МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
им. ГУМАРБЕКА ДАУКЕЕВА»
Кафедра IT - инжиниринг

«ДОПУЩЕН К ЗАЩИТЕ»
Зав.кафедрой PhD, доцент Досжанова Алия Амантаевна
(ученая степень, звание,
Ф.И.О.)
____________«____»______________202__ г.
(подпись)

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Разработка компьютерной игры «Roleplay game» на платформе Unity 3D
Специальность 5В060200 – «Информатика»
Выполнил: Жолдасов Мадияр Леспекулы Группа: Инф-16-2
(Ф.И.О.)
Научный руководитель: PhD, доцент, Маликова Феруза Умирзаховна
(ученая степень, звание, Ф.И.О.)
Консультанты:
по экономической части: к.э.н., профессор Габелашвили К.Р
(учѐная степень, звание, Ф.И.О.)
_____________________________________ «____» __________________2020 г.
по безопасности жизнедеятельности: ассистент Тыщенко Е.М
(учѐная степень, звание, Ф.И.О.)
_____________________________________ «____» __________________2020 г.
по программному обеспечению: ст.преп. Майкотов М.Н
(учѐная степень, звание, Ф.И.О.)
_____________________________________ «____» __________________2020 г.
Нормоконтролер: ст.преп. Абсатарова Б.Р
(учѐная степень, звание, Ф.И.О.)
_____________________________________ «____» __________________2020 г.
Рецензент:____
(учѐная степень, звание, Ф.И.О.)
_____________________________________ «____» __________________2020 г.

Алматы 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
им. ГУМАРБЕКА ДАУКЕЕВА»

Институт систем управления и информационных технологий

Специальность 5В060200 – «Информатика»

Кафедра IT- инжиниринг

ЗАДАНИЕ
на выполнение дипломного проекта

Студенту Жолдасову Мадияру Леспекулы

Тема проекта: Разработка компьютерной игры «Roleplay game» на платформе Unity 3D

Утверждена приказом по университету № ___ от «___» ________2020 г.

Срок сдачи законченного проекта «___» __________ 2020 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): Unity 3D – платформа, C# - используемый язык программирования.Visual Studio – среда программирования.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта:
а) изучение технологии производства компьютерных игр;
б) технологии разработки приложения;
в) функционал программного продукта;
г) экономическая эффективность проекта;
д) вопросы безопасности жизнедеятельности и охраны труда.

Перечень графического материала (с точным указанием обязательных чертежей): представлены 4 таблицы, 26 иллюстраций.

Основная рекомендуемая литература:

1 C# Game Programming Cookbook for Unity 3D, Jeff Murray, 2014;

2 Learning C# by Developing Games with Unity 3D Beginner's Guide, Terry Norton, 2013;

3 Head First C#, Jennifer Greene, Andrew Stellman (рус.: Изучаем C#, Д. Грин, Э. Стиллмен);

4 Язык программирования C# 5.0 и платформа .NET 4.5 - Эндрю Троелсен.

Консультация по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант	Сроки	Подпись
Экономическая часть	Габелашвили К.Р.	19.04.2020	
Безопасности жизнедеятельности	Тыщенко Е.М.	22.04.2020	
Программная часть	Майкотов М.Н.	15.05.2020	
Нормоконтролер	Абсатарова Б.Р.	13.05.2020	

ГРАФИК
подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечания
Анализ существующих систем	17.01.2020 21.01.2020	
Решение проблем	24.02.2020 28.02.2020	
Проектирование информационной системы	02.03.2020 13.03.2020	
Разработка информационной системы	16.03.2020 24.04.2020	
Интерфейс информационной	27.04.2020 09.05.2020	
Экономическое обоснование	19.04.2020	
Безопасность жизнедеятельности	22.04.2020	

Дата выдачи задания «____» __________________2020 г.

Заведующий кафедрой ___________________А.А.Досжанова

Научный руководитель проекта ____________Ф.У.Маликова

Задание принял к исполнению студент ____________М.Л.Жолдасов

АҢДАТПА

Дипломдық жобаның тақырыбы: «Unity 3D платформасында» Roleplay game «компьютерлік ойынын дамыту.»

Бұл дипломдық жоба itch.io онлайн-сервисінде орналастыру үшін ойын-сауық компьютерлік ойынын әзірлеуді жүзеге асырады.

Дипломдық жоба Windows жүйесіне арналған өтінім форматында орындалды. Әзірлеу кезінде Blender бағдарламасында тілде сөйлеу технологиялары, 3D модельдеу қолданылды. Қосымша Unity 3D платформасында жасалды.

Сондай-ақ, бағдарламаны әзірлеуге жұмсалған шығындар мен шығындардың экономикалық есебі, әзірленген жобаның экономикалық мақсаттылығы бағаланды.

АННОТАЦИЯ

Тема дипломного проекта: «Разработка компьютерной игры «Roleplay game» на платформе Unity 3D».

В данном дипломном проекте реализована разработка компьютерной игры развлекающего характера для размещения на онлайн-сервисе itch.io.

Дипломный проект был выполнен в формате приложения для Windows. При разработке использовались технологии языка программирования C#, 3D моделирование в Blender. Приложение разрабатывалось на платформе Unity 3D.

Также был проведен экономический расчет затрат и стоимости разработки программы, оценка экономической целесообразности разрабатываемого проекта.

ANNOTATION

Theme of the graduation project: "Development of the computer game" Role-playing game "on the Unity 3D platform."

This graduation project implements the development of a computer game of an entertaining nature for placement on an online service Itch.io.

The graduation project was executed in the format of an application for Windows. During development, C # programming language technologies were used, 3D modeling in Blender. The application was developed on the Unity 3D platform.

An economic calculation of the costs and costs of developing programs was carried out, an assessment of the economic feasibility of the developed project.

Содержание

Введение 9
1 Аналитическая часть 11
1.1 Постановка задачи 11
1.2 Анализ существующих систем 13
1.3 Постановка задачи 26
2 Проектирование системы...28
2.1 Диаграммы. Распределение ролей 28
2.2 Выбор игрового движка. Язык программирования 33
3 Программная реализация 42
3.1 Программа.. 42
3.2 Интерфейс 58
4 Безопасность жизнедеятельности...62
4.1 Введение 62
4.2 Роль анализаторов в операторской деятельности...............................62
4.3 Средства отображения информации 64
4.4 Оптимизация средств и систем отображения информации 64
4.5 Яркостная характеристика зрительной информации 65
4.6 Временная характеристика зрительной информации 65
5 Экономическая часть 68
5.1 Трудоемкость разработки приложения 68
5.2 Расчет затрат на разработку информационной системы 69
5.3 Затраты на оборудование и программное обеспечение 69
5.4 Затраты на электроэнергию 69
5.5 Затраты на оплату труда 70
5.6 Социальный налог 71
5.7 Расчет поступления денежных средств 73
5.8 Расчет прибыли и срока окупаемости от реализации игры 74
Заключение...75

Список литературы 76
Приложение А 77
Техническое задание...77
Приложение Б – Листинг программы 79
Приложение В – Акт внедрения..128

Введение

Индустрия компьютерных игр проделала долгий путь и шла рука об руку с развитием компьютерных технологий. По мере развития компьютеров и роста их вычислительных мощностей, росли возможности виртуальных развлечений.

С 2011 года компьютерные игры были официально признаны в США отдельным видом искусства. Это обоснованно большими возможностями компьютерных симуляций и большой вариативностью получаемых ощущений и эмоций от игры.

С момента, когда была написана первая компьютерная игра, в 1946 году, прошло уже больше полувека. С тех пор игры научились применять для обучения в самых разных областях, как для изучения двоичной логики в школе на уроках информатики, так и для обучения спецназа тактике.

Сейчас компьютерные игры уже входят в нашу повседневную жизнь, и по данным американского портала ESE на 2016 год, в США около 63% людей играют в компьютерные игры.

Компьютерные игры на данный момент применяются в ряде образовательных программ. Так, например Minecraft, а именно его образовательная версия MinecraftEdu является частью образовательной программы Швеции с 2013 года, в Австралии игру используют на уроках естествознания, а в США – для изучения истории. Более того, в Северной Ирландии Minecraft установили не только в средние школы, но и библиотеки.

С другой стороны, серия тактических игр Close Combat позволяет солдатам обучаться стратегии военных действий не выходя на полигон. Дело в том, что движок этой игры ("ядро" программы - набор базовых алгоритмов построения игрового мира, который может быть использован для создания других игр) позволяет легко воссоздавать различные сценарии боя, включая любые тактические сухопутные вооружения, карты и прочее. Фактически на базе Close Combat можно создать любое поле боя.

Жанр RPG. Игры жанра RPG подразумевают большую свободу выбора, так как сам термин “Roleplay game” означает ролевую игру, в которой игрок берет под контроль персонажа и может отыгрывать его позицию, принципы и характер, а может пойти по своему пути, отличающемуся от позиции его персонажа. Первая компьютерная игра была разработана еще в 1946. А жанр RPG берет начало в 1976 году, вместе с выходом pedit5 и dnd. Идейным вдохновителем компьютерных RPG является настольная игра Dungeons & Dragons. Именно она заложила основные механики, термины и игровую логику. В Dungeons & Dragons игрок мог играть за определенную роль, например война, мага или вора, имел очки здоровья Health Points, по истечению которых персонаж умирал и играть дальше становилась

невозможно, очки маны Mana Points, которые игрок тратил на использование различных умений, позволяющих наносить урон противникам, быстро перемещаться, восполнять здоровье, ману и другие способности влияющие на геймплей.

В современном мире огромное место в мировой экономике занимает индустрия развлечений. Наряду с музыкой, фильмами и анимацией есть такая ниша как компьютерные игры. Например, за 2018 год игровая индустрия обогатилась на 137.8 млрд долларов.

Жанр RPG имеет несколько классификаций и на данный момент подразделяется на множество меньших поджанров, имеющих свои характерные особенности, которые могут в корне отличаться друг от друга. Так, например популярные игры Ведьмак, Mass Effect и Dragon Age относятся к сюжетным RPG, а серия Fallout и The Elder Scrolls от студии Bethesda относятся к RPG – песочницам.

1 Аналитическая часть

1.1 Постановка задачи

Задача стоит в разработке развлекательного приложения, компьютерной игры, соответствующей современным требованиям игрока, а именно увлекательной, реиграбельной, стабильной игры.

Создание хорошей компьютерной игры всегда двигалось в направлении этих качеств:

1) Красивая графическая составляющая.
2) Красивая звуковая составляющая.
3) Отзывчивая игровая система.
4) Реиграбельность.
5) Отсутствие багов.

Под красивой графической составляющей подразумевается стилистическое соответствие визуального контента. За это отвечает правильный подбор 3D моделей, текстур, цветовой гаммы и постобработки изображения.

Под красивой звуковой составляющей имеется ввиду как стилистическое соответствие звука, так и высокое качество звуковых дорожек.

Отзывчивая игровая система это сочетание таких качеств как быстрота управления, визуальное отображение действий игрока, так называемая обратная связь мира с игроком. Визуальное отображение может как являться естественным – когда оно выражается в очевидных ответах на действия игрока, причинно-следственных связях, которые ожидаемы в реальном мире или похожи на те, что ожидаемы в реальности, и искусственным – когда на действия игрока в игре отображается несуществующие в реальности объекты, отображающие необходимую пользователю информацию. Явным плюсом естественного визуального отображения действий игрока является реалистичность, что способствует погружению игрока в виртуальный мир.

Реиграбельность – это особенность игры, позволяющая играть в нее более одного раза/прохождения с разным игровым опытом. Это означает свободу действий, что при каждом прохождении игрок может выбрать свой стиль игры, отличающийся от предыдущего. Не все игры являются реиграбельными, несмотря на то, что все можно проходить более одного раза. В последнее время очень популярны игры, имеющие мультиплеер и проработанную и разнообразную игровую механику. Такие игры позволяют создавать огромное множество игровых ситуаций, создаваемых путем перемешивания различных значений компонентов игры и различными

комбинациями игровых ситуаций, чему способствует развитая фантазия игроков и соревновательный дух.

Баг (в ряде случаев от англ. bug — клоп, любое насекомое, вирус) жаргонное слово в программировании, обычно обозначающее ошибку в программе. Под багами понимаются возможности ошибки во время игрового процесса. Такие ошибки очень фатальны для игры вышедшей из раннего доступа, ведь опыт от игры портится, если постоянно сменяется закрытием игры, невозможностью продвижения по сюжету и различными неприятными особенностями багов. Для игры, находящейся в раннем доступе баги являются допустимыми и исправляются с помощью обратной связи от компьюнити (от английского слова community (сообщество),группа людей с близкими интересами, которые общаются через Интернет друг с другом) игры.

Каждая из этих составляющих играет определенную роль в успехе продукта. Например, графическая составляющая это первое, что видит потенциальный игрок, она определяет количество игроков, которые захотят посмотреть рекламную презентацию игры. Звуковая составляющая работает уже на этапе рекламы, когда показывается игровой процесс. Звуковая составляющая играет огромную роль, отвечая за ощущение законченности и полноценности игры. Порою игры со значительно упрощенной визуальной составляющей, но качественными звуковыми эффектами и саундтреком становятся популярными и обретают большую фанатскую базу. Отзывчивая игровая система играет решающую роль для тех, кто уже посмотрел рекламу и возможно купил игру, она решает какой процент игроков решит оставить игру себе, не возвращая деньги и посоветовать эту игру своим друзьям. Игровая система отвечает за итоговый имидж игры уже после того, как некоторое множество игроков оставило о ней свои отзывы.

Из всех жанров игр был выбрал жанр RPG, а именно поджанр Action RPG.

К наиболее известным и успешным играм такого жанра относятся Skyrim, Fallout: New Vegas, Witcher 3.

Рисунок 1.4 – обложки известных игр в жанре Action RPG

Данный жанр отличается большим разнообразием игровых ситуаций и возможностью идентифицировать себя с главным персонажем игры. Поэтому данный жанр подходит для широкой аудитории.

Данные игры объединяет большой открытый мир, проработанная боевая система, работающая по принципу поощрения скорости реакции и углубленного изучения игровой механики, которая построена на логике игрового мира.

1.2 Анализ существующих систем

The Elder Scrolls V: Skyrim

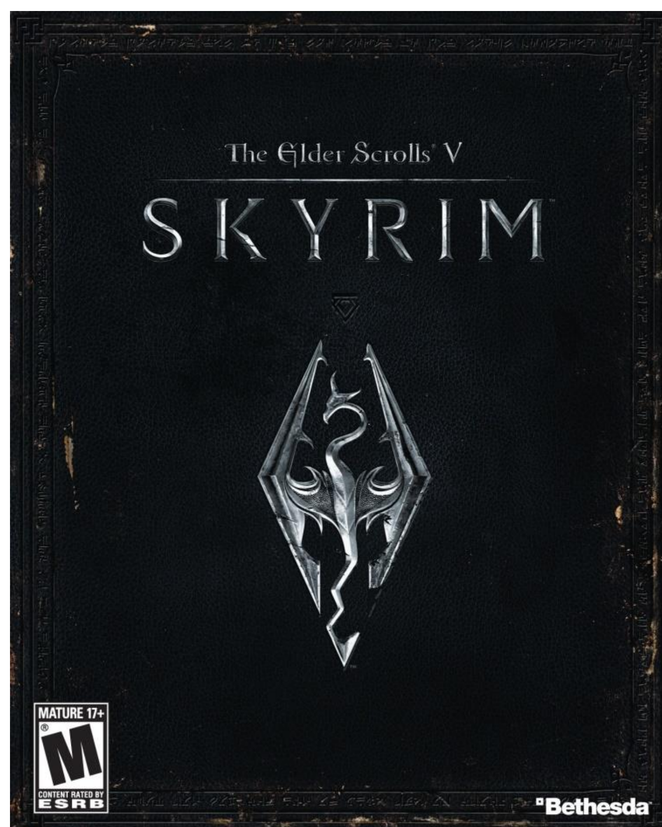

Рисунок 1.4 - обложка игры «The Elder Scrolls V:Skyrim»

The Elder Scrolls V: Skyrim (рис 1.4.), (дословно с англ. - «Древние свитки 5: Скайрим») - мультиплатформенная компьютерная ролевая игра с открытым миром, разработанная студией Bethesda Game Studios и выпущенная компанией Bethesda Softworks.

Skyrim создавался на движке собственного производства компании Bethesda - Creation Engine. Skyrim это первая игра произведенная на данном движке. Игровой движок Creation Engine был разработан первостепенно для использования в TES 5:Skyrim. Сам же движок является продолжением игрового движка Gamebryo, произведенного Emergent Game Technologies и

Numerical Design Limited, что было обоснованно желанием использовать средства, опробованные в других проектах.

При разработке движка создатели уделили большое внимание возможности отображать территории с большой дистанцией прорисовки, так как в Skyrim большой игровой мир, свободный для передвижения. Графический движок разрабатывался, двигаясь в сторону более правдоподобного освещения объектов и улучшени качества прорисовки воды. Особое внимание авторы уделили реалистичной проработке снега. Движок автоматически генерирует нужную текстуру снега на деревьях, камнях, кустах в зависимости от местности.

Вместо технологии SpeedTree, использовавшейся для прорисовки деревьев было использовано собственное програмное решение.

Искусственный интеллект персонажей, был реализован на системе Radioan AI – собственном изобретении Bethesda, использовавшемся в предыдущей части The Elder Scrolls: Oblivion. В Skyrim персонажи едят еду, работают, заходят в здания и т.д, то есть живут собственной жизнью.

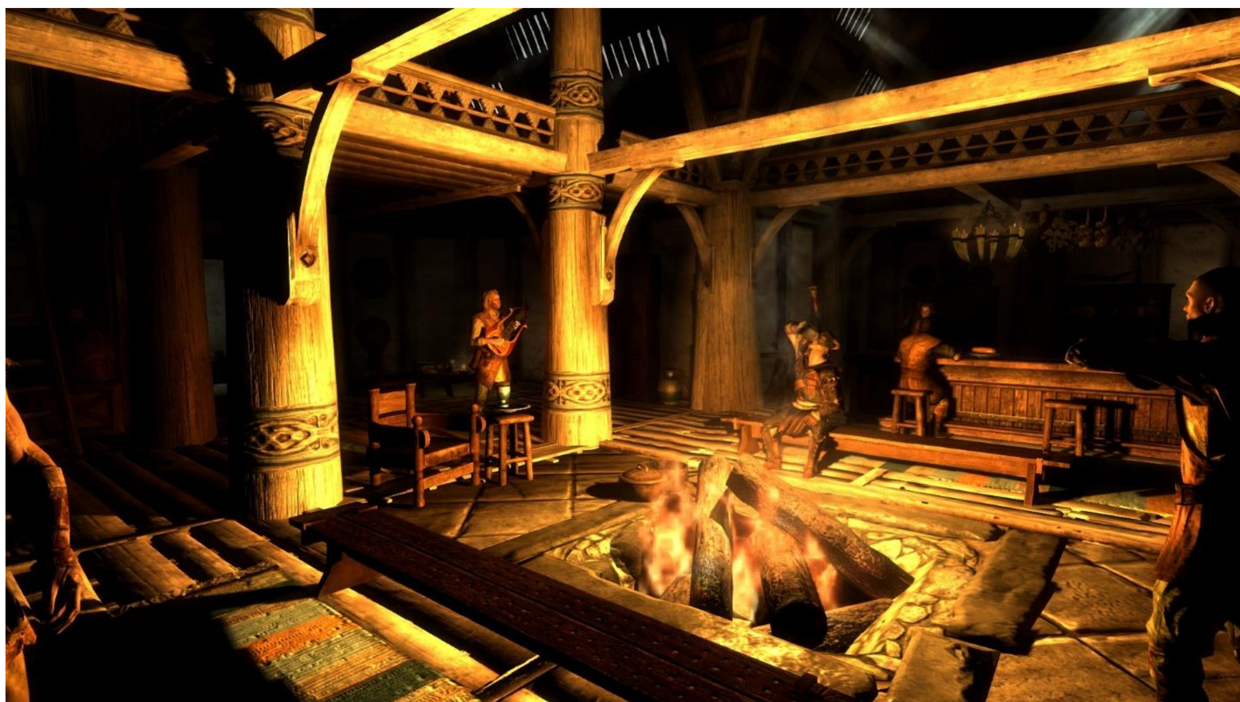

Рисунок 1.5 – пример поведения горожан в пабе

Проработанная система менеджмента сюжета позволяет разработчикам смешивать созданные вручную задания с заданиями, которые могут быть сгенерированы случайно из различных условий, кроме того задания могут появлятьсяв разном порядке и отличаться в зависимости от стиля прохождения игры.

Skyrim является продолжением серии The Elder Scrolls. Предыдущей игрой этой серии является The Elder Scrolls: Oblivion и Skyrim во многом похож и является ее продолжением. В Skyrim так же открытый мир со

свободным перемещением, так, что игрок может сам выбирать по какому пути пройти и какие локации изучать раньше, а какие позже.

Открытый мир дает большой простор вариативности прохождения игрока, но создает ряд проблем:

- сложность создания локаций;
- вероятность неправильной последовательности событий;
- необходимость оптимизации.

Поэтому, несмотря на большое количество игр с открытым миром на данный момент, графика в Skyrim имеет упор в реализм и большую детализацию. Все игровые объекты, здания, мебель, персонажи и оружие стилизованы под средневековье.

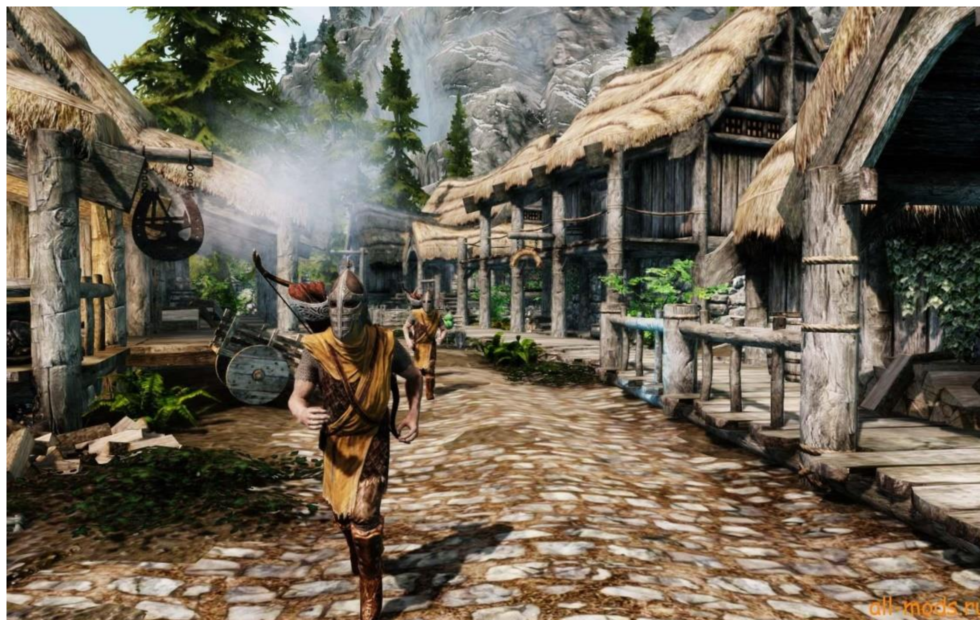

Рисунок 1.6 – пример графики игры «The Elder Scrolls V:Skyrim»

Такая графика приятна глазу и не отталкивает, но сложна в производстве.

Боевая система в Skyrim строится на физическом движке игры. При атаке (рис 1.5.), когда оружие персонажа прикасается к вражескому персонажу, вражескому персонажу передается урон в зависимости от базового урона, характеристик персонажа и особенностей оружия. Дальнее оружие работает по тому же принципу.

Рисунок 1.7 – демонстрация ближнего боя в Skyrim

Персонаж, получающий урон уменьшает его в зависимости от характеристик его сопротивляемости к данному типу урона. Так, например, огненные аттронахи имеют большую сопротивляемость к огненному урону, а маги имеют большую сопротивляемость к магическому урону.

Skyrim славится своей вариативностью действий и тем, что эти действия могут иметь последствия на историю в целом. Принимая сторону в конфликте вы решаете как сложится дальнейшая судьба игрового мира. В Skyrim есть редактор персонажа, игра позволяет выбрать одну из рас, каждая со своими особенностями, что идет в плюс к погружению в игровой мир. Skyrim обладает ключевыми особенностями стандартных RPG – вариативность отыгрыша, выбор, влияющий на историю, система навыков и улучшение характеристик персонажа.

Боевая система. Нанесение урона по противнику проверяется с помощью “хитбокса”, то есть некой “виртуальной коробки попадания”, попадание по которой рассчитывается с помощью внутриигрового движка персонажей и оружия или снарядов. Урон от оружия или снарядов рассчитывается с помощью характеристик атакующего персонажа и брони, принимающего урон.

Система улучшения характеристик персонажа в скайриме имеет два ключевых момента – навыки, визуально отображенных в виде созвездий (рис 1.7), каждое из которых представляет из себя отдельное древо навыков и общий параметр на каждое созвездие, представляющий из себя основной показатель характеристики, отвечающий за ее силу.

Рисунок 1.8 – древо навыков «The Elder Scrolls V:Skyrim»

Древа навыков лишь добавляют новые элементы игрового процесса, либо улучшают старые связанные с данным навыком, в то время как основной показатель характеристики отвечает за основную эффективность навыка. Основной показатель улучшается по мере использования соответствующих Такой подход дает обратную реакцию от стиля игры игрока, то есть игрок выбирает предпочтительный вариант игрового процесса и игра в этом его всячески поощряет. Такая система улучшения характеристик схожа с реальным миром, где мы со временем становимся лучше в том, что делаем. Для снижения урона в игре присутствует броня. Под броней подразумеваются кираса, шлем, перчатки и сапоги. Так же показатель брони поднимается путем экипировки щита, для которого имеется отдельная ветка навыков – блокирование. Сам показатель брони снижает исключительно физический урон, то есть урон от стрел, ударов мечом и так далее. Но в игре присутствует так же магия и различные виды урона, не относящиеся к физическому, а именно урон от огня, урон от холода, урон от яда, урон от молнии. Для их снижения на броню накладывают зачарования, снижающие процент от получаемого урона определенного типа.

Рисунок 1.9 – карта Тамриеля в игре «The Elder Scrolls V:Skyrim»

Skyrim имеет большой открытый мир, размерами 39 реальных квадратных километров. Это позволяет игроку самому выбирать в каком порядке играть. Оптимизация такого большого мира происходит за счет того, что части карты погружаются по мере передвижения персонажа прямо из физической памяти, за счет чего оперативной памяти не приходится держать в себе данные одновременно всех объектов на карте. Так же для оптимизации обрисовываются исключительно те объекты, на которые смотрит игрок, это позволяет сократить место на видеокарте и в оперативной памяти.

Большой проработанный Лор развивался с течением серий игр The Elder Scrolls, что, несомненно, является большим плюсом.

Хорошая боевая система, позволяющая на ходу менять заклинания и оружие, и заставляющая выбирать между снаряжением и ядами перед определенными противниками, например, тролль уязвим к огню, а маги зачастую носят одежду, дающую сопротивление магии.

The Witcher 3

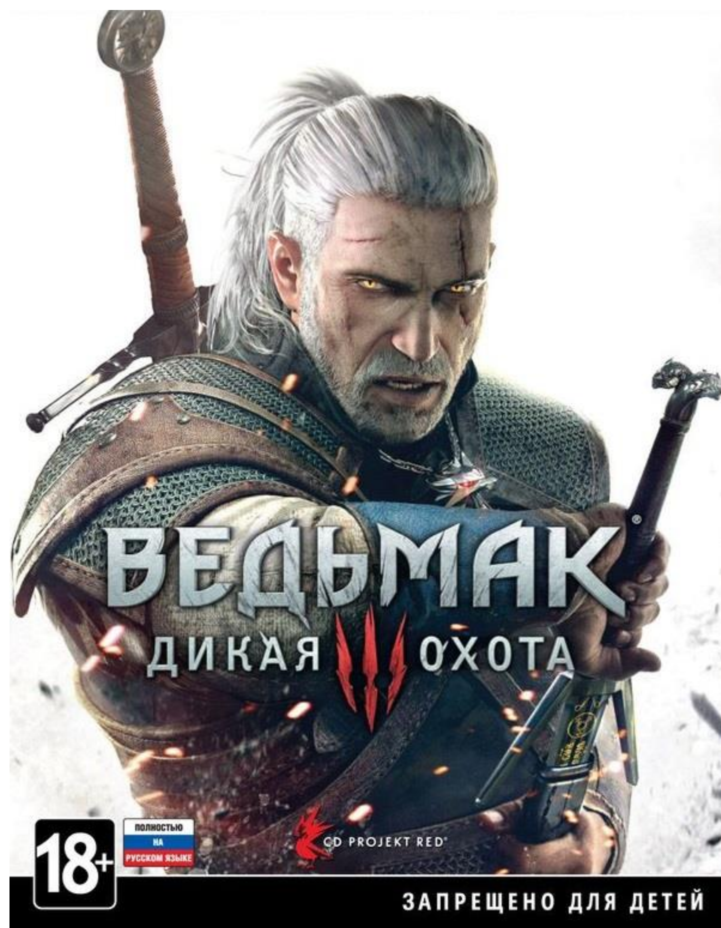

Рисунок 1.10 – обложка игры «The Witcher 3»

Witcher, или же Ведьмак является компьютерной игрой, созданной по серии книг Анджея Сапковского, повествующих о альтернативной реальности времен средневековья, где существует магия и магические создания, порой нападающие на людские селения, а защищать селения, то есть убивать монстров доверили Ведьмакам, людям, прошедшими через испытание, называемое «Испытание травами», превращенным в мутантов имеющих животную ловкость, силу, обоняние и зрение.

Безусловным плюсом данной игры является большой лор, воссозданный по книге, что делает игровой мир полнее и насыщеннее. Так же сценарий в таком случае имеет большую и твердую основу. Сюжет позволяет взглянуть на обиход людей самых разных чинов и рас от лица Геральта – главного героя. Геральт имеет серьезный характер, и предстоящие перед ним моральные выборы являются неоднозначными, благодаря чему любой выбор игрока имеет место быть и противоречит основному канону. В игре показана разница рас, вероисповеданий, из за которой порой происходят конфликты, что схоже с реальным миром. Например, церковь вечного огня, фанатично верящая в то, что всех инакомыслящих необходимо сжигать на костре, схожа со средневековым христианством с крестовыми походами и охотой на ведьм, когда женщин могли убить за рыжие волосы.

У персонажей имеется проработанный психологический портрет, благодаря чему им больше сопереживают и сюжет ощущается не наигранным, а наоборот, реальным.

Большой открытый мир позволяет самому прогуляться по окрестностям сказочных локаций, а так же дополнительные задания, встречающиеся на пути, не связанные с основным сюжетом, открывают вариативность действий и дают свободу выбора. Большой открытый мир при наличии последствий побочных заданий на основной сюжет влияет на сложность построения сюжетной линии, ведь игрок может раньше времени попасть на локацию, где к тому моменту он не должен находиться. разработчики пошли на хитрость и сделали так, что влияющие на сюжет дополнительные задания появляются только по мере сюжетной кампании и если зайти слишком далеко по основному сюжету, то автоматически считаются проваленными. Таким образом есть только основной сюжет и отдельные варианты сюжетных кампаний основного сюжета, а главные события происходят независимо от выбора игрока.

Боевая система. В Witcher имеется возможность как уничтожать врагов быстрыми и тяжелыми атаками меча, так и использовать «ведьмачьи знаки» - примитивные виды магии, позволяющие оттолкнуть, воспламенить, обмануть, замедлить или же защититься от противника. Так же на помощь приходят бомбы, делающие почти те же вещи.

Рисунок 1.11 – пример боевой ситуации в «The Witcher 3»

В данной игре используется автонаведение на врага для упрощения попадания по нему. Это значительно упрощает игровой процесс и позволяет сконцентрироваться на других аспектах игры. Сами же боевые ситуации

представляют себя случаи, когда на ведьмака нападают один или несколько врагов и задача в таких случаях правильно уворачиваться и успевать наносить удары по врагам. За счет легкости управления, это позволяет побеждать врагов куда сильнее ведьмака по уровню, однако это же и создает игровые ситуации, когда игрок может получить уровень и сильное оружие еще в начале. Такую проблему решает требования уровня у оружий и доспехов. За счет того, что игрок не может воспользоваться предметами выше своего уровня, у игрока пропадает желание нападать на большеуровневых врагов, тем самым спасая игровой процесс от ненужной напряженности .

Так же сильной стороной The Witcher 3 является удобный пользовательский интерфейс.

Интерактивные предметы всегда подсвечиваются легкой искрой, так что пользователь наверняка не пройдет мимо них.

Так же они выделяются желтым, если включено ведьмачье чутьё (рис 1.8).

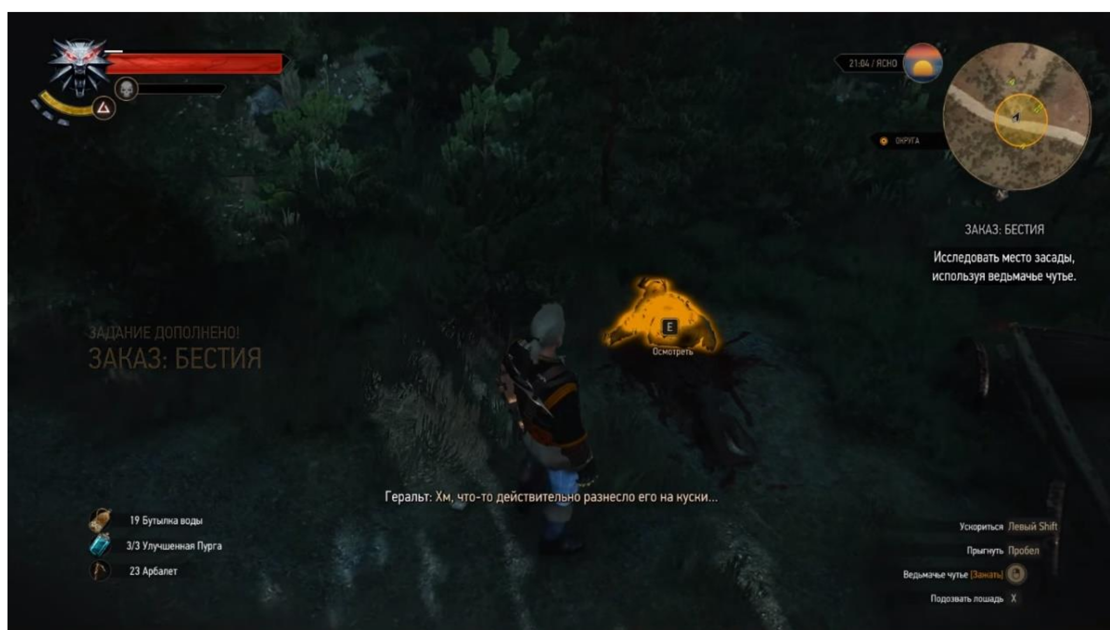

Рисунок 1.12 – подсвечивание интерактивных объектов ведмачьим чутьем.

При использовании ведьмачьего чутья так же видно следы, ведущие к квестовым ситуациям.

Рисунок 1.13 – подсвечивание квестовых объектов ведмачьим чутьем.

Fallout 3

Рисунок 1.14 – обложка игры «Fallout 3»

Fallout 3 – компьютерная игра в жанре Action RPG с открытым миром от студии Bethesda. Действие игры разворачивается в постапокалиптичном мире, в альтернативной версии истории, когда в ходе третьей мировой крупнейшие державы пустили в ход атомные бомбы, выжигая всю землю и загрязняя ее радиацией.

Игра начинается с того, как игрок выходит из убежища, защищенного от ядерных атак. В ходе исследования окружающего мира он замечает как сильно преобразилась природа, растительность и животные под действием радиации. Радиация превращает существ в мутантов, наделяя их необычными

способностями, что позволяет пустить в ход фантазии авторов, добавляя необычные игровые механики.

Fallout является типичным представителем жанра RPG - тут есть улучшения характеристик, свобода выбора, влияющая как на игровой процесс, так и отражающаяся на игровом мире и даже персонаж, создаваемый игроком с нуля, начиная с внешности и пола.

Для отображения большей части интерфейса было придумано необычное решение, вписывающееся в общую атмосферу игры – pip boy.

Рисунок 1.15 – наручный КПК в игре «Fallout 3»

Pip boy представляет из себя наручный КПК, имеющий несколько разделов:

- СТАТ — информация о параметрах, навыках и способностях персонажа;
- СНАР — все предметы, которые несёт сейчас персонаж;
- ИНФО — карты, информация о квестах, заметки, радиоприёмник.

В меню «Стат» (статистика) пять разделов: «Статус», где находится основная информация о здоровье персонажа, «SPECIAL», о котором пойдет речь дальше, «Навыки», «Способности», «Общее».

В меню «СНАР» (снаряжение) пять разделов: «Оружие», где показано всё имеющееся у персонажа оружие, «Костюмы», где перечислено всё, что можно носить: одежда, броня и аксессуары — например, шляпы, очки и т. д., «Помощь», где показано всё, что персонаж может использовать для изменения собственных параметров: стимуляторы, химикаты, любые книги и журналы, раздел «Разное» в котором показаны разнообразные объекты: ключи, заколки,

компоненты, мусор и пр., и «Боезапас» где показаны все имеющиеся у персонажа боеприпасы.

Уникальной особенностью серии Fallout является то, что характеристики влияют не только на стиль ведения боя и бой в целом, но и даже на диалоги и сюжетные выборы. Так, например, имея большую харизму если на вас напали, можно обхитрить врагов и не просто остановить бой, но и получить в качестве извинения от врага местную валюту – крышки.

Рисунок 1.16 – система SPECIAL в pip boy

Особенность Fallout – система характеристик SPECIAL, каждая из которых отвечает за параметры производных характеристик. Всего основных характеристик семь, начальные буквы которых и образовали название системы – S.P.E.C.I.A.L:

- Strength (Сила);
- Perception (Восприятие);
- Endurance (Выносливость);
- Charisma (Привлекательность);
- Intelligence (Интеллект);
- Agility (Ловкость);
- Luck (Удача).

Они отражают навыки персонажа в определённых аспектах игры и могут принимать значения от 1 до 10 для людей и до 16 для других рас.

Они впервые задаются в самом рождении персонажа, словно природные таланты и могут быть улучшены по мере увеличения уровня. Количество начальных характеристик – 40 для людей.

Влияние основных атрибутов S.P.E.C.I.A.L. на производные атрибуты и навыки:

a) Сила (Strength):
 - переносимый вес (Carry Weight);
 - холодное оружие (Melee Weapons).

b) Восприятие (Perception):
 - время появления красных меток на компасе (Compass Markers);
 - энергооружие (Energy Weapons), Взрывчатка (Explosives), Взлом (Lockpick).

c) Выносливость (Endurance):
 - здоровье (Health), Сопротивляемости (Resistances);
 - тяжелое оружие (Big Guns), Без оружия (Unarmed).

d) Харизма (Charisma):
 - расположение (Disposition);
 - бартер (Barter), Красноречие (Speech).

e) Интеллект (Intelligence):
 - число очков навыков при достижении нового уровня (Skill Points per Level);
 - медицина (Medicine), Ремонт (Repair), Наука (Science).

f) Ловкость (Agility):
 - количество очков действия (Action Points);
 - легкое оружие (Small Guns), Скрытность (Sneak).

g) Удача (Luck):
 - шанс критического попадания (Critical Chance);
 - все навыки.

Такая система позволяет создавать большое количество разнообразных игровых ситуации и делает игровой опыт каждого игрока уникальным.

Бой в Fallout 3, в отличие от прошлых игр серии Fallout, может вестись как в обычном шутере — «в режиме реального времени», так и в специальном режиме V.A.T.S..

Оружие разделяется на несколько типов:

- стрелковое (лёгкое, тяжелое и энергооружие);
- холодное;
- кастеты;
- гранаты;
- мины.

Урон, наносимый оружием зависит от навыков персонажа, состояния самого оружия и устойчивости противника к типу урона оружия. Как и во

всех шутерах, для нанесени урона противнику достаточно прицелиться из оружия по нему и нажать кнопку мыши.

При использовании автоматического оружия достаточно зажать кнопку мыши, чтобы вести непрерывный огонь. Оружие перезаряжается автоматически при истечении патронов. Но чтобы перезарядить оружие в нужный вам момент, нажмите R. Урон оружия и даже точность зависит от соответствующих навыков.

В Fallout имеется такая характеристика оружия как состояние, отражающая уровень износа. Состояние оружия также влияет на количество наносимого урона, равно как и на вероятность, что оружие заклинит при перезарядке. Починка оружия может увеличить количество наносимого им урона и понизить вероятность заклинивания. С помощью правой кнопки мыши вы можете целиться и стрелять точнее, но медленнее. Если вы прячетесь, ваш персонаж будет двигаться медленнее, зато у него появится возможность провести скрытную атаку по любой цели, которой он ещё не был замечен.

Fallout 3 в отличие от предыдущих двух частей, где графика изометрическая, имеет 3D графику и объемный открытый мир, в чем есть как положительные так и отрицательные стороны. С одной стороны 3D мир открывает новые возможности для исследования и добавляет реализма, но с другой, 3D имеет существенные недостатки, а именно:

1) Сложность изготовления графики, ведь приходится помимо самих текстур работать над 3D моделями и всеми вытекающими из этого сложностями;
2) Большинство современных игровых движков не имеет достаточно реалистичной физики твердых тел, и дело не в недостатках движка, а в отсутствии необходимости добавлять во многие игры такие аспекты как деформацию, нагрев, симуляцию жидкости в виду нецелесообразности таких производственных затрат и уменьшения производительности при их использовании;
3) Большая вероятность возникновения ошибок, например в случаях, когда игрок проникает туда, куда не должен был попасть.

Fallout 3 использует улучшенную версию движка Gamebryo, использовавшемся в The Elder Scrolls: Oblivion. Как раз в этом движке имеется ошибка, позволяющая перемещаться быстрее положенного.

1.3 Постановка задачи

Я планирую создать компьютерную игру, содержащую основные аспекты игры в жанре RPG. Главным достоинством данной игры будет сочетание физики и механики боя, а так же большая интерактивность игрового мира.

Для этого игре придется пройти через несколько стадий:

1) Демо-версия, показывающая основные игровые механики, но не являющаяся полноценным игровым продуктом;

2) Альфа версия, показывающая реальный игровой процесс, но являющееся предварительной версией, не прошедшей массовое тестирование;

3) Бета версия, развивающаяся вместе с ростом игрового сообщества;

4) Полноценная платная версия.

В демо-версии я планирую показать основные механики игры, такие как:

- Инвентарь и подбор предметов;
- Боевая система;
- Смена анимаций при разном оружии;
- Возможность ближней атаки;
- Возможность дальней атаки;
- Возможность модификации урона оружия;
- Возможность модификации здоровья персонажа;
- Искусственный интеллект противников.

В альфа-версии будут доступны и доработаны следующие аспекты:

- Возможность сохранения и загрузки прогресса игры;
- Возможность смены персонажа;
- Возможность улучшения характеристик посредством древа навыков;
- Возможность открывать чужой инвентарь и сундуки;
- Возможность улучшения навыков и взаимодействие навыков с характеристиками персонажа, прямые и обратные зависимости;
- Возможность рыбачить и охотиться;
- Собирательство;
- Искусственный интеллект групп;
- Искусственный интеллект горожан;
- Возможность выполнения миссий;
- Основной сюжет;
- Улучшение графической составляющей;
- Звуковая составляющая;
- Выбор профессий;
- Дополнительные внесюжетные миссии;
- Модификации оружия;
- Экипировка, броня;
- Заклинания.

В демо версии будет присутствовать все от альфа версии, но по мере увеличения числа пользователей и роста активной части сообщества будут вноситься правки в игровую составляющую, как исправление ошибок, так и добавление или удаление функций.

2 Проектирование системы

2.1 Диаграммы. Распределение ролей

1) Начальное представление о модификаторах, для возможности изменять различные параметры с возможностью отката:

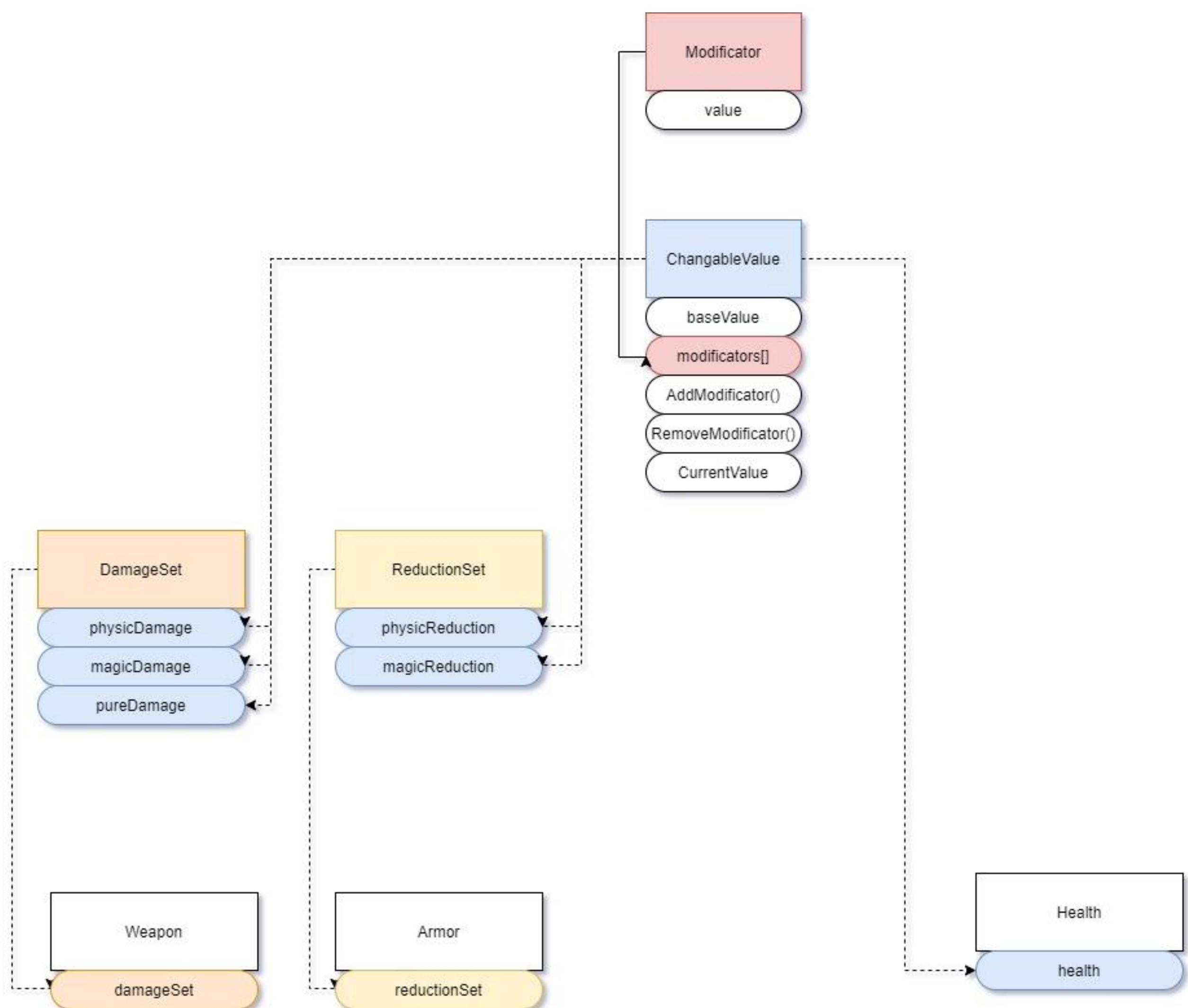

Рисунок 2.1 – Начальное представление о модификаторах

2) Изначальное представление о событиях внутри Body:

Рисунок 2.2 – Изначальное представление о событиях внутри Body

3) Изначальное представление о функциях эффектов, срабатывающих с течением времени:

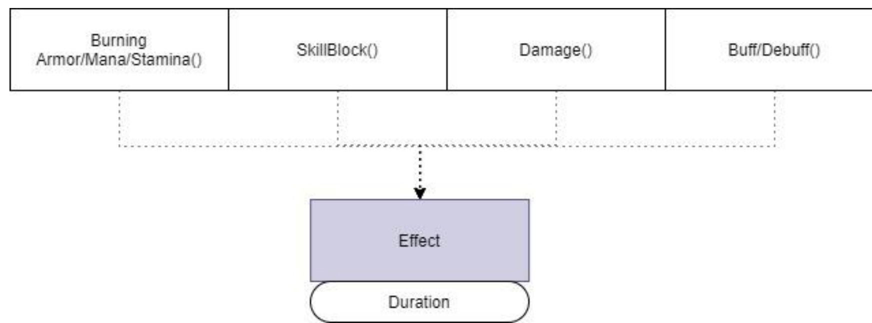

Рисунок 2.3 – Изначальное представление о функциях эффектов

4) Изначальный вид BodyBase:

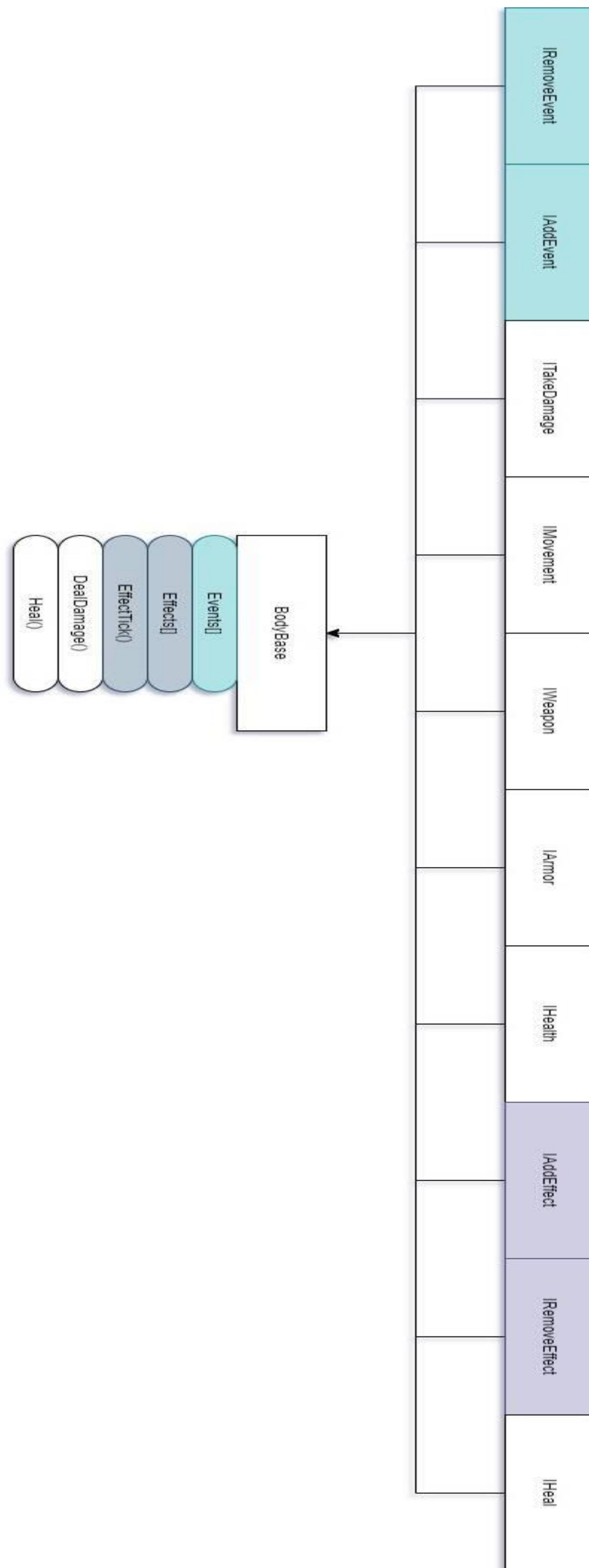

Рисунок 2.4 – Изначальный вид BodyBase

Где BodyBase является основным представлением персонажа, в дальнейшем был разделен на две отдельные части – Body, содержащий пассивную часть и Character, содержащий активную часть персонажа.

Предпологалось использование интерфейсов для любых взаимодействий между BodyBase и окружающим миром, но для упрощения разработки на данный момент все происходит непосредственно через экземпляр BodyBase.

ChangableValue был придумал для расширения возможности реализации модификаций параметров, например, здоровья, урона, брони.

Ивенты, или события были придуманы для возможности реализации различных действий, срабатывающих при возникновении события.

Цепочка взаимодействий для атаки и нанесения урона:

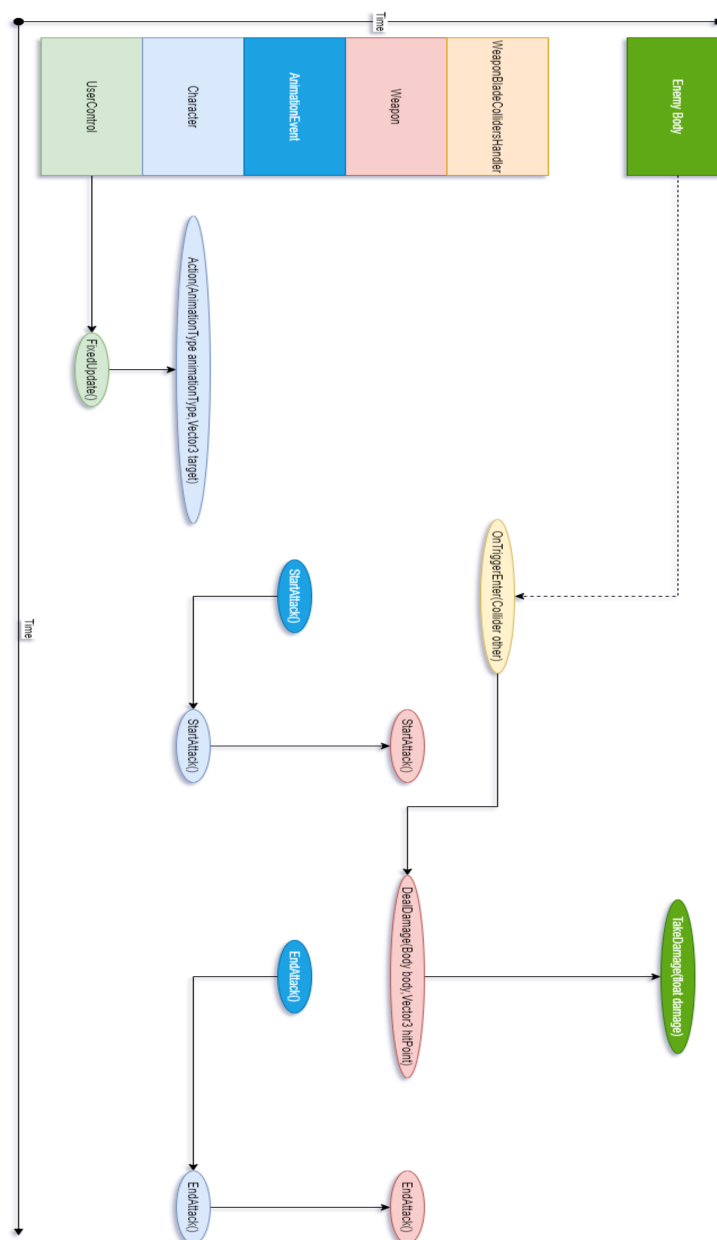

Рисунок 2.5 – Цепочка взаимодействий для атаки и нанесения урона

Шкала времени показывает прогресс выполнения функций.

Представление о цепочке взаимодействий для получения экземпляра Attack и анимации атаки, во время атаки:

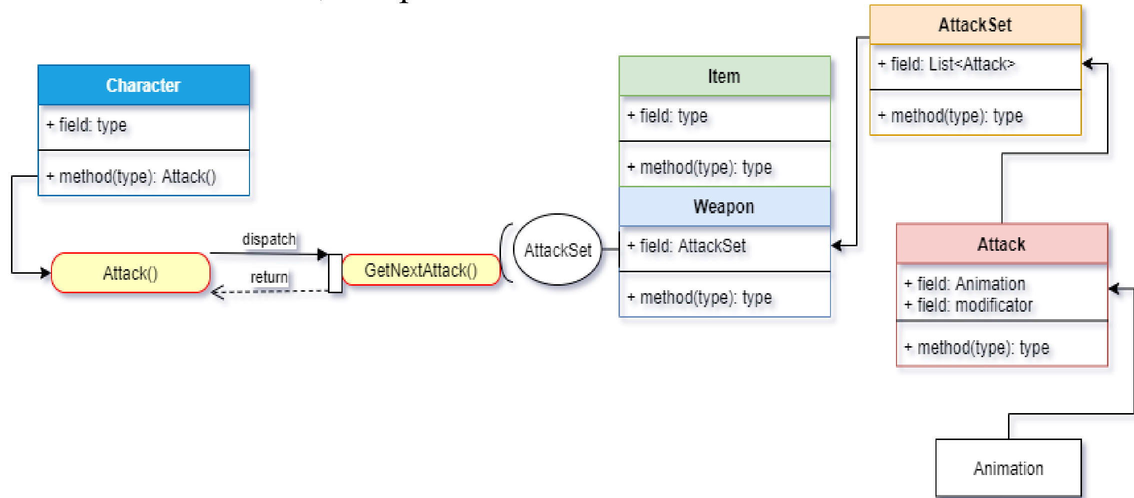

Рисунок 2.6 – Представление о цепочке взаимодействий для получения экземпляра Attack

Раннее представление о skill компоненте, являющемся навыком древа навыков:

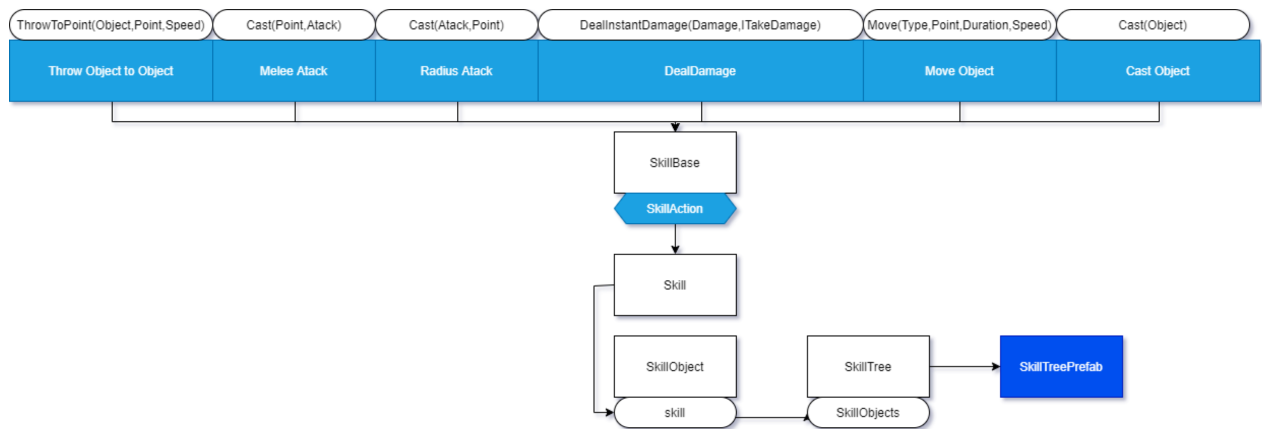

Рисунок 2.7 – Раннее представление о skill компоненте

2.2 Выбор игрового движка. Язык программирования

Для работы с кодом была выбрана программа Visual Studio 2019, значительно упрощающая редактирование кода.

Рисунок 2.8 – Эмблема Visual Studio.

Она позволяет редактировать код значительно быстрее и обладает множеством функций, среди которых:

- подсказки при названии переменных и функций;
- автоматическое соблюдение визуальной чистоты кода, формат кода под стандарты написания;
- автоматическое определение ошибок компилятора во время написания кода;
- нахождение источника элемента при помощи специальных клавиш.

Я выбрал Visual Studio Enterprise, поскольку это бесплатная версия.

Игровой движок. В настоящее время существует множество движков для компьютерных игр. Они облегчают построение игровой логики и обычно имеют в себе готовый физический и графический движки. Под физическим движком понимается набор алгоритмов, прописанный в программе, позволяющий имитировать полностью или частично поведения объектов в реальном мире, такие как столкновения, гравитация, трение, скорость, ускорение, вращение, деформации, а так же иногда позволяют моделировать поведение сложных объектов и субстанций, например жидкости.

Рисунок 2.9 – Эмблемы популярных игровых движков.

Самыми популярными являются Source, Unreal Engine и Unity3D.

Рисунок 2.10 – Эмблема Unreal Engine

Unreal Engine характерен своей большой производительностью, благодаря тому, что он использует язык C++. Unreal Engine на данный момент является прямым конкурентом Unity 3D.

Ввиду стоимости подписки на Unreal Engine и того, что движок Source не является коммерческим продуктом, а используется только компанией Valve, выбор пал на Unity3D. Unity является кроссплатформенным игровым движком, поддерживающим как 3d, так и 2d, и ввиду легкости освоения является популярным среди начинающих разработчиков. Unity поддерживает два языка программирования – C# и собственный язык UniScript, напоминающий JavaScript. Выбор пал на C#, язык входящий в пятерку самых популярных языков программирования.

Рисунок 2.11 – эмблема Unity

Unity. Unity имеет в себе графический и физический движок. Физический движок Unity может просчитывать коллизии объектов любых

форм, симулировать физику реального мира, на основе законов физики в виде коллизии, трения, ускорения, упругости и все это как в трехмерном пространстве, так и в двухмерном. Графический движок необходим, чтобы отрисовывать изображения объектов в трехмерном пространстве с учетом света, теней, прозрачности и различных эффектов вроде замыливания и свечения.

Unity 3D это бесплатный игровой движок, с готовой графикой и физикой, позволяющий работать с этими компонентами в рамках игрового мира. Unity простой в освоении и использовании, но обладает большим потенциалом.

Unity имеет встроенную оптимизацию графической части, когда те объекты, на которые в данный момент не направлена камера, не отрисовываются. Физика Unity использует формулы законов физики реального мира и позволяет обрабатывать более десяти тысяч твердых тел одновременно.

Так же приятным бонусом идет то, что движок работает с языком C#, простым в освоении, но многофункциональным и модульным языком программирования, построенным на платформе .net.

Выбор пал на Unity 3D.

Unity имеет удобный и приятный графический интерфейс. Панели инструментов можно масштабировать, расширять и сжимать, а так же перемещать, отрывать от главного окна и закреплять вместе с побочными окнами.

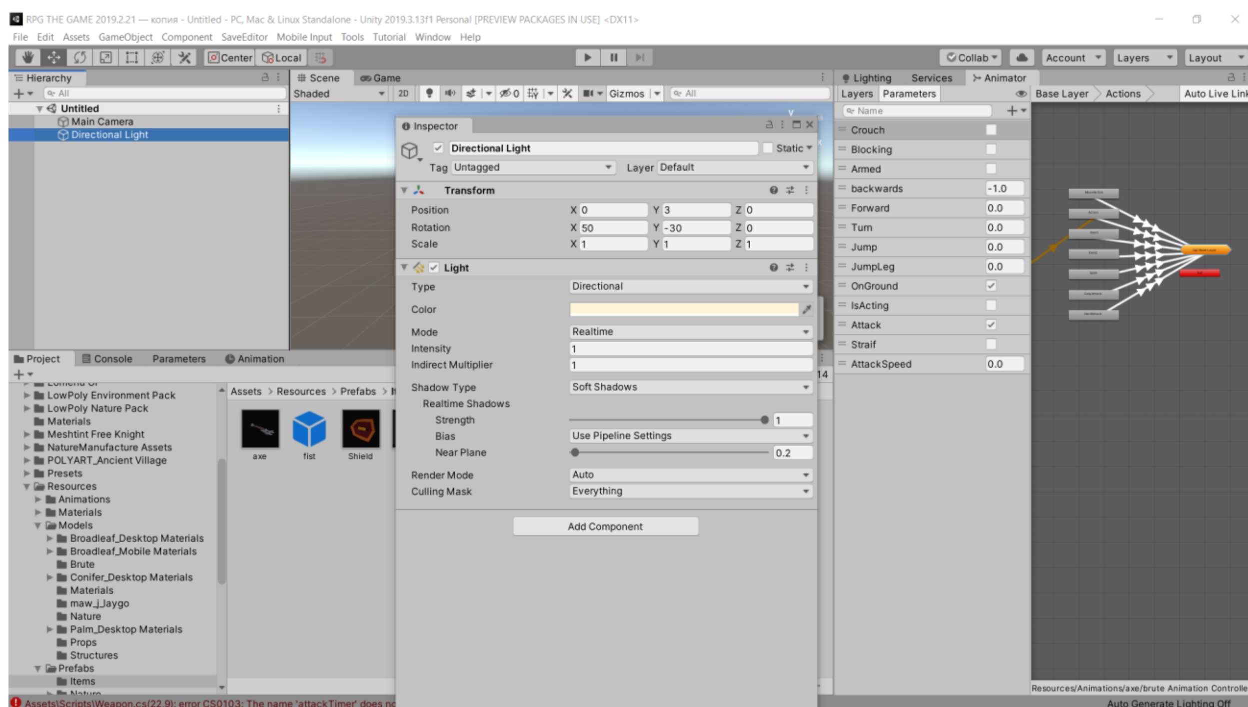

Рисунок 2.12 – демонстрация окна редактора Unity

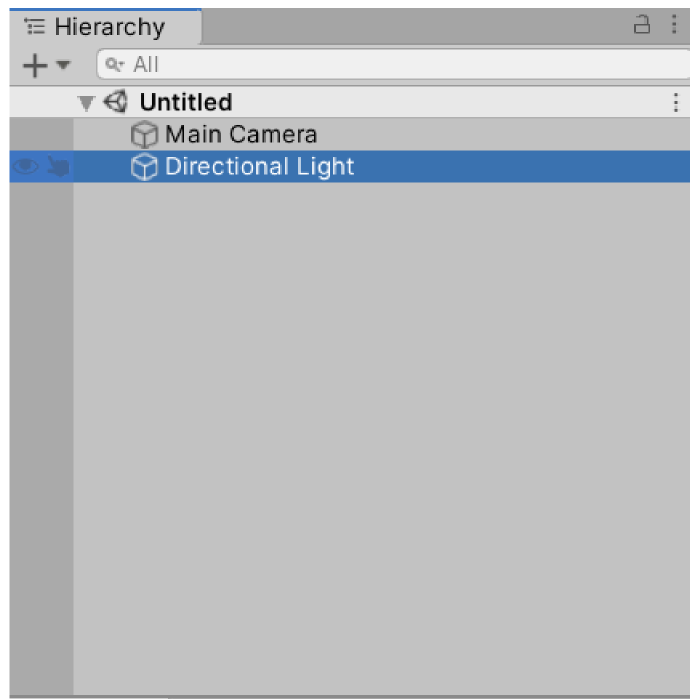

Рисунок 2.13 – окно иерархии

Окно иерархий отвечает за иерархию объектов на сцене, так, при перемещении и вращении объекта-родителя перемещаются и вращаются дочерние объекты. При уничтожении объекта-родителя, дочерние объекты так же уничтожаются.

В окне иерархий находятся GameObject(game – игровой, object – объект) объекты, представляющие из себя игровые объекты существующие на сцене, под игровыми объектами понимаются любые объекты, локализированные на сцене типа GameObject. Transform, отвечающий за вращение и перемещение объекта, его локальное и глобальное положение является неотъемлемой частью GameObject и не может создаваться извне.

GameObject объекты являются носителями компонентов, характеризующие их.Так, например, источник освещения, солнце это GameObject, содержащий на себе компонент Light, позволяющий освещать объекты и взаимодействующий с визуальным движком.

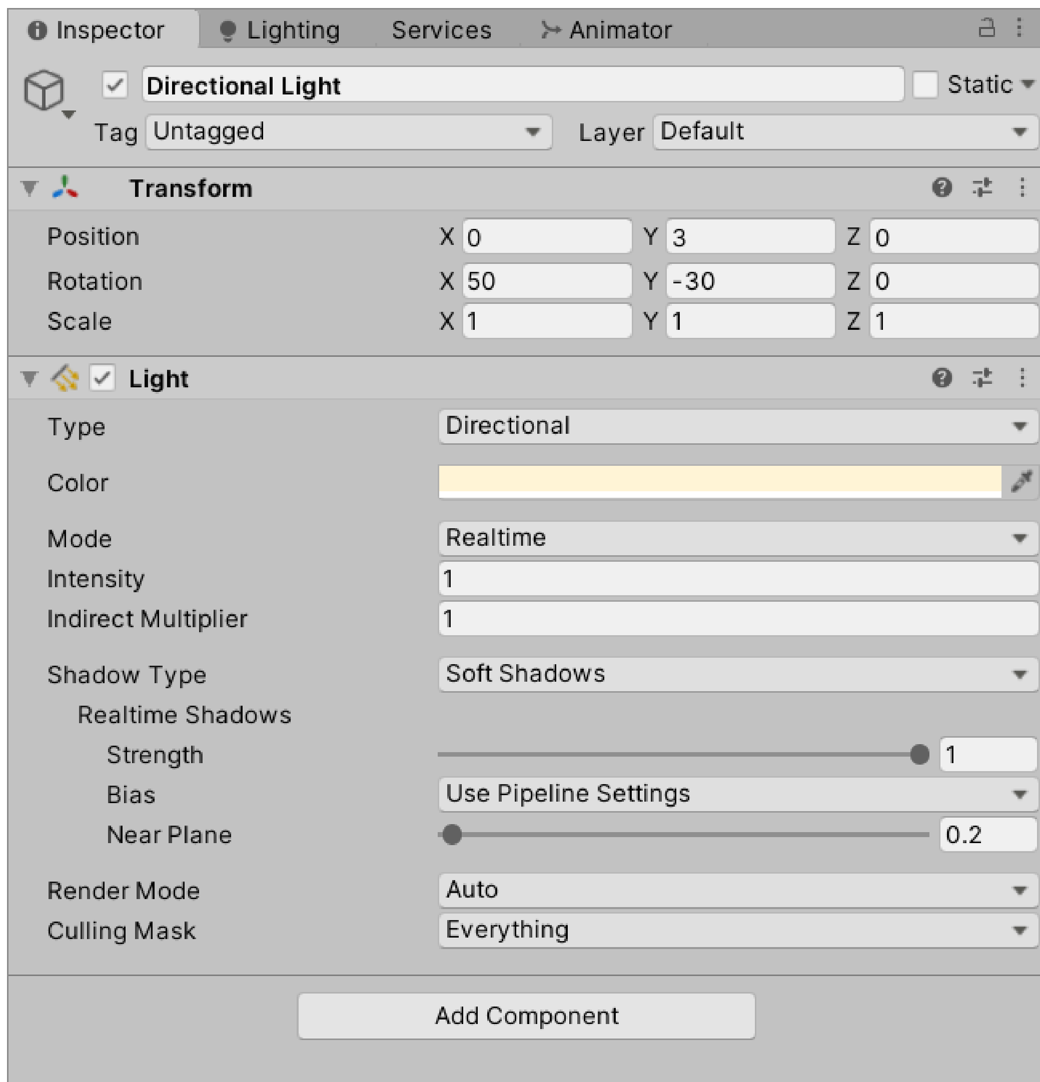

Рисунок 2.14 – Окно инспектора

Unity предоставляет возможность создания собственных компонентов-скриптов, за логику которых отвечает код. Такие компоненты наследуют от MonoBehavior и являются классами в C#.

Рисунок 2.15 – скриптовый компонент

Unity имеет собственный оптимизированный физический и графический движок, с которыми можно взаимодействовать через собственный код.

Для взаимодействия с физическим движком на GameObject-ы устанавливаются соответствующие компоненты – Rigidbody, отвечающий за логику твердого тела, Collider, отвечающий за обнаружение столкновений и т.д.

Так же, взаимодействовать с физикой можно как через компоненты, так и через встроенный класс UnityEngine.Physics.

Для взаимодействия с графическим движком в Unity предусмотрена возможность создания собственных шейдеров, обрабатываемых через код и постобработки, влияющей на итоговое изображение. Код для написания шейдеров пишется на отдельном языке ShaderLab, поддерживающем вставки GL и HLSL.

Так же Unity поддерживает возможность установки Shader Pipeline вместо того, что имеется по умолчанию. При помощи Package Manager имеется возможность установки модификаций в движок Unity и в сам редактор. Например, таким образом можно загрузить Light Weight Render Pipeline или High Definition Render Pipeline, улучшающие графику и позволяющие графически создавать и модифицировать шейдеры. При этом создается граф Shader Graph, от которого в свою очередь производятся материалы, с различными параметрами, что позволяет создавать один шейдер на разные материалы, не отличающиеся друг от друга методом взаимодействия с графикой.

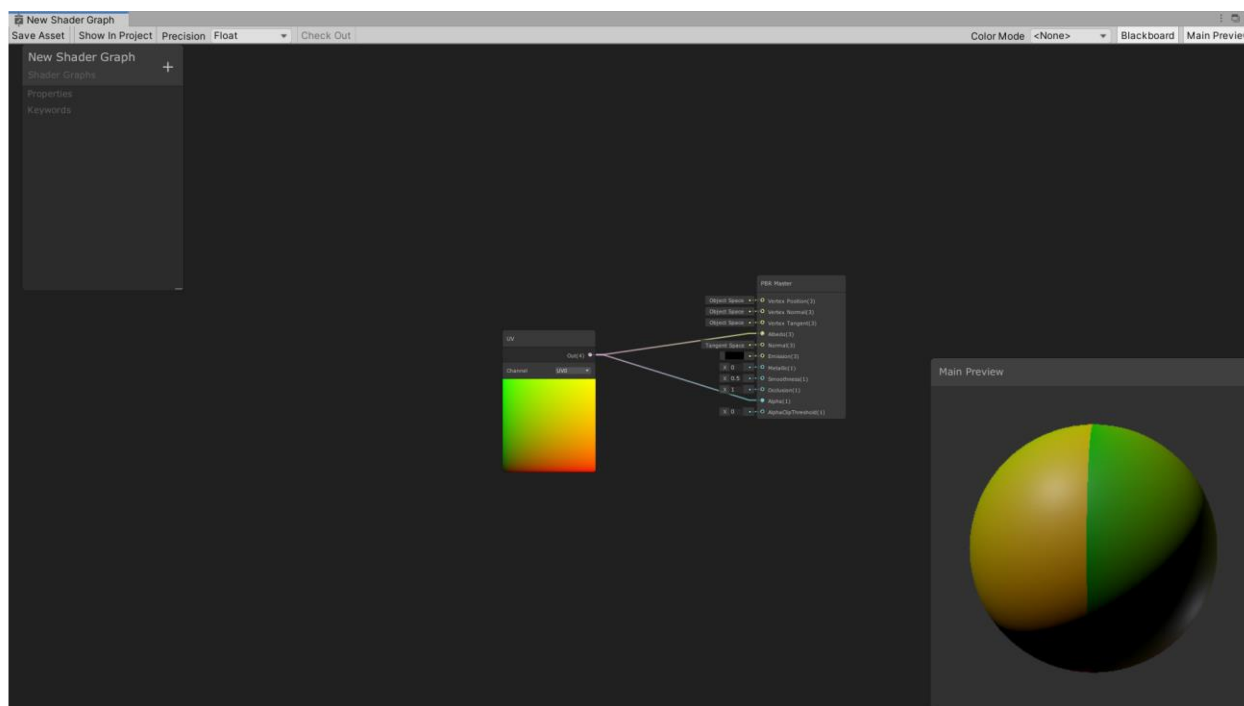

Рисунок 2.16 – окно создания shader graph-а

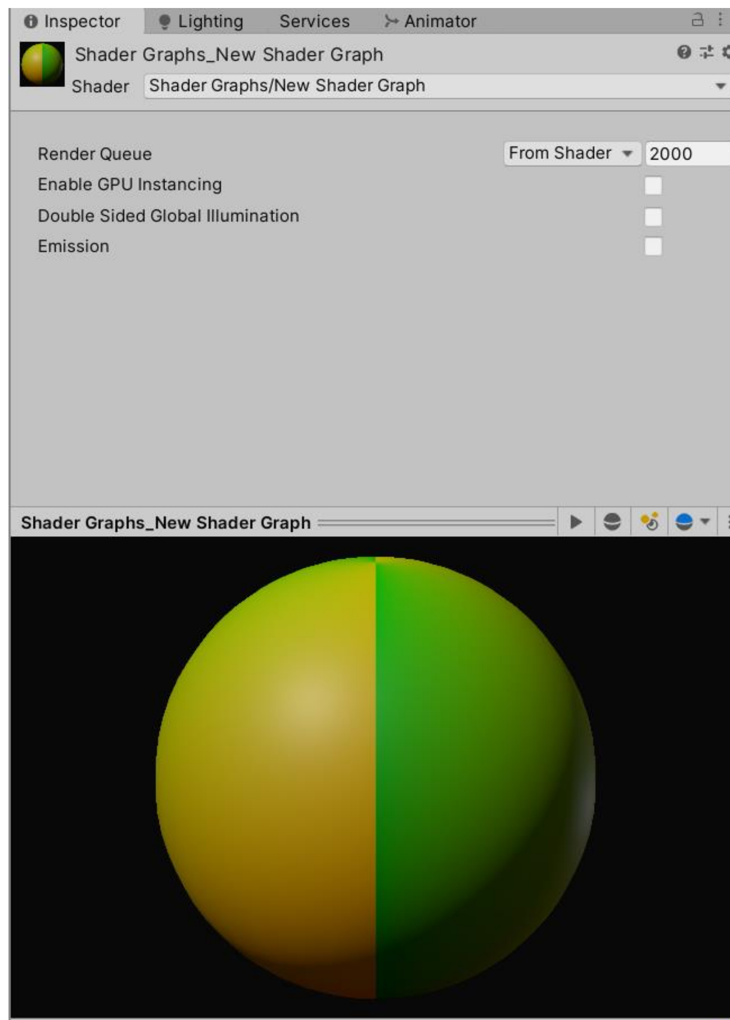

Рисунок 2.17 – Материал – результат работы шейдера

Рисунок 2.18 – Лого языка C#

Из множества языков программирования был выбран язык C#.

C# это кроссплатформенный язык программирования, позволяющий написать что угодно, начиная от приложения для рабочего стола и веб сервера, заканчивая собственной операционной системой.

Благодаря обилию различных библиотек под этот язык, не приходится писать весь код с нуля.

Кроссплатформенность обеспечивается особым способом компиляции – сначала код переводится в промежуточный язык, который преобразуется в машинный код уже во время выполнения приложения. Такой подход является более ресурсоемким, по сравнению с простой компиляцией, но значительно упрощает перенос приложения с одной платформы на другую. Язык C# является флагманским языком Microsoft, и они регулярно выпускают обновленные версии языка, которые обычно не меняют синтаксис, благодаря чему код, написанный на более ранних версиях языка можно использовать в более поздних. Язык работает на базе платформы .Net Framework, имеющей множество библиотек для самых различных задач.

Объектно-ориентированный подход в программировании подходит как для описания объектов, так и абстракций.

3 Программная реализация

3.1 Программа

В Unity запуск приложения начинается со сцены. Сцена представляет собой визуально отображаемую локацию, на которой разворачивается действия программы и содержит объекты в локации и их иерархию. Такие объекты называются “GameObject”, то есть “игровой объект”. Такие объекты содержат информацию о своем глобальном и локальном местоположении, размере и вращении, и, самое главное, к ним могут крепиться компоненты, например компонент отрисовки 3d модели, спрайта, компонент “коллайдера”, освещение и скриптовые компоненты, наследуемые от класса “MonoBehavior”, для которого прописан свой интерфейс взаимодействия со средой и GameObject носителем в первую очередь.

Следовательно, все пользовательские скрипты, отвечающие за определенные свойства объектов GameObject, будут наследоваться от MonoBehavior.

Скрипты, отвечающие за хранение информации будут наследоваться от компонента ScriptableObject, который позволяет использовать [CreateAssetMenu()] для создания файла с форматом .asset в папке ресурсов, - “Resources/”.

Для упрощения разработки и более быстрых тестов работоспособности будем использовать производные от класса Editor.

В целом, игровую логику можно разбить на такие наиболее обособленные части как:

1) Боевая система.
2) Интерактивность окружающего мира.
3) Сюжет, задания, персонажи и их реплики.
4) Визуальная составляющая.
5) Звуковая составляющая.

В Unity запуск приложения начинается со сцены. Сцена представляет собой визуально отображаемую локацию, на которой разворачивается действия программы и содержит объекты в локации и их иерархию. Такие объекты называются “GameObject”, то есть “игровой объект”. Такие объекты содержат информацию о своем глобальном и локальном местоположении, размере и вращении, и, самое главное, к ним могут крепиться компоненты, например компонент отрисовки 3d модели, спрайта, компонент “коллайдера”,

освещение и скриптовые компоненты, наследуемые от класса “MonoBehavior”, для которого прописан свой интерфейс взаимодействия со средой и GameObject носителем в первую очередь.

Следовательно, все пользовательские скрипты, отвечающие за определенные свойства объектов GameObject, будут наследоваться от MonoBehavior.

Скрипты, отвечающие за хранение информации будут наследоваться от компонента ScriptableObject, который позволяет использовать [CreateAssetMenu()] для создания файла с форматом .asset в папке ресурсов, - “Resources/”.

Для упрощения разработки и более быстрых тестов работоспособности будем использовать производные от класса Editor.

В целом, игровую логику можно разбить на такие наиболее обособленные части как:

1) Боевая система.
2) Интерактивность окружающего мира.
3) Сюжет, задания, персонажи и их реплики.
4) Визуальная составляющая.
5) Звуковая составляющая.

Основным компонентом, участвующим в бою является компонент Body. Он содержит пассивную часть логики для каждого конкретного объекта, то есть получение урона, количество здоровья, действия при “смерти”.

Body Содержит поля:

1) currentHealth, представляет собой количество здоровья объекта;

```
public float currentHealth;
```

2) IsAlive, показывающий, жив ли объект;

```
ссылок: 0
public bool IsAlive { get; private set; } = true;
```

3) animator, Animator компонент, привязанный к данному GameObject-у;

```
Animator animator;
```

4) defaultPushTimer, таймер длительности толчка по умолчанию, необходим для корректной работы отталкивания персонажа;

```
float defaultPushTimer = 0.3f;
```

5) pushTimer, таймер толчка, необходим для отсчитывания времени, в которое не работает аниматор, позволяя толкать персонажа по горизонтали;

```
float pushTimer;
```

6) isPushing, если true, то в данный момент объект толкается;

```
bool isPushing;
```

7) capsuleCollider, CapsuleCollider компонент данного GameObject-а;

```
CapsuleCollider capsuleCollider;
```

8) rigidbody, Rigidbody компонент данного GameObject-а;

```
Rigidbody rigidbody;
```

9) ragdollColliders, список компонентов Collider данного GameObject-а, используемые для реалистичной физики тряпичной куклы;

```
List<Collider> ragdollColliders = new List<Collider>();
```

10) ragdollRigidbodies, список компонентов Rigidbody данного GameObject-а используемые для реалистичной физики тряпичной куклы.

```
List<Rigidbody> ragdollRigidbodyes = new List<Rigidbody>();
```

И методы:

- Start, наследуемый метод от MonoBehavior, вызывается в момент создания экземпляра класса, сразу после инициализации GameObject-а.

В данном случае происходит цепочка событий:

- в GameDataOptimizer, созданный для оптимизации поиска объектов вносится данный компонент Body;

- полю animator присваивается ссылка на Animator компонент данного GameObject-а;

- полю rigidbody присваивается ссылка на Rigidbody компонент данного GameObject-а;

- полю capsuleCollider присваивается ссылка на CapsuleCollider компонент данного GameObject-а;

- текущее здоровье становится равным 100;

- полю ragdollColliders присваивается список Collider компонентов данного GameObject-а;

- полю ragdollRigidbodies присваивается список Rigidy компонентов данного GameObject-а;

- вызывается метод EnableRagdoll, с параметром false, выключая физику тряпичной куклы.

```
ссылок: 0
void Start()
{
    GameDataOptimizer.InsertValue(this);
    animator = GetComponent<Animator>();
    rigidbody = GetComponent<Rigidbody>();
    capsuleCollider = GetComponent<CapsuleCollider>();


    currentHealth = 100;

    ragdollColliders = new List<Collider>(GetComponentsInChildren<Collider>());
    ragdollRigidbodyes = new List<Rigidbody>(GetComponentsInChildren<Rigidbody>());

    EnableRagdoll(false);
}
```

1) FixedUpdate(), наследуемый от MonoBehavior, вызываемый каждую фиксированную еденицу времени, не зависящую от частоты кадров.

В данном методе происходит обработка толчка персонажа;

```
rigidbody.AddForce(delayedForce, ForceMode.Impulse);
delayedForce = Vector3.zero;
pushTimer -= Time.fixedDeltaTime;
if (pushTimer <= 0)
    isPushing = false;
if (isPushing)
{
    animator.applyRootMotion = false;
}
```

2) TakeDamage(float Damage), получает значение урона и отнимает его от текущего здоровья, а так же для визуальности создает текстовый объект, показывающий количество полученного урона;

При истечении количества здоровья, срабатывает метод Die()

```
public void TakeDamage(float damage)
{
    currentHealth -= damage;
    var dn = Instantiate<GameObject>(Resources.Load<GameObject>(DamageNumber.Path)).GetComponent<DamageNumber>();
    dn.text = damage.ToString();
    Vector3 v3 = transform.position;
    v3.y += 2;
    dn.transform.position = v3;
    if (currentHealth<=0)
    {
        Die();
    }
    Debug.Log("Body.TakeDamage()");
}
```

3) Die(), при срабатывании которого отключается Animator падает и объект считается мертвым.

```
ссылка: 1
void Die()
{
    Debug.Log("Body.Die()");
    EnableRagdoll(true);

}
ссылка: 1
```

Character отвечает за активную часть логики объектов, то есть передвижение, анимации и физику.

Character содержит поля:

```
[SerializeField] float m_MovingTurnSpeed = 360;
[SerializeField] float m_StationaryTurnSpeed = 180;
[SerializeField] float m_JumpPower = 12f;
[SerializeField] float m_RunCycleLegOffset = 0.2f; //specifi
[SerializeField] float m_MoveSpeedMultiplier = 1f;
[SerializeField] float m_AnimSpeedMultiplier = 1f;
[SerializeField] float m_GroundCheckDistance = 0.1f;
```

1) float m_MovingTurnSpeed, отвечает за скорость поворота персонажа во время ходьбы в градусах, по умолчанию 360;

2) float m_StationaryTurnSpeed , отвечает за скорость поворота при стационарном положении, по умолчанию 180;

3) m_JumpPower, сила прыжка, по умолчанию 12;

4) m_RunCicleOffset, смещение при завершении цикла ходьбы, разная для каждого персонажа;

5) m_MoveSpeedMultiplier, множитель скорости передвижения;

6) m_AnimSpeedMultiplier, множитель скорости анимации ходьбы;

7) m_GrounCheckDistance, дальность луча для проверки наличия земли под персонажем;

```
[SerializeField]
string defaultWeaponPath;

Weapon defautWeapon;

public Transform leftHand;
public Transform rightHand;
```

8) defaultWeaponPath, временная переменная отвечающая за расположение оружия по умолчанию для персонажа;

9) defaultWeapon, поле , для хранения оружия по умолчанию;

10) LeftHand, Transform левой руки;

11) RightHand, Transform правой руки;

```
Rigidbody m_Rigidbody;
Animator m_Animator;
bool m_IsGrounded;
float m_OrigGroundCheckDistance;
const float k_Half = 0.5f;
float m_TurnAmount;
float m_ForwardAmount;
Vector3 m_GroundNormal;
float m_CapsuleHeight;
Vector3 m_CapsuleCenter;
CapsuleCollider m_Capsule;
bool m_Crouching;
Weapon weapon;
protected AnimatorOverrideController animatorOverrideController;
```

12) m_Rigidbody, Rigidbody компонент данного GameObject-а;

13) m_Animator, Animator компонент данного GameObject-а;

14) m_IsGrounded, поле, отвечающее за то, есть ли земля под этим персонажем;

15) weapon, экипированное оружие;

16) animatorOverrideController, необходимый для замены анимаций во время игры, например при экипировке разного оружия;

```
int jumpTimer = 0;//frames
ссылок: 4
bool IsActing { get; set; } = false;
[SerializeField]
public Inventory inventory;
Body m_body;
float landingAnimationLength;
```

17) jumpTimer, необходимый для задержки проверки наличия земли под персонажем при прыжке;

18) IsActing, если True, то персонаж совершает какое либо действие не являющееся передвижением;

19) inventory, инвентарь персонажа;

20) m_Body, Body компонент данного GameObject-а. И методы:

```
public void ChangeWeapon(Weapon weapon)
{
    if (weapon != null)
    {
        weapon.Equip(rightHand);
        weapon.owner = gameObject;
        this.weapon = weapon;
    }
    else
    {
        if (this.weapon != null)
            this.weapon.Pick(transform);
        this.weapon = defautWeapon;
        this.weapon.Equip(rightHand);
    }


}
```

ChangeWeapon(Weapon weapon), меняет экипированное оружие weapon на полученное значение, а при полученном значении равном null меняет экипированное оружие на оружие по умолчанию, defaultWeapon.

```
void DropWeapon()
{
    if (defautWeapon != null)
        Destroy(defautWeapon.gameObject);
    if (weapon != null)
        weapon.Throw();
}
```

DropWeapon(), позволяет бросить экипированное оружие на землю.

```
public void AddItemToInventory(Item item)
{
    Debug.Log("Character.AddItemToInventory(), Inventory:");
    Debug.Log(inventory);
    inventory.InsertItem(item);

}
```

AddItemToInventory(Item item), добавляет предмет в инвентарь.

```
public void OnActionAnimationEnd()
{
    IsActing = false;
    //Debug.Log("Character.OnActionAnimationEnd()");
}
```

OnAnimationEnd(), срабатывает при завершении анимации в состоянии Action машины состояний аниматора и сообщает об окончании анимации.

```
public void Move(Vector3 move, bool crouch, bool jump)
{
    //Debug.Log("isActing: " + isActing);

    if (IsActing)
    {
        if (move.magnitude > 1f) move.Normalize();
        move = transform.InverseTransformDirection(move);
        move = Vector3.ProjectOnPlane(move, m_GroundNormal);
        m_TurnAmount = Mathf.Atan2(move.x, move.z);
        m_ForwardAmount = move.z;
        ApplyExtraTurnRotation();
    }
    else
    {
        // convert the world relative moveInput vector into a local-relative
        // turn amount and forward amount required to head in the desired
        // direction.
        if (move.magnitude > 1f) move.Normalize();
        move = transform.InverseTransformDirection(move);
        move = Vector3.ProjectOnPlane(move, m_GroundNormal);
        m_TurnAmount = Mathf.Atan2(move.x, move.z);
        m_ForwardAmount = move.z;

        ApplyExtraTurnRotation();

        // control and velocity handling is different when grounded and airborne:
        if (m_IsGrounded)
        {
            if (jump && !crouch && m_Animator.GetCurrentAnimatorStateInfo(0).IsName("Grounded"))
            {
                Jump();
            }
        }
        else
        {

        }

        //ScaleCapsuleForCrouching(crouch);
        //PreventStandingInLowHeadroom();

        // send input and other state parameters to the animator

    }
    UpdateAnimator(move);
}
```

Move(Vector3 move, bool crouch, bool jump), передвигает персонажа в направлении вектора с помощью аниматора, за счет чего движение выглядит плавным. Атрибуты crouch и jump отвечают за полуприсяд и прыжок соответственно.

```
public void StartAttack()
{
    if (weapon != null)
        weapon.StartAtack();
}
```

StartAttack(), callback функция, срабатывающая при срабатывании соответствующего события в анимации. Запускает начало атаки в экипированном оружии.

```
public void EndAttack()
{
    if (weapon != null)
        weapon.EndAttack();
}
ссылка: 1
```

EndAttack(), callback функция, срабатывающая при срабатывании соответствующего события в анимации. Останавливает атаку в экипированном оружии.

```
void Jump()
{
    //Debug.Log("Jump()");
    jumpTimer = 10;
    m_Rigidbody.velocity = new Vector3(m_Rigidbody.velocity.x, m_JumpPower, m_Rigidbody.velocity.z);
    m_IsGrounded = false;
    m_Animator.applyRootMotion = false;
}
```

Jump(), при срабатывании изменяет абсолютную скорость персонажа, подбрасывая его вверх.

```
void HandleAirborneMovement()
{
    if (!m_IsGrounded)
    {
        //Debug.Log("HandleAirborneMovement();");
        m_Animator.applyRootMotion = false;
        if (jumpTimer < 0)
        {
            if (m_Rigidbody.velocity.y < 0)//falling down
            {
                //Debug.Log("Physics.gravity.y: " + Physics.gravity.y);
                //Debug.Log("m_Rigidbody.velocity.y: " + m_Rigidbody.velocity.y);
                float landingDistance = (m_Rigidbody.velocity.y + (Physics.gravity.y * landingAnimationLength)) * landingAnimationLength * (-0.05f);
                //Debug.Log("landingDistance: " + landingDistance);
                if (Physics.Raycast(transform.position + (Vector3.up * 0.1f), Vector3.down, out RaycastHit hit, landingDistance))
                {
                    //landing
                    m_Animator.Play("Landing");
                }
                else
                {
                    //falling
                    m_Animator.Play("Falling");
                }
            }
            else//going up
            {
                m_Animator.Play("Jump");
            }


        }
        else
        {
            jumpTimer--;
        }
    }
}
```

HandleAirborneMovement(), проверяет состояние персонажа в полете, если земля под персонажем близко, то проигрывается анимация приземления, в ином случает играется анимация свободного падения.

```
void ScaleCapsuleForCrouching(bool crouch)
{
    if (m_IsGrounded && crouch)
    {
        if (m_Crouching) return;
        m_Capsule.height = m_Capsule.height / 2f;
        m_Capsule.center = m_Capsule.center / 2f;
        m_Crouching = true;
    }
    else
    {
        Ray crouchRay = new Ray(m_Rigidbody.position + Vector3.up * m_Capsule.radius * k_Half, Vector3.up);
        float crouchRayLength = m_CapsuleHeight - m_Capsule.radius * k_Half;
        if (Physics.SphereCast(crouchRay, m_Capsule.radius * k_Half, crouchRayLength, Physics.AllLayers, QueryTriggerInteraction.Ignore))
        {
            m_Crouching = true;
            return;
        }
        m_Capsule.height = m_CapsuleHeight;
        m_Capsule.center = m_CapsuleCenter;
        m_Crouching = false;
    }
}
```

ScaleCapsuleForCrouching(bool crouch), проверяет наличие препятствий над персонажем, и не позволяет тому встать с полуприседа, если такие имеются.

```
void UpdateAnimator(Vector3 move)
{
  // Debug.Log("m_GroundCheckDistance: " + m_GroundCheckDistance);
    // update the animator parameters
    m_Animator.SetFloat("Forward", m_ForwardAmount, 0.1f, Time.deltaTime);
    m_Animator.SetFloat("Turn", m_TurnAmount, 0.1f, Time.deltaTime);
    m_Animator.SetBool("Crouch", m_Crouching);
    m_Animator.SetBool("OnGround", m_IsGrounded);
    if (!m_IsGrounded)
    {
        m_Animator.SetFloat("Jump", m_Rigidbody.velocity.y);
    }

    // calculate which leg is behind, so as to leave that leg trailing in the jump animation
    // (This code is reliant on the specific run cycle offset in our animations,
    // and assumes one leg passes the other at the normalized clip times of 0.0 and 0.5)
    float runCycle =
        Mathf.Repeat(
            m_Animator.GetCurrentAnimatorStateInfo(0).normalizedTime + m_RunCycleLegOffset, 1);
    float jumpLeg = (runCycle < k_Half ? 1 : -1) * m_ForwardAmount;
    if (m_IsGrounded)
    {
        m_Animator.SetFloat("JumpLeg", jumpLeg);
    }

    // the anim speed multiplier allows the overall speed of walking/running to be tweaked in the inspector,
    // which affects the movement speed because of the root motion.
    if (m_IsGrounded && move.magnitude > 0)
    {
        m_Animator.speed = m_AnimSpeedMultiplier;
    }
    else
    {
        // don't use that while airborne
        m_Animator.speed = 1;
    }
}
```

UpdateAnimator (Vector3 move), обновляет параметры аниматора на соответствующие параметры данного персонажа.

```
void ApplyExtraTurnRotation()
{
    // help the character turn faster (this is in addition to root rotation in the animation)
    float turnSpeed = Mathf.Lerp(m_StationaryTurnSpeed, m_MovingTurnSpeed, m_ForwardAmount);
    transform.Rotate(0, m_TurnAmount * turnSpeed * Time.deltaTime, 0);
}
```

ApplyExtraTurnRotation(), добавляет дополнительное вращение персонажу, помимо вращения, полученного от аниматора.

```
    void CheckGroundStatus()
    {
        if (jumpTimer <= 0)
        {
            //Debug.Log("CheckGroundStatus");

#if UNITY_EDITOR
            // helper to visualise the ground check ray in the scene view
            Debug.DrawLine(transform.position + (Vector3.up * 0.1f), transform.position + (Vector3.up * 0.1f) + (Vector3.down * m_GroundCheckDistance));
#endif
            // 0.1f is a small offset to start the ray from inside the character
            // it is also good to note that the transform position in the sample assets is at the base of the character
            if (Physics.Raycast(transform.position + (Vector3.up * 0.1f), Vector3.down, out RaycastHit hitInfo, m_GroundCheckDistance))
            {
                m_GroundNormal = hitInfo.normal;
                m_IsGrounded = true;
                m_Animator.applyRootMotion = true;
            }
            else
            {
                m_IsGrounded = false;
                m_GroundNormal = Vector3.up;
                m_Animator.applyRootMotion = false;
            }
        }
    }
```

CheckGroundStatus(), запускает луч вниз от персонажа, и если он пересекается с препятствием, то считается, что персонаж находится на земле.

```
public void Action(AnimationType animationType,Vector3 target)
{
    if (!IsActing)
    {
        Vector3 copy = transform.eulerAngles;
        transform.LookAt(target);
        copy.y = transform.eulerAngles.y;
        transform.eulerAngles = copy;

        if (animationType == AnimationType.EasyAttack )
        {
            var a = weapon.comboSet.NextEasyAttack();
            weapon.modifier = a.damageModifier * multiplier;
            m_Animator.SetFloat("AttackSpeed", a.speedModifier);
            animatorOverrideController[AnimationType.EasyAttack.ToString()] = a.animation;
            m_Animator.Update(0.0f);
        }else if(animationType == AnimationType.HardAttack)
        {
            var a = weapon.comboSet.NextHardAttack();
            weapon.modifier = a.damageModifier * multiplier;
            m_Animator.SetFloat("AttackSpeed", a.speedModifier);
            animatorOverrideController[AnimationType.HardAttack.ToString()] = a.animation;
            m_Animator.Update(0.0f);
        }
        m_Animator.Play(animationType.ToString());
        IsActing = true;
    }



}
```

Action(AnimationType animationType,Vector3 target), запускает соответствующую типу анимации действие, направленное в сторону вектора target.

Weapon компонент крепится к GameObject-у оружия и отвечает за основную логику оружия.

Weapon содержит следующие поля:

```
public float damage;
public GameObject owner;
public AttackComboSet comboSet;
public float pushForce = 30;
public float modifier;
```

1) damage, базовый урон от оружия;

2) owner, GameObject владельца оружия, нужен для предотвращения урона по владельцу оружия;

3) comboSet, комбинации атак, содержащие анимации и множители урона анимаций;

4) pushForce, сила отталкивания оружия;

```
bool isAtacking = false;

int attackTimer;
List<Body> hitBodies =  new List<Body>();
```

5) isAtacking, если true, то оружие атакует и может нанести урон;

6) hitBodies, список Body объектов, в который добавляются Body при атаке, чтобы предотвратить повторное нанесение урона.

И методы:

```
public void StartAtack()
{
    attackTimer = 3;
    isAtacking = true;
}
```

StartAtack(), при срабатывании позволяет оружию наносить урон.

```
public void EndAttack()
{
    hitBodies.Clear();
    Debug.Log("Weapon.EndAttack()");
    isAtacking = false;
}
```

EndAttack(), при срабатывании запрещает оружию наносить урон.

```
public void DealDamage(Body body,Vector3 hitPoint)
{

    if (isAtacking&& owner.GetComponent<Body>()!= body && attackTimer==0&&!hitBodies.Contains(body))
    {
        hitBodies.Add(body);
        body.Push(((hitPoint ).normalized * pushForce*modifier));
        body.TakeDamage(damage * modifier);// * multiplyer);
        Debug.Log("Final Damage: " + damage );
    }
}
```

DealDamage(Body body,Vector3 hitPoint) callback метод, срабатывающий при соприкосновении коллайдера оружия и вражеского GameObject-а с

компонентом Body. Вызывает метод TakeDamage у Body объекта, тем самым нанося урон.

Для управления Character используются два скрипта, один для пользователя – UserControl, другой для искусственного интеллекта AIControl.

UserControl отслеживает нажатие клавиш пользователем и позволяет пользователю управлять персонажем, его действиями и передвижениями через Character.

UserControl содержит поля:

```
private Character m_Character; // A reference to th
private Transform m_Cam;                  // A refe
private Vector3 m_CamForward;             // The cu
private Vector3 m_Move;
private bool m_Jump;                      // the wo
//private Body m_Body;
public static bool isControllable = true;

Vector3 prevPosition;
[SerializeField]
float interactionRadius;
InteractiveObject selectedInteractiveObj;
private CanvasController canvas;
```

1) m_Character, Character компонент данного GameObject-a;
2) m_Cam, главная камера на сцене;
3) m_CamForward, вектор направления камеры;
4) m_Move, вектор направления движения персонажа;
5) m_Jump, если true, то нажата кнопка прыжка;
6) isControllable, если true, то пользователь может контролировать персонажа;
7) interactionRadius, радиус каста сферического коллайдера, для нахождения интерактивного объекта;
8) selectedInteractiveObj, выделенный интерактивный объект;
9) canvas, Canvas объект, содержащий пользовательский интерфейс.

И методы:

```
public void PlayLeftClickAnim()
{
    m_Character.Action(AnimationType.EasyAttack, transform.position + transform.forward);

}
```

PlayLeftClickAnim(), метод, вызываемый при нажатии левой кнопки пользователем, по умолчанию легкая атака.

```
public void PlayRightClickAnim()
{
    m_Character.Action(AnimationType.HardAttack, transform.position + transform.forward);


}
```

PlayRightClickAnim(), метод, вызываемый при нажатии правой кнопки пользователем, по умолчанию сильная атака.

```
private void Update()
{
    UpdateSphereCast();
    if (Input.GetKeyDown(KeyCode.U))
    {
        canvas.weaponSlider.Next();
    }

    if (isControllable)
    {
        if (Input.GetKeyDown(KeyCode.I))
        {
            canvas.inventoryItemPanelContainer.gameObject.SetActive(true);
            isControllable = false;
        }
        if (Input.GetKeyDown(KeyCode.F))
        {
            if (selectedInteractiveObj != null)
                selectedInteractiveObj.Interact(m_Character);
        }
    }
    else
    {
        if (Input.GetKeyDown(KeyCode.I))
        {
            canvas.inventoryItemPanelContainer.gameObject.SetActive(false);

            isControllable = true;
        }
        if (Input.GetKeyDown(KeyCode.K))
        {
            canvas.weaponSlider.Expand();
        }
    }

}
```

Update(), метод, вызываемый каждый кадр, проверяет нажатия клавиш и вызывает соответствующие клавишам методы.

```
void UpdateSphereCast()
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    Debug.DrawRay(ray.origin,ray.direction);
    RaycastHit[] raycastHits =  Physics.SphereCastAll(ray,interactionRadius);
    foreach (var hit in raycastHits)
    {
        var hint = hit.collider.gameObject.GetComponent<IShowHint>();
        var interactive = hit.collider.gameObject.GetComponent<InteractiveObject>();
        if (hint != null)
        {
            //Debug.Log("UserControl.ShowHint");
            hint.ShowHint();
        }
        if (interactive != null)
        {
            selectedInteractiveObj = interactive;
        }
    }
}
```

UpdateSphereCast(), метод, проверяющий наличие в месте указателя мыши интерактивного объекта.

```
private void FixedUpdate()
{

    prevPosition = transform.position;
    if (!m_Jump)
    {
        m_Jump = CrossPlatformInputManager.GetButtonDown("Jump");
    }
    if (isControllable)
    {

        // read inputs
        if (Input.GetMouseButton(0))
        {
            PlayLeftClickAnim();
        }
        if (Input.GetMouseButton(1))
        {
            PlayRightClickAnim();
        }

        float h = CrossPlatformInputManager.GetAxis("Horizontal");
        float v = CrossPlatformInputManager.GetAxis("Vertical");
        bool crouch = Input.GetKey(KeyCode.C);

        // calculate move direction to pass to character
        if (m_Cam != null)
        {
            // calculate camera relative direction to move:
            m_CamForward = Vector3.Scale(m_Cam.forward, new Vector3(1, 0, 1)).normalized;
            m_Move = v * m_CamForward + h * m_Cam.right;
        }
        else
        {
            // we use world-relative directions in the case of no main camera
            m_Move = v * Vector3.forward + h * Vector3.right;
        }
!MOBILE_INPUT
        // walk speed multiplier
        if (Input.GetKey(KeyCode.LeftShift)) m_Move *= 0.5f;
dif

        // pass all parameters to the character control script
        m_Character.Move(m_Move, crouch, m_Jump);
        m_Jump = false;
    }
    else
    {
        m_Character.Move(m_Move*0, false, m_Jump);
    }

}
```

FixedUpdate(), метод вызываемый каждую фиксированную еденицу времени, не зависящий от частоты кадров, отвечает за передвижение.

При нажатии пользователем клавиш передвижения активируется следующая цепочка событий: клавиши регистрируются в методе FixedUpdate() и их вектор относительно камеры передается в m_Character.Move(), поворачивая персонажа и проигрывая анимацию ходьбы в аниматоре.

При нажатии пользователем кнопок атак происходит более сложная последовательность методов: Update() регистрирует нажатие клавиши атаки и вызывает метод Action(AnimationType animationType,Vector3 target) с параметром animationType = AnimationType. EasyAttack и направлением атаки. В m_Character анимация атаки заменяется на следующую анимацию из AttackComboSet и вызывается анимация атаки.

При анимации атаки срабатывает событие StartAttack() в Character и вызывается метод StartAttack в оружии m_Character.weapon. При соприкосновении коллайдера дочернего к m_Character.weapon

WeaponBladeCollidersHandler –а с GameObject-ом с компонентом Body срабатывает метод m_Character.weapon.DealDamage(), наносящий урон полученному вражескому Body (диаграмма 1).

AIControl отвечает за поведение персонажей под контролем искусственного интеллекта.

AIControl имеет поля:

```
private Character m_Character; // A reference
private Transform m_Cam;                 //
private Vector3 m_CamForward;            //
private Vector3 m_Move;
private bool m_Jump;                     //
//private Body m_Body;
public static bool isControllable = true;

Vector3 prevPosition;
[SerializeField]
float interactionRadius;
InteractiveObject selectedInteractiveObj;

State state = State.Attack;

Character curentTarget;


float attackDistance = 1.3f;
```

1) m_Character, компонент Character данного GameObject-а;

2) state, отвечает за текущее состояние ИИ, бывает трех типов: Attack, Idle,Run;

Общий вид State:

```
enum State
{
    Attack,
    Run,
    Idle
}
```

3) currentTarget, текущая вражеская цель, Character;

4) attackDistance, если расстояние до персонажа меньше этого значения, то применяется атака.

И методы:

```
private void FixedUpdate()
{
    if ((int)(Time.time * 10) % 5 == 0)
        CheckForEnemy();
    switch (state)
    {
        case State.Attack:
            if (curentTarget != null)
            {
                if (Vector3.Distance(transform.position, curentTarget.transform.position) < attackDistance)
                {
                    float r = UnityEngine.Random.Range(0f,1f);
                    if (r < 0.3f)
                    {
                        m_Character.Action(AnimationType.HardAttack, curentTarget.transform.position);

                    }
                    else if(r<0.9f)
                    {
                        m_Character.Action(AnimationType.EasyAttack, curentTarget.transform.position);
                    }
                    else
                    {
                        m_Character.Move((curentTarget.transform.position - transform.position).normalized, false, false);
                    }
                }
                else
                {
                    m_Character.Move((curentTarget.transform.position - transform.position).normalized, false, false);

                }
            }

            break;

        case State.Idle:
            break;
        case State.Run:
            break;
    }
}
```

FixedUpdate(), вызывается каждый фиксированный отрезок времени, в данном методе проверяется наличие врагов вокруг и текущее состояние, и в зависимости от текущего состояния даются соответствующие команды m_Character.

```
void CheckForEnemy()
{
    //Debug.Log("AIControl.CheckForEnemy()");
    List<Character> characters = GameDataOptimizer.GetValue<Character>();
    if (characters!=null&&characters.Count > 0)
    {
        //Debug.Log("characters.Count: " + characters.Count);
        Character closestCharacter = null;
        float closestDistance =float.MaxValue;
        foreach (var c in characters)
        {

            if (c != m_Character&&c.GetComponent<Body>().isDead==false)
            {


                float d = Vector3.Distance(transform.position, c.transform.position);
                if (d < closestDistance)
                {
                    closestDistance = d;
                    closestCharacter = c;
                }
            }


        }
        if (closestCharacter != m_Character)
        {
            curentTarget = closestCharacter;

        }

    }

}
```

CheckForEnemy(), метод, в котором проверяется наличие врагов поблизости.

В случае если state = State.Attack и если обнаружен враг, то персонаж передвигается к врагу до тех пор, пока не подойдет на дистанцию атаки, после чего начнет атаковать врага, пока тот снова не отойдет.

Для оптимизации поиска врагов был создан класс GameDataOptimizer. В нем хранятся ссылки на определенные типы классов. В начале игры все экземпляры определенных классов вносят в GameDataOptimizer.container ссылки на себя. В итоге во время игры можно получить легкий доступ ко всем экземплярам определенного класса.

3.2 Интерфейс

При старте программы пользователя встречает главное меню с возможностью продолжить с последнего сохранения, если таковое имеется, начать новую игру и выйти из игры.

Рисунок 3.1 – Главное меню игры

Игра автоматически сохраняется каждую минуту. При сохранении игры, программа использует физическую память, таким образом, если выйти и зайти в игру снова, сохранения не пропадут.

При начале новой игры сохранения обнуляются.

Перемещение в игре осуществляется клавишами WASD для передвижения и кнопкой пробел для прыжка.

В левой нижней части экрана находится панель выбранного оружия. Она позволяет хранить два оружия одновременно и переключаться между ними. Его можно сменить на оружие в инвентаре и наоборот, поместить оружие в инвентарь. Клавишей I открывается инвентарь. Инвентарь позволяет хранить несколько предметов одновременно. Клавишей U можно сменить оружие в панели оружия на следующее.

При экипировке оружия меняются анимации атаки персонажа на соответствующие данному оружию.

Рисунок 2.2 – Демонстрация инвентаря

Предметы лежащие на земле можно подобрать клавишей F. Подобранные предметы помещаются в инвентарь.

Рисунок 2.3 – Демонстрация отображения подсказок интерактивных объектов

При наведении курсора на интерактивный объект высвечивается текстовая подсказка.

При нажатии на левую кнопку мыши активируется анимация легкой атаки, при нажатии на правую – тяжелой атаки. При этом персонаж проигрывает атакующую анимацию и если оружие персонажа задевает объекты, которые могут получать урон, им наносится урон.

При нанесении урона высвечивается подсказка, отображающая численное значение урона.

Рисунок 2.4 – Демонстрация отображения урона в виде цифр

4 Безопасность жизнедеятельности

4.1 Введение

Дипломный проект посвящен созданию 3D-игры для персональных компьютеров. Использоваться данный программный продукт будет на персональных компьютерах с мониторами различных характеристик. Особенности данных мониторов должны соответствовать нормам отображения картинки для более комфортного использования программного продукта. Для этого необходимо провести оптимизацию средств и систем отображения информации. Рассмотрим физиологические особенности, а также, для сравнения, технические особенности мониторов нескольких разных марок с разными типами матриц, разным временем отклика, разными углами обзора и разной частотой обновления кадров.

4.2 Роль анализаторов в операторской деятельности

Физиологической основой формирования информационной модели являются исследование факторов, посредством которых человек осуществляет исследование раздражений.

Информация, поступающая через исследуемые показатели, называются сенсорной (чувственной), а процесс ее приема – сенсорной деятельностью или сенсорным восприятием.

Любой исследуемый фактор содержит:

- рецептор;
- проводящие нервные пути;
- центр в коре больших полушарий головного мозга.

Основная функция рецептора – превращение действующего раздражителя в нервный процесс. Вход рецептора имеет приспособление к приему сигналов определенной модальности (вида) – свет, звук, вибрация и т.п. Его выход посылает сигнал, по своей природе единый для любого входа нервной системы. Это дает возможность рассматривать рецепторы в качестве устройства кодирования информации.

Проводящие нервные пути осуществляют передачу информации в кору головного мозга.

Информация подвергается определенной переработке и снова возвращается в рецепторы.

Наибольшее значение для деятельности оператора имеет *орган зрения* – 90% всей информации поступает к оператору именно через него. За ним по значимости следует орган слуха, и на третьем месте – *тактильный (или*

осязательный). Участие других органов чувств в деятельности оператора невелико.

Основными характеристиками любого органа чувств являются пороги, которые могут быть:

- абсолютными (верхний и нижний);
- дифференциальными;
- оперативными.

Минимальная величина раздражителя, вызывающая едва заметное ощущение, носит название нижнего абсолютного порога чувствительности. Сигналы, величина которых меньше нижнего абсолютного порога чувствительности, человеком не считываются.

Увеличение же сигнала выше верхнего абсолютного порога чувствительности вызывает у человека болевые ощущения. Интервал между нижним и верхним абсолютными порогами чувствительности носит название Охвата чувствительности анализатора.

Дифференциальный порог – наименьшее отличие между двумя раздражителями, либо между двумя состояниями одного раздражителя, вызывающее ели-ели заметное различие ощущений.

Оперативный порог определяется той минимальной величиной различия между сигналами, при которой точность и скорость различения достигает своего наибольшего значения.

Исходя из всего вышеуказанного, можно вывести общие условия к сигналам раздражителям:

Интенсивность сигнала должна соответствовать средним значениям Охвата чувствительности анализатора, которая создает оптимальные условия для приема и переработки информации.

Следует обеспечить различие между сигналами, превышающее дифференциальный порог чувствительности.

Перепады между сигналами не должны превышать оперативный порог чувствительности, иначе возникает утомление.

Наиболее важные и значимые сигналы следует располагать в тех зонах, которые соответствуют участкам рецепторной поверхности с наибольшей чувствительностью.

При конструировании индикаторных устройств следует правильно выбрать вид (модальность) сигнала и модальность анализаторов.

При конструировании средств отображения информации (СОИ) кроме изучения возможностей конкретных органов чувств следует учитывать межанализаторные связи. Это следует делать при предъявлении оператору полимодальных сигналов, т.е. сигналов различной модальности.

Полимодальные сигналы используют в следующих ситуациях:

дублирование – сигнал одновременно поступает на несколько органов чувств для повышения надежности передачи информации (сигналы тревоги);

распределение поступающей информации между органами чувств для исключения их перегрузки, требует учета пропускных способностей органов

чувств; переключение активности органов чувств – борьба с монотонностью операторского труда.

4.3 Средства отображения информации

Человек – оператор получает информацию с помощью *средств отображения информации (СОИ),* где в закодированном виде представлен ход процесса или состояние объекта наблюдения в форме, удобной для восприятия человеком.

Обычно средства отображения информации используют для одной или нескольких целей:

- считывания количественных и качественных показателей;
- контрольного считывания показателей;
- установки регулируемого параметра.

Любые СОИ должны удовлетворять следующим инженерно-психологическим условиям:

1) обеспечивать рабочего требуемой и достаточной информацией для оценивания ситуации и возможности принятия правильного решения, а также осуществлением контроля за его исполнением;
2) информация должна быть подана в тот момент, когда в ней возникает нужность;
3) форма предоставления информации должна учитывать психофизиологические возможности оператора по восприятию, специфике его деятельности и условиям работы;
4) получаемая информация должна правильно отражать положение и состояние объекта управления, предоставляться с определенным запасом времени, достаточным для ее обработки;
5) давать оператору дополнительные сведения по запросу, а также обеспечивать надежность восприятие аварийных сигналов;
6) поток информации должен соответствовать пропускной способности оператора.

4.4 Оптимизация средств и систем отображения информации

При проектировании и эксплуатации средств отображения рассматриваются три группы факторов:

- размещение средств отображения на рабочем месте и в оперативных залах;
- оптимальные размеры знаков и их элементов в разных системах отображения;
- оптимальная компоновка знаков на средствах отображения.

При ширине экрана меньше 10 м отношение ширины экрана к его высоте берется равным 1,3:1. Оптимальный угол наблюдения составляет

±15° к нормали экрана. При рассматривании изображения сбоку допустимый угол обзора составляет 45° к нормали экрана.

4.5 Яркостная характеристика зрительной информации

В оценку оптимальности яркостного режима включается нормирование уровня яркости и ее перепадов в поле зрения наблюдателя для достижения заданных показателей эффективности обработки зрительной информации. Для оценки качества изображения на индикационных устройствах нормируются значения контраста, контрастности или интервала яркостей, необходимого для передачи заданного числа градаций яркости и обеспечения четкости изображения, а также уровень и интервал яркостей для правильной передачи в изображении светлотных характеристик отображаемых объектов. Для яркости 300–200 кд/м^2 острота зрения составляет 90 % по сравнению с наибольшими ее значениями. Резкое падение остроты зрения наблюдается при выходе из диапазона яркости дневного зрения, то есть < 10 кд/м^2.

При выборе яркости следует учитывать знак контраста изображения. Острота зрения растет для обратного контраста с увеличением яркости до 30–31 кд/м^2, при дальнейшем ее росте острота зрения падает вследствие иррадиации.

4.6 Временная характеристика зрительной информации

Величину частоты мельканий необходимо учитывать для создания качественного изображения на различных устройствах отображения. Мелькание утомляет зрение и отрицательно влияет на качество работы оператора. Критическая частота мелькания зависит от частоты и относительной длительности светлой фазы. С увеличением длительности темного периода (скважность проблесков) с 0,35 до 0,5 при яркости 2,5–250 кд/м^2 КЧМ увеличивается на 3 ± 6 %.

Наиболее важные эргономические требования, предъявляемые к средствам отображения информации можно оценить по техническим характеристикам мониторов. В таблице 4.1 представлен сравнительный анализ 24' мониторов различных фирм-производителей.

Таблица 4.1 - Сравнительный анализ мониторов по техническим характеристикам на соответствие эргономическим требованиям

Параметр (эргономический показатель)	Технические характеристики мониторов	Марки мониторов		
		Samsung S24D300H	AOC 24B1XHS	Philips 246E9QDSB/00(01)

Отношение ширины к высоте	Габаритные параметры (Размер видимой области экрана)	569 x 342 мм (531x299 мм)	540.8x 419.9 мм (526.88x296.37 мм)	540.8x 419.9 мм (526.88x296.37 мм)

Продолжение таблицы 4.1

Одномоментная переработка информации	Время отклика пикселя	2 мс	7 мс	5 мс
Контрастность изображения (пределы регулировки)	Динамическая контрастность	Mega Contrast Ratio	20M:1	AMD FreeSync
Интервал яркости	Тип подсветки монитора	LED	LED	LED
Число градаций яркости	Величина контраста и яркости	1000:1 250 Кд/м²	1000:1 250 Кд/м²	1000:1 250 Кд/м²
Обеспечение четкости изображения	Угол обзора по вертикали и горизонтали, разрешение монитора	Угол обзора по вертикали 160° 1920x1080	Угол обзора по вертикали 178° 1920x1080	Угол обзора по вертикали 178° 1920x1080
Частота обновления кадров	Максимальная частота обновления экрана	60 Гц	60 Гц	76 Гц
Соотношение яркости полезного изображения и яркости помех	Тип покрытия матрицы	Матовое	Матовое	Матовое
Критическая частота мелькания	Тип матрицы	TN	IPS	IPS

При эргономическом обосновании выбора монитора следует особое внимание уделять следующим показателям:

1) Тип матрицы, влияющий на качество (контрастность, яркость, цветопередача, углы обзора и т.п.);

TN. Недостаток - неточная цветопередача, низкая контрастность и относительно небольшие углы обзора (особенно вертикальные)

IPS. отличаются самыми лучшими среди всех типов матриц углами обзора и цветопередачей. Контрастность и время отклика сильно зависят от конкретной реализации этой технологии. Можно отметить, что IPS-матрицы в среднем имеют большее время отклика, нежели TN, и худшую контрастность, *VA-матриц, а кроме того отсутствие эффекта «мерцания».

VA. Главным их достоинством является высокая контрастность в сочетании с хорошей цветопередачей, но, в отличие от IPS, есть отрицательная особенность, выражающаяся в пропадании деталей в тенях при перпендикулярном взгляде и зависимость цветового баланса изображения от угла зрения.

2) Использование технологий улучшения изображения;

Mega Contrast Ratio улучшает динамическую контрастность монитора и, как следствие, картинка получается более сочной

AMD FreeSync - технология, которая улучшает качество изображения после масштабирования (увеличения), улучшает размытость, а затем за счет технологии контроля резкости получается четкая, но не слишком резкая реалистичная картинка.

3) Время отклика отвечает за плавность воспроизведения картинки в динамических сценах, чем меньше значение этого параметра, тем лучше качество изображения;

4) Угол обзора по вертикали и горизонтали проявляется в ухудшении изображения при взгляде на экран под углом: падает контрастность и снижается точность передачи цветов. Хорошее значение углов обзора, позволяющее пользоваться монитором без особых ограничений, – 165-175 градусов по вертикали и столько же по горизонтали;

5) Разрешение монитора- отображение картинки с меньшим числом пикселей вызывает ее интерполяцию и снижает уровень четкости. Обычно 1920x1080 или 2560x1440, что обозначает количество точек по вертикали к количеству точек по горизонтали. Чем выше разрешение, тем больше информации может отображаться на мониторе;

6) Яркость характеризует интенсивность свечения экрана и измеряется в канделах на квадратный метр (кд/м^2). Если характеристика яркости недостаточно высока, то с таким монитором будет некомфортно работать;

7) Контрастность определяется как отношение яркости белого цвета на экране к яркости черного. Высокая контрастность делает изображение более «осязаемым» и «живым». Минимальный рекомендуемый уровень контрастности для монитора – 500:1.

При сравнении по техническим характеристикам мониторов марок Samsung, AOC и Philips наиболее эргономичным является Philips 246E9QDSB/00(01), несмотря на большое значение времени отклика

используется тип матрицы исключающий эффект «мерцания», что очень важно при длительном наблюдении за монитором, а также используются технологии улучшения качества изображения AMD FreeSync - технология, которая улучшает качество изображения после масштабирования (увеличения), улучшает размытость, а затем за счет технологии контроля резкости получается четкая картинка.

5 Экономическая часть

5.1 Трудоемкость разработки приложения

Игра это программный продукт, во многом похожий на фильмы, музыку и цифровые книги. Сам продукт представляет собой информацию, легко передающуюся между пользователями. в таких случаях правильно будет выдавать каждому пользователю цифровой ключ за право пользования информацией. Основная же задача разработчика сводится к созданию конкурентоспособного товара, или товара, занимающего определенную нишу. Доход в таком случае зависит от прогнозируемого количества покупателей и цены.

Для определения трудоемкости составляется перечень всех этапов разработки.

Таблица 5.1 – Перечень всех этапов разработки.

Название этапа	Описание этапа	Трудоемкость разработки	
		чел.× час.	час×день
Анализ требований	На данном этапе даются основные векторы направления разработки приложения	1×56	8×7
Анализ рынка	Сравниваются успешность аналогичных приложений с учетом их сильных и слабых сторон	1×16	8×2
Проектировка	Составление общего плана проекта	1×32	8×4
Разработка программной части	Разработка приложения	1×128	8×16
Создание моделей	Создание 3D моделей	1×88	8×11
Тестирование	Тестирование приложения	1×16	8×2

Внедрение и поддержка	Запуск приложения на платформу по продаже игр.	1×8	8×1
Итоговая трудоемкость выполнения работы:		344чел.× ч.	43 дня

5.2 Расчет затрат на разработку информационной системы

Нужно рассчитать следующие затраты для составления:
- материальные затраты;
- затраты на оплату труда;
- социальный налог;
- амортизация основных фондов;
- прочие затраты.

5.3 Затраты на оборудование и программное обеспечение

Таблица 5.2 – Затраты на оборудование и программное обеспечение.

Наименование материального ресурса	Единица измерения	Количество израсходованного материала	Цена за единицу, тг	Сумма, тг
Ноутбук Acer	Шт	1	330000	330000
Компьютерная мышь A4Tech Bloody	Шт	1	12000	12000
ОС Windows 10 Pro (ключ активации)	Шт	1	2 000	2 000
Программа Unity3D Enterprise	Шт	1	Бесплатно	0
Microsoft Visual Studio 2019 Enterprise	Шт	1	Бесплатно	0
ИТОГО затраты:				344000

5.4 Затраты на электроэнергию

Таблица 5.3 - Затраты на электроэнергию

Наименование оборудования	Паспортная мощность, кВт	Коэффициент использования мощности	Время работы оборудования для разработки	Цена электроэнергии, тг/кВт*ч	Сумма, тг

			ПП, ч		
Ноутбук Acer	0,135	0,9	344	17,79	743.5
ИТОГО затраты на электроэнергию					743.5

Рисунок 5.1 – Данные блока питания ноутбука

Рассчитывается мощность:

19(В) × 7.1 (А) = 135 Вт = 0, 135 кВт

Общая сумма затрат на электроэнергию ($З_Э$) рассчитывается по формуле:

$$Зэ = \sum Mi * Ki * Ti * Ц \qquad (5.1)$$

где M_i - паспортная мощность i-го электрооборудования, кВт;

K_i - коэффициент использования мощности i-го электрооборудования (принимается K_i=0.7¸0.9);

T_i - время работы i-го оборудования за весь период разработки ПП ч;

Ц - цена электроэнергии, тг/кВт×ч;

i - вид электрооборудования;

n - количество электрооборудования.

$З_э$ = 0,135 × 0,9 × 344 × 17, 79 = 743.5 тенге

5.5 Затраты на оплату труда

Затраты на оплату труда рассчитываются с помощью таких показателей, как общая трудоемкость, плановые сроки разработки и плановая численность работников.

Заработная плата взята из средней заработной платы по данной профессии. Из заработной платы (при 8 часовом рабочем дне). рассчитывается часовая ставка и путем перемножения часовой ставки и трудоемкости находим итоговую сумму затрат на оплату труда.

Таблица 5.4 – Сведения по работникам, задействованным в проекте.

Специалист - Исполнитель	Количество, человек	**Заработная плата в месяц, тенге**
Разработчик на Unity3D	1	300000
3D дизайнер	1	180000
Итого		480000

Часовая тарифная ставка находится делением месячной тарифной ставки на установленную при 40-часовой недельной норме рабочего времени расчетную среднемесячную норму рабочего времени в часах ($Ф_Р$):

$$Т_Ч = \frac{Т_М}{Ф_Р} = \frac{300000}{8 \cdot 21} = 1785 \text{ тг/ч} \quad (5.2)$$

$$Т_Ч = \frac{Т_М}{Ф_Р} = \frac{180000}{8 \cdot 21} = 1071 \text{ тг/ч} ,$$

где ТЧ - часовая тарифная ставка (тыс. тенге);

$ТМ$ - месячная тарифная ставка (тыс. тенге).

Таблица 5.5 - Затраты на оплату труда

Категория работника	Трудоемкость разработки ПП, чел.×ч	Часовая ставка, тг/ч	Сумма, тг
Разработчик на Unity3D	128	1785	228480
Backend-разработчик	88	1071	94248
ИТОГО затраты на оплату труда			322728

5.6 Социальный налог

Социальный налог для юридических лиц рассчитывается:

$$З_{СН} = (З_{ФОТ} - ОПВ - ВОСМС) \cdot 0{,}095 - СО, \qquad (5.3)$$

где Зфот – общий фонд оплаты труда разработчиков, тенге;

ОПВ – обязательный пенсионный взнос (10% от заработной платы);

ВОСМС – взнос на ОСМС (2% от заработной платы);

СО – социальные отчисления (3,5% от заработной платы с вычетом пенсионного взноса).

Обязательный пенсионный взнос вычисляется по формуле:

ОПВ = 322728× 10% = 32272тг.

Социальные отчисления вычисляются по формуле:

$$СО = (З_{ФОТ} – ОПВ) × 3{,}5\% \qquad (5.4)$$

Взнос на обязательное социальное медицинское страхование работодателем составит:

ВОСМС = 322728 × 2% = 6454.56тг.

Таким образом, социальные отчисления составят:

СО = (322728– 32272) × 3,5% = 10165.96тг.

Таким образом, социальный налог составит:

$З_{СН}$ = (322728– 32272– 6454) × 9,5% – 10165,9= 16814.29тг.

Итого, отчисления по социальному налогу:

$Зсз i = З_{СН} + СО + ВОСМС = 16814 + 10165 + 6454 = 33433$тенге

Таблица 5.6 - Смета затрат на разработку системы

Статьи затрат	Сумма, тг
Затраты на оборудование и ПО	344 000
Затраты на электричество	743
Затраты на оплату труда	322 728
Социальные налоги	33 433
ИТОГО	700 904

Рентабельность и прибыль по создаваемому ПО (Псi) определяются исходя из результатов анализа рыночных условий, переговоров с заказчиком (потребителем) и согласования с ним отпускной цены, включающей дополнительно налог на добавленную стоимость. В случае разработки ПО для использования внутри организации оценка программного продукта производится по действующим правилам и показателям внутреннего хозрасчета (по ценам, устанавливаемым для расчета за услуги между подразделениями). Прибыль рассчитывается по формуле:

$$П_{Oi} = C_{ni} \cdot \frac{У_{pni}}{100} \qquad (5.5)$$

где ПОi - прибыль от реализации ПО заказчику (тыс. тенге);

Урni - уровень рентабельности ПО (%), в дипломной работе брать 40-60%;

Cni - себестоимость ПО (тыс. тенге).

$$П_{Oi} = 700\ 904 \cdot\ 0.4 = 280\ 361{,}6 \text{ тенге}$$

Прогнозируемая цена ПО без налогов (Цпi):

$$Ц_{пi} = C_{ni} + П_{Oi} = 700\ 904 +\ 280\ 361{,}6 = 981\ 265{,}6 \text{ тенге} \quad (5.6)$$

Прогнозируемая отпускная цена *($Ц_{o1}$):*

$$Ц_{01} = Ц_{ni} + НДС \quad (5.7)$$

Ставка налога на добавленную стоимость НДС в РК на 2020 год составляет 12% от отпускной цены ПО.

$$Ц_{01} = 981\ 265{,}6\ + \frac{981\ 265{,}6}{100} = 1\ 099\ 018 \text{ тенге}$$

Организация-разработчик участвует в освоении ПО и несет соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком по договору. Затраты на освоение определяются по нормативу ($Н_O$=*10%)* от себестоимости ПО в расчете на 3 месяца и рассчитываются по формуле:

$$P_{Oi} =\ C_{ni}\ \cdot \frac{Н_O}{100} = 700\ 904\ \ \cdot 0{,}1 = 70\ 090{,}4 \text{ тенге} \quad (5.8)$$

Затраты на сопровождение ПО (P_{Ci}). Организация-разработчик осуществляет сопровождение ПО и несет соответствующие расходы, которые оплачиваются заказчиком в соответствии с договором и сметой на сопровождение. Затраты на сопровождение определяются по установленному нормативу (НС=*20%)* от себестоимости ПО (в расчете на год) и рассчитываются по формуле:

$$P_{Ci} =\ C_{ni}\ \cdot \frac{Н_C}{100} =\ 1\ 099\ 018\ \ \cdot 0{,}2 = 140\ 180{,}8 \text{ тенге} \quad (5.9)$$

Капиталовложения программного обеспечения с учетом затрат на освоение и сопровождение будет:

$$K\ = 1\ 099\ 018 + 70\ 090{,}4 + 140\ 180{,}8 = 1\ 309\ 290 \text{ тенге.}$$

5.7 Расчет поступления денежных средств

Для продажи игры “Roleplay game” будет использоваться онлайн магазин “itch.io”. Сайт устроен таким образом, что разработчик сам выбирает политику оплаты игры и устанавливает стоимость скачивания. Месячные поступления денежных средств рассчитаем по формуле:

$$Д_м = C_и * K_c, \quad (5.10)$$

где $C_и$ – стоимость скачивания игры, тг.;

K_c - количество скачиваний

Маркетинговым анализом установилась стоимость одного скачивания в 843.98 тенге. За месяц игру скачивали не менее 100 раз.

Следовательно, месячный доход составит:

$$Д_м = 843.98 * 100 = 84398 \ (тг)$$

Годовой доход от игры $Д_г$ рассчитаем по формуле :

$$Д_г = Д_м * 12$$

где $Д_м$- месячные поступления денежных средств, тг.

$$Д_г = 84398 * 12 = 1012680 \ (тг)$$

5.8 Расчет прибыли и срока окупаемости от реализации игры

Рассчитаем срок окупаемости затрат на разработку компьютерной игры «Roleplay game».

Расчетный коэффициент экономической эффективности капитальных вложений составляет:

$$Е_р = \frac{Э_{уг}}{К} \qquad (5.11)$$

где Ер - расчетный коэффициент экономической эффективности капитальных вложений;

Эуг — ожидаемая условно-годовая экономия, тенге;

К — капитальные вложения на создание системы, тенге.

$$Е_р = \frac{1\ 012\ 680}{1\ 309\ 290} = 0{,}77$$

Расчетный срок окупаемости капитальных вложений составляет:

$$Т_Р = \frac{1}{Е_Р} \qquad (5.12)$$

где Ер - коэффициент экономической эффективности капитальных вложений.

$$Т_Р = \frac{1}{1{,}16} = 1{,}29 \text{ года} \approx 15{,}5 \text{ месяцев}$$

Таблица 5.7 – Показатели сравнительной экономической эффективности от внедрения программного продукта

Наименование показателей	Значение
Ежегодная прибыль, тенге	1 012 680
Коэффициент экономической эффективности капитальных вложений (Ер)	0,77
Срок окупаемости капитальных вложении (Тр)	1,29

Таким образом, разработанная информационная система будет ежегодно приносить прибыль 1 012 680 тенге и по нашим маркетинговым исследованиям ежегодно будет увеличиваться примерно на 5%.

Заключение

Создание компьютерных игр в жанре RPG является трудоемким процессом, требующим как навыков программирования, так и иллюстратора, 3D скульптора и сценариста, от каждого из которых зависит успешность продукта.

В нынешнее время ниша видеоигр популярна и развивается вместе с компьютерными технологиями. Требования современных игроков становятся все изощрённее, но и количество игроков большое, вследствие чего, несмотря на конкуренцию, любые игры могут найти свою целевую аудиторию.

Видеоигры, в отличии от физических продуктов являются информационным товаром, поэтому их легко распространять, хранить и значительно упрощается политика продажи продукта. Создавая такой продукт однажды и размещая его на востребованных интернет магазинах игр можно обеспечиться почти непрерывным постоянным доходом практически без эксплуатационных издержек.

Список литературы

1 C# Game Programming Cookbook for Unity 3D, Jeff Murray, 2014;
2 Learning C# by Developing Games with Unity 3D Beginner's Guide, Terry Norton, 2013;
3 Head First C#, Jennifer Greene, Andrew Stellman (рус.: Изучаем C#, Д. Грин, Э. Стиллмен);
4 Язык программирования C# 5.0 и платформа .NET 4.5 - Эндрю Троелсен;
5 C# 4.0: полное руководство, Герберт Шилдт;
6 CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#;
7 Wikipedia.org.
8 https://blackforest.su/articles/skyrim-creation
9 Unity for Absolute Beginners. Sue Blackman
10 Unity Game Development in 24 Hours. Mike Geig
11 Learning C # Programming with Unity 3D. Alex Okita
12 Язык программирования C#. Классика Computers Science. 4-е изд. Авторы: А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд
13 Изучаем C#. 3-е изд. Авторы: Э. Стиллмен, Дж. Грин
14 CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. Автор: Дж. Рихтер
15 **metanit.com**
16 “C# 5.0 in a Nutshell” by Joseph Albahari, Ben Albahari, 5th Edition, 2012

Приложение А
Техническое задание

1 Основные требования:

- название разрабатываемой программы: Roleplay Game.
- цель разработки: реализация программного продукта, развлекательного характера.

Рекомендуемые платформы для разработки приложения (на выбор разработчика):

- Unity;
- UnrealEngine;
- Cryengine;
- Phaser;
- Turbulenz;
- GameSalad.

общий объем программной части системы: не более 4GB.

2 Технические требования:

- Операционная система: Windows, Linux;
- Процессор не менее Pentium IV;
- 1ГБ свободного пространства физического носителя.

3 Экономические требования.

Расчет сметы разработки программного продукта (подлежит обсуждению):

- стоимость готового продукта 1 309 290 тг;

- стоимость разработки 1012680.

Аудитория направлена на людей, владеющих компьютером и увлекающихся компьютерными играми.

Приложение Б – Листинг программы

Листинг файла Body.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Body : MonoBehaviour,IHealth,IArmor,ITakeDamage
{

    const string projectileFolder = "Prefabs/Projectiles/";

    public static bool isAtacking;


    public GameObject hips;//root bone

    public Transform leftWeaponT;
    public Transform rightWeaponT;

    public Transform twoHandedWeaponT;
    public float currentHealth;

    public Armor Armor_ { get; }
    float defaultPushTimer = 0.3f;
    float pushTimer;
    bool isPushing;
    CapsuleCollider capsuleCollider;
    Animator animator;
    bool weaponEnabled;
    Rigidbody rigidbody;
    Vector3 delayedForce;
    List<Collider> ragdollColliders = new List<Collider>();
    List<Rigidbody> ragdollRigidbodyes = new
List<Rigidbody>();
    int currentWeaponId;
    bool isArmed;
    public bool isDead;
    public event Action onDie;
    void Awake()
    {
        GameDataOptimizer.InsertValue(this);
        animator = GetComponent<Animator>();
```

```
        rigidbody = GetComponent<Rigidbody>();
        capsuleCollider = GetComponent<CapsuleCollider>();


        Health_.baseValue = 100;


        /*
        var joints =
GetComponentsInChildren<CharacterJoint>();
        foreach (var j in joints)
        {
            Destroy(j);
        }

        var rigB = GetComponentsInChildren<Rigidbody>();

        foreach (Rigidbody r in rigB)
        {
            if (r != rigidbody)
```

Продолжение приложения Б

```
                Destroy(r);
        }
        */
        ragdollColliders = new
List<Collider>(GetComponentsInChildren<Collider>());
        ragdollRigidbodyes = new
List<Rigidbody>(GetComponentsInChildren<Rigidbody>());

        EnableRagdoll(false);
    }
    void Start()
    {

    }
    // Update is called once per frame

    private void FixedUpdate()
    {
        if (delayedForce.magnitude > 0)
        {
```

```
            animator.applyRootMotion = false;

            rigidbody.AddForce(delayedForce,
ForceMode.Impulse);
            delayedForce = Vector3.zero;

        }


        pushTimer -= Time.fixedDeltaTime;
        if (pushTimer <= 0)
        {
            isPushing = false;
            animator.applyRootMotion = true;
        }
    }
    private void Update()
    {
        if(isPushing)
            animator.applyRootMotion = false;
    }
    public ChangableValue Health_ { get; } = new
ChangableValue();

    // Start is called before the first frame update

    public void TakeDamage(float damage)
    {
        currentHealth -= damage;
        var dn =
Instantiate<GameObject>(Resources.Load<GameObject>(DamageNumb
er.Path)).GetComponent<DamageNumber>();
        dn.text = damage.ToString();
        Vector3 v3 = transform.position;
        v3.y += 2;
        dn.transform.position = v3;
        if (currentHealth<=0)
        {
            Die();
        }
        Debug.Log("Body.TakeDamage()");
    }
    void Die()
```

```
    {
        Debug.Log("Body.Die()");
```

Продолжение приложения Б

```
        EnableRagdoll(true);
        isDead = true;
        onDie?.Invoke();
    }

    public void Push(Vector3 velocity)
    {
        Debug.Log("PushForce: " + (velocity -
rigidbody.centerOfMass));
        delayedForce = velocity - rigidbody.centerOfMass;
        delayedForce.y += rigidbody.mass / 2;
        pushTimer = defaultPushTimer;
        isPushing = true;
    }
    void EnableRagdoll(bool enable)
    {
        foreach (var rag in ragdollColliders)
        {
            rag.enabled = enable;
        }



        foreach (var rig in ragdollRigidbodyes)
        {
            rig.isKinematic = !enable;
        }



        capsuleCollider.isTrigger = enable;
        capsuleCollider.enabled = true;

        rigidbody.isKinematic = false;
        rigidbody.useGravity = true;
        if (!enable)
        {
            rigidbody.constraints =
RigidbodyConstraints.FreezeRotation;
```

```
        }
        else
        {
            rigidbody.mass = 0;
            rigidbody.constraints =
RigidbodyConstraints.None;
            FixedJoint fj = hips.AddComponent<FixedJoint>();
            fj.connectedBody = rigidbody;
        }
        animator.enabled = !enable;
    }
}
```

Листинг файла Character.cs

```
using System.Collections.Generic;
using UnityEngine;
public class Character : MonoBehaviour
{
    [SerializeField] float m_MovingTurnSpeed = 360;
```

Продолжение приложения Б

```
    [SerializeField] float m_StationaryTurnSpeed = 180;
    [SerializeField] float m_JumpPower = 12f;
    [Range(1f, 4f)] [SerializeField] float m_GravityMultiplier = 2f;
    [SerializeField] float m_RunCycleLegOffset = 0.2f; //specific to the
character in sample assets, will need to be modified to work with others
    [SerializeField] float m_MoveSpeedMultiplier = 1f;
    [SerializeField] float m_AnimSpeedMultiplier = 1f;
    [SerializeField] float m_GroundCheckDistance = 0.1f;

    public float multiplier;
    float defaultMultuplier;

    [SerializeField]
    string defaultWeaponPath;

    Weapon defautWeapon;

    public Transform leftHand;
    public Transform rightHand;

    Rigidbody m_Rigidbody;
```

```
Animator m_Animator;
bool m_IsGrounded;
float m_OrigGroundCheckDistance;
const float k_Half = 0.5f;
float m_TurnAmount;
float m_ForwardAmount;
Vector3 m_GroundNormal;
float m_CapsuleHeight;
Vector3 m_CapsuleCenter;
CapsuleCollider m_Capsule;
bool m_Crouching;
[SerializeField]
Weapon weapon;
protected AnimatorOverrideController animatorOverrideController;

int jumpTimer = 0;//frames
bool IsActing { get; set; } = false;
[SerializeField]
public Inventory inventory;
Body m_body;
float landingAnimationLength;
```

Продолжение приложения Б

```
public void LoadData()
{
}
public void ChangeWeapon(Weapon weapon)
{
   if (weapon != null)
   {
      weapon.Equip(rightHand);
      weapon.owner = gameObject;
      this.weapon = weapon;
   }
   else
   {
      if (this.weapon != null)
         this.weapon.Pick(transform);
      this.weapon = defautWeapon;
      this.weapon.Equip(rightHand);
   }
      }
void DropWeapon()
{
```

```
            if (defautWeapon != null)
                Destroy(defautWeapon.gameObject);
            if (weapon != null)
                weapon.Throw();
        }
        /*
        public void CreateProjectile(string projectileName)
        {
            GameObject projectile =
Instantiate<GameObject>(Resources.Load<GameObject>(projectileFolder +
projectileName));
            projectile.transform.position = rightHand.position;
            projectile.GetComponent<Projectile>().owner = this.gameObject;
            projectile.GetComponent<Rigidbody>().AddForce(transform.forward *
800, ForceMode.Acceleration);
        }*/
        public void AddItemToInventory(Item item)
        {
            Debug.Log("Character.AddItemToInventory(), Inventory:");
            Debug.Log(inventory);
            inventory.InsertItem(item);
```

Продолжение приложения Б

```
        }
        public void OnActionAnimationEnd()
        {
            IsActing = false;
            //Debug.Log("Character.OnActionAnimationEnd()");
        }

        void Start()
        {
            defautWeapon =
Instantiate(Resources.Load<GameObject>(defaultWeaponPath),
rightHand).GetComponent<Weapon>();
            defautWeapon.owner = this.gameObject;
            defautWeapon.Pick(this.transform);
            if (weapon != null)
            {
                ChangeWeapon(weapon);

            }
            else
            {
```

```
                ChangeWeapon(defautWeapon);
            }
            m_body = GetComponent<Body>();
            m_body.onDie += DropWeapon;
            inventory = GetComponent<Inventory>();
            m_Animator = GetComponent<Animator>();
            m_Rigidbody = GetComponent<Rigidbody>();
            m_Capsule = GetComponent<CapsuleCollider>();
            m_CapsuleHeight = m_Capsule.height;
            m_CapsuleCenter = m_Capsule.center;
            m_Rigidbody.constraints = RigidbodyConstraints.FreezeRotationX |
RigidbodyConstraints.FreezeRotationY | RigidbodyConstraints.FreezeRotationZ;
            m_OrigGroundCheckDistance = m_GroundCheckDistance;
            /*
            if (m_Animator != null)
            {
                var ac = m_Animator.runtimeAnimatorController as
UnityEditor.Animations.AnimatorController;
                UnityEditor.Animations.AnimatorStateMachine sm =
ac.layers[0].stateMachine;
```

Продолжение приложения Б

```
                for (int i = 0; i < sm.states.Length; i++)
                {
                    var state = sm.states[i];
                    if (state.state.name == "Landing")
                    {
                        AnimationClip clip = state.state.motion as AnimationClip;
                        if (clip != null)
                        {
                            landingAnimationLength = clip.length;
                            //Debug.Log("landingAnimationLength: " +
landingAnimationLength);
                        }
                    }
                }
            }
            */
            animatorOverrideController = new
AnimatorOverrideController(m_Animator.runtimeAnimatorController);
            m_Animator.runtimeAnimatorController = animatorOverrideController;
            GameDataOptimizer.InsertValue(this);
```

```
        }

        private void FixedUpdate()
        {
            CheckGroundStatus();
            HandleAirborneMovement();
        }
        public void Move(Vector3 move, bool crouch, bool jump)
        {
            //Debug.Log("isActing: " + isActing);

            if (IsActing)
            {
                if (move.magnitude > 1f) move.Normalize();
                move = transform.InverseTransformDirection(move);
                move = Vector3.ProjectOnPlane(move, m_GroundNormal);
                m_TurnAmount = Mathf.Atan2(move.x, move.z);
                m_ForwardAmount = move.z;
                ApplyExtraTurnRotation();
            }
            else
```

Продолжение приложения Б

```
            {
                // convert the world relative moveInput vector into a local-relative
                // turn amount and forward amount required to head in the desired
                // direction.
                if (move.magnitude > 1f) move.Normalize();
                move = transform.InverseTransformDirection(move);
                move = Vector3.ProjectOnPlane(move, m_GroundNormal);
                m_TurnAmount = Mathf.Atan2(move.x, move.z);
                m_ForwardAmount = move.z;
                ApplyExtraTurnRotation();

                // control and velocity handling is different when grounded and
airborne:
                if (m_IsGrounded)
                {
                    if (jump && !crouch &&
m_Animator.GetCurrentAnimatorStateInfo(0).IsName("Grounded"))
                    {
                        Jump();
                    }
                }
```

```
                else
                {
                }
                //ScaleCapsuleForCrouching(crouch);
                //PreventStandingInLowHeadroom();
                // send input and other state parameters to the animator
            }
            UpdateAnimator(move);
        }

        public void StartAttack()
        {
            if (weapon != null)
                weapon.StartAtack();
        }
        public void EndAttack()
        {
            if (weapon != null)
                weapon.EndAttack();
        }
        void Jump()
```

Продолжение приложения Б

```
        {
            //Debug.Log("Jump()");
            jumpTimer = 10;
            m_Rigidbody.velocity = new Vector3(m_Rigidbody.velocity.x,
m_JumpPower, m_Rigidbody.velocity.z);
            m_IsGrounded = false;
            m_Animator.applyRootMotion = false;
        }
        void HandleAirborneMovement()
        {
            if (!m_IsGrounded)
            {
                //Debug.Log("HandleAirborneMovement();");
                m_Animator.applyRootMotion = false;
                if (jumpTimer < 0)
                {
                    if (m_Rigidbody.velocity.y < 0)//falling down
                    {
                        //Debug.Log("Physics.gravity.y: " + Physics.gravity.y);
                        //Debug.Log("m_Rigidbody.velocity.y: " +
m_Rigidbody.velocity.y);
```

```
                    float landingDistance = (m_Rigidbody.velocity.y +
(Physics.gravity.y * landingAnimationLength)) * landingAnimationLength * (-
0.05f);
                    //Debug.Log("landingDistance: " + landingDistance);
                    if (Physics.Raycast(transform.position + (Vector3.up * 0.1f),
Vector3.down, out RaycastHit hit, landingDistance))
                    {
                        //landing
                        m_Animator.Play("Landing");
                    }
                    else
                    {
                        //falling
                        m_Animator.Play("Falling");
                    }
                }
                else//going up
                {
                    m_Animator.Play("Jump");
                }
            }
```

Продолжение приложения Б

```
            else
            {
                jumpTimer--;
            }
        }
    }

    void ScaleCapsuleForCrouching(bool crouch)
    {
        if (m_IsGrounded && crouch)
        {
            if (m_Crouching) return;
            m_Capsule.height = m_Capsule.height / 2f;
            m_Capsule.center = m_Capsule.center / 2f;
            m_Crouching = true;
        }
        else
        {
            Ray crouchRay = new Ray(m_Rigidbody.position + Vector3.up *
m_Capsule.radius * k_Half, Vector3.up);
```

```
                float crouchRayLength = m_CapsuleHeight - m_Capsule.radius *
k_Half;
                if (Physics.SphereCast(crouchRay, m_Capsule.radius * k_Half,
crouchRayLength, Physics.AllLayers, QueryTriggerInteraction.Ignore))
                {
                    m_Crouching = true;
                    return;
                }
                m_Capsule.height = m_CapsuleHeight;
                m_Capsule.center = m_CapsuleCenter;
                m_Crouching = false;
            }
        }

        /*void PreventStandingInLowHeadroom()
        {
            // prevent standing up in crouch-only zones
            if (!m_Crouching)
            {
                Ray crouchRay = new Ray(m_Rigidbody.position + Vector3.up *
m_Capsule.radius * k_Half, Vector3.up);
```

Продолжение приложения Б

```
                float crouchRayLength = m_CapsuleHeight - m_Capsule.radius *
k_Half;
                if (Physics.SphereCast(crouchRay, m_Capsule.radius * k_Half,
crouchRayLength, Physics.AllLayers, QueryTriggerInteraction.Ignore))
                {
                    m_Crouching = true;
                }
            }
        }*/


        void UpdateAnimator(Vector3 move)
        {
           // Debug.Log("m_GroundCheckDistance: " + m_GroundCheckDistance);
            // update the animator parameters
            m_Animator.SetFloat("Forward", m_ForwardAmount, 0.1f,
Time.deltaTime);
            m_Animator.SetFloat("Turn", m_TurnAmount, 0.1f, Time.deltaTime);
            m_Animator.SetBool("Crouch", m_Crouching);
            m_Animator.SetBool("OnGround", m_IsGrounded);
```

```
            if (!m_IsGrounded)
            {
                m_Animator.SetFloat("Jump", m_Rigidbody.velocity.y);
            }

            // calculate which leg is behind, so as to leave that leg trailing in the jump
animation
            // (This code is reliant on the specific run cycle offset in our animations,
            // and assumes one leg passes the other at the normalized clip times of 0.0
and 0.5)
            float runCycle =
                Mathf.Repeat(
                    m_Animator.GetCurrentAnimatorStateInfo(0).normalizedTime +
m_RunCycleLegOffset, 1);
            float jumpLeg = (runCycle < k_Half ? 1 : -1) * m_ForwardAmount;
            if (m_IsGrounded)
            {
                m_Animator.SetFloat("JumpLeg", jumpLeg);
            }

            // the anim speed multiplier allows the overall speed of walking/running
to be tweaked in the inspector,
```

Продолжение приложения Б

```
            // which affects the movement speed because of the root motion.
            if (m_IsGrounded && move.magnitude > 0)
            {
                m_Animator.speed = m_AnimSpeedMultiplier;
            }
            else
            {
                // don't use that while airborne
                m_Animator.speed = 1;
            }
        }


        /*void HandleAirborneMovement()
        {
            // apply extra gravity from multiplier:
            Vector3 extraGravityForce = (Physics.gravity * m_GravityMultiplier) -
Physics.gravity;
            m_Rigidbody.AddForce(extraGravityForce);
```

```
            m_GroundCheckDistance = m_Rigidbody.velocity.y < 0 ?
m_OrigGroundCheckDistance : 0.01f;
        }*/

        /*
        void HandleGroundedMovement(bool crouch, bool jump)
        {
            // check whether conditions are right to allow a jump:
            if (jump && !crouch &&
m_Animator.GetCurrentAnimatorStateInfo(0).IsName("Grounded"))
            {
                // jump!
                m_Rigidbody.velocity = new Vector3(m_Rigidbody.velocity.x,
m_JumpPower, m_Rigidbody.velocity.z);
                m_IsGrounded = false;
                m_Animator.applyRootMotion = false;
                m_GroundCheckDistance = 0.1f;
            }
        }
        */
        void ApplyExtraTurnRotation()
        {
```

Продолжение приложения Б

```
            // help the character turn faster (this is in addition to root rotation in the
animation)
            float turnSpeed = Mathf.Lerp(m_StationaryTurnSpeed,
m_MovingTurnSpeed, m_ForwardAmount);
            transform.Rotate(0, m_TurnAmount * turnSpeed * Time.deltaTime, 0);
        }


        /*public void OnAnimatorMove()
        {
            // we implement this function to override the default root motion.
            // this allows us to modify the positional speed before it's applied.
            if (m_IsGrounded && Time.deltaTime > 0)
            {
                Vector3 v = (m_Animator.deltaPosition * m_MoveSpeedMultiplier) /
Time.deltaTime;

                // we preserve the existing y part of the current velocity.
                v.y = m_Rigidbody.velocity.y;
                m_Rigidbody.velocity = v;
```

```
            }
        }
        */

        void CheckGroundStatus()
        {
            if (jumpTimer <= 0)
            {
                //Debug.Log("CheckGroundStatus");

    #if UNITY_EDITOR
                // helper to visualise the ground check ray in the scene view
                Debug.DrawLine(transform.position + (Vector3.up * 0.1f),
transform.position + (Vector3.up * 0.1f) + (Vector3.down *
m_GroundCheckDistance));
    #endif
                // 0.1f is a small offset to start the ray from inside the character
                // it is also good to note that the transform position in the sample assets
is at the base of the character
                if (Physics.Raycast(transform.position + (Vector3.up * 0.1f),
Vector3.down, out RaycastHit hitInfo, m_GroundCheckDistance))
                {
```

Продолжение приложения Б

```
                    m_GroundNormal = hitInfo.normal;
                    m_IsGrounded = true;
                    m_Animator.applyRootMotion = true;
                }
                else
                {
                    m_IsGrounded = false;
                    m_GroundNormal = Vector3.up;
                    m_Animator.applyRootMotion = false;
                }
            }
        }
        public void Action(AnimationType animationType,Vector3 target)
        {
            if (!IsActing)
            {
                Vector3 copy = transform.eulerAngles;
                transform.LookAt(target);
                copy.y = transform.eulerAngles.y;
                transform.eulerAngles = copy;
```

```
                if (animationType == AnimationType.EasyAttack )
                {
                    var a = weapon.comboSet.NextEasyAttack();
                    weapon.modifier = a.damageModifier * multiplier;
                    m_Animator.SetFloat("AttackSpeed", a.speedModifier);
                    animatorOverrideController[AnimationType.EasyAttack.ToString()]
= a.animation;
                    m_Animator.Update(0.0f);
                }else if(animationType == AnimationType.HardAttack)
                {
                    var a = weapon.comboSet.NextHardAttack();
                    weapon.modifier = a.damageModifier * multiplier;
                    m_Animator.SetFloat("AttackSpeed", a.speedModifier);
                    animatorOverrideController[AnimationType.HardAttack.ToString()]
= a.animation;
                    m_Animator.Update(0.0f);
                }
                m_Animator.Play(animationType.ToString());
                IsActing = true;
            }
```



```
        }
    }
```

Листинг файла UserControl.cs

```
using System;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;
[RequireComponent(typeof(Character))]
[RequireComponent(typeof(Body))]
public class UserControl : MonoBehaviour
{
    private Character m_Character; // A reference to the
ThirdPersonCharacter on the object
    private Transform m_Cam;                  // A reference
to the main camera in the scenes transform
    private Vector3 m_CamForward;             // The current
forward direction of the camera
    private Vector3 m_Move;
```

```
    private bool m_Jump;                      // the world-
relative desired move direction, calculated from the
camForward and user input.
    //private Body m_Body;
    public static bool isControllable = true;

    Vector3 prevPosition;
    [SerializeField]
    float interactionRadius;
    InteractiveObject selectedInteractiveObj;
    private CanvasController canvas;
    public void PlayLeftClickAnim()
    {
        m_Character.Action(AnimationType.EasyAttack,
transform.position + transform.forward);

    }

    public void PlayRightClickAnim()
    {
        m_Character.Action(AnimationType.HardAttack,
transform.position + transform.forward);
    }

    private void Start()
    {
        //Time.timeScale = 0.25f;
        prevPosition = transform.position;
        // get the transform of the main camera
        if (Camera.main != null)
        {
            m_Cam = Camera.main.transform;
        }
        else
        {
            Debug.LogWarning(
                "Warning: no main camera found. Third person
character needs a Camera tagged \"MainCamera\", for camera-
relative controls.", gameObject);
            // we use self-relative controls in this case,
which probably isn't what the user wants, but hey, we warned
them!
        }
```

```
        // get the third person character ( this should never
be null due to require component )
        m_Character = GetComponent<Character>();
        var canvasGO =
Instantiate(Resources.Load<GameObject>("Prefabs/Canvas"));
        canvas = canvasGO.GetComponent<CanvasController>();
        canvas.player = this.gameObject;
    }

    private void Update()
    {
        UpdateSphereCast();
        if (Input.GetKeyDown(KeyCode.U))
        {
            canvas.weaponSlider.Next();
        }

        if (isControllable)
        {
            if (Input.GetKeyDown(KeyCode.I))
            {

canvas.inventoryItemPanelContainer.gameObject.SetActive(true)
;
                isControllable = false;
            }
            if (Input.GetKeyDown(KeyCode.F))
            {
                if (selectedInteractiveObj != null)

selectedInteractiveObj.Interact(m_Character);
            }
        }
        else
        {
            if (Input.GetKeyDown(KeyCode.I))
            {

canvas.inventoryItemPanelContainer.gameObject.SetActive(false
);
```

```
                isControllable = true;
            }
            if (Input.GetKeyDown(KeyCode.K))
            {
                canvas.weaponSlider.Expand();
            }
        }

    }
    void UpdateSphereCast()
    {
        Ray ray =
Camera.main.ScreenPointToRay(Input.mousePosition);
        Debug.DrawRay(ray.origin,ray.direction);
        RaycastHit[] raycastHits =
Physics.SphereCastAll(ray,interactionRadius);
        foreach (var hit in raycastHits)
        {
            var hint =
hit.collider.gameObject.GetComponent<IShowHint>();
            var interactive =
hit.collider.gameObject.GetComponent<InteractiveObject>();
            if (hint != null)
            {
                //Debug.Log("UserControl.ShowHint");
                hint.ShowHint();
            }
            if (interactive != null)
            {
                selectedInteractiveObj = interactive;
```

Продолжение приложения Б

```
            }
        }
    }
    private void FixedUpdate()
    {

        prevPosition = transform.position;
        if (!m_Jump)
        {
            m_Jump =
CrossPlatformInputManager.GetButtonDown("Jump");
```

```
        }
        if (isControllable)
        {

            // read inputs
            if (Input.GetMouseButton(0))
            {
                PlayLeftClickAnim();
            }
            if (Input.GetMouseButton(1))
            {
                PlayRightClickAnim();
            }

            float h =
CrossPlatformInputManager.GetAxis("Horizontal");
            float v =
CrossPlatformInputManager.GetAxis("Vertical");
            bool crouch = Input.GetKey(KeyCode.C);

            // calculate move direction to pass to character
            if (m_Cam != null)
            {
                // calculate camera relative direction to
move:
                m_CamForward = Vector3.Scale(m_Cam.forward,
new Vector3(1, 0, 1)).normalized;
                m_Move = v * m_CamForward + h * m_Cam.right;
            }
            else
            {
                // we use world-relative directions in the
case of no main camera
                m_Move = v * Vector3.forward + h *
Vector3.right;
            }
#if !MOBILE_INPUT
            // walk speed multiplier
            if (Input.GetKey(KeyCode.LeftShift)) m_Move *=
0.5f;
#endif
```

```
            // pass all parameters to the character control
script
            m_Character.Move(m_Move, crouch, m_Jump);
            m_Jump = false;
        }
        else
        {
            m_Character.Move(m_Move*0, false, m_Jump);
        }

    }


    // Fixed update is called in sync with physics

}
```

Продолжение приложения Б

Листинг файла AIControl.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;


[RequireComponent(typeof(Character))]
[RequireComponent(typeof(Body))]
public class AIControl : MonoBehaviour
{
    private Character m_Character; // A reference to the
ThirdPersonCharacter on the object
    private Transform m_Cam;                  // A reference
to the main camera in the scenes transform
    private Vector3 m_CamForward;             // The current
forward direction of the camera
    private Vector3 m_Move;
    private bool m_Jump;                      // the world-
relative desired move direction, calculated from the
camForward and user input.
    //private Body m_Body;
    public static bool isControllable = true;
```

```
    Vector3 prevPosition;
    [SerializeField]
    float interactionRadius;
    InteractiveObject selectedInteractiveObj;

    State state = State.Attack;

    Character curentTarget;


    float attackDistance = 1.3f;


    private void Start()
    {
        // get the third person character ( this should never
be null due to require component )
        m_Character = GetComponent<Character>();
    }

    private void FixedUpdate()
    {
        if ((int)(Time.time * 10) % 5 == 0)
            CheckForEnemy();
        switch (state)
        {
            case State.Attack:
                if (curentTarget != null)
                {
                    if (Vector3.Distance(transform.position,
curentTarget.transform.position) < attackDistance)
                    {
                        float r =
UnityEngine.Random.Range(0f,1f);
                        if (r < 0.3f)
                        {
```

Продолжение приложения Б

```
m_Character.Action(AnimationType.HardAttack,
curentTarget.transform.position);
```

```
                        }
                        else if(r<0.9f)
                        {
m_Character.Action(AnimationType.EasyAttack,
curentTarget.transform.position);
                        }
                        else
                        {
m_Character.Move((curentTarget.transform.position -
transform.position).normalized, false, false);
                        }
                    }
                    else
                    {
m_Character.Move((curentTarget.transform.position -
transform.position).normalized, false, false);

                    }
                }

                break;

            case State.Idle:
                break;
            case State.Run:
                break;
        }
    }


    void CheckForEnemy()
    {
        //Debug.Log("AIControl.CheckForEnemy()");
        List<Character> characters =
GameDataOptimizer.GetValue<Character>();
        if (characters!=null&&characters.Count > 0)
        {
            //Debug.Log("characters.Count: " +
characters.Count);
            Character closestCharacter = null;
```

```
            float closestDistance =float.MaxValue;
            foreach (var c in characters)
            {

                if (c !=
m_Character&&c.GetComponent<Body>().isDead==false)
                {


                    float d =
Vector3.Distance(transform.position, c.transform.position);
                    if (d < closestDistance)
                    {
                        closestDistance = d;
                        closestCharacter = c;
                    }
                }


            }
            if (closestCharacter != m_Character)
            {
```

Продолжение приложения Б

```
                curentTarget = closestCharacter;

            }

        }

    }

    // Fixed update is called in sync with physics


    enum State
    {
        Attack,
        Run,
        Idle
    }

}
```

Листинг Weapon.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Weapon : Item
{
    public float damage;
    public GameObject owner;
    public AttackComboSet comboSet;
    public float pushForce = 30;
    public float modifier;

    bool isAtacking = false;

    List<Body> hitBodies =  new List<Body>();
    protected override void Start()
    {
        base.Start();
    }
    public void StartAtack()
    {
        attackTimer = 3;
        isAtacking = true;
    }
    public void EndAttack()
    {
        hitBodies.Clear();
        Debug.Log("Weapon.EndAttack()");
        isAtacking = false;
    }
    private void Update()
    {

    }
    public void DealDamage(Body body,Vector3 hitPoint)
    {

        if (isAtacking&& owner.GetComponent<Body>()!= body
&&!hitBodies.Contains(body))
```

Продолжение приложения Б

```
        {
            hitBodies.Add(body);
            body.Push(((hitPoint ).normalized *
pushForce*modifier));
            body.TakeDamage(damage * modifier);// *
multiplyer);
            Debug.Log("Final Damage: " + damage );
        }
    }
    private void OnTriggerEnter(Collider other)
    {
        Body body = other.GetComponent<Body>();
        if (body != null&&owner!=null)
            DealDamage(body, other.transform.position -
owner.transform.position);
    }

}
```

Листинг Item.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Item : MonoBehaviour,IInteract
{
    public Sprite sprite;
    public string itemName;
    public string objectPath;
    public ItemType itemType;
    public bool IsPickedUp { get; private set; }
    InteractiveObject _interactive;
    public InteractiveObject Interactive {
        get
        {
            if (_interactive==null)
            {
                _interactive =
Instantiate(Resources.Load<GameObject>(InteractiveObject.path
), transform).GetComponent<InteractiveObject>();
            }
            return _interactive;
```

```
        }
        set { _interactive = value; }
    }
    protected virtual void Start()
    {
        Interactive.text = itemName;
    }
    public void Throw()
    {
        IsPickedUp = false;
        Interactive.interactEnabled = true;
        gameObject.SetActive(true);
        transform.parent = null;
    }
    public void Pick(Transform parent)
    {
        IsPickedUp = true;
        Interactive.interactEnabled = false;

        //Debug.Log("Item.Pick");
        gameObject.SetActive(false);
        transform.SetParent(parent);
```

Продолжение приложения Б

```
    }
    public void Hide()
    {
        gameObject.SetActive(false);
    }
    public void Equip(Transform parent)
    {
        IsPickedUp = true;
        Interactive.interactEnabled = false;

        gameObject.SetActive(true);
        transform.SetParent(parent);
        transform.position = parent.transform.position;
        transform.rotation = parent.rotation;
    }
    public void Destroy()
    {
        Destroy(gameObject);
    }
```

```
    public void Interact(Character character,
InteractiveObject interactiveObject)
    {
        if (!IsPickedUp)
        {
            Interactive = interactiveObject;
            Pick(character.transform);
            character.AddItemToInventory(this);
        }



    }

    public enum ItemType
    {
        Weapon,
        Armor,
        Others,
    }
}
```

Листинг InteractiveObject:

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

public class InteractiveObject : MonoBehaviour,IShowHint
{
    public static readonly string path =
"Prefabs/InteractiveObject";
    [HideInInspector]
    public TextMeshPro meshPro;
    [HideInInspector]
    public SphereCollider sphereCollider;
    public bool showHint = true;
    public string text;
    public bool interactEnabled = true;
    public void ShowHint()
    {
        if (interactEnabled)
        {
```

```
            Debug.Log("InteractiveObject.ShowHint()");

            meshPro.enabled = true;
            transform.rotation = 
Camera.main.transform.rotation;
            enabled = true;
            showHint = true;
        }

    }
    public void Interact(Character character)
    {
        Debug.Log("InteractiveObject.Interact");
        IInteract interact = 
GetComponentInParent<IInteract>();
        if (interact != null)
            interact.Interact(character, this);
    }
    // Start is called before the first frame update
    void Start()
    {
        sphereCollider = GetComponent<SphereCollider>();
        meshPro = GetComponent<TextMeshPro>();
        meshPro.text = text;
    }

    // Update is called once per frame
    void Update()
    {

        if(meshPro!=null&&!showHint)
        meshPro.enabled = false;
        showHint = false;
    }
}
```

Листинг ArenaConroller:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ArenaController : MonoBehaviour
```

```
{
    [SerializeField]
    List<GameObject> bosses;
    [SerializeField]
    GameObject player;
    int currentBoss = 0;
    float sec = 10.0f;
    // Start is called before the first frame update
    bool coroutineIsRunnig = false;
    void Start()
    {
        bosses[currentBoss].SetActive(true);
    }

    // Update is called once per frame
    void Update()
    {
        if (currentBoss<bosses.Count&&
bosses[currentBoss].GetComponent<Body>().isDead)
        {
            if(!coroutineIsRunnig)
            StartCoroutine(WaitForSec());
        }
```

Продолжение приложения Б

```
    }
    IEnumerator WaitForSec()
    {
        coroutineIsRunnig = true;
        yield return new WaitForSeconds(sec);

        player.GetComponent<Body>().currentHealth =
400*((currentBoss+1)/(bosses.Count/2));
        player.GetComponent<Character>().multiplier *= 1.1f;

        bosses[currentBoss].SetActive(false);

        currentBoss++;
        if (currentBoss < bosses.Count)
            bosses[currentBoss].SetActive(true);
        coroutineIsRunnig = false;
    }
}
```

Листинг AttackSet

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(menuName = "custom/AttackComboSet", fileName
="newAttackComboSet")]
public class AttackComboSet : ScriptableObject
{
    public List<Attack> easyAttacks;

    public List<Attack> hardAttacks;


     int nextEasyAnimationID = -1;

     int nextHardAnimationID = -1;

    public Attack NextEasyAttack()
    {
        nextEasyAnimationID++;

        if (nextEasyAnimationID > easyAttacks.Count - 1)
            nextEasyAnimationID = 0;

        return easyAttacks[nextEasyAnimationID];
    }

    public Attack NextHardAttack()
    {
        nextHardAnimationID++;
        if (nextHardAnimationID > hardAttacks.Count - 1)
            nextHardAnimationID = 0;

        return hardAttacks[nextHardAnimationID];
    }
}
```

Листинг CameraController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    [SerializeField] Transform cameraY;
    [SerializeField] Transform cameraX;

    [SerializeField] float mouseSensetivity = 100f;
    [SerializeField] float yStopper = 0f;

    [SerializeField] float smoothSpeed = 0.125f;

    public Transform target;


    float xRotation = 0f;
    float yRotation = 0f;
    void Start()
    {


    }
    private void Update()
    {
        if (UserControl.isControllable)
        {
            //transform.LookAt(target);
            Cursor.lockState = CursorLockMode.Locked;

            float mouseX = Input.GetAxis("Mouse X") *
mouseSensetivity * Time.deltaTime;
            float mouseY = Input.GetAxis("Mouse Y") *
mouseSensetivity * Time.deltaTime;

            xRotation -= mouseY;
            // yRotation += ;
```

```
            xRotation = Mathf.Clamp(xRotation, -90f, 90f - 
yStopper);


            cameraY.localRotation = 
Quaternion.Euler(xRotation, 
cameraY.localRotation.eulerAngles.y, 0f);

            cameraY.Rotate(Vector3.up * mouseX);

        }
        else
        {
            Cursor.lockState = CursorLockMode.None;
        }

    }
    void FixedUpdate()
    {
            cameraY.position = Vector3.Lerp(cameraY.position, 
target.position, smoothSpeed);
    }
}
```

Продолжение приложения Б

Листинг InventoryItemPanelContainer

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InventoryItemPanelContainer : 
MonoBehaviour,IItemSwap
{
    Inventory playerInventory;
    List<ItemPanel> itemPanels = new List<ItemPanel>();

    public bool ItemSwap(ItemPanel itemPanel)
    {
        List<Item> items = new List<Item>();
        foreach(ItemPanel ip in itemPanels)
        {
            items.Add(ip.Item);
```

```
        }
        playerInventory.SetItems(items);
        return true;
    }
    void Start()
    {
        playerInventory =
GetComponentInParent<CanvasController>().player.GetComponent<
Character>().inventory;

        playerInventory.GetItems();
        //Debug.Log("Inventory.Start()");

        playerInventory.onItemsChange += UpdateItems;
        //Debug.Log("playerInventory.onItemsChange:");
        //Debug.Log(playerInventory);

        int capacity = playerInventory.Capacity;

        var items = new Item[capacity];

        foreach(Item item in items)
        {
            ItemPanel panel=
Instantiate(Resources.Load<GameObject>(ItemPanel.path),
transform).GetComponent<ItemPanel>();
            panel.Item = item;
            itemPanels.Add(panel);
        }
        gameObject.SetActive(false);
    }
    void UpdateItems(List<Item> items)
    {
        for(int i=0;i<items.Count;i++)
        {
            itemPanels[i].Item = items[i];
        }
    }
}
```

Продолжение приложения Б

Листинг Inventory

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
public class Inventory :MonoBehaviour
{

    int capacity = 10;
    public int Capacity { get { return capacity; } }
    List<Item> items;
    public delegate void OnItemsChange(List<Item> items);
    public OnItemsChange onItemsChange;
    int NextEmptySlotID
    {
        get
        {
            for (int i=0;i<items.Count;i++)
            {
                if (items[i] == null)
                {
                    return i;
                }
            }
            return -1;
        }
    }
    private void Update()
    {
    }
    private void Start()
    {
        items = new List<Item>(new Item[capacity]);
    }
    public void SetItems(List<Item> items)
    {
        this.items = items;
    }
```

```
    public List<Item> GetItems()
    {
        return new List<Item>(items);
    }
    public void ChangeItem(Item item, int index)
    {

    }
    public bool InsertItem(Item item)
    {
        //Debug.Log("Inventory.InsertItem()");
       // Debug.Log("NextEmptySlotID: "+NextEmptySlotID);
        if (NextEmptySlotID != -1)
        {

            items[NextEmptySlotID] = item;
            //Debug.Log(onItemsChange);
            onItemsChange?.Invoke(items);
            return true;
        }
        else
```

Продолжение приложения Б

```
      {
            return false;
        }

    }
    public void DeleteItem()
    {

    }
    public void ThrowItem()
    {

    }
}
```

Листинг ItemPanel

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.EventSystems;
using UnityEngine.UI;
public class ItemPanel :
MonoBehaviour,IBeginDragHandler,IDragHandler,IEndDragHandler,
IPointerClickHandler,IItemSwap
{
    public static readonly string path = "Prefabs/ItemPanel";
    public delegate void OnItemChange(ItemPanel panel);
    public event OnItemChange onItemChange;
    public bool useFilter = false;
    public  List<Item.ItemType> filterToPass;
    Item _item;
    Vector3 prevPosition;
     public Item Item
    {
        get { return _item; }
        set
        {
            _item = value;
            if (value != null)
            {

                GetComponentInChildren<Text>().text =
_item.itemName;
                GetComponent<Image>().sprite = Item.sprite;

                if (onItemChange != null)
                    onItemChange.Invoke(this);
            }
            else
            {
                GetComponentInChildren<Text>().text = "";
                GetComponent<Image>().sprite = null;
                if (onItemChange != null)
                    onItemChange.Invoke(this);
            }
        }

    }
    public bool ItemSwap(ItemPanel itemPanel)
    {
```

Продолжение приложения Б

```
        if (useFilter)
        {
            bool passed = false;
            if (itemPanel.Item != null)
            {
                foreach (var t in filterToPass)
                {
                    if (t == itemPanel.Item.itemType)
                    {
                        passed = true;
                        break;
                    }
                }
            }
            else
            {
                passed = true;
            }

            if (passed)
            {
                Item copy = itemPanel.Item;
                itemPanel.Item = this.Item;
                this.Item = copy;
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            Item copy = itemPanel.Item;
            itemPanel.Item = this.Item;
            this.Item = copy;

            return true;
        }

    }
```

```
    void Start()
    {
        prevPosition = transform.position;

        if (_item != null)
        {
            GetComponentInChildren<Text>().text =
_item.itemName;
            GetComponent<Image>().sprite = Item.sprite;

            if (onItemChange != null)
                onItemChange.Invoke(this);
        }
        else
        {
            GetComponentInChildren<Text>().text = "";
            GetComponent<Image>().sprite = null;
            if (onItemChange != null)
                onItemChange.Invoke(this);
```

Продолжение приложения Б

```
        }

    }
    public void OnBeginDrag(PointerEventData eventData)
    {
        prevPosition = this.transform.position;
    }
    public void OnDrag(PointerEventData eventData)
    {
        this.transform.position = Input.mousePosition;
    }

    public void OnEndDrag(PointerEventData eventData)
    {

            List<RaycastResult> raycastResults =
RaycastMouse();

            foreach (RaycastResult result in raycastResults)
            {
                if (result.gameObject != this.gameObject &&
result.gameObject.GetComponent<ItemPanel>() != null)
```

```
                {
result.gameObject.GetComponent<IItemSwap>().ItemSwap(this);
                    transform.position = prevPosition;
                    break;
                }

            }

            transform.position = prevPosition;


    }

    public void OnPointerClick(PointerEventData eventData)
    {
    }
    public List<RaycastResult> RaycastMouse()
    {

        PointerEventData pointerData = new
PointerEventData(EventSystem.current)
        {
            pointerId = -1,
        };

        pointerData.position = Input.mousePosition;

        List<RaycastResult> results = new
List<RaycastResult>();
        EventSystem.current.RaycastAll(pointerData, results);



        return results;
    }

}
```

Листинг MovePrediction

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MovePrediction
{
    public MovePrediction(int maxCount)
    {
        this.maxCount = maxCount;
    }
    public int Count { get { return positions.Count; } }
    LinkedList<Vector3> positions = new
LinkedList<Vector3>();
    public int maxCount;
    public void InsertVector(Vector3 vector)
    {
        positions.AddLast(vector);
        if (positions.Count > maxCount)
            positions.RemoveFirst();
    }
    public bool CheckIfPredicted(Vector3 vector)
    {
        float max = GetMaxDeviation();
        //Debug.Log((positions.Last.Value -
vector).magnitude);
        return (positions.Last.Value - vector).magnitude <=
max;
    }
    public Vector3 GetPrediction()
    {
        List<Vector3> v3 = new List<Vector3>();
        LinkedListNode<Vector3> node = positions.First;
        while (node.Next != null)
        {
            v3.Add(node.Next.Value - node.Value);
            node = node.Next;
        }
        if (positions.Count > 1)
        {
```

```
            if (positions.Count > 2)
            {
                Vector3 vectorSumm = new Vector3();
                for(int i = 0; i < v3.Count-1; i++)
                {
                    vectorSumm = v3[i + 1] - v3[i];
                }
                return positions.Last.Value + vectorSumm /
v3.Count;
            }
            else
            {
                return positions.Last.Value + ((v3[0] +
v3[1]) * 2) / (positions.Count - 1);
            }

        }
        else
        {
            return positions.Last.Value;
        }

    }
```

Продолжение приложения Б

```
    float GetMaxDeviation()
    {
        float max = 0;
        foreach(Vector3 v3 in positions)
        {
            if (v3.magnitude > max)
            {
                max = v3.magnitude;
            }
        }
        return max;
    }
}
```

Листинг UIWeaponsSlider:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.UI;
public class UIWeaponSlider : MonoBehaviour
{
    public Character playerCharacter;


    List<RectTransform> rects = new List<RectTransform>();
    List<Vector3> defaultPositions = new List<Vector3>();
    int nextRectID = 0;
    bool expanded = false;
    public void SetWeapons(List<Weapon> weapons)
    {
        foreach (Weapon w in weapons)
        {

rects.Add(Instantiate(Resources.Load<GameObject>(ItemPanel.pa
th), transform).GetComponent<RectTransform>());
        }
    }
    private void Start()
    {
        playerCharacter =
GetComponentInParent<CanvasController>().player.GetComponent<
Character>();
        var panels =  GetComponentsInChildren<ItemPanel>();
        foreach(var panel in panels)
        {
            panel.onItemChange += ChangeItemToBody;
            AddPanel(panel.GetComponent<RectTransform>());
        }
        foreach (var p in rects)
        {
            p.gameObject.SetActive(false);
        }
        rects[nextRectID].gameObject.SetActive(true);
    }
    public void AddPanel(RectTransform trans)
    {
        rects.Add(trans);
        defaultPositions.Add(trans.position);
        foreach(var rect in rects)
        {
            rect.gameObject.SetActive(false);
```

```
        }
```

Продолжение приложения Б

```
        rects[0].gameObject.SetActive(true);

    }
    public void ChangeItemToBody(ItemPanel panel)
    {
        Weapon we = null;
        if(panel.Item!=null &&
panel.Item.GetComponent<Weapon>() != null)
        {
            we = panel.Item.GetComponent<Weapon>();
        }
        playerCharacter.ChangeWeapon(we);
    }
    public void UpdateWeapons(Weapon weapon,RectTransform
rect)
    {

    }
    public void EnableMask(bool enable)
    {
        this.GetComponent<Mask>().enabled = enable;
    }

    public void Next()
    {

        rects[nextRectID].gameObject.SetActive(false);
        nextRectID++;
        if (nextRectID >= rects.Count)
            nextRectID = 0;
        rects[nextRectID].gameObject.SetActive(true);
        if (rects[nextRectID].GetComponent<ItemPanel>().Item
!= null)
        {

playerCharacter.ChangeWeapon(rects[nextRectID].GetComponent<I
temPanel>().Item.GetComponent<Weapon>());

        }
        else
```

```
        {
            playerCharacter.ChangeWeapon(null);
        }
    }

    private void Update()
    {

    }
    public void Expand()
    {
        if (!expanded)
        {
            EnableMask(false);
            Vector3 offset = Vector3.zero;
            for (int i = 0; i < rects.Count; i++)
            {
                rects[i].gameObject.SetActive(true);

                rects[i].transform.localPosition = offset;
                offset.x += rects[i].sizeDelta.x;
            }
            expanded = !expanded;
        }
```

Продолжение приложения Б

```
        else
        {
            EnableMask(true);
            for(int i = 0; i < rects.Count; i++)
            {
                Debug.Log(i);
                rects[i].transform.position =
defaultPositions[i];
                rects[i].gameObject.SetActive(false);
            }
            rects[nextRectID].gameObject.SetActive(true);
            expanded = !expanded;
        }

    }
}
```

Листинг SavePreprocessor

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using BayatGames.SaveGameFree;
using UnityEngine.SceneManagement;
public static class SavePreProcessor
{
    static Dictionary<string, int> prefabStringsCount = new
Dictionary<string, int>();
    static Dictionary<int, ObjectData> idDictionary =new
Dictionary<int, ObjectData>();
    private static int GetPrefabStringsCount(ObjectData
objectData)
    {

        Debug.Log("ID: " + objectData.id);
        if (objectData.id == -1)
        {
            if
(!prefabStringsCount.ContainsKey(objectData.prefabPath))
            {
                prefabStringsCount.Add(objectData.prefabPath,
1);
            }
            else
            {
                prefabStringsCount[objectData.prefabPath]++;
            }

            return prefabStringsCount[objectData.prefabPath];
        }
        else
        {
            return 0;
        }

    }
    private static int GetId(ObjectData objectData)
    {
        if (idDictionary.ContainsKey(objectData.id) &&
idDictionary[objectData.id] == objectData)
```

```
        {
            return objectData.id;
        }

        int currentID = 0;
        while (idDictionary.ContainsKey(currentID))
```

Продолжение приложения Б

```
        {
            currentID++;
        }
        return currentID;


    }
    private static string GetHierarchyString(Transform
transform)
    {
        string s = "";
        Transform t = transform.transform;
        while (t.parent != null)
        {
            t = t.parent.transform;
            s += t.gameObject.name+ "/";
        }
        return s;
    }
    public static void Save(Transform transform, ObjectData
objectData)
    {
        SaveMainData smd =
SaveGame.Load<SaveMainData>(GameSaveLoader.CurrentSavePath);/
/ + saveID);
        int id = GetId(objectData);
        string path = (GameSaveLoader.CurrentSavePath + "+" +
objectData.prefabPath + GetPrefabStringsCount(objectData));
        string hie = GetHierarchyString(transform);
        if (objectData.id == -1)
        {
            smd.paths.Add(path);
            smd.hierarchy.Add(hie);
            objectData.id = id;
        }
```

```
        Debug.Log("prefabStringsCount: " +
prefabStringsCount[objectData.prefabPath]);
        Debug.Log("Path:  " + path);
        Debug.Log("Hierarchy:  " + hie);
        SaveGame.Save(path, objectData);
        SaveGame.Save(GameSaveLoader.CurrentSavePath, smd);
    }
}
```

Листинг SaveMainData

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[SerializeField()]
public class SaveMainData
{
    public List<string> hierarchy =  new List<string>();
    public List<string> paths = new List<string>();

}
```

Продолжение приложения Б

Листинг SaveEditor

```
using BayatGames.SaveGameFree;
using System.Collections;
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;

using UnityEngine.SceneManagement;
public class SaveEditor
{
    [MenuItem("SaveEditor/Destroy All Save Objects")]
```

```
    public static void DestroyAllSaveObjects()
    {

    }
    [MenuItem("SaveEditor/Reload All Save Objects")]
    public static void ReloadAllSaveObjects()
    {

    }
    [MenuItem("SaveEditor/Save All Save Objects")]
    public static void SaveAllSaveObjects()
    {
        var dataWorkers =
GameObject.FindObjectsOfType<ObjectDataWorker>();
        foreach (var dW in dataWorkers)
        {
            SavePreProcessor.Save(dW.gameObject.transform,
dW.objectData);
        }
        Debug.Log(dataWorkers.Length + " Objects Saved");
    }
    [MenuItem("SaveEditor/Create Save")]
    public static void CreateSave()
    {

    }
    [MenuItem("SaveEditor/Clear All")]
    public static void ClearAll()
    {
        SaveGame.Clear();
    }
}
```

Приложение В – Акт внедрения

_____2020г.

Акт внедрения компьютерной игры «Roleplay game»

Настоящий акт свидетельствует, что компьютерная игра «Roleplay game», разработанная Жолдасовом Мадияром, внедрена на сайт itch.io.

Процесс внедрения проходил с 20 мая по _________ 2020г.

Заявленные характеристики игры предполагали наличие следующих основных возможностей:

- наличие боевой системы;

- графический интерфейс.

В ходе опытной эксплуатаций складского учета подтверждено, что приложение обладает всеми заявленными возможностями.

На момент подписания настоящего Акта игра загружена на сайт и используется для демонстрации.

Индивидуальный предприниматель ____________________ Leaf Corcoran