## **Введение**

Темой дипломного проектирования является разработка игрового приложения «OnionFree».

Цель дипломного проектирования состоит в разработке игрового приложения жанра roguelite, при помощи которого пользователь может практически бесконечно проходить уровни получая разнообразный эмоциональный опыт.

В нынешнее время многие пользователи, любят играть в компьютерные игры, так как игра – это получение уникального опыта. Существует большое многообразие компьютерных игр, где нужно применять различные навыки для успешного прохождения игры.

Созданное приложение рассчитано для пользователей любого возраста, которым нравятся игры жанра roguelite.

Далее приведём краткое описание разделов пояснительной записки.

Первый раздел «Анализ задачи» содержит описание предметной области и круга задач, которые должны быть автоматизированы, построение диаграммы вариантов использования и модели данных. Также раздел содержит обоснование выбора среды разработки и требования к приложению.

Во втором разделе «Проектирование задачи» содержится описание организации данных и процессов. Также в этом разделе приводится описание внешнего пользовательского интерфейса, разграничение прав доступа. Описываются тесты на использование.

В разделе «Реализация» содержится описание реализации игрового приложения.

Четвертый раздел «Тестирование». В данном разделе описывается проведение тщательного тестирования программного продукта. Также здесь представлен отчет о результатах полного тестирования программного продукта.

В пятом разделе «Применение» описано назначение и условия применения программы, а также инсталляция, сообщения системы и выполнения программы.

Раздел «Охрана труда, энерго- и ресурсосбережение» содержит систему законодательных актов, социально-экономических, организационных, технических, гигиенических и лечебно-профилактических мероприятий и средств, обеспечивающих безопасность, сохранение здоровья и работоспособности человека в процессе труда.

Экономический раздел содержит технико-экономические расчеты, в которых устанавливаются затраты на разработку программного обеспечения и определяется их экономическая эффективность.

В «Заключении» предоставлена краткая формулировка задачи, проанализированы результаты проделанной работы, выполнено сравнение поставленной задачи и реализованного проекта.

В «Списке использованных источников» приведен список используемых при разработке источников с информацией.

В приложении «А» приведен листинг программы.

Графическая часть дипломного проекта представлена диаграммой вариантов использования, диаграммой классов и диаграммой деятельности.

## **1 Анализ задачи**

## **Описание предметной области**

Игра – это концентрация на процессе, игровое приложение — на результате. Игровое приложение строится на внешней мотивации: дети играют, чтобы получить виртуальные бонусы. Видеоигра строится на внутренней мотивации: дети играют не только ради наград, но и ради самой игры [1].

Наиболее популярные жанры игр:

* RPG – жанр компьютерных игр, основанный на элементах игрового процесса традиционных настольных ролевых игр. В ролевой игре игрок управляет одним или несколькими персонажами, каждый из которых описан набором численных характеристик, списком способностей и умений [2];
* Shooter – жанр компьютерных игр. На момент зарождения жанра за рубежом укрепилось слово «шутер», как вариант описания игрового процесса и перевод для слова shooter, в России и некоторых других странах СНГ жанр изначально был назван в среде любителей как «стрелялка» [3];
* Roguelite – Игра, которая использует большинство правил roguelike — но не все. Например, нет перманентной смерти, или время реальное вместо пошагового [4].

Представителями игр жанра roguelite являются такие игры как Enter The Gungeon, Soul Knight, The Binding of Isaac. В этих представителях система построения уровней слишком сложна и не удобна, в разрабатываемом проекте система построения уровней будет проще и удобнее в использовании. Так же данные представители являются сложными для понимания не знакомому с данным жанром игр пользователю, проект планируется сделать проще для понимания.

## **1.2 Инструменты разработки**

Visual Studio 2019 – это мощное средство разработчика, которое можно использовать для единого завершения всего цикла разработки. Это комплексная интегрированная среда разработки, которую можно использовать для написания, редактирования, отладки и сборки кода, а затем для развертывания приложения. Помимо редактирования и отладки кода, Visual Studio включает компиляторы, средства завершения кода, систему управления версиями, расширения и многие другие функции для улучшения каждого этапа процесса разработки программного обеспечения.

Adobe Photoshop CS6 – многофункциональный растровый графический редактор, разрабатываемый и распространяемый компанией Adobe Systems. В основном работает с растровыми изображениями, однако имеет некоторые векторные инструменты. Продукт является лидером рынка в области коммерческих средств редактирования растровых изображений и наиболее известной программой разработчика. Adobe Photoshop пподдерживается на рынке уже много лет и выпустил достаточное количество продуктов для популярных операционных систем, систем обработки изображений и так далее. Adobe Photoshop CS6 отличается своей высокой производительностью и малыми системными требованиями, а также добавлением множества новых инструментов для создания и обработки изображений.

Unity – кроссплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие. Данная среда разработки имеет наименьший порог вхождения и многочисленные инструменты разработки, улучшающие взаимодействие с разробатываемым программным продуктом. К тому же игровой движок Unity является кроссплатформенным.

**1.3 Диаграмма вариантов использования**

Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

Основные элементы диаграммы – участник (actor) и прецедент (вариант). Участник — это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Участником может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности. Графически участник изображается «человечком», как на рисунке 1.



Рисунок 1 – Графическое обозначения актёра

Вариант использования – описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому участником результату. Прецедент представляет поведение сущности, описывая взаимодействие между участниками и системой. Прецедент не показывает, как достигается некоторый результат, а только что именно выполняется. Прецеденты обозначаются очень простым образом – в виде эллипса, внутри которого указано его название, как на рисунке 2.



Рисунок 2 – Графическое обозначение вариантов использования

Отношение ассоциации является главным понятием языка UML и используется при построении всех графических моделей. Оно служит для обозначения роли актера в отдельном варианте использования. На диаграмме отношение ассоциации обозначается сплошной линией между актером и вариантом использования, как на рисунке 3.

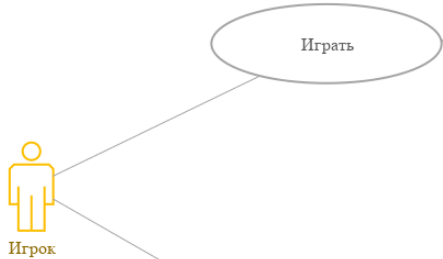


Рисунок 3 – Графическое обозначение отношения ассоциации

Для отображения взаимосвязи экземпляра отдельного варианта использования с более общим вариантом, используется отношение расширения, данная линия помечается ключевым словом «extended», как на рисунке 4, обозначаемое направленной пунктирной линией со стрелкой от исходного варианта.

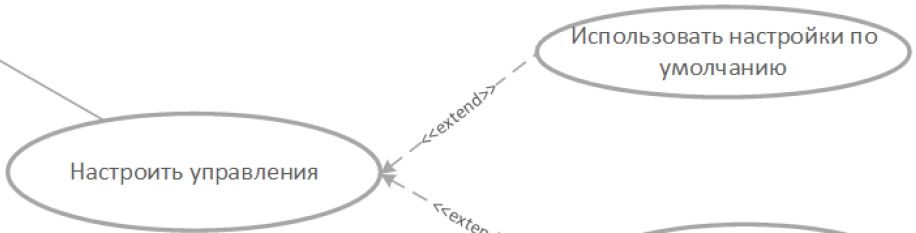


Рисунок 4 – Графическое обозначение отношения расширения

Отношение включения между двумя вариантами использования указывает, что поведение одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. Графически данное отношение обозначается пунктирной линией со стрелкой, направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом «include», как показано на рис. 5.



Рисунок 5 – Графическое обозначения отношения включения

Определяя для выбранного актера варианты использования и устанавливая отношения между вариантами использования, получим полную диаграмму вариантов использования, которая представлена в графической части дипломного проекта на листе 1.

На диаграмме вариантов использования указаны 1 участник и 2 варианта использование: «Играть» и «Настроить управление».

Вариант использования «Играть» имеет 2 отношения включения:

* управление персонажем;
* взаимодействие с объектами.

Вариант использования «Настроить управление» имеет 2 отношения расширения:

* использовать настройки по умолчанию;
* изменить настройки по умолчанию.

**1.4 Требования к программе**

Проект представляет собой двумерную компьютерную игру в жанре roguelite.

Программный продукт должен отвечать следующим требованиям:

* игра должна обладать простым и понятным интерфейсом;
* в игре должно быть простое управление;
* случайная генерация уровней;
* полная доступность всех игровых действий с самого начала игры;
* самостоятельное изучение мира;
* враги движутся слегка медленнее игрока и в направлении игрока;
* на уровне должны иметься собираемые объекты, увеличивающие здоровье игрока;
* должна быть реализована прогрессия игрока – получение очков за уничтожение врагов и повышение характеристик за счёт этих очков.

**2 Проектирование задачи**

**2.1 Организация данных**

Средством организации данных, требующих длительного хранения, будут являться игровые спрайты - графические объекты в компьютерной графике. Чаще всего — растровые изображения, которые можно отобразить на экране [5]. Для удобства использования файлы спрайтов будут храниться в формате png в папке «Sprites».

Спрайты будут делиться на 3 вида:

* cпрайты персонажей;
* cпрайты окружения;
* cпрайты предметов.

Спрайты персонажей представляют собой отдельные части врагов и персонажа, таких как голова, руки, ноги и т.д., которые позже будут соединены вместе для более простого анимирования.

Спрайты окружения – это спрайты стен, дверей и портала.

Спрайты предметов – это спрайты ключа, аптечки, оружия, снаряда и т.д.

**2.2 Процессы**

Ранее были описаны функциональные требования, которые необходимо реализовать в игровом приложении. На основании этого описания выделим основные процессы, из которых будет состоять разрабатываемый проект. Процессы, которые необходимо реализовать в программе приведены в таблице 1.

Таблица 1 – Процессы

|  |  |
| --- | --- |
| Процесс | Описание процесса |
| Играть | Передвигаться |
| Атаковать |
| Взаимодействовать с объектами |
| Проходить уровни |
| Пауза | Вернуться в игру |
| Начать игру сначала |
| Возвращение в главное меню |
| Улучшение | Улучшить здоровье |
| Улучшить защиту |
| Улучшить урон |
| Перейти к главному меню |
| Настроить управление | Изменить управление движением |
| Вернуть настройки по умолчанию |

**2.3 Описание внешнего пользовательского интерфейса**

Основными требованиями к оформлению интерфейса являются:

* преобладание зелёного, коричневого, белого и чёрного цветов, а также их оттенков;
* наличие одного шрифта (для оформления всего текста игрового приложения);
* персонажи и окружение должны быть выполнены в векторном стиле.

Для упрощённой разработки интерфейса были разработаны макеты для разных частей программного продукта, таких как: главное меню, представлено на рисунке 6, интерфейс персонажа, представлено на рисунке 7, меню паузы, представлено на рисунке 8, и экран улучшения, представлено на рисунке 9.

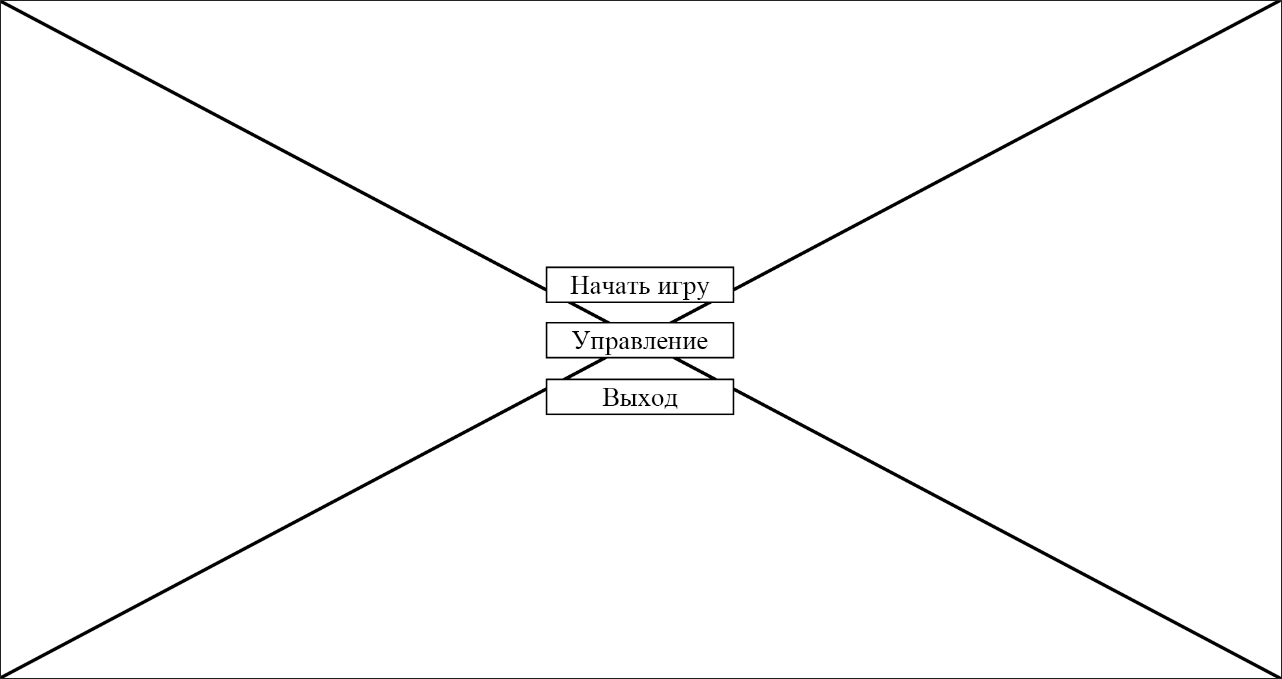


Рисунок 6 – Макет интерфейса главного меню

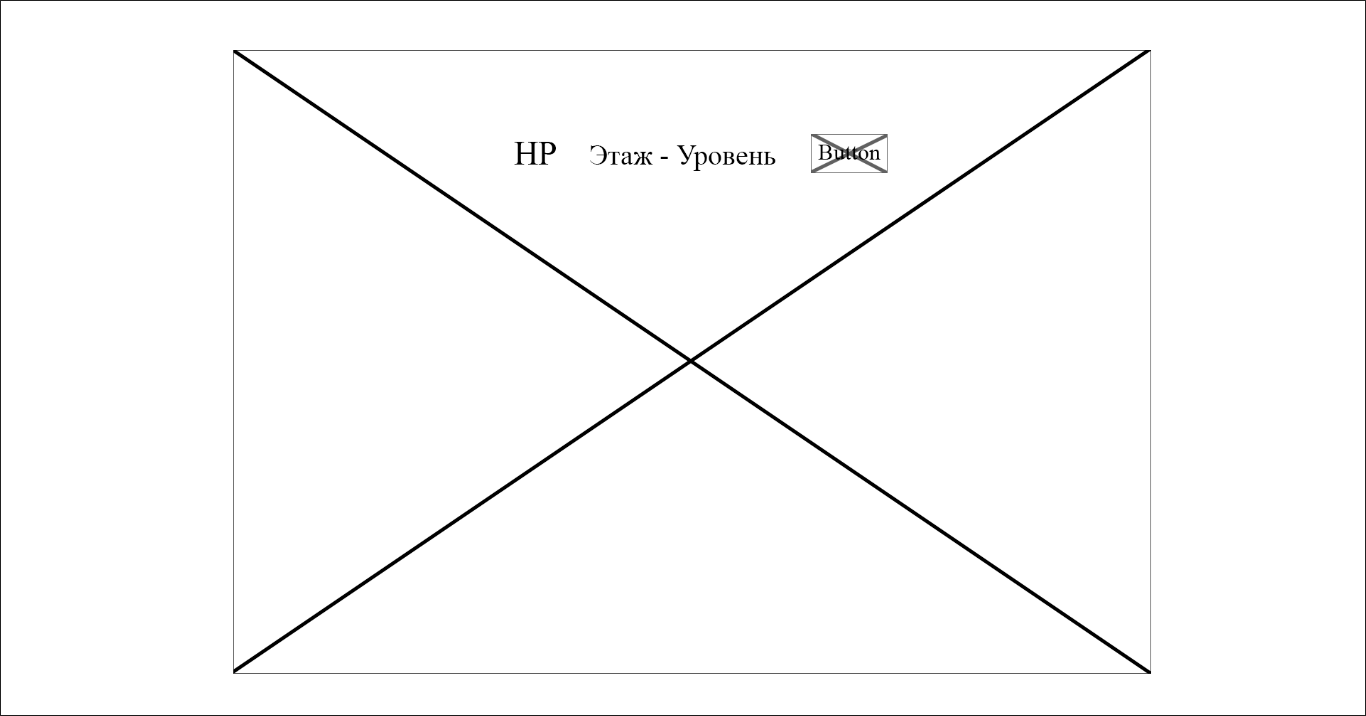


Рисунок 7 – Макет интерфейса персонажа

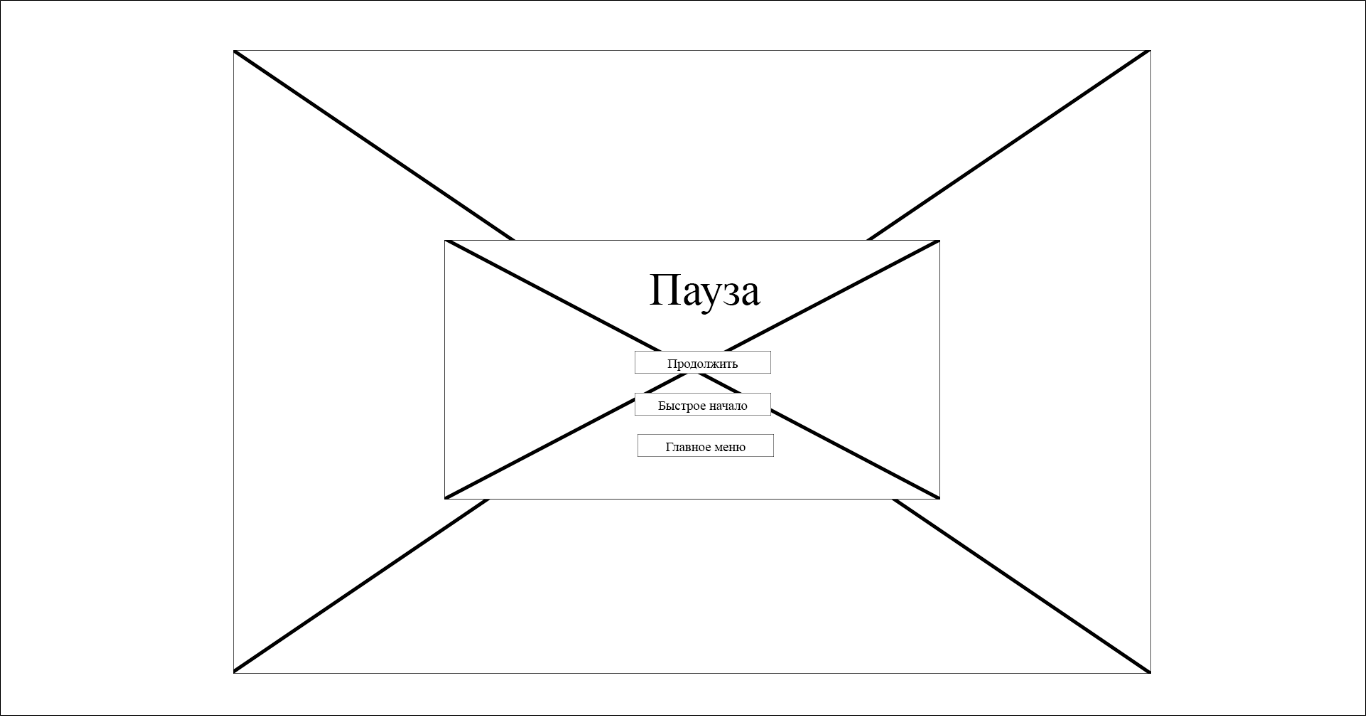


Рисунок 8 – Макет меню паузы

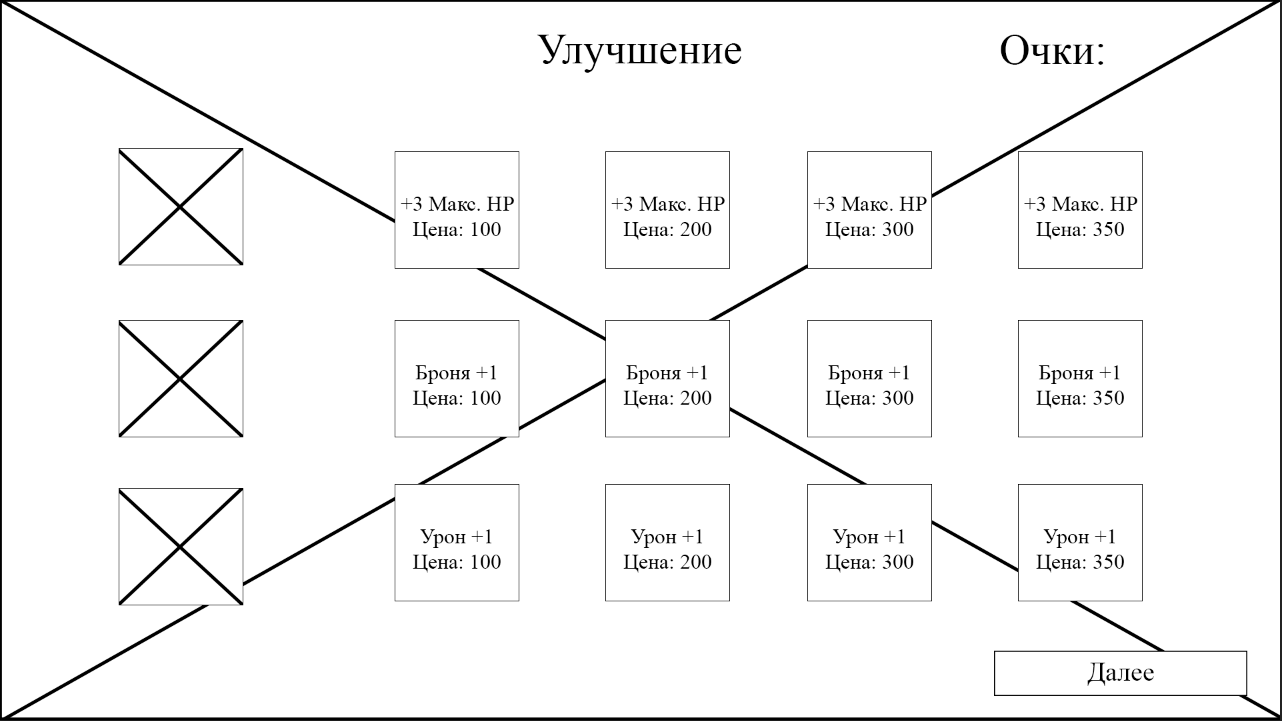


Рисунок 9 – Макет экрана улучшения

**2.4 Описание системы помощи**

В качестве системы помощи пользователю на фоне в главной комнате этажа указано управление персонажем по умолчанию.

Так же в качестве системы помощи добавлено краткое описание к улучшениям, в котором описано что делает улучшение и какова его стоимость.

Вся система помощи разработана с помощью встроенных средств Unity.

**2.5 Тестовый сценарий**

Тестовый сценарий представляет собой отдельный полноценный уровень, на котором присутствуют все объекты, и враги, с которыми может взаимодействовать пользователь.

Во время разработки, программа должна быть отлажена и протестирована. Для проведения тестирования тестировщику необходимо включить тестовый сценарий.

Первое что необходимо протестировать – передвижение. Для этого тестировщик должен поочерёдно нажать клавиши «W», «A», «S» и «D» на клавиатуре, в результате чего персонаж будет передвигаться вверх, влево, вниз и вправо соответственно и вернётся в исходную позицию.

Далее необходимо проверить прицеливание. Тестировщик должен провести курсором мыши вокруг персонажа. В результате – оружие в руках персонажа будет отслеживать положение курсора и указывать в его сторону. Одновременно можно провести тестирование атаки, для чего необходимо навести курсор мыши на небольшое удаление от персонажа и нажать «Левую кнопку мыши» из-за чего оружие в руках персонажа выпустит снаряд в сторону, указанную курсором.

Не маловажным элементом для прохождения игры является поднятие предметов, таких как ключ или аптечка. Для проверки этого элемента тестировшик должен передвинуть персонажа на позицию ключа, после чего ключ исчезнет, а вверху экрана появится кнопка с изображением ключа.

Далее идёт проверка открытия дверей, для чего нужно подойти в плотную к двери с замком и навести курсор мыши на кнопку с изображением ключа после чего нажать «Левую кнопку мыши». В результате этих действий кнопка с изображением ключа и дверь с замком пропадут.

Для дальнейшего тестирования нужно нажать клавишу «Esc» на клавиатуре, что откроет меню паузы и остановит время в игре.

Первое что необходимо проверить в меню паузы – это возможность продолжить игру, для чего необходимо при открытом меню паузы выполнить одно из следующих действий:

* нажать кнопку «Продолжить» в меню паузы;
* нажать клавишу «Esc» на клавиатуре.

В результате обоих действий меню паузы закроется, а время в игре запустится.

Следующий элемент — это возможность быстрого перезапуска игры. Для его тестирования этого элемента нужно нажать кнопку «Быстрое начало» в меню паузы, в результате меню паузы закроется, а игра должна перезапуститься с самого начала.

Следующим элементом, который нужно проверить, является возможность выхода в главное меню из меню паузы. Для этого нужно открыть меню паузы с помощью клавиши «Esc» на клавиатуре и нажать кнопку «Главное меню» после чего откроется окно улучшения персонажа, в котором нужно нажать кнопку «Далее». В результате откроется главное меню игры.

1. **Реализация**
   1. **Функции: логическая и физическая организация**

Пользователь может управлять персонажем, которым необходимо добраться до конца уровня попутно побеждая врагов. Все функции хранятся в файлах скриптов формата cs в папке «Scripts». Перечень скриптов и за что они отвечают приведён в таблице 3.

Таблица 2 – Скрипты, функции и их назначение

|  |  |  |
| --- | --- | --- |
| Скрипт | Функции | Назначение |
| 1 | 2 | 3 |
| AddRoom.cs | Awake | Собирает информацию о всех существующих комнатах |
| Start | Добавляет новую комнату в список комнат |
| OnTriggerEnter2D | Заполняет комнату врагами и аптечками |
| CheckEnemies | Проверяет наличие врагов в комнате |
| DestroyWalls | Открывает все двери в комнате |
| OnTriggerStay2D | Дополнительно открывает двери, если те не открылись |
| OnTriggerExit2D | Останавливает искусственный интеллект врагов, если игрок вышел из комнаты |
| Bullet.cs | Start | Создаёт снаряд |
| Update | Обрабатывает столкновение снаряда |
| DestroyBullet | Создаёт эффект на месте столкновения снаряда |
| ChangeRoom.cs | Start | Ищет камеру на сцене |
| OnTriggerEnter2D | Переносит камеру и персонажа в следующую комнату |
| DeadEnd.cs | Update | Открывает меню смерти персонажа при его смерти |
| LoadMenu | Запускает главное меню |
| Restart | Запускает игру с самого начала |
| Enemy.cs | Start | Собирает информацию о сцене |
| Update | Управляет врагом |
| TakeDamage | Обрабатывает получение урона |
| OnTriggerStay2D | Включает и выключает анимацию атаки |
| OnEnemyAttack | Обрабатывает урон по игроку |
| Gun.cs | Start | Находит игрока на сцене |
| Update | Управляет положением оружия |
| Shoot | Создаёт снаряд |
| MainMenu.cs | PlayGame | Запускает игру |
| ExitGame | Выключает игру |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| options.cs | Start | Загружает клавиши управления из памяти |
| Update | Активирует или деактивирует кнопку сброса настроек |
| resetOptions | Сбрасывает управление до настроек по умолчанию |
| PauseMenu.cs | Update | Открывает или закрывает меню паузы |
| Resume | Закрывает меню паузы |
| Restart | Запускает игру с самого начала |
| LoadMenu | Возвращает пользователя в главное меню |
| Pause | Открывает меню паузы |
| Player.cs | | Start | Создаёт персонажа |
| Update | Управляет персонажем |
| FixedUpdate | Подстраховывает метод «Update» |
| Flip | Заменяет спрайт персонажа |
| ChangeHealth | Обрабатывает изменение здоровья персонажа |
| OnTriggerEnter2D | Обрабатывает заступ на триггер |
| onKeyButtonDown | Открывает дверь с замком |
| OnTriggerStay2D | Обрабатывает нахождение в триггере |
| RoomSpawner.cs | | Start | Запускает создание комнаты |
| Spawn | Создаёт комнату |
| OnTriggerStay2D | Удаляет точку создания комнаты |
| RoomVariants.cs | | Start | Запускает функцию «RandomSpawner» |
| RandomSpawner | Создаёт комнату босса и создаёт ключ в случайной комнате |
| Wall.cs | | OnTriggerEnter2D | Заменяет двери комнат на стены, если двери ведут в стену соседней комнаты |
| OnTriggerStay2D |

* 1. **Функции и элементы управления**

Часть функций связаны с элементами управления пользовательского интерфейса, с которыми может взаимодействовать пользователь.

Элементы управления и соответствующие им функции представлены в таблице 3.

Таблица 3 – Функции и элементы управления

|  |  |
| --- | --- |
| Элемент управления | Функция |
| 1 | 2 |
| Кнопка «Главное Меню» в меню смерти персонажа | LoadMenu |
| Кнопка «Быстрое Начало» в меню смерти персонажа | Restart |
| Кнопка «Начать игру» в главном меню | PlayGame |

Продолжение таблицы 3

|  |  |
| --- | --- |
| 1 | 2 |
| Кнопка «Выход» в главном меню  Клавиша «Esc» на клавиатуре | ExitGame |
| Кнопка «Продолжить» в меню паузы | Resume |
| Кнопка «Быстрое Начало» в меню паузы | Restart |
| Кнопка «Главное Меню» в меню паузы | LoadMenu |
| Клавиша «Esc» на клавиатуре | Pause |
| Кнопка с иконкой ключа в интерфейсе персонажа | onKeyButtonDown |
| Кнопка «По Умолчанию» в настройках управления | resetOptions |

* 1. **Спецификация**

Под спецификацией проекта подразумевается описание папок и файлов, которые получились при компиляции игры.

При создании проекта в Unity, он создает собственные папки и файлы. После получения готового проекта файлы игры помещены в папку «onionFree». Все скомпилированные файлы описаны в таблице 4.

Таблица 4 – Спецификация проекта.

|  |  |  |
| --- | --- | --- |
| Название папки | Название файла | Описание файла |
| 1 | 2 | 3 |
| OnionFree | OnionFree.exe | Файл запуска игры |
| UnityCrashHandler64.exe | Файл инструмента, который предназначенный для отслеживания и обработки ошибок, возникающих в процессе работы приложений, созданных с использованием игрового движка Unity. |
| UnityPlayer.dll | Файл UnityPlayer.dll предназначен для создания 3D игр или анимированных персонажей, и также для правильной работы игр созданных на движке Unity. |
| OnionFree\ MonoBleeding  \ EmbedRuntime | mono-2.0-bdwgc.dll | файл, используемый в дистрибутиве Visual C++ Redistributable для Visual Studio 2015. |
| MonoPosixHelper.dll | Файл с процедурами и функциями драйвера для Windows. |
| OnionFree\ OnionFree\_Data\ Resources | unity default resources, unity\_builtin\_extra | Файлы, которые содержат все элементы использованные для создания игры. Сюда входят спрайты, анимации и другие. |

Продолжение таблицы 4

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| OnionFree\ OnionFree \_Data | app.info | Файл с информацией о приложении. |
| boot.config | Файл с настройками запуска игры. |
| data.unity3d | Двоичный файл, используемый игровым движком Unity для хранения игровых данных, ассетов и ресурсов. |
| RuntimeInitializeOnLoads.json, ScriptingAssemblies.json | Файлы с параметрами запуска библиотек для правильного функционирования игры |

**4 Тестирование**

Тестирование является одним из важных этапов при создании и внедрении программного продукта. Исходя из поведения программы при определенном наборе тестов, нельзя сделать однозначных выводов о том, что программа будет исправно работать в любой момент использования. Опираясь на данный набор тестов, можно говорить о некоторой степени уверенности в правильности работы программного продукта.

Таким образом, основная часть ошибок и недоработок была исправлена на этапе реализации проекта. После завершения этапа реализации, было проведено тщательное тестирование.

Программный продукт был протестирован разработчиком – Колпаковым Даниилом Андреевичем. Тестирование выполнялось строго в соответствии с пунктом «2.5 Тестовый сценарий».

Тестирование проводилось на ноутбуке с операционной системой Windows 10.

Расписание проведения и время, затраченное на тестирование, описано в таблице 5.

Таблица 5 – Расписание проведения тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО | Дата | Описание | Длительность, час |
| Колпаков Даниил Андреевич | 05.06.2023 | Выполнение тестового сценария | 2 |
| Колпаков Даниил Андреевич | 06.06.2023 | Анализ выполнения тестового сценария | 3 |
| Колпаков Даниил Андреевич | 07.06.2023 | Повторное выполнение тестового сценария | 1 |

Количество выявленных по результатам тестирования ошибок приводится в таблице 6.

Таблица 6 – Статистика по выявленным ошибкам

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Статус | Количество ошибок | Важность | | | |
| критическая | высокая | средняя | минимальная |
| Исправлено | 10 | 2 | 0 | 6 | 2 |
| Проверено | 10 | 2 | 0 | 6 | 2 |
| Открыто заново | 0 | 0 | 0 | 0 | 0 |
| Найдено | 2 | 0 | 0 | 2 | 0 |
| Отклонено | 0 | 0 | 0 | 0 | 0 |

Таким образом при проведении тестирования, были выявлены следующие ошибки:

* комнаты создавались друг в друге;
* двери не заменялись на стены при их нахождении перед стеной.

Проведя анализ вышеуказанных ошибок, было принято решение оставить их, так как они не влияют на игровой процесс.

Опираясь на вышеуказанное, можно сделать вывод о стабильности программного продукта и его основного функционала. Все найденные ошибки были устранены и верифицированы. Соответственно можно сказать, что тестирование прошло успешно и программный продукт характеризуется как реализованный и готовый к работе.

**5 Применение**

**5.1 Назначение и условия применения программы**

Игровое приложение «OnionFree» предназначено для пользователей, которые хотят хорошо провести своё свободное время.

Программный продукт прост и удобен в применении, для работы с ним не требуется высокого навыка пользование компьютером.

Функционирование программного продукта тестировалось в следующей операционной системе (ОС): Microsoft Windows 10.

Игровое приложение «OnionFree» было реализовано на языке программирования высокого уровня C#.

Минимальные требования к программным и аппаратным средствам для нормального функционирования программного продукта:

- процессор Intel i3;

- объем ОЗУ 1 ГБ;

- видеосистема: 1024\*768 точек с глубиной цвета 16bit;

- операционная система Windows 10.

**5.2 Инсталляция**

Для того, чтобы установить программу, необходимо запустить файл Setup.exe. Появится окно установки игрового-развлекательного приложения OnionFree, как на рисунке 10.

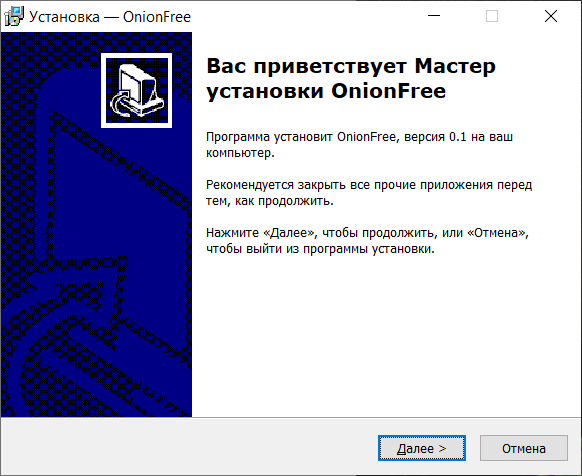


Рисунок 10 – Установка игрового приложения OnionFree

После нажатия на кнопку «Далее» появляется окно выбора каталога установки, как на рисунке 11. Нажав на кнопку «обзор», вы можете поменять место для сохранения файла.

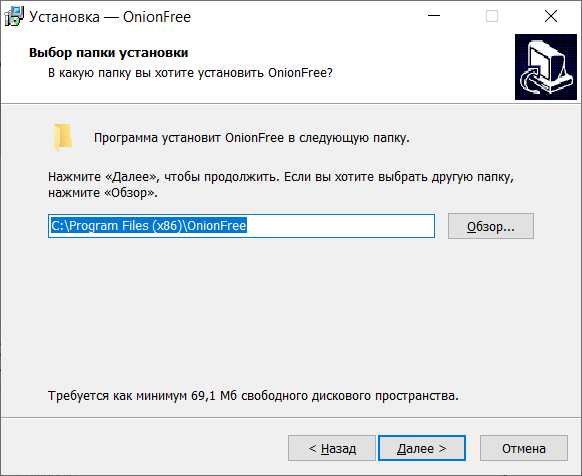


Рисунок 11 – Выбор каталога установки

Выбрав всё необходимое для установки программного продукта, вам будет предложено нажать на кнопку «установить». После её нажатия пойдет загрузка файлов и приложение будет готово к работе.

**5.3 Сообщения системы**

Из-за того, что есть возможность остановки установки программного продукта пользователем или из-за отключения питания, что приведёт к отсутствию некоторых важных файлов, было предусмотрено окно системной ошибки, в котором указано какой файл отсутствует и возможное решение проблемы. Пример окна системной ошибки при отсутствии файла «UnityPlayer.dll» приведён на рисунке 12.

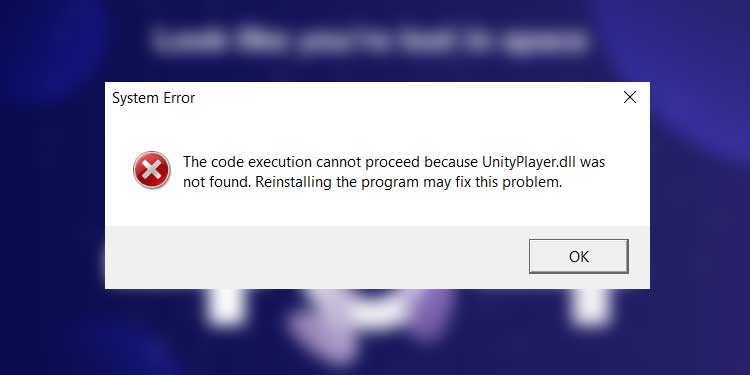


Рисунок 12 - Окно системной ошибки при отсутствии файла «UnityPlayer.dll»

**5.4 Выполнения программы**

Игровое приложение «OnionFree» можно запустить различными способами.

Первым из них является запуск с помощью ярлыка на рабочем столе. Необходимо дважды кликнуть левой кнопкой мышки на ярлыке, как на рисунке 13.



Рисунок 13 – Ярлык приложения OnionFree

Вторым способом является запуск через строку поиска Windows, в которую нужно ввести «OnionFree».

Третьим способ является запуск из каталога, в который устанавливалось приложение (По умолчанию C:\Program Files (x86)\OnionFree).

После запуска программы на экране появляется стандартная заставка, которую подставляет среда разработки Unity в программные продукты, разработанные с её помощью, как на рисунке 14.



Рисунок 14 – Стандартная заставка Unity

После заставки появляется главное меню, как на рисунке 15, откуда пользователь может начать играть, просмотреть или изменить управление, или выйти из игры.

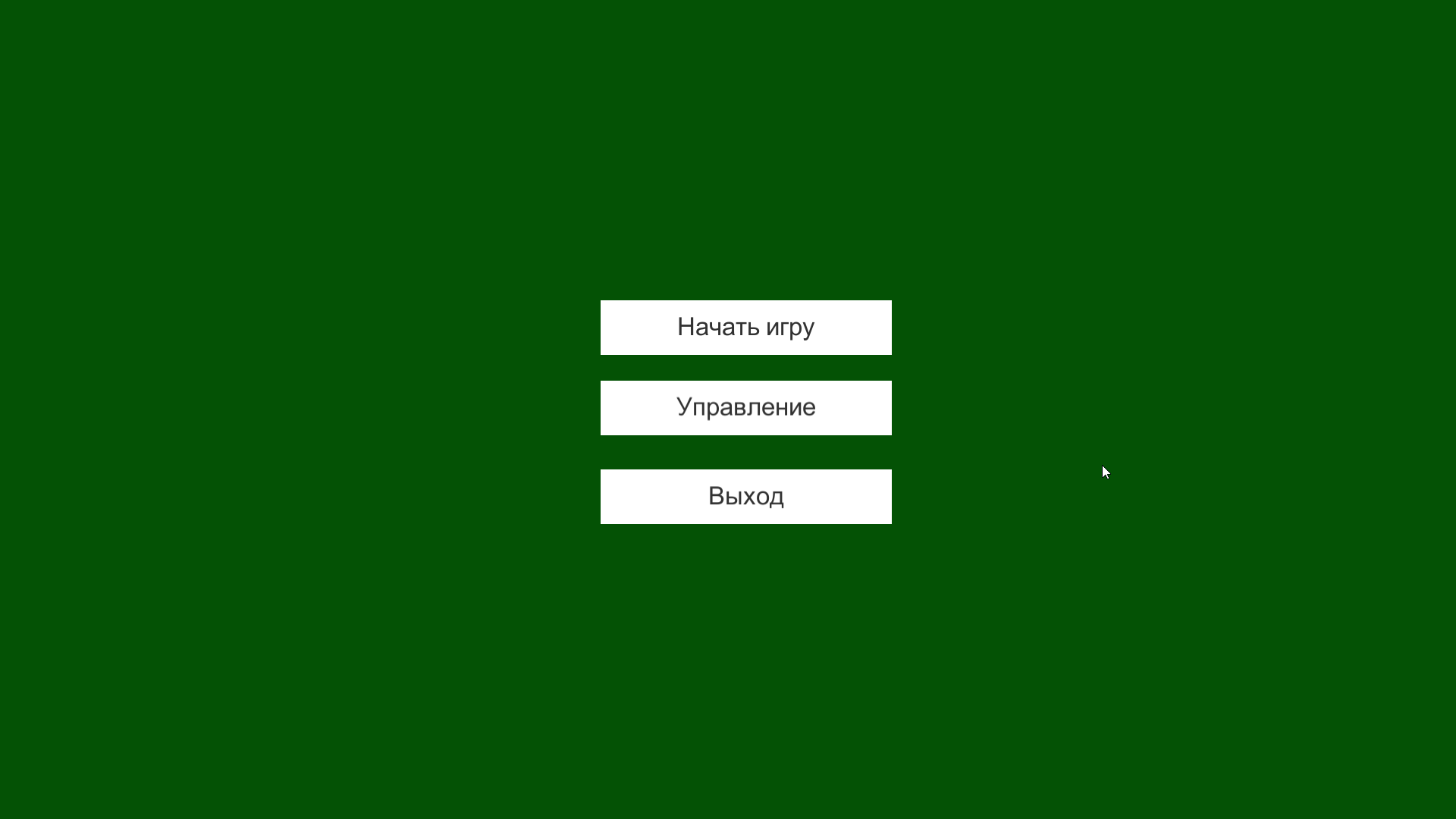


Рисунок 15 – Главное меню

При нажатии на кнопку «Начать игру» пользователя отправит на сцену с игрой, как на рисунке 16.

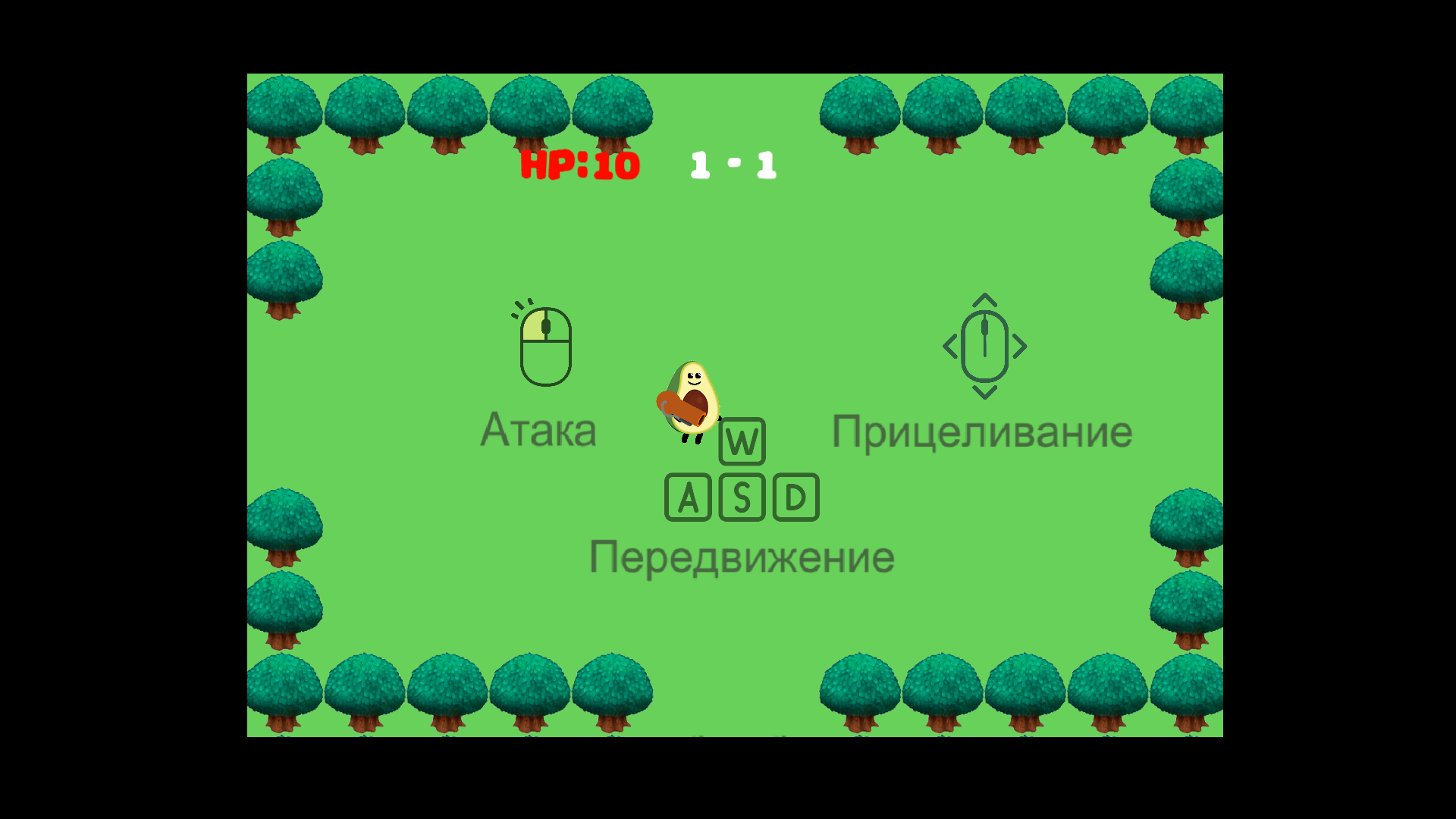


Рисунок 16 – Игра

Во время игры у пользователя есть возможность поставить игру на паузу, для чего нужно нажать на клавишу «Esc» на клавиатуре и тогда откроется меню паузы как на рисунке 17.



Рисунок 17 – Меню паузы

При смерти персонажа или после нажатия кнопки «Главное меню» в меню паузы, пользователю откроется окно улучшения, как на рисунке 18, в котором он может улучшить характеристики персонажа. Для того чтобы перейти в главное меню необходимо нажать кнопку «Далее», как на рисунке 19.



Рисунок 18 – Окно улучшения



Рисунок 19 – Кнопка «Далее»

Для получения доступа к настройкам управления нужно нажать на кнопку «Управление» и появится окно настроек управления, как на рисунке 20.

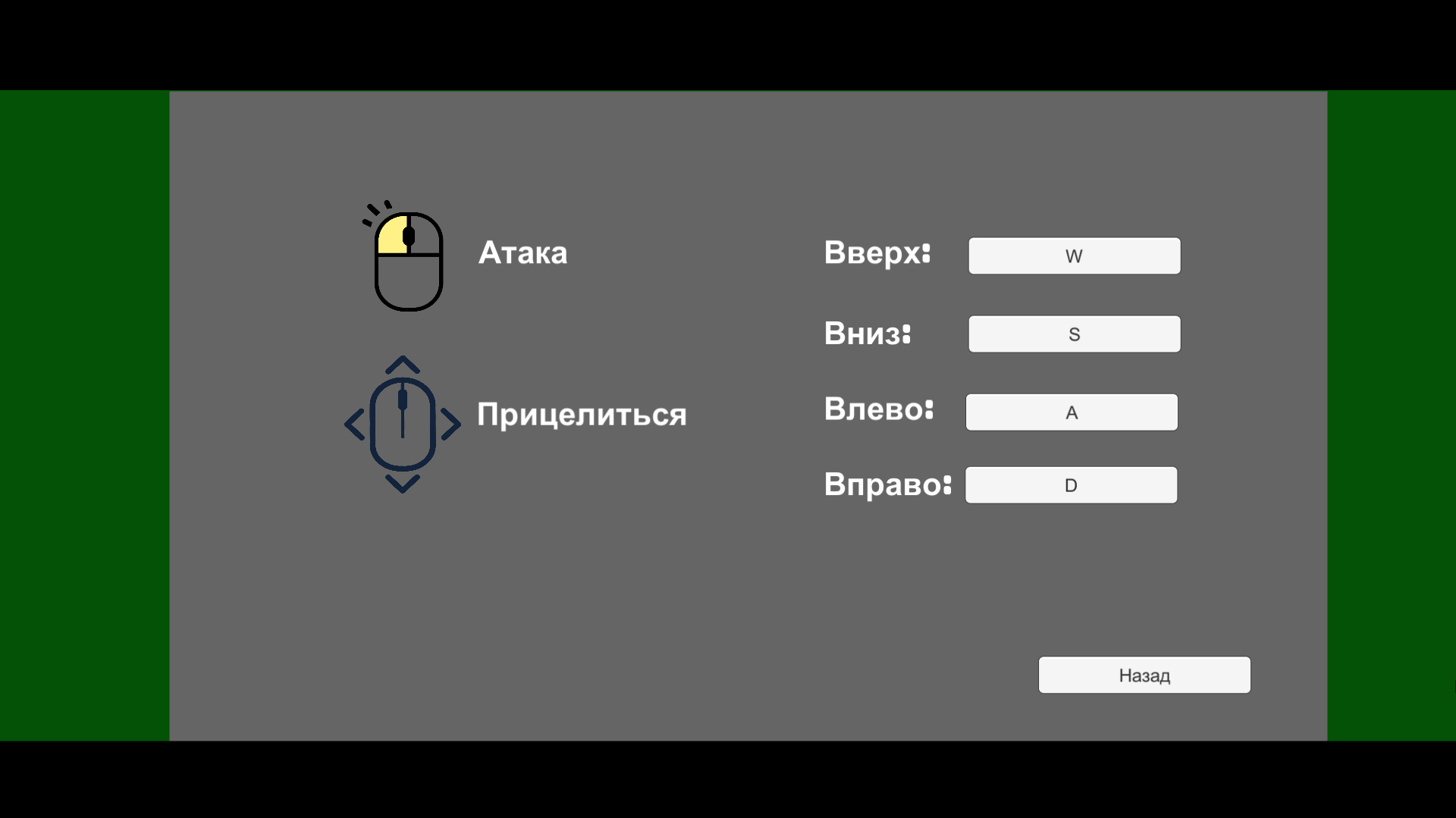


Рисунок 20 – Управление

Для выхода из приложения нужно нажать «Выход» в главном меню, как на рисунке 21.



Рисунок 21 – Кнопка «Выход»

**6 Охрана труда, энерго- и ресурсосбережение**

**6.1 Правовые, нормативные, социально-экономические и организационные вопросы охраны труда**

Организационно-правовую основу охраны труда составляет Конституция Республики Беларусь, законы Республики Беларусь, декреты и указы Президента Республики Беларусь.

Основополагающим документом по охране труда в республике является Конституция Республики Беларусь (статьи 41, 45, 46). Она гарантирует права граждан на здоровые и безопасные условия труда, право на отдых, охрану здоровья и право на благоприятную окружающую среду.

Статья 41 гарантирует гражданам Республики Беларусь право на труд, т.е. право на выбор профессии, рода занятий и работы в соответствии с призванием, способностями, образованием, профессиональной подготовкой и с учетом общественных потребностей, а также на здоровые и безопасные условия труда.

Статья 45 гарантирует гражданам Республики Беларусь право на охрану здоровья, включая бесплатное лечение в государственных учреждениях здравоохранения, а статья 46 – право на благоприятную окружающую среду и на возмещения вреда, причиненного нарушением этого права [6].

Трудовой кодекс Республики Беларусь определяет основные обязанности, права работников и нанимателей, гарантии прав работников, ответственность нанимателей и работников; предусматривает систему государственного и общественного надзора и контроля за соблюдением законодательства об охране труда; регламентирует деятельность службы охраны труда. Статья 222 Трудового кодекса Республики Беларусь устанавливает право работника на охрану труда [7].

Существует также целый ряд нормативных документов, регулирующих вопросы охраны труда, такие как, декреты и законы.

Декрет Президента Республики Беларусь от 30 июля 2003 г. № 18 «Об обязательном страховании от несчастных случаев на производстве и профессиональных заболеваний» в рамках общих вопросов страхования граждан предусматривает вопросы страхования их также от несчастных случаев на производстве и профзаболеваний [8].

Закон Республики Беларусь «О пожарной безопасности» от 15.06.1993 г. устанавливает государственный надзор за обеспечением пожарной безопасности министерствами, государственными комитетами, концернами, предприятиями, учреждениями, организациями независимо от форм собственности, а также гражданами. Закон определяет правовую основу и принципы организации пожарной безопасности [9].

Закон Республики Беларусь «О санитарно-эпидемическом благополучии населения» от 23.11.1993 г. №2583-ХИ направлен на предупреждение воздействия неблагоприятных факторов среды обитания на здоровье населения, устанавливает государственный санитарный надзор за соблюдением санитарных норм и гигиенических нормативов [10].

Порядок проведения инструктажей по охране труда установлен в ГОСТ 12.0.004 «Организация обучения рабочих безопасности труда». По характеру и времени проведения инструктажи по охране труда подразделяют на вводный, первичный на рабочем месте, повторный, внеплановый, целевой.

Вводный инструктаж по безопасности труда проводят:

* со всеми вновь принимаемыми на работу независимо от их образования, стажа работы по данной профессии или должности;
* с временными работниками;
* командированными, учащимися и студентами.

Вводный инструктаж на предприятии проводит инженер по охране труда или лицо, на которое возложены эти обязанности.

Первичный инструктаж на рабочем месте проводят:

* со всеми вновь принятыми на предприятие, переводимыми из одного подразделения в другое;
* с работниками, выполняющими новую для них работу, командированными, временными работниками;
* со строителями, выполняющими строительно-монтажные работы на территории действующего предприятия.

Повторный инструктаж проходят все рабочие не реже одного раза в полугодие.

Внеплановый инструктаж проводят:

* при введении в действие новых или переработанных стандартов, правил, инструкций по охране труда, а также изменений к ним;
* при изменении технологического процесса, замене или модернизации оборудования, приспособлений и инструмента, исходного сырья, материалов и других факторов, влияющих на безопасность труда;
* при нарушении работающими требований безопасности труда, которые могут привести или привели к травме, аварии, взрыву или пожару, отравлению;
* по требованию органов надзора;
* при перерывах в работе – для работ, к которым предъявляют дополнительные (повышенные) требования безопасности труда более чем на 30 календарных дней, а для остальных работ – 60 дней.

Целевой инструктаж проводят при выполнении разовых работ, не связанных с прямыми обязанностями по специальности (погрузка, выгрузка, уборка территории, и т.п.); ликвидации последствий аварий; производстве работ, на которые оформляется наряд-допуск, разрешение и другие документы; проведении экскурсии на предприятии, организации массовых мероприятий с учащимися (экскурсии, спортивные соревнования и др.).

Первичный инструктаж на рабочем месте, повторный, внеплановый и целевой проводит непосредственный руководитель работ. Они регистрируются в журнале регистрации инструктажа по охране труда [11].

**6.2 Эргономические основы безопасности труда**

**6.2.1 Организация рабочего места программиста**

При работе за ПК необходимо соблюдать требования СанПиН. Санитарные нормы и правила устанавливают требования к видеодисплейным терминалам (ВДТ), электронно-вычислительным машинам (ЭВМ), персональным электронно-вычислительным машинам (ПЭВМ), в том числе к портативным (нетбук, ноутбук и другое) и периферийным устройствам (принтеры, сканеры, клавиатуры, модемы внешние, электрические компьютерные сетевые устройства, внешние устройства хранения информации, блоки бесперебойного питания и другое), используемым на производстве.

Согласно СанПиН 9-131 РБ 2000 Санитарные правила и нормы «Гигиенические требования к видеодисплейным терминалам, электронно-вычислительным машинам и организации работы», помещения в которых расположены ЭВМ и ПЭВМ, необходимо проветривать согласно составленному графику, для поддержания микроклимата. В следствии несоблюдения профилактических мер по обеспечению надлежащего микроклимата, могут возникать сбои в работе техники.

Запрещается выполнение основной работы с использованием ВДТ, ЭВМ и ПЭВМ на постоянных рабочих местах без естественного освещения, если это не обусловлено технологическим процессом. Естественное освещение на рабочих местах с ВДТ, ЭВМ и ПЭВМ должно осуществляться через световые проемы, ориентированные преимущественно на север, северо-восток, восток, запад или северо-запад и обеспечивать коэффициент естественной освещенности не ниже 1,5%. Оконные проемы должны быть оборудованы регулируемыми устройствами типа жалюзи, занавесей, внешних козырьков и др. Площадь одного рабочего места для пользователей ВДТ, ЭВМ и ПЭВМ на базе электронно-лучевой трубки (далее – ЭЛТ) должна составлять не менее 6 м2.

Площадь одного рабочего места должна составлять не менее 6 м2, объем не менее 20 м3, высота 3 м. при следующих условиях:

* отсутствие на рабочем месте периферийных устройств (принтер, сканер и другое);
* продолжительность работы должна составлять не более 4 часов в день;
* площадь одного рабочего места для пользователей ВДТ, ЭВМ и ПЭВМ на базе плоских дискретных экранов (жидкокристаллические, плазменные и другое) должна составлять не менее 4,5 м2.

Важным моментом являются и такие требования как:

* Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.
* Конструкция рабочего стула (кресла) должна обеспечивать: ширину и глубину поверхности сиденья не менее 400 мм; поверхность сиденья с закругленным передним краем; регулировку высоты поверхности сиденья в пределах 400–550 мм и угла наклона вперед до 15 градусов и назад до 5 градусов; высоту опорной поверхности спинки 300±20 мм, ширину не менее 380 мм и радиус кривизны горизонтальной плоскости – 400 мм; угол наклона спинки в вертикальной плоскости в пределах 0±30 градусов; регулировку расстояния спинки от переднего края сиденья в пределах 260–400 мм; стационарные или съемные подлокотники длиной не менее 250 мм и шириной – 50–70 мм; регулировку подлокотников по высоте над сиденьем в пределах 230±30 мм и внутреннего расстояния между подлокотниками в пределах 350–500 мм.

Для внутренней отделки интерьера помещений, где расположены ВДТ, ЭВМ и ПЭВМ, должны использоваться диффузно отражающие материалы с коэффициентом отражения для потолка – 0,7…0,8; для стен – 0,5…0,6; для пола – 0,3…0,5.

Помещения должны быть оборудованы системами вентиляции, отопления или же эффективной приточно-вытяжной вентиляцией [12].

**6.2.2 Электробезопасность**

Электробезопасность – система организационных мероприятий и технических средств, предотвращающих вредное и опасное воздействие на работающих от электрического тока, электрической дуги, электромагнитного поля и статического электричества.

Методами защиты является:

* проектирование;
* снижение напряжения прикосновения;
* заземление;
* электрическое разделение сетей.

Во избежание повреждения изоляции проводов и возникновения коротких замыканий не разрешается:

* вешать что-либо на провода;
* закрашивать и белить шнуры и провода;
* закладывать провода и шнуры за газовые и водопроводные трубы, за батареи отопительной системы;
* выдергивать штепсельную вилку из розетки за шнур, усилие должно быть приложено к корпусу вилки.

Проходя через организм, электрический ток вызывает термическое, электролитическое и биологическое воздействие. Термическое действие тока вызывает ожоги отдельных участков тела, нагрев кровеносных сосудов, нервов, крови и т.п.

Программисты должны выполнять следующие требования по электробезопасности согласно Типовой инструкции по охране труда при использовании в работе офисного оборудования, утвержденной постановлением Министерства труда и социальной защиты Республики Беларусь 14.04.2021 №25.

Перед началом работы необходимо убедиться в исправности и целостности питающих и соединительных кабелей, разъемов и штепсельных соединений, защитного заземления (зануления).

Для исключения поражения электрическим током запрещается:

* часто включать и выключать компьютер без необходимости;
* прикасаться к экрану и к тыльной стороне блоков компьютера;
* работать на средствах вычислительной техники и периферийном оборудовании мокрыми руками;
* работать на средствах вычислительной техники и периферийном оборудовании, имеющих нарушения целостности корпуса, нарушения изоляции проводов, неисправную индикацию включения питания, с признаками электрического напряжения на корпусе;
* класть на средства вычислительной техники и периферийном оборудовании посторонние предметы.

При повреждении оборудования, кабелей, проводов, неисправности заземления, появлении запаха гари, возникновении необычного шума и других неисправностях следует немедленно отключить электропитание оборудования и сообщить о случившемся непосредственному руководителю или иному уполномоченному должностному лицу нанимателя.

**6.3 Соблюдение требований производственной санитарии**

Работающие с ПЭВМ могут подвергаться воздействию различных опасных и вредных производственных факторов: физических (повышенные уровни: электромагнитного, рентгеновского, ультрафиолетового и инфракрасного излучения; статического электричества; запыленности воздуха рабочей зоны; повышенное или пониженное содержание аэроионов в воздухе рабочей зоны и др.); химических (содержание в воздухе рабочей зоны оксида углерода, озона, аммиака, фенола, формальдегида и полихлорированных фенилов); психофизиологических (напряжение зрения, внимания; длительное статическое напряжение; монотонность труда; нерациональная организация рабочего места; эмоциональные перегрузки).

Техника безопасности – система организационных мероприятий, технических средств и методов, предотвращающих воздействие на работающих опасных производственных факторов.

Производственная санитария – это система организационных мероприятий и технических средств, предотвращающих или уменьшающих воздействие на работающих вредных производственных факторов.

**Типовая инструкция по охране труда при использовании в работе офисного оборудования** ПЭВМ с 14 апреля 2021 г. №25, возможно воздействие на работающих следующих вредных и (или) опасных производственных факторов:

* повышенный уровень электромагнитных излучений; повышенный уровень ионизирующих излучений; повышенный уровень статического электричества; повышенная напряженность электростатического поля; повышенная или пониженная ионизация воздуха; повышенная яркость света; прямая и отраженная блёскость;
* повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека;
* статические перегрузки костно-мышечного аппарата и динамические локальные перегрузки мышц кистей рук;
* перенапряжение зрительного анализатора;
* умственное перенапряжение;
* эмоциональные перегрузки;
* монотонность труда.

Работающие, с учетом воздействующих на них вредных и (или) опасных производственных факторов, обеспечиваются в соответствии с законодательством средствами индивидуальной защиты [13].

Основной принцип нормирования микроклимата – создание оптимальных условий для теплообмена тела человека с окружающей средой. Оптимальные и допустимые микроклиматические условия для указанной категории работ приведены в таблице 7 в соответствии с СанНП и ГН МЗ РБ №33 от 30.04.2013.

Таблица 7 – Допустимые микроклиматические условия

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Период года | Норма | Температура, °С | Относительная влажность, % | Скорость движения воздуха |
| Теплый | Оптимальная | 20-25 | 40-60 | 0,3 |
| Допустимая | 25 | 75 | 0,5 |
| Холодный | Оптимальная | 17-21 | 40-60 | 0,2 |
| Допустимая | 18-21 | 75 | не более 0,3 |

Примечание [14]

Вентиляционные системы соответствуют требованиям ГОСТ 12.4.021-75 «Система стандартов безопасности труда. Системы вентиляционные. Общие требования». Приточная вентиляция обеспечивает подачу в производственные помещения чистого воздуха. Она применяется в производственных помещениях со значительными тепловыделениями и малой концентрацией вредных веществ [15].

**6.4 Пожарная безопасность**

Пожарная безопасность – состояние защищённости личности, имущества, общества и государства от пожаров.

Типовая инструкция по охране труда при использовании в работе офисного оборудования от 20 мая 2021 год является правовым документом по выполнению требований пожарной безопасности.

Помещения должны содержаться в чистоте. Горючие отходы, мусор необходимо ежедневно удалять в контейнеры на специально выделенные площадки. Коридоры, лестничные клетки, двери эвакуационных выходов, подходы к средствам тушения всегда должны быть свободны и ничем не загромождены.

К числу организационных мероприятий по обеспечению пожарной безопасности относятся обучение рабочих и служащих правилам пожарной безопасности, разработка и внедрение норм и правил пожарной безопасности, инструкций о порядке работы с пожароопасными веществами и материалами, организация пожарной охраны объекта.

Основным документом, регулирующим деятельность по обеспечению пожарной безопасности, является Закон Республики Беларусь «О пожарной безопасности» (с изменениями и дополнениями №2/2332 от 01.01.2016 г.), который определяет правовую основу и принципы организации системы пожарной безопасности и государственного пожарного надзора.

Помещения должны содержаться в чистоте. Горючие отходы, мусор необходимо ежедневно удалять в контейнеры на специально выделенные площадки. Коридоры, лестничные клетки, двери эвакуационных выходов, подходы к средствам тушения всегда должны быть свободны и ничем не загромождены.

Запрещается:

* хранить и применять горючие жидкости, взрывчатые вещества, баллоны с газами и др.;
* использовать электронагревательные приборы;
* эксплуатировать провода электроприборов с поврежденной изоляцией;
* пользоваться поврежденными розетками, рубильниками, вилками и прочим электрооборудованием;
* обертывать (накрывать) светильники, бытовые приборы бумагой, тканью и другими горючими материалами;
* оставлять без наблюдения включенную в сеть радиоэлектронную аппаратуру, ПЭВМ, оргтехнику, бытовую технику;
* пользоваться неисправной или незаземленной аппаратурой;
* нарушать правила эксплуатации ПЭВМ и оргтехники, а также инструкции по работе на ПЭВМ и средствах оргтехники;
* включать в сетевые фильтры, блоки бесперебойного питания и специализированные розетки.

**6.5 Охрана окружающей среды, ресурсо- и энергосбережение**

Согласно Закону Республики Беларусь от 26 ноября 1992 г. № 1982-XІІ, охрана окружающей среды (природоохранная деятельность) – деятельность государственных органов, общественных объединений, иных юридических лиц и граждан, направленная на сохранение и восстановление природной среды, рациональное (устойчивое) использование природных ресурсов и их воспроизводство.

Ресурсосбережение – это совокупность мер по бережливому и эффективному использованию фактов производства (капитала, земли, труда).

Энергосбережение – комплекс мер, направленных на эффективное использование топливно-энергетических ресурсов.

Во время работы с ПЭВМ, расходуется достаточно большое количество электроэнергии. Все напрямую зависит от мощности блока питания и то, какое напряжение он создает в сети. При использовании большого количества сетевых фильтров одновременно, происходит большая нагрузка на сеть в следствии чего может произойти короткое замыкание. Одним из опасных факторов для окружающей среды при работе за ПЭВМ является, тот факт, что при большой нагрузке на него, при попадании пыли в корпус на нагретые элементы, такие, как ЦП, видеокарта и другие, в воздух могут попадать вредные вещества.

Стоит отметить, что на сегодняшний день, компьютеры стали более ресурсоемкими, в ходе чего происходит снижение на цепи питания, потребления энергии, а также снижению уровня шума.

Профилактические меры при работе за ПЭВМ:

* своевременное отключение электроприборов, по окончанию работы;
* поддержание рабочего места в чистоте и своевременная чистка ПЭВМ;
* проветривание, ионизация и поддержание нужного уровня влажности.

**7 Экономический раздел**

Широкое применение вычислительной техники (ВТ) требует постоянного обновления и совершенствования программного обеспечения. Выбор эффективных проектов ПО требует их экономической оценки и расчета экономического эффекта. Экономический эффект у разработчика выступает в виде роста чистой прибыли (чистого дохода, ЧД) или чистого дисконтированного дохода (ЧДД), научно-технической организации от реализации ПО. Экономический эффект зависит от объема затрат на разработку проекта, уровня цены на разработанный программный продукт и объема продаж.

Создание по-настоящему востребованного, удобного и надежного программного продукта невозможно без определенных затрат.

**7.1 Определение объема и трудоемкости ПО**

Базой для расчета плановой сметы затрат на разработку игрового приложения «OnionFree» является объем ПО.

Оценивание объема программного продукта связано с выбором наиболее подходящей единицы измерения размера продукта. В зарубежной практике получили распространение следующие единицы измерения:

* количество строк исходного кода (LinеsОfСоdе, LОС);
* функциональные точки (Fuпсtiоnроint, FР);
* точки свойств (Рrореrtуроint, РР);
* количество «пузырьков» на диаграмме потока данных (Dаtеflоw  
  diаgrаm, DFD);
* количество сущностей на диаграмме сущностей (Епtitуrеlаtiоn-  
  shiрdiаgrаm, ЕRD);
* количество «квадратиков», соответствующих процес-  
  су/контролю ( РSРЕС/СSРЕС);
* количество различных элементов в составе управленческой спе-  
  цификации (element);
* объем документации (количество строк, quantitylines);
* количество объектов, атрибутов и служб на объектной диаграм-  
  ме (subjects, attributes, services).

Несмотря на довольно значительный перечень видов единиц измерения объема ПО, наиболее широкое распространение получили лишь первые три. Причем функциональные точки и точки свойств до сих пор используются только в сочетании с количеством строк исходного кода (LОС). Все остальные виды единиц измерения применяются в основном при разработке специализированных проектов.

В данном дипломном проекте в качестве единиц измерения объема  
ПО используется строка исходного кода (LОС). Преимущества использования строк кода как единиц измерения заключаются в том, что эти единицы:

* отражают сущность труда программистов;
* широко распространены и могут легко адаптироваться;
* позволяют выполнять сопоставление размеров ПО и производительности в различных группах разработчиков;
* непосредственно связаны с конечным продуктом;
* могут использоваться для оценки работ до завершения проекта;
* позволяют автоматизировать сбор данных о количестве LОС от  
  начала до конца проекта;
* дают возможность учитывать мнение разработчика об объеме  
  ПО на основе количества написанных строк кода.

Строка исходного кода (LОС) является универсальной метрикой, так  
как может применяться при создании любых программных продуктов. При  
подсчете LОС следует придерживаться следующих рекомендаций:

* учитывать «строку исходного кода» как одну, если в ней содер-  
  жится лишь один оператор (если в одной строке содержатся два  
  выполняемых оператора, разделяемых точкой с запятой, то нуж-  
  но считать две строки, а если один выполняемый оператор раз-  
  бит на две «физические» строки, то он будет учитываться как  
  один оператор);
* учитывать все имеющиеся выполняемые операторы, поддерживаемые данным продуктом;
* определение данных учитывать лишь один раз;
* не учитывать строки, содержащие комментарии;
* не учитывать отладочный код или другой временный код (пробное ПО, средства тестирования, инструменты разработки и прототипирования и другие инструментальные средства);
* учитывать каждую инициализацию, вызов или включение макроса в качестве части исходного кода;
* не учитывать повторно используемые операторы исходного кода.

Расчет объема программного продукта (количества строк исходного кода) предполагает определение типа программного обеспечения, всестороннее техническое обоснование функций ПО и определение объема каждой функции. На стадии технико-экономического обоснования проекта невозможно рассчитать точный объем функций. Могут быть получены только ориентировочные (прогнозные) оценки на основе имеющихся фактических данных по аналогичным проектам, выполненным ранее, или путем применения действующих нормативов, которые в организациях должны периодически обновляться, уточняться и утверждаться как нормативы. На основании информации о функциях разрабатываемого ПО по каталогу функций определяется объем функций и общий объем ПО, который уточняется (корректируется) с учетом условий разработки ПО в организации[16].

Объем программного средства рассчитывается по формуле(1):

, (1)

где V0 – общий объем ПС;

Vi – объем функций ПС;

n – общее число функций;

V0 = 1128 команд.

На основании общего объема (Vo) и нормативных данных определяется нормативная трудоемкость с учетом группы сложности программного средства (ТН).

Выделяют три группы сложности программного продукта в зависимости от наличия или отсутствия у него одной либо нескольких основных характеристик:

* языковый интерфейс;
* ввод-вывод;
* организация данных;
* режимы работы.
* операционная и техническая среда.

На основании общего объема программного средства и его группы сложности по таблицам определяется нормативная трудоемкость программного продукта Тн = 38 человеко-дней.

Трудоемкость программного средства по стадиям определяется с учетом новизны и степени использования в разработке типовых программ и программного средства по формуле(2):

Тi = dстiхКнхКтх То, (2)

где Кт - поправочный коэффициент, учитывающий степень использования в разработке типовых (стандартных) программ;

Кн - коэффициент, учитывающий степень новизны программного средства;

Dстi – удельный вес трудоемкости i-ой стадии разработки ПС в общей трудоемкости разработки ПС.

При создании программных средств выделяют следующие стадии разработки:

* техническое задание (ТЗ);
* эскизный проект (ЭП);
* технический проект (ТП);
* рабочий проект (РП);
* внедрение (ВН).

На основании общей трудоемкости рассчитывается уточненная трудоемкость с учетом распределения по стадиям (Тут) по формуле(3):

Тут = , (3)

где Тi - трудоемкость разработки ПС на i-той стадии (чел.-дн.),

m - количество стадий разработки.

Расчеты отражаются в таблице 8.

Таблица 8 - Расчет уточненной трудоемкости

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии | | | | | Итого |
| Техническое задание | Эскизный проект | Технический проект | Рабочий проект | Внедрение |
| Коэффициенты удельных весов трудоемкости стадий, dст | 0,10 | 0,03 | 0,09 | 0,67 | 0,11 | 1,0 |
| Коэффициент новизны, Кн | 0,7 | 0,7 | 0,7 | 0,7 | 0,7 | 0,7 |
| Коэффициент, учитывающий использование типовых программ, Кт | 0,6 | 0,6 | 0,6 | 0,6 | 0,6 | 0,6 |
| Уточненная трудоемкость, Тут, стадий, чел.- дн. | 1.59 | 0.48 | 1.44 | 10.69 | 1.75 | 15.95 |

Примечание - Источник: собственная разработка.

Для определения потребности в специалистах занятых разработкой программного продукта необходимо рассчитать эффективный фонд времени работы.

Эффективный фонд времени работы одного работника (Фэф) рассчитывается по формуле(4):

Фэф = Дг - Дпв – До, (4)

где Дг - количество дней в году;

Дп - количество нерабочих дней (выходные и праздничные) в году;

До - количество дней отпуска (рекомендуется 21 раб. день).

Согласно постановлению Министерства труда и социальной защиты Республики Беларусь № 67 от 02.11.2022 г. для пятидневной рабочей недели в 2023 году количество праздничных и выходных дней устанавливается в количестве 113 (103 + 10), количество рабочих (обычных и предпраздничных) дней составляет 252 (247 + 5) дней.

Фэф = 365 - 113 - 21 = 231 дн.

На основании уточненной трудоемкости разработки программного средства и установленного периода разработки (0,63 года) рассчитывается общая плановая численность исполнителей Чр по формуле(5):

, (5)

где Чразоб- общая плановая численность разработчиков (чел.);

Тпл - плановая продолжительность разработки ПС (лет);

Фэф - эффективный фонд времени работы одного работника в течение года (дн/год).

При установленной плановой продолжительности разработки (Тр = 0,15 года) численность разработчиков составит 1 человек:

Чразоб = 15.95 / (0,15 х 231) = 0.46 чел = 1 чел.

**7.2** **Заработная плата исполнителей: основная и дополнительная**

Расчет основной заработной платы исполнителей программного средства (Зо) производится в соответствии с данными таблице 9.

Таблица 9- Исполнители программного средства

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнитель | Тарифный разряд | Тарифный коэффициент | Продолжительность участия в разработке, дней |
| Техник-программист 1 категории | 10 | 2,48 | 15.95 |

Примечание - Источник: собственная разработка.

Основная заработная плата исполнителей на конкретное ПС определяется за фактически отработанное время по формуле(6):

ЗПосн= , (6)

где n – количество исполнителей, занятых разработкой конкретного ПС;

Счас i – часовая тарифная ставка i-го исполнителя (руб.);

Фэф i – эффективный фонд рабочего времени i-го исполнителя (дн.);

Тр –количество часов работы в день (час.);

Кпр – коэффициент премирования.

Тарифная ставка первого разряда согласно локальному нормативному акту, разработанному в организации, устанавливается исходя из размера базовой ставки (приравнивается к ней). В соответствии с постановлением Совета Министров Республики Беларусь от 19 декабря 2022 г. № 887 «О размере базовой ставки» установлен размер базовой ставки с 1 января 2023 года базовая ставка 228 рублей.

Таблица 10 – Расчет основной заработной платы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Исполнители | С мес, руб | Н мес, час | С час, руб | С дн, руб | Продолжительность участия в разработке, дней | К пр, % | ЗПосн, руб |
| Техник-программист 1-ой категории | 565.44 | 167.6 | 3.37 | 26.96 | 15.95 | 1.2 | 516 |

Примечание – Источник: собственная разработка

Дополнительная заработная плата на конкретное программное средство (Зд) включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью) и определяется по нормативу в процентах к основной заработной плате по формуле(7):

ЗП доп = ЗПосн  \* Нд, (7)

где ЗПдоп – дополнительная заработная плата исполнителей на конкретное программное средство (тыс.руб.);

Нд  – норматив дополнительной заработной платы ( 8 %).

ЗП доп = 516 \* 0.08 = 41.28 руб.

**7.3** **Отчисления в фонд социальной защиты**

Отчисления в фонд социальной защиты населения (Офсзн) определяется в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей.

Сумма отчислений в фонд социальной защиты населения (Офсзн) определяется по формуле(8):

Офсзн = (ЗПосн+ЗПдоп)\*Нсзс/100, (8)

где Нфсзн – норматив отчислений в фонд социальной защиты (34%).

Офсзн = (516 + 41.28) \* 34% = 189.47 руб.

Отчисления по обязательному страхованию (Оос) определяются по нормативу по аналогии с фондом социальной защиты населения по формуле(9):

Оос = (ЗПосн+ЗПдоп)\*Нос/100, (9)

где Hос− норматив отчислений по обязательному страхованию (0.3%).

Оос = (516 + 41.28) \* 0.3% / 100 = 1.67 руб.

**7.4 Материалы**

Расходы по статье «Материалы» (М) определяются на основании сметы затрат, разрабатываемой на программное средство, с учетом действующих нормативов. По статье «Материалы» отражаются расходы на магнитные носители, перфокарты, бумагу, красящие ленты и другие материалы, необходимые для разработки программного средства[7].

Таблица 11 – Затраченные материалы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Материалы | Единицы измерения | Количество | Цена за единицу, руб | Сумма, руб |
| Бумага | Пачка | 0.47 | 12.61 | 5.92 |
| Тонер | Грамм | 13.2 | 0.04 | 0.53 |
| Папка со скоросшивателем | Штук | 1.0 | 1,34 | 1.34 |
| Папка с завязками | Штук | 1.0 | 0.92 | 0.92 |
| Ватман | Штук | 3.0 | 2.46 | 7.38 |
| Диск | Штук | 2.0 | 1.08 | 2.16 |
| Итого | | | | 18.25 |

**7.5 Машинное время**

Расходы по статье "Машинное время" (ВРмаш) включают оплату машинного времени, необходимого для разработки и отладки программного средства. Для разработки ПО использовался ноутбук Lenovo ideapad 330-15IKB, балансовая стоимость которого составляет 1144 руб, паспортная мощность 0.035 кВт-ч.

Стоимость машино-часа рассчитывается по формуле(10):

Смаш = Зоборуд/Фгод,, (10)

где Зоборуд - суммарные затраты, связанные с содержанием и эксплуатацией ПЭВМ за год, руб.;

Фгод - годовой действительный фонд времени работы ПЭВМ, час.

Суммарные затраты, связанные с содержанием и эксплуатацией ПЭВМ за год рассчитываются по формуле(11):

Зоборуд = Зматер + А + Зрем+ Зэл/эн + Зобслуж + Зпроч, (11)

где Зматер - затраты на материалы, руб.;

А - амортизационные отчисления вычислительной техники, руб.;

Зрем - расходы на текущий ремонт, руб.;

Зэл/эн - расходы на электроэнергию, руб.;

Зобслуж - расходы на обслуживание ПЭВМ, руб.;

Зпроч - прочие расходы, руб.

Затраты на материалы (магнитные носители, расходные материалы) определяются в процентах (п) от балансовой стоимости оборудования по формуле(12):

Зматер = Сбаланс\* п (%), (12)

где Сбаланс - стоимость ПЭВМ, вспомогательного оборудования с учетом расходов на транспортировку и установку (2% от стоимости оборудования).

Зматер = 1144 \* 2% /100 = 22,88 руб.

Расходы на текущий ремонт оборудования составляют 5% от балансовой стоимости оборудования и рассчитываются по формуле(13):

Зрем = Сбаланс\*0,05; (13)

Зрем = 1144\*0,05 = 57.2 руб.;

Расчет расходов на электроэнергию производится по формуле(14):

3эл/эн=Фгод\*M\*Cэл/эн, (14)

где Зэл\эн - расходы на электроэнергию, руб.

Фгод - годовой действительный фонд рабочего времени машины, час;

М - паспортная мощность оборудования, кВт-ч;

Сэл/эн - стоимость одного киловатт-часа электроэнергии, руб./кВт-ч

Годовой действительный фонд времени работы ПЭВМ определяется по формуле(15):

Фгод=((365-Дв-Дп)\*8-Дпп\*1)\*Kпот, (15)

где Дв=103 - количество выходных дней (при пятидневной рабочей неделе);

Дп= 10 количество праздничных дней;

Дпп - 5 количество предпраздничных дней (рабочий день короче 1 час);

Кпот - коэффициент потерь времени; Кпот=0,9.

Фгод = ((365-103-10)\*8 - 5\*1)\*0,9=2011\*0,9=1809,9= 1810ч.

3эл/эн = 1810 \* 0.035 \* 0.40659 = 25.75 руб.

Расчёт амортизации производится по формуле(16):

А = (Сбаланс \* На)/100, (16)

А = (1144 \* 10%)/100 = 114.4 руб.

Расходы на обслуживание ПЭВМ составляют 15% от балансовой стоимости оборудования и рассчитываются по формуле(17):

Зобслуж = Сбаланс\*0,15; (17)

Зобслуж = 1144 \* 0,15 = 171.6 руб.;

Прочие расходы составляют 1% от балансовой стоимости оборудования и рассчитываются по формуле(18):

Зпроч = Сбаланс\*0,01; (18)

Зпроч = 1144\*0,01 = 11.44 руб.

Зоборуд = 22.88 + 114.4 + 57.2 + 25.75 + 171.6 + 11.44 = 403.23 руб.

Смаш.час= 403.23 / 1810 = 0.22 руб.

Расчёт машинного времени производится по формуле(19):

ВРмаш = Смаш \* Тр \* 8; (19)

ВРмаш = 0.22\* 15.95\* 8 = 28.07 руб.

**7.6 Прочие затраты**

Расходы по статье "Прочие затраты" (ПрЗ) включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяются по нормативу в процентах к основной заработной плате по формуле(21):

ПрЗ = \* ЗПосн, (20)

где Нпз - норматив прочих затрат, ( 5%).

ПрЗ = 5 / 100 \* 516 = 25.8 руб.

**7.7 Накладные расходы**

Затраты по статье «Накладные расходы» (НР), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды, относятся на программное средство по нормативу (Ннр) в процентном отношении к основной заработной плате исполнителей по формуле(21):

НР=\*ЗПосн, (21)

где Ннр – норматив накладных расходов, (7-10 %).

НР = 7 / 100 \* 516 = 36.12 руб.

**7.8 Смета затрат и цена программного продукта**

Смета затрат на разработку программного средства включает следующие статьи:

* заработная плата исполнителей основная (Зосн) и дополнительная (З доп), руб.;
* отчисления в фонд социальной защиты (Осз), руб.;
* отчисления на обязательное страхование (Оос), руб.;
* материалы (М), руб.;
* машинное время (ВРмаш), руб.;
* прочие затраты (ПрЗ), руб.;
* косвенные затраты (НР), руб.;
* затраты на сопровождение и адаптацию программного средства (Зса) руб.;

На основании сметы затрат рассчитывается себестоимость и отпускная цена программного средства.

Исходные данные для расчета себестоимости и цены ПП приведены в таблице 12:

Таблица 12 **-** Исходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателей | Буквенные обозначения | Единицы измерения | Количество |
| 1 | 2 | 3 | 4 |
| Коэффициент новизны | Кн | единиц | 0.7 |
| Группа сложности |  | единиц | 3 |
| Поправочный коэффициент, учитывающий использование типовых программ | Кт | единиц | 0.6 |
| Установленная плановая продолжительность разработки | Тпл | лет | 0.63 |
| Годовой эффективный фонд времени | Фэф | дней | 231 |
| Продолжительность рабочего дня | Тр | час | 8 |
| Базовая ставка 1-го разряда, месячная | Тм | рублей | 228 |
| Коэффициент премирования | Кпр | % | 20 |
| Норматив дополнительной заработной платы | Нд | % | 8 |

Продолжение таблицы 12

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Ставка отчислений в фонд социальной защиты населения | Нсзс | % | 34 |
| Норматив прочих затрат | Нпз | % | 5 |
| Норматив накладных расходов | Нрн | % | 7 |
| Норматив на сопровождение и адаптацию программного средства | Нса | % | 3 |
| Уровень рентабельности | Урп | % | 15 |
| Ставка налога на добавленную стоимость | С НДС | % | 20 |

Общая сумма расходов рассчитывается по формуле(22):

Спр = ЗПосн + ЗПдоп + Офсзн + Оос + М + ВРмаш +ПрЗ + НР (22)

В целях систематизации данных составим следующую таблицу:

Таблица 13 – Смета затрат

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № пп | Наименование статей затрат | Обозначение | Норматив | Единица измерения | Сумма |
| 1 | Основная заработная плата разработчиков | ЗП осн |  | руб | 516 |
| 2 | Дополнительная заработная плата | ЗП доп | 8 % | руб | 41.28 |
| 3 | Отчисления в фонд социальной защиты населения | Осз | 34 % | руб | 189.47 |
| 4 | Обязательное страхование | Оос | 0,3% | руб | 1.67 |
| 5 | Материалы | М |  | руб | 18.25 |
| 6 | Машинное время | ВРмаш |  | руб | 28.07 |
| 7 | Прочие затраты | ПрЗ | 5 % | руб | 25.8 |
| 8 | Накладные расходы | НР | 7 % | руб | 36.12 |
| 9 | Производственная себестоимость | Спр |  | руб | 856.66 |

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию программного средства (Зса). Эти затраты определяются по нормативу в процентах от производственной себестоимости по формуле(23):

Зса=\*Спр, (23)

где Нса – норматив отчислений, 3%.

Зса = 3 / 100 \* 856.66 = 25.70 руб.

Полная себестоимость разработки программного средства определяется по формуле(24):

Спол=Спр+Зса. (24)

Спол= 856.66 + 25.70 = 882.36 руб.

**7.9 Экономическая эффективность программного средства**

Экономическая эффективность программных средств вычислительной техники достигается за счет экономии трудовых, материальных и финансовых ресурсов на основе:

Расчет экономического эффекта от программных средств осуществляется у разработчика и у потребителя. У разработчика экономический эффект выступает в виде прибыли, остающейся в распоряжении предприятия от реализации программного средства.

У пользователя затраты на программное средство складываются из единовременных и эксплуатационных (текущих) затрат. На основе технико-экономического анализа функциональных и конструктивных особенностей создаваемого программного средства и источников экономической эффективности определяются показатели, по которым рассчитывается экономия всех видов ресурсов, оказывающих влияние на величину экономического эффекта.

У разработчика экономический эффект выступает в виде прибыли, остающейся в распоряжении предприятия от реализации программного средства. Одним из самых распространенных и наиболее наглядных показателей оценки эффективности денежных вложений является период окупаемости. При анализе срока окупаемости коэффициент покажет, за какой промежуток времени окупятся затраты, связанные с созданием, продвижением и реализацией программного продукта.

При реализации программного продукта «OnionFree» предполагается использовать интернет-площадку «Itch.io». Данная площадка предоставляет пользователю возможность самостоятельно сформировать оплату за скачивание игры. Можно предположить, что пользователи, желая поддержать разработчика, будут делать хотя бы небольшие платежи. Кроме того, подобный путь реализации программного продукта даст возможность разработчику сформировать собственную аудиторию, которая в дальнейшем будет постоянно расширяться, что даст возможность применять различные методы монетизации данной деятельности и таким образом зарабатывать определённые денежные средства.

**Заключение**

Результатом дипломного проектирование является разработанное игровое приложение «OnionFree».

Программный продукт позволяет пользователю убить время и развлечься.

В качестве инструментов разработки были использованы Visual Studio 2019, Unity, Adobe Photoshop CS6.

В полной мере были реализованы требования, предъявляемые к программному продукту, такие как:

- простой и понятный интерфейс;

- простое управление;

- случайная генерация уровней;

- полная доступность всех игровых действий с самого начала игры;

- присутствие собираемых объектов, увеличивающих здоровье игрока;

- прогрессия игрока.

Также во время разработки игрового приложения была уделена особая важность проверке и отладке функциональности. Практически все видимые и скрытые ошибки были выявлены и исправлены, что обеспечивает стабильную работу программного продукта.

При разработке программного продукта соблюдались требования безопасности труда, электробезопасности, пожарной безопасности и производственной санитарии.

Был проведён анализ экономической эффективности программного продукта. Были рассчитаны затраты на разработку игрового приложения и оценена его потенциальная прибыль.

Также была разработана графическая часть проекта, представляющая собой 3 диаграммы:

* диаграмму вариантов использования,
* диаграмму классов,
* диаграмму деятельности.

В Приложении А приведён листинг программы.

Главную задачу дипломного проектирования можно считать выполненной успешно, так как разработанное игровое приложение «OnionFree» является логически завершённым и готовым к использованию.

**Список использованных источников**

1. EduNeo [Электронный ресурс]/ Цифровая обучающая игра в помощь учителю – Режим доступа: https://www.eduneo.ru/cifrovaya-obuchayushhaya-igra-v-pomoshh-uchitelyu/#:~:text=Игра%20—%20это%20концентрация%20на%20процессе,но%20и%20ради%20самой%20игры – Дата доступа: 07.05.2023.
2. КАРТАСЛОВ.РУ – Карта слов и вырожений русского языка [Электронный ресурс]/ Компьютерная ролевая игра – Режим доступа: <https://kartaslov.ru/%D0%BA%D0%B0%D1%80%D1%82%D0%B0-%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D0%B9/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F+%D1%80%D0%BE%D0%BB%D0%B5%D0%B2%D0%B0%D1%8F+%D0%B8%D0%B3%D1%80%D0%B0> – Дата доступа: 07.05.2023.
3. КАРТАСЛОВ.РУ – Карта слов и вырожений русского языка [Электронный ресурс]/ Шутер – Режим доступа: https://kartaslov.ru/значение-слова/шутер – Дата доступа: 07.05.2023.
4. NEOLURK [Электронный ресурс]/ Roguelite – Режим доступа: https://neolurk.org/wiki/Roguelite#:~:text=Roguelite%20(он%20же%20Roguelike-like,или%20время%20реальное%20вместо%20пошагового – Дата доступа: 07.05.2023.
5. C# ~ Си шарп для начинающих [Электронный ресурс]/ Спрайтовая анимация – Режим доступа: https://c-sharp.pro/спрайтовая-анимация/ – Дата доступа: 13.05.2023.
6. Национальный правовой интернет-портал Республики Беларусь [Электронный ресурс] / Нац. центр правовой информ. Респ. Беларусь / Конституция Республики Беларусь. — Режим доступа: https://pravo.by/pravovaya-informatsiya/normativnye-dokumenty/konstitutsiya-respubliki-belarus/ — Дата доступа: 17.06.2023.
7. Информационно-поисковая система «ЭТАЛОН-ONLINE» [Электронный ресурс] / Трудовой кодекс Республики Беларусь. — Режим доступа: https://etalonline.by/document/?regnum=HK9900296 — Дата доступа: 17.06.2023.
8. Декрет Президента Республики Беларусь от 30 июля 2003 г. № 18 «Об обязательном страховании от несчастных случаев на производстве и профессиональных заболеваний»
9. Закон Республики Беларусь «О пожарной безопасности» от 15.06.1993 г.
10. Закон Республики Беларусь «О санитарно-эпидемическом благополучии населения» от 23.11.1993 г. №2583-ХИ. — Дата доступа: 17.06.2023.
11. ГОСТ 12.0.004 «Организация обучения рабочих безопасности труда».
12. СанПиН 9-131 РБ 2000 Санитарные правила и нормы «Гигиенические требования к видеодисплейным терминалам, электронно-вычислительным машинам и организации работы».
13. **Типовая инструкция по охране труда при использовании в работе офисного оборудования** ПЭВМ с 14 апреля 2021 г. №25.
14. СанНП «Требования к микроклимату рабочих мест в производственных и офисных помещениях» и ГН «Показатели микроклимата производственных и офисных помещений» МЗ РБ №33 от 30.04.2013.
15. ГОСТ 12.4.021-75 «Система стандартов безопасности труда. Системы вентиляционные. Общие требования».
16. Палицын В.А. Технико – экономическое обоснование дипломных проектов: Метод. Пособие для студ. всех спец. БУГУИР. В 4 частях. Часть 4 – Проекты программного обеспечения/ В.А. Палицын. – Минск: БУГУИР, 2006. – 76с.
17. Онлайн магазин канцелярии «Деловой» [Электронный ресурс] - Режим доступа: <https://www.delovoy.by> – Дата доступа: 10.06.2023.

**­­**

**Приложение А**

**Листинг программы**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AddRoom : MonoBehaviour

{

[Header("Walls")]

public GameObject[] walls;

public GameObject door;

[Header("Walls")]

public GameObject portal;

[Header("Enemies")]

public GameObject[] enemyTypes;

public Transform[] enemySpawner;

[HideInInspector] public List<GameObject> enemies;

[HideInInspector] public bool isBossRoom;

[HideInInspector] public bool isBossDefeated;

[HideInInspector] public bool isLastRoom;

private RoomVariants variants;

private bool spawned;

private bool wallsDestroyed;

private void Awake()

{

variants = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomVariants>();

}

private void Start()

{

variants.rooms.Add(gameObject);

}

private void OnTriggerEnter2D(Collider2D other)

{

if(other.CompareTag("Player") && !spawned)

{

spawned = true;

if (!isBossRoom && gameObject.name != "MainRoom" && gameObject.name != "MainRoom(Clone)")

{

foreach (Transform spawner in enemySpawner)

{

int rand = Random.Range(0, 9);//до 11

if (rand < 9)

{

GameObject enemyType = enemyTypes[Random.Range(0, enemyTypes.Length)];

GameObject enemy = Instantiate(enemyType, spawner.position, Quaternion.identity) as GameObject;

enemy.transform.SetParent(gameObject.transform);

enemies.Add(enemy);

} /\*else if (rand == 9)

{

Instantiate(healthPotion, spawner.position, Quaternion.identity);

} else if (rand == 10)

{

Instantiate(shield, spawner.position, Quaternion.identity);

}\*/

}

}

else if (isBossRoom && gameObject.name != "MainRoom" && gameObject.name != "MainRoom(Clone)")

{

int rand = Random.Range(0, enemySpawner.Length);

GameObject enemy = Instantiate(enemyTypes[0], enemySpawner[rand].position, Quaternion.identity) as GameObject;

enemy.GetComponent<Enemy>().isBoss = true;

enemy.GetComponent<Enemy>().health \*= 2;

enemy.GetComponent<Enemy>().speed = 1;

enemy.GetComponent<Enemy>().damage = 2;

enemy.GetComponent<Enemy>().isBoss = true;

enemy.transform.localScale = new Vector3((float)(gameObject.transform.localScale.x \* 0.5), (float)(gameObject.transform.localScale.y \* 0.5));

enemy.transform.SetParent(gameObject.transform);

enemies.Add(enemy);

}

else

{

DestroyWalls();

}

StartCoroutine(CheckEnemies());

} else if (other.CompareTag("Player") && spawned)

{

foreach(GameObject enemy in enemies)

{

enemy.GetComponent<Enemy>().playerNotInRoom = false;

}

}

}

IEnumerator CheckEnemies()

{

yield return new WaitForSeconds(1f);

yield return new WaitUntil(() => enemies.Count == 0);

if (isBossRoom && isBossDefeated || isLastRoom)

portal.SetActive(true);

DestroyWalls();

}

public void DestroyWalls()

{

foreach(GameObject wall in walls)

{

if(wall != null && wall.transform.childCount != 0)

{

Destroy(wall);

}

}

wallsDestroyed = true;

}

private void OnTriggerStay2D(Collider2D other)

{

if(wallsDestroyed && other.CompareTag("Wall"))

{

Destroy(other.gameObject);

}

}

private void OnTriggerExit2D(Collider2D other)

{

if (other.CompareTag("Player") && spawned)

{

foreach (GameObject enemy in enemies)

{

enemy.GetComponent<Enemy>().playerNotInRoom = true;

}

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Bullet : MonoBehaviour

{

public float speed;

public float lifetime;

public float distance;

public int damage;

public LayerMask whatIsSolid;

public GameObject destroyEffect;

[SerializeField] bool enemyBullet;

private void Start()

{

Invoke("DestroyBullet", lifetime);

}

// Update is called once per frame

void Update()

{

RaycastHit2D hitInfo = Physics2D.Raycast(transform.position, transform.up, distance, whatIsSolid);

if(hitInfo.collider != null)

{

if(hitInfo.collider.CompareTag("Enemy"))

hitInfo.collider.GetComponent<Enemy>().TakeDamage(damage);

if (hitInfo.collider.CompareTag("Player") && enemyBullet)

hitInfo.collider.GetComponent<Player>().ChangeHealth(-damage);

DestroyBullet();

}

transform.Translate(Vector2.up \* speed \* Time.deltaTime);

}

public void DestroyBullet()

{

Instantiate(destroyEffect, transform.position, Quaternion.identity);

Destroy(gameObject);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ChangeRoom : MonoBehaviour

{

public Vector3 cameraChangePos;

public Vector3 playerChangePos;

private Camera cam;

private void Start()

{

cam = Camera.main.GetComponent<Camera>();

}

private void OnTriggerEnter2D(Collider2D other)

{

if (other.CompareTag("Player"))

{

other.transform.position += playerChangePos;

cam.transform.position += cameraChangePos;

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class DeadEnd : MonoBehaviour

{

private void Update()

{

Time.timeScale = 0f;

}

public void LoadMenu()

{

Time.timeScale = 1f;

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Enemy : MonoBehaviour

{

private float stopTime;

public float startStopTime;

private Animator anim;

private Player player;

private AddRoom room;

public GameObject destroyEffect;

private float timeBtwAttack;

public float startTimeBtwAttack;

public float speed;

public int health;

public int damage;

[HideInInspector] public bool playerNotInRoom;

[HideInInspector] public bool isBoss;

private bool stopped;

// Start is called before the first frame update

void Start()

{

anim = GetComponent<Animator>();

player = FindObjectOfType<Player>();

room = GetComponentInParent<AddRoom>();

stopped = false;

}

// Update is called once per frame

void Update()

{

if (!playerNotInRoom)

{

if (stopTime <= 0)

stopped = false;

else

{

stopped = true;

stopTime -= Time.deltaTime;

}

}

else

{

stopped = true;

}

if (room == null)

{

Debug.Log("Room is Null");//Попробуй присваивать комнату волкам при спавне

}

if (health <= 0)

{

room.enemies.Remove(gameObject);

if (isBoss)

room.isBossDefeated = true;

Destroy(gameObject);

}

if (player.transform.position.x > transform.position.x)

transform.eulerAngles = new Vector3(0, 180, 0);

else

transform.eulerAngles = new Vector3(0, 0, 0);

if(!stopped)

transform.position = Vector2.MoveTowards(transform.position, player.transform.position, speed \* Time.deltaTime);

}

public void TakeDamage(int damage)

{

stopTime = startStopTime;

health -= damage;

}

private void OnTriggerStay2D(Collider2D other)

{

if (other.CompareTag("Player"))

{

if (timeBtwAttack <= 0)

{

anim.SetTrigger("attack");

}

else

{

timeBtwAttack -= Time.deltaTime;

}

}

}

public void OnEnemyAttack()

{

Instantiate(destroyEffect, player.transform.position, Quaternion.identity);

player.ChangeHealth(-damage);

timeBtwAttack = startTimeBtwAttack;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Gun : MonoBehaviour

{

public GameObject bullet;

public Transform shotPoint;

public GunType gunType;

public float startTimeBtwShots;

public float offset;

public enum GunType { Default, Enemy }

private float timeBtwShots;

private float rotZ;

private Vector3 difference;

private Player player;

// Start is called before the first frame update

void Start()

{

player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();

}

// Update is called once per frame

void Update()

{

if (gunType == GunType.Default)

{

difference = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;

rotZ = Mathf.Atan2(difference.y, difference.x) \* Mathf.Rad2Deg;

}

else if (gunType == GunType.Enemy)

{

difference = player.transform.position - transform.position;

rotZ = Mathf.Atan2(difference.y, difference.x) \* Mathf.Rad2Deg;

}

transform.rotation = Quaternion.Euler(0f, 0f, rotZ + offset);

if (timeBtwShots <= 0)

{

if (Input.GetMouseButton(0) || gunType == GunType.Enemy)

Shoot();

} else

{

timeBtwShots -= Time.deltaTime;

}

}

public void Shoot()

{

Instantiate(bullet, shotPoint.position, shotPoint.rotation);

timeBtwShots = startTimeBtwShots;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

using UnityEngine;

public class MainMenu : MonoBehaviour

{

public void PlayGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

public void ExitGame()

{

Debug.Log("Exit");

Application.Quit();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour

{

public static bool isPaused = false;

public GameObject pauseMenuUI;

// Update is called once per frame

void Update()

{

if (Input.GetKeyUp(KeyCode.Escape))

if (isPaused)

Resume();

else

Pause();

}

public void Resume()

{

pauseMenuUI.SetActive(false);

Time.timeScale = 1f;

isPaused = false;

}

public void LoadMenu()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);

}

void Pause()

{

pauseMenuUI.SetActive(true);

Time.timeScale = 0f;

isPaused = true;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class Player : MonoBehaviour

{

public float speed;

public int health;

public Text healthDisplay;

public Text level;

public Text floor;

private Rigidbody2D rb;

private Vector2 moveInput;

private Vector2 moveVelocity;

private Animator anim;

private bool facingRight = true;

private bool keyButtonPushed;

private int levelNum;

private int floorNum;

public GameObject mainRoom;

public GameObject Room;

public GameObject deadScreen;

public GameObject HUD;

[Header("Key")]

public GameObject keyIcon;

/\*public GameObject portalIcon;\*/

// Start is called before the first frame update

void Start()

{

rb = GetComponent<Rigidbody2D>();

anim = GetComponent<Animator>();

levelNum = int.Parse(level.text);

floorNum = int.Parse(floor.text);

}

// Update is called once per frame

void Update()

{

healthDisplay.text = health.ToString();

moveInput = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"));

moveVelocity = moveInput.normalized \* speed;

if(moveInput.x == 0)

anim.SetBool("isRunning", false);

else

anim.SetBool("isRunning", true);

if (!facingRight && moveInput.x>0)

Flip();

else if(facingRight && moveInput.x<0)

Flip();

if (health <= 0)

{

deadScreen.SetActive(true);

HUD.SetActive(false);

}

}

private void FixedUpdate()

{

rb.MovePosition(rb.position + moveVelocity \* Time.fixedDeltaTime);

}

private void Flip()

{

facingRight = !facingRight;

Vector3 Scaler = transform.localScale;

Scaler.x \*= -1;

transform.localScale = Scaler;

}

public void ChangeHealth(int healthValue)

{

health += healthValue;

}

private void OnTriggerEnter2D(Collider2D other)

{

if (other.CompareTag("Key"))

{

keyIcon.SetActive(true);

Destroy(other.gameObject);

}

}

public void onKeyButtonDown()

{

keyButtonPushed = !keyButtonPushed;

}

/\*public void onPortalBurronDown()

{

portalButtonPushed = !portalButtonPushed;

}\*/

private void OnTriggerStay2D(Collider2D other)

{

if (other.CompareTag("Door") && keyButtonPushed && keyIcon.activeInHierarchy)

{

keyIcon.SetActive(false);

other.gameObject.SetActive(false);

keyButtonPushed = false;

}

if (other.CompareTag("Portal"))

{

if (int.Parse(floor.text) < 3)

{

floorNum++;

floor.text = floorNum.ToString();

}

else

{

levelNum++;

level.text = levelNum.ToString();

floorNum = 1;

floor.text = floorNum.ToString();

}

if (level.text != "3")

{

GameObject[] rooms = GameObject.FindGameObjectsWithTag("Room");

foreach(var room in rooms)

{

Destroy(room);

}

Instantiate(Room, Camera.main.GetComponent<Camera>().transform.position, Quaternion.identity);

Instantiate(mainRoom, Camera.main.GetComponent<Camera>().transform.position, Quaternion.identity);

gameObject.transform.position = Camera.main.GetComponent<Camera>().transform.position;

}

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RoomSpawner : MonoBehaviour

{

public Direction direction;

public enum Direction

{

Top,

Bottom,

Left,

Right,

None

}

private RoomVariants variants;

private int rand;

private bool spawned = false;

private float waitTime = 3f;

private void Start()

{

variants = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomVariants>();

Invoke("Spawn", 0.2f);

Destroy(gameObject, waitTime);

}

public void Spawn()

{

if (!spawned)

{

if(direction == Direction.Top)

{

rand = Random.Range(0, variants.topRooms.Length);

Instantiate(variants.topRooms[rand], transform.position, variants.topRooms[rand].transform.rotation);

} else if (direction == Direction.Bottom)

{

rand = Random.Range(0, variants.bottomRooms.Length);

Instantiate(variants.bottomRooms[rand], transform.position, variants.bottomRooms[rand].transform.rotation);

}else if (direction == Direction.Left)

{

rand = Random.Range(0, variants.leftRooms.Length);

Instantiate(variants.leftRooms[rand], transform.position, variants.leftRooms[rand].transform.rotation);

}else if (direction == Direction.Right)

{

rand = Random.Range(0, variants.rightsRooms.Length);

Instantiate(variants.rightsRooms[rand], transform.position, variants.rightsRooms[rand].transform.rotation);

}

spawned = true;

}

}

private void OnTriggerStay2D(Collider2D other)

{

if(other.CompareTag("RoomPoint") && other.GetComponent<RoomSpawner>().spawned)

{

Destroy(gameObject);

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RoomVariants : MonoBehaviour

{

public GameObject[] topRooms;

public GameObject[] bottomRooms;

public GameObject[] leftRooms;

public GameObject[] rightsRooms;

public GameObject key;

private Player player;

[HideInInspector] public List<GameObject> rooms;

private void Start()

{

StartCoroutine(RandomSpawner());

player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();

}

IEnumerator RandomSpawner()

{

yield return new WaitForSeconds(5f);

AddRoom lastRoom = rooms[rooms.Count - 1].GetComponent<AddRoom>();

lastRoom.isLastRoom = true;

if (int.Parse(player.floor.text) == 3)

lastRoom.isBossRoom = true;

int rand = Random.Range(0, rooms.Count - 2);

Instantiate(key, rooms[rand].transform.position, Quaternion.identity);

lastRoom.door.SetActive(true);

lastRoom.DestroyWalls();

Destroy(gameObject);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Wall : MonoBehaviour

{

public GameObject block;

private void OnTriggerEnter2D(Collider2D other)

{

if (other.CompareTag("Block"))

{

Debug.Log("Block behind the Wall!");

Instantiate(block, transform.GetChild(0).position, Quaternion.identity);

Instantiate(block, transform.GetChild(1).position, Quaternion.identity);

Destroy(gameObject);

}

}

private void OnTriggerStay2D(Collider2D other)

{

if (other.CompareTag("Block"))

{

Debug.Log("Block behind the Wall!");

Instantiate(block, transform.GetChild(0).position, Quaternion.identity);

Instantiate(block, transform.GetChild(1).position, Quaternion.identity);

Destroy(gameObject);

}

}

}

Граф часть

ДВИ

Диаграмма Классов

Диаграмма Деятельности