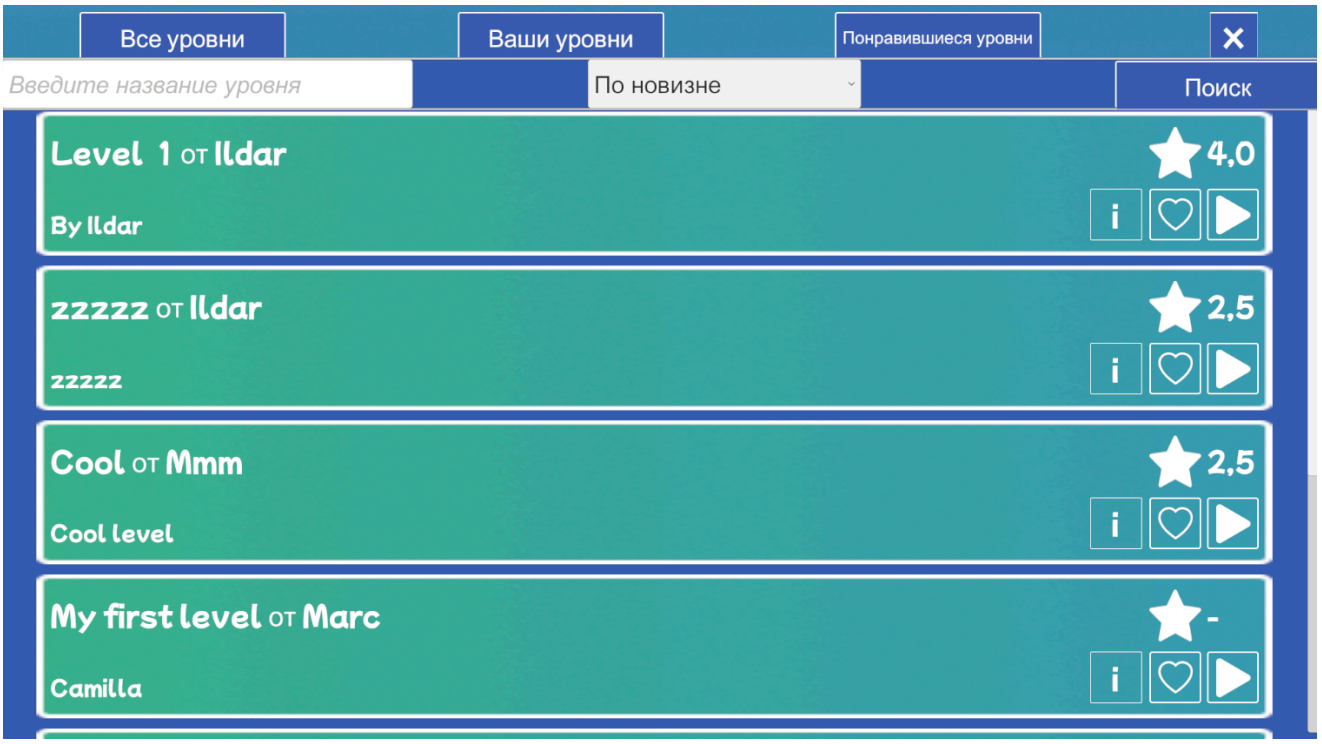

Рисунок 5.5 Каталог уровней

В данном каталоге пользователь может выбрать интересующий его уровень и поиграть, после чего в этом же каталоге поставить лайк уровню.

5.3 Редактор уровня

Выбрав «Редактор уровня» в главном меню, пользователь переходит на сцену редактора. На экране отображаются панель блоков, кнопки «Вернуться в меню», «Точка возврата» и «Тестовый забег». На сцене присутствуют игрок, зелёный флаг старта и черно-белый флаг финиша. Персонаж может прыгать, ходить в стороны и телепортироваться при выходе за границы экрана. При получении урона или падении в пропасть игрок возвращается на стартовый флаг.

Цель игрока — создать маршрут от старта до финиша, используя предоставленные блоки и препятствия. Для установки объекта необходимо левой кнопкой мыши выбрать блок на верхней панели, затем правой кнопкой мыши указать место на сцене (Рисунок 5.6).

Рисунок 5.6 Редактор уровня

Поставив игровой объект, игрок может выделить конкретный игровой объект. Выделенный объект окрашивается в желтый цвет и зажатой левой

кнопкой мыши может передвигаться. Также появляются кнопки управления объектом: «Убрать выделение», «Удалить», «Повернуть» (Рисунок 5.7).

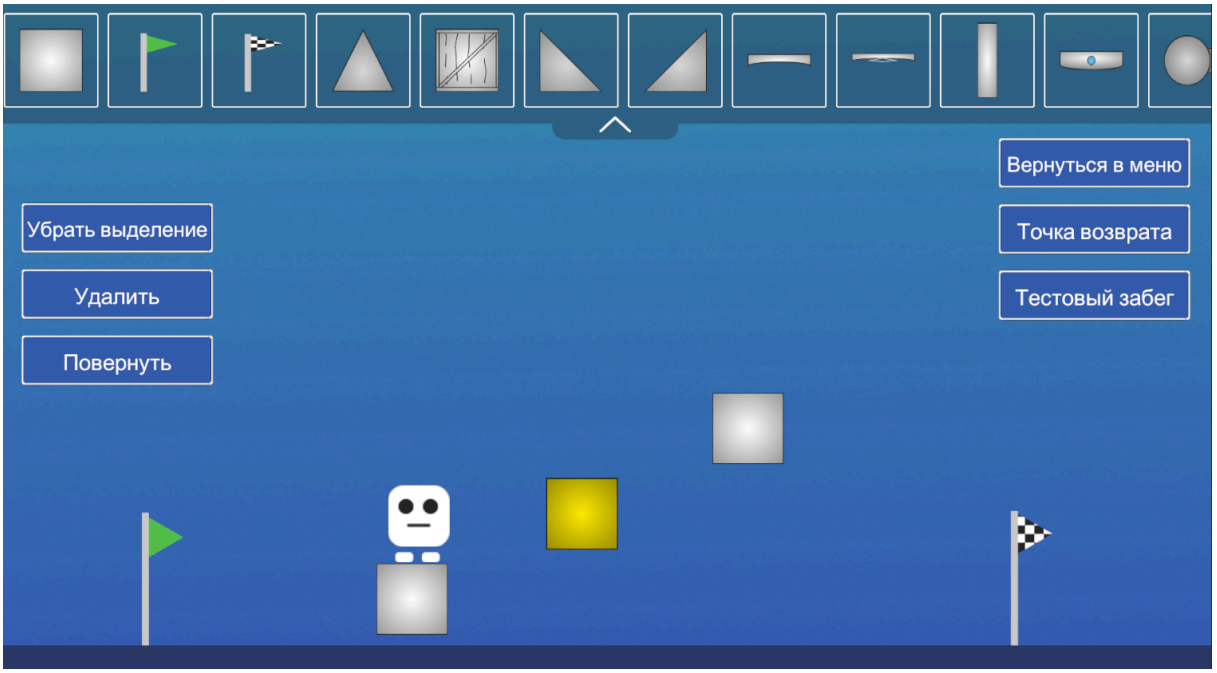

Рисунок 5.7 Действия над объектом

Для более удобного тестирования уровня можно поставить точку возврата, нажав на соответствующую кнопку. На карте появится желтая точка и кнопки «Вернуться к точке» и «Убрать точку» (Рисунок 5.8).

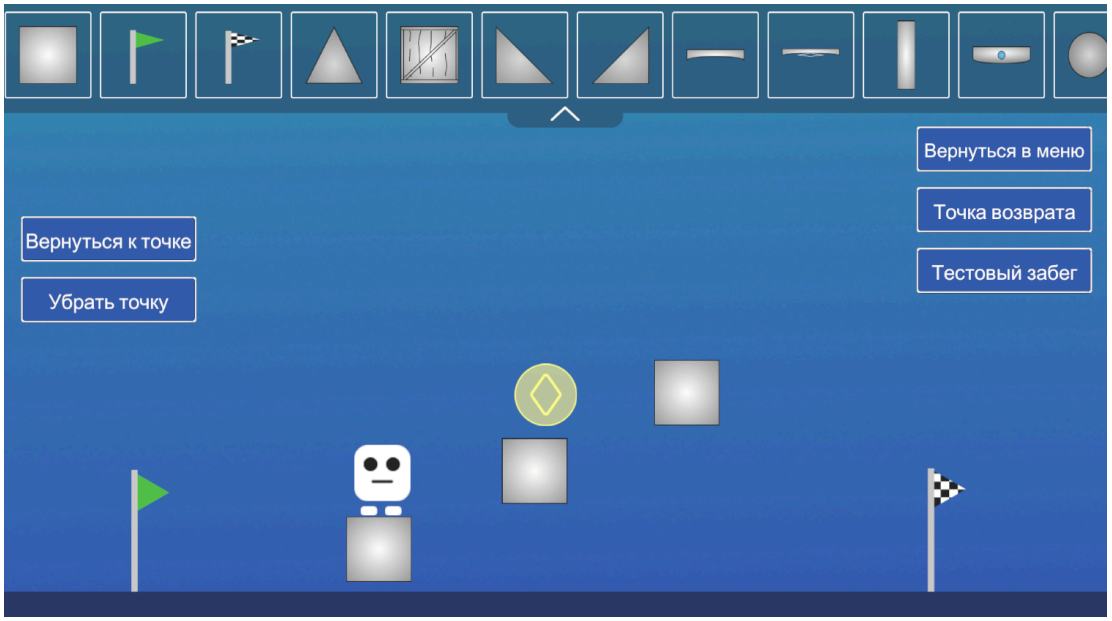

Рисунок 5.8 Действия с точкой возврата

При заходе на сцену редактора уровня если у игрока уже есть начатый уровень, то в начале выйдет панель с вопросом «Хотите ли вы редактировать предыдущий уровень?» и кнопки «Да» и «Нет» (Рисунок 5.9).

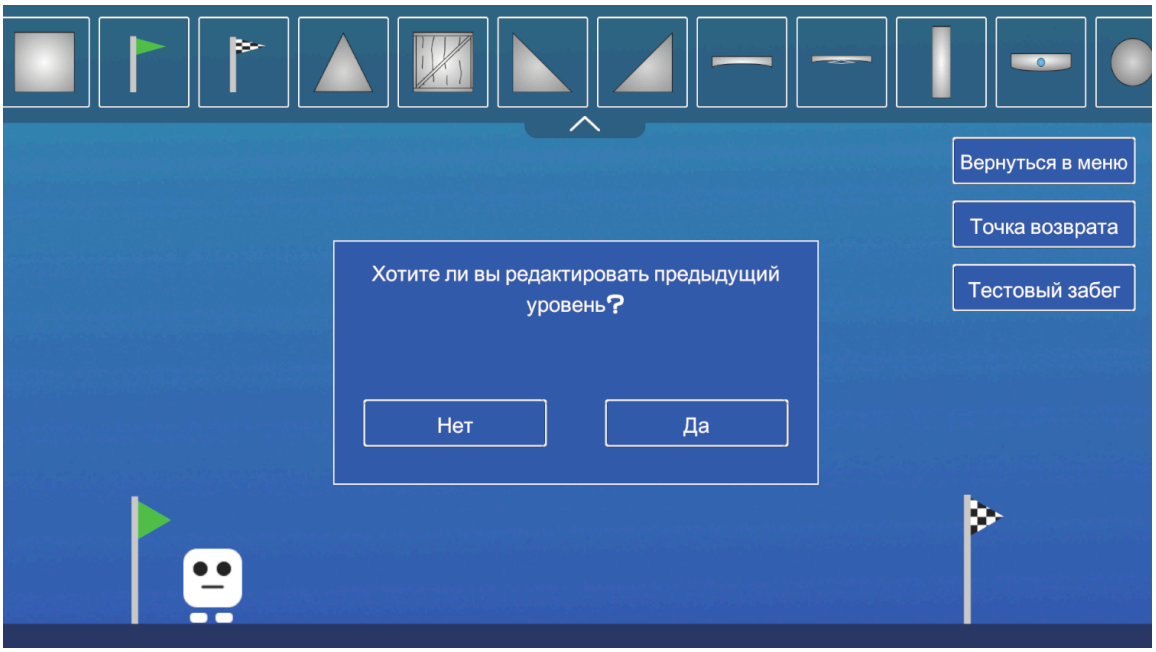

Рисунок 5.9 Диалог продолжения редактирования

При выборе «Да» загрузится предыдущий уровень, при выборе «Нет» загрузится пустая сцена, где можно создать новый уровень.

Построив уровень, игрок может нажать «Тестовый забег», где открывается сцена, в которой загружается созданный уровень. На интерфейсе появляется кнопка «Вернуться к редактированию», которая возвращает игрока к редактированию уровня (Рисунок 5.10).

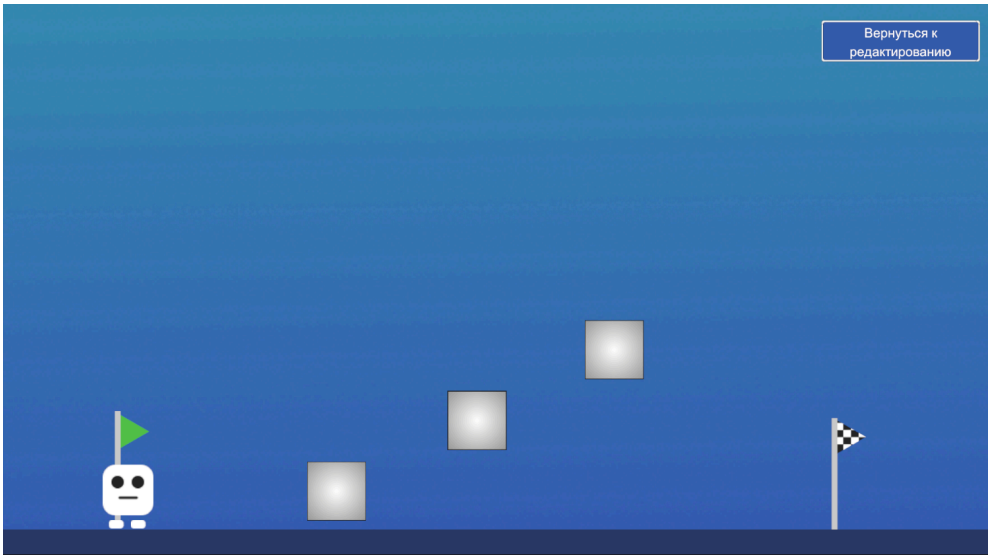

Рисунок 5.10 Тестовый забег

Цель игрока — добраться от флага старта до флага финиша без использования точек возврата. Если уровень успешно пройден, на экране появляется панель с полями для ввода названия и описания уровня, а также кнопка «Опубликовать» (Рисунок 5.11).

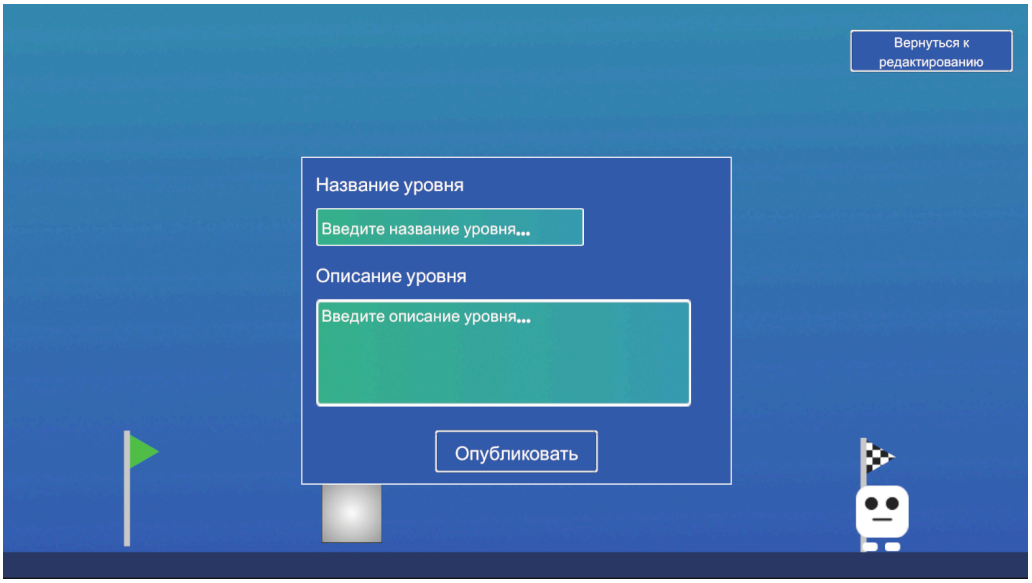

Рисунок 5.11 Диалог публикации уровня

Введя название и описания уровня можно нажать кнопку «Опубликовать». Опубликованный уровень попадает в каталог уровней.

5.4 Игровой процесс

Выбрав уровень и нажав играть в каталоге уровней, игрок переходит на сцену, где появляется созданный автором маршрут. На интерфейсе отображается таймер (Рисунок 5.12)

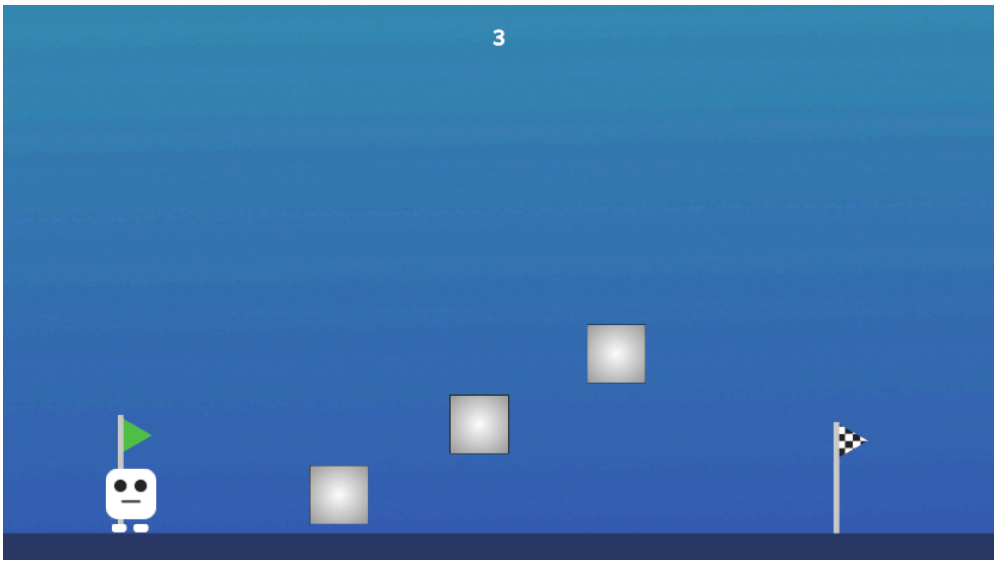

Рисунок 5.12 Игровой процесс

Цель игрока — добраться от флага старта до флага финиша. Дойдя до конца уровня открывается панель с кнопками «Играть заново», «Вернуться в меню» и звездным рейтингом (Рисунок 5.13)

Рисунок 5.13 Панель завершения уровня

Уровень оценивается от 1 до 5. Рейтинг сохраняется в базе данных. При повторном открытии панели оценок показывается предыдущее выбранное пользователем значение.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта разработана и реализована 2D-игра-конструктор уровней на движке Unity с использованием языка программирования C# и базы данных PostgreSQL. В процессе разработки реализованы следующие функции: регистрация и авторизация пользователей; настройка игры; визуальный редактор уровней; механика горизонтального зацикливания пространства, при которой персонаж, выходящий за правый край экрана, появляется слева, и наоборот; система контрольных точек для отладки уровня в процессе создания; режим чистового прохождения для проверки уровня на проходимость перед публикацией; публикация успешно пройденных уровней на сервере с возможностью указания названия и описания.

Также в игре реализован каталог опубликованных уровней других игроков с возможностью прохождения, постановки лайков и выставления числовых оценок по шкале от одного до пяти. Для каждого уровня ведётся таблица рекордов, отображающая лучшие времена прохождения всех игроков, что формирует соревновательную составляющую. Пользователь может проходить один уровень многократно, улучшая свой результат и поднимаясь выше в таблице лидеров.

Разработанная игра обладает интуитивно понятным интерфейсом, построенным на основе Unity UI. Программный продукт прошёл все основные этапы жизненного цикла: анализ предметной области, исследование существующих аналогов, проектирование схемы базы данных, реализацию клиентской и серверной частей, а также функциональное тестирование. Тестирование подтвердило стабильность и корректность работы всех реализованных механик, критических ошибок и сбоев не выявлено.

Серверное приложение построено на платформе ASP.NET Core с использованием Entity Framework Core для взаимодействия с PostgreSQL. Реализованы контроллеры для регистрации, авторизации, CRUD уровней, лайков, оценок и рекордов времени. Для хранения структуры уровня применён тип данных JSONB. Хеширование паролей выполнено с использованием

алгоритма SHA256. Авторизация пользователей поддерживается через временные GUID-токены. Все API-эндпоинты поддерживают асинхронную обработку запросов, что обеспечивает отзывчивость сервера при одновременной работе нескольких клиентов.

Игровой клиент реализован на Unity 2D. Визуальный редактор уровней поддерживает размещение, перемещение, вращение и удаление объектов. Сохранение и загрузка уровней выполняются в формате JSON. Игровой процесс включает управление персонажем с проверкой земли через лучи, плавное следование камеры и телепортацию между границами экрана. Каталог уровней интегрирован с серверным API через UnityWebRequest. Все сетевые запросы выполняются асинхронно, не создавая задержек в основном игровом процессе. Контрольные точки ускоряют отладку сложных участков, а режим чистового прохождения гарантирует публикацию только проходимых уровней.

Проведено тестирование методами белого и чёрного ящика. Модульные тесты проверили корректность сериализации, поворота объектов, условий телепортации и уникальности лайков. Функциональное тестирование подтвердило работоспособность всех сценариев использования. Составлены тест-кейсы, каждый из которых успешно пройден. Разработана документация для конечного пользователя и руководства для разработчика, SCRUM-мастера и владельца проекта. Визуальная концепция игры создана в графическом редакторе Figma, включая логотип, дизайн игровых объектов и интерфейс.

Разработанная 2D-игра-конструктор уровней полностью соответствует поставленным требованиям технического задания и готова использоваться игровым сообществом для создания, публикации и соревновательного прохождения пользовательских уровней.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unity Documentation — URL: https://docs.unity.com
2. Microsoft C# Documentation — URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/
3. PostgreSQL 14 Documentation — URL: https://www.postgresql.org/docs/14/
4. Microsoft Visual Studio Documentation — URL: https://docs.microsoft.com/ru-ru/visualstudio/
5. JSON Specification (RFC 8259) — URL: https://tools.ietf.org/html/rfc8259
6. Entity Framework Core Documentation — URL: https://docs.microsoft.com/ru-ru/ef/core/
7. ASP.NET Core Documentation — URL: https://docs.microsoft.com/ru-ru/aspnet/core/
8. UML 2.5.1 Specification — URL: https://www.omg.org/spec/UML
9. Unity Test Framework Documentation — Текст: электронный // Unity Documentation. — URL: https://docs.unity3d.com/Manual/testing-editortestsrunner.html
10. Npgsql .NET Data Provider for PostgreSQL — URL: https://www.npgsql.org
11. UnityWebRequest Documentation — URL: https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html
12. JsonUtility API Reference — URL: https://docs.unity3d.com/ScriptReference/JsonUtility.html
13. Physics2D.Raycast Documentation — URL: https://docs.unity3d.com/ScriptReference/Physics2D.Raycast.html
14. JSON Serialization in Unity — URL: https://docs.unity3d.com/Manual/JSONSerialization.html
15. PostgreSQL JSON Types Documentation — URL: https://www.postgresql.org/docs/current/datatype-json.html

16. Figma Help Documentation — Текст: электронный // Figma Help Center — URL: https://help.figma.com/hc/en-us

17. Unity UI (uGUI) Documentation — Текст: электронный // Unity Manual — URL: https://docs.unity3d.com/Manual/UI.html

18. Unity Client-Server Communication Guide — URL: https://learn.unity.com/tutorial/client-server-communication

19. Password Hashing with SHA256 in C# — URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.security.cryptography.sha256

20. Coroutines in Unity Documentation — URL: https://docs.unity3d.com/ScriptReference/Coroutine.html

ПРИЛОЖЕНИЕ А

ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ИГРОВОГО ПРИЛОЖЕНИЯ «КУБИКС»

1. Общие сведения

1.1. Наименование работы: Проектирование и разработка игрового приложения «Кубикс».

1.2. Краткое наименование программного продукта: Игра «Кубикс».

1.3. Исполнитель: Студент группы 421П Валиуллин И.И.

1.4. Сроки выполнения проекта: 01.04.2026 – 13.06.2026.

2. Назначение и цели создания системы

2.1. Назначение системы: 2D-платформер со встроенным редактором уровней для создания, публикации и соревновательного прохождения пользовательского контента.

2.2. Цели проекта:

Предоставить пользователям инструмент для самостоятельного проектирования игровых уровней.

Сформировать сообщество вокруг публикации и оценки пользовательских уровней.

Реализовать соревновательную механику через таблицы рекордов времени.

3. Требования к программе и функционалу

3.1. Регистрация и авторизация:

Регистрация нового пользователя с уникальным логином.

Авторизация существующего пользователя.

Хеширование паролей перед сохранением в базу данных.

3.2. Редактор уровней:

Размещение объектов из готовых префабов: платформы, ловушки, стартовый флаг, финишный флаг, контрольные точки.

Перемещение, удаление и вращение объектов на 90 градусов.

Сохранение уровня в JSON-файл локально.

Загрузка ранее сохранённого уровня.

Тестовый забег с контрольными точками для отладки.

Чистовой забег без контрольных точек для проверки проходимости.

Публикация уровня на сервере после успешного чистового прохождения.

3.3. Игровой процесс:

Управление: ходьба влево/вправо, прыжок.

Проверка касания земли с помощью лучей.

Плавное следование камеры за игроком.

Телепортация между левым и правым краями экрана.

Фиксация времени прохождения от старта до финиша.

3.4. Каталог уровней:

Просмотр списка уровней других пользователей.

Сортировка уровней по популярности, дате, рейтингу.

Прохождение выбранного уровня.

Повторное прохождение для улучшения результата.

3.5. Оценки и рекорды:

Постановка лайка уровню (один пользователь — один лайк).

Выставление оценки от 1 до 5 (одна оценка на пользователя).

Отображение среднего рейтинга уровня.

Таблица рекордов с лучшими временами всех игроков.

Автоматическое обновление личного рекорда при улучшении времени.

3.6. Настройки:

Регулировка громкости звуков и музыки.

Выбор разрешения экрана.

Локальное сохранение настроек.

4. Требования к видам обеспечения

4.1. Техническое и программное обеспечение:

Клиент: Unity 2D, язык C#.

Сервер: ASP.NET Core REST API.

База данных: PostgreSQL (поле JSONB для хранения уровней).

Сетевые запросы: UnityWebRequest, асинхронный режим.

4.2. Интерфейс пользователя:

Интерфейс на базе Unity UI.

Интуитивно понятное управление редактором: размещение объектов одним кликом, перемещение и удаление минимальным числом действий.

Визуальная концепция разрабатывается в Figma.

ПРИЛОЖЕНИЕ Б

```
using System;
using System.Collections;
using
System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class APIManager :
MonoBehaviour
{
    public static APIManager
Instance;
  [SerializeField] private string
_baseUrl =
"http://localhost:7251/api";
  private string _currentToken;
  private int _currentUserId;
  private void Awake()
  {
    if (Instance == null)
    {
      Instance = this;

DontDestroyOnLoad(gameObject
);
    }
    else
    {
      Destroy(gameObject);
    }
  }
  private void Start()
  {
    if (_currentUserId == 0 &&
PlayerPrefs.HasKey("UserId"))
    {
      _currentUserId =
PlayerPrefs.GetInt("UserId");
      _currentToken =
PlayerPrefs.GetString("UserToke
n");

Debug.Log($"APIManager
auto-restored:
UserId={_currentUserId}");
    }
  }
  #region Auth
    public IEnumerator
Register(string login, string
password, Action<bool, string>
callback)
  {
    var requestData = new
LoginRequest { login = login,
password = password };
    string jsonData =
JsonUtility.ToJson(requestData);
    using (UnityWebRequest
request =
UnityWebRequest.Post($"{_base
Url}/Auth/register", jsonData,
"application/json"))
    {
      yield return
request.SendWebRequest();
      if (request.result ==
UnityWebRequest.Result.Success
)
      {
        var response =
JsonUtility.FromJson<RegisterRe
sponse>(request.downloadHandle
r.text);
        callback(true,
response.message);
      }
      else
      {
        callback(false,
request.error);
      }
    }
  }
    public IEnumerator
Login(string login, string
password, Action<bool, int,
string> callback)
  {
    var requestData = new
LoginRequest { login = login,
password = password };
    string jsonData =
JsonUtility.ToJson(requestData);
    using (UnityWebRequest
request =
UnityWebRequest.Post($"{_base
Url}/Auth/login", jsonData,
"application/json"))
    {
      yield return
request.SendWebRequest();
      if (request.result ==
UnityWebRequest.Result.Success
)
      {
        var response =
JsonUtility.FromJson<LoginResp
onse>(request.downloadHandler.t
ext);
        _currentToken =
response.token;
        _currentUserId =
response.userId;

UserSession.SetUser(response.us
erId, response.login,
response.token);
        callback(true,
response.userId, response.token);
      }
      else
      {
```

```
                callback(false, 0, request.error);
            }
        }
    }

    #endregion
    #region Levels
        public IEnumerator GetAllLevels(Action<bool, List<LevelResponse>> callback)
    {
        using (UnityWebRequest request = UnityWebRequest.Get($"{_baseUrl}/Levels"))
        {
            yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
            {
                string json = "{\"levels\":" + request.downloadHandler.text + "}";
                var wrapper = JsonUtility.FromJson<LevelListWrapper>(json);
                callback(true, wrapper.levels);
            }
            else
            {
                callback(false, null);
            }
        }
    }
        public IEnumerator GetLevelById(int levelId, Action<bool, LevelResponse> callback)
    {
        using (UnityWebRequest request = UnityWebRequest.Get($"{_baseUrl}/Levels/{levelId}"))
        {
            yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
            {
                var level = JsonUtility.FromJson<LevelResponse>(request.downloadHandler.text);
                callback(true, level);
            }
            else
            {
                callback(false, null);
            }
        }
    }
        public IEnumerator CreateLevel(string name, string description, string levelDataJson, Action<bool, int, string> callback)
    {
        if (_currentUserId == 0)
        {
            int savedUserId = PlayerPrefs.GetInt("UserId", 0);
            if (savedUserId != 0)
            {
                _currentUserId = savedUserId;
                Debug.Log($"Restored UserId from PlayerPrefs in CreateLevel: {_currentUserId}");
            }
            else
            {
                callback(false, 0, "User not logged in");
                yield break;
            }
        }

        var requestData = new CreateLevelRequest { name = name, description = description, levelDataJson = levelDataJson };
        string jsonData = JsonUtility.ToJson(requestData);
        using (UnityWebRequest request = UnityWebRequest.Post($"{_baseUrl}/Levels?userId={_currentUserId}", jsonData, "application/json"))
        {
            yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
            {
                var response = JsonUtility.FromJson<CreateLevelResponse>(request.downloadHandler.text);
                callback(true, response.id, response.message);
            }
            else
            {
                callback(false, 0, request.error);
```

```
        }
    }
}
        public IEnumerator UpdateLevel(int levelId, string name, string description, string levelDataJson, Action<bool, string> callback)
{
        var requestData = new UpdateLevelRequest { name = name, description = description, levelDataJson = levelDataJson };
        string jsonData = JsonUtility.ToJson(requestData);
        using (UnityWebRequest request = new UnityWebRequest($"{_baseUrl}/Levels/{levelId}?userId={_currentUserId}", "PUT"))
    {
        byte[] bodyRaw = System.Text.Encoding.UTF8.GetBytes(jsonData);
        request.uploadHandler = new UploadHandlerRaw(bodyRaw);
        request.downloadHandler = new DownloadHandlerBuffer();
request.SetRequestHeader("Content-Type", "application/json");
        yield return request.SendWebRequest();
        if (request.result == UnityWebRequest.Result.Success)
        {
            var response = JsonUtility.FromJson<MessageResponse>(request.downloadHandler.text);
            callback(true, response.message);
        }
        else
        {
            callback(false, request.error);
        }
    }
}
        public IEnumerator DeleteLevel(int levelId, Action<bool, string> callback)
{
        using (UnityWebRequest request = UnityWebRequest.Delete($"{_baseUrl}/Levels/{levelId}?userId={_currentUserId}"))
    {
        yield return request.SendWebRequest();
        if (request.result == UnityWebRequest.Result.Success)
        {
            var response = JsonUtility.FromJson<MessageResponse>(request.downloadHandler.text);
            callback(true, response.message);
        }
        else
        {
            callback(false, request.error);
        }
    }
}
        public IEnumerator LikeLevel(int levelId, Action<bool, int> callback)
{
        using (UnityWebRequest request = UnityWebRequest.Post($"{_baseUrl}/Levels/{levelId}/like?userId={_currentUserId}", "", "application/json"))
    {
        yield return request.SendWebRequest();
        if (request.result == UnityWebRequest.Result.Success)
        {
            var response = JsonUtility.FromJson<LikeResponse>(request.downloadHandler.text);
            callback(true, response.likesCount);
        }
        else
        {
            callback(false, 0);
        }
    }
}
#endregion
#region Ratings
        public IEnumerator RateLevel(int levelId, int rating, Action<bool, double> callback)
{
        Debug.Log($"RateLevel called: levelId={levelId}, rating={rating}, _currentUserId={_currentUserId}");
```

```
        var requestData = new RatingRequest { rating = rating };
        string jsonData = JsonUtility.ToJson(requestData);
        using (UnityWebRequest request = UnityWebRequest.Post($"{_baseUrl}/Ratings/level/{levelId}?userId={_currentUserId}", jsonData, "application/json"))
        {
            yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
            {
                var response = JsonUtility.FromJson<RatingResponse>(request.downloadHandler.text);
                callback(true, response.averageRating);
            }
            else
            {
                callback(false, 0);
            }
        }
    }
    public IEnumerator GetLevelRatings(int levelId, Action<bool, double, int> callback)
    {
        using (UnityWebRequest request = UnityWebRequest.Get($"{_baseUrl}/Ratings/level/{levelId}"))
        {
            yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
            {
                var response = JsonUtility.FromJson<RatingResponse>(request.downloadHandler.text);
                callback(true, response.averageRating, response.totalVotes);
            }
            else
            {
                callback(false, 0, 0);
            }
        }
    }
    public IEnumerator GetUserRating(int levelId, Action<bool, int> callback)
    {
        using (UnityWebRequest request = UnityWebRequest.Get($"{_baseUrl}/Ratings/level/{levelId}/user/{_currentUserId}"))
        {
            yield return request.SendWebRequest();

            if (request.result == UnityWebRequest.Result.Success)
            {
                var response = JsonUtility.FromJson<UserRatingResponse>(request.downloadHandler.text);
                callback(true, response.rating);
            }
            else
            {
                callback(false, 0);
            }
        }
    }
    #endregion
    #region Records
    public IEnumerator SubmitRecord(int levelId, int timeMs, Action<bool, bool, List<RecordResponse>> callback)
    {
        var requestData = new RecordRequest { timeMs = timeMs };
        string jsonData = JsonUtility.ToJson(requestData);
        using (UnityWebRequest request = UnityWebRequest.Post($"{_baseUrl}/Records/level/{levelId}?userId={_currentUserId}", jsonData, "application/json"))
        {
            yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
            {
                var response = JsonUtility.FromJson<RecordSubmitResponse>(request.downloadHandler.text);
                callback(true, response.isNewRecord, response.bestRecords);
            }
            else
            {
```

```
            callback(false, false, null);
        }
    }
  }
        public IEnumerator GetTopRecords(int levelId, int limit, Action<bool, List<RecordResponse>> callback)
  {
        using (UnityWebRequest request = UnityWebRequest.Get($"{_baseUrl}/Records/level/{levelId}/top?limit={limit}"))
    {
                yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
        {
                string json = "{\"records\":" + request.downloadHandler.text + "}";
                var wrapper = JsonUtility.FromJson<RecordListWrapper>(json);
                callback(true, wrapper.records);
        }
        else
        {
          callback(false, null);
        }
    }
  }
        public IEnumerator GetUserRecord(int levelId, Action<bool, int, string> callback)
  {
        using (UnityWebRequest request = UnityWebRequest.Get($"{_baseUrl}/Records/level/{levelId}/user/{_currentUserId}"))
    {
                yield return request.SendWebRequest();
            if (request.result == UnityWebRequest.Result.Success)
        {
                var response = JsonUtility.FromJson<UserRecordResponse>(request.downloadHandler.text);
            if (response.timeMs != -1)
            {
                callback(true, response.timeMs, response.achievedAt);
            }
            else
            {
                callback(true, -1, null);
            }
        }
        else
        {
          callback(false, 0, null);
        }
    }
  }
  #endregion
  #region Helpers
    public void SetToken(string token)
  {
    _currentToken = token;
  }
    public void SetUserId(int userId)
  {
    _currentUserId = userId;
  }
  public int GetUserId()
  {
    return _currentUserId;
  }
  public bool IsLoggedIn()
  {
                return !string.IsNullOrEmpty(_currentToken);
  }
  public void Logout()
  {
    _currentToken = null;
    _currentUserId = 0;
  }
  #endregion
}
#region Request/Response Classes
[System.Serializable]
class LoginRequest
{
  public string login;
  public string password;
}
[System.Serializable]
class RegisterResponse
{
  public string message;
  public int userId;
}
[System.Serializable]
class LoginResponse
{
```

```
    public int userId;
    public string login;
    public string token;
}
[System.Serializable]
public class LevelResponse
{
    public int id;
    public string name;
    public string description;
    public int authorId;
    public string authorLogin;
    public string levelDataJson;
    public int likesCount;
    public double averageRating;
    public string createdAt;
}
[System.Serializable]
class LevelListWrapper
{
        public  List<LevelResponse>
levels;
}
[System.Serializable]
class CreateLevelRequest
{
    public string name;
    public string description;
    public string levelDataJson;
}
[System.Serializable]
class CreateLevelResponse
{
    public int id;
    public string message;
}
[System.Serializable]
class UpdateLevelRequest
{
    public string name;
    public string description;
    public string levelDataJson;
}
[System.Serializable]
class MessageResponse
{
    public string message;
}
[System.Serializable]
class LikeResponse
{
    public int likesCount;
}
[System.Serializable]
class RatingRequest
{
    public int rating;
}
[System.Serializable]
class RatingResponse
{
    public double averageRating;
    public int totalVotes;
}
[System.Serializable]
class RecordRequest
{
    public int timeMs;
}
[System.Serializable]
public class RecordResponse
{
    public int userId;
    public string userLogin;
    public int timeMs;
    public string achievedAt;
}
[System.Serializable]
class RecordListWrapper
{
      public List<RecordResponse>
records;
}
[System.Serializable]
class RecordSubmitResponse
{
    public bool isNewRecord;
      public List<RecordResponse>
bestRecords;
}
[System.Serializable]
class UserRecordResponse
{
    public int timeMs;
    public string achievedAt;
}
[System.Serializable]
class UserRatingResponse
{
    public int rating;
}
#endregion
```