**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное   
учреждение высшего образования   
«Санкт-Петербургский политехнический университет Петра Великого»  
ИНСТИТУТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

|  |  |
| --- | --- |
|  | **Допустить к защите** Заместитель директора  по УМР \_\_\_\_\_\_\_\_\_\_\_\_ Е.Г. Конакина\_  (Подпись) (ФИО)  «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2024 г. |
|  |

**КУРСОВОЙ ПРОЕКТ**Тема: «ИНФОРМАЦИОННАЯ СИСТЕМА библиотеки»

специальность 09.02.07 группа 32919/5

Студент (ка) Внукова М.А.  
 (подпись) (ФИО)  
Преподаватель Андреев В.А.

(подпись) (ФИО)

Санкт-Петербург  
2024

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 2](#_Toc168313195)

[1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ 3](#_Toc168313196)

[1.1 Описание предметной области 3](#_Toc168313197)

[1.2 Анализ методов решения 4](#_Toc168313198)

[1.3 Обоснование и выбор средств проектирования и разработки 5](#_Toc168313199)

[1.3.1 Обоснование выбора языка программирования 5](#_Toc168313200)

[1.3.2 Обоснование выбора используемой среды разработки 6](#_Toc168313201)

[1.3.3 Обоснование выбора игрового движка 7](#_Toc168313202)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 9](#_Toc168313203)

[2.1 Постановка задачи 9](#_Toc168313204)

[2.1.1 Основания для разработки 9](#_Toc168313205)

[2.1.2 Назначение программы 9](#_Toc168313206)

[2.2 Проектирование приложения 9](#_Toc168313207)

[2.2.1 Диаграмма прецедентов 10](#_Toc168313208)

[2.2.2 Диаграмма последовательностей 11](#_Toc168313209)

[2.2.3 Диаграмма классов 12](#_Toc168313210)

[2.2.4 Диаграмма активности 13](#_Toc168313211)

[2.3 Текст программы 14](#_Toc168313212)

[2.4 Описание программы 14](#_Toc168313213)

[2.4.1 Общие сведения 14](#_Toc168313214)

[2.4.2 Функциональное назначение 14](#_Toc168313215)

[2.4.3 Описание логической структуры прототипа 14](#_Toc168313216)

[2.4.4 Описание логической структуры программы на Unity 15](#_Toc168313217)

[2.4.5 Используемые технические и программные средства 16](#_Toc168313218)

[2.4.6 Вызов и загрузка 16](#_Toc168313219)

[2.5 Руководство оператора 16](#_Toc168313220)

[2.5.1 Назначение программы 16](#_Toc168313221)

[2.5.2 Выполнение программы и сообщения оператору 17](#_Toc168313222)

[2.6 Программа и методика испытаний 18](#_Toc168313223)

[2.6.1 Объект испытаний 18](#_Toc168313224)

[2.6.2 Цель испытаний 18](#_Toc168313225)

[2.6.3 Требования к программе 18](#_Toc168313226)

[2.6.4 Требования к программной документации 19](#_Toc168313227)

[2.6.5 Средства и порядок испытаний 20](#_Toc168313228)

[2.6.6 Методы испытаний 21](#_Toc168313229)

[2.7 Протокол испытаний 23](#_Toc168313230)

[ЗАКЛЮЧЕНИЕ 26](#_Toc168313231)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 27](#_Toc168313232)

[ПРИЛОЖЕНИЕ А 28](#_Toc168313233)

[ПРИЛОЖЕНИЕ Б 35](#_Toc168313234)

[ПРИЛОЖЕНИЕ В 47](#_Toc168313235)

# ВВЕДЕНИЕ

Action RPG дают игроку более полный контроль над персонажами, нежели их тактические собратья, это наиболее заметно в отношении движения и боя. Термин Action RPG чаще используется, когда речь идет о консольных играх. На ПК этот жанр изначально называли hack'n slash.

Данный проект актуален, поскольку сейчас очень популярны компьютерные игры. Компьютерная игра про рыцарей очень интересна пользователям.

Цель курсового проекта – разработка и создание собственной игры с названием «Драконы и рыцари».

Задачи курсового проекта:

* Собрать требования, написать техническое задание с полным перечнем характеристик будущей игры.
* Спроектировать UML-диаграммы деятельности, классов, последовательности и активности.
* Написать и протестировать первый прототип игры.
* Оформить документацию к первому прототипу.
* Изучить Unity.
* Написать на Unity и протестировать второй (окончательный) вид игры.

## ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ

### Описание предметной области

Компьютерная игра — компьютерная программа, служащая для организации игрового процесса (геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра. Геймплей — компонент игры, отвечающий за интерактивное взаимодействие игры и игрока. Геймплей описывает, как игрок взаимодействует с игровым миром, как игровой мир реагирует на действия игрока и как определяется набор действий, который игра предлагает игроку.

Компьютерная ролевая игра (англ. computer role-playing game, обозначается аббревиатурой CRPG или RPG) — жанр компьютерных игр, в котором игрок управляет одним или несколькими персонажами, каждый из которых описан набором численных характеристик, списком способностей и умений; примерами таких характеристик могут быть очки здоровья (англ. hit points, HP), показатели силы, ловкости, интеллекта, защиты, уклонения, уровень развития того или иного навыка и т. п. Многие ролевые видеоигры берут свое начало в настольных ролевых играх и используют во многом одну и ту же терминологию, жаргон, сеттинг, игровые механики с такими настольными играми, как Dungeons and Dragons. Ключевая особенность жанра является то, что персонажи растут в способностях, и этот рост, зачастую, контролируется игроком. Ролевые игры реже полагаются на физическую координацию и реакцию, в отличие от жанра «action RPG». Ролевые игры превратились из простых текстовых игр в визуально богатые трехмерные игры.

Экшен (action в переводе с англ. — «действие») или боевик (по аналогии с киножанром) — жанр компьютерных игр, в котором делается упор на эксплуатацию физических возможностей игрока, таких как зрительно-моторная координация и скорость реакции. Жанр включает в себя большое количество поджанров — от файтингов, шутеров и платформеров, которые считаются наиболее важными для жанра, до MOBA и некоторых стратегий в реальном времени, которые также возможно отнести к жанру экшен.

Action-RPG, ролевая игра в жанре «экшн» или ролевой экшн (англ. action role-playing game, action RPG, ARPG) — поджанр компьютерных игр, в котором объединяются ключевые элементы жанров экшн и ролевых игр.

### Анализ методов решения

В наше время существует множество компьютерных игр. Все они очень разные, но разделить их можно на 2D и 3D.

Для проекта был выбран стиль игры 2D.

В таблице 1 представлено сравнение стилей игр 2D и 3D.

Таблица 1 – Таблица сравнения 2D и 3D.

|  |  |  |
| --- | --- | --- |
| **Различия** | **2D** | **3D** |
| Оси движения | Представляют собой игры только с двумя осями движения. Это плоские игры, в которых можно перемещаться влево и вправо, а также вверх и вниз | Игрок может перемещаться в «реальном мире», где он может поворачиваться на 360 градусов, и в котором объекты имеют длину, высоту и глубину |
| Простота перемещения игрока | Из-за того, что у них не так много вариантов передвижения, 2D-игры проще, чем их 3D-аналоги | Включают в себя полное движение в трехмерных плоскостях |
| Управление | Управление в 2D-играх часто относительно простое | Полный диапазон 3D движений |
| Вид объектов | Во многих 2D-играх объекты существуют в виде так называемого «спрайта» | Объекты могут рассматриваться под любым углом |
| Камера | Сильно упрощена | С перспективой |

Обоснование выбора:

* 2D игры проще в создании, что идеально подходит для учебного проекта.
* Не нужно создавать или скачивать тяжелые 3D-модели, достаточно нарисовать плоские спрайты.
* Не нужно настраивать камеру и плоскости.

### Обоснование и выбор средств проектирования и разработки

#### Обоснование выбора языка программирования

Был выбран язык программирования C# (си-шарп).

В таблице 2 представлено сравнение языков программирования С#, С++, Java, Python и R.

Таблица 2 – Таблица сравнения языков программирования С#, С++, Java, Python и R.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Отличия** | **С#** | **С++** | **Java** | **Python** | **R** |
| Встроенные библиотеки для работы с данными | - | - | - | + | + |
| Высокая производительность | + | + | + | - | - |
| Бесплатная лицензия | + | + | + | + | + |
| Простота в изучении | + | - | - | + | + |
| Универсальность | + | + | + | + | - |
| Строгая типизация | + | + | + | - | - |
| Эффективность для работы с данными | + | - | - | + | + |

Причины выбора языка C#:

* C# создавался параллельно с каркасом Framework .Net и в полной мере учитывает все его возможности - как FCL, так и CLR;
* C# является полностью объектно-ориентированным языком, где даже типы, встроенные в язык, представлены классами;
* C# является мощным объектным языком с возможностями наследования и универсализации;
* C# является наследником языков C/C++, сохраняя лучшие черты этих популярных языков программирования. Общий с этими языками синтаксис, знакомые операторы языка облегчают переход программистов от С++ к C#;
* Сохранив основные черты своего великого родителя, язык стал проще и надежнее. Простота и надежность, главным образом, связаны с тем, что на C# хотя и допускаются, но не поощряются такие опасные свойства С++ как указатели, адресация, разыменование, адресная арифметика;
* Мощная библиотека каркаса поддерживает удобство построения различных типов приложений на C#, позволяя легко строить Web-службы, другие виды компонентов, достаточно просто сохранять и получать информацию из базы данных и других хранилищ данных;
* Реализация, сочетающая построение надежного и эффективного кода, является немаловажным фактором, способствующим успеху C#.

#### Обоснование выбора используемой среды разработки

Для разработки пользовательского приложения было выбрано инструментальное ПО «Microsoft Visual Studio».

В таблице 3 представлено сравнение сред разработки Microsoft Visual Studio, KDevelopи Gambas.

Таблица 3 – Таблица сравнения сред разработки Microsoft Visual Studio, KDevelopи Gambas.

|  |  |  |  |
| --- | --- | --- | --- |
| **Отличия** | **MS Visual Studio** | **KDevelop** | **Gambas** |
| Совместимость с Windows | + | + | - |
| Совместимость с Linux | + | + | + |
| Совместимость с macOS | - | + | - |
| Язык, на котором написана среда разработки | C ++/C# | C/C ++ | BASIC |
| Поддерживает C# | + | + | - |
| Возможность разработки Windows Forms | + | - | - |

Причины выбора Microsoft Visual Studio:

* Удобная среда разработки форм Windows, в которых будет создан первый прототип игры;
* Удобная настройка событий нажатия на кнопку;
* Возможность отладки и отслеживания всех переменных в коде;
* Поддержка C#.

#### Обоснование выбора игрового движка

Для разработки второй версии проекта был выбран игровой движок «Unity» – кроссплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies.

В таблице 4 представлено сравнение игровых движков Unity и Unreal Engine.

Таблица 4 – Таблица сравнения игровых движков Unity и Unreal Engine.

|  |  |  |
| --- | --- | --- |
| **Отличия** | **Unity** | **Unreal Engine** |
| Определение | Можно определить как передовое программное обеспечение для разработки игр для создания 2D и 3D игр | Можно определить как игровой движок, который используется для создания различных типов видеоигр |
| Разработчик | Unity Technologies | Epic Games |
| Лицензия | Персональная – бесплатно; лицензия pro за 1800 долларов в год, т.е. 150 долларов в месяц | Игра бесплатна, но при заработке в 1 миллион долларов взимается роялти в размере 5% |
| Рендеринг | Медленнее | Быстрее |
| Открытый исходный код | Нет | Есть |
| Доля на рынке | 11% | 23% |
| Язык, на котором написан движок | C # и C ++ | C ++ и Blueprint |
| Сообщество | Большое сообщество, насчитывающее около 210 тыс. участников | Меньшее сообщество по сравнению с Unity |

Причины выбора Unity:

* Поддержка C# ассетов;
* Большое количество информации и обучающих материалов в свободном доступе в Интернете;
* Бесплатная лицензия;
* Популярность движка.

## ПРАКТИЧЕСКАЯ ЧАСТЬ

### Постановка задачи

#### Основания для разработки

Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Разработка программных модулей» и утверждена Институтом среднего профессионального образования.

#### Назначение программы

Игровая программа с графическим интерфейсом «Драконы и рыцари»: игрок выбирает персонажа, с помощью которого он сражается с драконом. При победе игрок переходит на следующий уровень.

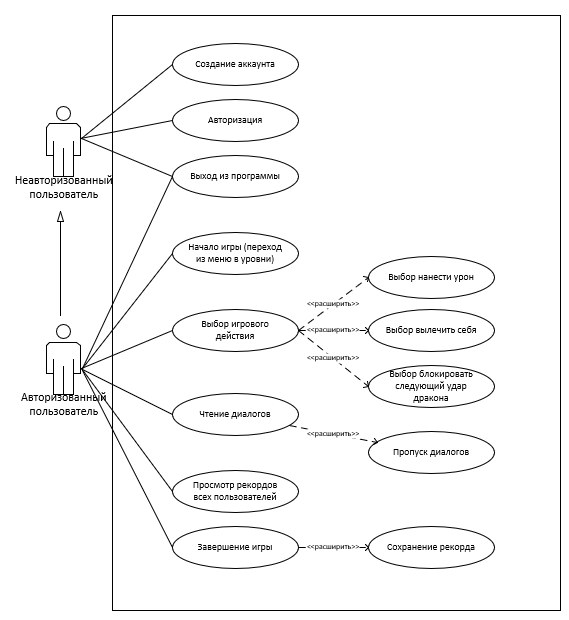
Полный текст технического задания представлен в Приложении А.

### Проектирование приложения

На этапе проектирования были разработаны диаграмма прецедентов, диаграмма последовательностей, диаграмма классов и диаграмма активности.

#### Диаграмма прецедентов

Диаграммы прецедентов частично описывает use case – прецедент использования проектируемой системы, давая частичное описание частичного применения системы с точки зрения условного внешнего обозревателя. (рисунок 1).

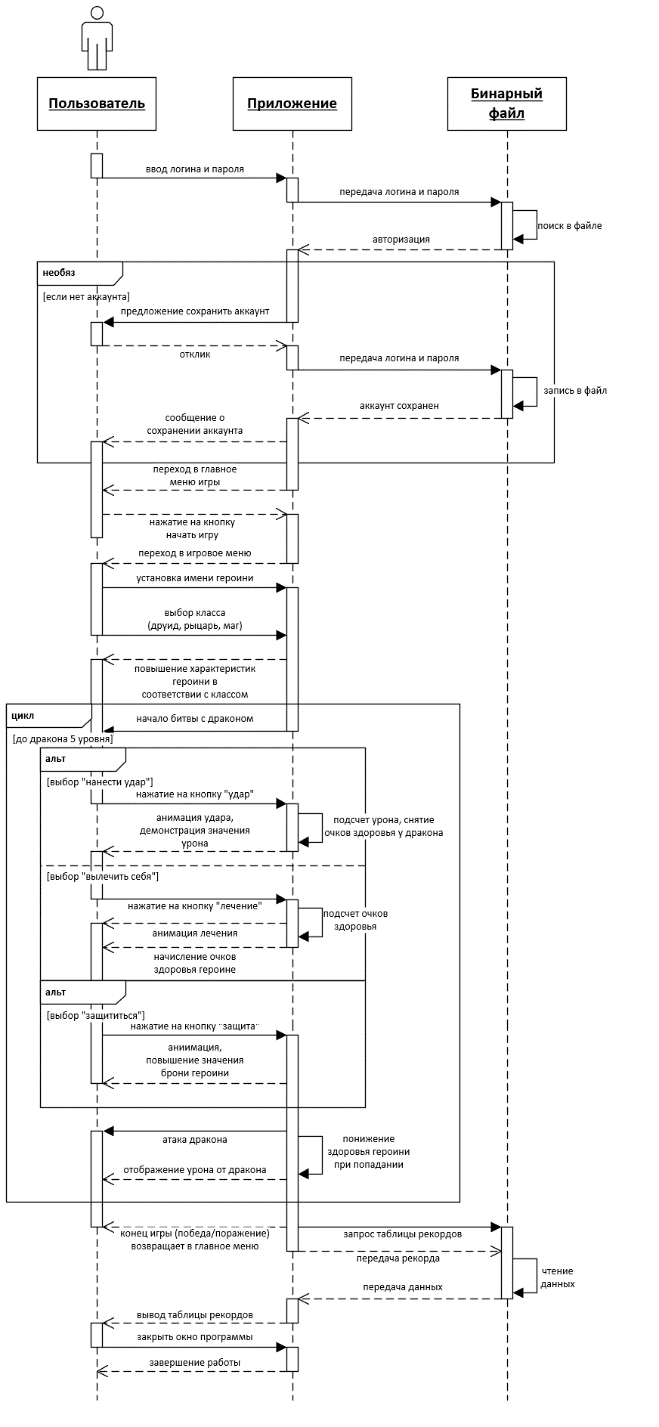


1. Диаграмма прецедентов

#### Диаграмма последовательностей

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования.

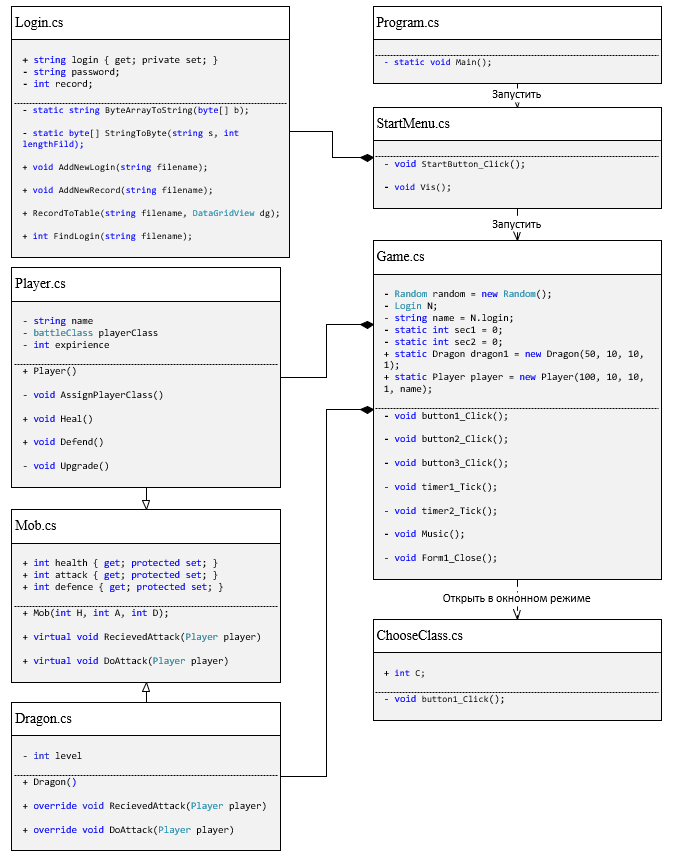
Все действующие лица показаны в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между объектами для выполнения требуемых функций (рисунок 2).



1. Диаграмма последовательностей

#### Диаграмма классов

На рисунке 3 показана диаграмма классов проекта. Программа содержит несколько основных классов: Game – класс формы игры. Класс Login.cs содержит методы работы с бинарным файлом логинов и рекордами игроков. Этот класс использует модуль ChooseClass во время авторизации и при отображении таблицы рекордов в форме Records. Классы Mob.cs, Dragon.cs и Player.cs (последние два наследуют от Mob.cs) – классы с методами мобов и игрока.

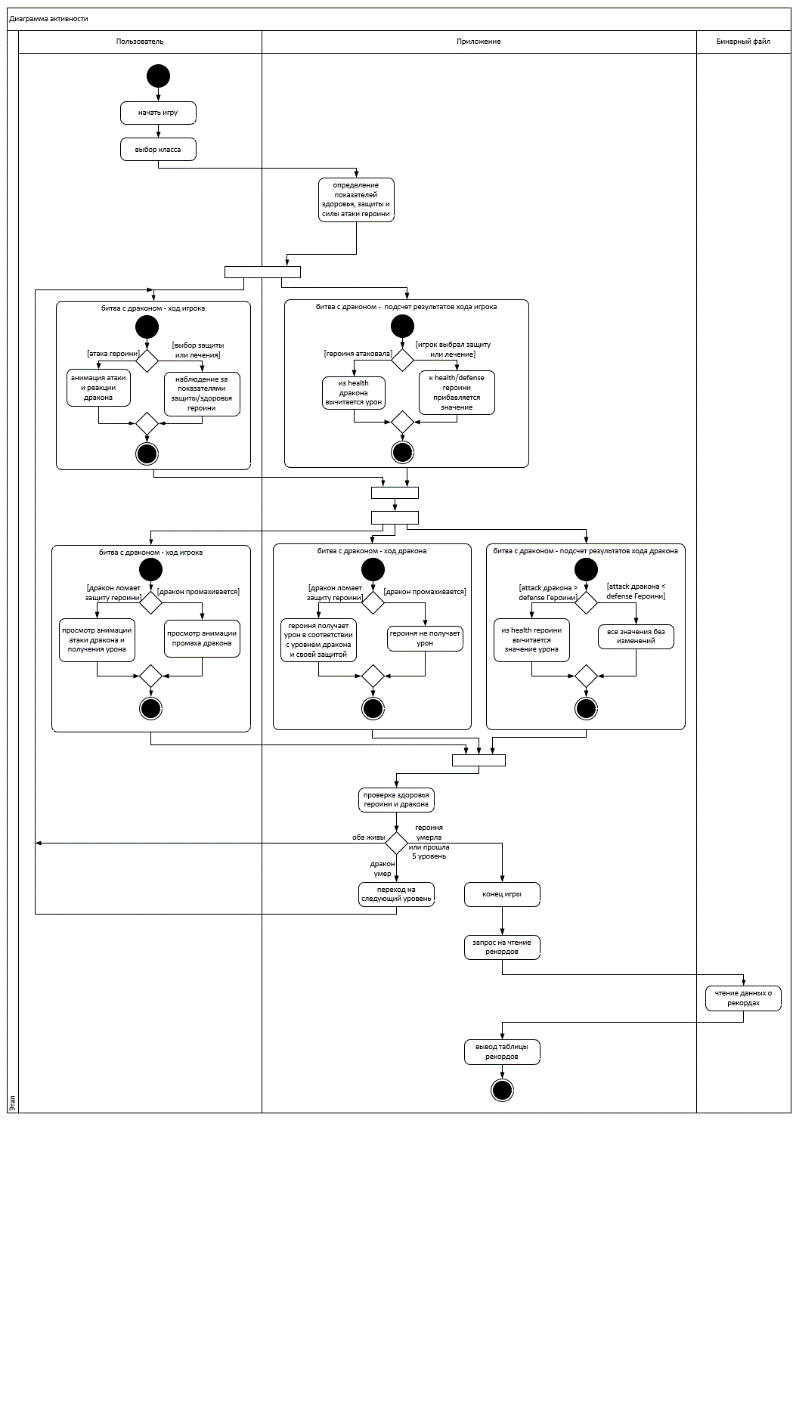


1. Диаграмма классов

#### Диаграмма активности

На рисунке 4 показана диаграмма активности (деятельности) во время сеанса игры.

Диаграмма активности – это UML-диаграмма, на которой показаны действия и состояния.



1. Диаграмма активности

### Текст программы

Текст программы в соответствии с ГОСТ 19.101-77 (СТ СЭВ 1626-79) и ГОСТ 19.401-79 (СТ СЭВ 3746-82) представляет собой запись программы на исходном языке программирования с необходимыми комментариями. Текст программы представляет собой документ, выполненный машинным способом, и приведен в приложении В.

### Описание программы

#### Общие сведения

Игровая программа, цель которой заключается в победе над драконом всех пяти уровней.

#### Функциональное назначение

Основное назначение программного продукта: компьютерная игра в жанре «Action/RPG».

Эксплуатационное назначение программного продукта: программа предназначена для широкого круга пользователей, без ограничения по возрасту, не требует внесения денежных средств или использования платёжных систем для игры, предназначена для развлекательных целей.

#### Описание логической структуры прототипа

Прототип содержит 2 основных модуля:

Game.cs — модуль формы игры, модуль Login.cs — содержит методы работы с бинарным файлом логинов и рекордов игроков. Этот класс использует модуль Game во время авторизации и при отображении таблицы рекордов.

Программа использует функции следующих библиотек среды С#:

System, System.Collections.Generic, System.ComponentModel, System.Data, System.Drawing, System.Linq, System.Text, System.Threading.Tasks, System.Windows.Forms.

Исполняемый файл прототипа создан средствами среды C#, имеет имя “прототип1.exe” и размер 1 252 352 байт.

#### Описание логической структуры программы на Unity

Программа, написанная на Unity, содержит 5 сцен и 3 основных скрипта:

Сцены:

StartScene – загрузочная сцена с кнопками начала игры, просмотра рекорда и выхода из игры, SetNameScene – сцена с вводом имени и выбором класса персонажа, DialogScene – сцена отображения диалога, ActualGameScene – сцена, на которой происходит сама игра и переход на новые уровни, RecordsScene – сцена с таблицей рекордов.

Основные скрипты:

Game.cs – скрипт, используемый на сцене ActualGameScene, Login.cs – скрипт сохранения логина и рекорда в файл, PlayerDataScript.cs – скрипт, содержащий данные, введенные игроком, такие как выбранный класс и имя, Dialog.cs содержит в себе текст диалогов и методы их воспроизведения, DialogScript связывает методы прошлого класса со сценой диалога в игре.

Программа использует функции следующих библиотек C# и Unity:

System, System.Collections, System.Collections.Generic, UnityEngine, UnityEngine.UI, TMPro, UnityEngine.SceneManagement.

Исполняемый файл основной программы создан при помощи движка Unity, имеет имя “My project v1.exe”. и размер 666 624 байт.

#### Используемые технические и программные средства

Для нормального функционирования информационной системы необходим компьютер, клавиатура, мышь и следующие технические средства:

- процессор Intel или другой совместимый;

- объем свободной оперативной памяти ~500 Кб;

- объем необходимой памяти на жестком диске ~20Мб;

- стандартный VGA-монитор или совместимый;

- стандартная клавиатура;

- манипулятор «мышь»;

- дополнительно: звуковая карта, колонки или наушники.

#### Вызов и загрузка

Программа может быть загружена как с диска, так и с жесткого диска. В последнем случае требуется предварительно переписать программу с диска на жесткий диск.

Исполняемым файлом программы является файл My project v1.exe. Для его запуска необходимо дважды щелкнуть по исполняемому файлу левой кнопкой мышки.

### Руководство оператора

#### Назначение программы

Игровая программа с графическим интерфейсом «Драконы и рыцари»: игрок выбирает персонажа, с помощью которого он сражается с драконом. При победе игрок переходит на следующий уровень. Цель игры – победа над драконом пятого уровня.

#### Выполнение программы и сообщения оператору

Для запуска программы дважды щелкните левой кнопкой мыши по исполняемому файлу My project v1.exe.

Процесс игры, этапы работы приложения показаны в виде иллюстраций в приложении Б.

При запуске работы приложения загружается первая сцена StartScene, на которой расположены кнопки начала игры, просмотра рекордов и выхода из игры. При нажатии на кнопку начала игры происходит переход на сцену ChangeName, на которой появляется панель для ввода имени и выбора класса (по умолчанию выбран класс Рыцарь). При вводе имени становится доступна кнопка «Продолжить», которая переводит на сцену с диалогом. На сцене DialogScene можно прочитать диалог, нажимая на кнопку «Дальше», или пропустить его нажатием на соответствующую кнопку. После диалога программа переходит непосредственно к сцене игры ActualGameScene. Перед началом игры пользователю дается начальное здоровье в соответствии с выбранным классом. У пользователя есть выбор игрового действия – атака, защита или лечение, и соответствующие кнопки на сцене. При атаке персонаж ударяет по дракону, снимая ему жизни. Дракон бьет в ответ. При защите дракон бьет персонажа пользователя, но тот не получает урона. При лечении персонаж восстанавливает себе часть здоровья. Игра заканчивается при победе над драконом 5 уровня.

Новый рекорд пользователя записывается вместо предыдущего, если во время сеанса игры удалось превысить предыдущий рекорд.

### Программа и методика испытаний

#### Объект испытаний

Объектом испытаний является исполняемый файл программы «My project v1.exe». Игровая программа с графическим интерфейсом «Драконы и рыцари»: игрок выбирает персонажа, с помощью которого он сражается с драконом. При победе игрок переходит на следующий уровень.

Уровень дракона повышается при победе над ним, вплоть до 5 уровня, при победе над драконом 5 уровня игра заканчивается.

#### Цель испытаний

Целью испытаний является проверка соответствия программы требованиям Технического Задания.

#### Требования к программе

1 версия:

* программа должна иметь графический интерфейс в Windows Forms;
* программа должна позволять создание героини: ввод имени и выбор одного из трех классов (рыцарь, Волшебница, друид). Каждый класс обладает тремя статами: уровень здоровья, сила урона, уровень защиты (сопротивление урону);
* программа должна начинать бой с драконом после нажатия на кнопку «Начать игру». Дракон и героиня атакуют по очереди. В бою программа должна генерировать случайный (в определенном диапазоне) урон в соответствии с классом героини. Урон дракона зависит от его уровня (также задается в диапазоне);
* при победе над драконом героиня переходит на этап сражения с драконом следующего уровня, также она прокачивает свои навыки;
* игра заканчивается поражением при потере всех очков здоровья героиней, и возвращает пользователя на начальный экран;
* игра заканчивается победой при прохождении героиней всех уровней игры, и возвращает пользователя на начальный экран;
* программа может завершать работу при закрытии ее пользователем в любой момент игры.

2 версия:

* программа должна иметь графический интерфейс в стиле 8-bit, созданный при помощи Unity;
* после победы над драконами определенных уровней становятся доступны диалоги с NPC, которые можно пропустить;
* программа должна высчитывать рекорд в соответствии с количеством нанесенного драконам урона и его уровнем;
* программа должна сохранять рекорд пользователя;
* программа должна обеспечивать просмотр таблицы рекордов всех пользователей;
* программа должна иметь звуковое сопровождение, которое можно включить или отключить.

#### Требования к программной документации

На испытания должны быть представлены следующие программные документы:

* техническое задание;
* текст программы;
* описание программы;
* руководство оператора;
* описание языка.

#### Средства и порядок испытаний

Для проведения испытаний необходимы следующие технические средства:

- процессор Intel или другой совместимый;

- объем свободной оперативной памяти ~500 Кб;

- объем необходимой памяти на жестком диске ~3Мб;

- стандартный VGA-монитор или совместимый;

- стандартная клавиатура;

- манипулятор «мышь»;

- дополнительно: звуковая карты, колонки или наушники.

Для проведения испытаний необходимы следующие программные средства:

Операционная система Windows 10.

Испытания проводятся в следующем порядке:

1. проверяется наличие и комплектность программной документации (п.2.6.4);
2. проверяется соответствие требованиям к маркировке и упаковке (п.2.6.3.3);
3. проверяется соответствие требованиям к функциональным характеристикам (п.2.6.3.1);
4. проверяется соответствие требованиям к информационной и программной совместимости (п.2.6.3.2).

#### Методы испытаний

##### Для проверки способности программы инициализировать персонажа с определенными статами в зависимости от класса, необходимо:

* запустить программу;
* ввести имя и выбрать класс Рыцарь;
* посмотреть данные, выведенные в игровой сцене;
* выйти из программы и повторить шаги для классов Волшебница и Друид;
* убедиться, что разные классы обладают разным значением здоровья

##### Для проверки способности программы обеспечивать пошаговую битву (ход персонажа и ответный ход дракона), необходимо:

* запустить программу;
* в игре нажать на «Атаку»;
* убедиться, что сначала атакует персонаж, а потом дракон, и они наносят друг другу урон

##### Для проверки способности программы переходить на дракона следующего уровня при победе, и прокачивать уровень и здоровье персонажа, необходимо:

* запустить программу;
* начать сражение с драконом и победить его на 1 уровне;
* убедиться, что при победе уровень дракона изменился на 2, а здоровье персонажа увеличилось

##### Для проверки способности программы вернуться на начальный экран при потере всех очков здоровья, необходимо:

* запустить программу;
* выбрать класс «Волшебница», открыть сцену игры;
* дать персонажу погибнуть;
* убедиться, что программа возвращается на сцену StartScene

##### Для проверки способности программы вернуться на начальный экран при победе над драконом 5 уровня, необходимо:

* запустить программу;
* выбрать любой класс, кроме класса «Волшебница»;
* победить дракона на 5 уровне;
* убедиться, что программа возвращается на сцену StartScene

##### Для проверки способности программы закрыться в любой момент по желанию пользователя, необходимо:

* запустить программу;
* нажать на кнопку «Выйти из игры» в любой момент игры;
* убедиться, что игра закрывается

##### Для проверки способности программы показывать и пропускать диалоги с NPC, необходимо:

* запустить программу;
* ввести имя, перейти к следующей сцене;
* в диалоге нажимать на кнопку «Дальше»;
* после победы над драконом 1 уровня в диалоге нажать на кнопку «Пропустить»;
* пройти игру;
* убедиться, что программа показывает диалог при начале игры, после 1, 3 и 5 уровней и при гибели персонажа, также убедиться, что игра позволяет пропускать диалоги

##### Для проверки способности программы высчитывать, сохранять и отображать рекорд пользователя, необходимо:

* запустить программу;
* открыть RecordsScene из меню;
* убедиться, что программа сохранила рекорд

##### Для проверки способности программы обеспечивать музыкальное сопровождение, необходимо:

* запустить программу;
* открыть сцену игры;
* нажать на кнопку включения/выключения звука;
* убедиться, что программа проигрывает/останавливает музыку

##### Для проверки способности программы отображать анимацию лечения, необходимо:

* запустить программу;
* нажать на кнопку лечения;
* убедиться, что программа отображает лечение и анимацию

### Протокол испытаний

Результаты испытаний программы представлены в таблице 5, рисунки приведены в приложении Б.

Таблица 5 – результаты испытаний программы

| № | Проверяемые требования | Сообщения программы и вводимые значения | Ожидаемые результаты | Результат |
| --- | --- | --- | --- | --- |
| 1 | Способность программы инициализировать персонажа с определенными статами в зависимости от класса | запустить программу;  ввести имя и выбрать класс Рыцарь;  посмотреть данные, выведенные в игровой сцене;  выйти из программы и повторить шаги для классов Волшебница и Друид | Разные классы обладают разным значением здоровья, что видно в игровой сцене | Пройден  Рис. Б1-Б5 |
| 2 | Способность программы обеспечивать пошаговую битву (ход персонажа и ответный ход дракона) | запустить программу;  в игре нажать на «Атаку» | Сначала атакует персонаж, а потом дракон, и они наносят друг другу урон | Пройден  Рис. Б6-Б7 |
| 3 | Способность программы переходить на дракона следующего уровня при победе, и прокачивать уровень и здоровье персонажа | запустить программу;  начать сражение с драконом и победить его на 1 уровне | При победе уровень дракона изменился на 2, а здоровье персонажа увеличилось | Пройден  Рис. Б8-Б9 |
| 4 | Способность программы вернуться на начальный экран при потере всех очков здоровья | запустить программу;  выбрать класс «Волшебница», открыть сцену игры;  дать персонажу погибнуть | Программа возвращается на сцену StartScene | Пройден  Рис. Б10-Б11 |
| 5 | Способность программы вернуться на начальный экран при победе над драконом последнего, 5 уровня | запустить программу;  выбрать любой класс, кроме класса «Волшебница»;  победить дракона на 5 уровне | убедиться, что программа возвращается на сцену StartScene | Пройден  Рис. Б12-Б13 |
| 6 | Способность программы закрыться в любой момент по желанию пользователя | запустить программу;  нажать на кнопку «Выйти из игры» в любой момент игры; | игра закрывается | Пройден  Рис. Б14 |
| 7 | Способность программы показывать и пропускать диалоги с NPC | запустить программу;  ввести имя, перейти к следующей сцене;  в диалоге нажимать на кнопку «Дальше»;  после победы над драконом 1 уровня в диалоге нажать на кнопку «Пропустить»;  пройти игру | программа показывает диалог при начале игры, после 1, 3 и 5 уровней и при гибели персонажа, также игра позволяет пропускать диалоги | Пройден  Рис. Б15-21 |
| 8 | Способность программы высчитывать, сохранять и отображать рекорд пользователя | запустить программу;  открыть RecordsScene из меню | Программа отобразила рекорд | Пройден  Рис. Б22 |
| 9 | Способность программы обеспечивать музыкальное сопровождение | запустить программу;  открыть сцену игры;  нажать на кнопку включения/ выключения звука | программа проигрывает/ останавливает музыку | Пройден  Рис. Б23 |
| 10 | Способность программы отображать анимацию лечения | запустить программу;  открыть сцену игры;  нажать на кнопку лечения; | убедиться, что программа отображает лечение и анимацию | Пройден  Рис. Б24 |

# ЗАКЛЮЧЕНИЕ

Разработанная в ходе выполнения курсового проекта программа удовлетворяет всем требованиям технического задания, что подтверждается протоколом испытаний.

Разработанная программа может быть использована в развлекательных целях, для отдыха и релаксации.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд ., Язык программирования C#, 4-е издание, СПб:-ПИТЕР, 2020
2. Бейт М., Unity за 24 часа, 2-е издание, СПб:-ПИТЕР, 2022
3. Культин Н., Microsoft Visual C# в задачах и примерах, 2-е издание, СПб:-БХВ, 2015
4. Жанры игр и их особенности [Электронный ресурс] — URL: [https://sreda.temadnya.com/ 1043184354218215685/zhanry-igr-i-ih-osobennosti/](https://sreda.temadnya.com/%201043184354218215685/zhanry-igr-i-ih-osobennosti/) (дата обращения: 12.09.2020)
5. Описание языка C# [Электронный ресурс] — URL: https://techrocks.ru/2019/02/16/c-sharp-programming-language-overview/ (дата обращения: 03.12.2019)
6. Стандарты Единой Системы Программной Документации:

ГОСТ 19.105-78 Общие требования к программным документам

ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом

ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению

ГОСТ 19.301-78 Программа и методика испытаний. Требования к содержанию и оформлению

ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению

ГОСТ 19.402-78 Описание программы. Требования к содержанию и оформлению

ГОСТ 19.505-79 Руководство оператора. Требования к содержанию и оформлению

ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

# ПРИЛОЖЕНИЕ А

Министерство ОБРАЗОВАНИЯ И науки Российской Федерации  
Федеральное автономное образовательное учреждение высшего профессионального образования  
«Санкт-Петербургский государственный политехнический университет Петра Великого»  
(ФГАОУ ВО «СПбПУ)  
**Институт среднего профессионального образования**

УТВЕРЖДАЮ  
Председатель ПЦК   
\_\_\_\_\_\_\_\_\_\_\_ Андреев В.А.  
\_\_ .\_\_. 2024

Игра с графическим интерфейсом «Драконы и рыцари»  
**Техническое задание**  
Листов 7

ПРОВЕРИЛ  
Преподаватель   
\_\_\_\_\_\_\_\_\_\_\_ Девятко Н.С.  
22.01.2024

ВЫПОЛНИЛ  
Студент группы 32919/6   
\_\_\_\_\_\_\_\_\_\_\_ Внукова М.А.  
22.01.2024

2024

ВВЕДЕНИЕ

* 1. Полное наименование программной разработки: «ИГРА С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ «ДРАКОНЫ И РЫЦАРИ».
  2. Игровая программа с графическим интерфейсом «Драконы и рыцари»: человек играет против машины. Человек играет за героиню (рыцаря), компьютер играет за дракона. Присутствует небольшой сюжет.
  3. В соответствии с заданием программный продукт состоит из теоретической и практической частей. Теоретическая часть включает описание предметной области задачи, анализ методов её решения, обзор и обоснование выбора средств программирования. Практическая часть включает подробное описание проектирования программного продукта, разработку и реализацию на языке программирования C# программных модулей программного продукта с использованием среды программирования Visual Studio, и графического движка Unity для создания графического интерфейса игры.

1. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ
   1. Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Разработка программных модулей» и утверждена Институтом среднего профессионального образования.
2. НАЗНАЧЕНИЕ РАЗРАБОТКИ
   1. Основное назначение программного продукта: компьютерная игра в жанре «Action/RPG».
   2. Эксплуатационное назначение программного продукта: программа предназначена для широкого круга пользователей, без ограничения по возрасту, не требует внесения денежных средств или использования платёжных систем для игры, предназначена для развлекательных целей.

4. ТРЕБОВАНИЯ К РАЗРАБОТКЕ

4.1. Требования к функциональным характеристикам:

**1 версия:**

* программа должна иметь графический интерфейс в Windows Forms;
* программа должна позволять создание героини: ввод имени и выбор одного из трех классов (рыцарь, Волшебница, друид). Каждый класс обладает тремя статами: уровень здоровья, сила урона, уровень защиты (сопротивление урону);
* программа должна начинать бой с драконом после нажатия на кнопку «Начать игру». Дракон и героиня атакуют по очереди. В бою программа должна генерировать случайный (в определенном диапазоне) урон в соответствии с классом героини. Урон дракона зависит от его уровня (также задается в диапазоне);
* при победе над драконом героиня переходит на этап сражения с драконом следующего уровня, также она прокачивает свои навыки;
* игра заканчивается поражением при потере всех очков здоровья героиней, и возвращает пользователя на начальный экран;
* игра заканчивается победой при прохождении героиней всех уровней игры, и возвращает пользователя на начальный экран;
* программа может завершать работу при закрытии ее пользователем в любой момент игры.

**2 версия:**

* программа должна иметь графический интерфейс в стиле 8-bit, созданный при помощи Unity;
* после победы над драконами определенных уровней становятся доступны диалоги с NPC, которые можно пропустить;
* программа должна высчитывать рекорд в соответствии с количеством нанесенного драконам урона и его уровнем;
* программа должна сохранять рекорд пользователя;
* программа должна обеспечивать просмотр таблицы рекордов всех пользователей;
* программа должна иметь звуковое сопровождение, которое можно включить или отключить.

4.2. Требования к надежности:

* использование лицензированного программного обеспечения;
* проверка программы на наличие вирусов;
* организация бесперебойного питания.

4.3. Требования к составу и параметрам технических средств

Для нормального функционирования информационной системы необходим компьютер, клавиатура, мышь и технические средства:

- процессор Intel или другой совместимый;

- объем свободной оперативной памяти ~500 Кб;

- объем необходимой памяти на жестком диске ~20Мб;

- стандартный VGA-монитор или совместимый;

- стандартная клавиатура;

- манипулятор «мышь»;

- дополнительно: звуковая карта, колонки или наушники.

4.4. Требования к информационной и программной совместимости

Для полноценного функционирования данной системы необходимо наличие операционной системы выше Microsoft Windows 10 или совместимой. Язык интерфейса – русский.

4.5. Требования к маркировке и упаковке

Программа должна поставляться в виде проекта, исполняемого (еxе) файла, установщика и документации.

4.6. Требования к транспортировке и хранению

Программа распространяется в электронном виде. Требования к транспортировке и хранению не предъявляются.

4.7. Специальные требования

Теоретическая часть включает описание предметной области задачи, анализ методов её решения, обзор и обоснование выбора средств программирования. Практическая часть включает подробное описание проектирования программного продукта, разработку и реализацию на языке программирования C# с использованием среды программирования Visual Studio программных модулей программного продукта.

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.1. Предварительный состав программной документации:

- «Техническое задание»;

- разрабатываемые программные модули должны быть самодокументированы, т. е. тексты программ должны содержать все необходимые комментарии;

- разрабатываемое программное обеспечение должно включать справочную систему.

5.2. Перечень материалов пояснительной записки

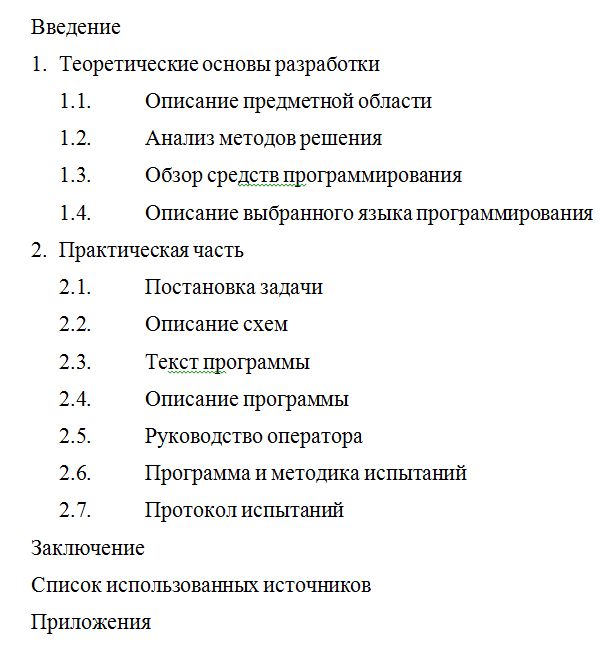


Рисунок 1 – Перечень материалов пояснительной записки

6. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Технико-экономические показатели не рассчитываются.

7. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

Таблица 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Содержание стадии** | **Содержание этапа** | **Срок 2024 г.** | | **Форма  отчетности** |
| **начало** | **конец** |
| Техническое задание | Составление технического задания | 15.01 | 21.01 | Техническое задание |
| Эскизный проект | Проектирование программы | 22.01 | 18.02 | UML-диаграммы |
| Рабочий проект | Разработка спецификаций | 19.02 | 03.03 | Спецификации отдельных компонентов |
| Составление программы | 04.03 | 31.03 | Программная документация |
| Приёмо-сдаточные испытания | 01.04 | 14.04 | Протокол испытаний (п. 2.7 пояснительной записки) |
| Приёмка | Защита курсового проекта | 15.04 | 20.04 | Оценка за курсовой проект |

8. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

8.1. Порядок контроля

Контроль выполнения должен осуществляться руководителем курсового проекта (преподавателем) в соответствие с п.7.

8.2. Порядок приемки

Приемка должна осуществляться с участием руководителя после проведения приемо-сдаточных испытаний. В результате защиты курсового проекта должна быть выставлена оценка за курсовой проект.

# ПРИЛОЖЕНИЕ Б



Рисунок Б1 – запуск программы



Рисунок Б2 – ввод имени и выбор класса «Рыцарь»

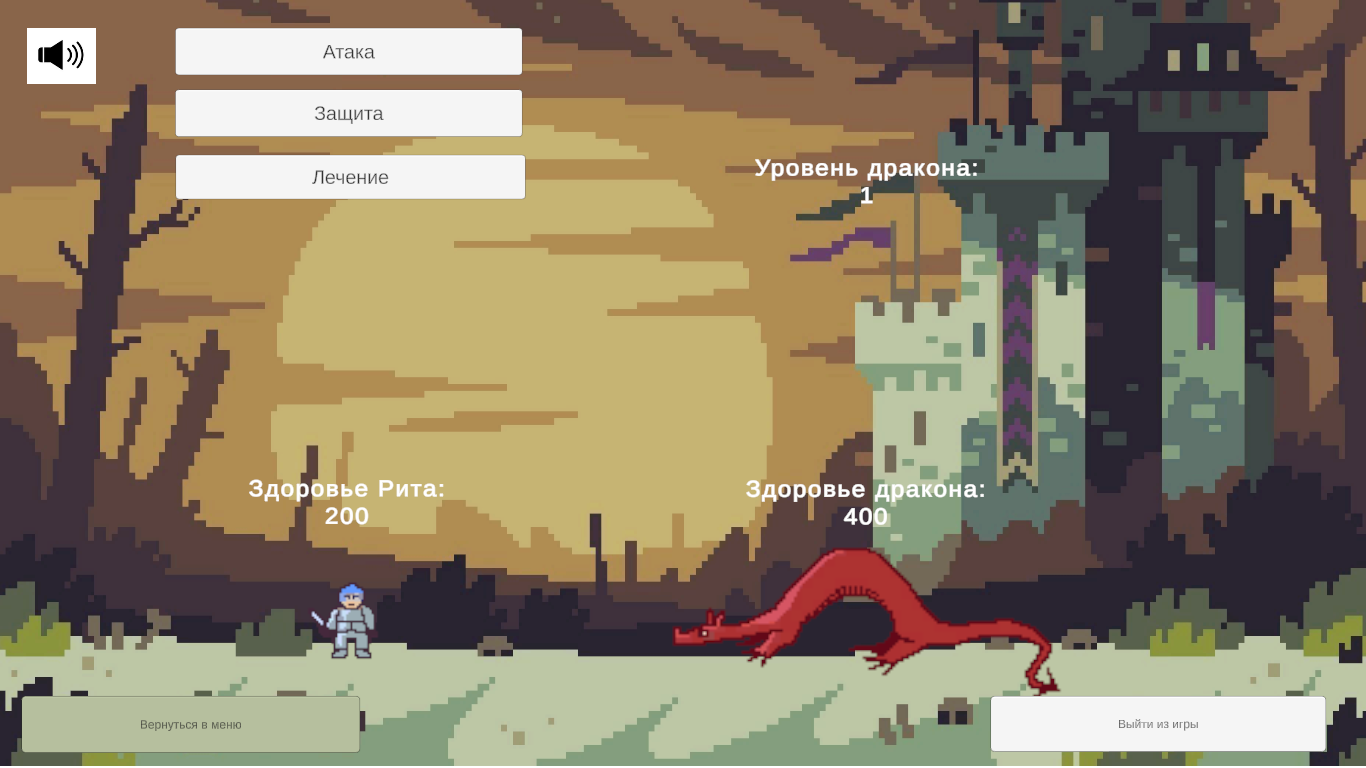


Рисунок Б3 – здоровье рыцаря

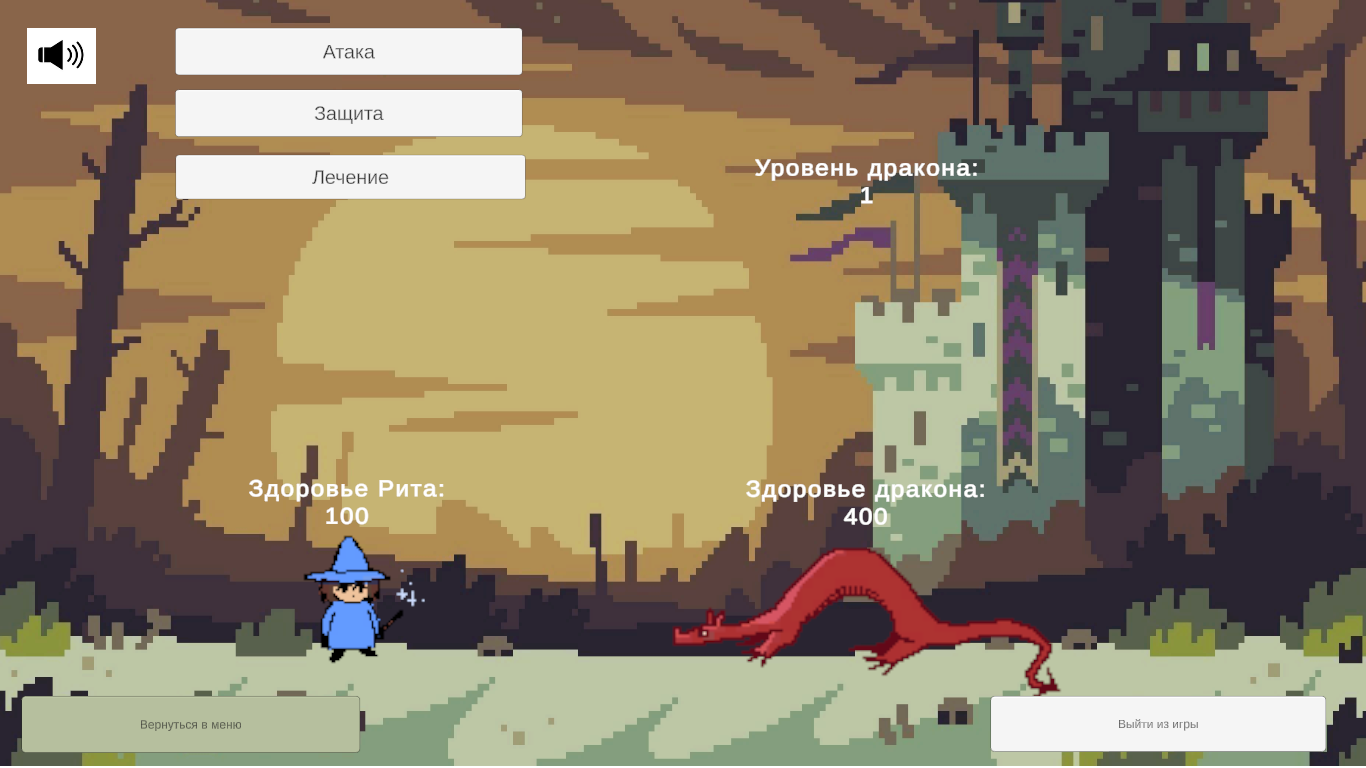


Рисунок Б4 – здоровье волшебницы

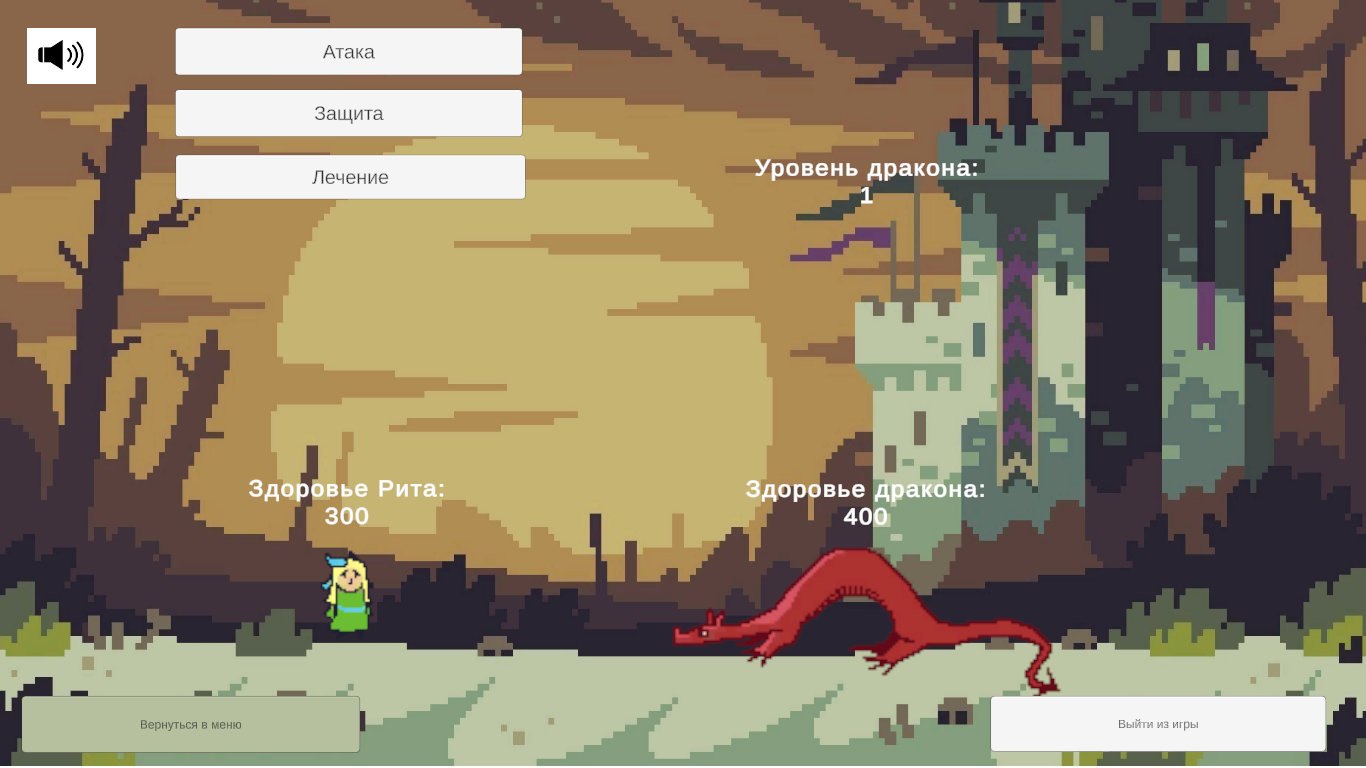


Рисунок Б5 – здоровье друида

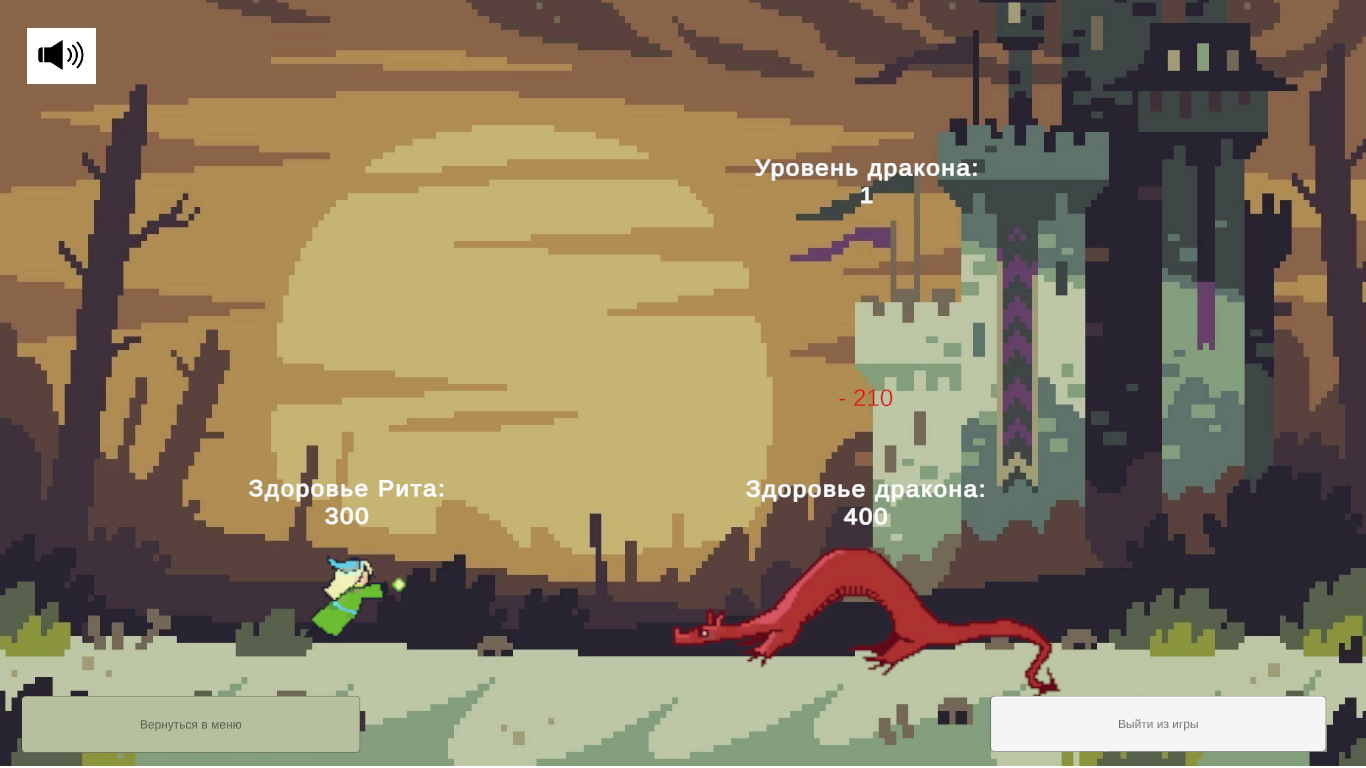


Рисунок Б6 – атака персонажа

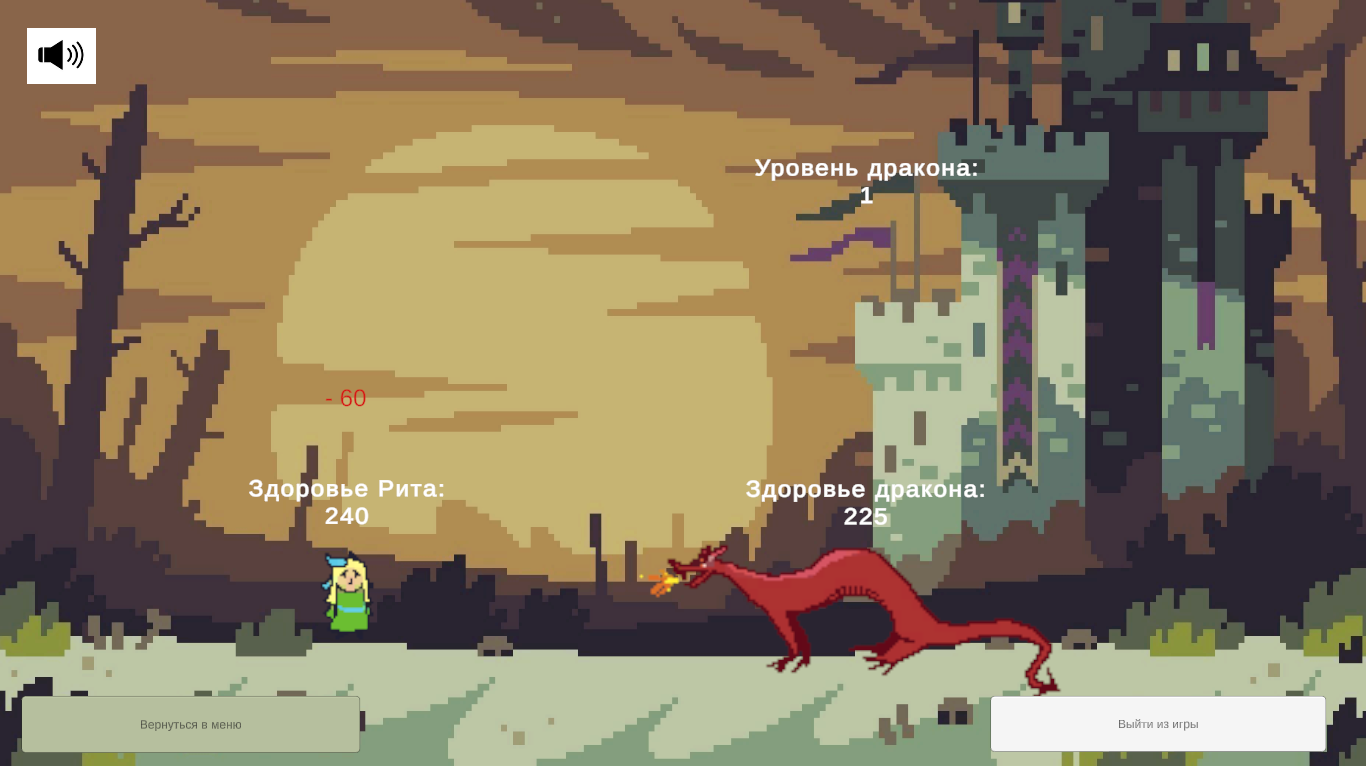


Рисунок Б7 – ответная атака дракона

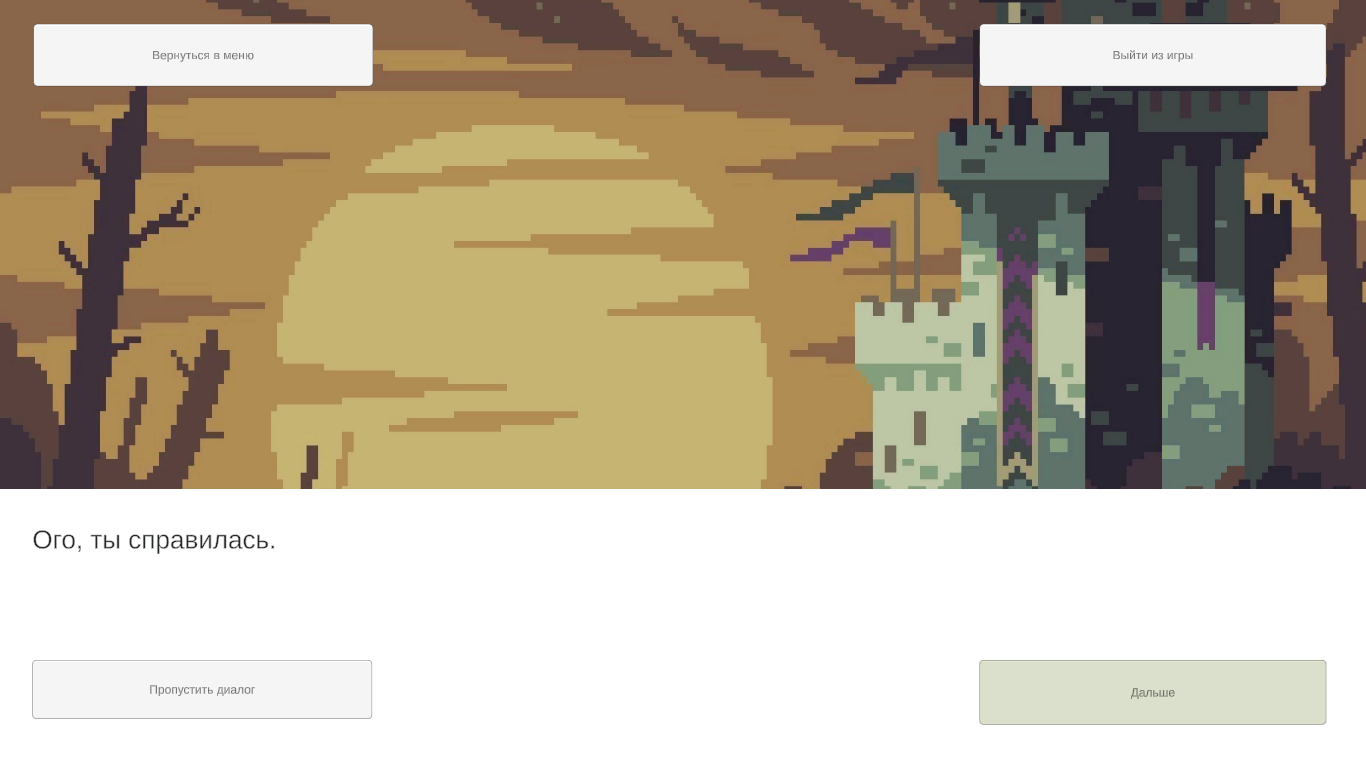


Рисунок Б8 – победа над драконом

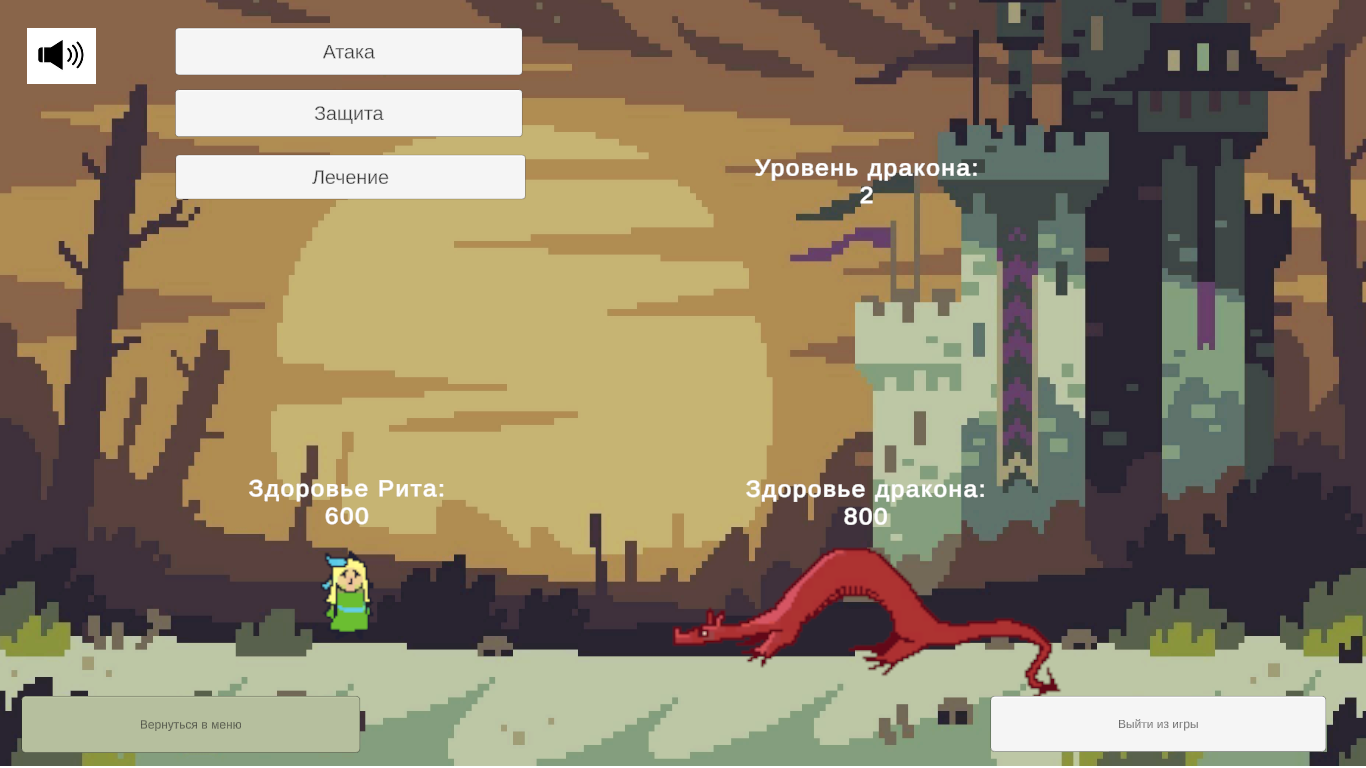


Рисунок Б9 – переход на 2 уровень

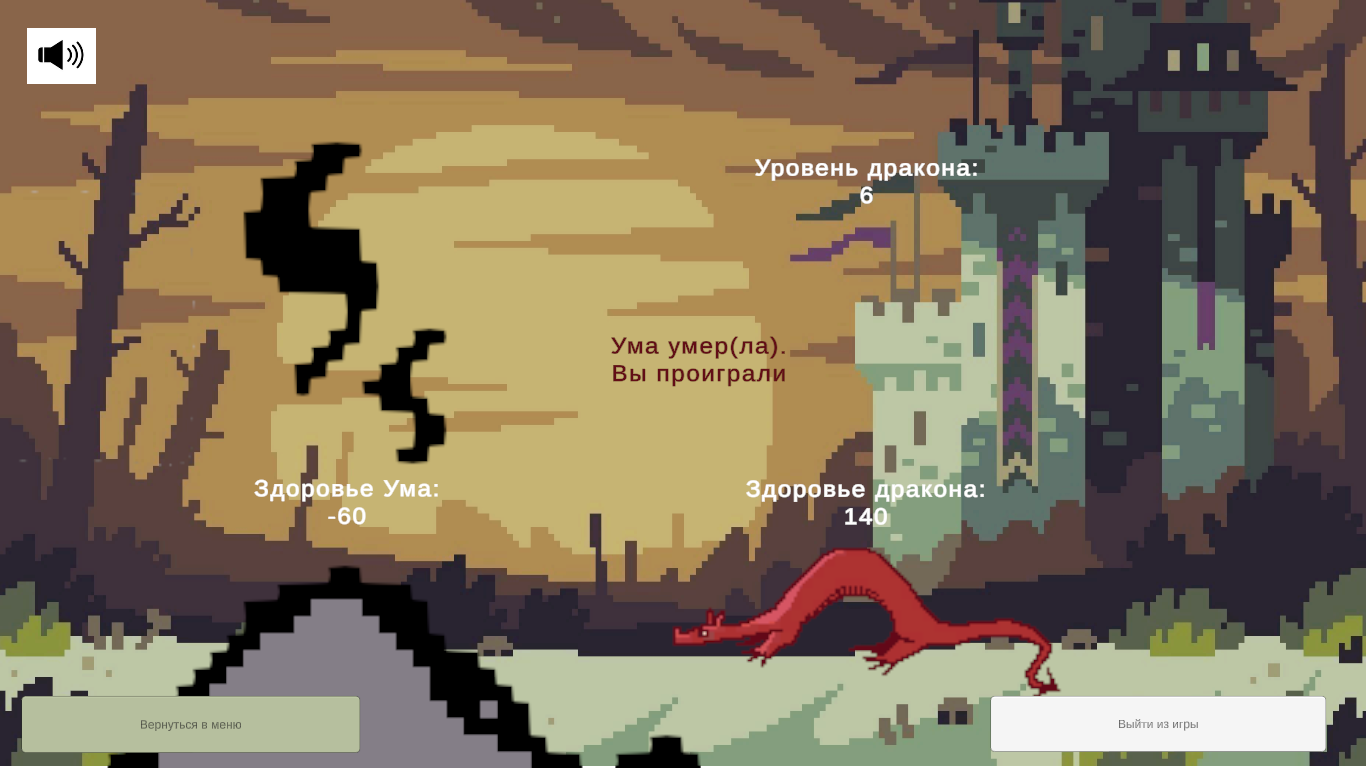


Рисунок Б10 – смерть персонажа



Рисунок Б11 – возвращение в меню

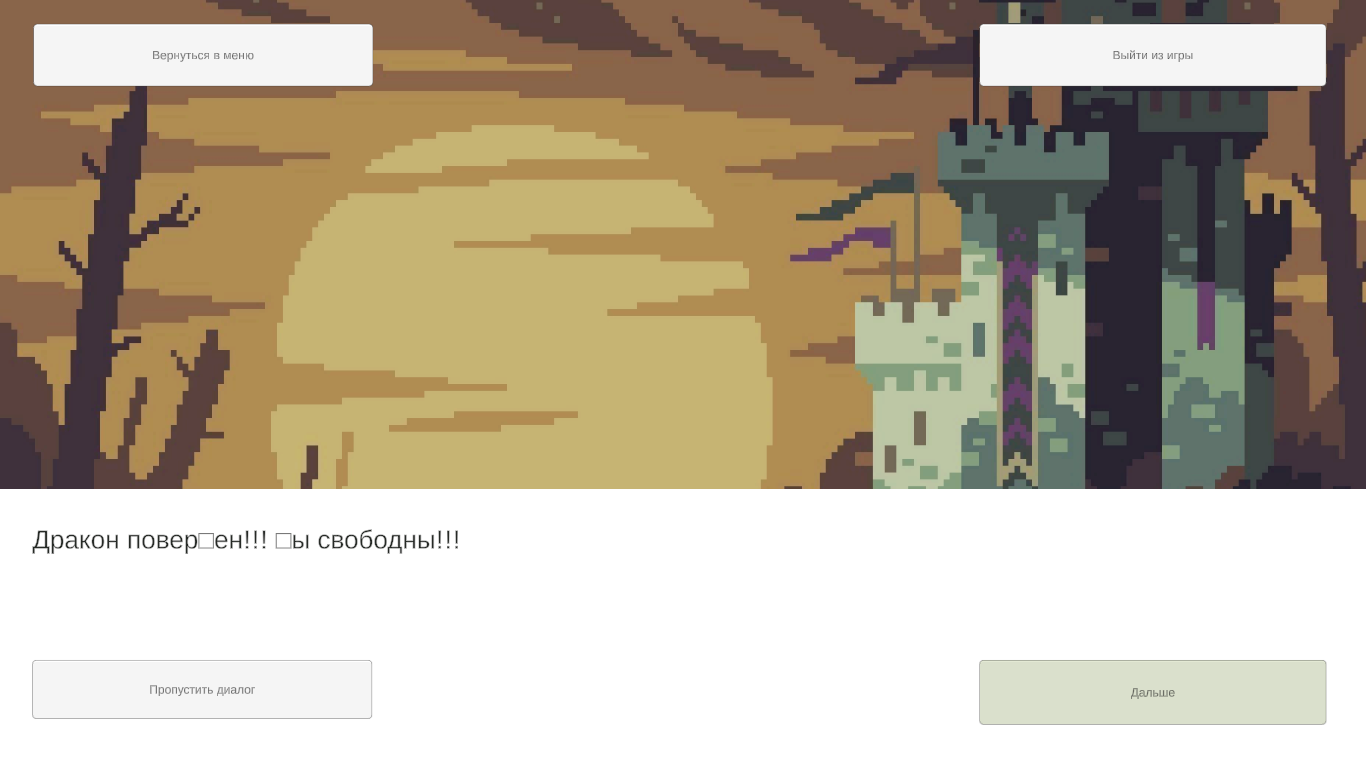


Рисунок Б12 – победа над драконом 5 уровня



Рисунок Б13 – возвращение в меню



Рисунок Б14 – игра закрылась

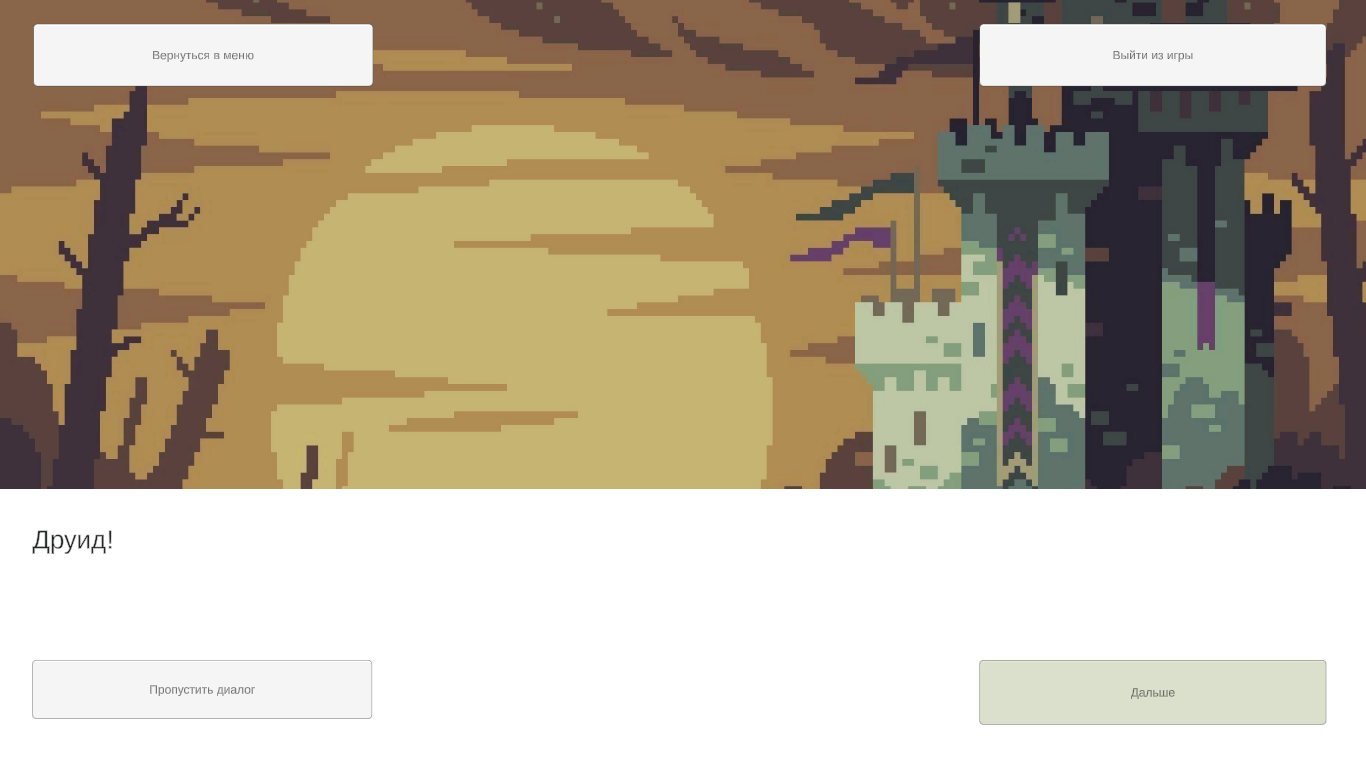


Рисунок Б15 – диалог при начале игры

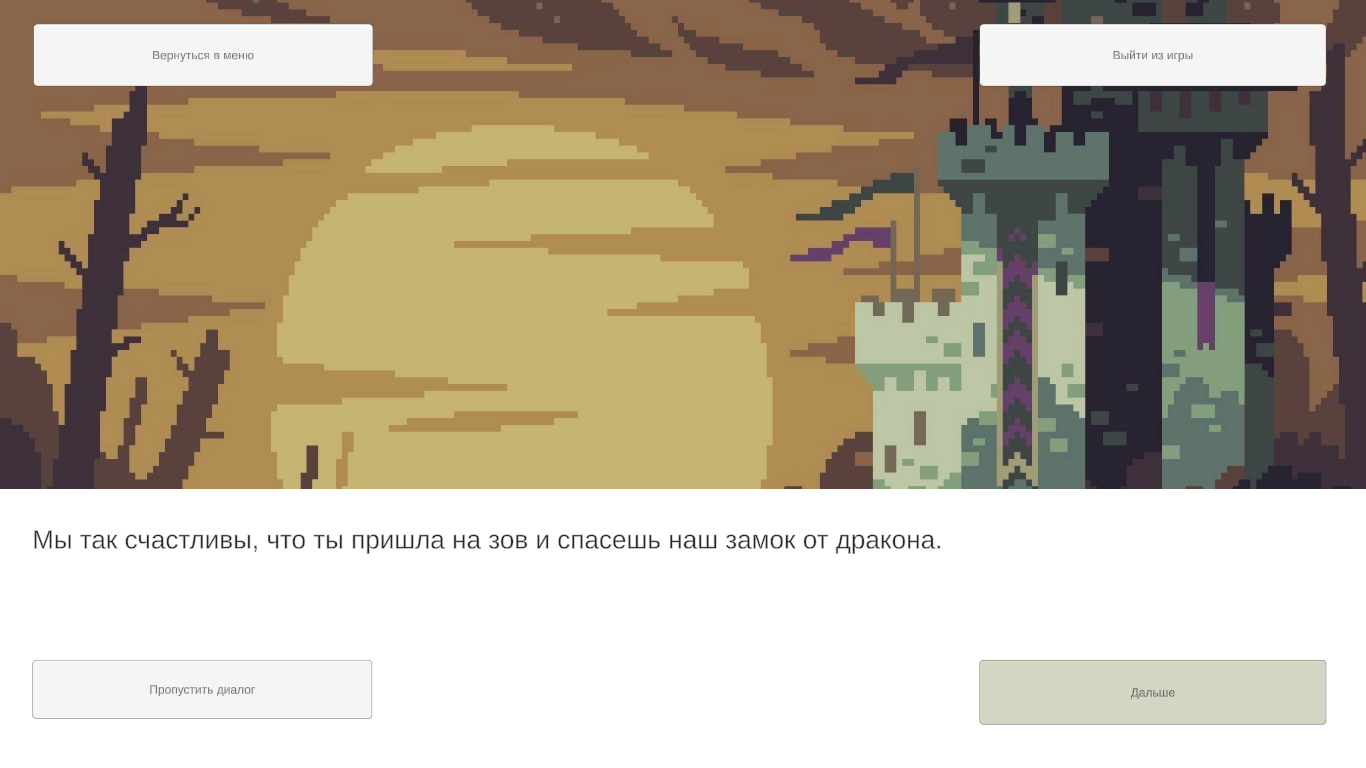


Рисунок Б16 – дальше

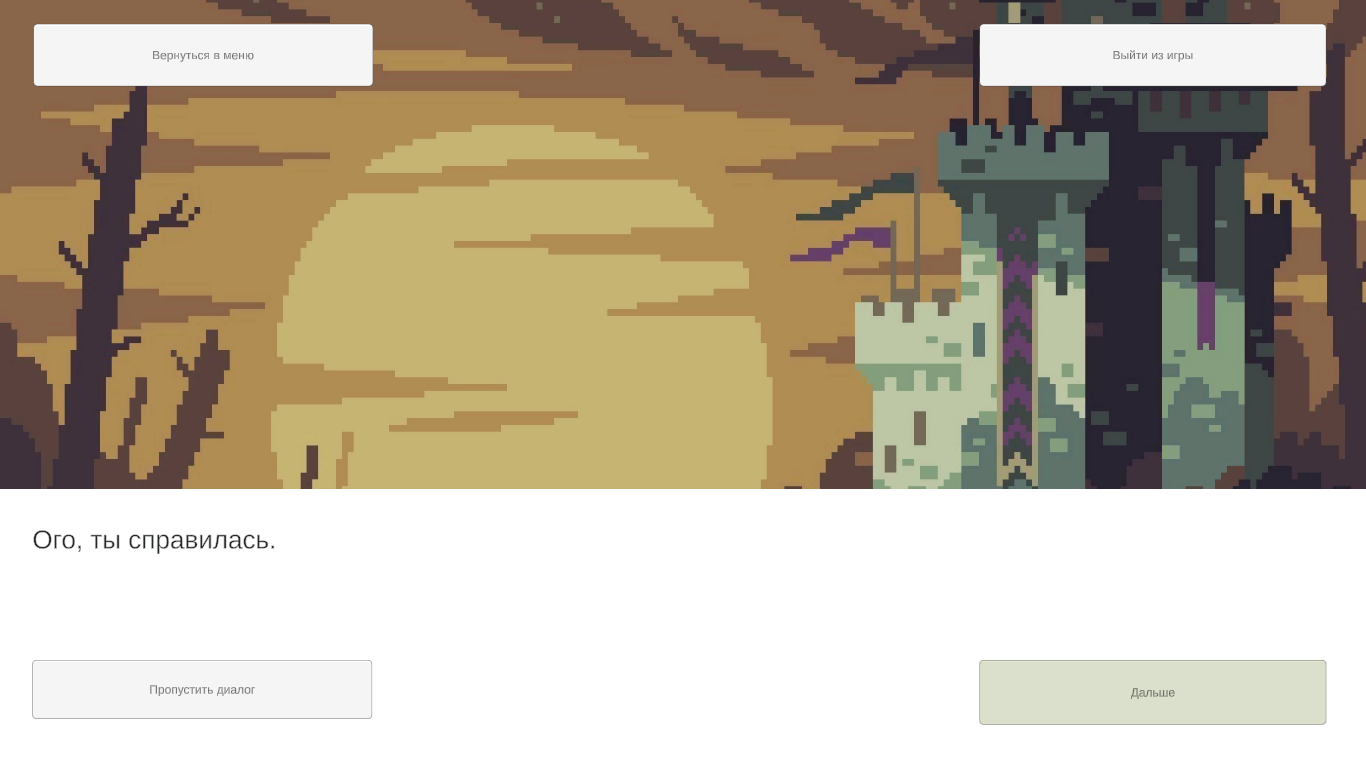


Рисунок Б17 – диалог после 1 уровня

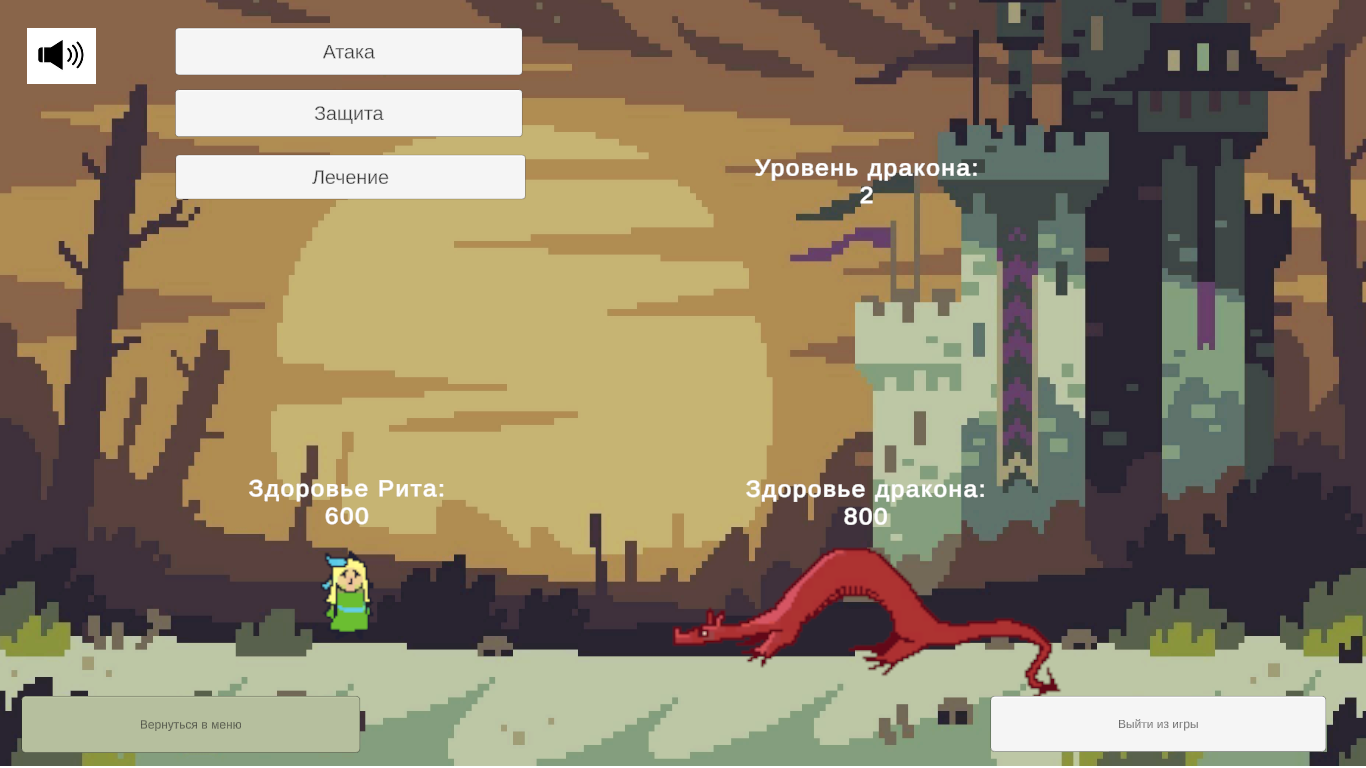


Рисунок Б18 – пропустить

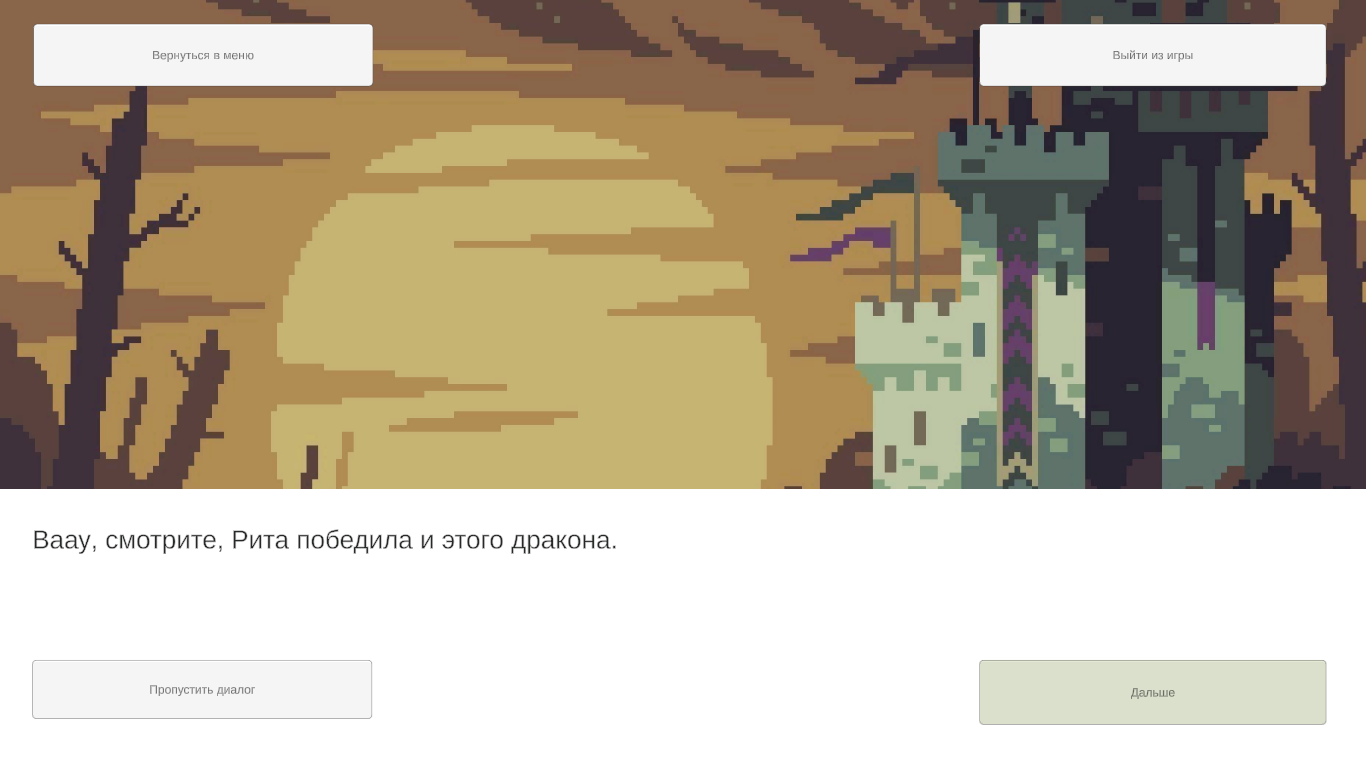


Рисунок Б19 – диалог после 3 уровня

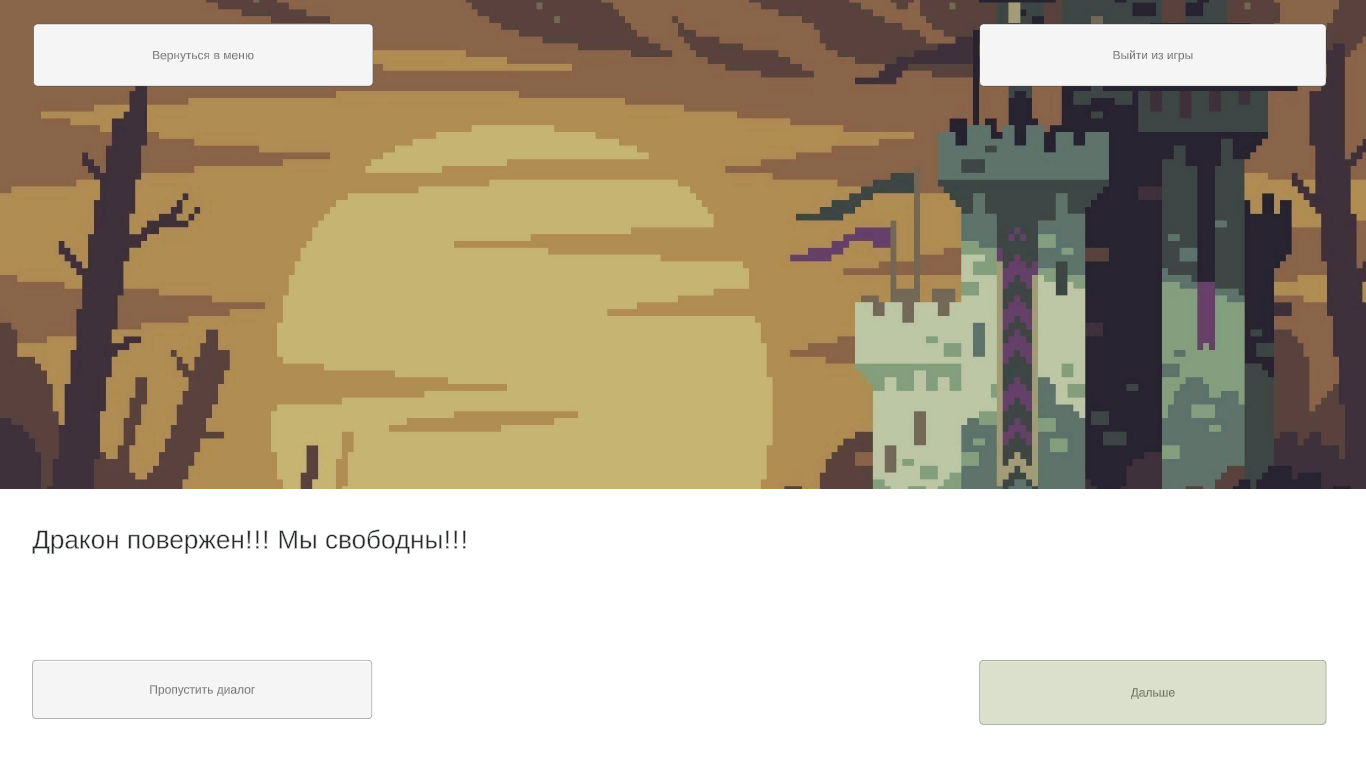


Рисунок Б20 – диалог после 5 уровня

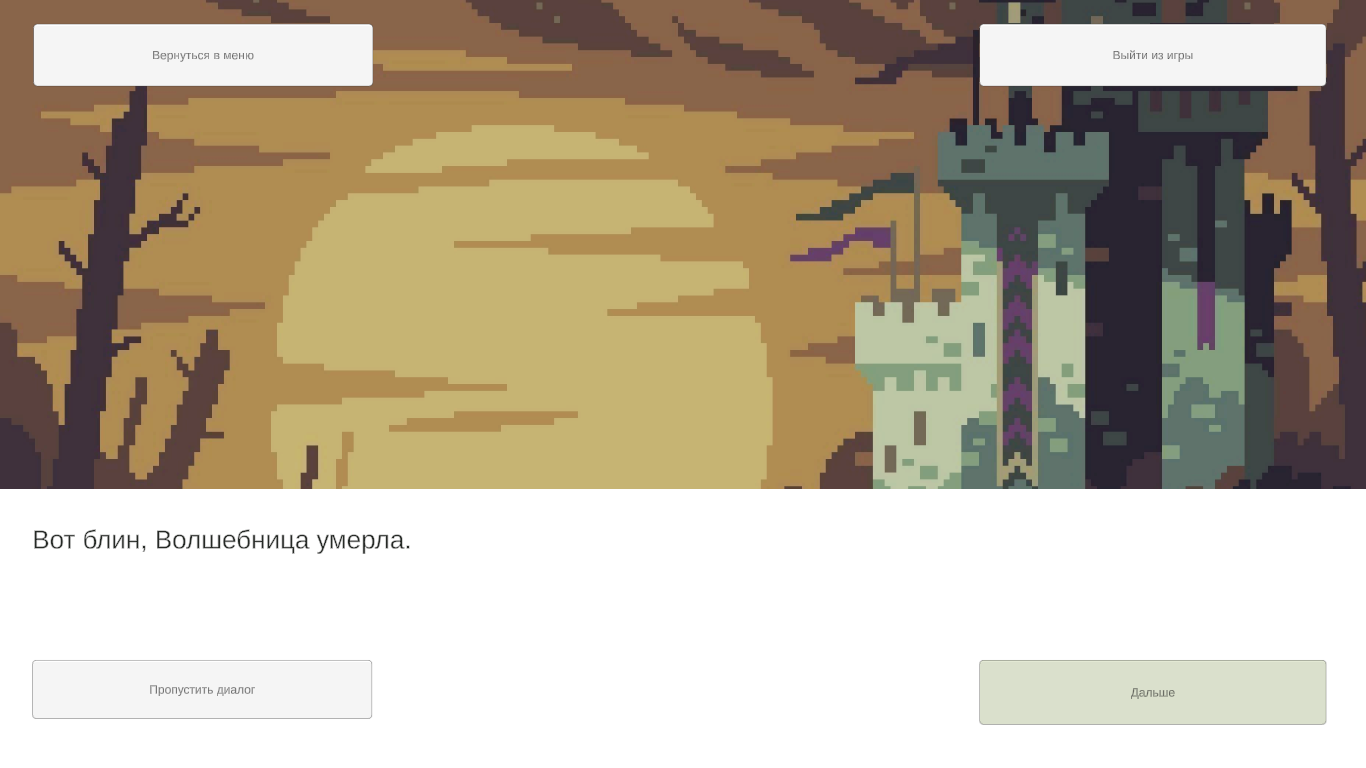


Рисунок Б21 – диалог при смерти персонажа



Рисунок Б22 – отображение рекордов

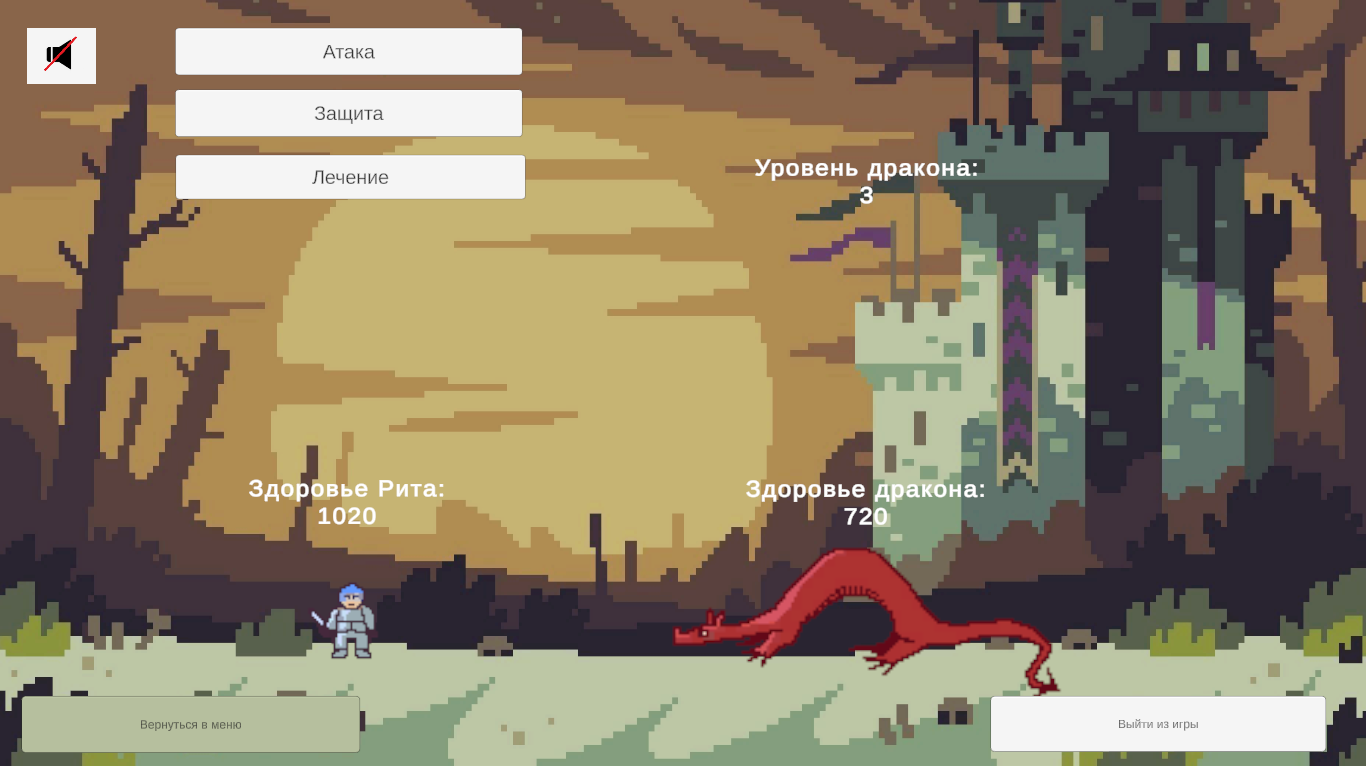


Рисунок Б23 – выключение звука

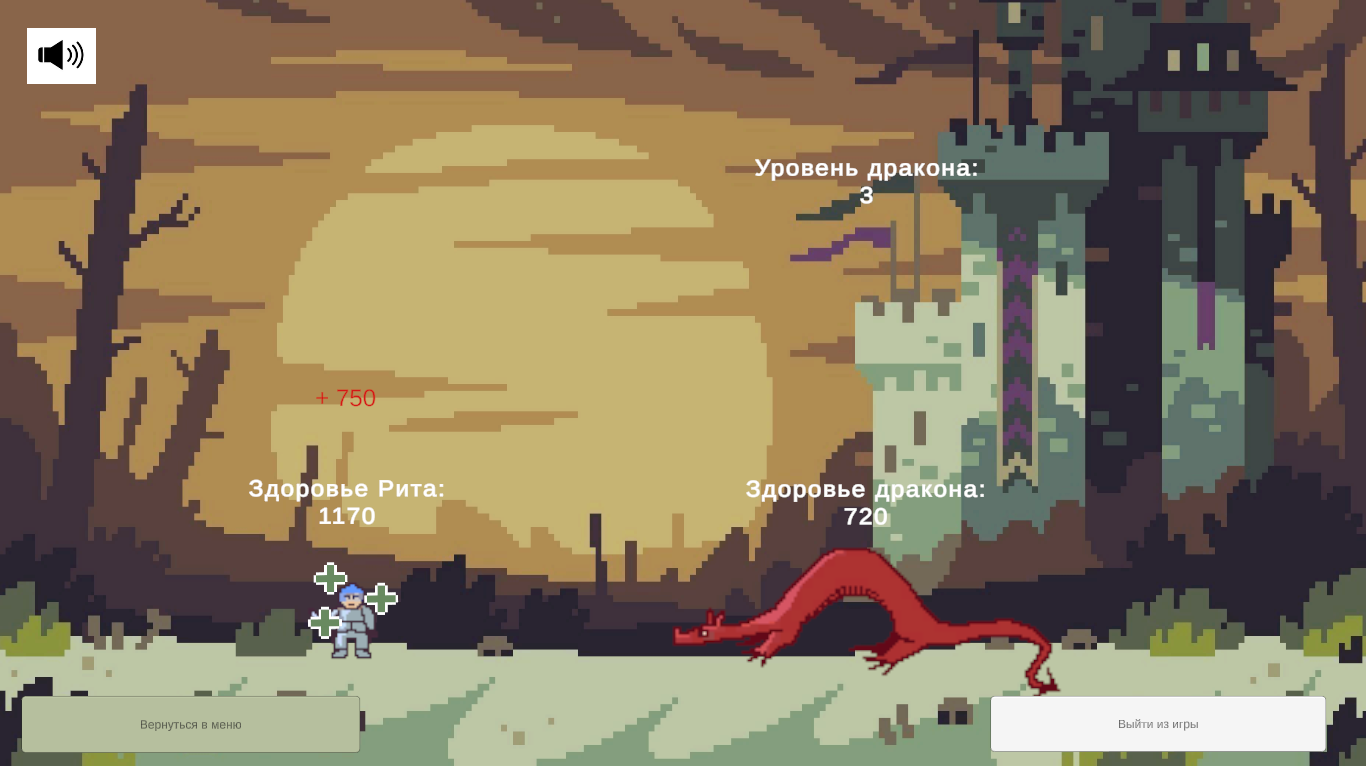


Рисунок Б24 – анимация лечения

# ПРИЛОЖЕНИЕ В

Текст программы

###### Assets/Script/ObjScript/

Game.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

using static Mob;

using static Player;

using static Dragon;

using UnityEngine.SceneManagement;

public class Game : MonoBehaviour

{

//инициализация переменных отсчета времени

float sec = 1f;

float timeLeft = 0;

//все кнопки

public GameObject AttackButton;

public GameObject DefenceButton;

public GameObject HealButton;

//все подписи

public TMP\_Text dragonHealthLabel;

public TMP\_Text dragonLevelLabel;

public TMP\_Text playerHealthLabel;

public TMP\_Text minusDragonLabel;

public TMP\_Text minusPlayerLabel;

public TMP\_Text StatusLabel;

public GameObject PlayerObj;

public GameObject DragonObj;

public GameObject HealObj;

Sprite neededSpr;

public Sprite kn1;

public Sprite mg1;

public Sprite dr1;

List<Sprite> sprites = new List<Sprite>();

public Sprite kn2;

public Sprite mg2;

public Sprite dr2;

List<Sprite> attackSprites = new List<Sprite>();

public Sprite death;

public Sprite dragon1;

public Sprite dragon2;

public Sprite dragon3;

SpriteRenderer playerSprRen;

SpriteRenderer dragonSprRen;

int C = 0;

public static int dragonLevel = 1;

static int baseDragonHealth = Dragon.baseDragonHealth;

Player player = new Player(100, 10, 1);

Dragon dragon = new Dragon(baseDragonHealth, 5, dragonLevel);

string name = PlayerDataScript.name;

//инициализация логина

public Login login = new Login();

public static string filename;

void Start()

{

HealObj.SetActive(false);

filename = Application.dataPath + "/records.bin";

sprites.Add(kn1);

sprites.Add(mg1);

sprites.Add(dr1);

attackSprites.Add(kn2);

attackSprites.Add(mg2);

attackSprites.Add(dr2);

playerSprRen = PlayerObj.GetComponent<SpriteRenderer>();

dragonSprRen = DragonObj.GetComponent<SpriteRenderer>();

CreatePlayer();

dragon = new Dragon(dragonLevel);

PrintOnLabels();

//инициализация логина

login = new Login(name);

}

void CreatePlayer()

{

player = new Player(100, 10, 1);

C = PlayerDataScript.myClass;

if (C > 0 && C <= 3)

{

player.ChangeClass(C);

neededSpr = sprites[C - 1];

playerSprRen.sprite = neededSpr;

}

player.Upgrade(dragonLevel);

}

public void Attack()

{

dragon.RecievedAttack(player);

UnableButtons();

neededSpr = attackSprites[C - 1];

playerSprRen.sprite = neededSpr;

minusDragonLabel.text = $"- {dragon.damageTaken}";

timeLeft = sec;

StartCoroutine(StartPlayersAttackTimer());

}

//защита

public void Defence()

{

if (!player.IsDefended)

{

player.Defend();

timeLeft = sec;

StartCoroutine(StartDragonsTimer1());

}

else

minusPlayerLabel.text = $"{name} уже защищена";

PrintOnLabels();

}

public void Heal()

{

//если уровень здоровья не выше максимума

if (player.health < player.maxHealth)

{

HealObj.SetActive(true);

player.Heal(dragonLevel);

//minusPlayerLabel.ForeColor = Color.Olive;

minusPlayerLabel.text = $"+ {player.healingDone}";

playerHealthLabel.text = player.health.ToString();

//отключение кнопки исполнения действия, чтобы нельзя было лечиться много раз подряд

UnableButtons();

timeLeft = sec / 2;

StartCoroutine(StartHealthTimer());

//атака дракона

timeLeft = sec;

StartCoroutine(StartDragonsTimer1());

}

else

minusPlayerLabel.text = $"здоровье {name} на максимуме";

PrintOnLabels();

}

IEnumerator StartPlayersAttackTimer()

{

while (timeLeft > 0)

{

timeLeft -= Time.deltaTime;

yield return null;

}

minusDragonLabel.text = "";

neededSpr = sprites[C - 1];

playerSprRen.sprite = neededSpr;

if (dragon.health <= 0)

{

minusDragonLabel.text = "";

//если это был дракон меньше 5 уровня, то

if (dragonLevel < 5)

{

//вывод сообщения

StatusLabel.text = $"Дракон умер. Вы переходите на следующий уровень {dragonLevel + 1}";

//повышение уровня дракона на 1

dragonLevel++;

//вывод диалога

if (dragonLevel == 2 || dragonLevel == 4)

SceneManager.LoadScene("DialogScene");

//повышение уровня рыцаря

player.Upgrade(dragonLevel);

//оборажение нового уровня дракона

dragonLevelLabel.text = dragonLevel.ToString();

//создание нового объекта класса Дракон с новым здоровьем и атакой

dragon = new Dragon(dragonLevel);

//отображение нового уровня здоровья рыцаря

playerHealthLabel.text = player.health.ToString();

//отображение нового уровня здоровья дракона

dragonHealthLabel.text = dragon.health.ToString();

//кнопка действия снова доступна

EnableButtons();

}

//если это был дракон 5 уровня, то

else

{

StatusLabel.text = "Дракон умер полностью\nПОЗДРАВЛЯЮ С ПОБЕДОЙ!!!";

HideAllLabels();

SceneManager.LoadScene("DialogScene");

neededSpr = dragon3;

dragonSprRen.sprite = neededSpr;

//запись рекорда

int rec = player.health;

if (rec > login.Record)

{

login.Record = rec;

login.AddNewRecord(filename);

login.CorrectResult(filename);

}

}

}

//если дракон не умер от атаки, то

else

{

//запуск таймера атаки дракона

timeLeft = sec;

StartCoroutine(StartDragonsTimer1());

}

PrintOnLabels();

}

//на 15 секунде дракон атакует

IEnumerator StartDragonsTimer1()

{

while (timeLeft > 0)

{

timeLeft -= Time.deltaTime;

yield return null;

}

minusDragonLabel.text = "";

dragon.DoAttack(player);

neededSpr = dragon2;

dragonSprRen.sprite = neededSpr;

//minusPlayerLabel.ForeColor = Color.Firebrick;

minusPlayerLabel.text = $"- {player.damageTaken}";

playerHealthLabel.text = player.health.ToString();

timeLeft = sec / 2;

StartCoroutine(StartDragonsTimer2());

PrintOnLabels();

}

IEnumerator StartDragonsTimer2()

{

while (timeLeft > 0)

{

timeLeft -= Time.deltaTime;

yield return null;

}

minusPlayerLabel.text = "";

//изображение меняется обратно

neededSpr = dragon1;

dragonSprRen.sprite = neededSpr;

//если рыцарь умер, то

if (player.health <= 0)

{

StatusLabel.text = $"{name} умер(ла).\nВы проиграли";

neededSpr = death;

playerSprRen.sprite = neededSpr;

dragonLevel = 6;

//показать диалог

SceneManager.LoadScene("DialogScene");

}

//если рыцарь не умер

else

//включить кнопку действия

EnableButtons();

PrintOnLabels();

}

IEnumerator StartHealthTimer()

{

while (timeLeft > 0)

{

timeLeft -= Time.deltaTime;

yield return null;

}

HealObj.SetActive(false);

minusPlayerLabel.text = "";

PrintOnLabels();

}

void PrintOnLabels()

{

dragonLevelLabel.text = $"Уровень дракона:\n{dragonLevel.ToString()}";

dragonHealthLabel.text = $"Здоровье дракона:\n{dragon.health.ToString()}";

playerHealthLabel.text = $"Здоровье {name}:\n{player.health.ToString()}";

}

void UnableButtons()

{

StatusLabel.text = "";

//отключение кнопки исполнения действия, чтобы нельзя было атаковать много раз подряд

AttackButton.SetActive(false);

DefenceButton.SetActive(false);

HealButton.SetActive(false);

}

void EnableButtons()

{

AttackButton.SetActive(true);

DefenceButton.SetActive(true);

HealButton.SetActive(true);

}

void HideAllLabels()

{

dragonHealthLabel.text = "";

dragonHealthLabel.text = "";

dragonLevelLabel.text = "";

playerHealthLabel.text = "";

minusDragonLabel.text = "";

minusPlayerLabel.text = "";

}

}

PlayerDataScript.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public static class PlayerDataScript

{

public static string name { get; set; }

public static int myClass { get; set; }

}

###### Assets/Script/ObjScript/Mob/

Mob.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Mob : MonoBehaviour

{

public int health { get; protected set; }

public int attack { get; protected set; }

public int defence { get; protected set; }

public bool IsDefended { get; protected set; }

public int damageTaken { get; protected set; }

public int healingDone { get; protected set; }

public GameObject player;

protected System.Random random = new System.Random();

public Mob()

{

health = 0;

attack = 0;

defence = 0;

IsDefended = false;

}

public Mob(int H, int A)

{

health = H;

attack = A;

}

public virtual void RecievedAttack(Player player)

{

damageTaken = player.attack \* random.Next(5, 8);

health -= damageTaken;

}

public virtual void DoAttack(Player player)

{

if (player.IsDefended)

{

player.IsDefended = false;

}

else

{

player.damageTaken = attack \* random.Next(5, 8);

player.health -= player.damageTaken;

}

}

}

Player.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public enum battleClass

{

Рыцарь = 1,

Волшебница,

Друид

}

public class Player : Mob

{

public battleClass playerClass { get; private set; }

public int maxHealth { get; private set; }

int healthCoef = 0;

public Player()

: base() { }

public Player(int H, int A, int C)

: base(H, A)

{

ChangeClass(C);

}

public void ChangeClass(int C)

{

playerClass = (battleClass)C;

switch (C)

{

case 1:

maxHealth = health = 200;

healthCoef = 2;

attack = 30; //150 - 240

IsDefended = true;

break;

case 2:

maxHealth = health = 100;

healthCoef = 1;

attack = 40; //200 - 320

IsDefended = false;

break;

case 3:

maxHealth = health = 300;

healthCoef = 3;

attack = 25; //125 - 200

IsDefended = false;

break;

}

}

public void Upgrade(int dragonLevel)

{

health = maxHealth = dragonLevel \* healthCoef \* 100;

attack += dragonLevel \* 10;

}

public void Heal(int dragonLevel)

{

healingDone = 25 \* healthCoef \* random.Next(3, 6) \* dragonLevel;

health += healingDone;

}

public void Defend()

{

IsDefended = true;

}

}

Dragon.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Dragon : Mob

{

int level;

public static int dragonLevel = 1;

//инициализация базового здоровья дракона

public static int baseDragonHealth = 400; //400 - 2000

public Dragon()

: base() { }

public Dragon(int H, int A, int L)

: base(H, A)

{

level = L;

health \*= level;

IsDefended = false;

}

public Dragon(int L)

: base(baseDragonHealth, 0)

{

level = L;

health = baseDragonHealth \* level;

attack = level \* 10;

IsDefended = level == 5;

}

public override void RecievedAttack(Player player)

{

base.RecievedAttack(player);

}

public override void DoAttack(Player player)

{

base.DoAttack(player);

}

}

###### Assets/Script/UIScript/

DontDestroyOnLoad.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class DontDestroy : MonoBehaviour

{

void Start()

{

DontDestroyOnLoad(this);

}

}

RecordsScript.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class RecordsScript : MonoBehaviour

{

//все подписи

public TMP\_Text Logins;

public TMP\_Text Records;

string filename = "records.bin";

// Start is called before the first frame update

void Start()

{

ShowRecords();

}

void ShowRecords()

{

Login.RecordToTable(filename, Logins, Records);

}

}

SceneGhangerScript.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class SceneGhangerScript : MonoBehaviour

{

public void ChangeScene(string sceneName)

{

SceneManager.LoadScene(sceneName);

}

public void Exit()

{

Application.Quit();

}

}

###### Assets/Script/UIScript/Dialog/

Dialog.cs:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class Dialog

{

public List<string> text { get; private set; }

public Dialog()

{

this.text = new List<string>();

}

public Dialog(List<string> text)

{

this.text = text;

}

public void ReplaceClassName(string className, string name)

{

for(int i = 0; i < text.Count; i++)

{

text[i] = text[i].Replace("className", className);

text[i] = text[i].Replace("name", name);

}

}

public string this[int i]

{

get => text[i];

}

}

public class Dialogs

{

public List<Dialog> All = new List<Dialog>();

public Dialogs()

{

//

List<string> text1 = new List<string>()

{

"className!",

"Мы так счастливы, что ты пришла на зов и спасешь наш замок от дракона.",

"За работу."

};

Dialog Begin = new Dialog(text1);

//

List<string> text2 = new List<string>()

{

"Ого, ты справилась.",

"Можно узнать имя нашей героини?",

"Вы:\nname.",

"Приятно познакомиться, name!",

"Но теперь дракон стал еще сильнее и злее, как же так?",

"Придется начинать все сначала."

};

Dialog FirstWin = new Dialog(text2);

//

List<string> text3 = new List<string>()

{

"Ваау, смотрите, name победила и этого дракона.",

"Осталось совсем чуть-чуть, поднажми, дорогая!"

};

Dialog ThirdWin = new Dialog(text3);

//

List<string> text4 = new List<string>()

{

"Дракон повержен!!! Мы свободны!!!",

"Спасибо, className name.",

"Можешь идти."

};

Dialog Victory = new Dialog(text4);

//

List<string> text5 = new List<string>()

{

"Вот блин, className умерла.",

"Каждый раз одно и то же."

};

Dialog Death = new Dialog(text5);

All = new List<Dialog>() { Begin, FirstWin, ThirdWin, Victory, Death };

}

public void ReplaceAllClassNames(string className, string name)

{

foreach(Dialog d in All)

{

d.ReplaceClassName(className, name);

}

}

public Dialog this[int i]

{

get => All[i];

}

//public void ShowDialogInGame(TMP\_Text DialogBox)

//{

//}

}

DialogScript.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

using UnityEngine.SceneManagement;

public class DialogScript : MonoBehaviour

{

public TMP\_Text DialogLabel;

public GameObject NextButton;

Dialogs AllDialogs = new Dialogs();

int dialogNumber = -1;

int lineNumber = 0;

int dragonLevel = Game.dragonLevel;

void Start()

{

string c = ((battleClass)PlayerDataScript.myClass).ToString();

AllDialogs.ReplaceAllClassNames(c, PlayerDataScript.name);

switch (dragonLevel)

{

case 1:

dialogNumber = 0;

break;

case 2:

dialogNumber = 1;

break;

case 4:

dialogNumber = 2;

break;

case 5:

dialogNumber = 3;

break;

case 6:

dialogNumber = 4;

break;

default:

dialogNumber = -1;

break;

}

if (dialogNumber >= 0)

DialogLabel.text = AllDialogs[dialogNumber][lineNumber];

}

public void NextLine()

{

lineNumber++;

if (lineNumber < AllDialogs[dialogNumber].text.Count)

DialogLabel.text = AllDialogs[dialogNumber][lineNumber];

else

NextButton.SetActive(false);

}

public void Skip()

{

if (dragonLevel < 5)

SceneManager.LoadScene("ActualGameScene");

else

SceneManager.LoadScene("StartScene");

}

}

###### Assets/Script/UIScript/Login/

Login.cs

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

[Serializable]

public class Login

{

public byte[] login { get; private set; }

public int record { get; private set; }

public static readonly int length = 20;

[NonSerialized]

static Encoding encoding = Encoding.GetEncoding(1251);

public Login() // конструктор по умолчанию

{

login = StringToByte("rita", length);

record = 0;

}

public Login(string login)

{

this.login = StringToByte(login, length);

this.record = 0;

}

public int Record { get => record; set => record = value; }

public string GetLogin => ByteArrayToString(login);

/// <summary>

/// преобразование из массива байт в строку

/// </summary>

/// <param name="ar">массив байт</param>

/// <returns>строка</returns>

private static string ByteArrayToString(byte[] b)

{

char[] buff = encoding.GetChars(b); // раскодирование массива байт

string result = "";

for (int i = 0; i < buff.Length; i++)

{

result += (char)buff[i]; // добавление каждого символа в строку

}

return result;

}

/// <summary>

/// преобразование из строки в массив байт

/// </summary>

/// <param name="s">строка переменной длины</param>

/// <param name=" lengthField ">фиксированная длина поля</param>

/// <returns>массив байт фиксированной длины</returns>

private static byte[] StringToByte(string s, int lengthField)

{

char[] ar = new char[lengthField];

for (int i = 0; i < lengthField; i++)

{

if (i < s.Length)

ar[i] = s[i];

// преобразование в символьный массив фиксированного размера

else

ar[i] = ' ';

// дополнение строки пробелами до нужного размера

}

byte[] buff = encoding.GetBytes(ar); // кодирование в массив байт

return buff;

}

/// <summary>

/// Метод добавления данных в конец файла

/// </summary>

/// <param name="filename">имя файла</param>

public void AddNewRecord(string filename)

{

FileStream fa = null;

try

{

fa = new FileStream(filename, FileMode.Append);

Debug.LogError($"fw name = {fa.Name}");

BinaryFormatter bw = new BinaryFormatter();

bw.Serialize(fa, this);

}

catch { }

finally { fa.Close(); } //закрытие потока чтения

}

/// <summary>

/// Метод корректировки отдельной записи (следует с байта num)

/// </summary>

/// <param name="filename">имя файла</param>

/// <param name="num">номер байта, позиция записи</param>

/// <returns>удалась ли корректировка (true/false)</returns>

public bool CorrectResult(string filename)

{

long num = FindPosition(filename);

bool result = false;

if (num >= 0)

{

FileStream fw = null;

try

{

fw = new FileStream(filename, FileMode.Open);

BinaryFormatter bf = new BinaryFormatter();

fw.Seek(num, SeekOrigin.Begin);

// установим указатель файла на запись с позицией num

bf.Serialize(fw, this);

// записываем в файл изменения

result = true;

}

catch { }

finally { fw.Close(); } //закрытие потока записи

}

return result;

}

private long FindPosition(string filename)

{

long position = -1;

long currentPosition = -1;

FileStream fs = null;

try

{

fs = new FileStream(filename, FileMode.OpenOrCreate);

BinaryFormatter bf = new BinaryFormatter();

Login l = new Login();

if (fs.Length > 0)

{

while (fs.Position < fs.Length)

{

position = fs.Position;

l = (Login)bf.Deserialize(fs);

if (l.login == login)

{

currentPosition = position;

break;

}

}

}

}

catch { }

finally { fs.Close(); } //закрытие потока чтения

return currentPosition;

}

/// <summary>

///

/// </summary>

/// <param name="filename"></param>

/// <param name="dg"></param>

public static void RecordToTable(string filename, TMP\_Text Logins, TMP\_Text Records)

{

List<Login> list = Result(filename);

for (int i = 0; i < list.Count; i++)

{

if (i > 10)

break;

string login = ByteArrayToString(list[i].login).TrimEnd();

int record = list[i].record;

Logins.text += login + '\n';

Records.text += record.ToString() + '\n';

}

}

/// <summary>

/// Метод добавления данных из файла в список

/// </summary>

/// <param name="filename">название файла</param>

/// <returns>список, содержащий логины</returns>

public static List<Login> Result(string filename)

{

List<Login> list = new List<Login>();

FileStream fs = null;

try

{

fs = new FileStream(filename, FileMode.OpenOrCreate);

BinaryFormatter bf = new BinaryFormatter();

Login l = new Login();

if (fs.Length > 0)

{

while ((l = (Login)bf.Deserialize(fs)) != null)

{

list.Add(l);

}

}

}

catch { }

finally { fs.Close(); } //закрытие потока записи

return list;

}

}

NameScript.cs:

using System;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

using UnityEngine.SceneManagement;

public class PlayerName : MonoBehaviour

{

public GameObject OkButton;

public TMP\_InputField NameField;

string playerName;

public TMP\_Dropdown ClassIndexDropdown;

int playerClass;

public void CheckOrSaveName(string sceneName)

{

try

{

playerName = NameField.text;

playerClass = ClassIndexDropdown.value;

}

catch (Exception ex)

{

Debug.LogError("My mistake: " + ex.Message);

}

finally

{

if (!string.IsNullOrWhiteSpace(playerName))

{

PlayerDataScript.name = playerName;

PlayerDataScript.myClass = playerClass + 1; //потому что исчисление классов начинается с 1

SceneManager.LoadScene(sceneName);

Game.dragonLevel = 1;

}

else

{

Debug.LogError("No.");

}

}

}

}

###### Assets/Script/UIScript/Music/

Music.cs:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

using TMPro;

public class Music : MonoBehaviour

{

bool MusicOn = true;

public GameObject MusicButton;

public GameObject MusicManager;

public Sprite MicOn;

public Sprite MicOff;

public void MusicOnAndOffButton()

{

if (MusicOn)

{

MusicButton.GetComponent<Button>().image.sprite = MicOff;

MusicManager.GetComponent<AudioSource>().Pause();

MusicOn = false;

}

else

{

MusicButton.GetComponent<Button>().image.sprite = MicOn;

MusicManager.GetComponent<AudioSource>().Play();

MusicOn = true;

}

}

}