Оглавление

[Перечень условных обозначений и сокращений 7](#_Toc105192763)

[Введение 8](#_Toc105192764)

[1 Средства разработки игровых](#_Toc105192765) [приложений для персональных компьютеров 9](#_Toc105192766)

[1.1 Игровой движок *Unity*3*D* 9](#_Toc105192767)

[1.2 Игровой движок *Godot Engine* 11](#_Toc105192768)

[1.3 Игровой движок *RPG Maker* 12](#_Toc105192769)

[1.4 Анализ игрового жанра «*Roguelike* с элементами *RPG»* 14](#_Toc105192770)

[2 Архитектура игрового приложкения](#_Toc105192771) [«*Gone Summer: Overgrowth*» 19](#_Toc105192772)

[2.1 Основные этапы разработки игровых приложений 19](#_Toc105192773)

[2.2 Основные механики игрового приложения «*Heroic Adventure*» 20](#_Toc105192774)

[2.3 Иерархия классов игрового приложения «*Heroic Adventure*» 21](#_Toc105192775)

[2.4 Архитектура пользовательского интерфейса игрового](#_Toc105192776) [приложения «*Heroic Adventure*» 25](#_Toc105192777)

[3 Программная реализация игрового приложения](#_Toc105192778) [«*Heroic Adventure*» 29](#_Toc105192779)

[3.1 Методология проектирования приложения 29](#_Toc105192780)

[3.2 Принцип работы классов системы характеристик игрового](#_Toc105192781)

[приложения «*Heroic Adventure*» 30](#_Toc105192782)

[3.3 Принцип работы классов системы квестов игрового приложения](#_Toc105192783)

[«*Heroic Adventure*» 35](#_Toc105192784)

[3.4 Принцип работы классов системы инвентаря и экипировки](#_Toc105192785)

[игрового приложения «*Heroic Adventure*» 38](#_Toc105192786)

[4 Валидация и верификация результатов работы](#_Toc105192787) [игрового приложения «*Heroic Adventure*» 42](#_Toc105192788)

[4.1 Виды тестирования игр 42](#_Toc105192789)

[4.2 Функциональное тестирование 42](#_Toc105192790)

[4.3 Юзабилити-тестирование 46](#_Toc105192791)

[4.4 Итоги тестирования 47](#_Toc105192792)

[5 Экономическое обоснование игрового приложения «*Heroic Adventure*» 49](#_Toc105192793)

[5.1 Технико-экономическое обоснование целесообразности разработки](#_Toc105192794)

[программного продукта и оценка его конкурентоспособности 49](#_Toc105192795)

[5.2 Оценка трудоёмкости работ по созданию программного](#_Toc105192796)

[обеспечения 50](#_Toc105192797)

[5.3 Расчёт затрат на разработку программного продукта 54](#_Toc105192798)

[5.4 Расчёт договорной цены и частных экономических эффектов от](#_Toc105192799)

[производства и использования программного продукта 57](#_Toc105192800)

[6 Охрана труда и техника безопасности 59](#_Toc105192801)

[6.1 Причины возникновения, параметры и физические](#_Toc105192802) [характеристики вибрации 59](#_Toc105192803)

[7 Энергосбережение и ресурсосбережение при](#_Toc105192804) [эксплуатации программного обеспечения 62](#_Toc105192805)

[7.1 Энергосбережение и ресурсосбережение при внедрении и](#_Toc105192806)

[эксплуатации программного обеспечения 62](#_Toc105192807)

[Заключение 65](#_Toc105192808)

[Список использованных источников 66](#_Toc105192809)

[Приложение А Листинг программы 67](#_Toc105192810)

[Приложение Б Руководство пользователя 84](#_Toc105192811)

[Приложение В Руководство системного программиста 88](#_Toc105192812)

[Приложение Г Руководство программиста 89](#_Toc105192813)

[Приложение Д Формулы расчёта экономической эффективности 90](#_Toc105192814)

[Приложение Ж Список опубликованных работ 98](#_Toc105192815)

# СРЕДСТВА РАЗРАБОТКИ ИГРОВЫХ

# ПРИЛОЖЕНИЙ ДЛЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

## Игровой движок *Unity*3*D*

*Unity* – это мощный инструмент для разработки игр, который объединяет в себе разнообразные инструменты, необходимые для создания игрового контента. Этот движок позволяет разработчикам быстро и легко создавать игры, при этом обладая полным контролем над процессом разработки.

Благодаря обширной документации и сообществу разработчиков, *Unity* является одним из самых популярных инструментов для создания игр на сегодняшний день. Его графический интерфейс интуитивно понятен и удобен в использовании, что позволяет создавать игры различных жанров и сложности. *Unity* также обеспечивает высокую производительность, что делает его идеальным выбором для создания игр для мобильных устройств, консолей и ПК.

Благодаря удобному *Drag* *&* *Drop* интерфейсу и функциональному графическому редактору движок позволяет рисовать карты и расставлять объекты в реальном времени и сразу же тестировать получившийся результат.

Такой подход дает возможность быстро и эффективно прототипировать игры, а также ускоряет процесс создания уровней и сцен в проекте. Все это делает *Unity* одним из самых удобных и популярных игровых движков на сегодняшний день.

Движок поддерживает два скриптовых языка: *C*# и *JavaScript*. Расчёты физики производит физический движок *PhysX* от *NVIDIA*. Среду разработки *Unity*, можно увидеть на рисунке 1.1.



Рисунок 1.1 – Среда разработки *Unity*

*Unity* 3*D* поддерживает систему *Level* *Of* *Detail* (сокр. *LOD*), суть которой заключается в том, что на дальнем расстоянии от игрока высокодетализированные модели заменяются на менее детализированные, и наоборот, а также систему *Occlusion* *culling*, суть которой в том, что у объектов, не попадающих в поле зрения камеры, не визуализируется геометрия и коллизия, что снижает нагрузку на центральный процессор и позволяет оптимизировать проект. При компиляции проекта создается исполняемый (.*exe*) файл игры (для *Windows*), а в отдельной папке – данные игры (включая все игровые уровни и динамически подключаемые библиотеки).

Движок поддерживает множество популярных форматов. Модели, звуки, текстуры, материалы, скрипты можно запаковывать в формат *unityassets* и передавать другим разработчикам, или выкладывать в свободный доступ. Этот же формат используется во внутреннем магазине *Unity* *Asset* *Store*, в котором разработчики могут бесплатно и за деньги выкладывать в общий доступ различные элементы, нужные при создании игр. Чтобы использовать *Unity* *Asset* *Store*, необходимо иметь аккаунт разработчика *Unity*. *Unity* имеет все нужные компоненты для создания мультиплеера. Также можно использовать подходящий пользователю способ контроля версий. К примеру, *Tortoise* *SVN* или *Source* *Gear*.

В *Unity* входит *Unity* *Asset* *Server* – инструментарий для совместной разработки на базе *Unity*, являющийся дополнением, добавляющим контроль версий и ряд других серверных решений. Как правило, игровой движок предоставляет множество функциональных возможностей, позволяющих их задействовать в различных играх, в которые входят моделирование физических сред, карты нормалей, динамические тени и многое другое.

В отличие от многих игровых движков, у *Unity* имеется два основных преимущества: наличие визуальной среды разработки и межплатформенная поддержка. Первый фактор включает не только инструментарий визуального моделирования, но и интегрированную среду, цепочку сборки, что направлено на повышение производительности разработчиков, в частности, этапов создания прототипов и тестирования. Под межплатформенной поддержкой предоставляется не только места развертывания (установка на персональном компьютере, на мобильном устройстве, консоли и т. д.), но и наличие инструментария разработки (интегрированная среда может использоваться под *Windows* и *Mac* *OS*);

Преимущества *Unity*:

– мультиплатформенность;

– визуальный редактор;

– широкий список технологий реализации приложений;

– поддержка различных языков программирования;

– кроссплатформенная разработка;

– расширяемость и настраиваемость;

– активное сообщество разработчиков.

Недостатки *Unity*:

– размер файлов разрабатываемых приложений;

– зависимость конечных приложений от *Unity*;

– ограничения бесплатной версии движка;

– проблемы с производительностью на мобильных устройствах.

## Игровой движок *Godot Engine*

*Godot Engine* – открытый кроссплатформенный 2*D* и 3*D* игровой движок под лицензией *MIT*, который разрабатывается сообществом *Godot* *Engine* *Community* и является максимально интегрированной и самодостаточной средой для разработки игр. Среда позволяет разработчикам создавать игры с нуля, не пользуясь более никакими инструментами, за исключением тех, которые необходимы для создания игрового контента (элементы графики, музыкальные треки и т. д.). Процесс программирования также не требует внешних инструментов (хотя при необходимости использовать внешний редактор – это можно сделать относительно легко). Общая архитектура движка построена вокруг концепции дерева из наследуемых «сцен». Каждый элемент сцены (узел), в любой момент сам может стать полноценной сценой. Поэтому при разработке можно легко изменять полностью всю архитектуру проекта, расширять её элементы в любую сторону и работать с комплексными сценами на уровне простых абстракций.

*Godot* поддерживает различные платформы, включая *Windows*, *macOS*, *Linux*, *Android*, *iOS*, *Web* и другие. Это позволяет разработчикам создавать игры для различных устройств и платформ, что увеличивает их охват и аудиторию.

Интерфейс *Godot Engine* представлен на рисунке 1.2.

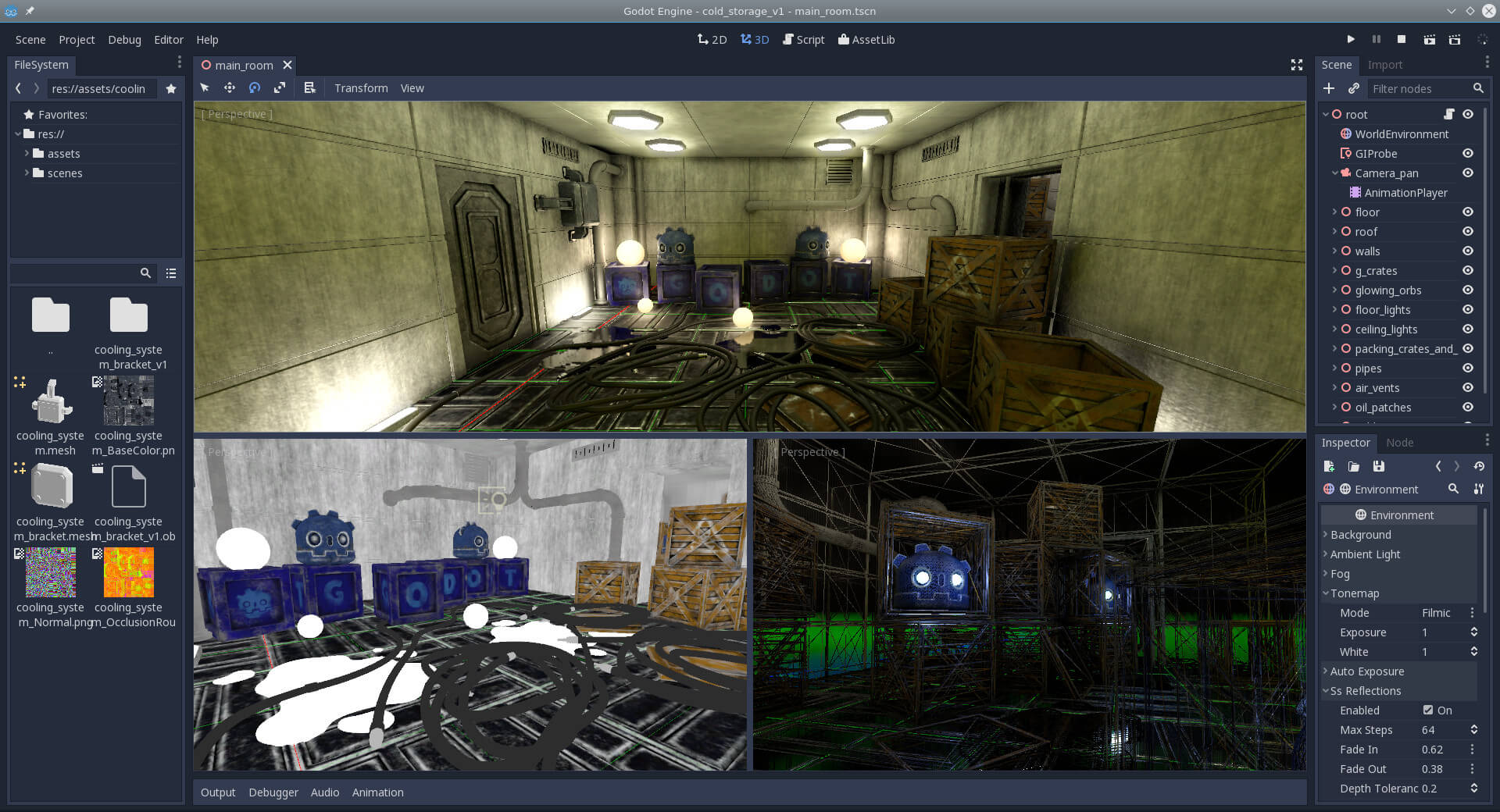


Рисунок 1.2 – Интерфейс *Godot Engine*

Все игровые ресурсы, от скриптов до графических заготовок и игровых сцен, хранятся в папке проекта как обычные файлы, и не являются частью сложной базы данных проекта. Ресурсы, которые не представляют собой комплексных данных, хранятся в простых текстовых форматах (например, скрипты и сцены, в отличие от моделей и текстур). Эти решения позволяют значительно упростить разным командам разработчиков работу с системами управления версиями.

*Godot* также имеет свой собственный встроенный язык сценариев, *GDScript*, высокого уровня, динамически типизированных [язык программирования](https://en.wikipedia.org/wiki/Programming_language) очень похож на [*Python*](https://en.wikipedia.org/wiki/Python_(programming_language)). В отличие от *Python*, *GDScript* имеет строгую типизацию переменных и оптимизирован для архитектуры *Godot* на основе сцен. Разработчики *Godot* заявили, что многие альтернативные сторонние языки сценариев, такие как *Lua*, *Python* и *Squirrel*, были протестированы, прежде чем было решено, что использование пользовательского языка обеспечивает превосходную оптимизацию и интеграцию редактора. Движок также поддерживает визуальное кодирование через собственный встроенный [язык визуального программирования.](https://en.wikipedia.org/wiki/Visual_programming_language) *VisualScript*, разработанный как визуальный эквивалент *GDScript*.

В конце разработки проект может быть экспортирован на разные [целевые платформы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0), которые можно разделить на ПК, мобильные, веб и консоли. Для разных платформ можно задавать разные параметры, такие как способ хранения данных (и их защита при необходимости), компрессия текстур, разрешение, а также некоторые уникальные параметры, характерные только для какой-то конкретной платформы (например разрешения для *Android*).

Текущая поддержка платформ включает *Windows*, *MacOS*, *Android OS,* *iOS*, *HTML*5. Также можно производить экспорт на другие платформы вручную через [компилирование](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) движка для *SDK* целевой платформы. Использование *Godot* незначительного количества внешних библиотек облегчает этот процесс.

Преимущества:

– система узлов;

– относительно простой в понимании язык программирования – *GDScript*;

– умеренно активное сообщество разработчиков;

– мультиплатформенность;

– удобный и простой редактор анимации и изображений.

Недостатки:

– слабая оптимизация конечных игровых продуктов;

– посредственная известность и меньший рынок труда;

– разработка исключительно на языке программирования *GDScript*;

– отсутствие интегрированной разработки для *VR*/*AR*;

– малое количество обучающих материалов;

– отсутствие интеграции систем управления версиями.

## Игровой движок *RPG* Maker

*RPG Maker* – это игровой движок, предназначенный для создания классических ролевых игр. Он был создан компанией *Enterbrain* в Японии и был выпущен в 1988 году. С тех пор множество версий были выпущены, каждая с улучшенными функциями и возможностями. Движок имеет простой и понятный пользовательский интерфейс, который позволяет даже новичкам создавать игры без необходимости в знании программирования. *RPG* *Maker* включает в себя редакторы миров, персонажей, монстров, событий, диалогов и битв, что позволяет создавать различные сценарии и настройки для игры. Пример интерфейса *RPG Maker* представлен на рисунке 1.3.



Рисунок 1.3 – Интерфейс игрового движка *RPG Maker*

*RPG* *Maker* имеет большое сообщество пользователей, которые создают и делятся своими играми и ресурсами. Есть множество бесплатных и платных *DLC*, которые добавляют новые функции и ресурсы, такие как текстуры, музыка и звуки. Также существует огромное количество бесплатных скриптов и плагинов, которые расширяют возможности движка.

Одним из основных преимуществ *RPG* *Maker* является его доступность. Даже если у разработчика нет опыта в программировании или дизайне игр, он может легко создать свою игру с помощью инструментария движка. Кроме того, благодаря огромному сообществу пользователей и разработчиков можно всегда получить помощь и поддержку в создании своей игры.

Основной причиной, по которой начинающие разработчики выбирают данный игровой движок является его простота и полнота в рамках создания определённого жанра игр, а также обилие проектов, уже созданных с использованием инструментария *RPG Maker*.

Хотя *RPG* *Maker* может быть ограничен в сравнении с другими игровыми движками в терминах графики и функциональности, он все еще является одним из самых популярных и доступных инструментов для создания классических ролевых игр.

*RPG Maker* известен своей простотой использования и доступностью для новичков. Он предлагает интуитивный интерфейс и удобные инструменты для создания игровых миров, персонажей, диалогов, событий и системы боя без необходимости в глубоких знаниях программирования.

*RPG Maker* поставляется с богатой библиотекой готовых графических ресурсов, музыки, звуковых эффектов и шаблонов, что позволяет разработчикам быстро начать создавать свою игру, не тратя время на создание всего с нуля. Движок имеет встроенную систему скриптинга, которая позволяет более опытным разработчикам создавать собственные функции, расширять возможности движка и настраивать игровую логику. Это позволяет добавлять уникальные особенности и адаптировать игру под свои потребности. Данный игровой движок также имеет большое и активное сообщество разработчиков, которые делятся опытом, обучающими материалами, скриптами и ресурсами. Это облегчает получение поддержки и помощи в случае возникновения проблем или вопросов.

*RPG Maker* в основном ориентирован на разработку игр для настольных компьютеров и может иметь ограниченную поддержку для других платформ, таких как мобильные устройства или консоли. Если нужно создать игру для определенной платформы, может потребоваться дополнительная настройка или использование сторонних инструментов.

*RPG Maker* предлагает набор предопределенных функций и систем, позволяющих быстро и просто реализовать основные игровые механики, однако иногда может быть сложно реализовать сложные или нестандартные механики игры, которые выходят за рамки предоставленных возможностей. В этом случае потребуется более глубокое понимание скриптинга и настройки, однако несмотря на эти недостатки, *RPG Maker* остается популярным выбором для начинающих разработчиков, которые хотят создать свои собственные 2*D* ролевые игры с относительной легкостью и без необходимости глубоких знаний программирования.

Преимущества:

– простота интерфейса и реализации основных игровых механик;

– обширная библиотека готовых игровых ресурсы;

– поддержка пользовательских скриптов;

– активное сообщество разработчиков.

Недостатки:

– ограниченные возможности в рамках жанра;

– визуальные ограничения движка;

– зависимость от готовых игровых ресурсов;

– ограниченный выбор платформ;

– ограниченный контроль над функциональностью;

– ограниченная поддержка внешних разработчиков.

## 1.4 Анализ игрового жанра «*Roguelike* с элементами *RPG»*

*Roguelike* с элементами *RPG* (*Role-Playing Game*) – это игровой жанр, объединяющий особенности рогалика и классической ролевой игры. В нем часто встречаются такие характеристики, как обширность и разнообразие уровней, постоянное наличие вызова для персонажа, высокая степень сложности и акцент на стратегическом принятии решений.

Одна из ключевых особенностей *Roguelike* – это обширное количество уровней и контента, что означает, что каждый раз, когда игрок начинает новую игру, мир и задания будут уникальными. Это способствует повышению реиграбельности, создает ощущение непредсказуемости и позволяет игрокам исследовать новые возможности и стратегии.

Игры такого рода обычно используют механику постоянного повышения сложности вместе с прогрессией персонажа, где игроки получают соответствующей сложности вызов при повышении навыков игры. Это создает высокую степень увлечённости и стремление к улучшению и изучению игрового мира, а также поддерживает взаимодействие с игроком в форме прогресса или открытия новых элементов игры после каждого прохождения.

*Roguelike* с элементами *RPG* обычно предлагают разнообразие игровых предметов и механик, будь то различная амуниция и предметы или механики взаимодействия с окружением и врагами. Игроки могут развивать своего персонажа по мере прохождения, дополняя его способности новыми или улучшая старые атрибуты и показатели, находить и улучшать экипировку и создавать уникальные комбинации. Это позволяет игрокам создавать свои уникальные стратегии и адаптироваться к различным ситуациям в игре.

Игры подобного жанра обычно представляют собой сложные игры, требующие от игрока хорошего планирования, стратегического мышления и умения принимать быстрые решения. Они могут представлять вызов для игроков, но при этом предлагать ощущение удовлетворения и достижения при успешном прохождении.

*Roguelike* с элементами *RPG* часто позволяют игрокам экспериментировать с различными стратегиями и подходами к игре. Игроки могут находить необычные комбинации предметов и способностей, исследовать различные варианты прокачки и расширять свои знания о мире игры. Это способствует созданию глубокого и разнообразного игрового опыта.

Разнообразие игровых стилей в такого рода играх не является редкостью. Некоторые игры могут сконцентрироваться на боевых сражениях и тактике, где игроки должны выбирать правильные навыки и стратегии во время сражений. Другие могут уделять больше внимания исследованию мира, квестам и интерактивности с окружающей средой. Это позволяет игрокам выбирать игровой стиль, который больше всего соответствует их предпочтениям.

Важно отметить, что игра в таком жанре может иметь разные вариации и подходы, и каждая игра в этом жанре может иметь свои уникальные особенности и механики. В итоге, любители вызовов, стратегической игры и постоянного исследования найдут в рогаликах с элементами *RPG* увлекательный игровой опыт. Пример игры в стиле Roguelike с элементами RPG проиллюстрирован на рисунке 1.4.



Рисунок 1.4 – Игра в стиле *Roguelike* с элементами *RPG*

Глубокий игровой мир – ещё одна особенность, которую могут предложить игры в жанре *Roguelike*. Обширные и детально проработанные игровые миры могут быть заполнены различными локациями, загадками, секретами, персонажами и историями. Игрокам предоставляется возможность погрузиться в уникальную атмосферу и исследовать разнообразные места, открывая новые возможности и раскрывая глубину игрового мира.

Несмотря на преимущества, есть и некоторые потенциальные недостатки *Roguelike* с элементами *RPG*, например сложность и не прощение ошибок, вель игры в этом жанре могут быть очень сложными и не прощать ошибок. Очередная смерть персонажа может означать потерю всех достижений и прогресса. Это может вызывать фрустрацию у некоторых игроков, особенно у тех, кто предпочитает более доступные и мягкие игровые опции.

*Roguelike* могут требовать значительных временных и эмоциональных инвестиций. Игрокам может потребоваться провести много часов, чтобы достичь успеха в игре и раскрыть все ее возможности. Некоторые игроки могут не иметь достаточно времени или терпения для такого типа игр.

Хотя случайная генерация уровней и контента может быть преимуществом в играх жанра *Roguelike*, она также может быть и недостатком. Некоторые игроки могут предпочитать более предсказуемые и хорошо настроенные игровые сценарии, где у них есть больше контроля и понимания того, что ожидать.

*Roguelike* с элементами *RPG* могут больше акцентироваться на игровых системах и механиках, чем на глубоком сюжете и персонажах. Если для пользователя важен сильный сюжет и эмоциональное вовлечение, то он может найти *Roguelike* с элементами *RPG* менее удовлетворительными, однако для аудитории, что ценит в игре в первую очередь обилие механик и разнообразие такого рода жанр покажется крайне располагающим к изучению.

В целом, *Roguelike* с элементами *RPG* предлагают уникальный игровой опыт, сочетая в себе элементы стратегии, случайности и ролевых игр. Они предлагают игрокам возможность исследовать случайно генерируемые миры, развивать персонажей и создавать свои стратегии. Однако, они также могут представлять значительные вызовы и требовать больших усилий для достижения успеха в игре.

*Roguelike* – жанр компьютерных игр, в котором делается упор на эксплуатацию тактических и физических умениях игрока, прогрессирующий по мере изучения игровых механик и логики игрового мира. Жанр не ограничивается самобытностью и представлен во множестве разновидностей и комбинайций, начиная от ролевых игр и платформеров, и заканчивая шутерами и файтингами.

Первые два игровых жанра всё же занимают основную часть и являются основой для комбинации с другими игровыми стилями и жанрами, при этом всё также оставаясь самобытными, оставляя пространство для создания новых сочетаний и идей в процессе разработки или игровой сессии.

Представителем жанра *Roguelike Beat’Em Up* является уникальная игра, разработанная студией *Sloclap* – *Sifu*. Она предлагает захватывающий и динамичный игровой опыт, сочетающий элементы *Roguelike* и файтинга.

*Sifu* использует случайную генерацию уровней, что означает, что каждый проход будет уникальным. Это добавляет повторяемость и вызов игре, поскольку игроку предстоит исследовать различные уровни, сталкиваться с разными врагами и решать разнообразные задачи при каждой новой попытке.

Боевая система *Sifu* основана на классической боевой системе, характерной для жанра *Beat’Em Up*, где игрок сражается с противниками в ближнем бою. В игре представлены разнообразные стили и приемы боя, которые можно использовать для совершения комбинаций и контроля над противниками. Игра поддерживает интенсивные и кинематографичные боевые сцены, создающие атмосферу драки и динамизма.

Прогресс и развитие персонажа в *Sifu* заключается в том, что игроку предстоит играть в роли молодого бойца, у которого в процессе игры будут развиваться различного рода способности. Таким образом игрок усиливает своего персонажа улучшая его навыки, а также собственную тактику боя. Каждая смерть персонажа имеет последствия, но при этом не исключается то, что игрок будет накапливать опыт, который позволит ему становиться сильнее и эффективнее в следующих попытках.

Кинематографический стиль и атмосфера в *Sifu* отличается своим стилизованным кинематографическим подходом, создавая интригующую и захватывающую атмосферу.

Отличная анимация и детализация окружающего мира позволяют погрузиться в увлекательное путешествие героя из чего можно заключить, что *Sifu* представляет собой захватывающую комбинацию жанров *Roguelike* и *Beat’Em Up* с упором на файтинг. Она предлагает игрокам динамичные боевые сражения, случайную генерацию уровней и возможность развития персонажа и подходит для игроков, ценящих интенсивный и стильный игровой процесс.

Скриншот игры *Sifu* показан на рисунке 1.5.



Рисунок 1.5 – Пример игры в жанре *Roguelike* *Beat’Em Up*

Ещё одним представитель смешанного жанра является игра *Bullet Per Minute* (сокр. *BPM*) – это игра в жанре *Roguelike* шутер, разработанная студией *Awe* *Interactive*. Она предлагает уникальный геймплей, сочетающий стрельбу от первого лица и элементы рогалика.

Ритмичный геймплей *Bullet Per Minute* полностью ориентирована на взаимодействие игрового окружения и механик с музыкой. Каждое действие в игре, включая атаку, перемещение и перезарядку оружия, выполняется в соответствии с музыкальным ритмом. Это создает динамичную и интенсивную игровую атмосферу, требующую от игрока точного синхронизирования с ритмом игры.

Игра включает в себя случайную генерацию уровней и прогрессирующую систему, что делает каждую новую игру уникальной. Игрок исследует разнообразные уровни, сталкивается с различными врагами и собирает предметы и артефакты, чтобы улучшить своего персонажа и его способности. При этом смерть персонажа имеет последствия, исходя из чего игрок начинает игру с самого начала, исключая возможность пользоваться ранее приобретёнными способностями или предметами. Также это исключает возможность узнать каким образом построен новый уровень, ведь после каждой смерти заново работает и генератор локаций.

*Bullet Per Minute* предлагает динамичные бои в режиме от первого лица. Игроку предстоит сражаться с многочисленными врагами, используя разнообразное оружие, такое как пистолеты, автоматы, ружья и другие. Быстрые рефлексы и точность стрельбы играют важную роль в успешном прохождении игры.

Уникальный арсенал и способности в игре представлены разнообразным выбором оружия и навыками главного героя, которые игрок может использовать в бою. Каждое оружие имеет свои особенности и стиль игры, а способности позволяют проявить креативность в уничтожении врагов и выживании в схватках.

Особенным качеством *Bullet Per Minute* является крайне подходящее музыкально сопровождение, сочетающим различные жанры музыки, включая рок, металл и электронику. Музыка не только создает атмосферу игры, но также помогает игроку синхронизироваться с игровым окружением и действиями, что происходят на экране, ведь основной механикой игры является именно синхронизация действий не только игрока и его персонажа, но и остальных участников сцены с музыкальным сопровождением. В игре представлено разнообразие оружия и способностей, которые можно использовать в битве. От пистолетов и дробовиков до магических артефактов, игрок может настраивать свои тактики и выбирать подходящее оружие для справления с врагами. Кроме того, игроку предоставляются способности, которые можно разблокировать и улучшить в процессе игры, что добавляет стратегический элемент в выбор стиля игры.

Быстрые и интенсивные бои в *Bullet Per Minute* подчёркивают внутриигровую динамику. Игроку приходится стрелять, уклоняться от атак противников и поддерживать ритм игры, чтобы добиться успеха. Каждая смерть является уроком, который показывает на слабые и сильные стороны игрока и позволяет улучшать свои навыки, чтобы стать сильнее и эффективнее в следующей попытке.

В целом, *Bullet Per Minute* предлагает захватывающий и ритмичный игровой опыт, сочетая в себе элементы рогалика и шутера. Быстрые бои, случайная генерация уровней и уникальный стиль делают игру интересной для любителей интенсивных и атмосферных игр.

Таким образом можно пронаблюдать, что совмещение различных игровых жанров на основе систем и механик, характерных исключительно для жанра *Roguelike* является крайне стильным и эффективным приёмом при создании игровых миров. В результате скрещивания жанра шутер с элементами *Roguelike* у команды *Awe Interactive* получилось создать крайне увлекательную и динамичную игровую среду, в которой в качестве видимой основы выступает типичные для жанра шутеров игровые аспекты, такие как стрельба от первого лица, управление персонажем в основном с помощью нажатия клавиш на клавиатуре, большой список различных игровых механик предметов, например арсенал, состоящий из различного вида вооружения, а также акцент на игровом окружении вкупе с гармонично взаимодействующей звуковой составляющей игры, а в качестве заглавной темы – элементы, присущие жанру *Roguelike*. Такими элементами в игре являются, например, крайне обширный список различных характеристик для предметов, сами предметы, от прохождения к прохождению меняющие свои свойства, а также система, при которой игрок в случае поражения, лишается большей части накопленного прогресса, но оставляет себе весомую часть опыта.

Скриншот из данной игры представлен на рисунке 1.6.



Рисунок 1.6 – Пример игры в жанре *Roguelike* шутер

Таким образом на основе анализа программных средств предназначенных для разработки игровых приложений для персональных компьютеров среди рассматриваемых движков, таких как *Unity*, *Godot Engine*, *RPG Maker*, был выбран *Unity.* Данный игровой движок является оптимальным по требуемым техническим характеристика, предоставляемому набору функциональных инструментов и качеству оптимизации проектов для персональных компьютеров различной комплектации, вкупе с удобным и простым интерфейсом, а также наличием большого количества систематизированной и практичной документации.

Также на основе проведённого исследования игрового жанр *Roguelike* с элементами *RPG* были выделены поджанры, основанные и поддерживаемые на механиках и системах жанра *Roguelike*, и черты, такие как динамический геймплей, сюжетное повествование, развитие персонажа игрока, взаимодействие с игровым миром, конечная смерть игрового персонажа. А также были рассмотрены различные представители жанра *Roguelike* имеющие в себе преобладание каждого из поджанров.

Для игрового приложения «*Gone Summer: Overgrowth*» были выбраны такие основные механики присущие жанру *Roguelike*, как система инвентаря, система взаимодействия с окружением и интерактивными объектами сцены, а также система получения игрового опыта в итоге полного или частичного прохождения.

# АРХИТЕКТУРА ИГРОВОГО ПРИЛОЖКЕНИЯ

# «*GONE SUMMER: OVERGROWTH*»

## 2.1 Основные этапы разработки игровых приложений

Условно, весь процесс разработки игрового приложения можно разбить на несколько этапов:

* концептирование;
* прототипирование;
* вертикальный срез;
* производство контента;
* закрытое бета-тестирование;
* открытое бета-тестирование;
* релиз.

Ниже приведён подробный разбор каждого из этапов:

* концептирование. На данном этапе продумывается концепция игры, и проводится начальная проработка игрового дизайна;
* прототипирование. Важный этап проектирования любой игры – это создание прототипа. То, что хорошо выглядит «на бумаге», совершенно не обязательно будет интересно в реальности. Прототип реализуется для оценки основного игрового процесса, проверки различных гипотез, проведения тестов игровых механик, для проверки ключевых технических моментов;
* вертикальный срез. Цель данного этапа – получить минимально возможную полноценную версию игры;
* производство контента. На этом этапе производится достаточное количество контента для первого запуска на внешнюю аудиторию. Реализуются все возможности, запланированные к закрытому бета-тестированию. Это наиболее продолжительный этап;
* закрытое бета-тестирование. На данном этапе продукт впервые демонстрируется достаточно широкой публике, хотя и лояльной продукту или компании. Наиболее важная задача – поиск и исправление гейм-дизайнерских ошибок, проблем игровой логики и устранение критических багов;
* открытое бета-тестирование позволяет проверить тестовую версию приложения в большой группе пользователей, опубликовав её на игровых платформах с открытым доступом. На этом этапе любые пользователи могут присоединяться к открытому бета-тестированию и отправить разработчику личные отзывы об игре;
* релиз. Процесс публикации в магазины достаточно важен для выделения приложения среди конкурентов, а также очень важен для поисковой оптимизации приложения в магазинах и поисковых системах. Подготовка к релизу всегда начинается заранее, чтобы успеть проанализировать подачу конкурентов и создать свой контент, который состоит из названия, краткого описания, развернутого описания, скриншотов с заголовками, свойств, политики конфиденциальности и рекламного видеоролика.

## 2.2 Основные механики игрового приложения «*Gone Summer: Overgrowth*»

В качестве основных механик или функций игрового приложения можно выделить несколько блоков:

* система передвижения персонажа игрока и его взаимодействия с миром;
* система инвентаря и его предметов;
* система быстрого использования предметов из инвентаря;
* система перехода с уровня на уровень;
* системы главное меню и меню паузы;
* анимированные кат-сцены;
* пользовательский интерфейс.

Система передвижения персонажа игрока и его взаимодействие с объектами игрового мира представляет собой набор инструкций по поведению персонажа при нажатии игроком клавиши на клавиатуре или компьютерной мыши. Таким образом при нажатии на клавиши передвижения (*W*, *A*, *S*, *D*) персонаж передвигается по игровому уровню. При нажатии клавиши инвентаря – открывается игровой инвентарь персонажа и показывается его содержимое, при этом повторное нажатие на ту же клавишу закроет игровой инвентарь, при этом сохранив содержимое. При нажатии на левую кнопку мыши, персонаж взмахивает мечом и таким образом атакует противников, стоящих перед ним. После нажатия клавиши взаимодействия происходит проверка взаимодействия, например, если под персонажем лежит предмет, то он поднимает его и помещает в свободную ячейку инвентаря. В случае, если персонаж стоит возле выхода с локации, нажатие кнопки взаимодействия перенесёт его на следующий уровень.

Система инвентаря, предметов и снаряжения представляют собой список отдельных объектов, которые могут быть получены в процессе игры и применяться к персонажу игрока, занимая своё место в соответствующем слоте инвентаря. После применения, предметы дают тот или иной эффект на персонажа или на его окружение. Эти объекты могут храниться в непосредственном доступе для применения, а также могут быть использованы без открытия инвентаря по нажатию клавиши.

Система быстрого использования предметов из инвентаря представляет собой дополнительную систему, связанную с системой инвентаря. Для быстрого использования игрового предмета необходимо нажать на клавишу, соответствующую слоту инвентаря, в котором лежит игровой предмет. Такой вид использования предметов позволяет быстро производить манипуляции с игровым миром.

Система перехода персонажа с одного игрового уровня на другой подразумевает под собой менеджер игрового окружения, который позволяет настраивать определённые ограничения для прохождения дальше. Таким образом возможна установка ограничений таких как: ограничение по врагам, в таком случае уровень не может быть пройден, пока не будут уничтожены все враги, или ограничение по предметам, в таком случае необходимо иметь определённый для прохождения дальше предмет у себя в игровом инвентаре.

Главное меню представляет собой стартовый экран при запуске игрового приложения с возможностью начать игру, проверить информацию об игровом процессе и механиках или выйти из приложения. Меню паузы представляет собой внутриигровое меню, которое позволяет приостановить игровой процесс, вернуться в главное меню либо выйти из игры.

Анимированные кат-сцены представляют собой отрезок игрового времени, во время которого происходят действия, поясняющие игроку основную завязку игрового сюжета, либо показывающие действия, происходящие с игровым персонажем в определённые моменты игрового времени. Анимированной кат сценой можно назвать не только полностью срежиссированный игровой отрезок времени, не и частный игровой момент, например, смерть героя при поражении противником.

Пользовательский интерфейс представляет собой графическое отображение показателей персонажа игрока, например здоровье, показатель очков и т.д. Отображение показателей динамически обновляется в соответствии с изменениями и событиями игрового процесса.

Таким образом игровой процесс состоит из перемещений по игровому миру, сражений с противниками, изучение игровых локаций на предмет содержания каких-либо полезных приспособлений или ключей, а также прохождения на следующий уровень с выполнением условий нынешнего уровня.

## 2.3 Иерархия классов игрового приложения «*Gone Summer: Over-*

## *growth*»

Классы игрового приложения можно разделить на шесть групп:

* классы-менеджеры, отвечающие за набор определённых действий или системных игровых механик;
* классы пользовательского интерфейса или его фрагментов, регулирующие отображение, обновление параметров игрового персонажа во время игры;
* классы, реализующие основу игровой механики;
* класс игровых сущностей-противников;
* класс, наследуемые от типа *ScriptableObject*, позволяющие переносить игровые параметры с уровня на уровень, таким образом сохраняя прогресс игрока;
* базовый пользовательские класс, используемый в рамках реализации игровой механики.

Классы-менеджеры представлены на рисунке 2.2.

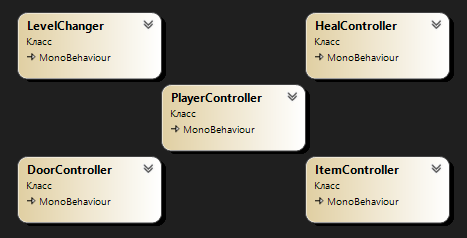


Рисунок 2.1 – Классы-менеджеры

Классы, продемонстрированные на рисунке 2.1, будут описывать следующие внутриигровые менеджеры процессов:

* *LevelChanger* – класс-менеджер отвечающий за доступ к условиям прохождения персонажа на следующий уровень игры. Класс наследуется от стандартного класса движка *MonoBehavior*;
* *DoorController* – класс-менеджер отвечающий за доступ к параметрам и функциям регулирующим доступность открытия двери персонажем, которая может зависеть от нахождения у персонажа в инвентаре определённого предмета или от определённого события, случившегося на игровой сцене. Класс наследуется от стандартного класса движка *MonoBehavior*;
* *ItemController* – класс-менеджер отвечающий за доступ к параметрам и функциям отвечающим за взаимодействие персонажа с предметами, которые могут быть использованы только для достижения задачи на уровне, а не для усиления персонажа. Класс наследуется от стандартного класса движка *MonoBehavior*;
* *HealController* – класс-менеджер отвечающий за доступ к параметрам и функциям отвечающим за применение эффектов использующихся предметов игрока. При этом такие предметы не обязательны для прохождения игрового уровня и влияют лишь на параметры игрока. Класс наследуется от стандартного класса движка *MonoBehavior*;
* *PlayerController* – класс отвечающий за функционал игрового персонажа и объединяющий в себе характеристики других классов, таким образом составляя основную часть игровых программных классов. Данный класс отвечает за отображение корректного количества показателей на экране и их валидного значения на протяжении всей игры, также данный класс относится и к менеджер-классам, так как имеет в себе реализацию проверки взаимодействия игрового персонажа с окружающим его игровым миром, включая проверки при взаимодействии с вражеским персонажем, при взаимодействии с предметами, находящимися на уровнях, при взаимодействии с зонами, назначение которых заключается в требовании от игрового персонажа определённого игрового предмета, находящегося у него в инвентаре. Данный класс также обрабатывает нажатие игроком клавиш на клавиатуре, связанных с игровыми механиками, такими как быстрое использование предметов в слотах инвентаря, а также вызов визуальной составляющей инвентаря на сцену.

Классы пользовательского интерфейса и его отдельных фрагментов представлены на рисунке 2.2.



Рисунок 2.2 – Классы пользовательского интерфейса

Классы, продемонстрированные на рисунке 2.3, будут описывать следующие игровые сущности, параметры и ситуации:

* *MainMenu* – класс отвечающий за функционал главного меню. Данный класс реализует обработку нажатий игроком кнопок в меню. Универсальность класса обуславливает его удобство для применения с разными составляющими. Обработка нажатий не является конечной, т. е. после нажатия некоторых кнопок сцена не будет перезагружаться или сразу же переходить на новую, таким образом достигается Удобность и повышенная производительность игрового приложения во время загрузки и отгрузки игровых сцен. Класс наследуется от стандартного класса движка *MonoBehavior*;
* *PauseMenu* – класс отвечающий за функционал меню паузы. Данный класс реализует проверку на состояние сцены в данный момент времени, а также изменение параметров самой сцены на момент нажатия клавиши паузы на клавиатуре игроком. Класс наследуется от стандартного класса движка *MonoBehavior*;
* *HealthCount* – класс отвечающий за корректное отображение значения показателя здоровья игрового персонажа. Класс наследуется от стандартного класса движка *MonoBehavior*.

Классы игровых сущностей представлены на рисунке 2.4.

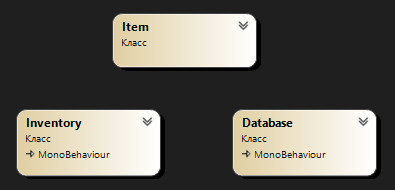


Рисунок 2.3 – Классы основы игровой механики

Классы, продемонстрированные на рисунке 2.3, будут описывать следующие игровые сущности и их характеристики:

* *Database* – класс описывающий игровую базу данных со всеми существующими предметами и связанный с реализуемыми классами *Inventor* и *Item*. Данный класс описывает создание списка игровых предметов с определёнными параметрами и значениями, которые затем могут быть использованы для обработки игровых событий, взаимодействий и ситуаций по мере прохождения игроком уровня. Такая реализация игровой базы данных позволит практично и быстро получать доступ к самым основным параметрам игровых предметов, не меняя их самих как таковых. Таким образом любому игровому предмету может быть присвоен любой независимый параметр, описанный в классе *Item* и при этом другие классы не будут затронуты большим изменением, что способствует работоспособности не только игрового приложения в целом, но и отдельно взятой механики инвентаря, в частности. Класс наследуется от стандартного класса движка *MonoBehavior*;
* *Inventory* – класс описывающий сущность игрового инвентаря и игровых значений, необходимых для реализации хранения предметов игровым персонажем на протяжении игрового уровня. Данный класс описывает реализацию процессов добавления, использования предметов из игрового инвентаря персонажа, а также проверки наличия необходимого предмета в инвентаре в случае связанных процессов, например, при взаимодействии игрового персонажа с окружением, в случае которого требуется наличие определённого игрового предмета в инвентаре персонажа, либо при ручной активации игроком определённого события, связанного с использованием игрового предмета из инвентаря;
* *Item* – класс описывающий сущность игрового предмета, предназначенный для использования и реализации основной механики сохранения игровых предметов на протяжении всей игры в инвентаре игрового персонажа. Данный класс отвечает за корректность и полноту реализации игрового предмета как набора определённых параметров и значений, исключая различную трактовку игровых предметов по мере их разработки, таким образом задавая стандарт игрового предмета.

Класс игровых сущностей-противников представлены на рисунке 2.4.

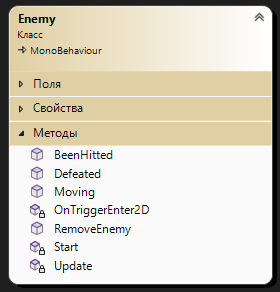


Рисунок 2.4 – Класс игровых сущностей-противников

Класс, продемонстрированный на рисунке 2.4, будет описывать следующие игровые сущности и их характеристики:

* *Enemy* – класс, описывающий сущность игрового персонажа-противника. Данный класс описывает взаимодействие игрового персонажа с окружением и другими игровыми объектами, в частности – с игровым персонажем игрока. Также данный класс взаимодействует с состоянием персонажа-противника на сцене в данный момент. Таким образом применение различной анимации и других игровых ситуаций становится более расширенным, так как класс позволяет изменять состояние игрового персонажа в определённые моменты времени, например, после проигрывания анимации. Данный класс также описывает логику перемещения противников по игровому уровню, если она необходима, а также задаёт начальные игровые параметры для персонажа-противника, что является практичным и удобным доступом к данным сущности. Таким образом игровой персонаж-противник может быть настроен под нужды игрового окружения, а его параметры могут быть изменены динамически, что позволяет реализовать систему комплексных противников, параметры сложности которых могут меняться либо со временем, либо по достижению игроком какого-то определённого прогресса на уровне. Класс наследуется от стандартного класса движка *MonoBehavior*.

Класс наследник типа *ScriptableObject* представлены на рисунке 2.5.

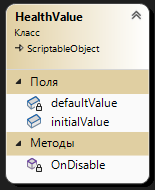


Рисунок 2.5 – Класс наследник типа *ScriptableObject*

Класс, продемонстрированный на рисунке 2.5, является наследником класса *ScriptableObject*, что обеспечивает удобство в использовании сохранения игровых параметров на протяжении всей игровой сессии. Объекты такого класса могут быть использованы не только для сохранения одиночных параметров, но и для переноса целых состояний на другие уровни. Данный класс будет описывать следующие игровые сущности:

* *HealthValue* – класс, описывающий состояние параметров игрового персонажа на момент изменения уровня, что происходит в момент окончания игровой сессии, прохождения уровня или смерти персонажа игрока при каких-либо обстоятельствах.

Базовый пользовательский класс представлен на рисунке 2.6.

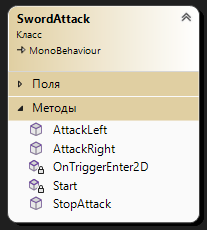


Рисунок 2.6 – Базовый пользовательский класс

Класс, продемонстрированный на рисунке 2.6, будет описывать следующую игровую сущность:

* *SwordAttack* – класс, описывающий сущность атаки. Так как предмет, находящийся на вооружении игрового персонажа, должен иметь определённые параметры, связанные с жанром и реализацией игровой графики, в данном классе описываются необходимые для данного игрового события переменные, такие как направление взамаха оружием (удара), проверка на соприкосновение с любым другим игровым объектом на сцене, например персонажем-противником, а также реализация контроля анимации, в результате чего игровой персонаж не будет создавать алогичные ситуации в процессе анимации, что часто можно наблюдать при редактировании игровых механик. Класс наследуется от стандартного класса движка *MonoBehavior*.

## 2.4 Архитектура пользовательского интерфейса игрового

## приложения «*Gone Summer: Overgrowth*»

Процессу разработки пользовательского интерфейса любого приложения предшествует этап макетирования, на котором разрабатываются схемы предполагаемого внешнего вида отдельных элементов игрового интерфейса, включая открывающиеся с помощью клавиш клавиатуры панели.

Была разработана схема состава интерфейса основного окна игрового меню, которая представлена на рисунке 2.7:

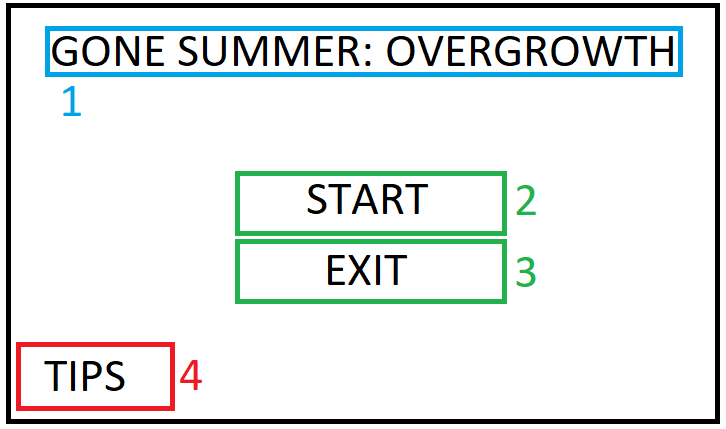


Рисунок 2.7 – Схема элементов игрового меню

Элементы схемы интерфейса, представленной на рисунке 2.7, будут описывать следующие игровые элементы:

* 1 – часть игрового меню, выделенное для демонстрации названия игрового приложения. Выделение места обусловлено привязанной к элементу главного меню анимацией;
* 2 – кнопка, отвечающая за начало игровой сессии;
* 3 – кнопка, отвечающая за выход игрока из игрового приложения из главного меню;
* 4 – кнопка, отвечающая за открытие экрана подсказок в главном меню.

При нажатии на кнопку «2» на рисунке 2.7, игровое меню сменяется экраном начала игры, сразу же стартуя первый уровень игрового приложения.

При нажатии на кнопку «3» на рисунке 2.7, игровое приложение заканчивает свою работу, выходя на рабочий экран компьютера.

При нажатии на кнопку «4» на рисунке 2.7, игровая сцена главного меню изменяется и представляет игровое меню подсказок, содержащих информацию об игровых объектах

Была разработана схема состава интерфейса дополнительного окна игрового меню, содержащая подсказки для игрока, представленная на рисунке 2.8:

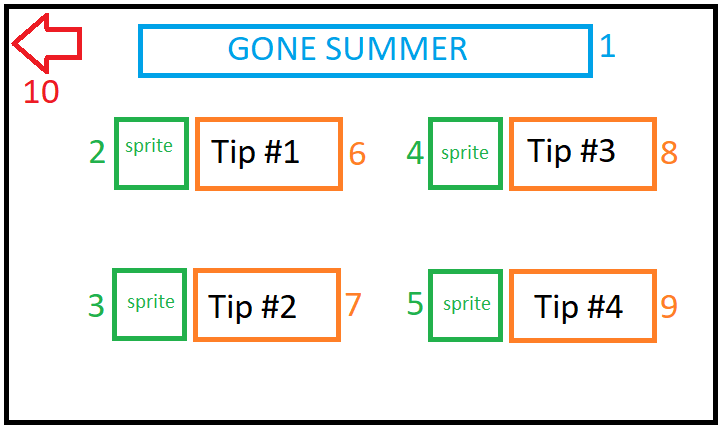


Рисунок 2.8 – Схема элементов игрового меню подсказок

Элементы схемы интерфейса, представленной на рисунке 2.8, будут описывать следующие игровые элементы:

* 1 – часть игрового меню, выделенное для демонстрации названия игрового приложения;
* 2, 3, 4, 5 – элементы визуальной составляющей в виде спрайтов, демонстрирующих игровой предмет игроку;
* 6, 7, 8, 9 – элементы подсказок, реализованных в виде стилизованного текста. Текст располагается рядом с изображением игрового предмета, к которому он относится;
* 10 – кнопка, отвечающая за возвращения игрока в главное меню после ознакомления со всем контентом игрового меню подсказок.

После того, как игрок начнёт игровую сессию, игровое приложение приступит к загрузке нового игрового уровня. Ввиду игрового жанра и стилистики, игровой интерфейс на уровнях придерживается минималистичного дизайна игровых элементов, что помогает в перестановке акцента с интерфейса игрока, на окружающий его персонажа уровень.

Была разработана схема состава интерфейса игрового приложения во время игровой сессии, которая представлена на рисунке 2.9

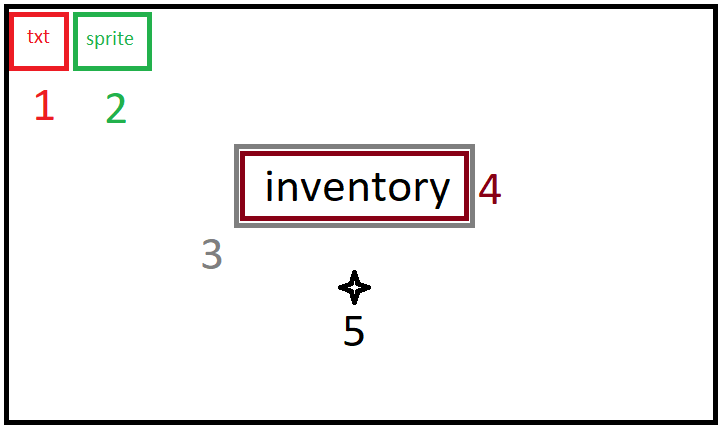


Рисунок 2.8 – Схема интерфейса игрового приложения во время игровой сессии

Элементы игрового интерфейса, представленные на рисунке 2.7, представляют собой:

* 1 – элемент интерфейса, отображающий точное численное количество очков здоровья игрового персонажа;
* 2 – элемент интерфейса, демонстрирующий и объясняющий игроку связанное значение. Данный элемент связан непосредственно с отображаемым количеством очков здоровья персонажа;
* 3 – элемент интерфейса, обозначающий внешний контур инвентаря игрового персонажа. Видимость основной части инвентаря зависит от отображения данного элемента;
* 4 – элемент интерфейса, обозначающий рабочий контур инвентаря игрового персонажа. В нём непосредственно происходят манипуляции с предметами, а также их отображение в слотах инвентаря;
* 5 – центрированная позиция игрока, за которой следит камера. В обычном виде не присутствует на экране, однако может быть свободна настроена для соответствия уровню, на котором находится персонаж игрока.

При нажатии на кнопку «I» на клавиатуре, в зоне 3появляется панель игрового инвентаря персонажа игрока, схема которой изображена на рисунке 2.9:

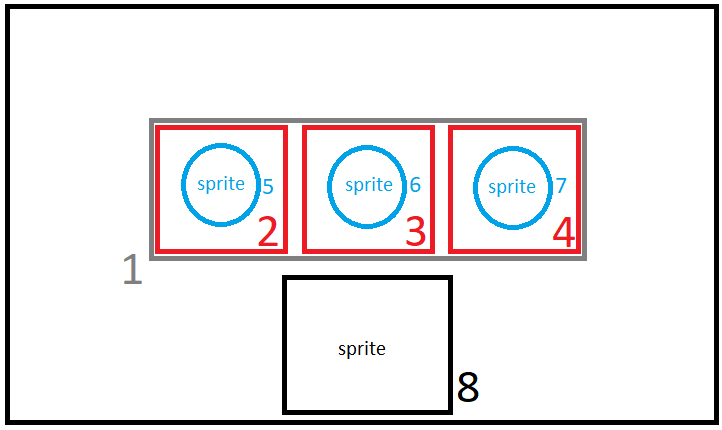


Рисунок 2.9 – Схема панели инвентаря персонажа игрока

Элементы интерфейса панели инвентаря персонажа игрока, представленные на рисунке 2.9, представляют собой:

* 1 – элемент интерфейса, обозначающий внешний контур инвентаря игрового персонажа. Видимость основной части инвентаря зависит от отображения данного элемента;
* 2, 3, 4 – слоты инвентаря, находящиеся на панели игрового объекта. Количество слотов можно менять. Данные элементы интерфейса представляют собой кнопки, считывающие нажатие по себе независимо друг от друга. В зависимости от того, что находится в ячейке, происходит обработка нужного игрового события;
* 5, 6, 7 – изображения игровых предметов, находящихся в соответствующих ячейках инвентаря. Каждое изображение подстраивается под нынешнее разрешение экрана игрока, таким образом избегая визуальных и игровых механических ошибок;
* 8 – область, выделенная под визуальное отображение персонажа игрока. Так как игровое время не останавливается на момент открытия игроком инвентаря, его персонаж должен оставаться видимым, дабы избежать дезориентации игрока после закрытия инвентаря и неожиданных игровых ситуаций, обусловленных неподходящим нахождением игровых элементов интерфейса на главном экран во время активного прохождения.

Макеты игровых окон, панелей, меню были разработаны с учётом ныне существующих образцов качественного и удобного игрового интерфейса, что способствует быстрому привыканию игрока к используемым в игровом приложении интерфейсу, графике и механикам, предписанным данным игровым элементам программно.

# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ

# «*GONE SUMMER: OVERGROWTH*»

## 3.1 Иерархия игрового приложения «*Gone Summer: Overgrowth*»

Игровое приложение разработано при помощи среды разработки *Unity* и состоит из нескольких основных частей, обеспечивающих корректную и быструю работоспособность всего игрового приложения.

Графическая часть представляет собой спрайты, нарисованные при помощи подходящих по инструментарию программ. Игровая механика, которая включает включает в себя методы и классы, позволяющие взаимодействовать с игровым миром. Логическая часть, которая позволяет объектам иметь все необходимые свойства для обработки и реализации основных логических свойств игровых объектов.

По значению и типу большинство классов разделяются на классы, обрабатывающие игровые события, связанные с взаимодействием игровых объектов и сущностей, классы, отвечающие за корректное отображение и обработку визуальной составляющей игры, в том числе и пользовательского интерфейса, а также классы, сохраняющие прогресс игрового персонажа по мере прохождения игры. Во время игровой сессии пользователь в первую очередь ориентируется на игровые показатели, находящиеся на игровом интерфейсе. Данные показатели являются результатом работы различных классов и игровых механик, производящих обработку данных во время игры. На рисунке 3.1 изображена иерархия работы блоков игрового приложения.

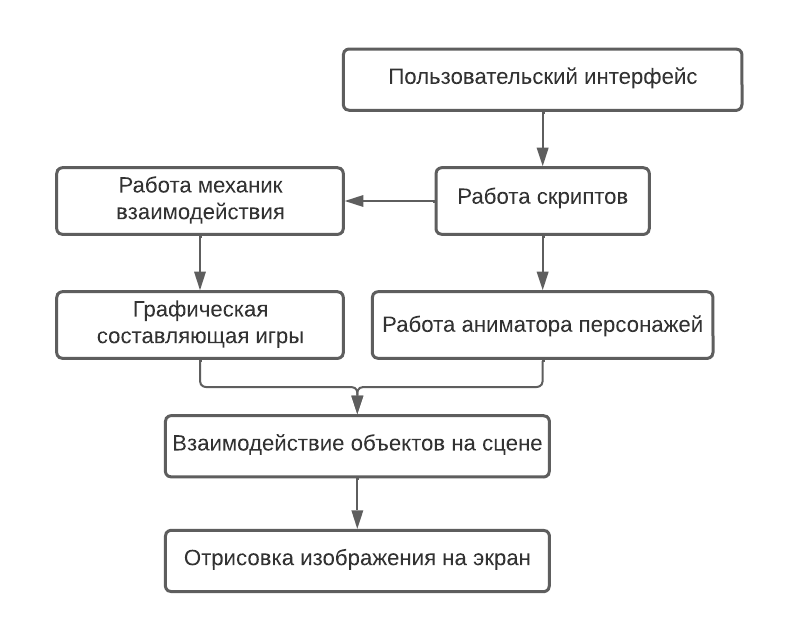


Рисунок 3.1 – Иерархия работы блоков игрового приложения

Программная часть игрового приложения реализована в виде скриптов,

благодаря чему пользователю не нужно вводить никакие данные и как-либо взаимодействовать с кодом программы.

Помимо графической части приложения и ее логики, немаловажной частью является построение анимации при помощи временной шкалы. Аниматор игрока содержит некоторые переменные, которые позволяют связывать некоторые анимации логически, при помощи булевой операции. Происходят большое количество проверок, к примеру, если игрок пойдет влево, то анимация поменяется, а потом встанет в первоначальное положение.

## Принцип работы классов игрового приложения «*Gone Summer:*

## *Overgrowth*»

В данном игровом приложении можно выделить перечень классов, отвечающих за реализацию системы характеристик персонажа игрока:

* *PlayerStats*;
* *Stat*;
* *PlayerManager*;
* *LevelManager*.

***3.2.1*** Класс *PlayerStats* описывает сущность персонажа, а точнее его характеристики и реакции на некоторые игровые события. Данный класс наследуется от класса *CharacterStats* и использует его базовые поля и методы для персонализации стандартной игровой сущности.

Все поля класса *PlayerStats* приведены в таблице 3.1.

Таблица 3.1 – Поля класса *PlayerStats*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| *maxHealth* | *SerializeField* | *public* | *int* | *100* |
| *currentHealth* | *SerializeField* | *public* | *int* | 100 |
| *vitality* | *SerializeField* | *public* | *Stat* | 10 |
| *speed* | *SerializeField* | *public* | *Stat* | 3 |
| *damage* | *SerializeField* | *public* | *Stat* | 5 |
| *armor* | *SerializeField* | *public* | *Stat* | 3 |
| *healthBar* | *SerializeField* | *public* | *Slider* | – |
| *playerStats* | *SerializeField* | *public* | *Stats* | – |
| *money* | *SerializeField* | *public* | *int* | 0 |

Поля класса *PlayerStats* в таблице 3.1 описывают следующую логику:

* *maxHealth* – определяет максимальное количество здоровья персонажа игрока;
* *currentHealth* – определяет количество здоровья персонажа игрока в настоящий момент;
* *vitality* – представляет характеристику отвечающую за живучесть персонажа игрока;
* *speed* – представляет характеристику отвечающую за скорость персонажа игрока;
* *damage* – представляет характеристику отвечающую за урон, наносимый персонажем игрока;
* *armor* – представляет характеристику отвечающую за броню персонажа игрока;
* *healthBar* – представляет префаб шкалы здоровья персонажа игрока;
* *playerStats* – представляет перечисление, хранящее названия и последовательность характеристик персонажа игрока;
* *money* – представляет количество денег персонажа игрока.

Чтобы игрок мог выполнять какие-либо действия и взаимодействовать с окружающими объектами в игре ему, помимо полей, также требуются и методы.

Методы класса *PlayerStats* представлены в таблице 3.2.

Таблица 3.2 – Методы класса *PlayerStats*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| 1 | 2 | 3 | 4 | 5 |
| *Awake* | *–* | *privat* | *void* | *–* |

Продолжение таблицы 3.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *Die* | *–* | *public* | *void* | *–* |
| *OnEquipmentChanged* | *–* | *public* | *void* | *Equipment newItem, Equipment oldItem* |
| *TakeDamage* | *–* | *private* | *void* | *int damage* |

Методы класса *PlayerStats* в таблице 3.2 описывают следующую логику:

* *Awake* – устанавливает количество очков здоровья и привязывает к префабу шкалы здоровья;
* *Die* – определяет сценарий смерти;
* *OnEquipmentChanged* – меняет экипированный и экипируемый предметы местами;
* *TakeDamage* – наносит урон с учётом брони.

***3.2.2*** Класс *Stat* описывает параметры характеристики игрока персонажа.

Все поля класса *Stat* приведены в таблице 3.3.

Таблица 3.3 – Поля класса *Stat*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| *baseValue* | *SerializeField* | *public* | *int* | *–* |
| *modifiers* | *SerializeField* | *privat* | *List*<*int*> | *–* |

Поля класса *Stat* в таблице 3.3 описывают следующую логику:

* *baseValue* – базовое значение характеристики персонажа игрока;
* *modifiers* – список модификаций, влияющих на характеристику и накладываемых предметами экипировки.

Методы класса *Stat* представлены в таблице 3.4.

Таблица 3.4 – Методы класса *Stat*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| *GetValue* | *–* | *public* | *int* | *–* |
| *GetModifiersValue* | *–* | *public* | *int* | *–* |
| *UpValue* | *–* | *public* | *void* | *–* |
| *DownValue* | *–* | *public* | *void* | *–* |
| *AddModifier* | *–* | *public* | *void* | *int modifier* |
| *RemoveModifier* | *–* | *public* | *void* | *int modifier* |

Методы класса *Stat* в таблице 3.4 описывают следующую логику:

* *GetValue* – позволяет получить значение характеристики с учётом модификаций экипировки;
* *GetModifiersValue* – позволяет получить сумму модификаций характеристики;
* *UpValue* – повышает значение характеристики на единицу;
* *DownValue* – понижает значение характеристики на единицу;
* *AddModifier* – добавляет модификацию к характеристике при экипировке соответствующего предмета;
* *RemoveModifier* – снимает модификацию с характеристики при снятии предмета экипировки.

***3.2.3*** Класс *PlayerManager* реализует механизм доступа к характеристикам персонажа игрока и прочие манипуляции вроде воскрешения персонажа в специальном месте после смерти. В данном классе реализуется паттерн проектирования *Singletone* обеспечивающий единственность экземпляра класса и глобальный доступ к нему.

Все поля класса *PlayerManager* приведены в таблице 3.5.

Таблица 3.5 – Поля класса *PlayerManager*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| *player* | *–* | *public* | *GameObject* | *–* |

Поля класса *PlayerManager* в таблице 3.5 описывают следующую логику:

* *player* – комплексное представление графики и функционала, являющееся персонажем игрока.

Методы класса *PlayerManager* представлены в таблице 3.6.

Таблица 3.6 – Методы класса *PlayerManager*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| *KillPlayer* | *–* | *public* | *void* | *–* |
| *GetPlayerStats* | *–* | *public* | *Stat* | *string statName* |

Методы класса *PlayerManager* в таблице 3.6 описывают следующую логику:

* *KillPlayer* – перемещает персонажа игрока на точку возрождения и обновляет некоторые его параметры;
* *GetPlayerStats* – в соответствии с параметром, передаваемым данному методу параметром возвращает определённую характеристику персонажа игрока.

***3.2.4*** Класс *LevelManager* реализует маханизмы получения персонажем игрока опыта и повышения уровня. В данном классе реализуется паттерн проектирования *Singletone* обеспечивающий единственность экземпляра класса и глобальный доступ к нему.

Все поля класса *LevelManager* приведены в таблице 3.7.

Таблица 3.7 – Поля класса *LevelManager*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| *slider* | *SerializeField* | *public* | *Slider* | *–* |
| *onLevelUp* | *–* | *public* | *OnLevelUp* | *–* |
| *levelText* | *SerializeField* | *public* | *Text* | *–* |
| *pointsText* | *SerializeField* | *public* | *Text* | *–* |
| *currentExp* | *SerializeField* | *public* | *int* | *–* |
| *targetExp* | *SerializeField* | *public* | *int* | *–* |
| *currentlevel* | *SerializeField* | *public* | *int* | 1 |
| *currentPoints* | *SerializeField* | *public* | *int* | 0 |
| *buffPoints* | *SerializeField* | *public* | *int* | *–* |

Поля класса *LevelManager* в таблице 3.7 описывают следующую логику:

* *slider* – префаб шкалы опыта;
* *onLevelUp* – делегат, отвечающий за вызов метода обновления параметров при повышении уровня;
* *levelText* – текстовое поле игрового интерфейса, отображающее показатель уровня персонажа игрока;
* *pointsText* – текстовое поле игрового интерфейса, отображающее показатель очков повышения характеристик персонажа игрока;
* *currentExp* – численное значение очков опыта в настоящий момент времени;
* *targetExp* – численное значение очков опыта требуемых для повышения уровня персонажа игрока;
* *currentlevel* – численное значение уровня персонажа игрока;
* *currentPoints* – численное значение имеющихся у персонажа игрока очков повышения характеристик;
* *buffPoints* – численное значение распределённых очков повышения характеристик до подтверждения распределения.

Методы класса *LevelManager* представлены в таблице 3.8.

Таблица 3.8 – Методы класса *LevelManager*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| 1 | 2 | 3 | 4 | 5 |
| *Start* | *–* | *privat* | *void* | *–* |

Продолжение таблицы 3.8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *AddExp* | *–* | *public* | *void* | *int exp* |
| *SetExp* | *–* | *public* | *void* | *–* |
| *SetMaxExp* | *–* | *public* | *void* | *–* |
| *UpdateUI* | *–* | *public* | *void* | *–* |

Методы класса *LevelManager* в таблице 3.8 описывают следующую логику:

* *Start* – при старте устанавливает значения очков опыта и привязывает к шкале опыта;
* *AddExp* – добавляет персонажу игрока установленное параметром количество опыта;
* *SetExp* – отвечает за установку актуального количества очков опыт для шкалы опыта;
* *SetMaxExp* – отвечает за установку максимального количества очков опыт для шкалы опыта при повышении уровня игрока, в соответствии с численным показателем уровнем;
* *UpdateUI* – обновляет элементы графического интерфейса.

## Принцип работы классов системы квестов игрового приложения

## «*Gone Summer: Overgrowth*»

В данном игровом приложении можно выделить перечень классов, отвечающих за реализацию системы квестов персонажа игрока:

* *QuestNew*;
* *QuestManager*;
* *Interactable.*

***3.3.1*** Класс *QuestNew* описывает параметры и методы игрового квеста. Содержит название квеста, текстовое описание цели квеста, показатели прогресса выполнения квеста и список и количество наград, которые получит персонаж игрока за исполнение цели квеста.

Все поля класса *QuestNew* приведены в таблице 3.9.

Таблица 3.9 – Поля класса *QuestNew*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| *isActive* | *SerializeField* | *public* | *bool* | *false* |
| *goal* | *SerializeField* | *public* | *QuestGoal* | *–* |
| *title* | *SerializeField* | *public* | *string* | *–* |
| *description* | *SerializeField* | *public* | *string* | *–* |
| *expReward* | *SerializeField* | *public* | *Int* | *–* |
| *moneyReward* | *SerializeField* | *public* | *int* | *–* |

Поля класса *Stat* в таблице 3.9 описывают следующую логику:

* *isActive* – указывает активен ли в данный момент квест;
* *goal* – содержит значение цели квеста;
* *title* – содержит название квеста;
* *description* – содержит текстовое описание квеста;
* *expReward* – указывает количество опыта, которое получит персонаж игрока за выполнение квеста;
* *moneyReward* – указывает количество денег, которое получит персонаж игрока за выполнение квеста.

Методы класса *QuestNew* представлены в таблице 3.10.

Таблица 3.10 – Методы класса *QuestNew*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| *Complete* | *–* | *public* | *void* | *–* |

Методы класса *QuestNew* в таблице 3.10 описывают следующую логику:

* *Complete* – завершает квест и присваивает персонажу игрока награду за выполнение квеста.

***3.3.2*** Класс *QuestManager* реализует механизм динамического принятия квестов персонажем игрока у специальных игровых объектов, которые выдают квесты. В данном классе реализуется паттерн проектирования *Singletone* обеспечивающий единственность экземпляра класса и глобальный доступ к нему.

Все поля класса *QuestManager* приведены в таблице 3.11.

Таблица 3.11 – Поля класса *EquiomentManager*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| *quest* | *–* | *public* | *Quest* | *–* |
| *onQuestAccept* | *–* | *public* | *OnQuestAccept* | *–* |
| *onQuestDone* | *–* | *public* | *OnQuestDone* | *–* |

Поля класса *QuestManager* в таблице 3.11 описывают следующую логику:

* *quest* – экземпляр квеста, который персонаж игрока принимает к исполнению;
* *onQuestAccept* – делегат, отвечающий за вызов метода и передачу объекта игрового квеста классу, отвечающему за обновление пользовательского интерфейса на панели со списком активных квестов;
* *onQuestDone* – делегат, отвечающий за вызов метода регламентирующего течение завершения квеста.

Методы класса *QuestManager* представлены в таблице 3.12.

Таблица 3.12 – Методы класса *QuestManager*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| *Start* | *–* | *privat* | *void* | *–* |
| *AcceptQuest* | *–* | *public* | *void* | *Quest newQuest* |
| *DoneQuest* | *–* | *public* | *void* | *Quest quest* |

Методы класса *QuestManager* в таблице 3.12 описывают следующую логику:

* *Start* – при старте добавляет делегатам соответствующие методы;
* *AcceptQuest* – добавляет принятый квест;
* *DoneQuest* – описывает течение завершения квеста.

***3.3.3*** Класс *Interactable* реализует базовые механизмы взаимодействия персонажа игрока с объектами игрового мира.

Все поля класса *Interactable* приведены в таблице 3.13.

Таблица 3.13 – Поля класса *Interactable*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| *radius* | *–* | *public* | *float* | *3f* |
| *interactionTransform* | *–* | *public* | *Transform* | *–* |
| *isFocus* | *–* | *public* | *bool* | *false* |
| *player* | *–* | *public* | *Transform* | *–* |
| *hasInteracted* | *–* | *public* | *bool* | *false* |

Поля класса *Interactable* в таблице 3.13 описывают следующую логику:

* *radius* – радиус объекта, в пределах которого персонаж игрока может с ним взаимодействовать;
* *interactionTransform* – представление объекта взаимодействия в сцене;
* *isFocus* – сфокусировано ли внимание на объекте;
* *player* – представление персонажа игрока в сцене;
* *hasInteracted* – было ли произведено взаимодействие.

Методы класса *Interactable* представлены в таблице 3.14.

Таблица 3.14 – Методы класса *Interactable*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| 1 | 2 | 3 | 4 | 5 |
| *Update* | *–* | *private* | *void* | *–* |

Продолжение таблицы 3.14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *Interact* | *–* | *public* | *void* | *–* |
| *OnFocused* | *–* | *public* | *void* | *Transform playerTransform* |
| *OnDefocused* | *–* | *public* | *void* | *–* |
| *OnDrawGizmosSelected* | *–* | *public* | *void* | *–* |

Методы класса *Interactable* в таблице 3.14 описывают следующую логику:

* *Update* – обновляет состояние в каждом кадре, определяя возможность взаимодействия;
* *Interact* – виртуальный метод для определения различного характера взаимодействия для наследуемых классов;
* *OnFocused* – «фокусирует» внимание персонажа игрока на объекте взаимодействия, путём изменения значений параметров;
* *OnDefocused* – снимает внимание персонажа игрока с объекта после взаимодействия, путём изменения значений параметров;
* *OnDrawGizmosSelected* – позволяет видеть графическое отображение радиуса взаимодействия в среде игрового движка.

## Принцип работы классов системы инвентаря и экипировки

## игрового приложения «*Heroic Adventure*»

В данном игровом приложении можно выделить перечень классов, отвечающих за реализацию системы инвентаря и экипировки:

* *Equipment*;
* *EquipmentManager*;
* *Inventory*.

***3.4.1*** Класс *Equipment* описывает сущность предмета экипировки. Данный класс наследуется от класса *Item*, который является наследником типа *ScriptableObject*.

Все поля класса *Equipment* приведены в таблице 3.15.

Таблица 3.15 – Поля класса *Equipment*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| 1 | 2 | 3 | 4 | 5 |
| *equipSlot* | *–* | *public* | *EquipmentSlot* | *–* |
| *armorModifier* | *–* | *public* | *int* |  |
| *damegeModifier* | *–* | *public* | *int* |  |

Продолжение таблицы 3.15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *vitalityModifier* | *–* | *public* | *int* | *–* |
| *speedModifier* | *–* | *public* | *int* |  |

Поля класса *Equipment* в таблице 3.15 описывают следующую логику:

* *equipSlot* – обозначает тип слота экипировки, который должен занимать предмет;
* *armorModifier* – значение модификации, которая при экипировке предмета накладывается на показатель брони;
* *damegeModifier* – значение модификации, которая при экипировке предмета накладывается на показатель урона;
* *vitalityModifier* – значение модификации, которая при экипировке предмета накладывается на показатель живучести;
* *speedModifier* – значение модификации, которая при экипировке предмета накладывается на показатель скорости.

Методы класса *Equipment* представлены в таблице 3.16.

Таблица 3.16 – Методы класса *Equipment*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| *Use* | *–* | *public* | *void* | *–* |

Методы класса *Equipment* в таблице 3.16 описывают следующую логику:

* *Use* – применяет предмет экипировки и удаляет его из слота инвентаря.

***3.4.2*** Класс *EquiomentManager* реализует маханизм динамической смены экипировки. В данном классе реализуется паттерн проектирования *Singletone* обеспечивающий единственность экземпляра класса и глобальный доступ к нему.

Все поля класса *EquiomentManager* приведены в таблице 3.17.

Таблица 3.17 – Поля класса *EquiomentManager*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Значение по умолчанию |
| 1 | 2 | 3 | 4 | 5 |
| *currentEquipment* | *–* | *public* | *Equipment*[] | *–* |
| *onEquipmentChanged* | *–* | *public* | *OnEquiomentChanged* | *–* |
| *onEquipmentUse* | *–* | *public* | *OnEquipmentUse* | *–* |
| *inventory* | *–* | *public* | *Inventory* | *–* |

Поля класса *EquiomentManager* в таблице 3.17 описывают следующую логику:

* *currentEquipment* – массив, хранящий предметы экипировки, которые были применены персонажем игрока и влияют на его характеристики в настоящий момент;
* *onEquipmentChanged* – делегат, отвечающий за вызов метода выполняющего смену экипированного ранее предмета на предмет экипируемый в настоящий момент;
* *onEquipmentUse* – делегат, отвечающий за вызов метода экипирующего предмет на персонажа игрока;
* *inventory* – переменная, ссылающаяся на существующий инвентарь персонажа игрока.

Методы класса *EquiomentManager* представлены в таблице 3.18.

Таблица 3.18 – Методы класса *EquiomentManager*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| *Start* | *–* | *privat* | *void* | *–* |
| *Equip* | *–* | *public* | *void* | *Equipment newItem* |
| *Unequip* | *–* | *public* | *void* | *int slotIndex* |
| *UnequipAll* | *–* | *public* | *void* | *–* |

Методы класса *EquiomentManager* в таблице 3.18 описывают следующую логику:

* *Start* – при старте сцены получает список значений имён ячеек для используемой экипировки и создаёт массив, соответствующие места в котором будут занимать экипированные предметы экипировки согласно значению имени слота;
* *Equip* – экипирует выбранный предмет и применяет на персонажа игрока модификации для характеристик предусмотренные конкретным предметом экипировки;
* *Unequip* – снимает выбранный предмет экипировки и возвращает его в инвентарь;
* *UnequipAll* – снимает все экипированные предметы и возвращает их в инвентарь.

***3.4.3*** Класс *Inventory* реализует механизм хранения предметов в инвентаре. Позволяет поднимать предметы в игровой сцене и выкидывать их обратно. Инвентарь имеет ограниченное количество ячеек для хранения предметов. В данном классе реализуется паттерн проектирования *Singletone* обеспечивающий единственность экземпляра класса и глобальный доступ к нему.

Все поля класса *Inventory* приведены в таблице 3.19.

Таблица 3.19 – Поля класса *Inventory*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Модификатор доступа | Тип | Значение по умолчанию |
| *onItemChangedCallback* | *–* | *public* | *OnItemChanged* | *–* |
| *space* | *–* | *public* | *int* | *20* |
| *items* | *–* | *public* | *List<Item>* | *–* |

Поля класса *Inventory* в таблице 3.19 описывают следующую логику:

* *onItemChangedCallback* – делегат, отвечающий за обновление пользовательского интерфейса при добавлении или удалении предмета;
* *space* – количество ячеек инвентаря;
* *items* – список предметов, находящихся в инвентаре.

Методы класса *Inventory* представлены в таблице 3.20.

Таблица 3.20 – Методы класса *Inventory*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Атрибут | Мод. доступа | Тип | Параметры |
| *Add* | *–* | *public* | *bool* | *Item item* |
| *Remove* | *–* | *public* | *void* | *Item item* |

Методы класса *Inventory* в таблице 3.20 описывают следующую логику:

* *Add* – добавляет предмет, передающийся в параметре в свободную ячейку инвентаря;
* *Remove* – удаляет предмет из инвентаря.

# ВАЛИДАЦИЯ И ВЕРИФИКАЦИЯ РЕЗУЛЬТАТОВ РАБОТЫ

# ИГРОВОГО ПРИЛОЖЕНИЯ «*HEROIC ADVENTURE*»

## Виды тестирования игр

Перед публикацией финальной версии разработанное игровое приложение было несколько раз протестировано с помощью различных методов тестирования.

Существует перечень видов тестирования игровых приложений:

* функциональное тестирование – тестирование, целью которого является выявление отклонений от функциональных требований к игровому приложению, его фрагменту или отдельной функции [7]. Данное тестирование сводится к многократному запуску и прохождению игры, выявлению неполадок, багов либо несоответствий с заявленным функционалом и поиску методов или способов решения выявленных проблем и устранения неполадок. Данный тип тестирования применялся на протяжении всех этапов разработки игрового приложения и является основным;
* нагрузочное тестирование – тестирование игровых приложений, при котором воссоздаются ситуации, которые требуют большой вычислительной мощности и максимально нагружают устройство, на котором запущено игровое приложение. Таким образом, тестировщик может проверить производительность системы в стрессовой ситуации или на потенциально слабых устройствах. Данный метод облегчает поиск потенциально ненадёжных фрагментов кода, которые требуют оптимизации. Для данного вида тестирования моделировались ситуации, которые потенциально могли перегружать возможности мобильного устройства. Например, в игровую сцену добавлялось большое количество предметов экипировки и противников, что моделировало ситуацию перегрузки сцены графикой, если игрок решил бы пробежаться по локациям и собрать за собой большое количество преследующих его противников для более острых ощущений от игрового процесса. Благодаря подобным тестам ужалось убедиться в исправном функционировании игрового приложения при избытке объектов и графики в сцене;
* тестирование на совместимость – вид нефункционального тестирования, основной целью которого является проверка корректной работы продукта в определенном окружении. Проверке подвергалась возможность запуска игрового приложения на мобильных устройствах, имеющих различные технически характеристики.

## Функциональное тестирование

Как было упомянуто выше функциональное тестирование – это вид тестирования ПО, цель которого в проверке реализуемости функциональных требований, а именно способности ПО в определенных условиях корректно решать поставленные перед ним задачи. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает. Тестирование было проведено путем запуска игрового приложения при определенных тестировщиком условиях для наблюдения и оценки корректности работы различных аспектов игрового приложения. В процессе функционального тестирования производился анализ приложения для выявления несоответствий между существующими деталями функционирования игрового приложения и требованиями к их корректному функционированию.

Тест № 1. Цель: проверить запуск приложения.

Ожидаемый результат: игра должна запускаться на мобильных устройствах с ОС *Android* 4.0 и выше.

Вывод: ожидаемый и полученный результат совпали, цель теста достигнута.

Тест № 2. Цель: проверить соответствие отображения игрового интерфейса на устройстве с отображением в среде разработки.

Ожидаемый результат: на экране отображен интерфейс игры, отображаются: панель игрового интерфейса, содержащая шкалу здоровья, шкалу опыта, кнопки вызова панелей инвентаря, характеристик и квестов, а также кнопка, вызывающая меню паузы. Все элементы интерфейса выглядят уместно и не подвергнуты деформации.

Вывод: ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 3. Цель: проверить работоспособность кнопок главного меню.

Ожидаемый результат: в зависимости от нажатой в главном меню кнопки происходит запуск игровой сцены, отображение окна регулировки звука или выход из приложения.

Вывод: ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 4. Цель: проверить соответствие отображения игрового интерфейса на устройстве с отображением в среде разработки.

Ожидаемый результат: на экране отображен интерфейс игры, отображаются: панель игрового интерфейса, содержащая шкалу здоровья, шкалу опыта, кнопки вызова панелей инвентаря, характеристик и квестов, а также кнопка, вызывающая меню паузы. Все элементы интерфейса выглядят уместно и не подвергнуты деформации.

Вывод: ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 5. Цель: проверить работоспособность меню паузы.

Ожидаемый результат: при нажатии кнопки паузы время в игре останавливается, появляется панель функциональными с кнопками меню.

Вывод: ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 6. Цель: проверить работоспособность передвижения игрока по игровой сцене.

Ожидаемый результат: при нажатии на точку на земле игроком персонаж должен переместиться в указанное место кратчайшим путём, при этом персонаж игрока должен быть повёрнут лицом в сторону движения. Также персонаж игрока не должен проходить сквозь объекты сцены, блокирующие проход.

Вывод: персонаж игрока успешно перемещается по игровой сцене, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 7. Цель: проверить корректность работоспособности атаки и получения урона персонажем.

Ожидаемый результат: персонаж игрока при нажатии на противника должен наносить ему урон, а также терять очки здоровья при ответных атаках.

Вывод: игрок атакует противника, наносит урон и теряет очки здоровья при ответной атаке, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 8. Цель: проверить корректность функционирования механизма смерти и возрождения персонажа.

Ожидаемый результат: когда здоровье персонажа становится равным нулю, он должен быть перемещён в точку возрождения.

Вывод: персонаж умирает и возрождается корректно, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 9. Цель: проверить корректность функционирования механизма поднятия предметов, их появления в инвентаре и удаления из него.

Ожидаемый результат: после двойного нажатия на лежащий на земле предмет его трёхмерное представление в сцене должно исчезнуть, а в инвентаре при наличии свободного слота появиться иконка соответствующего предмета. Так же при нажатии крестика в углу заполненного слота иконка должна исчезнуть, очистив слот, а в игровой сцене появиться трёхмерное представление предмета.

Вывод: предметы корректно добавляются в инвентарь и удаляются из него, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 10. Цель: проверить корректность начисления очков опыта персонажу игрока.

Ожидаемый результат: очки опыта должны начисляться персонажу игрока в следующих случаях:

* при убийстве противника. Очки начисляются в соответствии с типом противника;
* при завершении квеста. Очки начисляются в соответствии с прописанной в квесте наградой

Вывод: на протяжении игровой сессии корректно начислялись очки опыта при убийстве противников и выполнении квестов, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 11. Цель: проверить возможность достижения нового уровня и присвоение очков характеристик.

Ожидаемый результат: при получении достаточного количества очков опыта происходит повышение уровня персонажа игрока, ему присваивается очко для повышения характеристик

Вывод: при получении достаточного количества опыта повышается уровень персонажа, что позволяет повысить одну из его характеристик с помощью очков повышения, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 12. Цель: проверить корректность улучшения характеристик персонажа игрока при использовании очков повышения. Проверить работу кнопок «+» и «-» в рамках имеющихся очков повышения характеристик и подтвердить изменения с помощью соответствующей кнопки.

Ожидаемый результат: при наличии очков повышения характеристик рядом с численным значением на панели характеристик персонажа появляется кнопка «+», нажатие на которую прибавляет единицу к выбранной характеристике, после чего появляется кнопка «-», позволяющая вернуть распределённое очко. Любые изменения численного показателя характеристик возможны только при наличии очков повышения характеристик и до нажатия кнопки подтверждения распределения.

Вывод: при наличии очков повышения характеристик и нажатии соответствующих кнопок происходит изменение численного значения повышаемой характеристики персонажа игрока, которое вступает в силу и не может быть изменено после нажатия кнопки подтверждения распределения, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 13. Цель: проверить правильное воспроизведение звуковых эффектов.

Ожидаемый результат: звуковые эффекты воспроизводятся в соответствии с прописанным сценарием.

Вывод: звуковые эффекты воспроизводятся корректно и в необходимый момент игровой сессии, ожидаемый и полученный результат совпадают, цель теста достигнута.

Тест № 14. Цель: проверить корректное обновление графического интерфейса на различных панелях.

Ожидаемый результат: детали графического интерфейса должны изменяться в случае:

* панель инвентаря – при сборе, экипировке или удалении предмета из инвентаря;
* панель характеристик персонажа – при экипировке снаряжения, которое повышает характеристики, при повышении уровня, при распределении очков повышения характеристик;
* панель квестов – при принятии квеста, в процессе прогресса квеста, после завершения квеста;
* шкала здоровья – при получении урона персонажем игрока, при повышении соответствующей характеристики при восстановлении здоровья;
* шкала опыта – при получении очков опыта путём победы над противниками и выполнения квестов;

Вывод: все элементы графического интерфейса, а именно панель инвентаря, панель характеристик персонажа, панель квестов, шкала здоровья, шкала опыта корректно обновляется после определённых действий и в соответствующих обстоятельствах, ожидаемый и полученный результат совпадают, цель достигнута.

## Юзабилити-тестирование

Юзабилити-тестирование – это исследование, выполняемое с целью определения степени удобства и понятности приложения для его потенциальных пользователей, другими словами, это метод оценки удобства продукта в использовании, основанный на привлечении пользователей в качестве тестировщиков, испытателей и суммировании полученных от них выводов. Юзабилити-тестирование является распространенным подходом при тестировании приложений.

***4.3.1*** Метод исследования «Мысли вслух». «Мысли вслух» – один из самых популярных методов юзабилити-тестирования был выбран в качестве метода исследования. Метод «Мысли вслух» позволяет понять, как пользователь подходит к интерфейсу и какими соображениями он руководствуется, используя функционал приложения.

***4.3.2*** План проведения тестирования:

* планирование – разработка заданий и набор участников тестирования;
* проведение тестирования;
* анализ полученных данных.

***4.3.3*** Методика проведения тестирования.

Участник тестирования получает список заданий целиком. Участник должен ознакомиться со списком, после чего приступить к самостоятельному выполнению, проговаривая свои мысли, чувства, мнения, ощущения, возникающие в процессе взаимодействия с игрой.

***4.3.4*** Протокол заданий. Задания для респондентов продемонстрированы в таблице 4.1.

Таблица 4.1 – Протокол заданий

|  |  |  |
| --- | --- | --- |
| № задания | Отправная точка | Задание |
| 1 | 2 | 3 |
| 1 | Сцена меню | Ознакомиться с главным меню |
| 2 | Сцена меню | Запустить игровую сцену |
| 3 | Игровая сцена | Переместить персонажа |
| 4 | Игровая сцена | Открыть и закрыть вкладки инвентаря, характеристик и квестов |
| 5 | Игровая сцена | Поднять предмет |
| 6 | Игровая сцена | Экипировать предмет |
| 7 | Игровая сцена | Уничтожить противника |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 8 | Игровая сцена | Повысить любую характеристику после получения нового уровня |
| 9 | Игровая сцена | Получить урон от противника, умереть, воскреснуть и зафиксировать изменение характеристик с позицией возрождения |
| 10 | Игровая сцена | Взять квест и проверить его корректное отображение во вкладке квестов |

***4.3.5*** Респонденты.

Было принято решение отбирать для тестирования людей различного возраста, увлечений и игрового опыта

Все респонденты успешно справились с задачами, отметив удобство интерфейса. Гармоничность оформления и интересный, разнообразный геймплей.

## Итоги тестирования

В результате тестирования было установлено соответствие разработанного приложения и исходных данных, указанных на этапе проектирования.

Список устройств, используемых в тестировании, и их технические характеристики указаны в таблице 4.2.

Таблица 4.2 – Устройства, используемые в тестировании

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Устройство | ОС | *CPU* | *GPU* | Объем оперативной памяти |
| 1 | 2 | *3* | *4* | 5 |
| *Xiaomi Redmi Note* 8 | *Android* 9.0 *Pie* | *Qualcomm* *Snapdragon* 665 | *Adreno* 610 | 4 Гб |
| *Xiaomi* *Mi* 4*s* | *Android* 5.1 *Lollipop* | *Qualcomm* *Snapdragon* 808 *MSM*8992 | *Adreno* 418 | 3 Гб |
| *Samsung* *Galaxy* *A*20*e* | *Android* 9.0 *Pie* | *Samsung* *Exynos* 7 *Octa* 788 | *Samsung* *Exynos* 7 *Octa* 788 | 3 ГБ |
| *Xiaomi* *Redmi* *Note* 7 | *Android* 9.0 *Pie* | *Qualcomm* *Snapdragon* 660 *MSM*8956 | *Adreno* 512 | 4 ГБ |

Продолжение таблицы 4.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *Xiaomi Redmi* 9 | *Android* 10.0 | *Mediatek Helio* *G*80 | *Mali*-*G*52 *MC*2 | 4 Гб |
| *Xiaomi Redmi* 7 | *Android* 9.0 *Pie* | *Qualcomm* *Snapdragon* 632 | *Adreno* 506 | 3 Гб |
| *Samsung Galaxy* *A*51 | *Android* 10.0 | *Samsung* *Exynos* 9611 | *Mali*-*G*72 *MP*3 | 6 Гб |

Несмотря на то, что список устройств, незначителен – стоит отметить широкое разнообразие конфигураций их технических характеристик. Использование лишь такого малого количества компьютеров позволило найти, идентифицировать и решить некоторые ошибки:

− отсутствие адаптивности интерфейса под различные разрешения экрана;

− отсутствие адаптивности анимаций к различной величине *FPS*;

− проблема отображения текстур на некоторых устройствах.

Что касается скорости работы, то все устройства позволили пользоваться приложением с приемлемым *FPS*. На всех устройствах этот показатель не падал ниже 30 кадров, что достаточно для комфортной игры. Следовательно, разработанное мобильное игровое приложение способно запускаться на большей части современных мобильных устройств на базе ОС *Android*. Минимальные системные требования довольно низкие.

За время проведения тестов, возникали ошибки в основном касающиеся визуальной составляющей.

Из всего вышеупомянутого можно сказать, что качество и стабильность итогового приложения находится на достаточно высоком уровне. Данное тестирование показывает, что на текущем этапе реализации проекта, игра соответствует большинству функциональных требований выделенных на этапе проектирования приложения.

# ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ИГРОВОГО

# ПРИЛОЖЕНИЯ «*HEROIC ADVENTURE*»

## 5.1 Технико-экономическое обоснование целесообразности разработки

## программного продукта и оценка его конкурентоспособности

Техническая прогрессивность разрабатываемого программного продукта определяется коэффициентом эквивалентности (). Расчет данного коэффициента осуществляется путем сравнения технического уровня разрабатываемого программного продукта по отношению к эталонному уровню программного продукта данного направления с использованием формулы (Д.1). Результат расчета коэффициента эквивалентности приведен в таблице 5.1. Полученное значение коэффициента эквивалентности больше единицы, следовательно, разрабатываемый программный продукт является технически прогрессивным.

Таблица 5.1 – Расчет коэффициентов эквивалентности

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Наименование  параметра | Вес  парамет­­ра, | Значения  параметра | | |  |  |  |  |
|  |  |  |
| Объем памяти | 0,1 | 6 | 8 | 5 | 1,2 | 1,6 | 0,12 | 0,16 |
| Время обработки  данных | 0,4 | 0,4 | 0,6 | 0,2 | 2 | 3 | 0,8 | 1,2 |
| Отказы | 0,5 | 1 | 2 | 1 | 1 | 2 | 0,5 | 1 |
| Итого | | | | | | | 2,36 | 1,42 |
| Коэффициент эквивалентности | | | | | | | 2,36/1,42=1,66 | |

Далее рассчитывается коэффициент изменения функциональных возможностей () нового программного обеспечения по формуле (Д.3). Расчет коэффициента изменения функциональных возможностей нового программного продукта приведен в таблице 5.2.

Таблица 5.2 – Расчет коэффициента изменения функциональных возможностей

|  |  |  |
| --- | --- | --- |
| Наименование показателя | Балльная оценка базового ПП | Балльная оценка нового ПП |
| 1 | 2 | 3 |
| Объем памяти | 4 | 4 |
| Быстродействие | 3 | 4 |
| Удобство интерфейса | 3 | 5 |
| Степень утомляемости | 3 | 2 |
| Производительность труда | 2 | 4 |

Продолжение таблицы 5.2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Итого | 15 | 19 |
| Коэффициент функциональных возможностей | 19/15 = 1,27 | |

Новый программный продукт превосходит по своим функциональным возможностям базовый в 1,27 раза.

Конкурентоспособность нового программного продукта по отношению к базовому можно оценить с помощью интегрального коэффициента конкурентоспособности, учитывающего все ранее рассчитанные показатели. Для расчета конкурентоспособности нового программного продукта по отношению к базовому была использована соответствующая формула (Д.4).

Коэффициент цены потребления рассчитывается как отношение договорной цены нового программного продукта к договорной цене базового (таблица 5.3).

Таблица 5.3 – Расчет уровня конкурентоспособности нового ПП

|  |  |
| --- | --- |
| Коэффициенты | Значение |
| Коэффициент эквивалентности | 1,66 |
| Коэффициент изменения функциональных возможностей | 1,27 |
| Коэффициент соответствия нормативам | 1 |
| Коэффициент цены потребления | 0,93 |
| Интегральный коэффициент конкурентоспособности |  |

Интегральный коэффициент конкурентоспособности () больше единицы, это значит, что новый программный продукт является более конкурентоспособным, чем базовый.

## 5.2 Оценка трудоёмкости работ по созданию программного

## обеспечения

Общий объем программного обеспечения () определяется исходя из количества и объема функций, реализуемых программой, по каталогу функций программного обеспечения по формуле (Д.5).

Уточненный объем программного обеспечения () определяется по формуле (Д.6).

Результаты произведённых вычислений объема функций ПО представлены в таблице 5.4.

Таблица 5.4 – Перечень и объем функций ПО

|  |  |  |  |
| --- | --- | --- | --- |
| Код  функции | Наименование (содержание) функции | Объем функции строк исходного кода | |
| по каталогу  () | Уточненный  () |
| 101 | Организация ввода информации | 140 | 160 |
| 102 | Контроль, предварительная обработка и ввод информации | 470 | 410 |
| 109 | Управление вводом-выводом | 2100 | 520 |
| 506 | Обработка ошибочных сбойных ситуаций | 1250 | 650 |
| 702 | Расчетные задачи | 13200 | 2050 |
| 707 | Графический вывод результатов | 620 | 510 |
| 709 | Изменение состояния ресурсов в интерактивном режиме | 570 | 420 |
| Итого | | 18350 | 4720 |

ПО относится к третьей категории сложности.

На основании принятого к расчету (уточненного) объема () и категории сложности ПО определяется нормативная трудоемкость ПО () выполняемых работ, представлена в таблице 5.5.

Таблица 5.5 – Нормативная трудоемкость на разработку ПО ()

|  |  |  |
| --- | --- | --- |
| Уточнённый объем, | 3-я категория сложности ПО | Номер нормы |
| 4720 | 197 | 47 |

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого ПО, учитываются посредством коэффициента повышения сложности ПО (). рассчитывается по формуле (Д.7):

.

Влияние фактора новизны на трудоемкость учитывается путем умножения нормативной трудоемкости на соответствующий коэффициент, учитывающий новизну ПО ().

Разработанная программа обладает категорией новизны Б, а значение  = 0,72.

Степень использования в разрабатываемом ПО стандартных модулей определяется их удельным весом в общем объеме ПО.

В данном программном комплексе используется до 45% стандартных модулей, что соответствует значению коэффициента = 0,65.

Программный модуль разработан с помощью объектно-ориентированных технологий, что соответствует коэффициенту, учитывающему средства разработки ПО, = 0,55. Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО определяются с учетом установленной категории новизны ПО и приведены в таблице 5.6.

Таблица 5.6 – Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Категория  новизны ПО | Без применения *CASE*-технологий | | | | |
| Стадии разработки ПО | | | | |
| ТЗ | ЭП | ТП | РП | ВН |
| Значения коэффициентов | | | | |
|  |  |  |  |  |
| Б | 0,10 | 0,20 | 0,30 | 0,30 | 0,10 |

Нормативная трудоемкость ПО () выполняемых работ по стадиям разработки корректируется с учетом коэффициентов: коэффициента повышения сложности ПО (), коэффициента новизны ПО (), коэффициента степени использования стандартных модулей (), коэффициент средств разработки ПО (). Данные коэффициенты определяются для стадии ТЗ по формуле (Д.8), для стадии ЭП по формуле (Д.9), для стадии ТП по формуле (Д.10), для стадии РП по формуле (Д.11), для стадии ВН по формуле (Д.12). Коэффициенты , и вводятся на всех стадиях разработки, а коэффициент вводится только на стадии РП.

Для уменьшения общей трудоёмкости разработки введем коэффициент используемости разработанного ПО, который равен 0,1.

чел.-дн.,

чел.-дн.,

чел.-дн.,

чел.-дн.,

чел.-дн.

Общая трудоемкость разработки программного обеспечения () определяется суммированием нормативной (скорректированной) трудоемкости программного обеспечения на всех стадиях разработки программного обеспечения по формуле (Д.13):

чел.-дн.

Параметры расчетов по определению нормативной и скорректированной трудоемкости программного обеспечения на всех стадиях разработки и общей трудоемкости разработки программного обеспечения представлены в таблице 5.7.

Таблица 5.7 – Расчет общей трудоемкости разработки ПО

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии разработки | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |
| Общий объем ПО (), кол-во строк *LOC* | – | – | – | – | – | 18350 |
| Общий уточненный объем ПО (), кол-во строк *LOC* | – | – | – | – | – | 4720 |
| Категория сложности разрабатываемого ПО | – | – | – | – | – | 3 |
| Нормативная трудоемкость разработки ПО (), чел.-дн. | – | – | – | – | – | 197 |
| Коэффициент повышения сложности ПО () | 1,06 | 1,06 | 1,06 | 1,06 | 1,06 | – |
| Коэффициент, учитывающий новизну ПО () | 0,72 | 0, 72 | 0, 72 | 0, 72 | 0, 72 | – |
| Коэффициент, учитывающий степень использования стандартных модулей () | – | – | – | 0,65 | – | – |
| Коэффициент, учитывающий средства разработки ПО() | 0,55 | 0,55 | 0,55 | 0,55 | 0,55 | – |
| Коэффициенты удельных весов трудоемкости стадий разработки ПО (,,,,) | 0,10 | 0,20 | 0,30 | 0,30 | 0,10 | 1,0 |
| Распределение скорректированной (с учетом , , ,) трудоемкости ПО по стадиям, чел.-дн. | 8,27 | 16,54 | 24,81 | 16,13 | 8,27 | 75 |
| Общая трудоемкость разработки ПО (), чел.-дн. | – | – | – | – | – | 75 |

Таким образом были вычислены параметры расчетов по определению нормативной и скорректированной трудоемкости разработки программного обеспечения на всех стадиях разработки, а также общая трудоемкость разработки программного обеспечения.

## 5.3 Расчёт затрат на разработку программного продукта

Суммарные затраты на разработку программного обеспечения () определяются по формуле (Д.14). Параметры расчета производственных затрат на разработку программного обеспечения приведены в таблице 5.8.

Таблица 5.8 – Параметры расчета производственных затрат на разработку ПО

|  |  |  |
| --- | --- | --- |
| Параметр | Единица измерения | Значение |
| Тарифная ставка 1-го разряда | руб. | 269,1 |
| Доплата за стаж | % (руб.) | 19,5 (93,57) |
| Доплата по контракту | % | 0,5 от оклада |
| Разряд разработчика | – | 11 |
| Тарифный коэффициент | – | 2,81 |
| Коэффициент | – | 1,8 |
| Норматив отчислений на доп. зарплату разработчиков () | % | 20 |
| Численность обслуживающего персонала | чел. | 1 |
| Средняя годовая ставка арендных платежей () (по результатам мониторинга предложений по аренде помещений) | руб./м2 | 13,69 |
| Площадь помещения (*S*) | м2 | 12 |
| Количество ПЭВМ () | шт. | 1 |
| Затраты на приобретение единицы ПЭВМ | руб. | 1200 |
| Стоимость одного кВт-часа электроэнергии () | руб. | 0,390852 |
| Коэффициент потерь рабочего времени () | – | 0,2 |
| Затраты на технологию () | руб. | – |
| Норматив общепроизводственных затрат () | % | 10 |
| Норматив непроизводственных затрат () | % | 5 |

Расходы на оплату труда разработчиков с отчислениями () определяются по формуле (Д.15).

Основная заработная плата разработчиков рассчитывается по формуле (Д.16).

Средняя часовая тарифная ставка определяется по формуле (Д.17).

Часовая тарифная ставка определяется путем деления месячной тарифной ставки на установленный при восьмичасовом рабочем дне фонд рабочего времени 170,83 ч (), формула (Д.18).

руб.

Дополнительная заработная плата рассчитывается по формуле (Д.19):

руб.

Отчисления от основной и дополнительной заработной платы рассчитываются по формуле (Д.20):

Годовые затраты на аренду помещения определяются по формуле (Д.27):

Сумма годовых амортизационных отчислений () определяется по формуле (Д.28):

Стоимость электроэнергии, потребляемой за год, определяется по формуле (Д.29).

Действительный годовой фонд времени работы ПЭВМ () рассчитывается по формуле (Д.30):

ч.,

Затраты на материалы (), необходимые для обеспечения нормальной работы ПЭВМ, составляют около 1% от балансовой стоимости ЭВМ, и определяются по формуле (Д.31):

руб.

Затраты на текущий и профилактический ремонт () принимаются равными 5% от балансовой стоимости ЭВМ и вычисляются по формуле (Д.32):

руб.

Прочие затраты, связанные с эксплуатацией ЭВМ (), состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций и составляют 5 % от балансовой стоимости и определяются по формуле (Д.33):

руб.

Для расчета машинного времени ЭВМ ( в часах), необходимого для разработки и отладки проекта специалистом, следует использовать формулу (Д.34):

ч.,

руб.

Расчет затрат на изготовление эталонного экземпляра () осуществляется по формуле (Д.35):

руб.

Так же рассчитываются затраты на материалы () по формуле (Д.36):

руб.

Общепроизводственные затраты () рассчитываются по формуле (Д.37):

И наконец, непроизводственные затраты () рассчитываются по формуле (Д.38):

Итого получаем суммарные затраты на разработку:

руб.

## 5.4 Расчёт договорной цены и частных экономических эффектов от

## производства и использования программного продукта

Оптовая цена программного продукта () определяется по., формулам (Д.39) и (Д.40):

руб.

Прогнозируемая отпускная цена программного продукта рассчитывается по формуле (Д.41). Налог на добавленную стоимость () рассчитывается по формуле (Д.42):

руб.

Годовой экономический эффект от производства нового программного обеспечения () определяется по формуле (Д.43).

После просмотра и анализа аналогов программного продукта можно сделать вывод, что цена на разработку данного программного продукта коррелирует в диапазоне от 9000 до 11000 руб., значит в среднем цена на аналогичный базовый программный продукт составляет 10400 руб.

Расчет годового экономического эффекта от производства нового программного продукта представлен в таблице 5.9.

Таблица 5.9 – Расчет годового экономического эффекта от производства нового программного продукта

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование параметра | Условн. обознач. | Базовый вариант | Новый вариант |
| 1 | 2 | 3 | 4 |
| Оптовая цена, руб. |  | 10400 |  |
| Норматив рентабельности |  | 0,15 | 0,15 |
| Себестоимость производства, руб. |  | 10400/(1+0,15) =  = 9043,48 |  |

Продолжение таблицы 5.9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | |
| Удельные капитальные вложения, руб. |  | 1200 | | |
| Нормативный коэффициент капитальных вложений |  | 0,15 | | 0,15 |
| Расчет | | | | |
| Удельные приведенные затраты на производство ПО, руб. |  | 9043,48+1200·0,15 = = 9223,48 | |  |
| Годовой экономический эффект от производства нового ПП, руб. |  | 9223,48–6990,97 =  = 2232,51 | | |
| Прирост прибыли, руб. |  | (8389,16 – 6990,97) – (10400 – – 9043,48) = 41,67 | | |

В таблице 5.10 представлены значения прибыли и рентабельности.

Таблица 5.10 – Параметры эффективности проекта

|  |  |
| --- | --- |
| Наименование параметра | Значение |
| Прибыль, руб. |  |
| Уровень рентабельности, % | 30 |

Срок окупаемости программного продукта (формула Д.44):

Исходя из расчётов, можно сделать вывод о том, что ПП окупится на четвёртом году эксплуатации.

Таким образом, по результатам проведенной оценки, установлено, что реализация проекта обоснована и является экономически целесообразной. Об этом свидетельствуют экономический эффект от производства нового программного продукта (= 2232,51 руб.). Все данные приведены в итоговой таблице Д.1.

# ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ

## Причины возникновения, параметры и физические

## характеристики вибрации

Вибрация (лат. *Vibratio* – колебание, дрожание) – это сложный колебательный процесс, возникающий при периодическом смещении центра тяжести какого-либо тела от положения равновесия, а также при периодическом изменении формы тела по сравнению с той, какую оно имеет при статическом состоянии.

Вибрации, воздействующие на человека, можно классифицировать по ряду признаков:

1. по способу передачи вибрации на человеческий организм:
2. общая (передающуюся через опорные поверхности на тело сидящего или стоящего человека);
3. локальная (передающуюся через руки или ноги человека, а также через предплечья, контактирующие с вибрирующими поверхностями).
4. по характеру спектра:
5. узкополосные вибрации, у которых контролируемые параметры в одной третьоктавной полосе частот более чем на 15 дБ превышает значения в соседних третьоктавных полосах;
6. широкополосные вибрации – с непрерывным спектром шириной более одной октавы;
7. по частотному составу:
8. низкочастотные вибрации – с преобладанием максимальных уровней в октавных полосах частот 1,4 Гц для общих вибраций, 8,16 Гц для локальных вибраций;
9. среднечастотные вибрации – с преобладанием максимальных уровней в октавных полосах частот 8,16 Гц для общих вибраций, 31.5,63 Гц для локальных вибраций;
10. высокочастотные вибрации – с преобладанием максимальных уровней в октавных полосах частот 31.5,63 Гц для общих вибраций, 125,1000 Гц для локальных вибраций;
11. по временным характеристикам:
12. постоянные вибрации, для которых величина нормируемых параметров изменяется не более чем в 2 раза (на 6 дБ) за время наблюдения;
13. непостоянные вибрации, для которых величина нормируемых параметров изменяется не менее чем в 2 раза (на 6 дБ) за время наблюдения не менее 10 минут при измерении с постоянной времени 1 с.

Непостоянные вибрации, описанные в пункте *d*.2 также делятся на:

* колеблющиеся во времени вибрации, для которых величина нормируемых параметров непрерывно изменяется во времени;
* прерывистые вибрации, когда контакт человека с вибрацией прерывается, причем длительность интервалов, в течение которых имеет место контакт, составляет более одной секунды;
* импульсные вибрации, состоящие из одного или нескольких вибрационных воздействий (например, ударов), каждый длительностью менее 1 с.

В зависимости от источника возникновения различают следующие виды вибраций:

* локальная вибрация, передающаяся человеку от ручного механизированного (с двигателями) инструмента;
* локальная вибрация, передающаяся человеку от ручного немеханизированного инструмента;
* общая вибрация 1 категории – транспортная вибрация, воздействующая на человека на рабочих местах транспортных средств, движущихся по местности, дорогам и пр. Пример: тракторы, грузовые автомобили;
* общая вибрация 2 категории – транспортно-технологическая вибрация, воздействующая на человека на рабочих местах машин, перемещающихся по специально подготовленным поверхностям производственных помещений и т. п. Пример: краны, напольный производственный транспорт;
* общая вибрация 3 категории – технологическая вибрация, воздействующая на человека на рабочих местах стационарных машин или передающаяся на рабочие места, не имеющих источников вибрации. Пример: станки, литейные машины;
* общая вибрация в жилых помещениях и общественных зданиях от внешних источников. Пример: вибрация от проходящего трамвая;
* общая вибрация в жилых помещениях и общественных зданиях от внутренних источников. Пример: лифты, холодильники.

Вибрация как раздражитель характеризуется колебательной мощностью, амплитудой, частотой колебаний, и длительностью действия. Мощность и длительность действия определяют общее количество энергии, получаемой структурой при действии внешней силы. Схема вибрации представлена на рисунке 6.1.

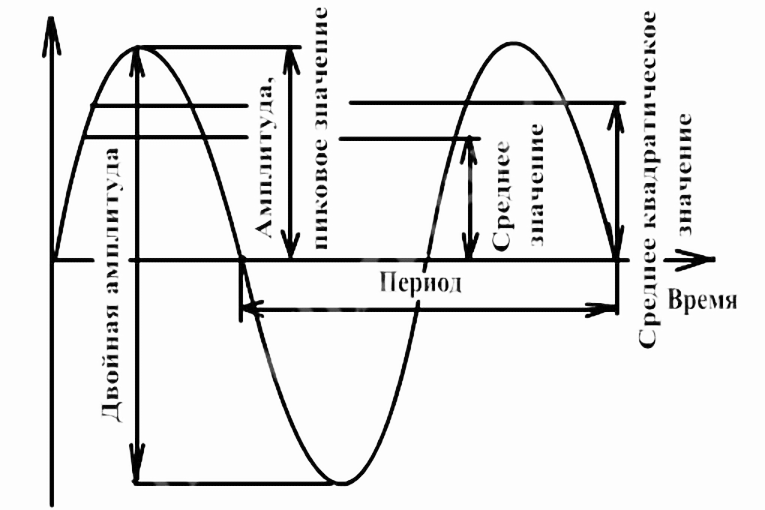


Рисунок 6.1 – Схема вибрации с основными параметрами

Основными физическими параметрами вибрации являются:

* частота *f*, Гц;
* период колебаний *Т*, с;
* амплитуда виброперемещения *А*, м;
* амплитуда колебательной скорости *V*, м/с;
* амплитуда колебательного ускорения *W*, м/.

Основными причинами возникновения вибрации являются неуравновешенные силы колеблющихся или вращающихся частей машины: несбалансированность, большие зазоры в сочленениях, не равномерный износ узлов машины, механизмов, неправильная центровка осей агрегатов при переходе вращения с помощью соединительной муфты, ослабление крепления оборудования на фундаменте или его устойчивость, применение масел, не отвечающих условиями работы оборудования, неудовлетворительное состояние подшипников, а также другие причины, вызванные местными условиями эксплуатации оборудования.

Динамика колебаний, воспринимаемых человеком, как вибрация, происходит непосредственно при контакте с колеблющейся поверхностью и находится от 12 Гц до 8000 Гц. Колебания частотой до 12 Гц воспринимаются как отдельные толчки.

Причиной возбуждения вибрации являются возникающие при работе машин являются неуравновешенные силовые воздействия:

* ударные нагрузки;
* возвратно-поступательные движения;
* дисбаланс

Наличие дисбаланса приводит к появлению неуравновешенных сил, вызывающих вибрацию. Причиной дисбаланса может являться неоднородность материала вращающегося тела, несовпадение центра массы тела с осью его вращения, деформация деталей от неравномерного нагреве и т. п.

# ЭНЕРГОСБЕРЕЖЕНИЕ И РЕСУРСОСБЕРЕЖЕНИЕ ПРИ

# ЭКСПЛУАТАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Энергосбережение и ресурсосбережение при внедрении и

## эксплуатации программного обеспечения

Под термином «энергосбережение» подразумевается реализация правовых, организационных, научных, производственных, технических и экономических мер, направленных на эффективное (рациональное) использование (и экономное расходование) топливно-энергетических ресурсови на вовлечение в хозяйственный оборот возобновляемых источников энергии.

Энергосбережение является важной экологической задачей по сохранению природных ресурсов и уменьшению загрязнения окружающей среды выбросами ТЭЦ и экономическая задача по снижению себестоимости товаров и услуг.

Энергетические ресурсы – это носители энергии, которые используются в настоящее время или могут быть полезно использованы в перспективе. Энергетическим ресурсом называют любой источник энергии, естественный или искусственно активированный [8, c. 15].

Ресурсосбережение – это совокупность мер по бережливому и эффективному использованию фактов производства (капитала, земли, труда). Данные меры обеспечиваются посредством использования ресурсосберегающих и энергосберегающих технологий. Ресурсосбережение охватывает не только факторы производства, но и продукцию, поскольку продукция одной отрасли может потребляться другой.

Соблюдение ресурсосбережения является важной характеристикой качества техники и технологии. Техника считается ресурсосберегающей, если она требует меньше расхода ресурсов на изготовление и эксплуатацию.

В настоящее время ресурсосбережение – одна из приоритетных задач экономики Республики Беларусь. Это связано прежде всего с дефицитом многих видов ресурсов: ростом стоимости их добычи (подготовки), а также серьезными экологическими проблемами.

При разработке программного обеспечения также требуется соблюдать основные принципы энерго- и ресурсосбережения.Экономия электроэнергии – крайне важный аспект жизни современного человеческого общества, затрагивающий и производственную сферу, и быт каждого отдельно взятого индивидуума. Неразумное потребление достаточно дорогостоящего вида энергии может привести к весьма значительным тратам, что может существенно сказаться как на благосостоянии человека, так и на развитии предприятия.

На сегодняшний день существуют самые разнообразные пути экономии электроэнергии, которые могут оказаться либо эффективными, либо не очень. Для уменьшения потребляемой электроэнергии необходимо проводить процедуры:

1. использовать персональные компьютеры лишь в дневное время, так как на многих предприятиях тариф электроэнергии в ночное время в несколько раз превышает тариф за неё в дневное время, что означает дополнительную трату средств.
2. все ПК предприятия обновить, обращая на стандарт энергопотребления;
3. заменить мониторы на аналоги, с меньшим энергопотреблением. Также при замене монитора нужно учитывать тот факт, что чем больше диагональ, тем выше потребление электроэнергии;
4. при выборе нового принтера отдавать предпочтение струйному, т.к. он потребляет на 80–90% энергии меньше, чем лазерный;
5. не оставлять ПК включённым на длительное время, если за ним никто не работает. Неиспользуемый два часа компьютер даже в «спящем режиме» потребляет 200–300 ватт, что за месяц составляет порядка 12 ;
6. рекомендуется всегда выключать периферийные устройства, если они не используются. Это позволит сэкономить порядка 2–3 за месяц.

Для уменьшения потребляемой энергии при обновлении ПК предприятия, что является одним из методов уменьшения потребляемой электроэнергии, приведённым в пункте *a*, предпочтительным является энергоэффективный стандарт *Energy* *Star* [9, c. 20]. Данный стандарт является международным стандартом энергоэффективности потребительских товаров. Впервые был принят Агентством по охране окружающей среды США, на основе государственной программы, в 1992 году. Позднее к программе присоединились Австралия, Канада, Япония, Новая Зеландия, Тайвань и Европейский союз. Подразумевается, что устройства, имеющие логотип *Energy* *Star*, имеют среднее энергопотребление на 20–30% меньше своих аналогов равной функциональности. Логотип стандарта *Energy* *Star* представлен на рисунке 7.1.



Рисунок 7.1 – Логотип стандарта *Energy* *Star*

Для реализации ресурсосбережения на предприятии необходимо решить следующие задачи:

* обеспечить высокий уровень квалификации работников предприятия;
* достичь высокого интеллектуального потенциала работников;
* обеспечить технико-технологичекую независимость предприятия;
* достичь высокого уровня конкурентоспособности;
* достичь высокой эффективности менеджмента деятельности предприятия;
* обеспечить необходимый уровень экологической деятельности предприятия;
* обеспечить правовую защищённость предприятия;
* обеспечить защиту информационной безопасности предприятия.

При организации хозяйственной деятельности на предприятии подлежат реализации следующие принципы ресурсосбережения:

* принцип научной обоснованности – предполагает разумное сочетание экологических и экономических интересов общества, обеспечивающих реальные гарантии прав человека на здоровую и комфортную для жизни окружающую среду;
* принцип комплексности – предусматривает многоцелевую направленность в использовании ресурсов, развитие малоотходных и безотходных производств, глубокую переработку сырья, использование вторичных ресурсов;
* принцип экономии – основывается на рациональном использовании имеющихся экономических ресурсов с учетом их ограниченности.

Человеческое здоровье – самый невосполнимый и дорогостоящий ресурс [10, c. 72]. С целью уменьшения излучение с экрана, графическое приложение было сделано ярким, но простым для зрительного восприятия. Для графики были использованы цвета, легко различаемые человеческим глазом. При этом контрастность игры позволяет легко следить за картинкой на протяжении длительного времени. Пользователю не приходится всматриваться в мелкие детали, читать тексты, написанные мелким почерком на неконтрастном фоне. Таким образом, усталость глаз пользователя сводится к низкому уровню.

Графическое оформление разработанного игрового приложения достаточно оптимизировано, что никак не отразилось на конечной картинке, которую видит пользователь. Используя относительно небольшой набор расцветок, удалось получить вполне яркое и запоминающееся изображение. Это опять же отразилось на конечном объеме приложения и позволило уменьшить системные требования к нему. В результате процессор не перегружен, нагрузка на видеоускоритель и оперативную память небольшая, следовательно, потребляемая устройством в процессе игры энергия тоже небольшая.

# ЗАКЛЮЧЕНИЕ

В ходе написания дипломной работы была проведена самостоятельная исследовательская работа, а также систематизированы, закреплены и углублены теоретические и практические знания, отработаны практические навыки, которые были применены для решения поставленных задач.

В дипломной работе были рассмотрены средства разработки игровых приложений, были выявлены их положительные и отрицательные стороны. Были изучены игровые жанры, в особенности «*Action-Adventure*»и составляющие его поджанры. Было установлено, что в успешной игре этого жанра должна присутствовать квестовая система, возможность развития персонажа и игровой мир, доступный для взаимодействия с персонажем игрока.

Также были изучены основные принципы работы с мультиплатформенным игровым движком *Unity*, который позволяет без особых сложностей создавать 2*D* и 3*D*-игры любой сложности и объёма.

Были выбраны технические средства реализации мобильного игрового приложения, среда разработки *Unity* и язык программирования *C*#.

В соответствии с заданием была разработана 3*D*-игра жанра «*Action-Adventure*» для мобильных устройств на базе операционной системы *Android*. В игру можно играть бесконечное количество раз, так как интерес заключается не в сюжете, а в непредсказуемости других игроков. Также реализованы системы квестов, предметов и характеристик персонажа игрока. В мобильном приложении реализована анимация графики и встроены звуковые эффекты.

Данная игра рассчитана на широкий круг пользователей, без ограничений по возрасту для игроков.

Игра способствует развитию моторики глаз и рук, учит рассуждать и логически мыслить, развивает образное и пространственное мышление. Также игра способствует расслаблению и захватывает ярким и разнообразным геймплеем.

Так как игра создана под платформу *Android*, в нее можно играть в любой момент, так как в современном мире смартфоны есть у подавляющего количества людей и всегда находятся под рукой. Благодаря разнообразным механикам и квестовой системе, игра приковывает к себе внимание и надолго удерживает интерес игрока к себе.

Также следует также отметить, что на основе годового экономического эффекта, был посчитан срок окупаемости данного приложения, который составляет пять лет.

# Список использованных источников

* 1. Хокинг, Д. В. Unity в действии. Мультиплатформенная разработка на C# / Д. В. Хокинг – СПб. : Питер, 2019. – 215 c.
  2. Гибсон, Б. Д. Unity и C#. Геймдев от идеи до реализации / Б. Д. Гибсон – СПб. : Питер, 2019. – 228 c.
  3. Куксон, А. А. Разработка игр на Unity / А. А. Куксон – М. : Бомбора, 2019. – 298 c.
  4. Рихтер, Д. Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Д. Д. Рихтер. – СПб. : Питер, 2017. – 917 с.
  5. Фримен, Э. К. Паттерны проектирования / Э. К. Фримен – СПб. : Питер, 2011. – 176 с.
  6. Feature driven development. – Электронные данные. – 2017. – Режим доступа: https://intellect.icu/feature-driven-development-5180. – Дата доступа: 16.05.2022.
  7. Различные виды тестирования ПО. – Электронные данные. – Режим доступа: https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing. – Дата доступа: 16.05.2022.
  8. Андрижиевский, А. А. Энергосбережение и энергетический менеджмент : учеб. пособие для студ. / А. А. Андрижиевский, В. И. Володин. – 2-е изд., испр. – Мн. : Вышэйшая школа, 2017. – 294 с.
  9. Ляшенко, Д. Д. Ресурсосбережение: инновации в энергосбережении / Д. Д. Ляшенко, В. И. Мартынович // Международное научное обозрение проблем и перспектив современной науки и образования Сборник статей по материалам XXX Международной научно-практической конференции, 2018. – С.32.
  10. Организация энергосбережения (энергоменеджмент). Решения ЗСМК-НКМКНТМК-ЕВРАЗ : учебное пособие / под ред. В. В. Кондратьева – М.: ИНФРАМ, 2017. – 108 с.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Листинг программы «*Heroic Adventure*»**

**Код программы для класса *PlayerController*:**

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.EventSystems;

[RequireComponent(typeof(PlayerMotor))]

public class PlayerController : MonoBehaviour

{

public Interactable focus;

public LayerMask movementMask;

private const float DOUBLE\_CLICK\_TIME = .2f;

private float lastClickTime;

Camera cam;

PlayerMotor motor;

/// <summary>

/// On start

/// </summary>

void Start()

{

cam = Camera.main;

motor = GetComponent<PlayerMotor>();

Debug.Log("start");

}

/// <summary>

/// Update is called once per frame

/// </summary>

void Update()

{

if (EventSystem.current.IsPointerOverGameObject())

return;

if (Input.GetMouseButtonDown(0))

{

// Shoot out a ray

Ray ray = cam.ScreenPointToRay(Input.mousePosition);

RaycastHit hit;

float timeSinceLastClick = Time.time - lastClickTime;

if (timeSinceLastClick > DOUBLE\_CLICK\_TIME)

{

//double

if (Physics.Raycast(ray, out hit, 100, movementMask))

{

Debug.Log("We hit " + hit.collider.name + "" + hit.point);

motor.MoveToPoint(hit.point);

RemoveFocus();

// SetFocus(null);

}

}

else

{

//normal

if (Physics.Raycast(ray, out hit, 100))

{

Interactable interactable = hit.collider.GetComponent<Interactable>();

if (interactable != null)

{

SetFocus(interactable);

// Debug.Log("We interact " + hit.collider.name);

}

// SetFocus(null);

}

}

lastClickTime = Time.time;

}

}

/// <summary>

/// Focus on interactive object

/// </summary>

/// <param name="newFocus"></param>

void SetFocus (Interactable newFocus)

{

if(newFocus != focus)

{

if(focus != null)

focus.OnDefocused();

focus = newFocus;

motor.FollowTarget(newFocus);

}

newFocus.OnFocused(transform);

}

/// <summary>

/// Remove focus on interactive object

/// </summary>

void RemoveFocus()

{

if(focus != null)

focus.OnDefocused();

focus = null;

motor.StopFollowingTarget();

}

}

**Код программы для класса *PlayerMotor*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

[RequireComponent(typeof(NavMeshAgent))]

public class PlayerMotor : MonoBehaviour

{

Transform target;

NavMeshAgent agent;

/// <summary>

/// On start

/// </summary>

void Start()

{

agent = GetComponent<NavMeshAgent>();

}

/// <summary>

/// Following interacted object

/// </summary>

/// <param name="newTarget">Target of following</param>

public void FollowTarget (Interactable newTarget)

{

agent.stoppingDistance = newTarget.radius \* .9f;

agent.updateRotation = false;

target = newTarget.interactionTransform;

}

/// <summary>

/// Stop following interacted object

/// </summary>

public void StopFollowingTarget()

{

agent.stoppingDistance = 0f;

agent.updateRotation = true;

target = null;

}

/// <summary>

/// Move to point

/// </summary>

/// <param name="point"></param>

public void MoveToPoint(Vector3 point)

{

agent.SetDestination(point);

}

/// <summary>

/// On Update scene

/// </summary>

void Update()

{

if (target != null)

{

agent.SetDestination(target.position);

FaseTarget();

}

}

/// <summary>

/// Rotate face to face with target of interacting

/// </summary>

void FaseTarget()

{

Vector3 direction = (target.position - transform.position).normalized;

Quaternion lookRotation = Quaternion.LookRotation(new Vector3(direction.x, 0f, direction.z));

transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation, Time.deltaTime \* 5f);

}

}

**Код программы для класса *CameraController*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraController : MonoBehaviour

{

public Transform target;

public Vector3 offset;

public float zoomSpeed = 4f;

public float minZoom = 5f;

public float maxZoom = 15f;

public float pitch = 1.7f;

public float yawSpeed = 100f;

private float currentZoom = 10f;

private float currentYaw = 0f;

/// <summary>

/// On update scene

/// </summary>

void Update()

{

// currentZoom -= Input.GetAxis("Mouse SccrollWheel") \* zoomSpeed;

// currentZoom = Mathf.Clamp(currentZoom, minZoom, maxZoom);

currentYaw -= Input.GetAxis("Horizontal") \* yawSpeed \* Time.deltaTime;

}

/// <summary>

/// On late update

/// </summary>

private void LateUpdate()

{

transform.position = target.position - offset \* currentZoom;

transform.LookAt(target.position + Vector3.up \* pitch);

transform.RotateAround(target.position, Vector3.up, currentYaw);

}

}

**Код программы для класса *EnemyController*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class EnemyController : MonoBehaviour

{

public float lookRadius = 10f;

Transform target;

NavMeshAgent agent;

CharacterCombat combat;

void Start()

{

target = PlayerManager.instance.player.transform;

agent = GetComponent<NavMeshAgent>();

combat = GetComponent<CharacterCombat>();

}

void Update()

{

float distance = Vector3.Distance(target.position, transform.position);

if (distance <= lookRadius)

{

agent.SetDestination(target.position);

if(distance <= agent.stoppingDistance)

{

CharacterStats targetStats = target.GetComponent<CharacterStats>();

if(targetStats != null)

{

combat.Attack(targetStats);

}

FaceTarget();

}

}

}

void FaceTarget()

{

Vector3 direction = (target.position - transform.position).normalized;

Quaternion lookRotation = Quaternion.LookRotation(new Vector3(direction.x, 0, direction.z));

transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation, Time.deltaTime \* 5f);

}

private void OnDrawGizmos()

{

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(transform.position, lookRadius);

}

}

**Код программы для класса *Inventory*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Inventory : MonoBehaviour

{

#region Singletone

public static Inventory instance;

private void Awake()

{

if (instance != null)

{

Debug.LogWarning("More then one inventory");

return;

}

instance = this;

}

#endregion

public delegate void OnItemChanged();

public OnItemChanged onItemChangedCallback;

public int space = 20;

public List<Item> items = new List<Item>();

public bool Add (Item item)

{

if (!item.isDefaultItem)

{

if (items.Count >= space)

{

Debug.Log("Not enough storage");

return false;

}

items.Add (item);

if(onItemChangedCallback != null)

onItemChangedCallback.Invoke();

}

return true;

}

public void Remove (Item item)

{

items.Remove (item);

if (onItemChangedCallback != null)

onItemChangedCallback.Invoke();

}

}

**Код программы для класса *InventorySlot*:**

using UnityEngine;

using UnityEngine.UI;

public class InventorySlot : MonoBehaviour

{

public Image icon;

public Button removeButton;

Item item;

public void AddItem( Item newItem)

{

item = newItem;

icon.sprite = item.icon;

icon.enabled = true;

removeButton.interactable = true;

}

public void ClearSlot()

{

item = null;

icon.sprite = null;

icon.enabled = false;

removeButton.interactable = false;

}

public void OnRemoveButton()

{

Inventory.instance.Remove(item);

}

public void UseItem()

{

if (item != null)

{

item.Use();

}

}

}

**Код программы для класса *InventoryUI*:**

using UnityEngine;

using UnityEngine.UI;

using System.Collections;

using System.Collections.Generic;

public class InventoryUI : MonoBehaviour

{

public Button openInventoryButton;

public Transform itemsPatent;

public GameObject inventoryUI;

Inventory inventory;

InventorySlot[] slots;

void Start()

{

inventory = Inventory.instance;

inventory.onItemChangedCallback += UpdateUI;

slots = itemsPatent.GetComponentsInChildren<InventorySlot>();

}

public void OpenInventory()

{

Debug.Log("ckick inv");

inventoryUI.SetActive(!inventoryUI.activeSelf);

}

void Update()

{

//if (Input.GetButtonDown("Inventory"))

//{

// inventoryUI.SetActive(!inventoryUI.activeSelf);

//}

}

void UpdateUI()

{

Debug.Log("update inventory ui");

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

{

if (i < inventory.items.Count)

{

slots[i].AddItem(inventory.items[i]);

}

else

{

slots[i].ClearSlot();

}

}

}

}

**Код программы для класса *Equipment*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "New Equipment", menuName = "Inventory/Equipment")]

public class Equipment : Item

{

public EquipmentSlot equipSlot;

public SkinnedMeshRenderer mesh;

public int armorModifier;

public int damegeModifier;

public override void Use()

{

base.Use();

EquipmentManager.instance.Equip(this);

RemoveFromInventory();

}

}

public enum EquipmentSlot { Head, Chest, Legs, Feet, Weapon1, Weapon2, Neck, Ring1, Ring2}

**Код программы для класса *Item*:**

using UnityEngine;

[CreateAssetMenu(fileName = "New Item", menuName = "Inventory/Item")]

public class Item : ScriptableObject

{

new public string name = "New Item";

public Sprite icon = null;

public bool isDefaultItem = false;

public virtual void Use()

{

//use

//effect

Debug.Log("Using" + name);

}

public void RemoveFromInventory()

{

Inventory.instance.Remove(this);

}

}

**Код программы для класса *ItemPickup*:**

using UnityEngine;

public class ItemPickup : Interactable

{

public Item item;

public override void Interact()

{

base.Interact();

PickUp();

}

void PickUp()

{

Debug.Log("Picking up " + item.name);

bool wasPickedUp = Inventory.instance.Add(item);

if(wasPickedUp)

Destroy(gameObject);

}

}

**Код программы для класса *CharacterStats*:**

using UnityEngine;

public class CharacterStats : MonoBehaviour

{

public int maxHealth = 100;

public int currentHealth { get; private set; }

public Stat damage;

public Stat armor;

private void Awake()

{

currentHealth = maxHealth;

}

private void Update()

{

if (Input.GetKeyDown(KeyCode.T))

{

TakeDamage(10);

}

}

public void TakeDamage( int damage)

{

damage -= armor.GetValue();

damage = Mathf.Clamp(damage, 0, int.MaxValue);

currentHealth -= damage;

Debug.Log(transform.name + " takes " + damage + " damage" );

if (currentHealth <= 0)

{

Die();

}

}

public virtual void Die()

{

//Die way

Debug.Log(transform.name + " died");

}

}

**Код программы для класса *EnemyStats*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyStats : CharacterStats

{

public override void Die()

{

base.Die();

//die effect-animation ragdoll

Destroy(gameObject);

}

}

**Код программы для класса *PlayerStats*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerStats : CharacterStats

{

void Start()

{

EquipmentManager.instance.onEquipmentChanged += OnEquipmentChanged;

}

void OnEquipmentChanged(Equipment newItem, Equipment oldItem)

{

if (newItem != null)

{

armor.AddModifier(newItem.armorModifier);

damage.AddModifier(newItem.damegeModifier);

}

if (oldItem != null)

{

armor.RemoveModifier(oldItem.armorModifier);

damage.RemoveModifier(oldItem.damegeModifier);

}

}

public override void Die()

{

base.Die();

PlayerManager.instance.KillPlayer();

}

}

**Код программы для класса *Stats*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[System.Serializable]

public class Stat

{

[SerializeField]

public int baseValue;

private List<int> modifiers = new List<int>();

public int GetValue()

{

int finalValue = baseValue;

modifiers.ForEach(x => finalValue += x);

return finalValue;

}

public void AddModifier(int modifier)

{

if (modifier != 0)

modifiers.Add(modifier);

}

public void RemoveModifier(int modifier)

{

if(modifier != 0)

modifiers.Remove(modifier);

}

}

**Код программы для класса *CharacterAnimation*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class CharacterAnimator : MonoBehaviour

{

const float locomotionAnimationSmoothTime = .1f;

NavMeshAgent agent;

Animator animator;

// Start is called before the first frame update

void Start()

{

agent = GetComponent<NavMeshAgent>();

animator = GetComponentInChildren<Animator>();

}

// Update is called once per frame

void Update()

{

float speedPercent = agent.velocity.magnitude / agent.speed;

animator.SetFloat("speedPercent", speedPercent, locomotionAnimationSmoothTime, Time.deltaTime);

}

}

**Код программы для класса *CharacterCombat*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(CharacterStats))]

public class CharacterCombat : MonoBehaviour

{

public float attackSpeed = 1f;

private float attackCooldown = 0f;

public float attackDelay = .6f;

public event System.Action OnAttack;

CharacterStats myStats;

private void Start()

{

myStats = GetComponent<CharacterStats>();

}

private void Update()

{

attackCooldown -= Time.deltaTime;

}

public void Attack(CharacterStats targerStats)

{

if(attackCooldown<= 0f)

{

StartCoroutine(DoDamage(targerStats, attackDelay));

if (OnAttack != null)

OnAttack();

attackCooldown = 1f / attackSpeed;

}

}

IEnumerator DoDamage (CharacterStats stats, float delay)

{

yield return new WaitForSeconds(delay);

stats.TakeDamage(myStats.damage.GetValue());

}

}

**Код программы для класса *Enemy*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(CharacterStats))]

public class Enemy : Interactable

{

PlayerManager playerManager;

CharacterStats myStats;

private void Start()

{

playerManager = PlayerManager.instance;

myStats = GetComponent<CharacterStats>();

}

public override void Interact()

{

base.Interact();

CharacterCombat playerCombat = playerManager.player.GetComponent<CharacterCombat>();

if(playerCombat != null)

{

playerCombat.Attack(myStats);

}

}

}

**Код программы для класса *Interactable*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Interactable : MonoBehaviour

{

public float radius = 3f;

public Transform interactionTransform;

bool isFocus = false;

Transform player;

bool hasInteracted = false;

public virtual void Interact()

{

//overwriten

// Debug.Log("Interacting with " + transform.name);

}

private void Update()

{

if (isFocus && !hasInteracted)

{

float distance = Vector3.Distance( player.position, interactionTransform.position);

if (distance <= radius)

{

Interact();

hasInteracted = true;

}

}

}

public void OnFocused (Transform playerTransform)

{

isFocus = true;

player = playerTransform;

hasInteracted = false;

}

public void OnDefocused()

{

isFocus = false;

player = null;

hasInteracted = false;

}

void OnDrawGizmosSelected()

{

if(interactionTransform == null)

interactionTransform = transform;

Gizmos.color = Color.yellow;

Gizmos.DrawWireSphere(interactionTransform.position, radius);

}

}

**Код программы для класса *PlayerManager*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class PlayerManager : MonoBehaviour

{

#region Singletone

public static PlayerManager instance;

private void Awake()

{

instance = this;

}

#endregion

public GameObject player;

public void KillPlayer()

{

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

}

}

**Код программы для класса *EquipmentManager*:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EquipmentManager : MonoBehaviour

{

#region Singletone

public static EquipmentManager instance;

private void Awake()

{

instance = this;

}

#endregion

Equipment[] currentEquipment;

public delegate void OnEquiomentChanged(Equipment newItem, Equipment oldItem);

public OnEquiomentChanged onEquipmentChanged;

Inventory inventory;

private void Start()

{

inventory = Inventory.instance;

int numSlots = System.Enum.GetNames(typeof(EquipmentSlot)).Length;

currentEquipment = new Equipment[numSlots];

}

public void Equip(Equipment newItem)

{

int slotIndex = (int)newItem.equipSlot;

Equipment oldItem = null;

if (currentEquipment[slotIndex] != null)

{

oldItem = currentEquipment[slotIndex];

inventory.Add(oldItem);

}

if (onEquipmentChanged != null)

{

onEquipmentChanged.Invoke(newItem,oldItem);

}

currentEquipment[slotIndex] = newItem;

}

public void Unequip (int slotIndex)

{

if (currentEquipment[slotIndex] != null)

{

Equipment oldItem = currentEquipment[slotIndex];

inventory.Add(oldItem);

currentEquipment[slotIndex] = null;

if (onEquipmentChanged != null)

{

onEquipmentChanged.Invoke(null, oldItem);

}

}

}

public void UnequipAll()

{

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

{

Unequip(i);

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Руководство пользователя**

**1. Введение.**

В данном программном документе приведено руководство пользователя по настройке и использованию игрового приложения «*Heroic Adventure*». Данное игровое приложение предназначено для проведения досуга.

Пользователь должен обладать общими навыками по использованию мобильных устройств на базе операционной системы *Android*, знать и соблюдать правила техники безопасности и охраны труда.

**2. Назначение и условия применения.**

Игровое приложение «*Heroic Adventure*» реализует следующие игровые функции:

* интерактивное взаимодействие с предметами;
* интерактивный игровой инвентарь;
* система игровых квестов;
* система экипировки;
* система характеристик персонажа игрока.

Для функционирования игрового приложения необходимо соблюдение следующих требований:

* наличие свободного места на мобильного устройства на базе операционной системы *Android* для установки игрового приложения.

**3. Подготовка к работе.**

Для использования программы требуется загрузить на мобильное устройство установочный файл, произвести установку приложения, после чего запустить игру из меню приложений.

Если все инструкции соблюдены и приложение не выдает никаких сообщений об ошибках, значит игровое приложение работает исправно.

**4. Технические требования к ПО.**

Минимальные технические требования к мобильному устройству, на котором может использоваться разработанное игровое приложение, следующие:

* *Android* не ниже версии 4.0;
* оперативная память не меньше 3 Гб;
* свободное место на устройстве 50 Мб.

Минимальные технические требования устанавливают порог, при котором достигается комфортное количество кадров в секунду

**5. Описание операций.**

После запуска игрового приложения, для начала игры необходимо нажать кнопку «*Play*» в главном меню. Открыть панель регулировки звука можно нажав на кнопку «*Options*», а выйти из игры нажав на кнопку «*Quit*». Главное меню представлено на рисунке Б.1.



Рисунок Б.1 – Главное меню

Во время игрового процесса игрок может поставить игру на паузу нажав кнопку меню паузы. После этого на экране появится панель меню, а игровой процесс будет приостановлен. Меню паузы представлено на рисунке Б.2



Рисунок Б.2 – Меню паузы

Открыв панель инвентаря, игрок получает доступ к собранным предметам экипировки. Их можно надеть, снять или выбросить из инвентаря. Вкладка инвентаря представлена на рисунке Б.3.



Рисунок Б.3 – Вкладка инвентаря

Открыв панель характеристик персонажа, игрок может получить информацию о текущих показателях характеристик и уровня персонажа, а также при наличии очков повышения характеристик может по собственному усмотрению увеличить показатели характеристик. Вкладка характеристик персонажа представлена на рисунке Б.4



Рисунок Б.4 – Вкладка характеристик персонажа

Открыв панель квестов, игрок может получить информацию о наличии и состоянии текущих квестов, принятых им к исполнению. Вкладка квестов представлена на рисунке Б.5.



Рисунок Б.5 – Вкладка квестов

**6. Аварийные ситуации.**

Игровое приложение «*Heroic Adventure*» не выдает никаких сообщений пользователю об ошибке. В случае возникновения аварийной ситуации рекомендуется выполнить перезапуск приложения.

# ПРИЛОЖЕНИЕ В

(обязательное)

**Руководство системного программиста**

**1. Назначение и условия применения.**

В данном программном документе приведено руководство системного программиста по настройке и использованию игрового приложения «*Heroic Adventure*». Данное игровое приложение предназначено для проведения досуга.

Игровое приложение «*Heroic Adventure*» реализует следующие игровые функции:

* интерактивное взаимодействие с предметами;
* интерактивный игровой инвентарь;
* система игровых квестов;
* система экипировки;
* система характеристик персонажа игрока.

**2. Структура программного комплекса.**

Программное средство состоит из одного мобильного приложения.

**3. Настройка программного комплекса.**

Для использования программы требуется загрузить на мобильное устройство установочный файл, произвести установку приложения, после чего запустить игру из меню приложений.

**4. Проверка программного комплекса.**

Если все инструкции соблюдены и после запуска приложение не выдает никаких сообщений об ошибках, значит игровое приложение работает исправно.

**5. Сообщения системному программисту.**

Игровое приложение «*Heroic Adventure*» не выдает никаких сообщений системному программисту.

# ПРИЛОЖЕНИЕ Г

(обязательное)

**Руководство программиста**

**1. Назначение и условия применения программного комплекса.**

В данном программном документе приведено руководство программиста по настройке и использованию игрового приложения «*Heroic Adventure*». Данное игровое приложение предназначено для проведения досуга.

Игровое приложение «*Heroic Adventure*» реализует следующие игровые функции:

* интерактивное взаимодействие с предметами;
* интерактивный игровой инвентарь;
* игровой журнал;
* система порталов.

**2. Характеристики программного комплекса.**

Игровое приложение разработано на игровом движке *Unity* 3*D* с использованием языка программирования *C*#.

Приложение может функционировать в ОС *Android*.

**3. Обращение к программному комплексу.**

Для внесения изменений в игровое приложение необходимо использовать игровой движок *Unity* 3*D*.

**4. Сообщения.**

Игровое приложение «*Heroic Adventure*» не выдает никаких сообщений программисту.

# ПРИЛОЖЕНИЕ Д

(справочное)

**Формулы расчета экономической эффективности**

Для расчёта экономической эффективности разработанного программного продукта, используются следующие формулы:

где , – коэффициенты технического уровня нового и базисного программного продукта, которые можно рассчитать по формуле (Д.2):

где – коэффициенты весомости *i*-го технического параметра;

*n* – число параметров;

– численное значение *i*-го технического параметра сравниваемого программного продукта;

– численное значение *i*-го технического параметра эталона.

где , – балльная оценка неизмеримых показателей нового и базового изделия соответственно.

где – коэффициент соответствия нового программного продукта нормативам ( = 1);

– коэффициент цены потребления.

где *Vi* – объем отдельной функции ПО;

*n* – общее число функций.

где – уточненный объем отдельной функции ПО в строках исходного кода.

где – коэффициент, соответствующий степени повышения сложности;

– количество учитываемых характеристик.

, (Д.8)

, (Д.9)

, (Д.10)

, (Д.11)

, (Д.12)

где , , , и – значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО.

где – нормативная (скорректированная) трудоемкость разработки ПО на  *i*-й стадии, чел.-дн.;

*n* – количество стадий разработки.

, (Д.14)

, (Д.15)

где – основная заработная плата разработчиков, руб.;

– дополнительная заработная плата разработчиков, руб.;

– сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

, (Д.16)

где – средняя часовая тарифная ставка, руб./час;

– общая трудоемкость разработки, чел.-час;

– коэффициент доплаты стимулирующего характера, = 1,8.

где – часовая тарифная ставка разработчика *i*-й категории, руб./час;

*ni* – количество разработчиков *i*-й категории.

где – тарифная ставка 1-го разряда;

– тарифный коэффициент.

где – норматив на дополнительную заработную плату разработчиков.

где – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы ( = 34%).

, (Д.21)

где – стоимость 1 часа машинного времени, руб./ч;

– коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ, = 1;

– машинное время ЭВМ, необходимое для разработки и отладки проекта, ч.

где – затраты на заработную плату обслуживающего персонала с учетом всех отчислений, руб./год;

– стоимость аренды помещения под размещение вычислительной техники, руб./год;

– амортизационные отчисления за год, руб./год;

– затраты на электроэнергию, руб./год;

– затраты на материалы, необходимые для обеспечения нормальной работы ПЭВМ (вспомогательные), руб./год;

– затраты на текущий и профилактический ремонт ЭВМ, руб./год;

– прочие затраты, связанные с эксплуатацией ПЭВМ, руб./год;

– действительный фонд времени работы ЭВМ, час/год.

где – основная заработная плата обслуживающего персонала, руб.;

– дополнительная заработная плата обслуживающего персонала, руб.;

– сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.;

– количество обслуживаемых ПЭВМ, шт.;

– месячная тарифная ставка *i*-го работника, руб.;

*n* – численность обслуживающего персонала, чел.;

– процент дополнительной заработной платы обслуживающего персонала от основной;

– процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы.

где – средняя годовая ставка арендных платежей, руб./м2;

– площадь помещения, м2.

где – затраты на приобретение *i*-го вида основных фондов, руб;

– коэффициент, дополнительных затраты, связанные с доставкой, монтажом и наладкой оборудования, = 13% от ;

– балансовая стоимость ЭВМ, руб;

– норма амортизации, %.

где – паспортная мощность ПЭВМ, кВт;

= 0,41 кВт;

– стоимость одного кВт-часа электроэнергии, руб;

*А* – коэффициент интенсивного использования мощности, А=0,98…0,9.

, (Д.30)

где – общее количество дней в году, = 365 дней;

, – число выходных и праздничных дней в году, = 121 дней;

– продолжительность 1 смены, = 8 часов;

– коэффициент сменности, = 1;

– коэффициент, учитывающий потери рабочего времени, связанные с профилактикой и ремонтом ЭВМ, примем = 0,2.

где – затраты на приобретение (стоимость) ЭВМ, руб.;

– коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, = 12–13 % от ;

– коэффициент, характеризующий затраты на вспомогательные материалы ( = 0,01).

где – коэффициент, характеризующий затраты на текущий и профилактический ремонт, = 0,05.

где – коэффициент, характеризующий размер прочих затрат, связанных с эксплуатацией ЭВМ ( = 0,05).

, (Д.34)

где – срок реализации стадии «Рабочий проект» (РП);

– срок реализации стадии «Ввод в действие» (ВП);

= 60;

– продолжительность рабочей смены, ч., = 8 ч.;

– количество рабочих смен, = 1.

, (Д.35)

где – коэффициент, учитывающий размер затрат на изготовление эталонного экземпляра, = 0,05.

где – цена *i*-го наименования материала полуфабриката, комплектующего, руб.;

– потребность в *i*-м материале, полуфабрикате, комплектующем, натур. ед.;

– коэффициент, учитывающий сложившийся процент транспортно-заготовительных расходов в зависимости от способа доставки товаров, = 0,1;

– цена возвратных отходов *i*-го наименования материала, руб.;

N0*i* – количество возвратных отходов *i*-го наименования, натур. ед.;

*n* – количество наименований материалов, полуфабрикатов, и т.д.

где – норматив общепроизводственных затрат.

где – норматив непроизводственных затрат.

, (Д.39)

где – себестоимость ПО, руб.;

– прибыль от реализации программного продукта, руб.;

– уровень рентабельности программного продукта, % (= 30 %).

, (Д.41)

где – ставка налога на добавленную стоимость, %, = 20 %.

где – приведенные затраты на единицу выпуска ПО по базовому и новому вариантам, руб.;

– годовой объем выпуска в расчетном году для реализуемого ПО, ед.

Коэффициент дисконтирования в расчетном периоде (K*t*):

где *r* – норма дисконта (применяется на уровне ставки рефинансирования и равна 0,085);

*t –* период реализации проекта.

Таблица Д.1 – Технико-экономические показатели проекта

|  |  |  |
| --- | --- | --- |
| Наименование показателя | Единица измерения | Проектный вариант |
| 1 | 2 | 3 |
| Интегральный коэффициент конкурентоспособности | – | 2,27 |
| Коэффициент эквивалентности | – | 1,66 |
| Коэффициент изменения функциональных  возможностей | – | 1,27 |
| Коэффициент соответствия нормативам | – | 1 |
| Коэффициент цены потребления | – | 0,93 |
| Общая трудоемкость разработки ПО | чел.- дн | 75 |
| Затраты на оплату труда разработчиков | руб. | 4613,01 |
| Затраты машинного времени | руб. | 88 |
| Затраты на изготовление эталонного экземпляра | руб. | 235,05 |
| Затраты на технологию | руб. | – |
| Затраты на материалы | руб. | 13,2 |
| Общепроизводственные затраты | руб. | 285,6 |
| Непроизводственные (коммерческие) затраты | руб. | 142,8 |
| Суммарные затраты на разработку ПО () | руб. | 5377,67 |

Продолжение таблицы Д.1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Оптовая цена ПП () без НДС | руб. | 6990,97 |
| Оптовая цена ПП () с НДС | руб. | 8389,16 |
| Экономический эффект от производства нового ПП | руб. | 2232,51 |

# ПРИЛОЖЕНИЕ Ж

(рекомендуемое)

**Список опубликованных работ**

* 1. Тарасенко, Д. В. Создание экранных эффектов в Unity с помощью рендер-текстур / Д. В. Тарасенко // Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях : материалы XXIV Республиканской научной конференции студентов и аспирантов, Гомель, 22–24 марта 2021 г. / ГГУ им. Франциска Скорины ; под ред: С. П. Жогаль [и др.] – Гомель, 2021. – С. 372–373.