МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2022 г.

РАЗРАБОТКА ТЕСТ-ИГРЫ ПО ОСНОВАМ ФИНАНСОВОЙ ГРАМОТНОСТИ ДЛЯ УЧЕНИКОВ НАЧАЛЬНЫХ КЛАССОВ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090401.2022.406 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____Е.С. Ярош
«___»_____2022 г.

Автор работы,
студент группы КЭ-406
_____Р.Р. Салахиев
«___»_____2022 г.

Нормоконтролёр,
к.п.н., доцент каф. ЭВМ
_____М.А. Алтухова
«___»_____2022 г.

Челябинск-2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2022 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Салахиеву Руслану Рашитовичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Разработка тест-игры по основам финансовой грамотности для учеников начальных классов» утверждена приказом по университету от 12 декабря 2021 г. №308/141.

2. **Срок сдачи студентом законченной работы:** 23 мая 2022 г.

3. **Исходные данные к работе.**

Тест-игра разрабатывается с целью повышения уровня финансовой грамотности учеников начальных классов. Игровой элемент заключается в наличии на каждом уровне диалога между двумя персонажами, на основе которого проходит тест с вопросами на выбор правильного варианта ответа. Игровыми персонажами являются «Пионер Волька» и «Старик Цифрабыч».

Система должна соответствовать следующим функциональным требованиям:

– наличие двух видов пользователей: простой пользователь и редактор;

– предоставление простому пользователю возможности выбирать уровень для прохождения;

– вывод на каждом уровне сцены с диалогом двух персонажей в виде текста;

– предоставление простому пользователю возможности пропустить диалог;

– предоставление простому пользователю возможности прохождения теста путем ответов на вопросы с выбором одного из предложенных вариантов;

– распознавание выбранных пользователем вариантов ответа и в случае правильного ответа на все вопросы засчитывать тест, в случае наличия ошибок предлагать пройти уровень заново;

– предоставление простому пользователю возможности повторного прохождения уровня;

– перевод простого пользователя переходить в режим редактора при корректном вводе пароля;

– предоставление возможности редактору изменять пароль;

– предоставление возможности редактору добавлять, удалять и переименовывать уровни;

– предоставление возможности редактору добавлять, удалять и изменять диалоги на разных уровнях;

– предоставление возможности редактору добавлять, удалять и изменять вопросы в тестах на разных уровнях.

Система должна соответствовать следующим нефункциональным требованиям:

приложение должно быть разработано на платформе, предоставляющей возможность бесплатного распространения готовой системы.

4. **Перечень подлежащих разработке вопросов:**

– провести анализ приложений тестов с аналогичным функционалом;

– анализ современных программных технологий для разработки собственного программного обеспечения;

– проектировка и разработка собственной тест-игры;

– оценка работоспособности разработанного программного комплекса в различных режимах.

– анализ родственных разработок, уточнение требований к приложению;

– выбор среды реализации;

– разработка архитектуры приложения и основных технических решений,

– реализация приложения;

– тестирование и оценка работоспособности приложения.

5. **Дата выдачи задания:** 1 декабря 2021 г.

Руководитель работы _____/Е.С. Ярош/

Студент _____/Р.Р. Салахиев/

КАЛЕНДАРНЫЙ ПЛАН

Этап	**Срок сдачи**	**Подпись руководителя**
Введение и обзор литературы, изучение аналогов и определение необходимого функционала	07.02.2022	
Разработка модели, проектирование системы	07.03.2022	
Реализация системы	04.04.2022	
Тестирование, отладка, эксперименты	10.05.2022	
Компоновка текста работы и сдача на нормоконтроль	16.05.2022	
Подготовка презентации и доклада	24.05.2022	

Руководитель работы _____/Е.С. Ярош/

Студент _____/Р.Р. Салахиев/

АННОТАЦИЯ

Р.Р. Салахиев. Разработка тест-игры по основам финансовой грамотности для учеников начальных классов. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2022, 65 с., 20 ил., библиогр. список – 21 наим., 7 прил.

Выпускная квалификационная работа была выполнена с целью разработки тест-игры с инструментарием для редактирования и изменения текстов и вопросов. Были изучены существующие методы разработки компьютерных игр, выполнен анализ требований, проектирование и тестирование разработанного приложения. Для разработки использовался движок Unity и язык программирования высокого уровня C#. Основной целью разработки являлось создание встроенного редактора с возможностью изменения внутри игровых текстов.

Результатом выполненной работы является правильно функционирующая тест-игра.

СОДЕРЖАНИЕ

ТЕРМИНЫ И СОКРАЩЕНИЯ

Таблица 1 – Перечень терминов и сокращений

Термин	Сокращение	Описание
Игровой движок		Программное обеспечение для разработки игр, которое предоставляет для этого все необходимые инструменты
Билд		Подготовленная для использования программа
Центральный Банк Российской Федерации	ЦБ РФ	

ВВЕДЕНИЕ

Актуальность темы, связанной с разработкой программного обеспечения для повышения финансовой грамотности среди учеников начальных классов, заключается в том, что умение формировать бюджет, откладывать часть средств на подушку безопасности, заполнять различные формы и документы и другие важные навыки необходимы в современном мире. По сообщению Центрального Банка Российской Федерации, с 2022 года в школах с 1 по 9 класс будут проводиться уроки финансовой грамотности для повышения финансовой грамотности среди населения [1].

Целью представленной выпускной квалификационной работы является разработка программного продукта, обеспечивающего необходимый функционал для создания и редактирования тестов с элементом игры.

Для достижения поставленной цели, необходимо решить следующие поставленные задачи:

1) рассмотреть существующие аналоги разрабатываемого приложения и осуществить постановку задачи;

2) провести детальный анализ современных программных технологий для разработки программного обеспечения;

3) провести анализ требований и спроектировать компьютерную тест-игру;

4) реализовать программное обеспечение;

5) провести тестирование реализованного программного обеспечения.

Данная работа состоит из пяти глав. В первой описаны существующие аналоги и технологические решения. Во второй представлены функциональные и нефункциональные требования. В третьей описан проект решения. В четвертой представлена реализация системы. В пятой описано тестирование представленной системы.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

На сегодняшний день существует множество различных вариантов для проведения тестирования, однако каждый из них имеет свои недостатки [2]. Разберем решения, аналогичные разрабатываемой системе по следующим параметрам:

– необходимость подключения к интернету для использования функционала представленного решения;

– возможность редактирования тестов и вопросов;

– наличие игрового элемента, представляющего собой текстовый диалог между двумя игровыми персонажами;

– стоимость представленного решения.

Moodle – представляет собой открытое веб-приложение, позволяющее создавать системы обучения и тестирования [3].

Данное решение является самой популярной платформой для ведения электронного обучения и тестирования, для использования требуется постоянное подключение к интернету. При создании тестов на выбор дается 16 различных вариантов вопросов, есть функционал редактирования и изменения созданных тестов. Данное решение стоит от 11000 руб./год.

На рисунке 1 представлен интерфейс выбора варианта вопроса в области создания тестов Moodle.

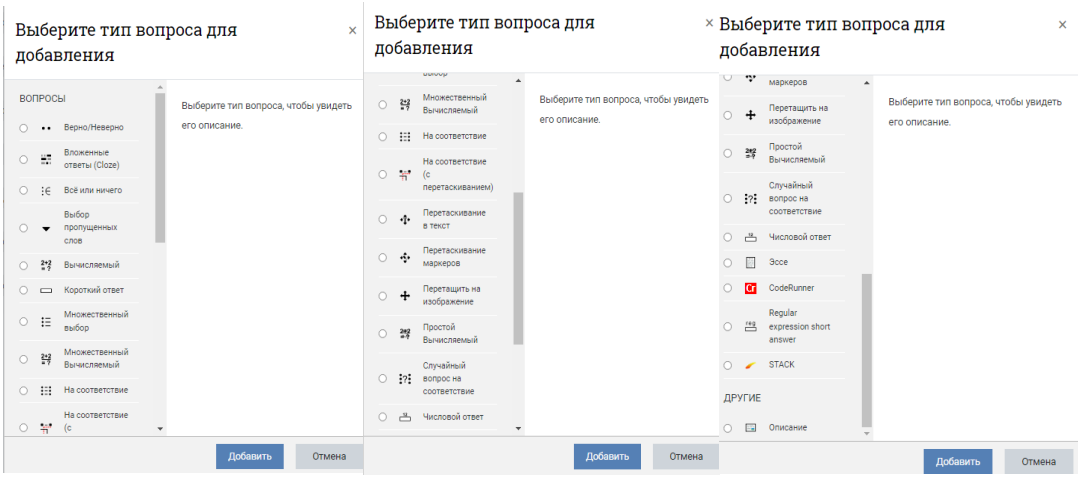

Рисунок 1 – Интерфейс выбора варианта вопроса Moodle

ClassMarker – программное обеспечение, позволяющее создавать и редактировать собственные тесты [4].

Данное решение имеет понятный и современный интерфейс, для использования программного обеспечения не требуется постоянное подключение к интернету, однако такие функции, как поделиться в социальных сетях и по почте будут недоступны. Есть возможность редактирования созданных тестов и вопросов в них. Отсутствует игровой элемент и функционал для его реализации. Стоимость программного продукта составляет от 2500 руб./мес.

На рисунке 2 представлен интерфейс редактора вопросов тестов и выбор правильного варианта ответа.

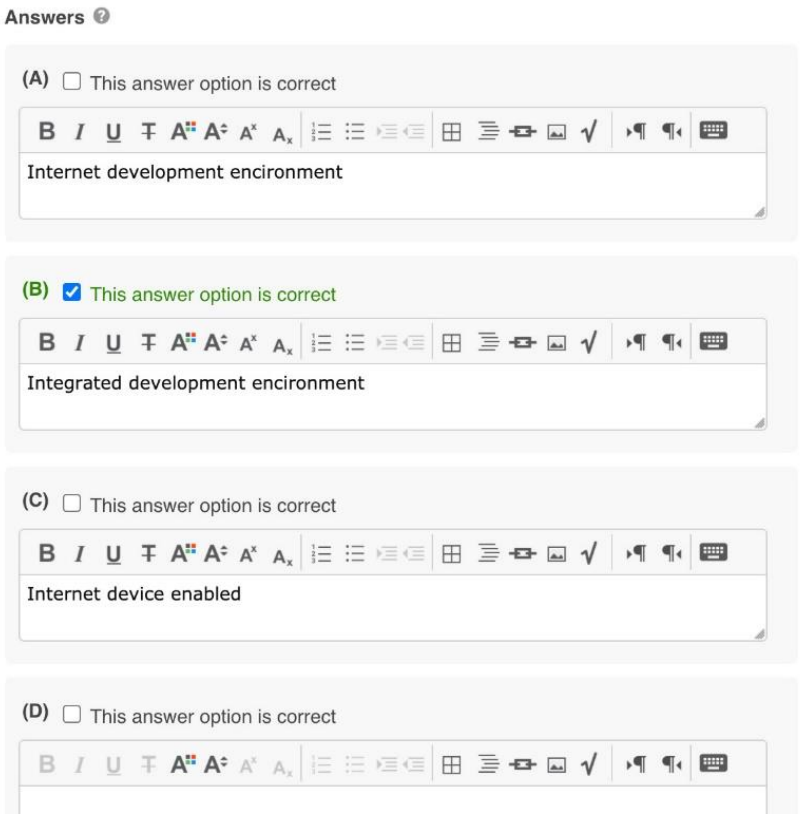

Рисунок 2 – Интерфейс редактора ClassMaker

Конструктор Тестов Ру – представляет собой веб-сайт с функционалом для создания и редактирования тестов в режиме онлайн [5].

Для использования данного решения требуется постоянное подключение к интернету. После регистрации пользователь получает возможность создания тестов, созданные тесты размещаются на сайте сервиса, доступ к ним открыт всем посетителям веб-сайта. Есть возможность редактирования тестов и вопросов. Сервис является полностью бесплатным, все тесты доступны для прохождения только на веб-сайте и в режиме онлайн. Отсутствует игровой элемент.

На рисунке 3 представлен интерфейс редактора тестов в данном решении.

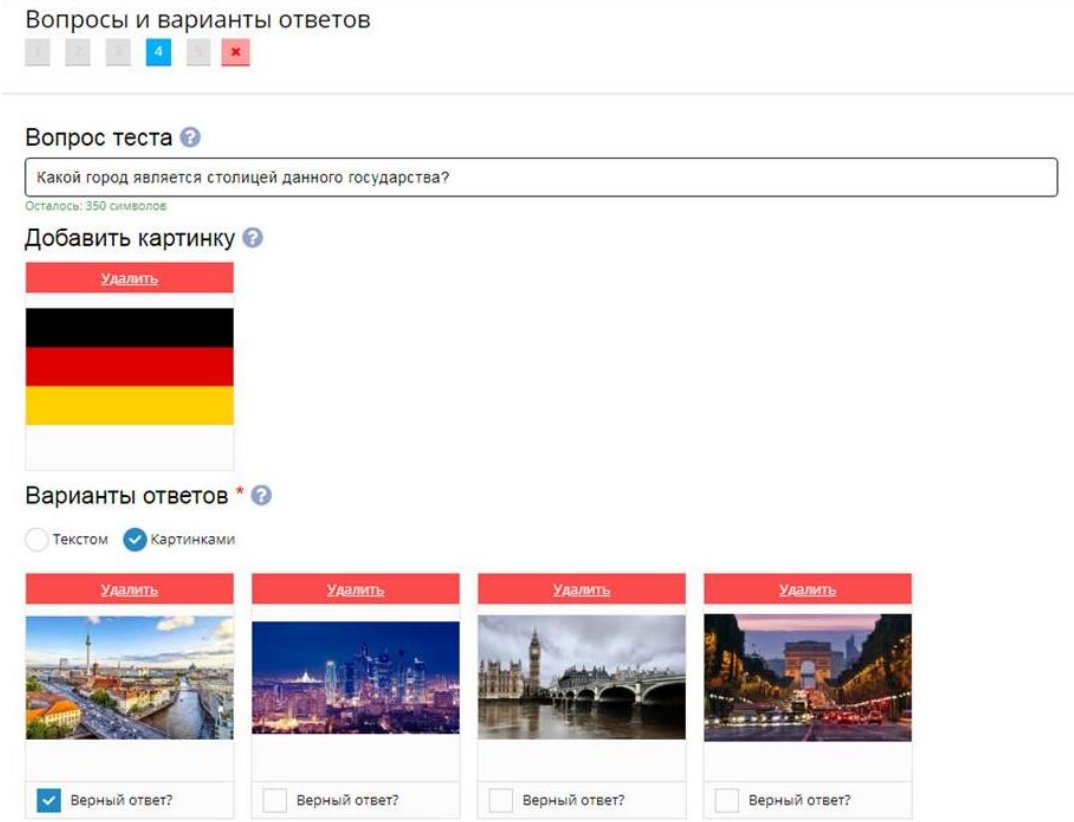

Рисунок 3 – Интерфейс редактора тестов Конструктор Тестов Ру

Testograf – конструктор онлайн тестов и опросов [6].

Для использования данного решения требуется регистрация на сайте и выбор лицензии, также требуется постоянное подключение к интернету. Присутствует гибкая настройка тестов: добавить доступ к тесту по паролю, убрать возможность изменения варианта ответа в вопросах, добавить ограничение по времени для прохождения теста. Есть возможность редактирования тестов и вопросов. Отсутствует игровой элемент. Минимальная цена – 6990 рублей за один тест-опрос.

На рисунке 4 представлен интерфейс редактора данного решения.

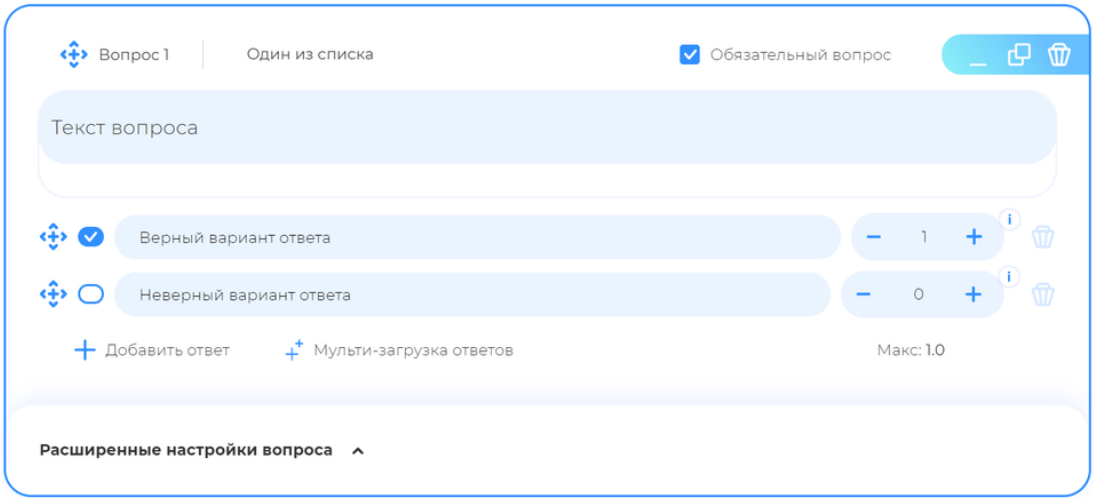

Рисунок 4 – Интерфейс редактора вопросов Testograf

Айрен – бесплатное программное обеспечение, позволяющее создавать тесты [7].

Данное решение обладает максимально простым интерфейсом, не требует подключения к интернету. На отдельной панели выводится список вопросов с возможностью переключения между ними. Присутствует возможность редактирования созданных тестов. Отсутствует игровой элемент. Данное программное обеспечение является полностью бесплатным, не требует платных лицензий.

На рисунке 5 представлен интерфейс редактора тестов данного программного обеспечения.

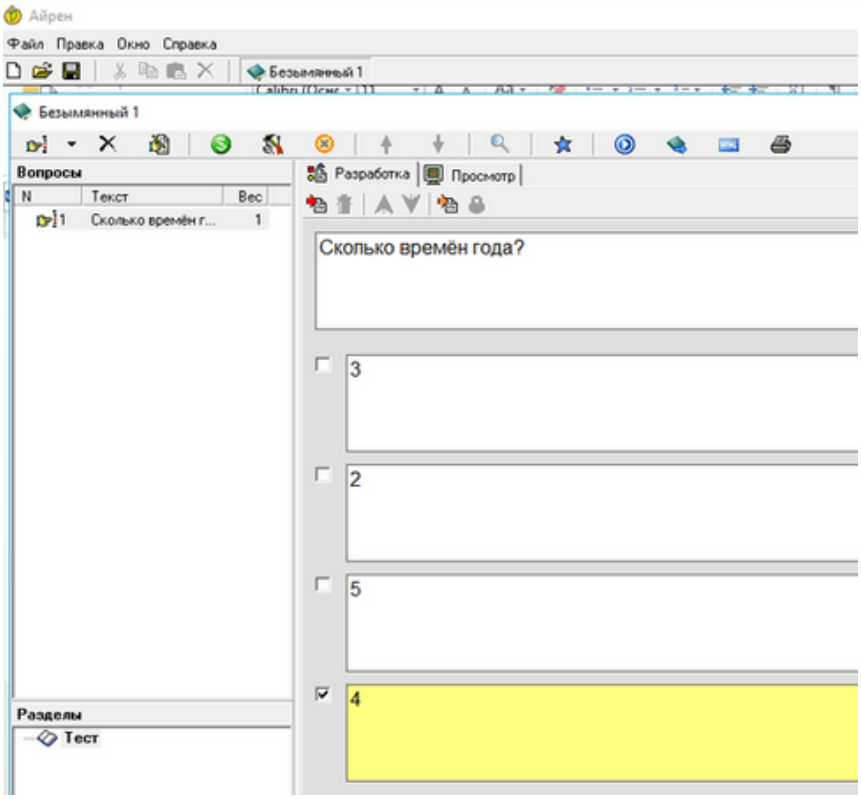

Рисунок 5 – Интерфейс редактора тестов Айрен

TestMaker – программное обеспечение для создания тестов [8].

Данное решение не требует подключения к интернету. Можно задать ограничение по времени и количество баллов за правильный ответ для каждого отдельного вопроса. Присутствует возможность редактирования тестов и вопросов к ним. Отсутствует игровой элемент и функционал для его реализации. Лицензионная версия данного программного обеспечения стоит 500 рублей.

На рисунке 6 представлен пример интерфейса редактора данного решения.

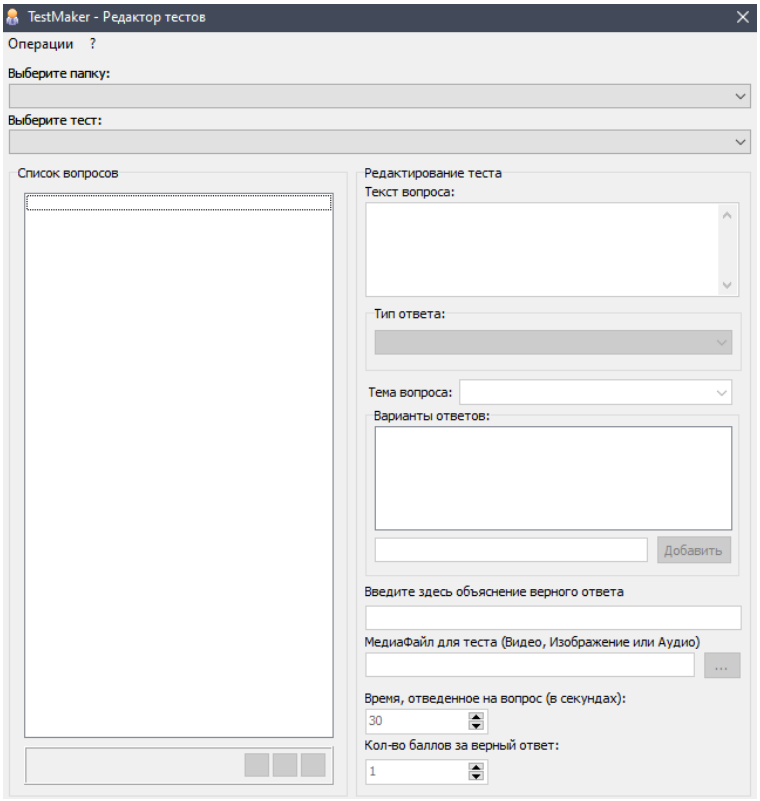

Рисунок 6 – Интерфейс редактора тестов TestMaker

Результаты анализа существующих аналогичных продуктов представлены в таблице 2.

Таблица 2 – Результаты анализа существующих аналогичных продуктов

Название	**Необходимость подключения к Интернету**	**Возможность редактирования тестов**	**Наличие игрового элемента**	**Стоимость**
Moodle	+	+	-	От 11000 руб./год
ClassMaker	-	+	-	От 2500 руб./месяц
Конструктор Тестов Ру	+	+	-	Бесплатно
Testograf	+	+	-	От 6990 руб.
Айрен	-	+	-	Бесплатно
TestMaker	-	+	-	500 руб.

Все существующие аналоги имеют возможность редактирования тестов и входящих в них вопросов, для использования некоторых из них требуется подключение к интернету. Также ни один из аналогов не обладает игровым элементом, который является одним из требований ЦБ РФ к разрабатываемой системе.

1.2 Анализ основных технологических решений

В настоящее время существуют различные среды для разработки компьютерных игр и программного обеспечения [9]. Они предоставляют широкий набор инструментов для работы с ними и облегчают процесс разработки приложений. Основную роль в разработке программного комплекса играет выбор игрового движка. Выделим несколько решений, отметив их достоинства и недостатки.

Unity – среда разработки компьютерных игр и программного обеспечения [10]. Для написания скриптов на данном игровом движке используется язык программирования C#.

Достоинства:

– поддержка 25 платформ, таких как: Windows, MacOS, iOS, Android, AR/VR и другие;

– удобный интерактивный интерфейс;

– лицензионная политика, платная подписка и проценты с продаж не требуются в случае если продукт распространяется бесплатно и без встроенной рекламы;

– более 750 часов профессионально записанных видео уроков и руководств [11].

Недостатки:

– имеется платный контент;

– сложность в оптимизации проектов с большим количеством 3D объектов.

Unreal Engine – игровой движок для создания 3D проектов, разработанный компанией Epic Games [12]. Для написания скриптов используется язык программирования C++.

Достоинства:

– оптимизация и возможности реализации сверхкачественной 3D графики;

– широкий выбор бесплатных шаблонов.

Недостатки:

– для распространения решений на данной платформе необходимо платить 5% с продаж и 1500$ в год;

– низкая оптимизация для устройств с низкими техническими характеристиками.

CryEngine – платформа для разработки игр под персональные компьютеры и консоли [13]. В данном движке интегрирована возможность выбора языка программирования для написания скриптов – C++ или C#.

Достоинства:

– встроенные шаблоны для работы с AI;

– бесплатные шаблоны 3D объектов различных поверхностей в высоком качестве.

Недостатки:

– нет поддержки операционных систем Linux и MacOS;

– стоимость лицензии составляет 400$ в год, также необходимо платить 5% с продаж, если готовое решение распространяется на платной основе.

1.3 Вывод

В данной работе для создания тест-игры наиболее целесообразно использовать платформу Unity и язык программирования C#. Разработка и

редактирование скриптов будет происходить в Visual Studio 2019 с использованием различных бесплатных расширений и плагинов для работы с Unity.

В разрабатываемой системе будут использоваться 2D объекты, созданные в программном обеспечении Adobe и представленные бесплатными инструментами Unity.

2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

К общим требованиям можно отнести минимальные системные требования для персональных компьютеров, использующих продукцию, созданную на движке Unity [14]:

– операционная система (ОС): Windows 7 или выше / Linux Ubuntu 20.04, Ubuntu 18.04, и CentOS;

– процессор не ниже: Intel Core i3 540/AMD A6-3620;

– оперативная память: не менее 4 GB;

– видеокарта: видеокарта с поддержкой DirectX 11.0 с 1 GB RAM (NVidia GeForce 460/ AMD Radeon 6850) или лучше;

– DirectX: версии 10 и выше;

– место на диске: 2GB;

– звуковая карта: звуковая карта, совместимая с DirectX.

Разрабатываемая система будет эксплуатироваться на персональных компьютерах с Windows 10, разрешением экрана 1920x1080, минимум 10GB свободного пространства на жестком или твердотельном накопителе, с комплектующими, превосходящими минимальные системные требования Unity.

2.1 Функциональные требования

Система должна соответствовать следующим функциональным требованиям:

– наличие двух видов пользователей: простой пользователь и редактор;

– предоставление простому пользователю возможности выбирать уровень для прохождения;

– количество уровней для прохождения не менее 4, не более 16;

– наличие двух игровых персонажей, которые представляют собой «Пионера Вольку» и «Старика Цифрабыча»;

– на каждом уровне перед тестом выводить сцену с диалогом двух персонажей в виде текста с количеством реплик не более 64, длиной реплики не более 512 символов;

– предоставление простому пользователю возможности пропустить диалог;

– наличие тестов на каждом уровне, вопросов в тесте не менее 4, не более 32;

– предоставление простому пользователю возможности прохождения теста путем ответа на вопросы с выбором одного из 4 предложенных вариантов;

– распознавание выбранных пользователем вариантов ответа и, в случае правильного ответа на все вопросы, засчитывать тест, в случае наличия ошибок предлагать пройти уровень заново;

– предоставление простому пользователю возможности повторного прохождения уровня;

– предоставление простому пользователю возможности перехода в режим редактора при корректном вводе пароля;

– предоставление возможности редактору изменять пароль;

– предоставление возможности редактору добавлять, удалять и переименовывать уровни;

– предоставление возможности редактору добавлять, удалять и изменять диалоги на разных уровнях;

– предоставление возможности редактору добавлять, удалять и изменять вопросы в тестах на разных уровнях.

2.2 Нефункциональные требования

Система должна соответствовать следующим нефункциональным требованиям:

приложение должно быть разработано на платформе, предоставляющей возможность бесплатного распространения готовой системы.

3 ПРОЕКТИРОВАНИЕ

3.1 Архитектура предлагаемого решения

На рисунках 7 – 10 приведена архитектура разрабатываемой системы. Для ее отображения было использовано программное обеспечение RAMUS Educational в соответствии с инструкциями руководства по эксплуатации [15].

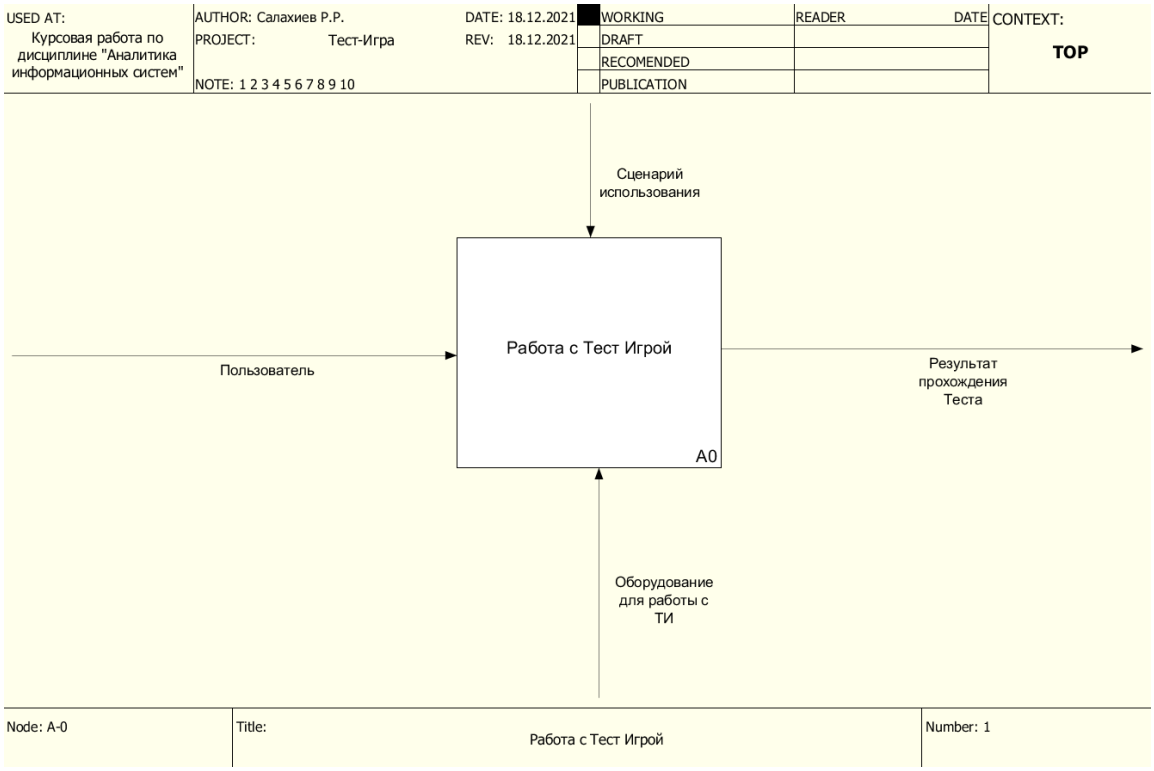

Рисунок 7 – Контекстная диаграмма

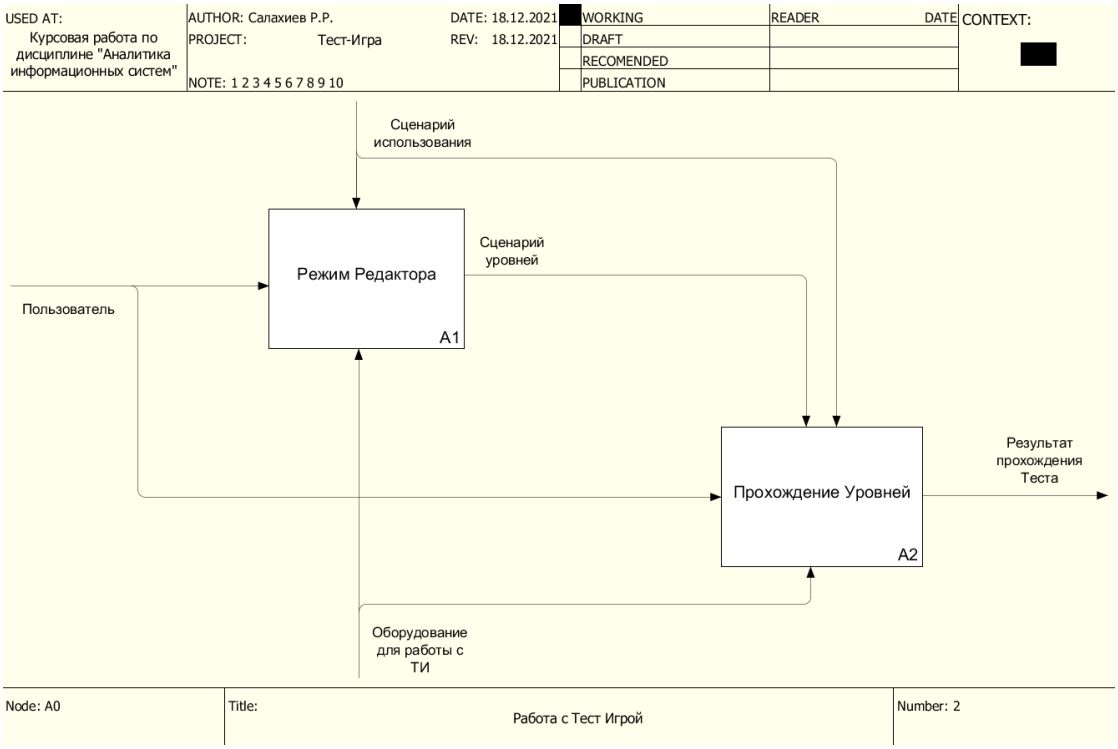

Рисунок 8 – Работа с Тест Игрой

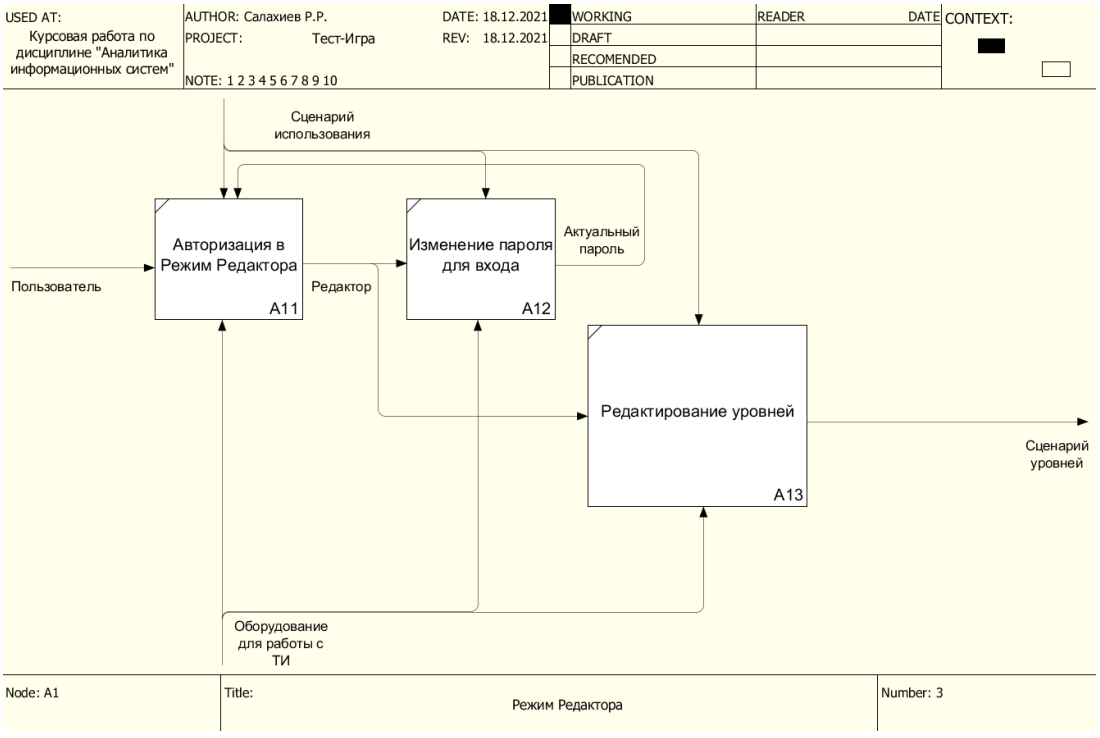

Рисунок 9 – Режим Редактора

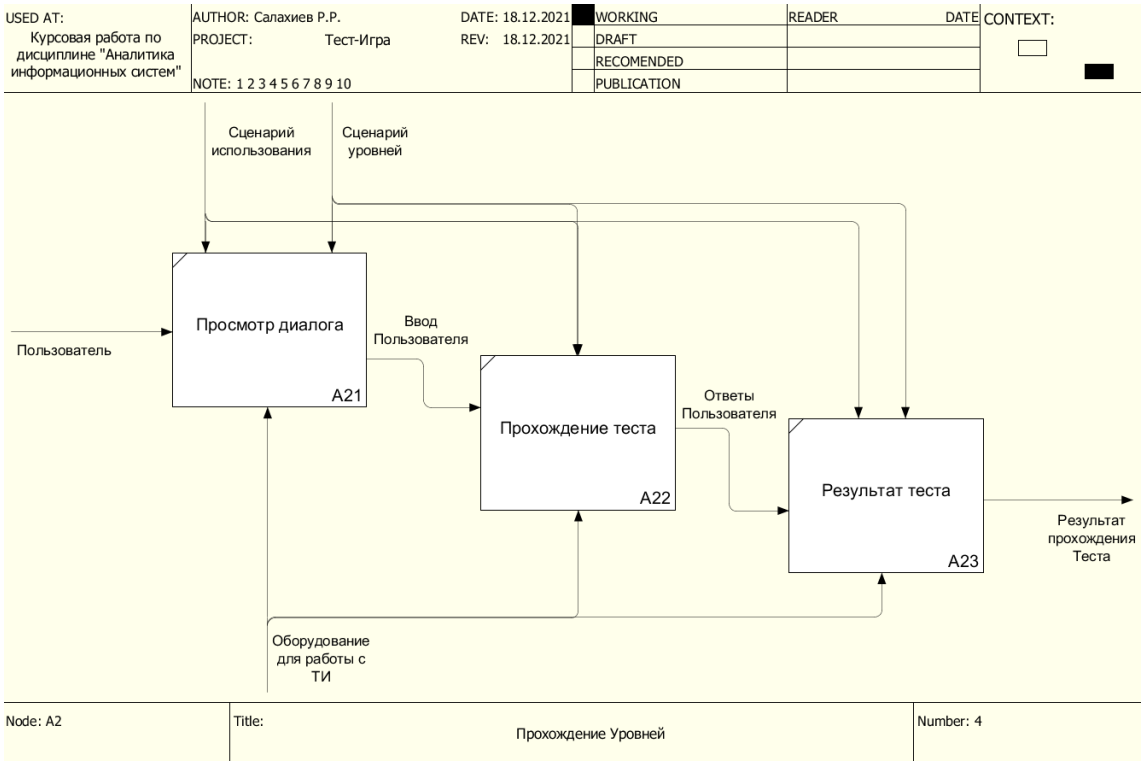

Рисунок 10 – Прохождение Уровней

На рисунке 11 представлены два вида пользователей: Простой Пользователь и Редактор. Простой Пользователь может выбрать уровень для прохождения, что включает в себя просмотр диалогов и прохождение тестов, а также перейти в Режим Редактора. Редактор может выбрать уровень для изменения, что включает в себя редактор диалогов и редактор тестов, а также может изменить пароль для перехода в Режим Редактора и перейти в Главное Меню, изменяя режим на Простого Пользователя.

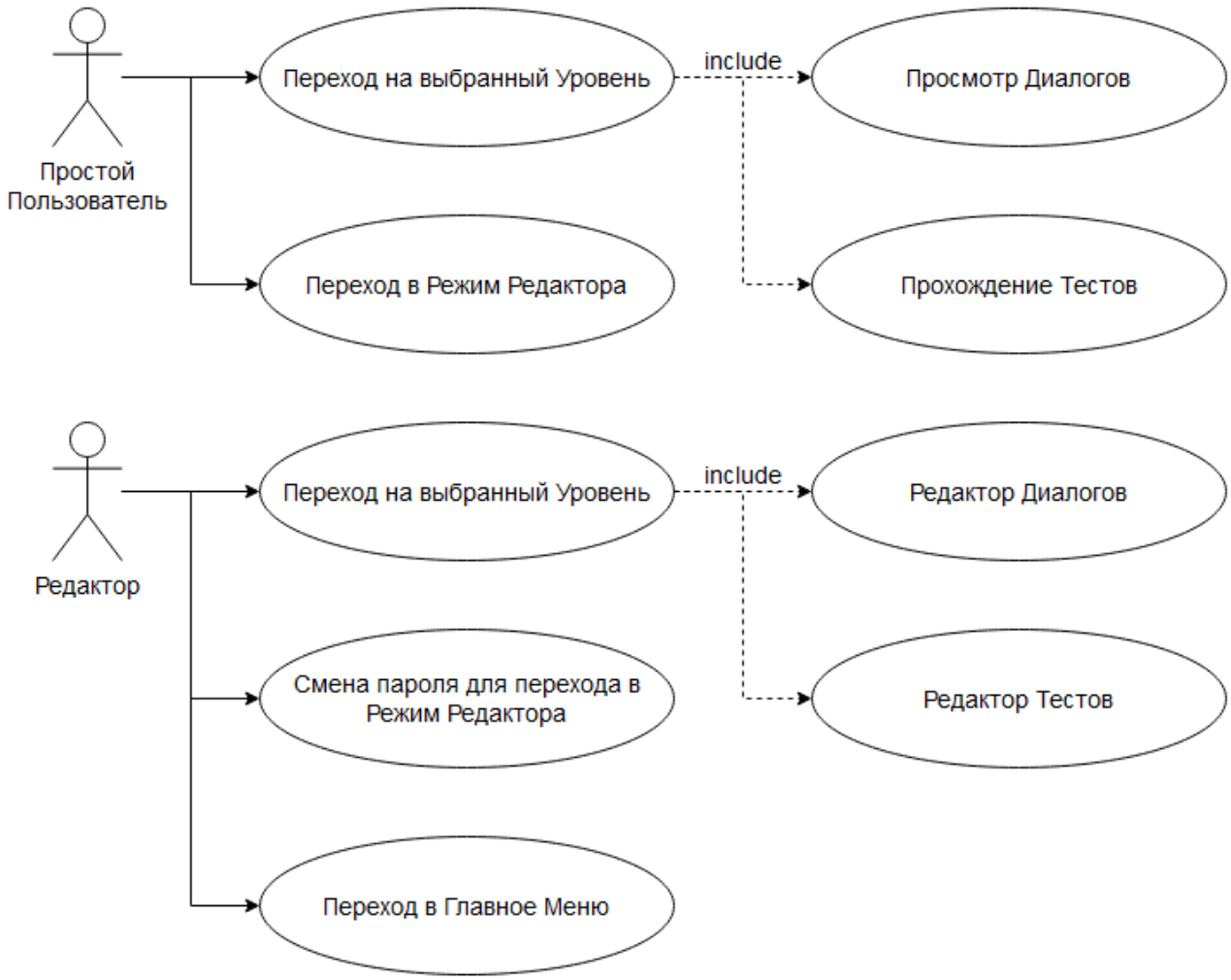

Рисунок 11 – Простой Пользователь и Редактор

3.2 Алгоритм решения задачи

Существуют различные варианты устройства алгоритмов [16]. В разрабатываемой системе используется циклический алгоритм прохождения теста. Пользователь выбирает нужный уровень, просматривает или пропускает диалоги персонажей, отвечает на представленные вопросы данного уровня.

В случае правильного ответа на все поставленные вопросы уровень считается пройдённым успешно, отображается меню с поздравлением и возможностью перехода в главное меню. При наличии хотя бы одной ошибки

уровень считается не пройденным, отображается меню с возможностью пройти уровень заново или перейти в главное меню.

Количество попыток для прохождения уровня не ограничено, после успешного прохождения теста и перехода в главное меню пользователь может пройти уровень заново.

На рисунке 11 приведен алгоритм прохождения теста.

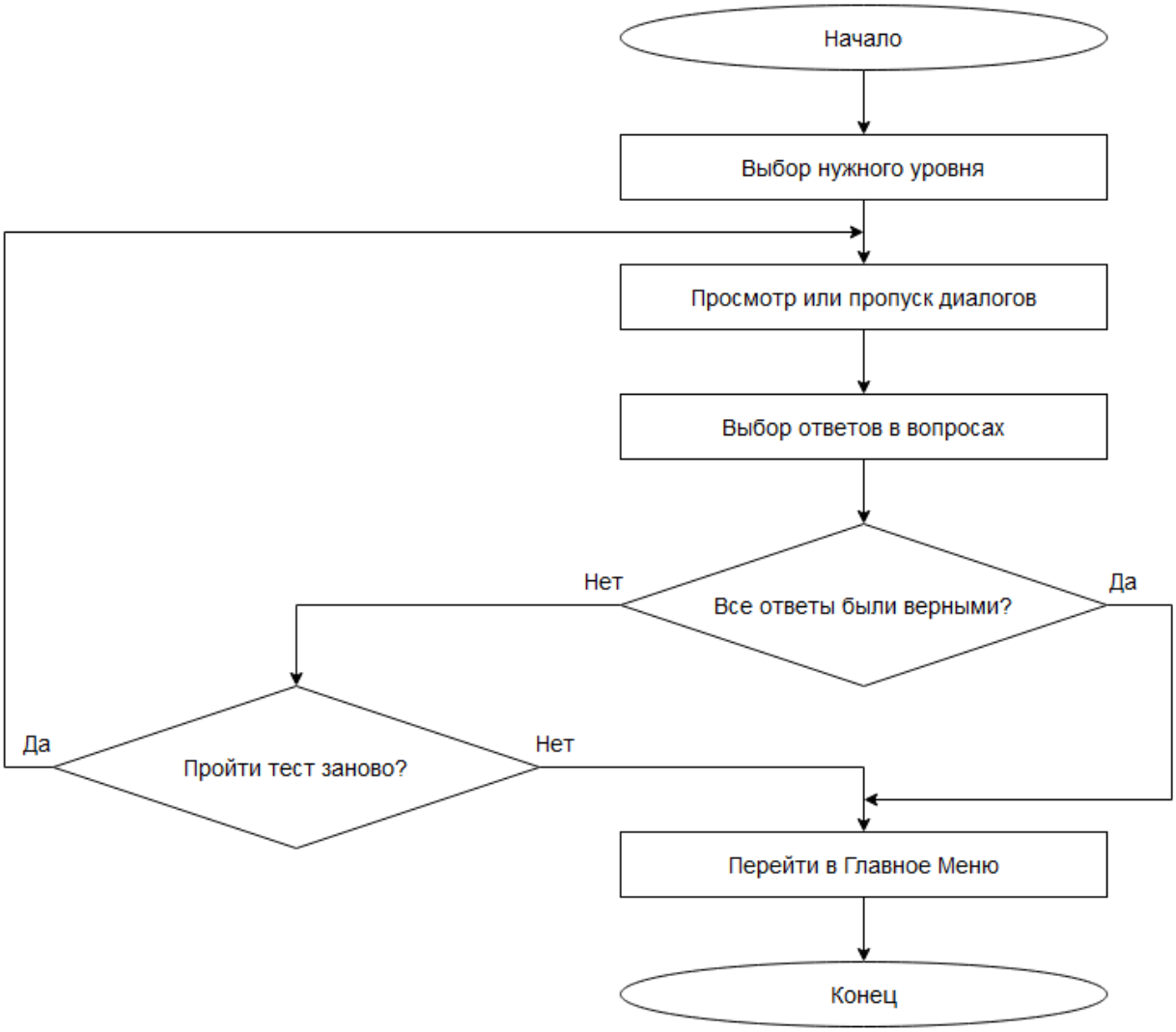

Рисунок 12 – Алгоритм прохождения теста

4 РЕАЛИЗАЦИЯ

4.1 Структура папок

Приложение помимо папок, хранящих обязательные исполнительные файлы для unity-приложений, имеет главную папку GameTest и 17 вложенных папок, представлены на рисунке 12. В папке Main хранятся данные о доступности уровней, пароле для перехода в режим редактора, данные о количестве диалогов и вопросов на каждом уровне. В папках Level_0 – Level_15 хранятся данные с текстами диалогов, вопросов, информацией о правильных ответах на вопросы, информацией соответствия диалога персонажу.

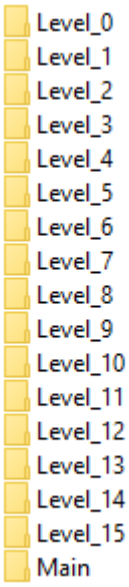

Рисунок 13 – Вложенные папки

4.2 Скрипты

Для реализации сцен были разработаны скрипты на языке C# [17]. Каждый скрипт содержит классы и методы этих классов. Классы содержат описание объектов в Unity, которые используются во всех методах. Назначение этих

объектов приведено в документации Unity [18]. Всего написано 7 скриптов. В таблице 3 представлены список и назначение каждого скрипта.

Таблица 3 – Использованные скриптов

Название скрипта	**Назначение**
Storage	Хранение данных, передаваемых между сценами, загрузка и сохранение информации, требуемой для их корректного функционирования
MainMenu	Скрипт для сцены Главного Меню, предназначен для перехода на выбранный уровень, перехода в режим редактора, выхода из приложения
GameDialogue	Скрипт для сцены Диалога между персонажами выбранного уровня, предназначен для перемещения между репликами и перехода на сцену Теста
GameTest	Скрипт для сцены Теста выбранного уровня, предназначен для ответа на вопросы, вывода результата и перехода на сцену Главного Меню
EditorMenu	Скрипт для сцены Режима Редактора, предназначен для переименования, активации и деактивации уровня, смены пароля для перехода в режим редактора, перехода на сцену Редактора Диалогов или Тестов выбранного уровня, сохранения и загрузки информации о названиях уровней и их доступности
EditDialogue	Скрипт для сцены Редактора Диалогов выбранного уровня, предназначен для добавления, удаления и изменения реплик диалогов, смены персонажа, сохранения и загрузки информации о диалогах выбранного уровня
EditTest	Скрипт для сцены Редактора Тестов выбранного уровня, предназначен для добавления, удаления и изменения вопросов теста, правильных ответов, сохранения и загрузки информации о вопросах теста выбранного уровня

Скрипт Storage – был написал для хранения и передачи данных между сценами в Unity, реализация представлена в листинге 1 приложения А. Для того, чтобы иметь возможность сохранить объекты системы, перенести приложение на другой персональный компьютер, была использована сериализация

объектов [19]. В корневой папке приложения имеются файлы, которые можно использовать для загрузки и сохранения нужной информации.

В скрипте Storage был написан одноименный класс Storage, данный класс хранит методы, позволяющие сохранять и загружать файлы формата data требуемые для корректного отображения информации на сценах. В таблице 4 представлен список методов класса Storage.

Таблица 4 – Методы класса Storage

Название метода	**Назначение**
LoadMenuData()	Загрузка из файла Menu.data данных о доступности уровней, пароле для перехода в режим редактора, о количестве диалогов и вопросов на каждом уровне
SaveMenuData()	Сохранение в файл Menu.data данных о доступности уровней, пароле для перехода в режим редактора, о количестве диалогов и вопросов на каждом уровне
LoadLevelData(int level)	Загрузка из файла Level_(значение переменной level).data данных с текстами диалогов, вопросов, информацией о правильных ответах на вопросы, информацией соответствия диалога персонажу по заданному уровню
SaveLevelData(int level)	Сохранение в файла Level_(значение переменной level).data данных с текстами диалогов, вопросов, информацией о правильных ответах на вопросы, информацией соответствия диалога персонажу по заданному уровню

Скрипт MainMenu – был написан для работы со сценой Главного Меню, реализация представлена в листинге 2 приложения Б. В данном скрипте был написан одноименный класс MainMenu, данный класс хранит методы, позволяющие загрузить информацию при переходе на сцену Главного Меню, выйти из приложения, выбрать уровень, перейти на выбранный уровень, перейти в Режим Редактора. В таблице 5 представлены методы класса MainMenu.

Таблица 5 – Методы класса MainMenu

Название метода	Назначение
void Start()	Вызов метода LoadMenuData() из скрипта Storage, присвоение названий всем соответствующим уровням
public void Exit()	Закрыть приложение
public void SelectLevel(int i)	Выбрать уровень для прохождения соответствующий значению переменной i
public void StartLevel()	Вызов метода LoadLevelData(int level) из скрипта Storage со значением selected_level, загрузка информации по выбранному уровню и переход на него
public void EditorMenu()	При корректном вводе пароля перейти в Режим Редактора

Скрипт GameDialogue – был написан для работы со сценой Диалога, реализация представлена в листинге 3 приложения В. В данном скрипте был написан одноименный класс GameDialogue, данный класс хранит методы, позволяющие вывести первый текст диалога выбранного уровня при переходе на сцену, вывести требуемый текст диалога, перейти к следующему тексту, перейти к предыдущему тексту, пропустить весь диалог, перейти в главное меню. В таблице 6 представлены методы класса GameDialogue.

Таблица 6 – Методы класса GameDialogue

Название метода	Назначение
private void LoadDialogue(int i)	Вывод текста диалога с индексом i, вывод номера диалога и количества диалогов на уровне
void Start()	Вызов метода LoadDialogue(int i) со значением 0, установка диалога в начальное положение
public void Next()	Переключиться на следующий текст, если текущий текст был последним вызов метода SkipAll()
public void Previous()	Переключиться на предыдущий текст
public void SkipAll()	Перейти на сцену Теста
public void MainMenu()	Перейти на сцену Главного Меню

Скрипт GameTest – был написан для работы со сценой Теста, реализация представлена в листинге 4 приложения Г. В данном скрипте был написан одноименный класс GameTest, данный класс хранит методы, позволяющие вывести первый вопрос из теста выбранного уровня при переходе на сцену, вывести требуемый текст вопроса, перейти к следующему вопросу, перейти к предыдущему вопросу, выбрать вариант ответа, завершить тест, перейти в главное меню. В таблице 7 представлены методы класса GameTest.

Таблица 7 – Методы класса GameTest

Название метода	**Назначение**
private void LoadTest(int i)	Вывод вопроса с индексом i, вывод номера вопроса и количества вопросов на уровне
void Start()	Вызов метода LoadTest(int i) со значением 0, установка теста в начальное положение
public void Next()	Переключиться на следующий вопрос, если текущий вопрос был последним вызов метода EndTest()
public void Previous()	Переключиться на предыдущий вопрос
public void EndTest()	Завершение теста, вывод окна результата где можно выйти в главное меню или пройти уровень заново при наличии ошибок в тесте
public void MainMenu()	Перейти на сцену Главного Меню
public void Click(int i)	Выбрать вариант с индексом i
public void Retry()	Пройти уровень заново

Скрипт EditorMenu – был написан для работы со сценой Режима Редактора, реализация представлена в листинге 5 приложения Д. В данном скрипте был написан одноименный класс EditorMenu, данный класс хранит методы, позволяющие активировать или деактивировать уровень, переименовать выбранный уровень, перейти в режим редактора диалогов или тестов выбранного уровня, сменить пароль для перехода в режим редактора, сохранить внесенные

изменения, отменить внесенные изменения, выйти в главное меню. В таблице 8 представлены методы класса EditorMenu.

Таблица 8 – Методы класса EditorMenu

Название метода	**Назначение**
void Start()	Присвоение соответствующих названий уровням, установка флагов активности
public void MenuEditLevel(int i)	Активация меню для перехода в Редактор Теста или Редактор Диалогов уровня с индексом i
public void EditDialogue()	Загрузка информации по выбранному уровню и переход в Редактор Диалогов данного уровня
public void EditTest()	Загрузка информации по выбранному уровню и переход в Редактор Тестов данного уровня
public void MenuChangePassword()	Активация меню для смены пароля
public void ChangePassword()	Смена пароля для перехода в Режим Редактора
public void MenuChangeName(int i)	Активация меню для переименования уровня с индексом i
public void ChangeName()	Смена названия выбранного уровня
public void SetActive(int i)	Переключение доступности уровня с индексом i
public void LoadData()	Вызов метода LoadMenuData() из скрипта Storage, перезагрузка сцены EditorMenu
public void SaveData()	Вызов метода SaveMenuData() из скрипта Storage
public void MainMenu()	Перейти на сцену Главного Меню

Скрипт EditDialogue – был написан для работы со сценой Редактора Диалогов, реализация представлена в листинге 6 приложения Е. В данном скрипте был написан одноименный класс EditDialogue, данный класс хранит методы, позволяющие вывести первый текст диалога выбранного уровня при переходе на сцену, вывести требуемый текст диалога, перейти к следующему тексту, перейти к предыдущему тексту, изменить текст диалога, добавить или

удалить диалог, изменить отображаемого персонажа, сохранить или отменить изменения, перейти в режим редактора. В таблице 9 представлены методы класса EditDialogue.

Таблица 9 – Методы класса EditDialogue

Название метода	**Назначение**
private void LoadDialogue(int i)	Вывод текста диалога с индексом i, вывод номера диалога и количества диалогов на уровне
private void SaveText()	Сохранить состояние диалога перед переходом
void Start()	Вызов метода LoadDialogue(int i) со значением 0, установка диалога в начальное положение
public void Next()	Переключиться на следующий текст
public void Previous()	Переключиться на предыдущий текст
public void AddDialogue()	Добавить диалог
public void RemoveDialogue()	Удаление диалога
public void ChangeCharacter()	Сменить отображаемого персонажа
public void SaveData()	Сохранить изменения
public void LoadData()	Отменить изменения
public void EditorMenu()	Переход в Режим Редактора

Скрипт EditTest – был написан для работы со сценой Редактора Тестов, реализация представлена в листинге 7 приложения Ж. В данном скрипте был написан одноименный класс EditTest, данный класс хранит методы, позволяющие вывести первый текст вопроса теста выбранного уровня при переходе на сцену, вывести требуемый текст теста, перейти к следующему

вопросу, перейти к предыдущему вопросу, изменить текст вопроса и вариантов ответа, добавить или удалить вопрос, изменить правильный ответ, сохранить или отменить изменения, перейти в режим редактора. В таблице 10 представлены методы класса EditTest.

Таблица 10 – Методы класса EditTest

Название метода	**Назначение**
private void LoadTest(int i)	Вывод текста вопроса и вариантов ответа с индексом i, вывод номера вопроса и количества вопросов на уровне
private void SaveText()	Сохранить состояние текстов вопроса перед переходом
void Start()	Вызов метода LoadTest (int i) со значением 0, установка теста в начальное положение
public void Next()	Переключиться на следующий вопрос
public void Previous()	Переключиться на предыдущий вопрос
public void AddQuestion() ()	Добавить вопрос в тест
public void RemoveDialogue()	Удаление вопрос из теста
public void RemoveQustion()	Сменить отображаемого персонажа
public void SaveData()	Сохранить изменения
public void LoadData()	Отменить изменения
public void EditorMenu()	Переход в Режим Редактора

На рисунке 14 представлена диаграмма переходов между сценами. Из главного меню пользователь может попасть на сцены Диалогов и Режима Редактора. Из сцены Диалога пользователь может перейти на сцену Теста или вернуться на сцену Главного Меню. Из сцены Теста пользователь может снова

повторить уровень, обновив сцену с тестом, либо перейти на сцену Главного Меню. Из сцены Режима Редактора пользователь может выйти на сцену Главного Меню или перейти на сцены редактора Диалогов и Тестов, из которых можно вернуться обратно на сцену Режима Редактора.

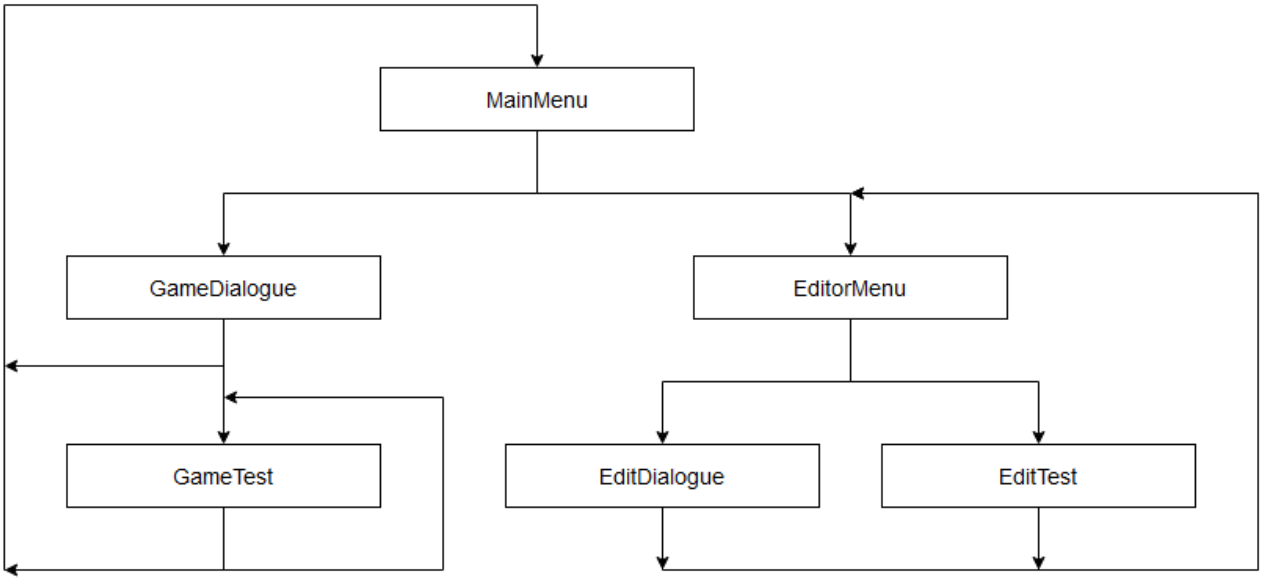

Рисунок 14 – Диаграмма переходов между сценами

4.3 Интерфейс

Проект был реализован на платформе Unity Editor версии 2020.3.26f1, с использованием встроенных инструментов [20]. Для создания изображений был использован редактор Adobe Photoshop 2021.

В разработанной системе были созданы следующие сцены:

1) Главное Меню;

2) Диалог;

3) Тест;

4) Режим Редактора;

5) Редактор Диалогов;

6) Редактор Тестов.

На рисунках 15 – 20 представлены интерфейсы каждой из этих сцен.

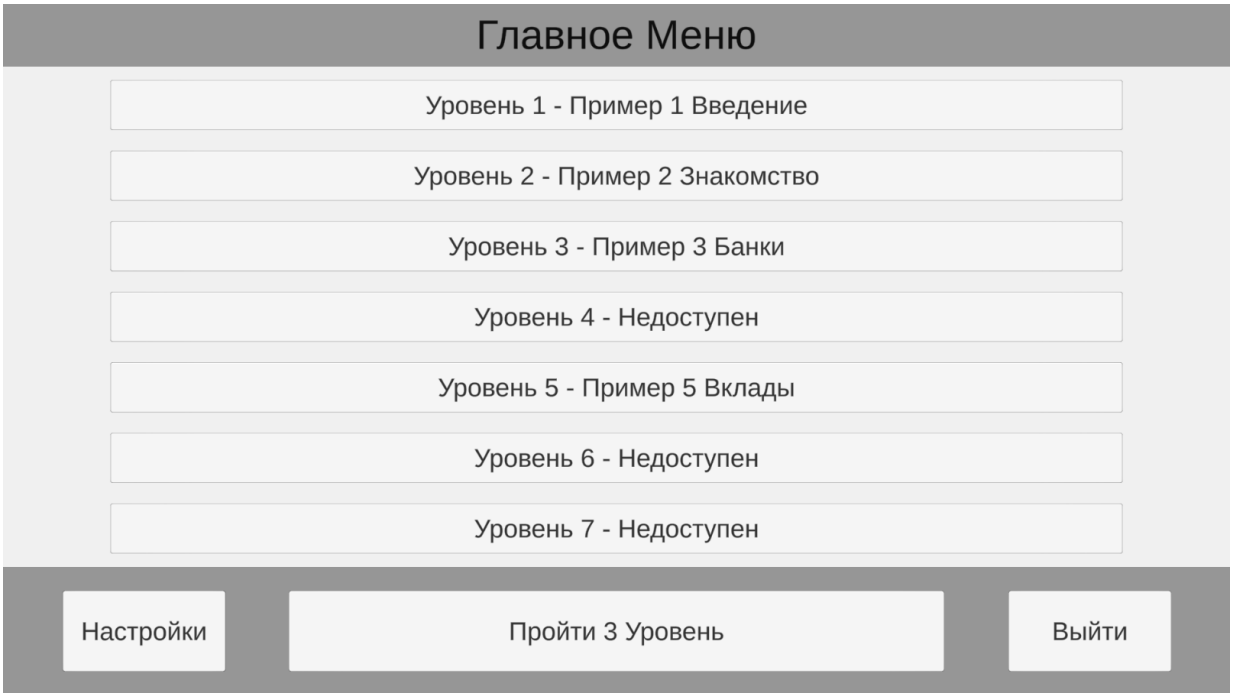

Рисунок 15 – Главное Меню

В главном меню пользователь может:

– выбрать уровень для прохождения;

– перейти на выбранный уровень;

– перейти в режим редактора;

– выйти из приложения.

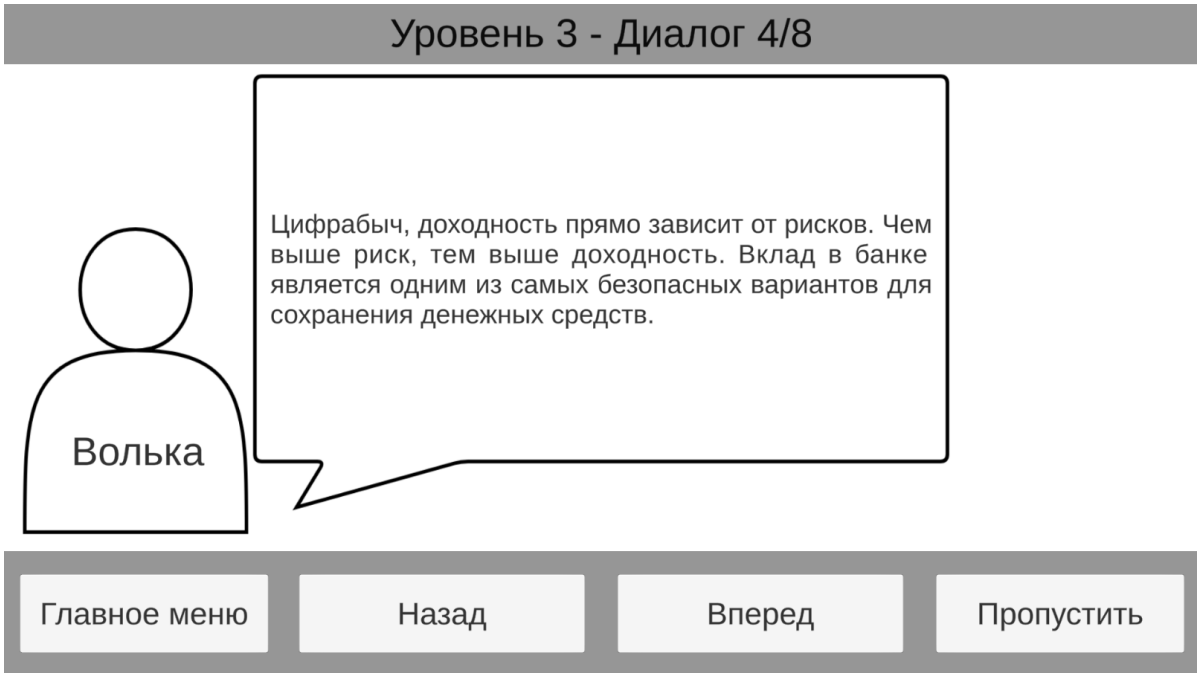

Рисунок 16 – Диалог

На сцене диалога пользователь имеет следующие возможности:

– выйти в главное меню;

– вернуться на предыдущий текст диалога;

– перейти к следующему тексту диалога;

– пропустить диалог и приступить к тесту.

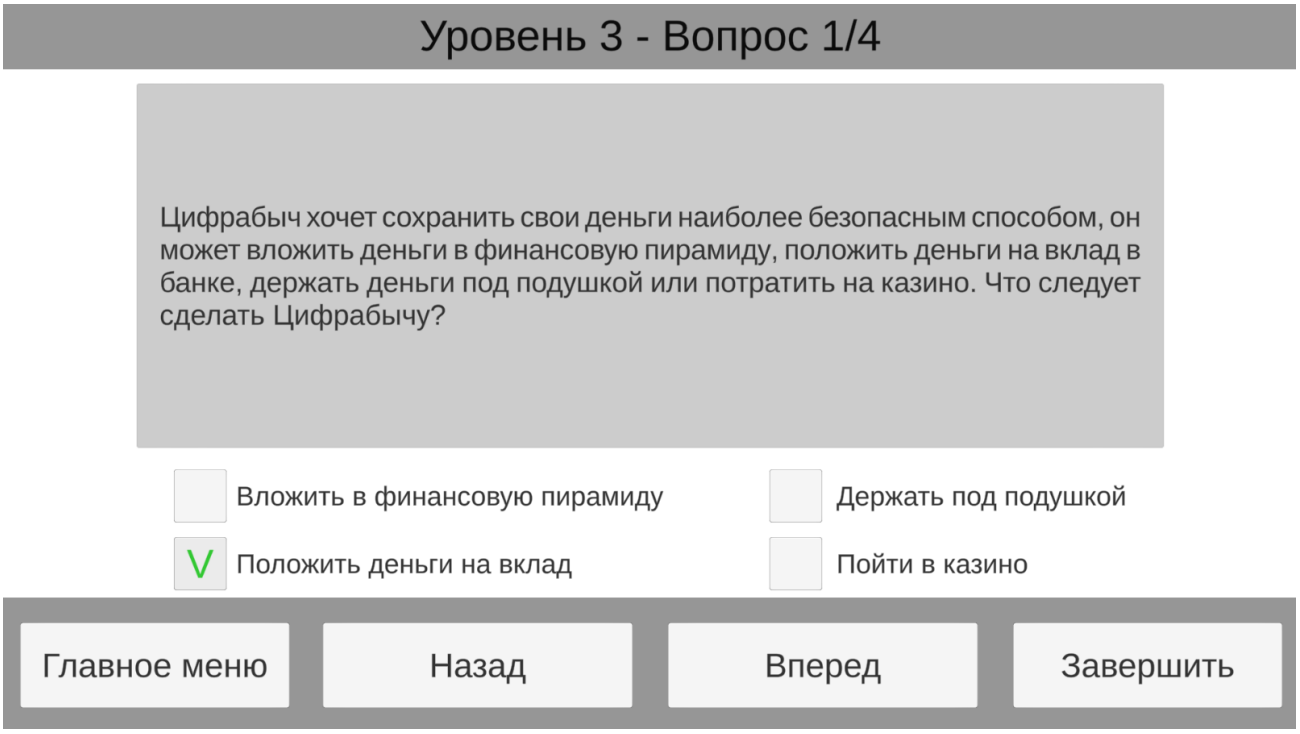

Рисунок 17 – Тест

На сцене теста пользователь может:

– выбрать вариант ответа;

– выйти в главное меню;

– вернуться к предыдущему вопросу;

– перейти к следующему вопросу;

– завершить прохождение теста.

Рисунок 18 – Режим Редактора

В меню редактора пользователь может:

– активировать или деактивировать уровень;

– переименовать уровень;

– перейти в режим редактора диалогов или тестов выбранного уровня;

– сменить пароль для перехода в режим редактора;

– сохранить изменения;

– отменить изменения;

– выйти в главное меню.

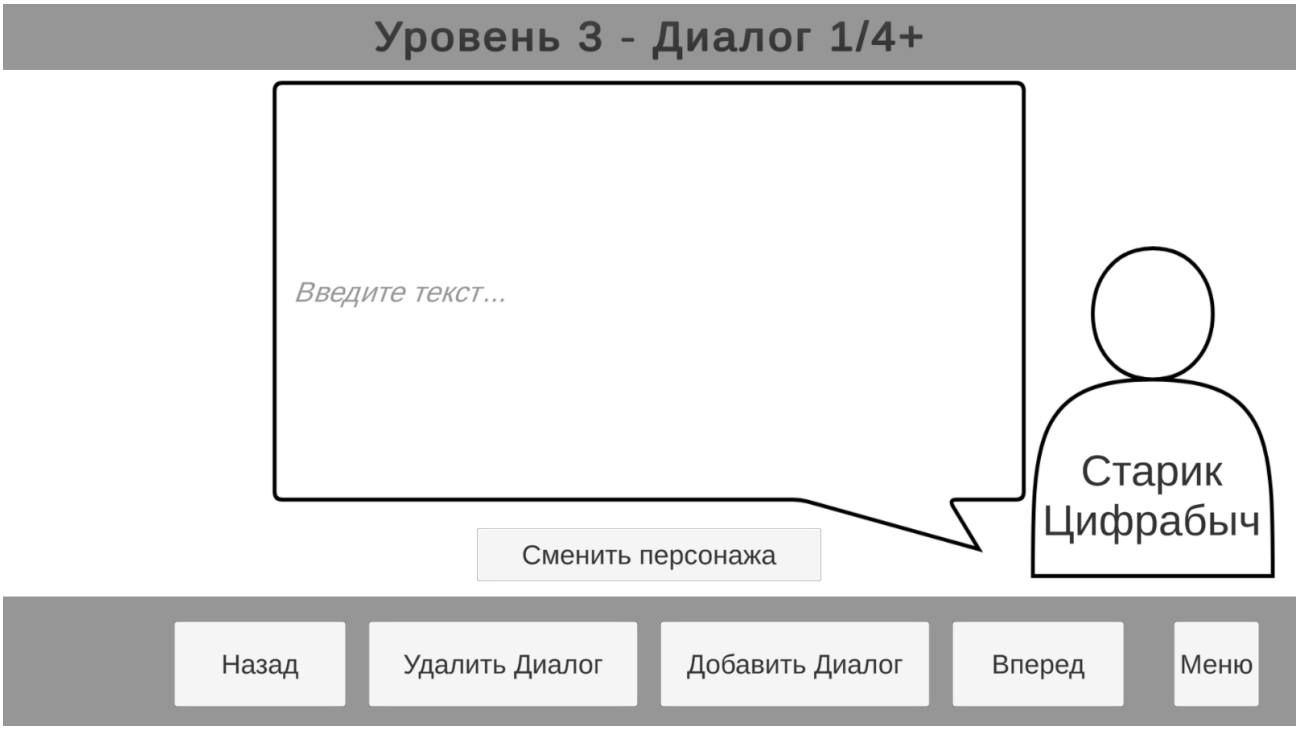

Рисунок 19 – Редактор Диалогов

На сцене редактора диалогов пользователь имеет следующие возможности:

– перейти к прошлому тексту диалога;

– перейти к следующему тексту диалога;

– удалить текущий диалог;

– добавить диалог;

– открыть меню, в котором можно сохранить или отменить изменения, перейти в меню редактора;

– ввести текст диалога;

– сменить отображаемого персонажа.

Рисунок 20 – Редактор Тестов

На сцене редактора теста пользователь может:

– перейти к прошлому вопросу;

– перейти к следующему вопросу;

– удалить текущий вопрос;

– добавить вопрос;

– открыть меню, в котором можно сохранить или отменить изменения, перейти в меню редактора;

– ввести текст в окно с текстом вопроса;

– ввести текст в варианты ответов;

– выбрать правильный ответ на вопрос.

5 ТЕСТИРОВАНИЕ

Тестирование разработанного приложения проводилось на персональном компьютере с характеристиками:

– процессор: Intel Core I7-7700HQ;

– видеокарта: NVidia GeForce 1050Ti;

– оперативная память: 8 ГБ;

– операционная система: Windows 10.

Для тестирования был выбран модульный подход: проверка корректности работы блоков системы с которыми может взаимодействовать пользователь [21]. На каждой созданной в Unity сцены были испытаны различные варианты взаимодействия с интерактивными объектами. Было проведено сравнение ожидаемого и полученного результатов, на основе которых были сделаны выводы о правильности работы приложения.

В таблице 11 представлены результаты корректности работы сцены Главного Меню, для остальных сцен были использованы аналогичные методы тестирования.

Таблица 11 – Тестирование сцены Главного Меню

Тест	**Действие**	**Ожидаемый результат**	**Полученный результат**	**Итог**
Проверка на корректный переход на уровень	Использование одной из 16 кнопок уровней, затем кнопки «Пройти Уровень»	Переход на выбранный уровень	Переход на выбранный уровень	Пройден
Проверка на корректность выхода	Использование кнопки «Выйти»	Закрытие приложения	Закрытие приложения	Пройден
Проверка на корректную работу открытия меню настроек	Использование кнопки «Настройки»	Открытие меню настроек для перехода в режим редактора	Открытие меню настроек для перехода в режим редактора	Пройден

Продолжение таблицы 11

Проверка на корректную работу закрытия меню настроек	Использование кнопки «X» в меню настроек	Закрытие меню настроек	Закрытие меню настроек	Пройден
Проверка на корректную работу перехода в меню редактора	Использование поля для ввода пароля, затем использование кнопки «Войти»	Переход в режим редактора, при корректном вводе пароля	Переход в режим редактора, при корректном вводе пароля	Пройден

При использовании программы не были выявлены критические ошибки, делающие программу неработоспособной. Все элементы приложения работают в соответствии с требуемым функционалом.

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы был разработан полностью функциональный билд компьютерной игры-теста на платформе Unity.

При этом были решены следующие задачи:

– проведен обзор аналогичных решений и осуществлена постановка задачи;

– проведен обзор средств реализации;

– проведен анализ требований и спроектировано приложение;

– реализована демонстрационная версия приложения;

– проведено тестирование реализованной версии приложения.

В дальнейшем предполагается последующая разработка и улучшение приложения, добавление профессиональных изображений и рисунков, добавление различных типов вопросов, наполнение уровней текстами диалогов и вопросов совместно с сотрудниками ЦБ РФ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Преподавание финансовой грамотности в школах. – Текст : электронный // Центральный Банк РФ : [официальный сайт]. – 8 июля 2021 года. – URL: https://cbr.ru/press/event/?id=11018 (дата обращения: 20.02.2022).

2. Villagra, M. G. Types of Test Item Formats / Villagra, M. G. – New York : W.W. Norton and Company, 2018. – 85 с. : ил.

3. Moodle Platform. – Текст : электронный // Moodle : [сайт]. – URL: https://moodle.com/ (дата обращения 25.02.2022).

4. Ответы на вопросы о продукте. – Текст : электронный // Программное обеспечение ClassMaker : [сайт]. – URL: https://www.classmarker.com/online-testing/faq/ (дата обращения 25.02.2022).

5. Онлайн конструктор тестов // Онлайн Конструктор Тестов Ру : [сайт]. – URL: https://konstruktortestov.ru (дата обращения 25.02.2022).

6. О продукте. – Текст : электронный // Конструктор тестов Testograf : [сайт]. – URL: https://www.testograf.ru/ru/about (дата обращения 25.02.2022).

7. Главная страница проекта Айрэн. – Текст : электронный // Программное обеспечение Айрэн : [сайт]. – URL: https://irenproject.ru/ (дата обращения 25.02.2022).

8. О продукте Testmaker. – Текст : электронный // Программное обеспечение TestMaker : [сайт]. – URL: http://s-soft.org/testmaker.php (дата обращения 25.02.2022).

9. Conde, D. Software Product Management: Managing Software Development / Condone, D. – New York: Aspatore, 2015. – 312 с. : ил.

10. Unity Platform. – Текст : электронный // Unity : [сайт]. – URL: https://unity.com/products/unity-platform (дата обращения 10.03.2022).

11. Unity Learn. – Текст : электронный // Unity : [сайт]. – URL: https://learn.unity.com/ (дата обращения 10.03.2022).

12. Unreal Engine Main Page. – Текст : электронный // Unreal Engine : [официальный сайт] – URL: https://www.unrealengine.com (дата обращения 10.03.2022).

13. CryEngine Features. – Текст : электронный // CryEngine : [сайт] – URL: https://www.cryengine.com/features (дата обращения 10.03.2022).

14. Unity Manual: System Requirements. – Текст : электронный // Unity Docs : [сайт]. – URL: https://docs.unity3d.com/Manual/system-requirements.html (дата обращения 11.03.2022).

15. Java-based IDEF0 & DFD Modeler. – Текст : электронный // RAMUS : [сайт]. – URL: https://ramussoftware.com (дата обращения 15.03.2022).

16. Томас Х. Алгоритмы: построение и анализ / Томас Х. – Москва : «Вильямс», 2013. — 1328 с. : ил.

17. Hocking J. Unity in Action. Game development in C# with Unity / Hocking J. – New York : Manning Publications, 2019. – 352 с. : ил.

18. Unity Manual: Important Classes. – Текст : электронный // Unity Docs : [сайт]. – URL: https://docs.unity3d.com/Manual/-ScriptingImportantClasses.html (дата обращения 15.03.2022).

19. Unity Manual: Script serialization. – Текст : электронный // Unity Docs : [сайт]. – URL: https://docs.unity3d.com/2022.1/Documentation/-Manual/script-Serialization.html. Дата обращения 15.03.2022.

20. Ferrone H. Game Programming Developing with Unity in C# / Ferrone H. – Birmingham : Pakt Publishing, 2021. – 428 с. : ил.

21. Ошероув, Р. The Art Of Unit Testing Second Edition With Examples In C# / Ошероув, Р. – Москва : ДМК Пресс, 2016. – 363 с. : ил.

ПРИЛОЖЕНИЕ А
Исходный код скрипта Storage

Листинг 1 – Хранилище Данных

```
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public class Storage : MonoBehaviour
{
    public static List<string> user_answer;
    public static int selected_level;

    //Menu Data
    [System.Serializable]
    public class MenuData
    {
        public string password;
        public List<bool> lvl_active;
        public List<string> lvl_name;
        public List<int> lvl_dialogues_num;
        public List<int> lvl_questions_num;
    }
    public static MenuData menu_data;

    //Levels Data
    [System.Serializable]
    public class LevelData
    {
        public List<int> character;
        public List<string> dialogue;
        public List<string> question;
        public List<string> choice_0, choice_1, choice_2, choice_3;
        public List<int> correct_answer;
    }
    public static List<LevelData> level_data;
```

```
    //Menu Load and Save
    public static void LoadMenuData()
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "Main/Menu.data";
        FileStream stream = new FileStream(path, FileMode.Open);
        menu_data = formatter.Deserialize(stream) as MenuData;
        stream.Close();
    }
    public static void SaveMenuData()
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "Main/Menu.data";
        FileStream stream = new FileStream(path, FileMode.Create);
        formatter.Serialize(stream, menu_data);
        stream.Close();
    }
    //Levels Load and Save
    public static void LoadLevelData(int level)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/Level_" +
level.ToString() + "/Level_" + level.ToString() + ".data";
        FileStream stream = new FileStream(path, FileMode.Open);
        level_data[level] = formatter.Deserialize(stream) as LevelData;
        stream.Close();
    }
    public static void SaveLevelData(int level)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/Level_" +
level.ToString() + "/Level_" + level.ToString() + ".data";
        FileStream stream = new FileStream(path, FileMode.Create);
        formatter.Serialize(stream, level_data[level]);
        stream.Close();
    }
}
```

ПРИЛОЖЕНИЕ Б
Исходный код скрипта MainMenu

Листинг 2 – Скрипт сцены Главного Меню

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;

public class MainMenu : MonoBehaviour
{
    public List<TMP_Text> button;
    public TMP_Text start_button;
    public TMP_Text password;

    void Start()
    {
        Storage.LoadMenuData();
        Storage.selected_level = -1;
        start_button.SetText("Выберите уровень для прохождения");
        for (int i = 0; i < 16; i++)
        {
            int num = i + 1;
            if (Storage.menu_data.lvl_active[i])
            {
                button[i].SetText("Уровень " + num.ToString() + " - " +
Storage.menu_data.lvl_name[i]);
            }
            else
            {
                button[i].SetText("Уровень " + num.ToString() + " - Недоступен");
            }
        }
    }

    public void Exit()
    {
```

```
        Application.Quit();
    }

    public void SelectLevel(int i)
    {
        if (Storage.menu_data.lvl_active[i])
        {
            Storage.selected_level = i;
            int num = i + 1;
            start_button.SetText("Пройти " + num.ToString() + " Уровень");
        }
    }
    public void StartLevel()
    {
        if (Storage.selected_level > -1)
        {
            Storage.LoadLevelData(Storage.selected_level);
            SceneManager.LoadScene("GameDialogue");
        }
    }
    public void EditorMenu()
    {
        if (password.ToString() == Storage.menu_data.password)
        {
            SceneManager.LoadScene("EditorMenu");
        }
    }
}
```

ПРИЛОЖЕНИЕ В
Исходный код скрипта GameDialogue

Листинг 3 – Скрипт сцены Диалогов

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;

public class GameDialogue : MonoBehaviour
{
    public TMP_Text dialogue, display;
    private int position;
    private void LoadDialogue(int i)
    {

dialogue.SetText(Storage.level_data[Storage.selected_level].dialogue[i]);
        int level = Storage.selected_level + 1;
        int displaypos = position + 1;
  if (Storage.level_data[Storage.selected_level].character[i] == 1)
        {
            obj1.SetActive(true);
            obj2.SetActive(false);
        }
        else
        {
            obj2.SetActive(true);
            obj1.SetActive(false);
        }

        display.SetText("Уровень " + level.ToString() + " - Диалог " +
displaypos.ToString() + "/" +
Storage.menu_data.lvl_dialogues_num[Storage.selected_level]);
    }

    void Start()
    {        position = 0;
```

```
LoadDialogue(0);
    }
    public void Previous()
    {
        if (position > 0)
        {
            position--;
            LoadDialogue(position);
        }
    }

    public void SkipAll()
    {
        SceneManager.LoadScene("GameTest");
    }

    public void Next()
    {
        if (position + 1 <
Storage.menu_data.lvl_dialogues_num[Storage.selected_level])
        {
            position++;
            LoadDialogue(position);
        }
        else
        {
            SkipAll();
        }
    }

    public void MainMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }

}
```

ПРИЛОЖЕНИЕ Г
Исходный код скрипта GameTest

Листинг 4 – Скрипт сцены Тестов

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;

public class GameTest : MonoBehaviour
{
    public TMP_Text question, display, end_text;
    public List<TMP_Text> choice;
    public List<GameObject> checkmark;
    public GameObject again_button, end_menu, obj1, obj2;
    private int position;
    private List<int> answer;

    private void LoadTest(int i)
    {
question.SetText(Storage.level_data[Storage.selected_level].question[i]);
        int level = Storage.selected_level + 1;
        int displaypos = position + 1;
        display.SetText("Уровень " + level.ToString() + " - Вопрос " +
displaypos.ToString() + "/" +
Storage.menu_data.lvl_dialogues_num[Storage.selected_level]);
choice[0].SetText(Storage.level_data[Storage.selected_level].choice_0[i]);
choice[1].SetText(Storage.level_data[Storage.selected_level].choice_1[i]);
choice[2].SetText(Storage.level_data[Storage.selected_level].choice_2[i]);
choice[3].SetText(Storage.level_data[Storage.selected_level].choice_3[i]);
        //Deactivate Checkmarks
        for (int k = 0; k < 4; k++)
        {
            checkmark[k].SetActive(false);
        }
```

```
if (answer[i] > -1)
        {
            checkmark[answer[i]].SetActive(true);
        }
    }
    void Start()
    {
        position = 0;
        for (int i = 0; i <
Storage.menu_data.lvl_questions_num[Storage.selected_level]; i++)
        {
            answer[i] = -1;
        }
        LoadTest(position);
    }
    public void EndTest()
    {
        bool pass = true;
        for (int i = 0; (pass) && (i <
Storage.menu_data.lvl_questions_num[Storage.selected_level]); i++)
        {
            if (answer[i] !=
Storage.level_data[Storage.selected_level].correct_answer[i])
            {
                pass = false;
            }
        }
        obj1.SetActive(false);
        obj2.SetActive(false);
        end_menu.SetActive(true);
        if (pass)
        {
            end_text.SetText("Молодец! Уровень пройден!");
        }
        else
        {
            end_text.SetText("Попробуй Снова");
            again_button.SetActive(true);
```

```
}
    }
    public void Next()
    {
        position++;
        if (position <
Storage.menu_data.lvl_questions_num[Storage.selected_level]){
LoadTest(position);
        }
        else
        {
            EndTest();
        }
    }
    public void Click(int i)
    {
        answer[position] = i;
        checkmark[i].SetActive(true);
        for (int k = 0; k < 4; k++) {
            if (k != i) checkmark[k].SetActive(false);
        }
    }
    public void Previous()
    {
        if (position > 0) {
            position--; LoadTest(position);
        }
    }
    public void MainMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }
    public void Retry()
    {
        SceneManager.LoadScene("GameTest");
    }
}
```

ПРИЛОЖЕНИЕ Д
Исходный код скрипта EditorMenu

Листинг 5 – Скрипт сцены Режима Редактора

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;

public class EditorMenu : MonoBehaviour
{
    public List<GameObject> checkmark;
    public GameObject menu_password, menu_level, menu_name, obj1, obj2;
    public TMP_Text new_pass, new_name;
    public List<TMP_Text> button;

    void Start()
    {
        for (int i = 0; i < 16; i++)
        {
            checkmark[i].SetActive(Storage.menu_data.lvl_active[i]);
        }
        for (int i = 0; i < 16; i++)
        {
            int num = i + 1;

            if (Storage.menu_data.lvl_active[i])
            {
                button[i].SetText("Уровень " + num.ToString() + " - " +
Storage.menu_data.lvl_name[i]);
            }
            else
            {
                button[i].SetText("Уровень " + num.ToString() + " - Недоступен");
            }
        }
    }
```

```
public void MenuEditLevel(int i)
{
    Storage.selected_level = i;
    obj1.SetActive(false);
    obj2.SetActive(false);
    menu_level.SetActive(true);
}
public void EditDialogue()
{
    Storage.LoadLevelData(Storage.selected_level);
    SceneManager.LoadScene("EditDialogue");
}
public void EditTest()
{
    Storage.LoadLevelData(Storage.selected_level);
    SceneManager.LoadScene("EditTest");
}
public void MenuChangePassword()
{
    obj1.SetActive(false);
    obj2.SetActive(false);
    menu_password.SetActive(true);
}
public void ChangePassword()
{
    Storage.menu_data.password = new_pass.ToString();
}
public void MenuChangeName(int i)
{
    Storage.selected_level = i;
    obj1.SetActive(false);
    obj2.SetActive(false);
    menu_name.SetActive(true);
}
public void ChangeName()
{
    Storage.menu_data.lvl_name[Storage.selected_level] = new_name.ToString();
```

```
        if (Storage.menu_data.lvl_active[Storage.selected_level])
        {
            int num = Storage.selected_level + 1;
            button[Storage.selected_level].SetText("Уровень " + num.ToString() +
" - " + Storage.menu_data.lvl_name[Storage.selected_level]);
        }
    }
    public void SetActive(int i)
    {
        Storage.menu_data.lvl_active[i] = !(Storage.menu_data.lvl_active[i]);
        checkmark[i].SetActive(Storage.menu_data.lvl_active[i]);
    }

    public void SaveData()
    {
        Storage.SaveMenuData();
    }

    public void LoadData()
    {
        Storage.LoadMenuData();
        SceneManager.LoadScene("EditorMenu");
    }

    public void MainMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }

}
```

ПРИЛОЖЕНИЕ Е
Исходный код скрипта EditDialogue

Листинг 6 – Скрипт сцены Редактора Диалогов

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro;
public class EditDialogue : MonoBehaviour
{
    public TMP_Text dialogue, display, text;
    public GameObject obj1, obj2;
    private int position, new_character;
    private void LoadDialogue(int i)
    {
      dialogue.SetText(Storage.level_data[Storage.selected_level].dialogue[i]);
        int level = Storage.selected_level + 1;
        int displaypos = position + 1;
        if (Storage.level_data[Storage.selected_level].character[i] == 1)
        {
            obj1.SetActive(true);
            obj2.SetActive(false);
        }
        else
        {
            obj2.SetActive(true);
            obj1.SetActive(false);
        }
        display.SetText("Уровень " + level.ToString() + " - Диалог " +
displaypos.ToString() + "/" +
Storage.menu_data.lvl_dialogues_num[Storage.selected_level] + "+");
    }
    private void SaveText()
    {
 Storage.level_data[Storage.selected_level].dialogue[position] = text.ToString();
 Storage.level_data[Storage.selected_level].character[position] = new_character;
    }
```

```
    void Start()
    {
        position = 0;
        LoadDialogue(0);
    }
    public void Previous()
    {
        SaveText();
        if (position > 0)
        {   position--;
            LoadDialogue(position);
        }
    }
    public void Next()
    {
        SaveText();
        position++;
        if (position <
Storage.menu_data.lvl_dialogues_num[Storage.selected_level])
        {
            LoadDialogue(position);
        }
    }
    public void AddDialogue()
    {
        SaveText();
        if (Storage.menu_data.lvl_dialogues_num[Storage.selected_level] < 64)
        {   Storage.level_data[Storage.selected_level].dialogue.Add("");
            Storage.level_data[Storage.selected_level].character.Add(1);
LoadDialogue(Storage.menu_data.lvl_dialogues_num[Storage.selected_level]);
            Storage.menu_data.lvl_dialogues_num[Storage.selected_level]++;
        }
    }
    public void RemoveDialogue()
    {   SaveText();
        if (Storage.menu_data.lvl_dialogues_num[Storage.selected_level] > 4)
        {
```

```
Storage.level_data[Storage.selected_level].dialogue.RemoveAt(position);
Storage.level_data[Storage.selected_level].character.RemoveAt(position);
            Storage.menu_data.lvl_dialogues_num[Storage.selected_level]--;
            if (position - 1 < 0)
            { position = 0;
            }
            if (position ==
Storage.menu_data.lvl_dialogues_num[Storage.selected_level])
            { position--;
            } LoadDialogue(position);
        }
    }
    public void ChangeCharacter()
    {
        if (Storage.level_data[Storage.selected_level].character[position] == 1)
        { new_character = 0; }
        else
        { new_character = 1; }
Storage.level_data[Storage.selected_level].character[position] = new_character;
        SaveText();
        LoadDialogue(position);
    }
    public void SaveData()
    {
        Storage.SaveLevelData(Storage.selected_level);
    }
    public void LoadData()
    {
        Storage.LoadLevelData(Storage.selected_level);
        SceneManager.LoadScene("EditDialogue");
    }

    public void EditorMenu()
    {
        SceneManager.LoadScene("EditorMenu");
    }
}
```

ПРИЛОЖЕНИЕ Ж
Исходный код скрипта EditTest

Листинг 6 – Скрипт сцены Редактора Тестов

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using TMPro; public class EditTest : MonoBehaviour
{
    public TMP_Text question, display, text;
    public List<TMP_Text> choice;
    public List<GameObject> checkmark;
    private int position;
    private void LoadTest(int i)
    {
    question.SetText(Storage.level_data[Storage.selected_level].question[i]);
        int level = Storage.selected_level + 1;
        int displaypos = position + 1;
        display.SetText("Уровень " + level.ToString() + " - Вопрос " +
displaypos.ToString() + "/" +
Storage.menu_data.lvl_dialogues_num[Storage.selected_level] + "+");
choice[0].SetText(Storage.level_data[Storage.selected_level].choice_0[i]);
choice[1].SetText(Storage.level_data[Storage.selected_level].choice_1[i]);
choice[2].SetText(Storage.level_data[Storage.selected_level].choice_2[i]);
choice[3].SetText(Storage.level_data[Storage.selected_level].choice_3[i]);
        for (int k = 0; k < 4; k++)
        { checkmark[k].SetActive(false); }
checkmark[Storage.level_data[Storage.selected_level].correct_answer[position]].S
etActive(true);
    }
    private void SaveText()
    {
Storage.level_data[Storage.selected_level].question[position] = text.ToString();
Storage.level_data[Storage.selected_level].choice_0[position] =
choice[0].ToString();
Storage.level_data[Storage.selected_level].choice_1[position] =
choice[1].ToString();
```

```
Storage.level_data[Storage.selected_level].choice_2[position] =
choice[2].ToString();
Storage.level_data[Storage.selected_level].choice_3[position]=
choice[3].ToString();
    }
    void Start()
    {   position = 0;
        LoadTest(0);
    }
    public void Previous()
    {   SaveText();
        if (position > 0)
        { position--;
          LoadTest(position); }
    }
    public void Next()
    {   SaveText();
        position++;
        if (position <
Storage.menu_data.lvl_questions_num[Storage.selected_level])
        { LoadTest(position); }
    }
public void AddQuestion()
    {   SaveText();
        if (Storage.menu_data.lvl_questions_num[Storage.selected_level] < 32)
        {
            Storage.level_data[Storage.selected_level].question.Add("");
            Storage.level_data[Storage.selected_level].choice_0.Add("");
            Storage.level_data[Storage.selected_level].choice_1.Add("");
            Storage.level_data[Storage.selected_level].choice_2.Add("");
            Storage.level_data[Storage.selected_level].choice_3.Add("");
            Storage.level_data[Storage.selected_level].correct_answer.Add(0);
LoadTest(Storage.menu_data.lvl_questions_num[Storage.selected_level]);
            Storage.menu_data.lvl_questions_num[Storage.selected_level]++;
        }
    }
    public void RemoveQustion()
    {
```

```
        SaveText();
        if (Storage.menu_data.lvl_questions_num[Storage.selected_level] > 4)
        {
Storage.level_data[Storage.selected_level].question.RemoveAt(position);
Storage.level_data[Storage.selected_level].choice_0.RemoveAt(position);
Storage.level_data[Storage.selected_level].choice_1.RemoveAt(position);
Storage.level_data[Storage.selected_level].choice_2.RemoveAt(position);
Storage.level_data[Storage.selected_level].choice_3.RemoveAt(position);
            Storage.menu_data.lvl_questions_num[Storage.selected_level]--;
            if (position - 1 < 0)
            {   position = 0; }
            if (position ==
Storage.menu_data.lvl_questions_num[Storage.selected_level])
            {   position--;   }
            LoadTest(position);
        }
    }
    public void ChangeAnswer(int i)
    {
        Storage.level_data[Storage.selected_level].correct_answer[position] = i;
        SaveText();
        LoadTest(position);
    }
    public void SaveData()
    {
        Storage.SaveLevelData(Storage.selected_level);
    }
    public void LoadData()
    {
        Storage.LoadLevelData(Storage.selected_level);
        SceneManager.LoadScene("EditTest");
    }

    public void EditorMenu()
    {
        SceneManager.LoadScene("EditorMenu");
    }
}
```