|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ИВАНОВСКОЙ ОБЛАСТИ**  Областное государственное бюджетное профессиональное образовательное учреждение  «Ивановский промышленно-экономический колледж» | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | |  | |
|  | | | | |  | |
|  | | | | |  | |
|  | | | | |  | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
| **КУРСОВОЙ ПРОЕКТ** | | | | | | |
|  | | | | | | |
| **Разработка мобильной игры «Постройка башни»** | | | | | | |
|  | | | | | | |
| **ИВПЭК. 09.02.07. №27** | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | **Специальность:** | | | **09.02.07 Информационные системы и программирование,**  **базовая подготовка** | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
| Руководитель курсового проекта | | | |  | | А.С. Горелов |
| Выполнил обучающийся группы 407а | | | |  | | Д.Г. Карташов |
|  | | | |  | |  |
|  | | | | | | |
|  | | | | | | |
| Курсовой проект выполнен и защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | |
|  | | | | | | |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | | | | |
|  |  | | | | | |
|  | | |  | | |  |
|  | | |  | | |  |
|  | Иваново 2024 | | | | |  |

Содержание

[Введение 3](#_Toc193802608)

[1 Концептуальное проектирование 4](#_Toc193802609)

[1.1 Теоретический вопрос 4](#_Toc193802610)

[1.2 Спецификация требований программного обеспечения 5](#_Toc193802611)

[2 Техно-рабочий проект 8](#_Toc193802612)

[2.1 Обоснование выбора средств разработки 8](#_Toc193802613)

[2.2 Разработка макета приложения 10](#_Toc193802614)

[3 Рабочая документация 14](#_Toc193802615)

[3.2 Тестирование приложения 23](#_Toc193802616)

[Заключение 26](#_Toc193802617)

[Список использованных источников 28](#_Toc193802618)

[Приложение А 29](#_Toc193802619)

Введение

Программирование является одной из ключевых дисциплин в современном мире информационных технологий. Оно играет важную роль в различных областях, начиная от разработки программного обеспечения и веб-приложений, и заканчивая созданием игр и решением задач искусственного интеллекта. Кроме того, программирование обеспечивает инструменты для автоматизации процессов, улучшения производительности и создания новых технологических решений.

Разработка мобильных игр становится все более актуальной и востребованной в современном мире. С развитием технологий и повсеместным распространением мобильных устройств, игровая индустрия привлекает все больше внимания как среди разработчиков, так и среди пользователей. В частности, казуальные игры, сочетающие в себе простоту механики с увлекательным геймплеем, пользуются особой популярностью среди мобильной аудитории.

Цель настоящего курсового проекта заключается в разработке мобильной игры «Постройка башни» с использованием современных веб-технологий и WebGL. Основные задачи проекта включают в себя:

* изучение возможностей и особенностей Three.js и физического движка Cannon.js для создания трехмерной игровой механики;
* проектирование архитектуры игры с использованием паттерна внедрения зависимостей (DI) и принципов SOLID;
* разработка игровой механики, включая физическое моделирование, систему очков и прогрессирующую сложность;
* реализация отзывчивого пользовательского интерфейса с использованием Alpine.js и современных CSS-технологий;
* тестирование и оптимизация производительности для обеспечения плавного игрового процесса на мобильных устройствах.
* Проект демонстрирует применение современного стека веб-технологий.

# 1 Концептуальное проектирование

1.1 Теоретический вопрос

Современная разработка игр является одной из динамичных областей IT-индустрии, которая позволяет создавать интерактивные развлекательные продукты. Игровые приложения охватывают широкий спектр жанров и механик, от простых головоломок до сложных симуляторов. Одной из актуальных задач является разработка казуальных игр, таких как "Постройка башни", так как растет спрос на доступные мобильные развлечения.

С точки зрения игровой разработки, важно учитывать несколько ключевых аспектов:

* игровая механика: Приложение должно иметь интуитивно понятные правила с четкой системой размещения блоков, физическим моделированием и подсчетом очков. Игроки часто играют короткими сессиями, поэтому механика должна быть простой для освоения, но обеспечивать достаточную глубину геймплея;
* пользовательский опыт: помимо базовой механики, важно обеспечить увлекательный игровой процесс. Это достигается за счет прогрессивного усложнения, разнообразных игровых эффектов и событий, а также плавной работы всех систем без задержек;
* архитектура и безопасность: Учитывая необходимость поддержки и обновления игры, важным аспектом является правильное проектирование архитектуры. Использование современных паттернов проектирования, разделение логики на модули и обеспечение надежного управления состоянием игры играют ключевую роль.

Таким образом, разработка игры "Постройка башни" опирается на современные принципы проектирования и разработки, которые учитывают как технические, так и пользовательские аспекты. Успешная реализация подобного проекта предполагает комплексный подход к проектированию, архитектуре и функциональности, что позволит обеспечить высокий уровень вовлеченности игроков и популярность игры на рынке.

1.2 Спецификация требований программного обеспечения

Спецификация требований программного обеспечения представляет собой комплексный документ, детально описывающий функциональные и нефункциональные требования к разработке интерактивной игры-головоломки с механикой строительства башни. Данная спецификация разработана в строгом соответствии с международными стандартами ISO/IEC/IEEE 29148:2011 и ISO/IEC 25010:2011, что обеспечивает высокий уровень качества и структурированности представленных требований.

1. Функциональные требования

1.1. Игровая механика

Система должна обеспечивать плавное движение блоков с настраиваемой скоростью падения, учитывая различные уровни сложности игры. Скорость падения блоков должна регулироваться в зависимости от прогресса игрока и текущего уровня.

Приложение должно реализовывать высокоточную систему определения перекрытия блоков с погрешностью не более 1% от размера блока. Система должна учитывать как горизонтальное, так и вертикальное перекрытие элементов.

Комбо-механика должна включать прогрессивную систему множителей очков, где каждое успешное размещение блока увеличивает множитель на определенный коэффициент. Максимальный множитель должен быть ограничен значением х10.

Динамическое усложнение игрового процесса должно происходить путем постепенного увеличения скорости падения блоков, уменьшения их размера и добавления дополнительных препятствий.

1.2. Физическая система

Приложение должно обеспечивать реалистичную симуляцию падения и столкновений блоков с использованием современного физического движка. Все взаимодействия должны учитывать массу, инерцию и силу трения между объектами.

Система должна производить непрерывный мониторинг стабильности конструкции в реальном времени, учитывая центр масс башни и распределение нагрузки по всей структуре. При критическом отклонении от вертикали должна срабатывать система предупреждения игрока.

Физические параметры, включая гравитацию, массу блоков и коэффициенты трения, должны быть настраиваемыми для каждого уровня сложности, обеспечивая оптимальный баланс между реалистичностью и играбельностью.

1.3. Пользовательский интерфейс

Приложение должно предоставлять информативный и интуитивно понятный интерфейс, отображающий ключевые игровые показатели: текущий счет, активный множитель комбо, достигнутую высоту башни и рекорды игрока.

Система должна обеспечивать плавные переходы между игровыми экранами, включая стартовое меню, игровой процесс и экран завершения игры. На экране завершения должна отображаться детальная статистика игровой сессии.

Управление должно осуществляться через простые и отзывчивые механики касания/клика, с временем отклика не более 50 миллисекунд. Визуальная обратная связь должна включать анимации, частицы и звуковые эффекты для всех значимых игровых событий.

2. Нефункциональные требования

2.1. Производительность (Performance)

Приложение должно поддерживать стабильную частоту кадров (60 FPS) на мобильных устройствах.

Система должна обеспечивать оптимизированные физические расчеты.

Приложение должно эффективно управлять памятью при длительной игре.

Время загрузки и инициализации игры должно быть минимальным.

2.2. Масштабируемость (Scalability)

Интерфейс должен быть адаптивным для различных размеров экрана.

Приложение должно поддерживать как мобильные устройства, так и десктопы.

Система должна обеспечивать возможность легкого добавления новых игровых механик.

Приложение должно иметь масштабируемую систему сложности.

2.3. Надежность (Reliability)

Физический движок должен работать стабильно при любых условиях.

Система должна корректно сохранять состояние игры.

Приложение должно обрабатывать все краевые случаи в физической симуляции.

Система должна предотвращать возможные эксплойты игровой механики.

3. Архитектурные принципы

Согласно принципам проектирования, ключевые аспекты реализации включают:

Модульность:

* использование инверсии зависимостей для обеспечения гибкости системы.

Разделение ответственности:

* четкое разграничение компонентов системы.

Интерфейсы:

* определение четких границ взаимодействия между модулями.

Масштабируемость:

* возможность расширения функциональности без изменения существующего кода.

Тестируемость:

* Архитектура должна поддерживать модульное и интеграционное тестирование.

# 2 Техно-рабочий проект

2.1 Обоснование выбора средств разработки

Для разработки браузерной игры "Строитель Башни" был выбран современный стек технологий, включающий TypeScript как основной язык программирования, в сочетании с мощными специализированными библиотеками Three.js, Cannon.js и Alpine.js, а также дополнительными инструментами разработки.

Основные технологии

TypeScript выступает фундаментальной технологией проекта, обеспечивая надежную статическую типизацию и расширенные возможности объектно-ориентированного программирования. Это позволяет создавать более структурированный и поддерживаемый код, значительно снижая количество потенциальных ошибок на этапе разработки.Three.js используется как основная библиотека для работы с 3D-графикой, предоставляя широкий набор инструментов для создания и управления трехмерными объектами, камерами, освещением и материалами. Библиотека обеспечивает высокую производительность рендеринга и кроссбраузерную совместимость.Cannon.js отвечает за физическое моделирование в игре, обеспечивая реалистичное поведение блоков, их взаимодействие и симуляцию различных физических эффектов, таких как гравитация, столкновения и трение.Alpine.js используется для управления пользовательским интерфейсом, предоставляя декларативный подход к созданию интерактивных элементов и управлению состоянием UI-компонентов.

Дополнительные инструменты

InversifyJS применяется для реализации внедрения зависимостей, что позволяет создать гибкую и легко тестируемую архитектуру приложения. Библиотека обеспечивает удобное управление зависимостями между различными компонентами игры.Vite используется как инструмент сборки проекта, обеспечивая быструю разработку с поддержкой горячей замены модулей (HMR) и эффективную сборку для продакшена.

Преимущества выбранного стека технологий

Надежность и безопасность типов:

* строгая типизация TypeScript предотвращает множество потенциальных ошибок;
* улучшенная поддержка IDE с автодополнением и статическим анализом;
* возможность создания надежных интерфейсов и абстракций.
* Высокая производительность:
* оптимизированный рендеринг 3D-графики через Three.js;
* эффективные физические расчеты с помощью Cannon.js;
* быстрая работа пользовательского интерфейса благодаря Alpine.js.
* Модульность и масштабируемость:
* четкое разделение ответственности между компонентами;
* возможность легкого добавления новых функций;
* гибкая архитектура благодаря использованию InversifyJS.

Удобство разработки:

* декларативный подход к созданию UI;
* богатая экосистема инструментов и библиотек;
* обширная документация и активное сообщество.

Кроссбраузерная совместимость:

* поддержка всех современных браузеров;
* адаптивность к различным устройствам;
* оптимизированная производительность.

Архитектурные решения

1. Компонентная архитектура:

* разделение логики на независимые модули;
* использование интерфейсов для определения контрактов;
* применение принципов SOLID.

Система событий:

* реализация паттерна Observer для игровых событий;
* гибкая система обработки пользовательского ввода;
* эффективное управление состоянием игры.

Физическое моделирование:

* реалистичная симуляция физики блоков;
* настраиваемые параметры физических взаимодействий;
* оптимизированные расчеты столкновений.

Управление состоянием:

* централизованное хранение игрового состояния;
* механизмы сохранения и загрузки прогресса;
* эффективное обновление UI.

Выбранный стек технологий обеспечивает оптимальный баланс между производительностью, надежностью и удобством разработки. TypeScript в сочетании со специализированными библиотеками позволяет создать качественный игровой продукт с реалистичной физикой, привлекательной графикой и отзывчивым пользовательским интерфейсом. Использование современных инструментов разработки и архитектурных паттернов обеспечивает масштабируемость проекта и возможность его дальнейшего развития.

2.2 Разработка макета приложения

Для разработки макета игры были использованы следующие ключевые элементы:

1. Навигация:

Использование трех основных экранов (Стартовый, Игровой, Завершение).

Возможность перехода между экранами с помощью интерактивных элементов.

1. Игровые экраны:

Стартовый экран с основными элементами управления.

Игровой экран с активными игровыми компонентами.

Экран завершения игры с результатами.

1. Визуальное оформление:

Использование единой системы компонентов и согласованной цветовой палитры.

Применение адаптивной сетки и интерактивных состояний элементов.

Реализация анимационных переходов между экранами.

Далее представлены макеты основных экранов игры (Рисунок 1-3).

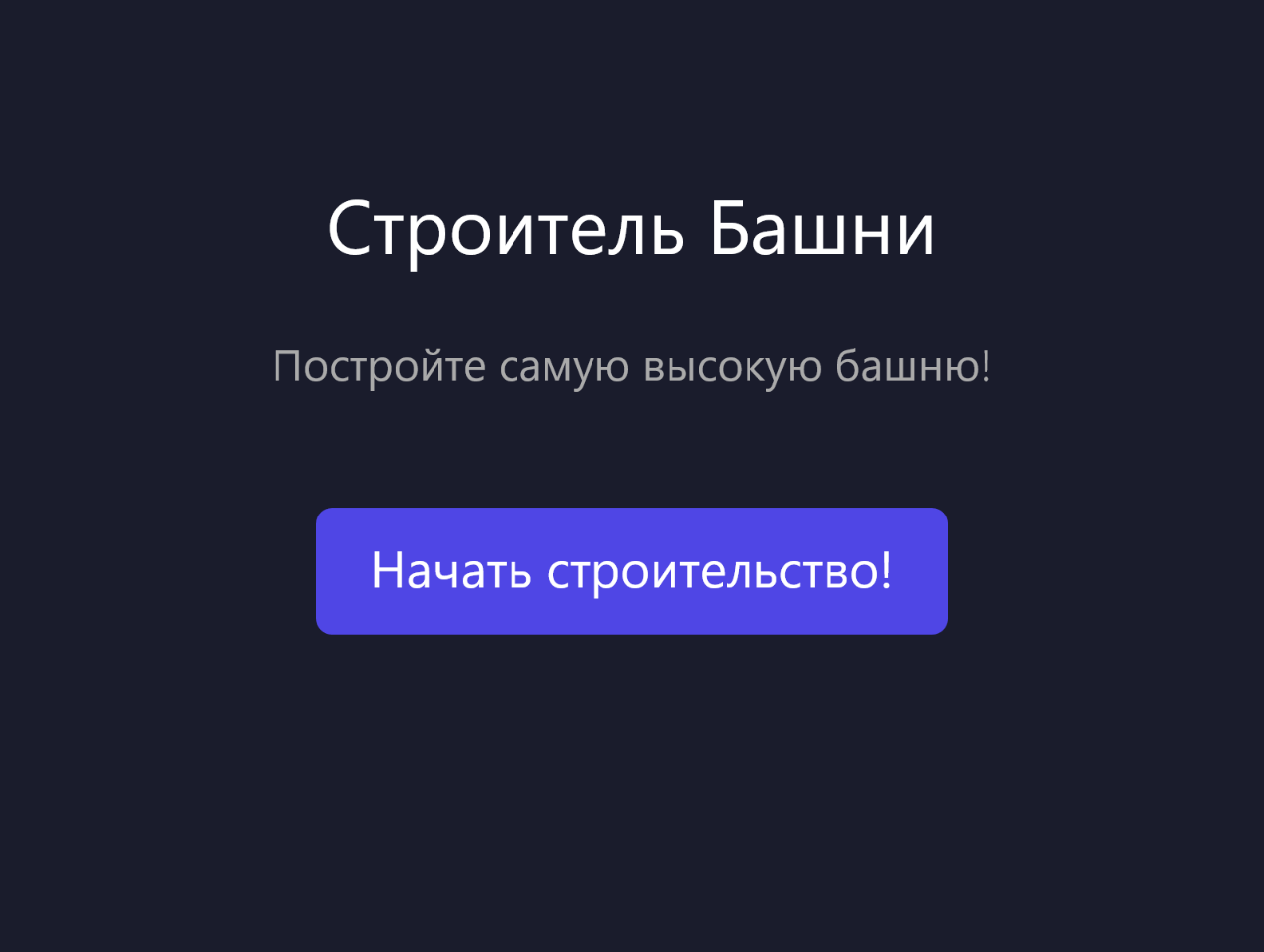


Рисунок 1 – Изображение стартового экрана игры

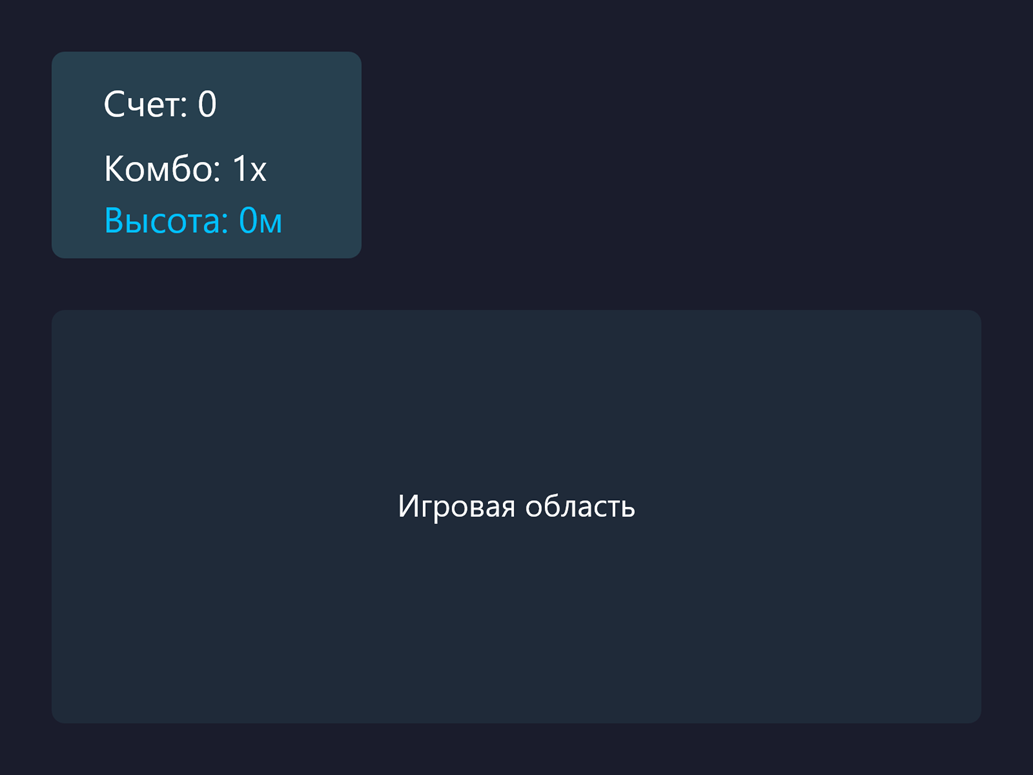


Рисунок 2 – Изображение игрового экрана

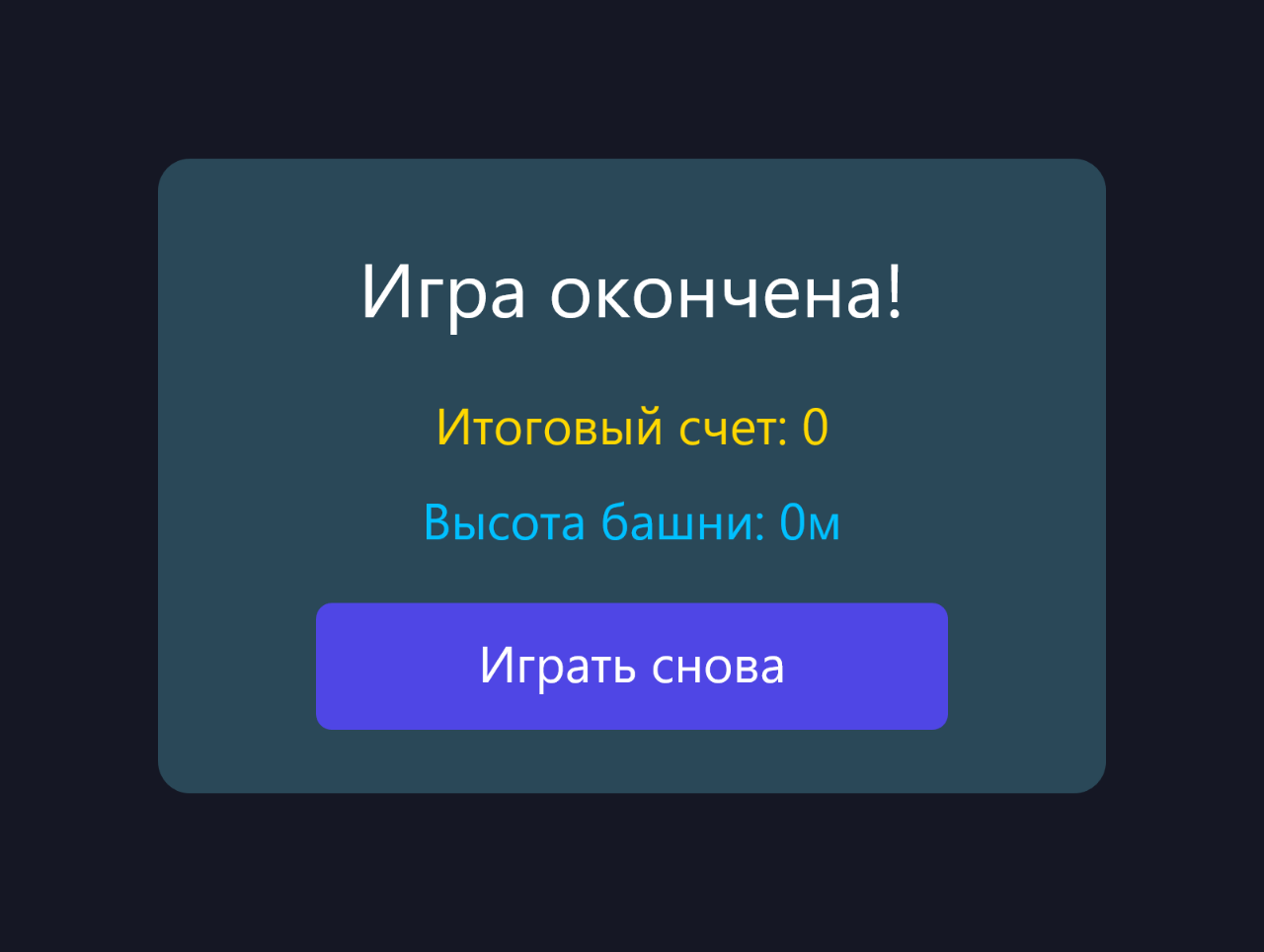


Рисунок 3 – Изображение экрана завершения игры

Данный макет был разработан в инструменте прототипирования Figma, что позволило создать точные прототипы, настроить интерактивные связи между экранами и обеспечить единообразие интерфейса. Проект учитывает современные принципы дизайна и обеспечивает оптимальный пользовательский опыт.

# 3 Рабочая документация

Разработка браузерной игры "Строитель Башни" представляет собой увлекательный проект, направленный на создание интерактивного и захватывающего игрового опыта. Игра позволяет пользователям испытать свои навыки точного размещения движущихся блоков для построения самой высокой башни. В проекте используются современные веб-технологии, обеспечивающие высокую производительность, реалистичность физического моделирования и привлекательный пользовательский интерфейс.

Технологический стек

* TypeScript: Статически типизированный язык программирования, обеспечивающий надежность и масштабируемость кода;
* THREE.js: Библиотека для создания и отображения 3D-графики в веб-браузерах;
* CANNON.js: Физический движок для симуляции реалистичного поведения объектов;
* Alpine.js: Легковесный фреймворк для создания реактивного и интерактивного пользовательского интерфейса;
* InversifyJS: Контейнер для внедрения зависимостей, облегчающий управление зависимостями компонентов.

Структура проекта

Проект организован согласно стандартной архитектуре, где все исходные файлы находятся в директории src. Основные компоненты проекта включают:

* components/: Содержит UI-компоненты, такие как GameOverlay, PauseButton и Toast;
* config/: Хранит конфигурационные файлы, например, GameConfig.ts, определяющий параметры игры;
* core/: Включает контейнер зависимостей (container.ts), управляющий связыванием интерфейсов и реализаций;
* game/: Содержит основные игровые классы и модули, такие как GameEngine, BlockManager, PhysicsEngine, и различные игровые события;
* interfaces/: Определяет интерфейсы для различных компонентов системы, обеспечивая слабую связь и удобство тестирования;
* types/: Содержит определения типов и токенов для InversifyJS;
* types/types.ts: Определяет идентификаторы для внедрения зависимостей;
* основные компоненты и классы.

1. Класс TowerGame

Отвечает за основную игровую логику и управление состоянием игры.

Основные методы:

* init(): Инициализирует физический движок, рендерер и менеджер блоков.;
* animate(): Выполняет шаги физического моделирования, обновляет состояние блоков и рендерит сцену.
* addNewBlock(height: number): Добавляет новый блок на определенной высоте;
* placeBlock(): Размещает текущий блок в башне;
* cleanup(): Очищает игровую сцену, удаляя все блоки и сбрасывая камеру;
* checkStability(): Проверяет стабильность башни;
* togglePause(): Переключает состояние паузы в игре;
* saveGameState(): Сохраняет текущее состояние игры в localStorage;
* loadGameState(): Загружает сохраненное состояние игры из localStorage.

2. Класс BlockManager

Управляет блоками в игре, включая их создание, обновление и проверку стабильности конструкции.

Основные методы:

* init(): Создает базовую платформу для башни;
* update(): Обновляет позиции текущего блока и всех размещенных блоков;
* addBlock(height: number): Создает новый блок на указанной высоте;
* placeCurrentBlock(): Размещает текущий блок на башне, обновляя физические свойства;
* cleanup(): Удаляет все блоки из сцены и физического мира;
* checkStability(): Проверяет стабильность последних блоков, определяя, не опрокинулась ли башня;
* getCurrentBlock(): Возвращает текущий активный блок;
* getBlocks(): Возвращает все размещенные блоки;
* setBlockSpeed(speed: number): Устанавливает скорость движения блоков.

3. Класс PhysicsEngine

Управляет физическим моделированием в игре, обеспечивая реалистичное поведение блоков и взаимодействие с окружающей средой.

Основные методы:

* init(): Инициализирует физический мир, задает гравитацию и материалы для объектов.
* step(): Выполняет шаг физического симулятора, обновляя состояния всех тел.
* addBody(body: CANNON.Body): Добавляет тело в физический мир.
* removeBody(body: CANNON.Body): Удаляет тело из физического мира.
* setGravity(value: number): Устанавливает силу гравитации.
* createBlockBody(dimensions: CANNON.Vec3, position: CANNON.Vec3, mass: number): Создает физическое тело для блока.
* getMaterial(name: string): Возвращает материал по имени.

4. Класс Renderer

Отвечает за визуализацию игрового мира, управление камерой и освещением сцены. Основные методы:

* init(): Инициализирует сцену, камеру, рендерер и освещение;
* render(): Рендерит текущую сцену;
* resetCamera(): Сбрасывает положение камеры до начального состояния;
* adjustCamera(height: number): Настраивает камеру под текущую высоту башни;
* cleanup(): Очищает сцену и освобождает ресурсы рендерера.

5. Класс GameEventManager

Управляет игровыми событиями, применяя их к блокам и создавая визуальные эффекты.Основные методы:

* init(): Инициализирует менеджер событий;
* updateLevel(level: number): Обновляет текущий уровень игры;
* applyEvents(block: any): Применяет применимые события к блоку;
* update(): Обновляет активные всплывающие уведомления;
* tryShowEventPopup(event: IGameEvent): Показывает всплывающее уведомление при активации события.

6. Классы событий

Проект включает несколько классов, реализующих интерфейс IGameEvent, каждый из которых отвечает за определенное игровое событие:

* WindEvent: Добавляет порывы ветра, влияющие на блоки;
* BlockShrinkEvent: Уменьшает размер блоков с течением времени;
* BlockRotationEvent: Вращает блоки вокруг своей оси;
* GravityEvent: Изменяет силу гравитации;
* ColorPulseEvent: Создает эффект пульсации цвета блоков;
* BlockSpinEvent: Заставляет блоки крутиться в воздухе;
* BounceEvent: Изменяет упругость блоков при столкновении.

7. UI-компоненты

1. Компонент GameOverlay.

Реализует пользовательский интерфейс игры, отображая игровую статистику, состояние игры и обрабатывая пользовательский ввод.Основные элементы:

* панель счета: Отображает текущий счет, комбо и высоту башни;
* экран окончания игры: Появляется при завершении игры, показывая итоговый счет и высоту башни, а также кнопку для перезапуска игры;
* стартовый экран: Отображает название игры, инструкции и кнопку для начала игры.

1. Компонент PauseButton

Управляет состоянием паузы в игре, позволяя игроку приостанавливать и возобновлять игру.

Основные методы:

* togglePause(): Переключает состояние паузы;
* resumeGame(): Возобновляет игру после паузы;
* restartGame(): Перезапускает игру;
* saveGame(): Сохраняет текущее состояние игры;
* loadGame(): Загружает сохраненное состояние игры;
* init(): Инициализирует обработчики событий, например, нажатие клавиши 'Escape' для возобновления игры.

1. Компонент Toast

Назначение: Отображает всплывающие уведомления (тосты) для информирования игрока о различных событиях, таких как сохранение игры или получение бонусов.

Основные элементы:

* контейнер уведомлений: Располагается в правом верхнем углу экрана и отображает список активных уведомлений;
* шаблон уведомления: Использует анимации для плавного появления и исчезновения, изменяет цвет в зависимости от типа уведомления (ошибка, предупреждение, успех, информация);
* стилизация: Использует CSS-анимации для эффектов входа и выхода.

8. Файлы конфигурации

Хранит все конфигурационные параметры игры, включая настройки для системы начисления очков, комбо, сложности, визуальных эффектов и физики.Основные разделы:

* scoring: Определяет пороги для различных уровней попаданий (perfect, great, good, ok) с соответствующими набранными очками и сообщениями;
* combo: Настройки для системы комбо, включая максимальное количество комбо и бонусные очки;
* difficulty: Параметры сложности игры, такие как базовая скорость блоков, увеличение скорости на уровне, максимальная скорость, количество блоков на уровне, скорость уменьшения блоков, минимальный размер блоков, эффекты ветра и гравитации, параметры отскока и вращения блоков;
* visuals: Цвета для различных уровней попаданий;
* physics: Настройки физического моделирования, включая время стабилизации, пороги скорости, параметры блоков, материалы и параметры решателя.

9. Контейнер зависимостей

Использует InversifyJS для управления зависимостями между различными компонентами системы, обеспечивая инверсию управления и легкость тестирования.Основные биндинги:

* IRenderer → Renderer;
* IPhysicsEngine → PhysicsEngine;
* IBlockManager → BlockManager;
* IGameEngine → GameEngineПосле биндинга зависимостей и получения экземпляра IGameEngine, вызывается метод init() для инициализации игры.

10. Компоненты блоков

1. Класс Block

Представляет отдельный блок в башне, управляет его физическими и визуальными свойствами.Основные методы:

* update(): Обновляет позицию и состояние блока;
* moveX(amount: number): Передвигает блок по оси X;
* setWindForce(force: number): Применяет силу ветра к блоку;
* getMesh(): Возвращает визуальный объект блока;
* getBody(): Возвращает физическое тело блока;
* wakeUp(): Активирует блок, восстанавливая его физическое состояние;
* setScale(scale: number): Устанавливает масштаб блока;
* setInitialRotation(rotation: number): Устанавливает начальный угол вращения блока;
* remove(): Удаляет блок из сцены и физического мира;
* isStabilized(): Проверяет, стабилизирован ли блок;
* isStable(): Определяет, стабильна ли позиция и ориентация блока;
* getCriticalTilt(): Проверяет критический наклон блока.

1. Класс BlockPhysics

Управляет физическими свойствами блока, включая массу, трение, упругость и взаимодействие с другими объектами.

Основные методы:

* getBody(): Возвращает физическое тело блока;
* setScale(scale: number): Изменяет масштаб блока
* setPosition(position: CANNON.Vec3): Устанавливает позицию блока;
* setRotation(rotation: number): Устанавливает угол вращения блока;
* applyWindForce(force: number): Применяет силу ветра к блоку;
* remove(): Удаляет тело блока из физического мира;
* preventIntersection(): Предотвращает пересечение блока с другими объектами.

11. Компоненты визуализации блоков

Класс BlockRenderer

Отвечает за визуальное представление блока в 3D-пространстве, управляет его геометрией, материалами и анимациями.Основные методы:

* updatePosition(position: THREE.Vector3, quaternion: THREE.Quaternion): Обновляет позицию и ориентацию блока в сцене;
* getMesh(): Возвращает визуальный объект блока;
* getColor(): Возвращает цвет блока;
* setScale(scale: number): Устанавливает масштаб блока;
* setPosition(position: THREE.Vector3): Устанавливает позицию блока;
* setRotation(rotation: number): Устанавливает угол вращения блока;
* remove(): Удаляет блок из сцены.

12. События блоков

Каждое событие реализуется как отдельный класс, наследующийся от интерфейса IGameEvent, который определяет методы isApplicable(level: number): boolean и apply(block: any): void.

* BlockRotationEvent: Вращает блоки на уровне 2 и выше с вероятностью 30%;
* BlockShrinkEvent: Уменьшает размер блоков в зависимости от количества блоков;
* BlockSpinEvent: Забрасывает блоки во вращение на уровне 4 и выше с вероятностью 30%;
* BounceEvent: Изменяет коэффициент упругости блоков на уровне 3 и выше;
* ColorPulseEvent: Создает эффект мерцания цвета блоков на уровне 2 и выше;
* GravityEvent: Меняет силу гравитации на основе текущего уровня;
* WindEvent: Применяет силу ветра к блокам на основе конфигурации сложности.

Управление состоянием игры

Для сохранения прогресса игрока используется встроенный в браузер механизм localStorage. Состояние игры сохраняется при активации паузы и восстанавливается при следующем запуске игры. Сохраненное состояние включает:

* позиции, вращение и масштаб всех блоков;
* текущий счет, комбо и высоту башни.Методы saveGameState() и loadGameState() в классе GameEngine отвечают за сериализацию и десериализацию данного состояния.

Физическое моделирование

CANNON.js используется для симуляции физики блоков, обеспечивая реалистичное поведение объектов. Основные аспекты физического моделирования:

* масса и инерция: Каждому блоку присваивается масса, влияющая на динамику его движения;
* трение и упругость: Настройки трения и упругости обеспечивают реалистичные столкновения между блоками и платформой;
* стабилизация: Реализована логика проверки стабильности блоков, предотвращающая опрокидывание башни;
* дополнительные силы: Применение сил ветра и изменения гравитации влияющие на поведение блоков.

Визуализация и рендеринг

THREE.js отвечает за создание и рендеринг 3D-сцены. Компонент Renderer управляет следующими аспектами:

* сцена: Центральная часть, содержащая все объекты игры;
* камера: Перспективная камера, динамически адаптирующая позицию под высоту башни;
* освещение: Используются различные источники света для создания реалистичной освещенности;
* обработка анимаций: Плавный рендеринг и обновление кадров для обеспечения высокой производительности.

Пользовательский интерфейс

Интерфейс пользователя разработан с использованием Alpine.js, обеспечивая отзывчивость и интерактивность. Основные компоненты UI включают:

* GameOverlay: Отображает счет, комбо и высоту башни, а также содержит экраны начала игры и окончания игры;
* PauseButton: Позволяет игроку приостанавливать и возобновлять игру, а также сохранять и загружать состояние;
* Toast: Отображает всплывающие уведомления о событиях в игре, таких как сохранение или получение бонусов.

Событийная система

Система событий организована через класс GameEventManager, который управляет применением событий к блокам и отображением соответствующих уведомлений. Это позволяет добавлять новые игровые механики, расширяя функциональность игры без изменения основной логики.

Управление зависимостями

Используется InversifyJS для внедрения зависимостей между компонентами. Это обеспечивает слабую связность, удобство тестирования и расширяемость архитектуры приложения. В файле container.ts определяются биндинги интерфейсов к их конкретным реализациям, что позволяет легко заменять или расширять функциональность компонентов.

3.2 Тестирование приложения

В процессе разработки игры "Строитель Башни" особое внимание было уделено комплексному тестированию всех компонентов приложения. Тестирование проводилось с использованием современного фреймворка Vitest, который обеспечивает удобную среду для написания и выполнения тестов, а также интеграцию с системой сборки проекта.

Цели тестирования

Основными целями тестирования являлись проверка корректности работы основных модулей игры, обеспечение стабильности и надежности приложения при различных сценариях использования, выявление и устранение возможных ошибок и уязвимостей на ранних этапах разработки, а также подтверждение соответствия реализованных функций требованиям и спецификациям проекта.

Структура тестирования

Тестирование было организовано по четырем ключевым модулям приложения: класс TowerGame, класс GameEventManager, класс BlockPhysics и компонент ToastManager.

1. Тестирование класса TowerGame

Для класса TowerGame было реализовано пять тестовых сценариев, направленных на проверку следующих аспектов: корректность инициализации всех зависимостей и начальных параметров игры, переключение состояния игры между паузой и активным режимом с сохранением и восстановлением состояния, проверка корректного обновления физического состояния игры и вызова рендеринга сцены методом animate(), тестирование добавления новых блоков в игру и их корректной интеграции в физический и визуальный мир, а также проверка метода checkStability(), обеспечивающего определение устойчивости башни после размещения новых блоков.

2. Тестирование класса GameEventManager

Для класса GameEventManager было реализовано два теста: проверка корректности обработки игровых событий, включая правильную идентификацию и применение к блокам в зависимости от текущего уровня сложности, а также тестирование применения эффектов к блокам при активации событий, таких как применение силы ветра или изменение гравитации.

3. Тестирование класса BlockPhysics

Для класса BlockPhysics было разработано пять тестовых сценариев: проверка корректности создания физических тел блоков с заданными параметрами, тестирование правильности обновления позиций блоков при симуляции физики, проверка корректности обработки вращения блоков и их угловой скорости, тестирование применения дополнительных сил и их влияния на поведение блоков, а также проверка эффективности метода preventIntersection() в предотвращении пересечения блоков.

4. Тестирование компонента ToastManager

Для компонента ToastManager было написано четыре теста: проверка корректности создания toast-сообщений с заданными параметрами, тестирование правильности отображения сообщений в пользовательском интерфейсе, проверка удаления toast-сообщений после истечения времени отображения, а также тестирование корректного применения стилей и цветов в зависимости от типа сообщения.

Настройка тестового окружения

Для обеспечения корректного тестирования было настроено специальное тестовое окружение, включающее моки для браузерных API, таких как requestAnimationFrame, performance и localStorage. Это позволяет проводить тестирование в изолированной среде, максимально приближенной к реальным условиям работы приложения.

Результаты тестирования

Результаты тестирования показывают высокую стабильность приложения — все 16 тестов успешно проходят выполнение. Общее время выполнения тестов составляет 9.50 секунд, включая время на подготовку окружения (5.06 секунд), трансформацию кода и непосредственное выполнение тестов (4.44 секунды).

Анализ и рекомендации

Анализ времени выполнения тестов выявил области для оптимизации, включая длительное время подготовки тестового окружения и оптимизацию трансформации кода. Рекомендуется расширить покрытие тестами, особенно для модуля GameEventManager, добавить интеграционные тесты для проверки взаимодействия между различными модулями игры, внедрить тесты производительности для измерения времени выполнения ключевых функций, а также интегрировать тестирование в процесс непрерывной интеграции и развертывания.

# Заключение

В рамках данного курсового проекта была разработана браузерная игра "Tower Builder" с использованием современных веб-технологий и TypeScript. В ходе работы над проектом был реализован основной функционал, позволяющий пользователям строить и управлять виртуальными башнями с применением Three.js для 3D-рендеринга и Cannon.js для физического моделирования. Это приложение направлено на создание увлекательного игрового процесса и обеспечение реалистичной физики взаимодействия блоков.

Основные достижения:

1. Создание и управление игровым процессом:

Игроки могут укладывать блоки друг на друга, стремясь построить максимально высокую башню.

Реализована сложная система физики, предотвращающая проскальзывание блоков, что делает игровой процесс более реалистичным.

2. Интерактивный интерфейс:

Использование Alpine.js и современных CSS-технологий позволило создать отзывчивый и интуитивно понятный пользовательский интерфейс.

Интерфейс включает систему всплывающих уведомлений и возможность паузы, что улучшает игровой опыт.

3. Оптимизация для мобильных устройств

Приложение было адаптировано для Android с использованием WebView, обеспечивая оптимальную производительность и поддержку аппаратного ускорения.

Созданное приложение имеет все предпосылки для дальнейшего развития и оптимизации.

Мы планируем внедрение следующих функций:

* расширение игровых механик: добавление новых типов блоков и специальных эффектов;
* многопользовательский режим: создание системы рейтингов и соревнований между игроками;
* улучшенная физика: внедрение более сложных физических взаимодействий и погодных эффектов.

Разработка браузерной игры "Tower Builder" стала значительным шагом в направлении создания качественных веб-игр с физическим моделированием. Приложение сочетает в себе современные технологии и увлекательный игровой процесс, что делает его актуальным в условиях растущей популярности браузерных игр. Мы уверены, что данный проект будет востребован и сможет предоставить игрокам уникальный опыт строительства виртуальных башен.

Таким образом, наша команда готова продолжать работу над игрой, улучшать её функционал и адаптировать к требованиям пользователей, чтобы обеспечить максимально увлекательный и качественный игровой опыт.

# Список использованных источников

1. Документация Three.js: [Электронный ресурс]. URL: https://threejs.org/docs/
2. Документация Cannon.js: [Электронный ресурс]. URL: https://schteppe.github.io/cannon.js/docs/
3. Документация TypeScript: [Электронный ресурс]. URL: https://www.typescriptlang.org/docs/
4. Документация InversifyJS: [Электронный ресурс]. URL: https://inversify.io/
5. Документация Alpine.js: [Электронный ресурс]. URL: https://alpinejs.dev/docs
6. Документация Tailwind CSS: [Электронный ресурс]. URL: https://tailwindcss.com/docs
7. Nystrom Robert Game Programming Patterns: Издательство: Genever Benning; 2014. – 354 с
8. Ericson Christer Real-Time Collision Detection: Издательство: Morgan Kaufmann; 2004. – 632 с
9. Freeman Jesse HTML5 Game Development: Издательство: Apress; 2012. – 412 с
10. Mak Thomas Mobile Game Development with HTML5: Издательство: Packt Publishing; 2014. – 328 с

# Приложение А

Фрагмент кода

BlockManager.ts:

import { injectable, inject } from 'inversify';

import { TYPES } from '../types/types';

import { Block } from './components/Block';

import { IBlockManager } from '../interfaces/IBlockManager';

import { IPhysicsEngine } from '../interfaces/IPhysicsEngine';

import { IRenderer } from '../interfaces/IRenderer';

import { GameConfig } from '../config/GameConfig';

@injectable()

export class BlockManager implements IBlockManager {

private blocks: Block[] = [];

private currentBlock: Block | null = null;

private blockSpeed: number = GameConfig.defaultBlockSpeed;

private movementDirection: number = 1;

constructor(

@inject(TYPES.IPhysicsEngine) private physicsEngine: IPhysicsEngine,

@inject(TYPES.IRenderer) private renderer: IRenderer

) {}

public init(): void {

this.blocks = [];

this.currentBlock = null;

}

public update(): void {

if (this.currentBlock) {

this.updateBlockMovement();

}

this.blocks.forEach(block => block.update());

}

private updateBlockMovement(): void {

if (!this.currentBlock) return;

const position = this.currentBlock.getPosition();

const newX = position.x + (this.blockSpeed \* this.movementDirection);

if (Math.abs(newX) > GameConfig.maxBlockOffset) {

this.movementDirection \*= -1;

}

this.currentBlock.setPosition(newX);

}

}

PhysicsEngine.ts:

import { injectable } from 'inversify';

import \* as CANNON from 'cannon-es';

import { IPhysicsEngine } from '../interfaces/IPhysicsEngine';

import { GameConfig } from '../config/GameConfig';

@injectable()

export class PhysicsEngine implements IPhysicsEngine {

public world: CANNON.World | null = null;

private materials: Map<string, CANNON.Material> = new Map();

private lastTime: number = 0;

private fixedTimeStep: number = 1.0 / 60.0;

private maxSubSteps: number = 3;

public init(): void {

this.world = new CANNON.World({

gravity: new CANNON.Vec3(0, -9.82, 0)

});

this.world.allowSleep = true;

this.world.defaultContactMaterial.contactEquationStiffness = 1e8;

this.world.defaultContactMaterial.contactEquationRelaxation = 3;

this.world.defaultContactMaterial.friction = 0.8;

this.world.splitImpulse = true;

this.initMaterials();

}

public step(): void {

if (!this.world) return;

const time = performance.now() / 1000;

if (!this.lastTime) {

this.lastTime = time;

}

const dt = time - this.lastTime;

this.lastTime = time;