  
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«Дальневосточный федеральный университет»**

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Департамент программной инженерии и искусственного интеллекта**

**РАЗРАБОТКА ИГРОВОГО ДВИЖКА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ КОЛЛЕКТИВНОЙ ПРОМЫШЛЕННОЙ РАЗРАБОТКИ**

КУРСОВОЙ ПРОЕКТ  
по дисциплине «Технологии коллективной промышленной разработки информационных систем» по образовательной программе подготовки бакалавров по направлению 09.03.04 «Программная инженерия»

Выполнили:  
студенты гр. Б9120-09.03.04прогин  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Рольщиков М.Е.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Мамедов Ш.М.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Фролов А.А.  
Руководитель:  
старший преподаватель департамента ПИиИИ  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Иваненко В.М.

г. Владивосток

2024

Оглавление

[Введение 3](#_Toc163683011)

[1.План проекта 4](#_Toc163683012)

[2.Регламент проведения инспекции 5](#_Toc163683013)

[3.Модель состояний задач 7](#_Toc163683014)

[4.Презентация проекта 8](#_Toc163683015)

[5.Требования к проекту 14](#_Toc163683016)

[6.Разработка архитектуры проекта 17](#_Toc163683017)

[7.Измерения проекта 18](#_Toc163683018)

[8.Перечень задач проекта 20](#_Toc163683019)

[9.Правила по кодированию 22](#_Toc163683020)

[10.Разработка плана тестирования проекта 24](#_Toc163683021)

[11.Тестирование проекта 31](#_Toc163683022)

[Заключение 36](#_Toc163683023)

[Список литературы 37](#_Toc163683024)

# Введение

Промышленная разработка информационных систем включает в себя множество этапов, начиная от разработки плана проекта, заканчивая его тестированием, для чего необходимо множество специалистов различных профилей, а также унифицированные методы коммуникации между ними, с помощью которых можно разделить обязанности членов команды по их специализации.

Исходя из описанного выше, необходимо использовать определенные технологии коллективной разработки для повышения эффективности работы в группе и соответствия конечного продукта заявленным требованиям.

В данной курсовой работе рассматривается задача коллективной разработки программного средства «Игровой движок» и составление технической документации к нему.

Таким образом, целью курсовой работы является разработка программного средства с использованием подходов коллективной промышленной разработки.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать план проекта.
2. Разработать регламент проведения инспекции.
3. Разработать модель состояний задач.
4. Разработать презентацию проекта.
5. Разработать требования к проекту.
6. Разработать архитектуру проекта.
7. Разработать измерения проекта.
8. Разработать перечень задач проекта.
9. Разработать рекомендации по кодированию.
10. Разработать план тестирования проекта.
11. Протестировать проект.

# План проекта

План проекта – это документ, содержащий подробную информацию о проекте: исполнителях, задачах и сроках. Документ является конечным результатом этапа планирования, утверждается до начала любых работ и становится самым главным и достоверным источником информации о грядущем проекте.

В нашем случае исполнителями являются следующие лица:

* Team Leader – Рольщиков Максим;
* Coder 1 – Рольщиков Максим;
* Coder 2 – Фролов Анатолий;
* Technical Writer – Мамедов Шамиль.

На рисунке 1 представлен перечень задач для выполнения и примерные сроки их реализации.

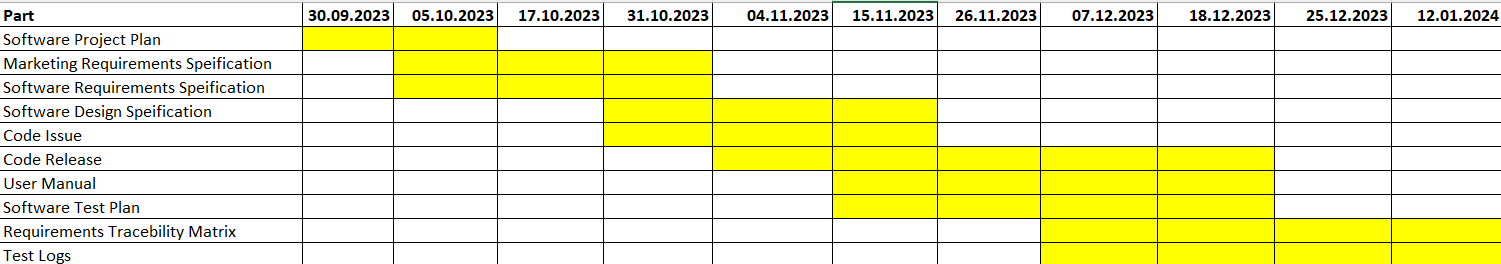


Рисунок 1 – План проекта

# Регламент проведения инспекции

Верификация рабочих продуктов является неотъемлемой частью процесса по обеспечению их качества. Современной технологией программирования выработаны специальные стандарты, подходы и механизмы проведения верификаций рабочих продуктов, в формате так называемых инспекций.

Инспекция – это мероприятие по обеспечению качества рабочих продуктов проектов по разработке ПО и иной деятельности, которая проводится разработчиками, возможно – с участием представителей заказчика.

Концептуально инспекция имеет следующие цели:

* обнаружить ошибки в функциях, логике, содержании или реализации рабочих продуктов на ранних этапах их разработки и предотвратить их наследование;
* рационально донести замысел или реализацию продукта до всех заинтересованных лиц (через их участие);
* оптимизировать, оценить или улучшить рабочий продукт.

#### **Перечень ролей участников инспекции и их обязанности**

Автор (Author) – сотрудник, разработавший инспектируемый рабочий продукт, либо сделавший инспектируемые изменения в существующем рабочем продукте.

Инспектор (Inspector) – сотрудник, ответственный за эффективную проверку инспектируемого рабочего продукта.

#### **Этапы инспекции**

Планирование инспекции.

При планировании инспекции коллективно выбирается дата, время, формат (очный или заочный) и платформа (при заочной инспекции) проведения инспекции.

Подготовка к инспекции.

Инспектор самостоятельно изучает предоставленный для инспекции рабочий продукт, используя накопленный опыт и стандарты.

Собрание по инспекции.

На собрании происходит обсуждение замечаний и рекомендаций инспектора по рабочему продукту. На собрании по инспекции обязательно присутствует инспектор и автор рабочего продукта, требующего инспекции. Присутствие остальных участников команды разработки по желанию.

Завершение инспекции.

Если рабочий продукт требует доработки, то автор фиксирует все замечания и рекомендации инспектора, разрабатывает план предстоящих работ и далее согласовывает его с инспектором. Если рабочий продукт не требует доработки, то инспектор подтверждает слияние рабочей ветки в «**master**» ветку.

#### **Перечень статусов**

1. Ошибка – проблема, которая найдена на той же фазе, на которой внесена.
2. Комментарий – это наблюдение, предложение, рекомендация или улучшение, предложенное для будущего выпуска рабочего продукта или вопрос, требующий разъяснения.

# Модель состояний задач

Каждая задача, являясь отражением делового процесса, проходит определенные состояния. Сначала идет создание задачи, потом идет выполнение работ по задаче, после выполнения задача завершается.

#### **Перечень состояний задач**

1. Ожидание – задача создана и находится в состоянии ожидания, над ней в данный момент никто не работает.
2. В работе – в данный момент задача находится в работе.
3. Тестирование – происходит тестирования внесенных изменений.
4. Закрыта – задача выполнена.

#### **Правила создания новой задачи**

Созданием новых задач могут заниматься все участники команды. Происходит это на начальном этапе, когда необходимо организовать структурированную работу над проектом и в процессе работы над проектом в случае присутствия в плане слишком объемных задач, требующих упрощения. Также, когда необходимо организовать баг-фиксинг или разработать новый функционал.

#### **Правила перехода задачи из состояния в состояние**

Состояния задач всегда идут последовательно друг за другом, в некоторых случаях пункты повторяться. Схема перехода из состояния в состояние показана на рисунке 2.

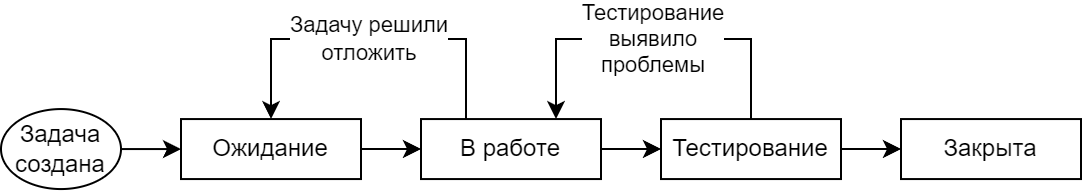


Рисунок  – Состояния задачи

# Презентация проекта

Была разработана презентация проекта, состоящая из 10 слайдов:

1. Титульный слайд [Рисунок 3]
2. Проблемы предметной области [Рисунок 4]
3. Решение проблем [Рисунок 5]
4. Архитектура программного средства [Рисунок 6]
5. Как работает [Рисунок 7]
6. Применение [Рисунок 8 – Рисунок 12]



Рисунок 3 – Титульная страница

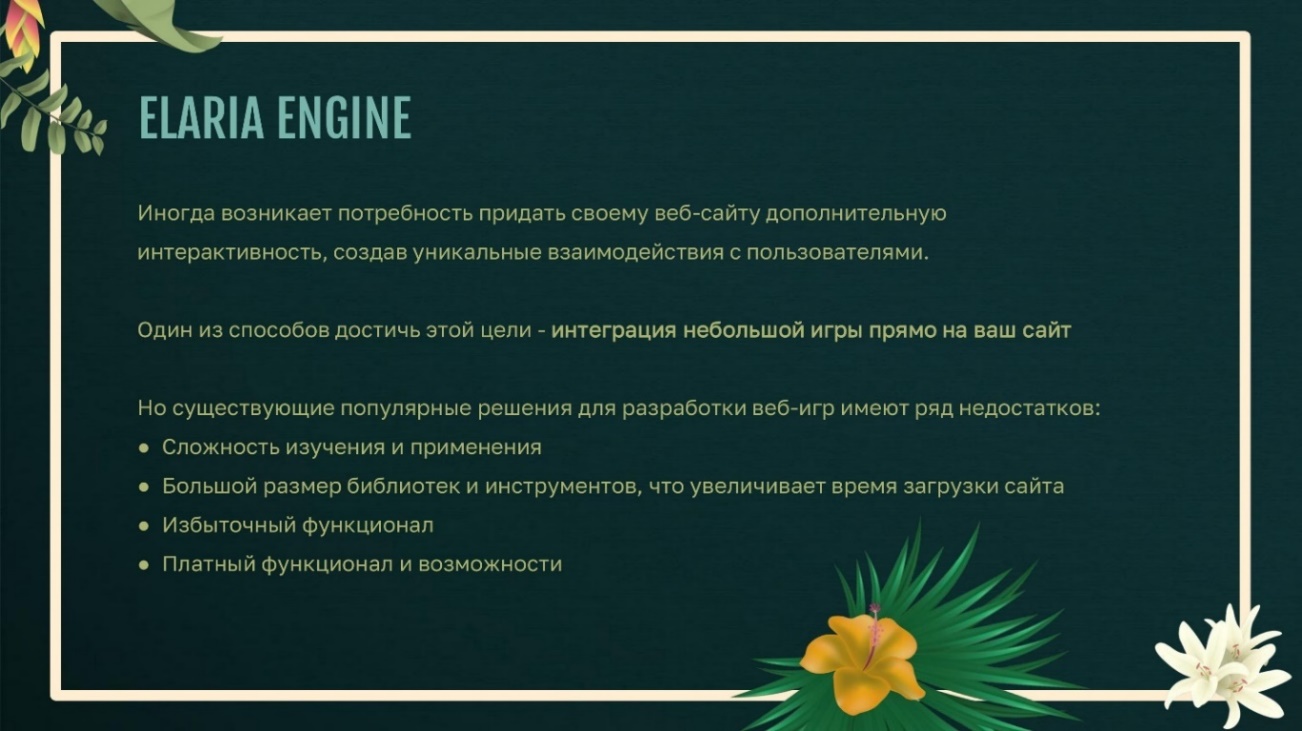


Рисунок 4 – Проблемы предметной области

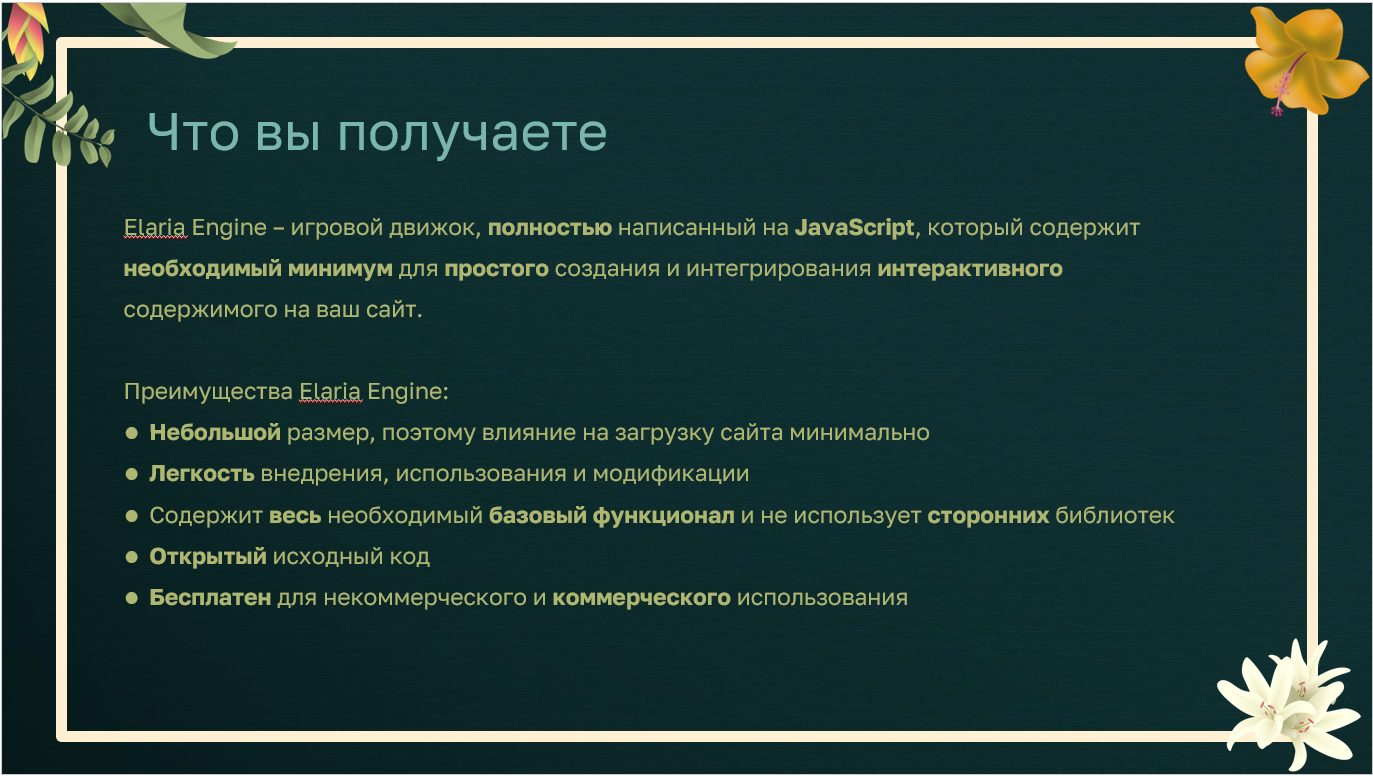


Рисунок 5 – Решение проблем

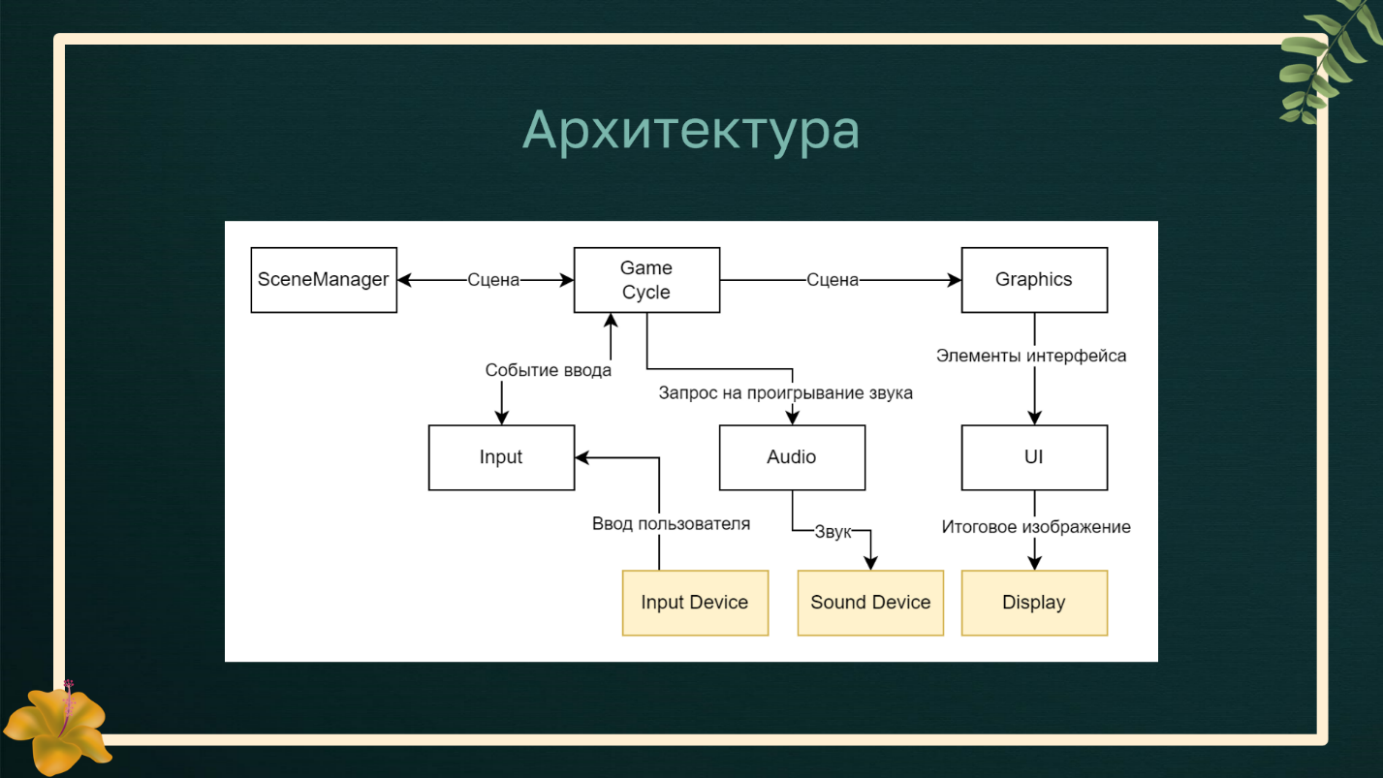


Рисунок 6 – Архитектура программного средства

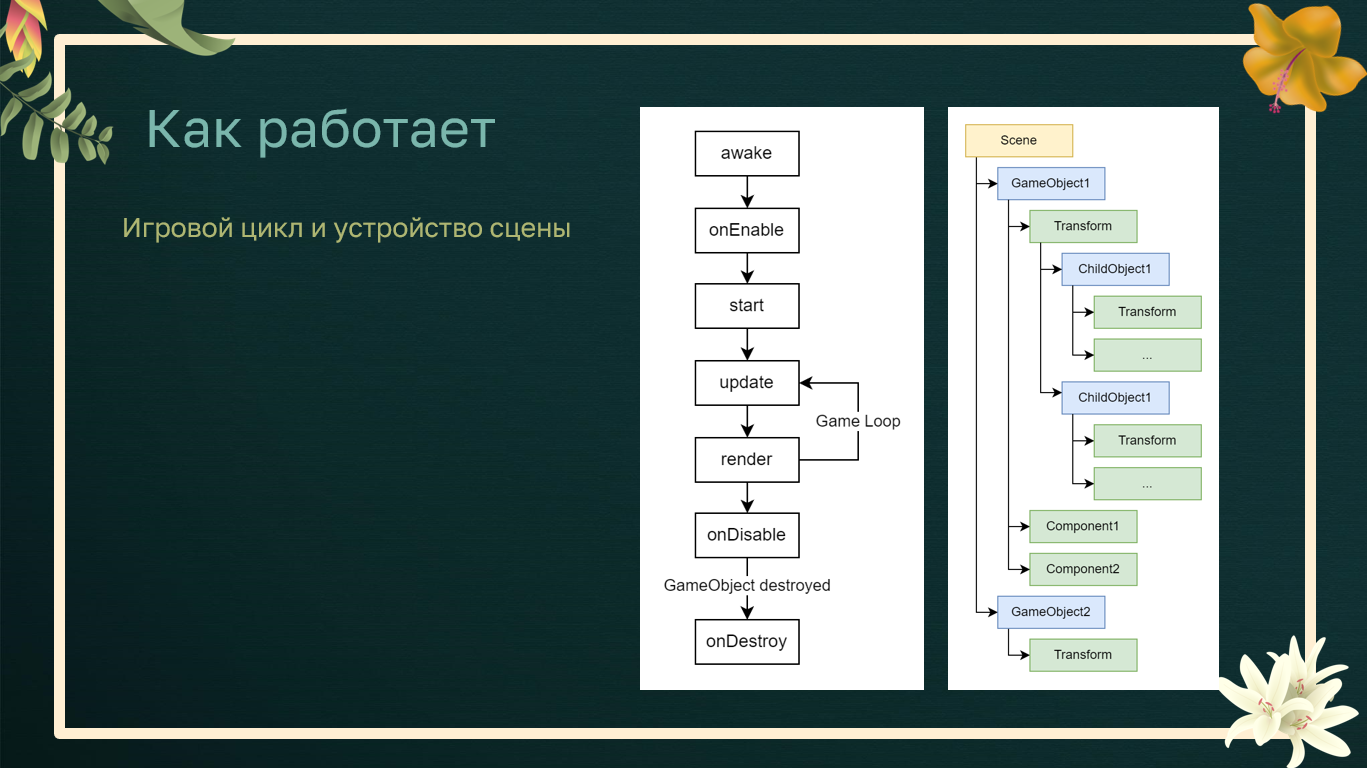


Рисунок 7 – Как работает

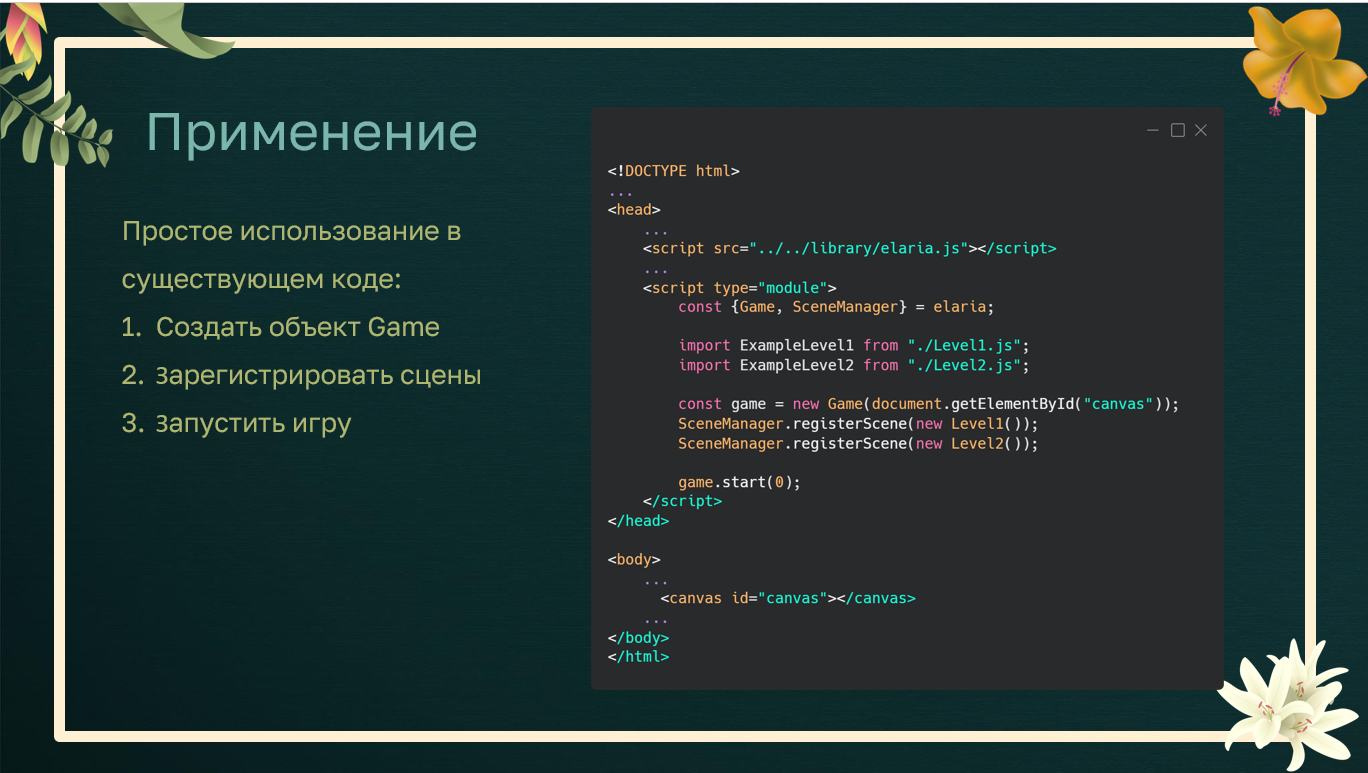


Рисунок – Применение. Начало использования.

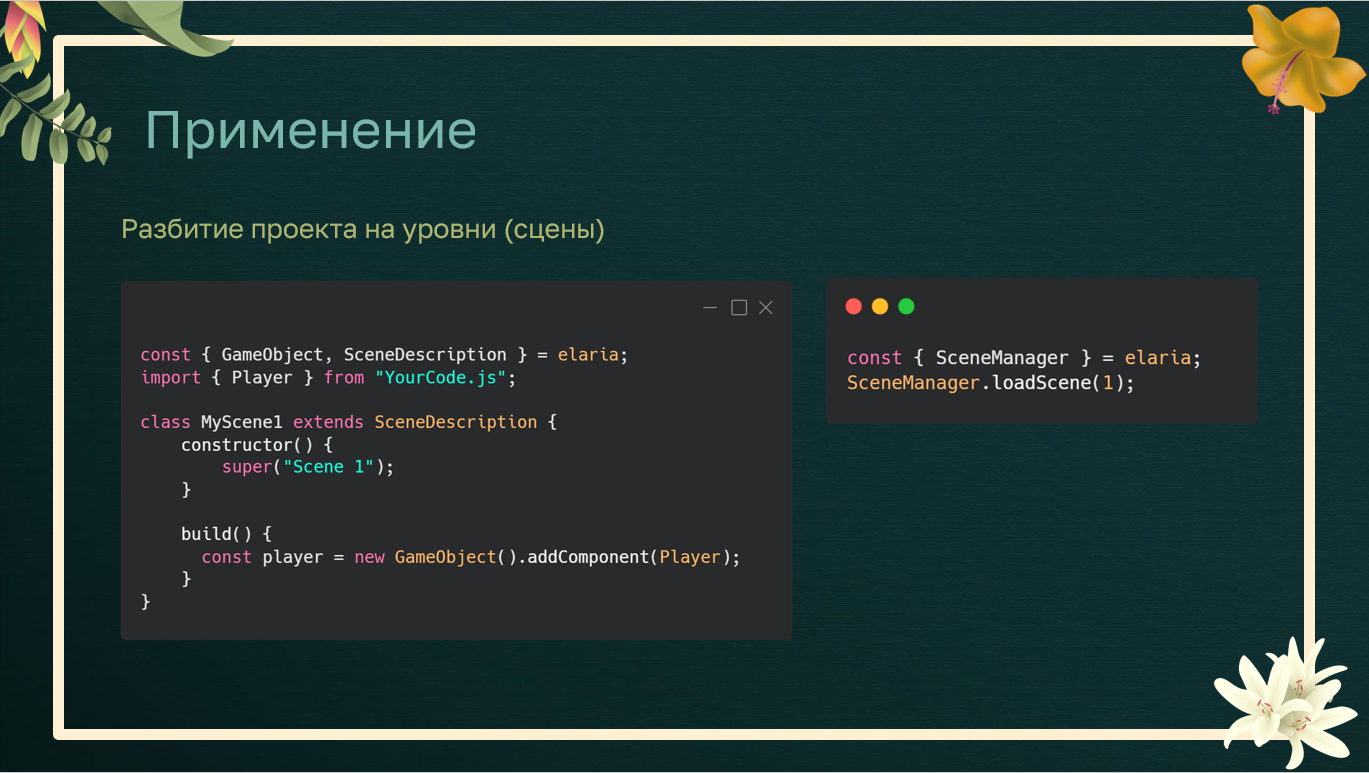


Рисунок – Применение. Создание своих сцен.

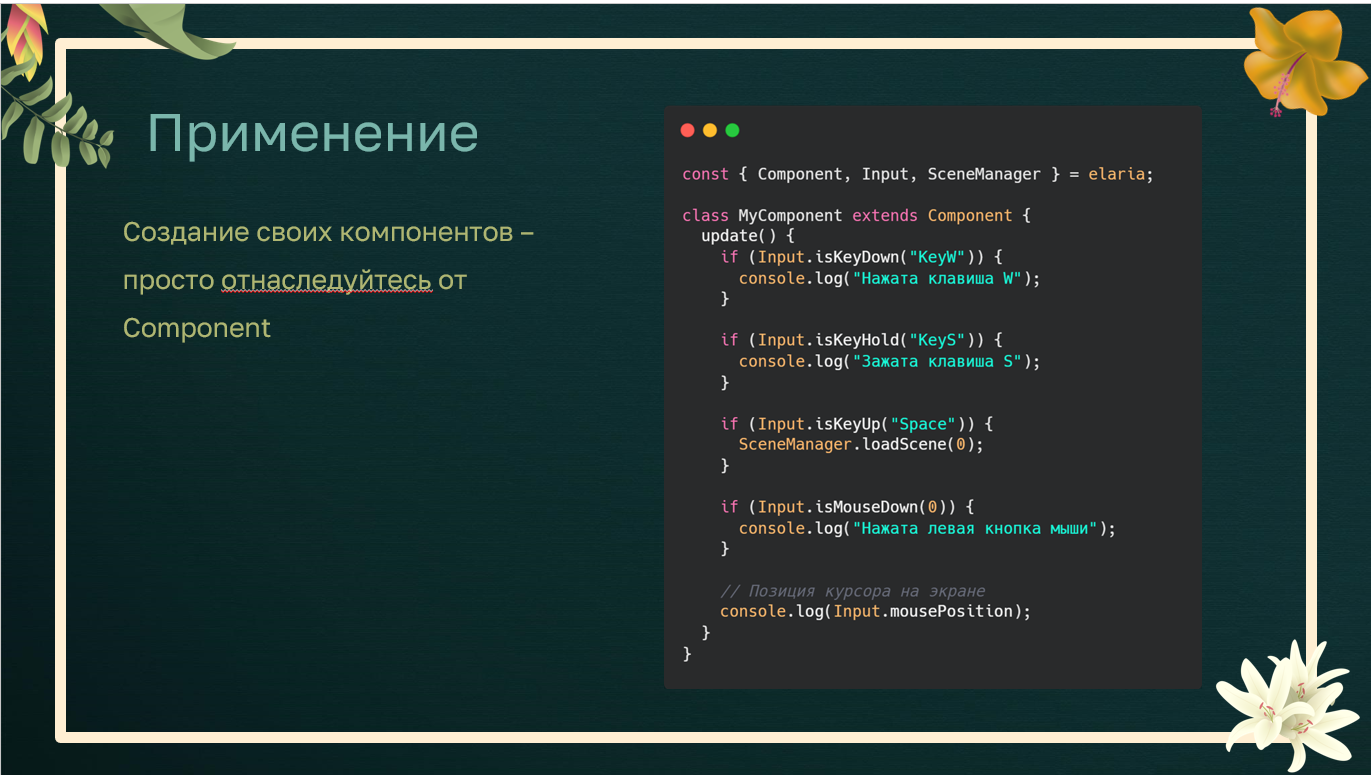


Рисунок – Применение. Создание своих компонентов.



Рисунок – Применение. Получение ввода пользователя

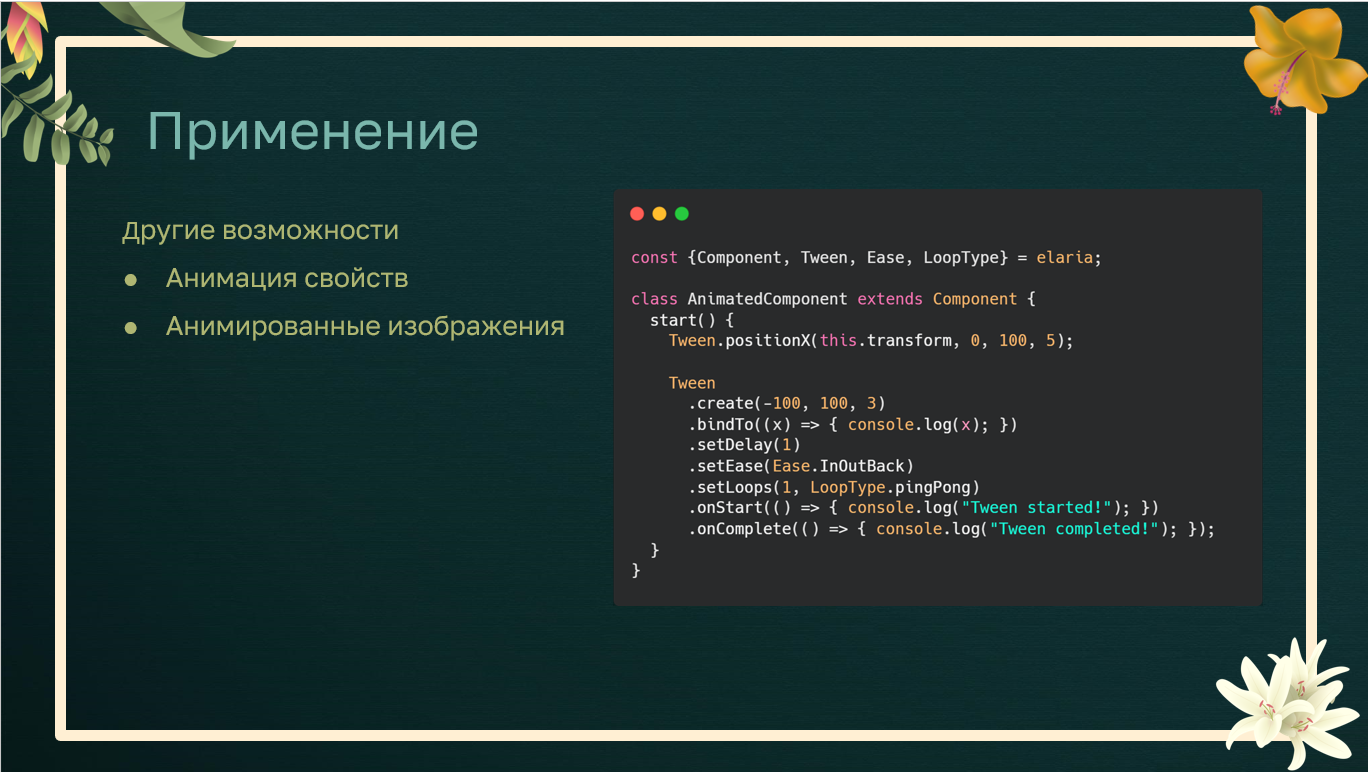


Рисунок – Применение. Другие возможности.

# Требования к проекту

Программный продукт «Игровой движок» предназначен для создания интерактивных элементов на сайте, с которыми пользователь может взаимодействовать.

Программный продукт «Игровой движок» состоит из следующих подсистем:

1. FE-1 – Ядро движка.
2. FE-2 – Графический модуль.
3. FE-3 – Модуль аудио.
4. FE-4 – Модуль получения пользовательского ввода.
5. FE-5 – Модуль пользовательского интерфейса.
6. FE-6 – Менеджер сцен.

#### **Матрица требований к подсистемам**

Ниже в таблице 1 представлены требования к подсистемам и их идентификаторы в виде матрицы требований.

Таблица 1 – матрица требований к подсистемам.

|  |  |
| --- | --- |
| **Требования** | **Идентификаторы** |
| Требования к ядру движка. | FE-1 CE |
| Ядро движка должно загружать начальную сцену. | REQ-CE-1 |
| Ядро движка должно запускать игровой цикл. | REQ-CE-2 |
| Ядро движка должно позволять создавать и уничтожать игровые объекты на сцене. | REQ-CE-3 |
| Ядро движка должно позволять добавлять, получать и удалять компоненты игровым объектам. | REQ-CE-4 |
| Требования к графическому модулю. | FE-2 GR |
| Графический модуль должен реализовывать компонент Sprite, который отрисовывает переданное изображение. | REQ-GR-1 |

*Окончание таблицы 1*

|  |  |
| --- | --- |
| **Требования** | **Идентификаторы** |
| Графический модуль должен реализовывать компонент AnimatedSprite, который отрисовывает последовательность изображений с указанной частотой кадров. | REQ-GR-2 |
| Графический модуль должен реализовывать компонент WorldText, который отрисовывает переданный текст. | REQ-GR-3 |
| Требования к аудио модулю. | FE-3 AU |
| Аудио модуль должен реализовывать компонент AudioSource, который проигрывает переданный аудиофайл. | REQ-AU-1 |
| Требования к модулю получения пользовательского ввода. | FE-4 UIN |
| Модуль пользовательского ввода должен уведомлять о событии нажатия клавиш клавиатуры. | REQ-UIN-1 |
| Модуль пользовательского ввода должен уведомлять о событии нажатия кнопок мыши. | REQ-UIN-2 |
| Модуль пользовательского ввода должен сохранять положение курсора. | REQ-UIN-3 |
| Требования к модулю пользовательского интерфейса. | FE-5 UI |
| Модуль пользовательского интерфейса должен реализовывать компонент RectTransform. Компонент RectTransform определяет положение и размеры графического элемента в экранном пространстве. | REQ-UI-1 |
| Модуль пользовательского интерфейса должен реализовывать компонент Button. Компонент Button обрабатывает событие нажатие левой кнопкой мыши на графический объект. | REQ-UI-2 |
| Модуль пользовательского интерфейса должен реализовывать компонент Text. Компонент Text выводит текст с определенным шрифтом и размером. | REQ-UI-3 |
| Требования к менеджеру сцен. | FE-6 SM |
| Менеджер сцен должен загружать переданную сцену. | REQ- SM-1 |
| Менеджер сцен должен автоматически выгружать текущую сцену после загрузки новой. | REQ- SM-2 |

#### **Расшифровка идентификаторов**

* CE – Core Engine (ядро движка).
* GR – Graphics (графический модуль).
* AU – Audio (аудио модуль).
* UIN – User’s input (пользовательский ввод).
* UI – User interface (пользовательский интерфейс).
* SM – Scene Manager (менеджер сцен).

# Разработка архитектуры проекта

Архитектура программного обеспечения относится к фундаментальным структурам программной системы и дисциплине создания таких структур и систем. Каждая структура включает элементы программного обеспечения, отношения между ними, а также свойства как элементов, так и отношений.

Одним из способов представления архитектуры проекта является архитектурно-контекстная диаграмма.

Архитектурно-контекстная диаграмма представляет систему как набор иерархических действий, в которой каждое действие преобразует некоторый объект или набор объектов. Цель такого представления — продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Была разработана архитектурно-контекстная диаграмма игрового движка, представленная на рисунке 13.

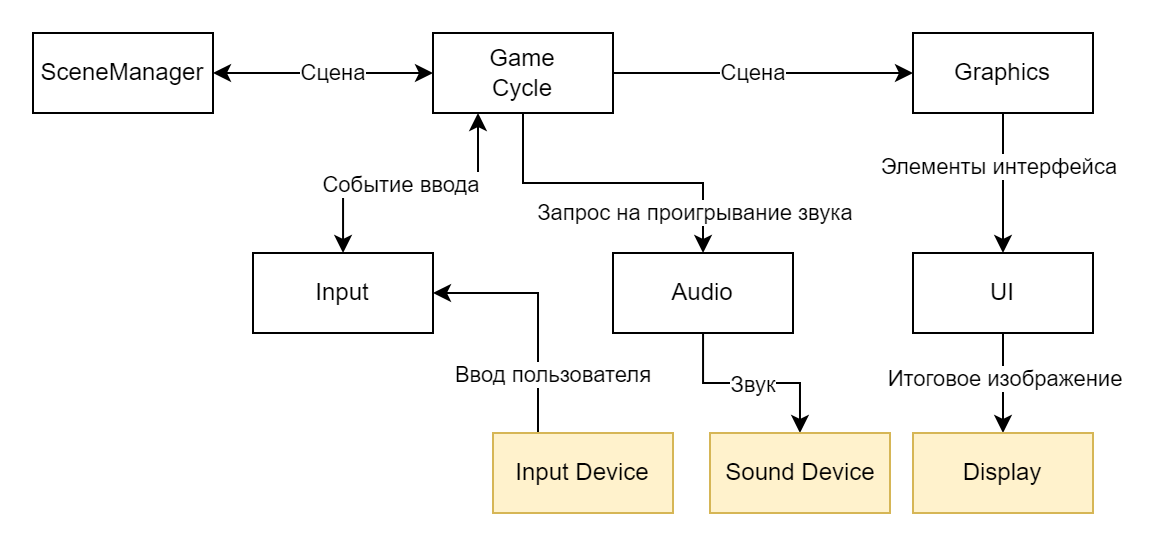


Рисунок 13 – Архитектурно-контекстная диаграмма игрового движка.

# Измерения проекта

Контроль над производственным процессом и его результатами является ключевым видом деятельности на современном предприятии, производящем программное обеспечение на заказ. В силу специфики такого продукта, как программное обеспечение, для оценки эффективности процесса и качества конечного продукта применяются особые методы.

Комплекс мероприятий, направленных на количественную оценку эффективности работы, называется программой измерений.

#### **Метрика эффективности процесса производства**

В рамках курсового проекта в качестве метрик эффективности процесса производства будут использоваться Faults Screening (FS) и Inspection Fault Density (IFD). Ниже представлено их описание.

* + 1. **Faults Screening**

FS = (Общее количество ошибок – Число не найденных ошибок)/ Общее количество ошибок \* 100%;

Стратегическая цель метрики – повысить качество разрабатываемого ПО.

Изучаемый объект метрики – проект, измеряемый атрибут – эффективность обнаружения дефектов.

Единица измерения – в процентах (%)

* + 1. **Inspection Fault Density (IFD)**

IFD = (Количество найденных ошибок / Размер рабочего продукта)

Стратегическая цель метрики – повысить качество разрабатываемого ПО.

Изучаемый объект метрики – инспекция, измеряемый атрибут – плотность найденных в ходе инспекции ошибок.

Количество найденных ошибок – ошибки, обнаруженные на этапе инспектирования.

Единица измерения – ошибка / <страница, требование**,** LOC, тест>.

#### **Метрика качества продукта**

В рамках курсового проекта в качестве метрики качества продукта была выбрана In Process Faults (IPF). Ниже представлено ее описание.

* + 1. **In Process Faults**

IPF = (Число обнаруженных ошибок до выпуска его релиза) / LOC

Стратегическая цель метрики – повысить качество разрабатываемого ПО.

Изучаемый объект метрики – продукт, измеряемый атрибут – плотность неполадок.

Число обнаруженных ошибок – количество ошибок, обнаруженных на этапе тестирования

Единица измерения – неполадка / LOC.

# Перечень задач проекта

Были выделены следующие задачи в ходе реализации программного средства.

#### **Задачи подсистемы FE-1**

* Разработать загрузку начальной сцены (см. REQ-CE-1).
* Реализовать игровой цикл и его запуск (см. REQ-CE-2).
* Реализовать функционал создания и удаления игровых объектов на сцене (см. REQ-CE-3).
* Реализовать возможность добавления, получения и удаления компонентов игровых объектов (см. REQ-CE-4).

#### **Задачи подсистемы FE-2**

* Разработать компонент Sprite (см. REQ-GR-1).
* Разработать компонент AnimatedSprite (см. REQ-GR-2).
* Разработать компонент WordText (см. REQ-GR-3).

#### **Задачи подсистемы FE-3**

* Разработать компонент AudioSource (см. REQ-AU-3).

#### **Задачи подсистемы FE-4**

* Реализовать функционал уведомления события нажатия клавиш клавиатур (см. REQ-UIN-1).
* Реализовать функционал уведомления события нажатия кнопок мыши (см. REQ-UIN-2).
* Реализовать функционал запоминания положения курсора (см. REQ-UIN-3).

#### **Задачи подсистемы FE-5**

* Разработать компонент RectTransform (см. REQ-UI-1).
* Разработать компонент Button (см. REQ-UI-2).
* Разработать компонент Text (см. REQ-UI-3).

#### **Задачи подсистемы FE-6**

* Реализовать функционал загрузки сцены(см. REQ-SM-1).
* Реализовать функционал загрузки текущей сцены после загрузки новой сцены (см. REQ-SM-2).

# Правила по кодированию

Для создания качественного кода на любом языке программирования, обладающего таким свойствами, как удобочитаемость (readability) и понятность (understandability), необходимо следовать хорошо определённым стандартам и руководствам. Особенно это актуально при коллективной разработке программ.

Любой стандарт кодирования призван определить набор правил, которые способствуют разработке более единообразного кода и минимизации числа общераспространенных ошибок в нем, не ущемляя при этом права разработчика на творчество.

Рекомендации:

1. Вставляйте дополнительный перевод строки туда, где это сделает код более читаемым. Не должно быть более 9 строк кода подряд без вертикального отступа.
2. Не должно быть неиспользуемых переменных в коде.

Требования:

1. Отступы обязательны и равны 4 пробелам.
2. Каждая операция должна заканчиваться точкой с запятой.
3. Строковые литералы должны оборачиваться в двойные кавычки. В одинарные только в случае необходимости разместить двойные внутри значения литерала.
4. Для именования переменных, функций и методов должен использоваться формат camelCase. Классы должны именоваться в PascalCase, то есть с большой буквы.
5. Вместо символа == должно использовать строгое сравнение с помощью ===.
6. Операторы и операнды должны разделяться пробелами.
7. Фигурные скобки пишутся в так называемом «египетском» стиле с открывающей скобкой на той же строке, что и соответствующее ключевое слово – не на новой строке. Перед открывающей скобкой должен быть пробел.
8. Переменные и функции должны быть объявлены до того, как будут использованы в коде. Исключение – методы классов, они могут быть объявлены в любом порядке.
9. Импортируемые модули должны заканчиваться на .js.

# Разработка плана тестирования проекта

В данной в главе описываются ситуации для тестирования программного средства на выполнение всех требований, описанных в главе 5.

#### **Тесты для подсистемы FE-1**

**Тест TEST\_CE\_1**

Тестируемые требования: REQ-CE-1, REQ-CE-2.

Тестовая ситуация: Зарегистрировано две сцены. Начальной сценой указана вторая.

Ожидаемый результат: Загрузилась вторая сцена. Игровой цикл запустился для объектов второй сцены.

**Тест TEST\_CE\_2**

Тестируемые требования: REQ-CE-1, REQ-CE-2.

Тестовая ситуация: Зарегистрировано две сцены. Начальная сцена не указана.

Ожидаемый результат: Загрузилась первая сцена. Игровой цикл запустился для объектов первой сцены.

**Тест TEST\_CE\_3**

Тестируемые требования: REQ-CE-1, REQ-CE-2.

Тестовая ситуация: Не зарегистрировано ни одной сцены.

Ожидаемый результат: В консоль вывелось сообщение с предупреждением “Not a single scene registered! Register at least one scene to start the game”, игровой цикл не запустился.

**Тест TEST\_CE\_4**

Тестируемые требования: REQ-CE-3.

Тестовая ситуация: Создание нового игрового объекта.

Ожидаемый результат: Игровой объект успешно создан.

**Тест TEST\_CE\_5**

Тестируемые требования: REQ-CE-3.

Тестовая ситуация: Уничтожение игрового объекта.

Ожидаемый результат: Игровой объект успешно уничтожен, у всех его компонентов вызвались методы в следующем порядке: onDisable, а затем onDestroy.

**Тест TEST\_CE\_6**

Тестируемые требования: REQ-CE-4.

Тестовая ситуация: Добавление компонента на игровой объект.

Ожидаемый результат: Компонент успешно добавлен на игровой объект и возвращен в качестве результата.

**Тест TEST\_CE\_7**

Тестируемые требования: REQ-CE-4.

Тестовая ситуация: Получение компонента, который есть на игровом объекте.

Ожидаемый результат: Компонент успешно найден и возвращен в качестве результата.

**Тест TEST\_CE\_8**

Тестируемые требования: REQ-CE-4.

Тестовая ситуация: Получение компонента, которого нет на игровом объекте.

Ожидаемый результат: Возвращено undefined.

**Тест TEST\_CE\_9**

Тестируемые требования: REQ-CE-4.

Тестовая ситуация: Удаление компонента у игрового объекта.

Ожидаемый результат: Компонент успешно удален.

**Тест TEST\_CE\_10**

Тестируемые требования: REQ-CE-4.

Тестовая ситуация: Удаление компонента, несуществующего на игровом объекте.

Ожидаемый результат: Ничего не происходит.

#### **Тесты для подсистемы FE-2**

**Тест TEST\_GR\_1**

Тестируемые требования: REQ-GR-1.

Тестовая ситуация: Отрисовка изображения с указанной шириной и высотой.

Ожидаемый результат: Переданное изображение отрисовывается и имеет соответствующие ширину и высоту.

**Тест TEST\_GR\_2**

Тестируемые требования: REQ-GR-2.

Тестовая ситуация: Отрисовка анимированного изображения из одного в соответствии с переданным разбиением и частотой кадров.

Ожидаемый результат: Переданное изображение корректно разбивается на кадры анимации. Анимация проигрывается с указанной частотой кадров.

**Тест TEST\_GR\_3**

Тестируемые требования: REQ-GR-3.

Тестовая ситуация: Вывод текста «Hello, world!», со шрифтом Times New Roman, размером 22 пункта и красного цвета в мировых координатах.

Ожидаемый результат: Выводится текст «Hello, world!» в мировых координатах объекта, в соответствии с указанными параметрами.

#### **Тесты для подсистемы FE-3**

**Тест TEST\_AU\_1**

Тестируемые требования: REQ-AU-1.

Тестовая ситуация: Проигрывание переданного звука.

Ожидаемый результат: Переданный звук проигрывается.

#### **Тесты для подсистемы FE-4**

**Тест TEST\_UIN\_1**

Тестируемые требования: REQ-UIN-1.

Тестовая ситуация: В момент шага игрового цикла клавиша H нажата в первый раз, клавиша J зажата, а клавиша K была отпущена.

Ожидаемый результат: Полученное состояние клавиш в данный шаг игрового цикла: клавиша H – нажата первый раз, клавиша J – зажата, а клавиша K – отпущена.

**Тест TEST\_UIN\_2**

Тестируемые требования: REQ-UIN-2.

Тестовая ситуация: В момент шага игрового цикла левая кнопка мыши нажата в первый раз, правая кнопка мыши зажата, а средняя кнопка мыши была отпущена.

Ожидаемый результат: Полученное состояние кнопок мыши в данный шаг игрового цикла: левая кнопка – нажата первый раз, правая кнопка – зажата, а средняя кнопка мыши – отпущена.

**Тест TEST\_UIN\_3**

Тестируемые требования: REQ-UIN-3.

Тестовая ситуация: Курсор мыши находится в центре экрана.

Ожидаемый результат: Полученные координаты курсора мыши равны половине ширины и высоты экрана.

#### **Тесты для подсистемы FE-5**

**Тест TEST\_UI\_1**

Тестируемые требования: REQ-UI-1.

Тестовая ситуация: Создание игрового объекта с компонентом RectTransform.

Ожидаемый результат: Игровой объект успешно создан с компонентом RectTransform.

**Тест TEST\_UI\_2**

Тестируемые требования: REQ-UI-1.

Тестовая ситуация: Установка значений ширины и высоты компонента RectTransform.

Ожидаемый результат: Значения ширины и высоты успешно установлены и соответствуют ожидаемым.

**Тест TEST\_UI\_3**

Тестируемые требования: REQ-UI-1.

Тестовая ситуация: Установка глобального положения компонента RectTransform.

Ожидаемый результат: Значения глобального положения соответствуют ожидаемым, а значения локального положения были правильно пересчитаны.

**Тест TEST\_UI\_4**

Тестируемые требования: REQ-UI-1.

Тестовая ситуация: Установка локального положения компонента RectTransform.

Ожидаемый результат: Значения локального положения успешно установлены и соответствуют ожидаемым.

**Тест TEST\_UI\_5**

Тестируемые требования: REQ-UI-1.

Тестовая ситуация: Установка значения pivot компонента RectTransform.

Ожидаемый результат: Значение pivot успешно установлено, координаты локального положения пересчитаны в соответствии с новым значением.

**Тест TEST\_UI\_6**

Тестируемые требования: REQ-UI-1.

Тестовая ситуация: Изменение размеров родительского объекта.

Ожидаемый результат: Компонент RectTransform автоматически реагирует на изменение размеров родительского объекта и подстраивает свои параметры в соответствии с параметрами якорей.

**Тест TEST\_UI\_7**

Тестируемые требования: REQ-UI-1.

Тестовая ситуация: Получение центральной точки прямоугольника.

Ожидаемый результат: Компонент RectTransform возвращает правильные координаты центральной точки описываемого им прямоугольника.

**Тест TEST\_UI\_8**

Тестируемые требования: REQ-UI-2.

Тестовая ситуация: Наведение и нажатие левой кнопкой по объекту Button.

Ожидаемый результат: Кнопка реагирует при наведении, а при нажатии вызывается соответствующий обработчик события.

**Тест TEST\_UI\_9**

Тестируемые требования: REQ-UI-3.

Тестовая ситуация: Вывод текста «Hello, world!», со шрифтом Times New Roman, размером 22 пункта и красного цвета.

Ожидаемый результат: Выводится текст «Hello, world!» в координатах объекта, в соответствии с указанными параметрами.

#### **Тесты для подсистемы FE-6**

**Тест TEST\_SM\_1**

Тестируемые требования: REQ-SM-1, REQ- SM-2.

Тестовая ситуация: Загрузка зарегистрированной сцены по её индексу.

Ожидаемый результат: Все игровые объекты активной сцены уничтожаются. Новая сцена загружается, создается и становится активной.

**Тест TEST\_SM\_2**

Тестируемые требования: REQ-SM-1.

Тестовая ситуация: Загрузка несуществующей сцены.

Ожидаемый результат: Ошибка “Error: Invalid scene index”.

#### **Матрица покрытия тестами требований**

В ходе проектирования тестовых ситуаций была составлена матрица покрытий тестами требований, показанная на рисунке 14.

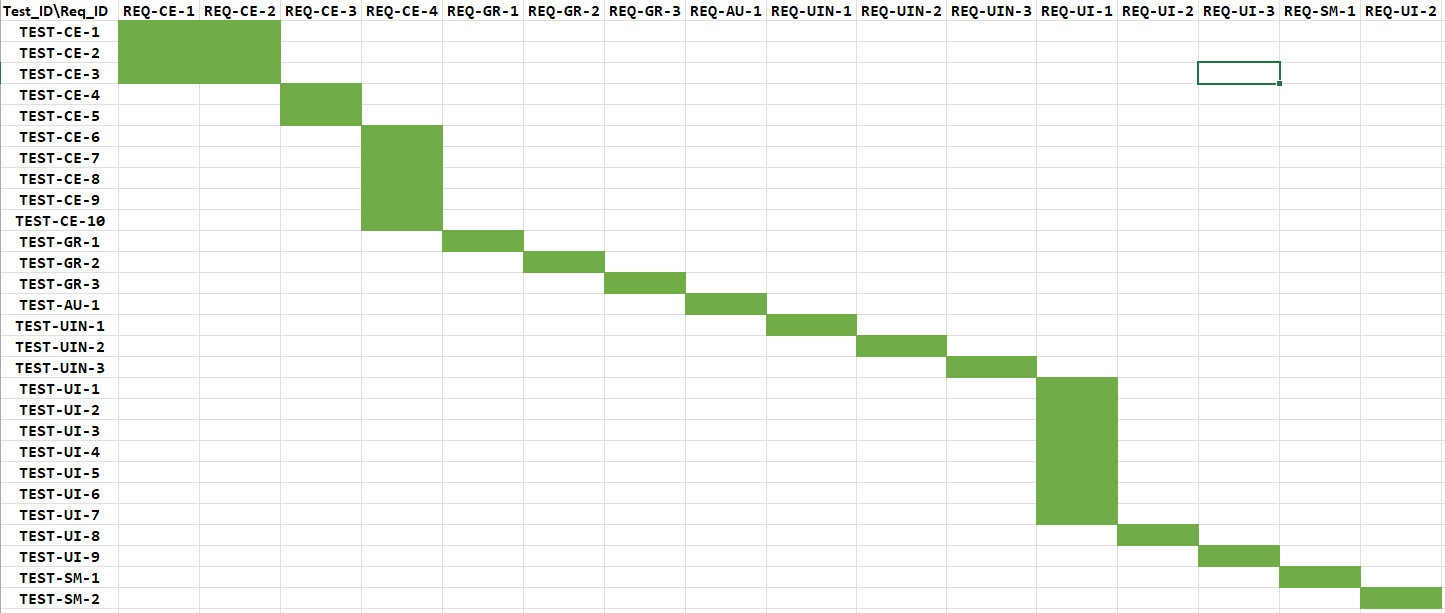


Рисунок 14 – Матрица покрытия тестами требований

# Тестирование проекта

В данной главе описываются результаты тестирования по ситуациям описанным в главе 10.

#### **Результаты тестирования подсистемы FE-1**

**Тест** **TEST\_CE\_1**

Видимый результат: Загрузилась вторая сцена. Игровой цикл запустился для объектов второй сцены.

Резюме: Тест пройден.

**Тест** **TEST\_CE\_2**

Видимый результат: Загрузилась первая сцена. Игровой цикл запустился для объектов первой сцены.

Резюме: Тест пройден.

**Тест** **TEST\_CE\_3**

Видимый результат: Вывелось сообщение об ошибке “Invalid scene index”. Игровой цикл не запустился.

Резюме: Тест не пройден.

**Тест** **TEST\_CE\_4**

Видимый результат: Игровой объект успешно создан.

Резюме: Тест пройден.

**Тест** **TEST\_CE\_5**

Видимый результат: Игровой объект успешно уничтожен, у всех его компонентов вызвались методы в следующем порядке: onDisable, а затем onDestroy.

Резюме: Тест пройден.

**Тест** **TEST\_CE\_6**

Видимый результат: Компонент успешно добавлен на игровой объект и возвращен в качестве результата.

Резюме: Тест пройден.

**Тест** **TEST\_CE\_7**

Видимый результат: Получение компонента, который есть на игровом объекте.

Резюме: Тест пройден.

**Тест** **TEST\_CE\_8**

Видимый результат: Возвращено undefined.

Резюме: Тест пройден.

**Тест** **TEST\_CE\_9**

Видимый результат: Компонент успешно удален.

Резюме: Тест пройден.

**Тест** **TEST\_CE\_10**

Видимый результат: Ничего не произошло.

Резюме: Тест пройден.

#### **Результаты тестирования подсистемы FE-2**

**Тест TEST\_GR\_1**

Видимый результат: Переданное изображение отрисовывается и имеет соответствующие ширину и высоту.

Резюме: Тест пройден.

**Тест TEST\_GR\_2**

Видимый результат: Переданное изображение корректно разбивается на кадры анимации. Анимация проигрывается с указанной частотой кадров.

Резюме: Тест пройден.

**Тест TEST\_GR\_3**

Видимый результат: Выводится текст «Hello, world!» в мировых координатах объекта, в соответствии с указанными параметрами.

Резюме: Тест пройден.

#### **Результаты тестирования подсистемы FE-3**

**Тест TEST\_AU\_1**

Видимый результат: Переданный звук проигрывается.

Резюме: Тест пройден.

#### **Результаты тестирования подсистемы FE-4**

**Тест TEST\_UIN\_1**

Видимый результат: Полученное состояние клавиш в данный шаг игрового цикла: клавиша H – нажата первый раз, клавиша J – зажата, а клавиша K – отпущена.

Резюме: Тест пройден.

**Тест TEST\_UIN\_2**

Видимый результат: Полученное состояние кнопок мыши в данный шаг игрового цикла: левая кнопка – нажата первый раз, правая кнопка – зажата, а средняя кнопка мыши – отпущена.

Резюме: Тест пройден.

**Тест TEST\_UIN\_3**

Видимый результат: Полученные координаты курсора мыши равны половине ширины и высоты экрана.

Резюме: Тест пройден.

#### **Результаты тестирования подсистемы FE-5**

**Тест TEST\_UI\_1**

Видимый результат: Игровой объект успешно создан с компонентом RectTransform.

Резюме: Тест пройден.

**Тест TEST\_UI\_2**

Видимый результат: Значения ширины и высоты успешно установлены и соответствуют ожидаемым.

Резюме: Тест пройден.

**Тест TEST\_UI\_3**

Видимый результат: Значения глобального положения соответствуют ожидаемым, а значения локального положения были правильно пересчитаны.

Резюме: Тест пройден.

**Тест TEST\_UI\_4**

Видимый результат: Значения локального положения успешно установлены и соответствуют ожидаемым.

Резюме: Тест пройден.

**Тест TEST\_UI\_5**

Видимый результат: Значение pivot успешно установлено, координаты локального положения пересчитаны в соответствии с новым значением.

Резюме: Тест пройден.

**Тест TEST\_UI\_6**

Видимый результат: Компонент RectTransform автоматически реагирует на изменение размеров родительского объекта и подстраивает свои параметры в соответствии с параметрами якорей.

Резюме: Тест пройден.

**Тест TEST\_UI\_7**

Видимый результат: Компонент RectTransform возвращает правильные координаты центральной точки описываемого им прямоугольника.

Резюме: Тест пройден.

**Тест TEST\_UI\_8**

Видимый результат: Кнопка реагирует при наведении, а при нажатии вызывается соответствующий обработчик события.

Резюме: Тест пройден.

**Тест TEST\_UI\_9**

Видимый результат: Выводится текст «Hello, world!» в координатах объекта, в соответствии с указанными параметрами.

Резюме: Тест пройден.

#### **Результаты тестирования подсистемы FE-6**

**Тест TEST\_SM\_1**

Видимый результат: Все игровые объекты активной сцены уничтожаются. Новая сцена загружается, создается и становится активной.

Резюме: Тест пройден.

**Тест TEST\_SM\_2**

Видимый результат: Ошибка “Error: Invalid scene index”.

Резюме: Тест пройден.

# Заключение

В рамках курсовой работы было разработано программное средство «Игровой движок» с использованием подходов коллективной промышленной разработки, для чего были решены следующие поставленные задачи:

1. Разработан план проекта.
2. Разработан регламент проведения инспекции.
3. Разработана модель состояний задач.
4. Разработана презентация проекта.
5. Разработаны требования к проекту.
6. Разработана архитектура проекта.
7. Разработаны измерения проекта.
8. Разработан перечень задач проекта.
9. Разработаны рекомендации по кодированию.
10. Разработан план тестирования проекта.
11. Протестирован проект.

Таким образом, цель данной курсовой работы была достигнута.

# Список литературы

1. Гриняк В.М. Лекции по дисциплине «Технологии коллективной промышленной разработки информационных систем» / Гриняк В.М. [Электронный ресурс] // Blackboard : [сайт]. — URL: https://bb.dvfu.ru/webapps/blackboard/content/listContent.jsp?course\_id=\_5025\_1&content\_id=\_172365\_1&mode=reset (дата обращения: 12.04.2024).