Частное учреждение образования

«Колледж бизнеса и права»

ОТЧЕТ по учебной ПРАКТИКЕ

по РАЗРАБОТКЕ и сопровождению

ОП Т.691015

Выполнил Н.С. Кочмарик

Руководители практики Е.В. Багласова

2019

Содержание

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

ОП Т.691015

Разраб.

*Кочмарик Н.С.*

Провер.

Т. Контр.

Н. Контр.

Утверд.

Отчет по практике

по разработке и сопровождению

Лит.

Листов

КБП

1 Программа практики 3

2 Объектно-ориентированный анализ и проектирование системы 4

2.1 Сущность задачи 4

2.2 Проектирование модели 5

3 Вычислительная система 11

3.1 Требования к аппаратным и операционным ресурсам 11

3.2 Инструменты разработки 11

4 Проектирование задачи 13

4.1 Требования к приложению 13

4.2 Концептуальный прототип 14

4.3 Организация данных 14

4.4 Функции и элементы управления 15

5 Описание программного средства 17

5.1 Общие сведения 17

5.2 Функциональное назначение 17

5.3 Входные и выходные данные 17

6 Программирование на языке С# в среде VisualStudio 18

6.1 Технические требования 18

6.2 Функциональное тестирование 18

7 Применение 20

7.1 Назначение программы 20

7.2 Условия применения 20

Заключение 21

Список используемых источников 22

м

# 1 Программа практики

Целями практики по разработке и сопровождению программного обеспечения являются:

- закрепление знаний, связанных с технологией обработки информации;

- приобретение навыков разработки программ с использованием современных средств обработки экономической и деловой информации;

- выработка умений применять средства стандартных библиотек в практических задачах.

Задачами практики по разработке и сопровождению ПО являются:

- углубленное изучение принципов организации программного обеспечения и технологии его проектирования;

- умение разрабатывать программы в соответствии с промышленными требованиями, обеспечивая высокий уровень качества программного обеспечения и экономической эффективности;

- изучение способов определения экономической себестоимости и эффективности разработки программного обеспечения;

- оформление комплекта документации на созданное программное обеспечение.

##### Таблица 1 - Календарный график работы

|  |  |
| --- | --- |
| Период | Наименование и содержание работ |
| 09.02.19 | Изучение требований к рабочей дисциплине и технике безопасности.  Изучение нормативных документов. |
| 11.02.19-  17.02.19 | Ознакомление с различным программным обеспечением, используемым для разработки программ |
| 18.02.19-  24.02.19 | Получение индивидуальных заданий по созданию программ для обработки экономической и деловой информации. Обсуждение тем индивидуальных заданий. |
| 25.02.19-  03.03.19 | Изучение общих требований к программному обеспечению обработки экономической и деловой информации. Требования к интерфейсу пользователя, выходным документам (печатные, экранные формы, форматы результирующих файлов с результатами расчетов). |
| 04.03.19-  24.03.19 | Анализ предметной области индивидуального задания. Выбор и обоснование языка и среды программирования. Описание функциональных возможностей задачи. |
| 25.03.19-  07.04.19 | Технология создания программного обеспечения. Правила построения диаграмм UML. Проектирование ПО при помощи CASE-средства Rational Rose |
| 08.04.19-  21.04.19 | Конструирование алгоритма и его реализация в выбранной визуальной среде программирования. Разработка интерфейса. Проектирование классов, определение свойств и методов, определение связей между классами. |
| 22.04.19-  05.05.19 | Создание информационной базы задачи. Тестирование и отладка программы с использованием подготовленных тестовых примеров. Проектирование модулей, позволяющих осуществить защиту программного продукта от несанкционированного доступа. |
| 06.05.19-  26.05.19 | Критерии оценки качества программного продукта. Изучение нормативных документов, регламентирующих оформление технического проекта. Создание документации по эксплуатации и сопровождению разработанного ПС. Проведение испытания ПС в нормальных и критических условиях. |
| 27.05.19-  02.06.19 | Анализ возможности модернизации проекта с учетом альтернативных решений. Оформление аннотации на программное средство. |
| 03.06.19-  16.06.19 | Оформление отчета по практике. Подготовка листингов программ с дополнением комментариев. Распечатка экранных форм. |
| 17.06.19-  26.06.19 | Демонстрация программ по индивидуальным заданиям.  Защита отчета по практике. |

# 2 Объектно-ориентированный анализ и проектирование системы

## 2.1 Сущность задачи

Компьютерная ролевая игра (обозначается аббревиатурой CRPG или RPG) — жанр компьютерных игр, основанный на элементах игрового процесса традиционных настольных ролевых игр. В ролевой игре игрок управляет одним или несколькими персонажами, каждый из которых описан набором численных характеристик, списком способностей и умений; примерами таких характеристик могут быть хит-пойнты, показатели силы, ловкости, интеллекта, защиты, уровень развития того или иного навыка В ходе игры они могут меняться. Одним из характерных элементов игрового процесса является повышение возможностей персонажей за счёт улучшения их параметров и изучения новых способностей.

CRPG обычно полагаются на продуманный сюжет и игровой мир. Сюжет обычно делится на серию заданий («квестов»). Игрок отдаёт персонажу команды, и он выполняет их в соответствии с численными показателями, отвечающими за качество выполнения команды. Когда персонаж набирает определённое количество очков опыта, он получает очередной уровень, и эти показатели увеличиваются. CRPG часто предлагают более сложное и динамичное взаимодействие игрока с компьютерными персонажами, чем остальные жанры — это может обеспечиваться как развитым искусственным интеллектом, так и жёстко запрограммированным поведением персонажей.

Признаки жанра и его соотношение с другими жанрами компьютерных игр:

Ролевая система – описывает способы создания, изменения и развития персонажей, повышающие их эффективность в игре. Необходимые элементы:

игрок управляет одним или несколькими уникальными персонажами. это необходимое условие для всех ролевых игр;

игрок постепенно улучшает характеристики и/или навыки (посредством внутриигровых значений, чаще всего очков опыта, получаемых за выполнение заданий, исследование мира, диалоги, сражения и так далее);

персонажи могут улучшать свои характеристики и/или способности и навыки при помощи элементов снаряжения;

игрок может создавать своих персонажей;

игрок должен планировать развитие персонажа. Этот элемент отражает стратегический аспект создания и развития персонажа;

основным способом решения проблем, взаимодействия с игровым миром и преодоления препятствий является тактическое применение навыков/способностей персонажа.

Исследование – описывает способы перемещения персонажа по игровому миру, всё, что он может найти, увидеть, с чем он может взаимодействовать. Необходимые элементы:

персонаж игрока может взаимодействовать с игровым миром и находить новые игровые области;

персонаж игрока может находить предметы и хранить их в инвентаре;

вы можете выбрать свой путь (хотя бы из нескольких вариантов);

персонаж может воздействовать на игровой мир (опускать рычаги, нажимать кнопки, открывать сундуки);

игровой мир воздействует на персонажа (погода, ловушки).

Сюжет – включает в себя все элементы повествования, такие как игровой мир, его историю, персонажей, диалоги, задания, описания, сюжетные линии и способы взаимодействия этих компонентов. Необходимые элементы:

персонаж игрока может получать информацию из информационных источников (намёки, цели, задания, навыки, заклинания, обучение);

сюжет зависит от решений игрока, действий персонажей и характеристик/способностей/навыков;

персонаж может взаимодействовать с источниками информации (например, разговаривать с персонажами);

в процессе взаимодействия персонаж может сделать выбор. Хотя бы некоторые из этих выборов должны иметь последствия.

Боевая система – объясняет влияние ролевой системы, исследований и сюжета на результаты сражения (или, более обобщённо, на результат разрешения конфликта). Необходимые элементы:

эффективность в сражении зависит от характеристик и навыков персонажа (количество повреждений, шанс попадания, возможность использования определённых видов оружия);

в сражениях присутствует элемент случайности.

Black sun – собирает воедино все лучшее из жанра CRPG. Игра представлена в изометрическом виде, что позволяет сосредоточится на геймплее. При разработке вдохновлялись такими играми как Diablo и Path of exile. Black sun перенял множество механик у них, но полным клоном этих игр не стал, Black sun обладает и своими уникальными чертами и механиками.

## 2.2 Проектирование модели

Главной целью проектирования моделей является отображение функциональной структуры объекта, то есть производимые ими действия и связи между этими действиями. Наиболее распространенным средством моделирования данных являются диаграммы «сущность-связь» (ERD), которые предназначены для графического представления моделей данных разрабатываемой программной системы и предлагают некоторый набор стандартных обозначений для определения данных и отношений между ними. С помощью этого вида диаграмм можно описать отдельные компоненты концептуальной модели данных и совокупность взаимосвязей между ними, имеющих важное значение для разрабатываемой системы. Основными понятиями данной нотации являются понятия сущности и связи. При этом под сущностью понимается произвольное множество реальных или абстрактных объектов, каждый из которых обладает одинаковыми свойствами и характеристиками. В этом случае каждый рассматриваемый объект может являться экземпляром одной и только одной сущности, должен иметь уникальное имя или идентификатор, а также отличаться от других экземпляров данной сущности. Связь определяется как отношение или некоторая ассоциация между отдельными сущностями. Примерами связей могут являться родственные отношения типа «отец-сын» или производственные отношения типа «начальник-подчиненный». Другой тип связей задается отношениями «иметь в собственности» или «обладать свойством».

Графическая модель данных строится таким образом, чтобы связи между отдельными сущностями отражали не только семантический характер соответствующего отношения, но и дополнительные аспекты обязательности связей, а также кратность участвующих в данных отношениях экземпляров сущностей. Информационная модель базы представлена на диаграмме «Сущность-связь». Данная диаграмма представлена на рисунке 2.1.

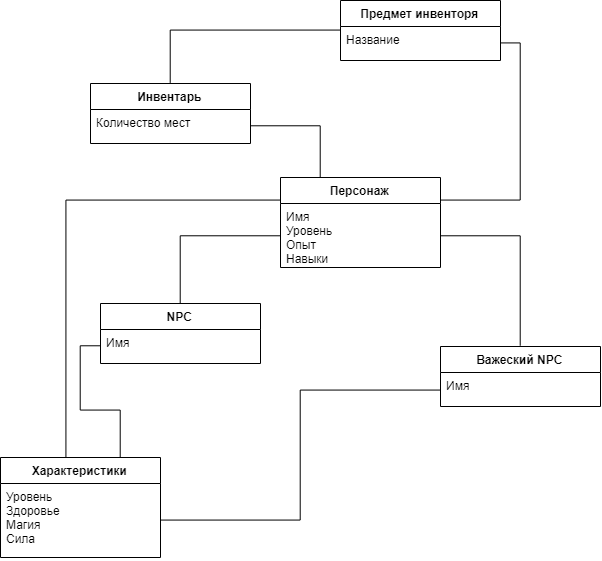


Рисунок 2.1 - Диаграмма «сущность-связь» в нотации Баркера

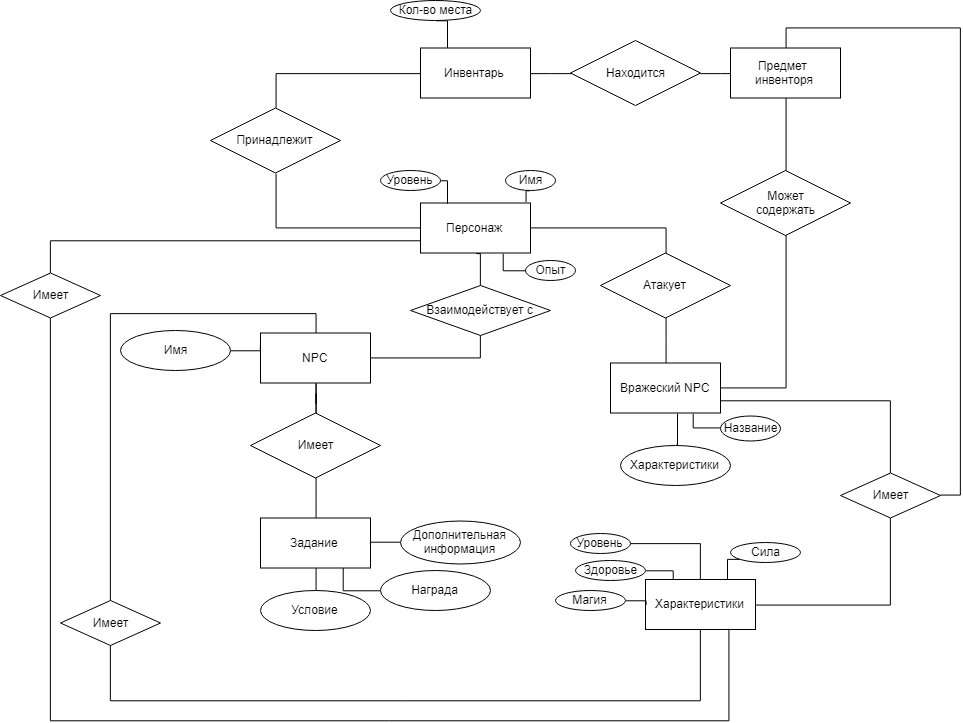


Рисунок 1.2 - Диаграмма ERD в нотации Ричарда Чена

Исходя из исследования предметной области, можно выделить следующие сущности разработки: персонаж, NPC, вражеский NPC, характеристики, инвентарь, предмет инвентаря.

Для сущности «Персонаж» можно выделить следующие атрибуты:

- уровень;

- имя;

- опыт.

Для сущности «NPC» можно выделить следующий атрибут - имя.

Для сущности «Вражеский NPC» можно выделить следующий атрибут: - название.

Для сущности «Характеристики» можно выделить следующие атрибуты:

- уровень;

- здоровье;

- магия;

- сила;

Для сущности «Инвентарь» можно выделить следующий атрибут: - количество мест.

Для сущности «Предмет инвентаре» можно выделить следующий атрибут: - название.

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций – диаграмм. В терминах языка UML определены следующие виды диаграмм: диаграмма вариантов использования, диаграмма классов, диаграмма деятельности, диаграмма последовательности, диаграмма компонентов.

Перечень этих диаграмм представляет собой неотъемлемую часть графической нотации языка UML, сам процесс объектно-ориентированного программирования (ООП) неразрывно связан с процессом построения этих диаграмм. Совокупность построенных таким образом диаграмм содержит всю информацию, необходимую для реализации проекта сложной системы.

Диаграмма – граф специального вида, состоящий из вершин в форме геометрических фигур, которые связаны между собой ребрами или дугами.

Суть диаграммы вариантов использования состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования.

Варианты использования описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности. Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы. Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы.

В данной проектируемой системе в качестве актера выступает игрок.

К основным функциям разрабатываемой программы относятся следующие функции:

* создание игрового персонажа;
* прокачка персонажа;
* использование навыков;
* передвижение;
* кастомизация персонажа;
* выполнение заданий;
* взаимодействие с NPC;
* генерация элементов карты.

К вспомогательным функциям программы относятся:

* предоставить информацию об игроке;
* предоставить информацию о навыках;
* предоставить информацию об игровом мире.

Диаграмма вариантов использования представлена на листе 1 графической части.

Центральное место в ООП занимает разработка логической модели системы в виде диаграммы классов. Нотация классов в языке UML проста и интуитивно понятна. Схожая нотация применяется и для объектов — экземпляров класса, с тем различием, что к имени класса добавляется имя объекта, и вся надпись подчеркивается.

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами кото­рого являются элементы типа «классификатор» и которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. Поэтому диаграмму классов принято считать графическим представлением таких структурных взаимосвязей логической модели системы, которые не зависят от времени.

Класс в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линями на разделы или секции. Здесь указывается: имя класса, атрибуты и операции (методы).

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельный класс может обозначаться простым прямоугольником с указанием только имени соответствующего класса. По мере проработки отдельных компонентов диаграммы классов дополняются атрибутами и операциями. Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Состав некоторых основных классов проекта представлен в таблице 1.1. Диаграмма классов представлена на листе 2 графической части.

Таблица 1.1 – Состав некоторых основных классов проекта

|  |  |  |
| --- | --- | --- |
| Имя класса | Имя поля | Методы |
| Spell | Name; Coldown. | Use. |
| BlackSunPlayerController | PointToMove; blsMoveToPoint; Spels. | SetupInputComponent; Interact. |
| NPC | Name; Points, IsMoveTopoint. | MoveToPoints. |
| EnemyNPC | Health; Level; Damage. | Patrol; MakeHit. |
| QuestNPC | Quests. | StartQuest; EndQuest; MakeRevard. |
| Quest | Name; Description; QuestType. |  |
| BlackSunPlayer | \_Inventory; \_ActionBa;, Health, Mana; Exp. | TopDawnController; AddToInventory; DropToActionBar; OnUpdateInventory; OnUpdateActionBar; SetupPlayerInpucomponent. |
| InventoryItem | Mesh; Name; Discription; InventoryIcon; ItemQuality; QualityEffect; WidgetComp. | BeginPlay; Oninteract; Show, Tick. |
| ConsumableInventoryItem | Quantity. | Oninteract; method. |
| StatsItem | Stats. | OnInteract; SetStats. |
| WeaponInventoryItem | Stats. | OnInteract; SetStats. |
| ArmorInventoryItem | Stats. | OnInteract; SetStats. |

При моделировании поведения проектируемой или анализируемой системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, переход в следующее состояние срабатывает только при завершении этой операции. Графически диаграмма деятельности представляется в форме графа, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому.

Диаграммы деятельности - частный случай диаграмм состояний. Основная цель использования таких диаграмм - визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. Диаграмма деятельности представлена на листе 3 графической части.

Для моделирования взаимодействия объектов в UML используется диаграмма последовательности. Диаграмма последовательности представлена на листе 4 графической части.

Все предыдущие диаграммы отражали концептуальные аспекты построения модели системы и относились к логическому уровню представления и оперировали понятиями, которые не имеют самостоятельного материального воплощения. Для описания реальных сущностей предназначен другой аспект модельного представления, а именно физическое представление модели. Для физического представления модели используются диаграмма компонентов.

Диаграмма компонентов описывает объекты реального мира – компоненты программного обеспечения. Эта диаграмма позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код.

Диаграмма компонентов представлена на листе 5 графической части и содержит следующие компоненты:

* BlackSun.sln;
* BlackSun.exe;
* BalckSun.Target.cs;
* BlackSunEditor.Target.cs;
* BlackSun.Build.cs;
* BlackSun.cpp;
* BlackSunCharacter.cpp;
* BlackSunCharacter.h;
* BlackSunGameMode.cpp;
* BlackSunPlayerController.cpp;
* BlackSunPlayerController.h;
* WeaponInventory.cpp;
* StatsManager.cpp;
* InventoryItem.cpp;
* ConsumableInventoryItem.cpp;
* WeaponInventoryItem.h;
* StateManager.h;
* InventoryItem.h;
* ConsumapleInventoryItem.h;
* BlackSunGameMode.h;
* WeaponInventory.h;
* BlackSun.h;

# 3 Вычислительная система

## 3.1 Требования к аппаратным и операционным ресурсам

Основными минимальными требованиями, выдвигаемыми к аппаратному обеспечению персонального компьютера, являются:

* процессор Intel Core i3-2365M CPU @ 1.40 ГГц и выше;
* оперативная память 4 Гбайт и более;
* свободное место на диске 100 Мбайт;
* видеокарта NVIDIA GT720M на 2 Гбайт и более;
* монитор;
* мышь, клавиатура;

Компьютер должен работать под управлением операционной системы, начиная с Windows 10, так как она ориентирована на максимальное использование всех возможностей ПК, сетевых ресурсов и обеспечение комфортных условий работы.

## 3.2 Инструменты разработки

Инструментами разработки для будущего программного приложения будут являться:

* операционная система Windows 10;
* программная среда разработки Microsoft Visual Studio 2017;
* язык программирования C++;
* язык программирования Blueprints;
* редактор трехмерной графики Autodesk Maya 2019;
* редактор трехмерной графики Autodesk 3D Max 2018;
* редактор трехмерной графики Pixologic ZBrush 2018;
* редактор Adobe Photoshop 2018;
* игровой движок Unreal Engine 4;
* программа для создания текстур и материалов Substance Painter;
* программа для создания текстур и материалов Substance Designer;
* приложение для построения диаграмм Draw.io.

Операционная система - это набор управляющих программ, предназначенных для управления ресурсами вычислительной системы как единого комплекса, другими словами операционная система – это набор программного обеспечения, который обеспечивает работу компьютера. Основными функциями операционной системы являются:

* управление файловой системой (запись, изменение, копирование файлов, контроль доступа);
* управление выполнением программ (распределение процессорного времени, загрузка программ с диска в оперативную память, перехват потенциально опасных действий);
* управление памятью (кэширование, распределение, контроль сохранности данных);
* диалог с пользователем (чтение команд с клавиатуры, с мыши, вывод информации на экран, на принтер);

Microsoft Visual Studio 2017 - бесплатная, полнофункциональная и расширяемая интегрированная среда разработки для создания современных приложений для Android, iOS и Windows, а также веб-приложений и облачных служб. Основные особенности:

* мощные инструменты программирования – c уверенностью программируйте на своем языке, находите и быстро устраняйте ошибки в коде;
* усовершенствованная отладка – локальная или удаленная отладка для разных языков. Диагностика проблем производительности непосредственно из рабочего процесса отладчика;
* веб-инструменты - разработка веб-приложений с помощью ASP.NET, Node.js, Python и JavaScript. Используйте мощные веб-платформы, такие как AngularJS, jQuery, Bootstrap, Django и Backbone.js;
* несколько языков - поддержка нескольких языков, включая C#, Visual Basic, F#, C++, JavaScript, TypeScript, Python и множество других. Visual Studio помогает писать, отлаживать и тестировать код независимо от того, какой язык выбран;

Язык программирования C++ – компилируемый, статически типизированный язык программирования общего назначения, поддерживает ООП. Язык имеет богатую стандартную библиотеку, которая включает в себя распространенные контейнеры и алгоритмы. С++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. С++ широко используется для разработки ПО, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, высокопроизводительных серверов, а также игр.

Blueprints – скриптовая система в Unreal Engine 4, которая представляет собой визуальный интерфейс для создания элементов геймплея. Система очень гибкая и очень мощная, и позволяет дизайнерам использовать концепцию и почти полный потенциал программирования.

Autodesk Maya 2019 – программа для 3D-анимации, моделирования и визуализации. Maya представляет мощный интегрированный инструментарий, который можно использовать для создания анимации, сред, графики движения, виртуальной реальности и персонажей. Широко применяется в кинематографии и игровой индустрии.

Autodesk 3D Max 2018 – Профессиональное ПО для 3D-моделирования, анимации и визуализации при создании игр и проектировании.

Adobe Photoshop 2018 – многофункциональный графический редактор. В основном работает с растровыми изображениями, однако имеет инструменты для работы с векторной графикой. Используется в различных областях от создания художественного арта, обработки фотографий до создания спецэффектов и анимации.

Unreal Engine 4 – игровой движок, разрабатываемый и поддерживаемый компанией Epic Games. Написан языке C++, движок позволяет создавать игры для большинства операционных систем и платформа, а также на различные портативные устройства. Для упрощения портирования движок использует модульную систему зависимых компонентов, воспроизведения звука, средства голосового воспроизведение текста, распознавание речи, модули для работы с сетью и поддержки различных устройств ввода.

Substance Painter – профессиональная программа для текстурирования трехмерных объектов, которая используется разработчиками многих крупных AAA-игр. Программа умеет работать с текстурами в сверхвысоком разрешении и позволяет создавать по-настоящему фотореалистичные объекты при наличии готовой модели с правильной геометрией.

Substance Designer – является мощным нодовым инструментом текстурирования для PBR рендера (Physically Based Rendering), который позволяет создавать текстуры. Используется для текстурирования моделей и для запекания различной информации о модели. Программа сдержит обширную библиотеку инструментов, материалов и процедурных эффектов.

Draw.io – приложение для моделирования диаграмм и блок-схем. В числе возможностей – построение диаграмм, графиков и UML-моделей. В библиотеке форм присутствует несколько десятков фигур, сгруппированных по категориям. Объекты можно форматировать, меняя цвет, градиент, толщину линий, непрозрачность. Благодаря возможности синхронизации с Google Диском над документации могут работать несколько пользователей.

# 4 Описание приложения

## 4.1 Требования к приложению

Данное приложение не требует никаких специальных средств защиты, либо ограничений прав доступа к данным.

Разрабатываемое приложение должно иметь понятный и удобный в использовании интерфейс, чтобы взаимодействие между программой и пользователем было максимально упрощено. Для того чтобы интерфейс был понятен пользователю, необходимо оформить диалог приложения с пользователем, обладающий кнопочным меню, позволяющим перемещаться по записям. Программное приложение должно обеспечивать оперативный поиск необходимой информации по предоставленным критериям, для этого будут разработаны процедуры отбора информации. Кроме всего вышеперечисленного, разрабатываемое приложение позволит контролировать и вести статистику работ.

Все входные данные должны проверяться на ошибки. При совершении пользователем неверных действий или введении неверных данных будут выводиться сообщения с пояснением ошибок.

Функциональные требования представлены на диаграмме SADT, которая отображает вначале всю систему в виде простейшей компоненты – одного блока и дуг, изображающих интерфейсы с функциями вне системы. Единственный блок представляет всю систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг – они также представляют полный набор внешних интерфейсов системы в целом. Диаграмма нулевого уровня представлена на рисунке 4.1.

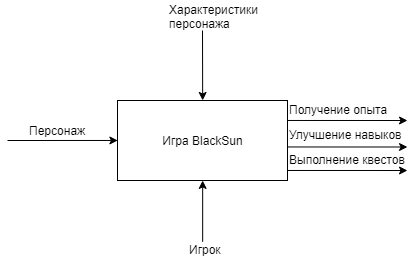


Рисунок 4.1 – Функциональная SADT диаграмма нулевого уровня

Далее блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки представляют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых представлена как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом для более детального представления. Диаграмма первого уровня детализации представлена на

рисунке 4.2.



Рисунок 4.2 – Функциональная SADT диаграмма первого уровня детализации

## 4.2 Концептуальный прототип

Концептуальный прототип представляет собой описание внешнего пользовательского интерфейса – систему меню и форм.

Разрабатываемое приложение должно иметь понятный и удобный в использовании интерфейс, чтобы взаимодействие между игрой и пользователем было интуитивно понятно. Через главное меню можно будет получить доступ к самой игре, настройкам, позволяющим настроить приложение (настроить графику, звук), возможность выйти из игры. Макет главного меню на рисунке 4.1:

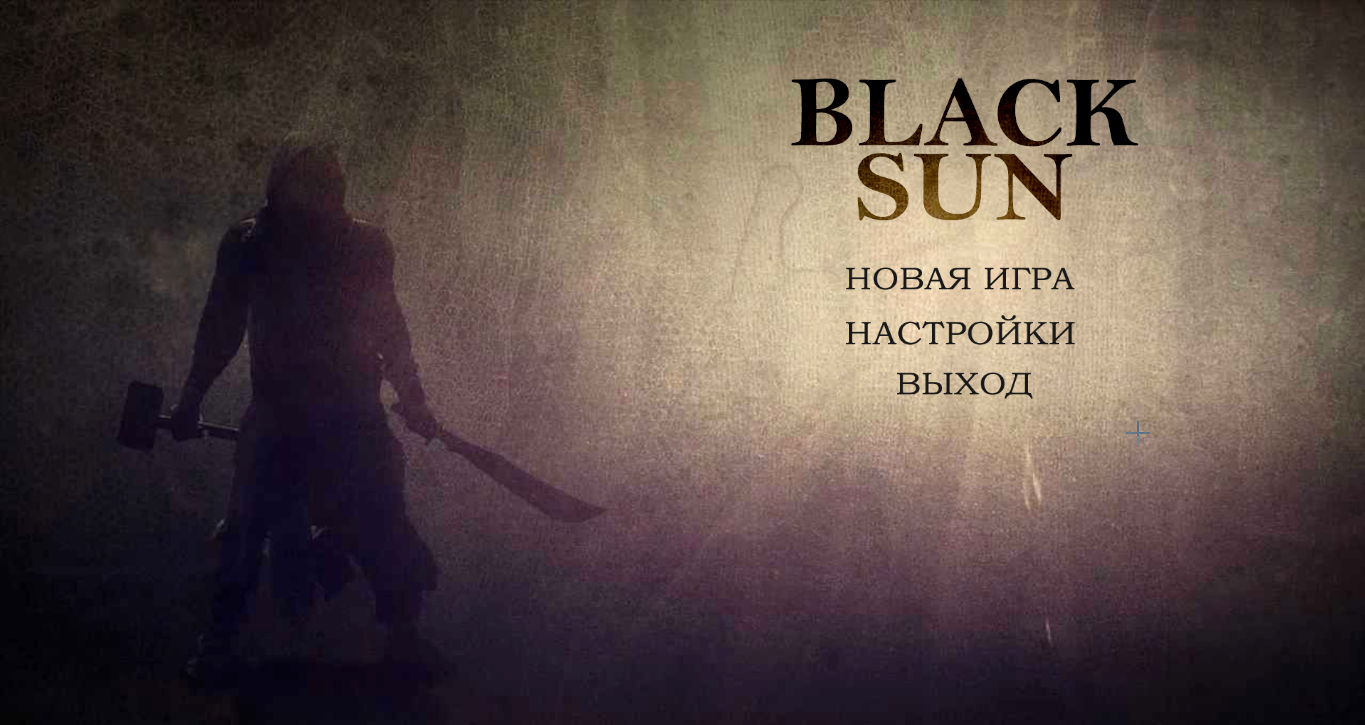


Рисунок 4.1

## 4.3 Описание разделов приложения

Главный экран игры будет состоять из фонового изображения, логотипа игры и пунктов меню:

* новая игра – служит для начала новой игры;
* настройки – пункт меню в котором можно будет изменить настройки приложения (настроить графику, звук);
* выход – служит для выхода из приложения.

Доступ к главному экрану также будет доступен из самой игры. В игре будут присутствовать кнопки, позволяющие вызвать окна инвентаря, карты, заданий, персонажа.

4.4 Функции и элементы управления

На основании диаграммы вариантов использования в программном приложении «Black Sun» были реализованы следующие функции:

* движение;
* добавление элементов в инвентарь;
* удаление элементов из инвенторя;
* взаимодействие с объектами.

Рассмотрим основные функции программы.

За движение отвечает метод Move, код которого представлен ниже

void ABlackSunCharacter::Move (float Value){

if ((Controller != NULL) && (Value != 0.0f)){

// find out which way is forward

const FRotator Rotation = Controller->GetControlRotation();

const FRotator YawRotation(0, Rotation.Yaw, 0);

// get forward vector

const FVector Direction = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);

AddMovementInput(Direction, Value);}}

За добавление элементов в инвентарь отвечает метод ABlackSunCharacter, код которого представлен ниже

void ABlackSunCharacter::AddToInventory(AInventoryItem \* Item){

for (int i = 0; i < \_Inventory.Num(); ++i){

if (\_Inventory[i] == nullptr){

Item->Show(false);

\_Inventory[i] = Item;

OnUpdateInventory.Broadcast(\_Inventory);

Item->OnPicked.Broadcast(Item);

break;}}}

За удаление объектов из инвентаря отвечает метод , код которого представлен ниже

void ABlackSunCharacter::RemoveFromInventory(AInventoryItem\* Item, INT index){

if (index != NULL){

DropItem(\_Inventory[index]);

\_Inventory[index] = nullptr;}

else for(int i = 0; i < \_Inventory.Num(); ++i)

if (\_Inventory[i] == Item){

DropItem(Item);

\_Inventory[i] = nullptr;

break;}

OnUpdateInventory.Broadcast(\_Inventory);}

За взаимодействие с объектами отвечает метод ABlackSunPlayerController, код которого представлен ниже

void ABlackSunPlayerController::Interact(){

FHitResult TraceHitResult;

this->GetHitResultUnderCursor(ECC\_Visibility, true, TraceHitResult);

if (ABlackSunCharacter\* MyPawn = Cast<ABlackSunCharacter>(GetPawn())) {

if (AInventoryItem\* item = Cast<AInventoryItem>(TraceHitResult.GetActor())) {

if (MyPawn->GetDistanceTo(TraceHitResult.GetActor()) <= 160.f){

item->OnInteract();MyPawn->AddToInventory(item);}

else{

MyPawn->isInputPress = false;

MoveToLocation(item->GetTargetLocation());}}}}

Весь программный код представлен в приложении А.

# 5 Описание программного средства

## 5.1 Общие сведения

Игра Black Sun создана в жанре CRPG и служит для увлекательного времяпрепровождения. Для функционирования программы не требуется никаких специальных программ. Инсталляция программного средства осуществляется по средствам установочника, нужно лишь запустить его и следовать инструкциям.

## 5.2 Функциональное назначение

У данной игры нет специальных средств защиты или классов решаемых задач. Данное программное средство служит для хорошего времяпрепровождения. Сетевая поддержка у игры отсутствует.

## 5.3 Входные и выходные данные

Выходными данными будут являться действия, которые происходят в ответ на действия игрока, также выходными данными будут являться статистика об игроке, информация об окружении, противниках, NPC, информация о способностях.

# 6 Методика испытаний

## 6.1 Технические требования

Минимальные системные требования к приложению:

* до 10 Гбайт свободного места на винчестере;
* 4 Гбайт свободной оперативной памяти;
* процессор Intel Core i3 последнего поколения и выше;
* видеокарта с объемом памяти не менее 2 Гбайт;
* операционная система Windows 10;
* клавиатура, мышь.

## 6.2 Функциональное тестирование

В процессе написания программного продукта необходимо производить тестирование на правильность работы приложения. Одной из основных задач тестирования является устранение ошибок, происходящих при вводе данных.

Функциональное тестирование – это тестирование функций приложения на соответствие требованиям. Оценка производится в соответствии с ожидаемыми и полученными результатами (на основании функциональной спецификации), при условии, что функции отрабатывали на различных значениях.

Тестирование программы будет производиться последовательно, переходя из одной части программы в другую. Во время теста будут проверяться все действия с программой, навигация пунктам меню, которые может произвести пользователь. После чего, все собранные и найденные ошибки будут исправлены.

Таблица 6.1– Тест-кейсы для проведения функционального тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № тест-  кейса | Модуль/  Функция | Шаги воспроизведения | Результат |
| 001 | Начать игру | 1. Запустить приложение   2. Нажать кнопку «Новая игра» | Ожидаемый результат:  загрузка первого уровня. |
| Фактический результат: загрузка первого уровня. Результаты тестирование предоставлены в приложении Б на рисунках: Б.1, Б.2, Б.3. |
| 002 | Подобрать предмет | 1. Запустить приложение 2. Начать игру 3. Найти подбираемый предмет 4. Подобрать предмет | Ожидаемый результат: предмет появляется в инвентаре. |
| Фактический результат: предмет появляется в инвентаре. Результаты тестирование предоставлены в приложении Б на рисунках: Б.4, Б.5. |
| 003 | Исчезновение препятствий перед игроком | 1. Запустить приложение 2. Начать игру 3. Встать перед препятствием | Ожидаемый результат: препятствие исчезнет если встать перед препятствием. |
| Фактические результат: препятствие исчезнет если встать перед препятствием. Результаты тестирование предоставлены в приложении Б на рисунках: Б.6, Б.7. |
| 004 | Выйти из игры | 1. Запустить приложение 2. Нажать на кнопку «Выход» | Ожидаемый результат: выход из игры. |
| Фактический результат: выход из игры. |

# 7 Применение

## 7.1 Назначение программы

Игра Black Sun создана в жанре CRPG и служит для увлекательного времяпрепровождения. Для функционирования программы не требуется никаких специальных программ. Инсталляция программного средства осуществляется по средствам установочника, нужно лишь запустить его и следовать инструкциям. У данной игры нет специальных средств защиты или классов решаемых задач. Данное программное средство служит для хорошего времяпрепровождения. Сетевая поддержка у игры отсутствует.

## 7.2 Условия применения

Для применения данного программного средства необходимы следующие технически требования:

* процессор Intel Core i3-2365M или выше;
* минимальный объем оперативной памяти — 2 Гбайт;
* операционная система Windows 10;
* устройство для чтения дисков;
* клавиатура;
* мышь.

# Заключение

За время практики по созданию и сопровождению программного обеспечения был изучены процессы создания, внедрения и сопровождения программных приложений в различных средах программирования. Во время практики удалось накопить неоценимый опыт в разработке приложений на движке Unreal Engine 4. Кроме того, изучить принципы организации и технологии реализации программного обеспечения, развить умения разработки программ в соответствии с требованиями технического задания, обеспечивая высокий уровень качества программного обеспечения и экономической эффективности. Получить навыки оформления комплекта документации на созданное программное обеспечение.

При создании программных приложений, входящих в обязательную программу практики, были отработаны навыки владения методами надежного программирования и эффективности разработки программного обеспечения.

# Список используемых источников

1 Видеохостинг [Электронный ресурс] / Режим доступа: <http://youtube.com>

2 Справочные материалы по Unreal Engine4 [Электронный ресурс] / Режим доступа : <http://docs.unrealengine.com>

# Приложение А (обязательное) Текст программы

Файл BalckSun.Target.cs

public class BlackSunTarget : TargetRules{

public BlackSunTarget(TargetInfo Target) : base(Target){

Type = TargetType.Game;

ExtraModuleNames.Add("BlackSun");}}

Файл BlackSunEditor.Target.cs

public class BlackSunEditorTarget : TargetRules{

public BlackSunEditorTarget(TargetInfo Target) : base(Target){

Type = TargetType.Editor;

ExtraModuleNames.Add("BlackSun");}}

Файл ConsumableInventory.cpp

AConsumableInventoryItem::AConsumableInventoryItem(){

Quantity = 1;}

Файл InventoryItem.cpp

#include "InventoryItem.h"

#include "Components/BoxComponent.h"

#include "Components/StaticMeshComponent.h"

#include "Components/WidgetComponent.h"

#include "Engine.h"

#include "BlackSunCharacter.h"

//class ABlackSunCharacter;

// Sets default values

AInventoryItem::AInventoryItem(){

// Set this actor to call Tick() every frame. You can turn this off to improve performance if you don't need it.

PrimaryActorTick.bCanEverTick = true;

BoxComponent = CreateDefaultSubobject<UBoxComponent>(TEXT("BoxComponent"));

BoxComponent->SetCollisionEnabled(ECollisionEnabled::QueryOnly);

BoxComponent->SetCollisionResponseToAllChannels(ECollisionResponse::ECR\_Block);

BoxComponent->SetCollisionResponseToChannel(ECollisionChannel::ECC\_Pawn, ECollisionResponse::ECR\_Ignore);

RootComponent = BoxComponent;

BoxComponent->SetBoxExtent(FVector(10,25,5));

WidgetBoom = CreateDefaultSubobject<USpringArmComponent>(TEXT("WidgetBoom"));

WidgetBoom->SetupAttachment(RootComponent);

WidgetBoom->bAbsoluteRotation = true; // Don't want arm to rotate when character does

WidgetBoom->TargetArmLength = 80;

WidgetBoom->RelativeRotation = FRotator(180.f, 0.f, 0.f);

WidgetBoom->bDoCollisionTest = false; // Don't want to pull camera in when it collides with level

Mesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MeshComp"));

Mesh->SetCollisionEnabled(ECollisionEnabled::QueryOnly);

Mesh->SetupAttachment(BoxComponent);

Mesh->SetRelativeRotation(FRotator(0, 0, 90));

Mesh->SetRelativeLocation(FVector(0, -50, 0));

QualityEffect = CreateDefaultSubobject<UParticleSystemComponent>(TEXT("EffectComp"));

QualityEffect->SetupAttachment(BoxComponent);

WidgetComp = CreateDefaultSubobject<UWidgetComponent>(TEXT("WidgetComp"));

WidgetComp->SetupAttachment(WidgetBoom, USpringArmComponent::SocketName);

static ConstructorHelpers::FObjectFinder<UClass> WidgetClass(TEXT("Class'/Game/UI/WBP\_DropItemName.WBP\_DropItemName\_C'"));

if (WidgetClass.Succeeded()){WidgetComp->SetWidgetClass(WidgetClass.Object);}

WidgetComp->SetWidgetSpace(EWidgetSpace::Screen);

WidgetComp->SetDrawAtDesiredSize(true);

WidgetSpaceComponent = CreateDefaultSubobject<UBoxComponent>(TEXT("WidgetSpaceComponent"));

WidgetSpaceComponent->SetCollisionEnabled(ECollisionEnabled::QueryOnly);

WidgetSpaceComponent->SetCollisionResponseToAllChannels(ECollisionResponse::ECR\_Block);

WidgetSpaceComponent->SetCollisionResponseToChannel(ECollisionChannel::ECC\_Pawn, ECollisionResponse::ECR\_Ignore);

WidgetSpaceComponent->SetCollisionResponseToChannel(ECollisionChannel::ECC\_Visibility, ECollisionResponse::ECR\_Ignore);

WidgetSpaceComponent->SetupAttachment(WidgetBoom, USpringArmComponent::SocketName);

WidgetSpaceComponent->SetBoxExtent(FVector(40, WidgetComp->GetDrawSize().Y/10, 5));

WidgetSpaceComponent->SetRelativeLocation(FVector(40,0,0));

isOnHoverd = false;}

void AInventoryItem::OnInteract(){

GEngine->AddOnScreenDebugMessage(1, 5, FColor::Red, TEXT("Base class"));}

void AInventoryItem::Show(bool visible){

ECollisionEnabled::Type collision = visible ?

ECollisionEnabled::QueryOnly :

ECollisionEnabled::NoCollision;

SetActorTickEnabled(visible);

Mesh->SetVisibility(visible);

Mesh->SetCollisionEnabled(collision);

BoxComponent->SetVisibility(visible);

BoxComponent->SetCollisionEnabled(collision);

QualityEffect->SetVisibility(visible);

WidgetComp->SetVisibility(visible);

WidgetSpaceComponent->SetVisibility(visible);

if (visible){

WidgetSpaceComponent->SetCollisionResponseToChannel(ECollisionChannel::ECC\_Visibility, ECollisionResponse::ECR\_Block);}

else{

WidgetSpaceComponent->SetCollisionResponseToChannel(ECollisionChannel::ECC\_Visibility, ECollisionResponse::ECR\_Ignore);}}

// Called when the game starts or when spawned

void AInventoryItem::BeginPlay(){

Super::BeginPlay();

WidgetBoom->SetVisibility(false);

WidgetComp->SetVisibility(false);

WidgetSpaceComponent->SetVisibility(false);

UObject\* Effect = nullptr;

switch (ItemQuality){

case Quality::Common: {

Effect = StaticLoadObject(UObject::StaticClass(), nullptr, TEXT("ParticleSystem'/Game/Effects/LayItemEffect/LightItemEffect\_Common.LightItemEffect\_Common'"));;break;}

case Quality::Rare: {

Effect = StaticLoadObject(UObject::StaticClass(), nullptr, TEXT("ParticleSystem'/Game/Effects/LayItemEffect/LightItemEffect\_Rare.LightItemEffect\_Rare'"));;break;}

case Quality::Mythical: {

Effect = StaticLoadObject(UObject::StaticClass(), nullptr, TEXT("ParticleSystem'/Game/Effects/LayItemEffect/LightItemEffect\_Mythical.LightItemEffect\_Mythical'"));;break;}

case Quality::Legendary: {

Effect = StaticLoadObject(UObject::StaticClass(), nullptr, TEXT("ParticleSystem'/Game/Effects/LayItemEffect/LightItemEffect\_Legendary.LightItemEffect\_Legendary'"));;

break;}}

if (UParticleSystem\* PS = Cast<UParticleSystem>(Effect)){QualityEffect->SetTemplate(PS);} }

// Called every frame

void AInventoryItem::Tick(float DeltaTime){

Super::Tick(DeltaTime);

if (WidgetComp->IsVisible() && !isOnHoverd){

GEngine->AddOnScreenDebugMessage(4, 0.1, FColor::Red, TEXT("hide"));

WidgetBoom->SetVisibility(false);

WidgetComp->SetVisibility(false);

WidgetSpaceComponent->SetVisibility(false);

WidgetSpaceComponent->SetCollisionResponseToChannel(ECollisionChannel::ECC\_Visibility, ECollisionResponse::ECR\_Ignore);}

isOnHoverd = false;}

void AInventoryItem::ShowItem(){

GEngine->AddOnScreenDebugMessage(4, 0.1, FColor::Red, TEXT("Show"));

isOnHoverd = true;

WidgetBoom->SetVisibility(true);

WidgetComp->SetVisibility(true);

WidgetSpaceComponent->SetVisibility(true);

WidgetSpaceComponent->SetCollisionResponseToChannel(ECollisionChannel::ECC\_Visibility, ECollisionResponse::ECR\_Block);

WidgetSpaceComponent->SetBoxExtent(FVector(WidgetBoom->TargetArmLength / 2, 30, 5));

WidgetSpaceComponent->SetRelativeLocation(FVector(WidgetBoom->TargetArmLength / 2, 0, 0));}

Файл StatsManager.cpp

#include "StatsManager.h"

void StatsManager::AddArmor(TMap<FName, INT>& Stats){Stats.Add("Armor", 5);}

Файл WeaponInventoryItem.cpp

#include "WeaponInventoryItem.h"

#include "StatsManager.h"

AWeaponInventoryItem::AWeaponInventoryItem(){Stats.Add("Armor", 10);}

void AWeaponInventoryItem::OnInteract(){

GEngine->AddOnScreenDebugMessage(1, 5, FColor::Red, TEXT("Recive class"));}

Файл ConsumableInventoryItem.h

#include "CoreMinimal.h"

#include "InventoryItem.h"

#include "ConsumableInventoryItem.generated.h"

UCLASS()

class BLACKSUN\_API AConsumableInventoryItem : public AInventoryItem{GENERATED\_BODY()

public: AConsumableInventoryItem();

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Item")

INT Quantity;

virtual void OnInteract();};

Файл InventoryItem.h

#include "CoreMinimal.h"

#include "GameFramework/Actor.h"

#include "InventoryItem.generated.h"

class UBoxComponent;

class UParticleSystemComponent;

class UWidgetComponent;

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_OneParam(FPickupItemDelegate, const AActor\*, Actor);

UENUM(BlueprintType)

enum class Quality : uint8{Common,Rare,Mythical,Legendary};

UCLASS()

class BLACKSUN\_API AInventoryItem : public AActor{GENERATED\_BODY()

public:

// Sets default values for this actor's properties

AInventoryItem();

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Item")

FName Name;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Item")

FText Discription;

protected:

UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = "Item")

UStaticMeshComponent\* Mesh;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "WidgetSpace", meta = (AllowPrivateAccess = "true"))

class USpringArmComponent\* WidgetBoom;

UPROPERTY(EditDefaultsOnly, Category = "Collision")

UBoxComponent\* BoxComponent;

UPROPERTY(EditDefaultsOnly, Category = "Collision")

UBoxComponent\* WidgetSpaceComponent;

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Item")

UTexture2D\* InventoryIcon;

UPROPERTY(EditAnywhere, Category = "Item")

Quality ItemQuality;

UPROPERTY()

UParticleSystemComponent\* QualityEffect;

UPROPERTY(EditAnywhere)

class UWidgetComponent\* WidgetComp;

// Called when the game starts or when spawned

virtual void BeginPlay() override;

public: UFUNCTION(BlueprintCallable)

virtual void OnInteract();

void ShowItem();

virtual void Show(bool visible);

// Called every frame

virtual void Tick(float DeltaTime) override;

UPROPERTY(BlueprintAssignable, Category = "Pickup")

FPickupItemDelegate OnPicked;

private:bool isOnHoverd; };

Файл StatsManager.h

#include "CoreMinimal.h"

class BLACKSUN\_API StatsManager{

public: StatsManager();

~StatsManager();

static void AddArmor(TMap<FName,INT> &Stats);};

Файл WeaponInventoryItem.h

#include "CoreMinimal.h"

#include "InventoryItem.h"

#include "WeaponInventoryItem.generated.h"

UCLASS()

class BLACKSUN\_API AWeaponInventoryItem : public AInventoryItem{

GENERATED\_BODY()

public: AWeaponInventoryItem();

UFUNCTION(BlueprintCallable)

TMap<FName, INT> GetStats()const { return Stats; }

private: UPROPERTY()

TMap<FName, INT> Stats;

public: virtual void BeginPlay() override;

virtual void OnInteract(); };

Файл BlackSun.Build.cs

using UnrealBuildTool;

public class BlackSun : ModuleRules{

public BlackSun(ReadOnlyTargetRules Target) : base(Target){

PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore", "HeadMountedDisplay", "NavigationSystem", "AIModule", "UMG" });}}

Файл VlackSun.cpp

#include "BlackSun.h"

#include "Modules/ModuleManager.h"

IMPLEMENT\_PRIMARY\_GAME\_MODULE( FDefaultGameModuleImpl, BlackSun, "BlackSun" );

DEFINE\_LOG\_CATEGORY(LogBlackSun)

Файл BlackSun.h

#include "CoreMinimal.h"

DECLARE\_LOG\_CATEGORY\_EXTERN(LogBlackSun, Log, All);

Файл BlackSunCharacter.cpp

#include "BlackSunCharacter.h"

#include "UObject/ConstructorHelpers.h"

#include "InventoryItem.h"

#include "Camera/CameraComponent.h"

#include "Components/DecalComponent.h"

#include "Components/CapsuleComponent.h"

#include "GameFramework/CharacterMovementComponent.h"

#include "GameFramework/PlayerController.h"

#include "GameFramework/SpringArmComponent.h"

#include "HeadMountedDisplayFunctionLibrary.h"

#include "Materials/Material.h"

#include "Engine/World.h"

#include "Components/SkeletalMeshComponent.h"

#include "Engine.h"

ABlackSunCharacter::ABlackSunCharacter(){

// Set size for player capsule

GetCapsuleComponent()->InitCapsuleSize(42.f, 96.0f);

BaseTurnRate = 45.f;

isRotating = false;

// Don't rotate character to camera direction

bUseControllerRotationPitch = false;

bUseControllerRotationYaw = false;

bUseControllerRotationRoll = false;

// Configure character movement

GetCharacterMovement()->bOrientRotationToMovement = true; // Rotate character to moving direction

GetCharacterMovement()->RotationRate = FRotator(0.f, 640.f, 0.f);

GetCharacterMovement()->bConstrainToPlane = true;

GetCharacterMovement()->bSnapToPlaneAtStart = true;

// Create a camera boom...

CameraBoom = CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"));

CameraBoom->SetupAttachment(RootComponent);

CameraBoom->bAbsoluteRotation = true; // Don't want arm to rotate when character does

CameraBoom->TargetArmLength = 800.f;

CameraBoom->RelativeRotation = FRotator(-60.f, 0.f, 0.f);

CameraBoom->bDoCollisionTest = false; // Don't want to pull camera in when it collides with level

// Create a camera...

TopDownCameraComponent = CreateDefaultSubobject<UCameraComponent>(TEXT("TopDownCamera"));

TopDownCameraComponent->SetupAttachment(CameraBoom, USpringArmComponent::SocketName);

TopDownCameraComponent->bUsePawnControlRotation = false; // Camera does not rotate relative to arm

// Create a decal in the world to show the cursor's location

CursorToWorld = CreateDefaultSubobject<UDecalComponent>("CursorToWorld");

CursorToWorld->SetupAttachment(RootComponent);

static ConstructorHelpers::FObjectFinder<UMaterial> DecalMaterialAsset(TEXT("Material'/Game/TopDownCPP/Blueprints/M\_Cursor\_Decal.M\_Cursor\_Decal'"));

if (DecalMaterialAsset.Succeeded()){CursorToWorld->SetDecalMaterial(DecalMaterialAsset.Object);}

CursorToWorld->DecalSize = FVector(16.0f, 32.0f, 32.0f);

CursorToWorld->SetRelativeRotation(FRotator(90.0f, 0.0f, 0.0f).Quaternion());

// Activate ticking in order to update the cursor every frame.

PrimaryActorTick.bCanEverTick = true;

PrimaryActorTick.bStartWithTickEnabled = true;

HeadMeshComponent = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("HeadComponent"));

HeadMeshComponent->SetCollisionEnabled(ECollisionEnabled::NoCollision);

HeadMeshComponent->SetupAttachment(GetMesh(),MeshComponentName);

TopMeshComponent = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("TopComponent"));

TopMeshComponent->SetCollisionEnabled(ECollisionEnabled::NoCollision);

TopMeshComponent->SetupAttachment(GetMesh(), MeshComponentName);

ArmsMeshComponent = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("ArmsComponent"));

ArmsMeshComponent->SetCollisionEnabled(ECollisionEnabled::NoCollision);

ArmsMeshComponent->SetupAttachment(GetMesh(), MeshComponentName);

PantsMeshComponent = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("PantsComponent"));

PantsMeshComponent->SetCollisionEnabled(ECollisionEnabled::NoCollision);

PantsMeshComponent->SetupAttachment(GetMesh(), MeshComponentName);

BootsMeshComponent = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("BootsComponent"));

BootsMeshComponent->SetCollisionEnabled(ECollisionEnabled::NoCollision);

BootsMeshComponent->SetupAttachment(GetMesh(), MeshComponentName);

MaxHealth = 100; Health = 100;

MaxMana = 100; Mana = 100;

ExpToNextLevel = 100; Exp = 0;

level = 1; inventoryMaxSize = 30;

\_Inventory.SetNum(inventoryMaxSize);

ActionBarMaxSize = 5;

\_ActionBar.SetNum(ActionBarMaxSize);

BoxInteractComponent = CreateDefaultSubobject<UBoxComponent>(TEXT("BoxInteract"));

BoxInteractComponent->SetCollisionEnabled(ECollisionEnabled::QueryOnly);

BoxInteractComponent->SetCollisionResponseToAllChannels(ECollisionResponse::ECR\_Overlap);

BoxInteractComponent->SetupAttachment(CameraBoom, USpringArmComponent::SocketName);

BoxInteractComponent->SetBoxExtent(FVector(CameraBoom->TargetArmLength / 2, 1, 1));

BoxInteractComponent->SetRelativeLocation(FVector(CameraBoom->TargetArmLength / 2, 0, 0));

BoxInteractComponent->OnComponentBeginOverlap.AddDynamic(this,&ABlackSunCharacter::HideOverlapObjects);

BoxInteractComponent->OnComponentEndOverlap.AddDynamic(this, &ABlackSunCharacter::ShowOverlapObjects);

CameraAngle = -45;}

void ABlackSunCharacter::Tick(float DeltaSeconds){

Super::Tick(DeltaSeconds);

HeathPercent = (float)Health / (float)MaxHealth;

ManaPercent = (float)Mana / (float)MaxMana;

ExpPercent = (float)Exp / (float)ExpToNextLevel;

if (CursorToWorld != nullptr){

if (UHeadMountedDisplayFunctionLibrary::IsHeadMountedDisplayEnabled()){

if (UWorld\* World = GetWorld()){

FHitResult HitResult;

FCollisionQueryParams Params(NAME\_Actor, FCollisionQueryParams::GetUnknownStatId());

FVector StartLocation = TopDownCameraComponent->GetComponentLocation();

FVector EndLocation = TopDownCameraComponent->GetComponentRotation().Vector() \* 2000.0f;

Params.AddIgnoredActor(this);

World->LineTraceSingleByChannel(HitResult, StartLocation, EndLocation, ECC\_Visibility, Params);

FQuat SurfaceRotation = HitResult.ImpactNormal.ToOrientationRotator().Quaternion();

CursorToWorld->SetWorldLocationAndRotation(HitResult.Location, SurfaceRotation);}}

else if (APlayerController\* PC = Cast<APlayerController>(GetController())){

FHitResult TraceHitResult;

PC->GetHitResultUnderCursor(ECC\_Visibility, true, TraceHitResult);

FVector CursorFV = TraceHitResult.ImpactNormal;

FRotator CursorR = CursorFV.Rotation();

CursorToWorld->SetWorldLocation(TraceHitResult.Location);

CursorToWorld->SetWorldRotation(CursorR);

if (AInventoryItem\* item = Cast<AInventoryItem>(TraceHitResult.GetActor())) {

item->ShowItem();}}}

if (Controller != NULL){

APlayerController\* const PC = CastChecked<APlayerController>(Controller);

FRotator rotation = PC->GetControlRotation();

rotation.Pitch = CameraAngle;

PC->SetControlRotation(rotation);}}

void ABlackSunCharacter::HideOverlapObjects(UPrimitiveComponent \* OverlappedComponent, AActor \* OtherActor, UPrimitiveComponent \* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult & SweepResult){

OtherComp->SetVisibility(false);}

void ABlackSunCharacter::ShowOverlapObjects(UPrimitiveComponent \* OverlappedComponent, AActor \* OtherActor, UPrimitiveComponent \* OtherComp, int32 OtherBodyIndex){OtherComp->SetVisibility(true);}

void ABlackSunCharacter::SetupPlayerInputComponent(UInputComponent \* PlayerInputComponent){

check(PlayerInputComponent);

//InputComponent->BindAxis("TopDownController", this, &ABlackSunCharacter::TopDownController);

//PlayerInputComponent->BindAction("ChangeControllerSettings", IE\_Pressed, this, &ABlackSunCharacter::Change);

PlayerInputComponent->BindAxis("MoveForward", this, &ABlackSunCharacter::MoveForward);

PlayerInputComponent->BindAxis("MoveRight", this, &ABlackSunCharacter::MoveRight);

PlayerInputComponent->BindAxis("Turn", this, &ABlackSunCharacter::Turn);

PlayerInputComponent->BindAxis("TurnRate", this, &ABlackSunCharacter::TurnAtRate);

PlayerInputComponent->BindAction("Rotate",IE\_Pressed, this, &ABlackSunCharacter::StartRotation);

PlayerInputComponent->BindAction("Rotate",IE\_Released,this, &ABlackSunCharacter::EndRotation);}

void ABlackSunCharacter::MoveForward(float Value){

if ((Controller != NULL) && (Value != 0.0f)){

// find out which way is forward

const FRotator Rotation = Controller->GetControlRotation();

const FRotator YawRotation(0, Rotation.Yaw, 0);

// get forward vector

const FVector Direction = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);

AddMovementInput(Direction, Value);}}

void ABlackSunCharacter::MoveRight(float Value){

if ((Controller != NULL) && (Value != 0.0f)){

// find out which way is right

const FRotator Rotation = Controller->GetControlRotation();

const FRotator YawRotation(0, Rotation.Yaw, 0);

// get right vector

const FVector Direction = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);

// add movement in that direction

AddMovementInput(Direction, Value);}}

void ABlackSunCharacter::TurnAtRate(float Rate){

// calculate delta for this frame from the rate information

AddControllerYawInput(Rate \* BaseTurnRate \* GetWorld()->GetDeltaSeconds());}

void ABlackSunCharacter::Turn(float Rate){

if (Rate != 0.f && Controller && Controller->IsLocalPlayerController() && isRotating){

APlayerController\* const PC = CastChecked<APlayerController>(Controller);

PC->AddYawInput(Rate);}}

void ABlackSunCharacter::StartRotation(){ isRotating = true;}

void ABlackSunCharacter::EndRotation(){isRotating = false;}

TArray<AInventoryItem\*> ABlackSunCharacter::GetInventory(){

UE\_LOG(LogTemp, Warning, TEXT("Size %i"), \_Inventory.Num());

return \_Inventory;}

void ABlackSunCharacter::AddToInventory(AInventoryItem \* Item){

for (int i = 0; i < \_Inventory.Num(); ++i){

if (\_Inventory[i] == nullptr){

Item->Show(false);

\_Inventory[i] = Item;

OnUpdateInventory.Broadcast(\_Inventory);

Item->OnPicked.Broadcast(Item);

break;}}}

void ABlackSunCharacter::DropToActionBar(AInventoryItem\* Item,INT SlotIndex){

if (SlotIndex > ActionBarMaxSize)return;

\_ActionBar[SlotIndex] = Item;

OnUpdateActionBar.Broadcast(\_ActionBar);}

void ABlackSunCharacter::DropItem(AInventoryItem \* Item){

Item->SetActorLocation(FVector(GetActorLocation().X, GetActorLocation().Y, GetActorLocation().Z - 80));

Item->Show(true);}

void ABlackSunCharacter::UpdateInventory(){

GEngine->AddOnScreenDebugMessage(4, 3, FColor::Purple, "\_Inventory: " + FString::FromInt(\_Inventory.Num()));

GEngine->AddOnScreenDebugMessage(3, 3, FColor::Purple, "\_ActionBar: " + FString::FromInt(\_ActionBar.Num()));}

void ABlackSunCharacter::RemoveFromInventory(AInventoryItem\* Item, INT index){

if (index != NULL){

DropItem(\_Inventory[index]);

\_Inventory[index] = nullptr;}

else for(int i = 0; i < \_Inventory.Num(); ++i)

if (\_Inventory[i] == Item){

DropItem(Item);\_Inventory[i] = nullptr;break;}

OnUpdateInventory.Broadcast(\_Inventory);}

void ABlackSunCharacter::RemoveFromActionBar(AInventoryItem \* Item){\_ActionBar.RemoveSingle(Item);}

void ABlackSunCharacter::SwapInActionBar(AInventoryItem \* ItemFrom, AInventoryItem \* ToFrom){

\_ActionBar[\_ActionBar.IndexOfByKey(ItemFrom)] = ToFrom;}

void ABlackSunCharacter::SwapInInventory(INT indexFrom, INT indexTo){

AInventoryItem\* temp = \_Inventory[indexFrom];

\_Inventory[indexFrom] = \_Inventory[indexTo];

\_Inventory[indexTo] = temp;

OnUpdateInventory.Broadcast(\_Inventory);}

Файл BlackSunCharacter.h

#include "CoreMinimal.h"

#include "GameFramework/Character.h"

#include "InventoryItem.h"

#include "BlackSunCharacter.generated.h"

class USkeletalMeshComponent;

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_OneParam(FUpdateInventoryDelegate, const TArray<AInventoryItem\*>&, InventoryItems);

UCLASS(Blueprintable)

class ABlackSunCharacter : public ACharacter{GENERATED\_BODY()

public:ABlackSunCharacter();

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera)

float BaseTurnRate;

// Called every frame.

virtual void Tick(float DeltaSeconds) override;

FORCEINLINE class UCameraComponent\* GetTopDownCameraComponent() const { return TopDownCameraComponent; }

FORCEINLINE class USpringArmComponent\* GetCameraBoom() const { return CameraBoom; }

FORCEINLINE class UDecalComponent\* GetCursorToWorld() { return CursorToWorld; }

private: UPROPERTY(EditDefaultsOnly, Category = "Collision")

UBoxComponent\* BoxInteractComponent;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

class UCameraComponent\* TopDownCameraComponent;

/\*\* Camera boom positioning the camera above the character \*/

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

class USpringArmComponent\* CameraBoom;

/\*\* A decal that projects to the cursor location. \*/

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

class UDecalComponent\* CursorToWorld;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

USkeletalMeshComponent\* HeadMeshComponent;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

USkeletalMeshComponent\* TopMeshComponent;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

USkeletalMeshComponent\* ArmsMeshComponent;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

USkeletalMeshComponent\* PantsMeshComponent;

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

USkeletalMeshComponent\* BootsMeshComponent;

protected: UFUNCTION()

void HideOverlapObjects(UPrimitiveComponent\* OverlappedComponent, AActor\* OtherActor, UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult &SweepResult);

UFUNCTION()

void ShowOverlapObjects(UPrimitiveComponent\* OverlappedComponent, AActor\* OtherActor, UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex);

virtual void SetupPlayerInputComponent(UInputComponent\* InputComponent);

void MoveForward(float value); void MoveRight(float value);

void TurnAtRate(float Rate); void Turn(float Rate);

void StartRotation(); void EndRotation();

private: TArray<AInventoryItem\*> \_Inventory;

TArray<AInventoryItem\*> \_ActionBar;

bool isRotating;

public: UPROPERTY(EditDefaultsOnly, Category = Camera, meta = (AllowPrivateAccess = "true"))

float CameraAngle;

UFUNCTION(BlueprintCallable)

TArray<AInventoryItem\*> GetInventory();

UPROPERTY(EditDefaultsOnly, Category = Inventory)

INT inventoryMaxSize;

UPROPERTY(EditDefaultsOnly, Category = ActionBar)

INT ActionBarMaxSize;

void AddToInventory(AInventoryItem\* Item);

UFUNCTION(BlueprintCallable)

void DropToActionBar(AInventoryItem\* Item, INT SlotIndex);

void DropItem(AInventoryItem\* Item);

UFUNCTION(BlueprintCallable)

void UpdateInventory();

UFUNCTION(BlueprintCallable)

void RemoveFromInventory(AInventoryItem \* Item, INT index);

UFUNCTION(BlueprintCallable)

void RemoveFromActionBar(AInventoryItem\* Item);

UFUNCTION(BlueprintCallable)

void SwapInActionBar(AInventoryItem\* ItemFrom, AInventoryItem\* ToFrom);

UFUNCTION(BlueprintCallable)

void SwapInInventory(INT indexFrom, INT indexTo);

UPROPERTY(BlueprintAssignable, Category = "Pickup")

FUpdateInventoryDelegate OnUpdateInventory;

UPROPERTY(BlueprintAssignable, Category = "Pickup")

FUpdateInventoryDelegate OnUpdateActionBar;

public: bool isInputPress;

private: int MaxHealth; int Health;

int MaxMana; int Mana;

int ExpToNextLevel; int Exp;

int level;

public: UPROPERTY(BlueprintReadOnly)

float HeathPercent;

UPROPERTY(BlueprintReadOnly)

float ManaPercent;

UPROPERTY(BlueprintReadOnly)

float ExpPercent;

void LevelUp();

private: void MakeDamage(AActor\* DamagedActor, float Damage, const class UDamageType\* DamageType, class AController\* InstigatedBy, AActor\* DamageCauser);};

Файл VlackSunGameMode.cpp

#include "BlackSunGameMode.h"

#include "BlackSunPlayerController.h"

#include "BlackSunCharacter.h"

#include "UObject/ConstructorHelpers.h"

ABlackSunGameMode::ABlackSunGameMode(){

// use our custom PlayerController class

PlayerControllerClass = ABlackSunPlayerController::StaticClass();

// set default pawn class to our Blueprinted character

static ConstructorHelpers::FClassFinder<APawn> PlayerPawnBPClass(TEXT("/Game/TopDownCPP/Blueprints/TopDownCharacter"));

if (PlayerPawnBPClass.Class != NULL){ DefaultPawnClass = PlayerPawnBPClass.Class;}}

Файл BlackSunGameMode.h

#include "CoreMinimal.h"

#include "GameFramework/GameModeBase.h"

#include "BlackSunGameMode.generated.h"

UCLASS(minimalapi)

class ABlackSunGameMode : public AGameModeBase{

GENERATED\_BODY()

public: ABlackSunGameMode();};

Файл VlackSunPlayerController.cpp

#include "BlackSunPlayerController.h"

#include "Blueprint/AIBlueprintHelperLibrary.h"

#include "Runtime/Engine/Classes/Components/DecalComponent.h"

#include "HeadMountedDisplayFunctionLibrary.h"

#include "BlackSunCharacter.h"

#include "Engine/World.h"

ABlackSunPlayerController::ABlackSunPlayerController(){

bShowMouseCursor = true; DefaultMouseCursor = EMouseCursor::Crosshairs;}

void ABlackSunPlayerController::PlayerTick(float DeltaTime){

Super::PlayerTick(DeltaTime);

if (bIsmoveToPoint){

if (ABlackSunCharacter\* MyPawn = Cast<ABlackSunCharacter>(GetPawn())) {

FVector Direction =PointToMove - MyPawn->GetActorLocation();

Direction.Normalize();

FRotator Rotator = FRotationMatrix::MakeFromX(Direction).Rotator();

Rotator.Pitch = 0;

Rotator.Roll = 0;

MyPawn->SetActorRotation(Rotator);

MyPawn->AddMovementInput(MyPawn->GetActorForwardVector(), 1);

if (FVector::Dist(PointToMove,MyPawn->GetActorLocation()) <= 160.f){

bIsmoveToPoint = false;}

if (MyPawn->isInputPress)bIsmoveToPoint = false;}}}

void ABlackSunPlayerController::MoveToLocation(FVector point){

bIsmoveToPoint = true; PointToMove = point;}

void ABlackSunPlayerController::SetupInputComponent(){

// set up gameplay key bindings

Super::SetupInputComponent();

InputComponent->BindAction("Interact", IE\_Pressed, this, &ABlackSunPlayerController::Interact);

//InputComponent->BindAction("SetDestination", IE\_Pressed, this, &ABlackSunPlayerController::OnSetDestinationPressed);

//InputComponent->BindAction("SetDestination", IE\_Released, this, &ABlackSunPlayerController::OnSetDestinationReleased);

//// support touch devices

//InputComponent->BindTouch(EInputEvent::IE\_Pressed, this, &ABlackSunPlayerController::MoveToTouchLocation);

//InputComponent->BindTouch(EInputEvent::IE\_Repeat, this, &ABlackSunPlayerController::MoveToTouchLocation);

//InputComponent->BindAction("ResetVR", IE\_Pressed, this, &ABlackSunPlayerController::OnResetVR);}

void ABlackSunPlayerController::Interact(){

FHitResult TraceHitResult;

this->GetHitResultUnderCursor(ECC\_Visibility, true, TraceHitResult);

if (ABlackSunCharacter\* MyPawn = Cast<ABlackSunCharacter>(GetPawn())) {

if (AInventoryItem\* item = Cast<AInventoryItem>(TraceHitResult.GetActor())) {

if (MyPawn->GetDistanceTo(TraceHitResult.GetActor()) <= 160.f){

item->OnInteract(); MyPawn->AddToInventory(item);}

else { MyPawn->isInputPress = false; MoveToLocation(item->GetTargetLocation());}}}}

Файл BlackSunPlayerController.h

#include "CoreMinimal.h"

#include "GameFramework/PlayerController.h"

#include "BlackSunPlayerController.generated.h"

UCLASS()

class ABlackSunPlayerController : public APlayerController{

GENERATED\_BODY()

public: ABlackSunPlayerController();

protected:

/\*\* True if the controlled character should navigate to the mouse cursor. \*/

uint32 bMoveToMouseCursor : 1;

// Begin PlayerController interface

virtual void PlayerTick(float DeltaTime) override;

//virtual void SetupInputComponent() override;

void MoveToLocation(FVector point);

virtual void SetupInputComponent(); void Interact();

private: bool bIsmoveToPoint;

FVector PointToMove;};

# Приложение Б Результаты тестирования



Рисунок Б.1 – Главное меню

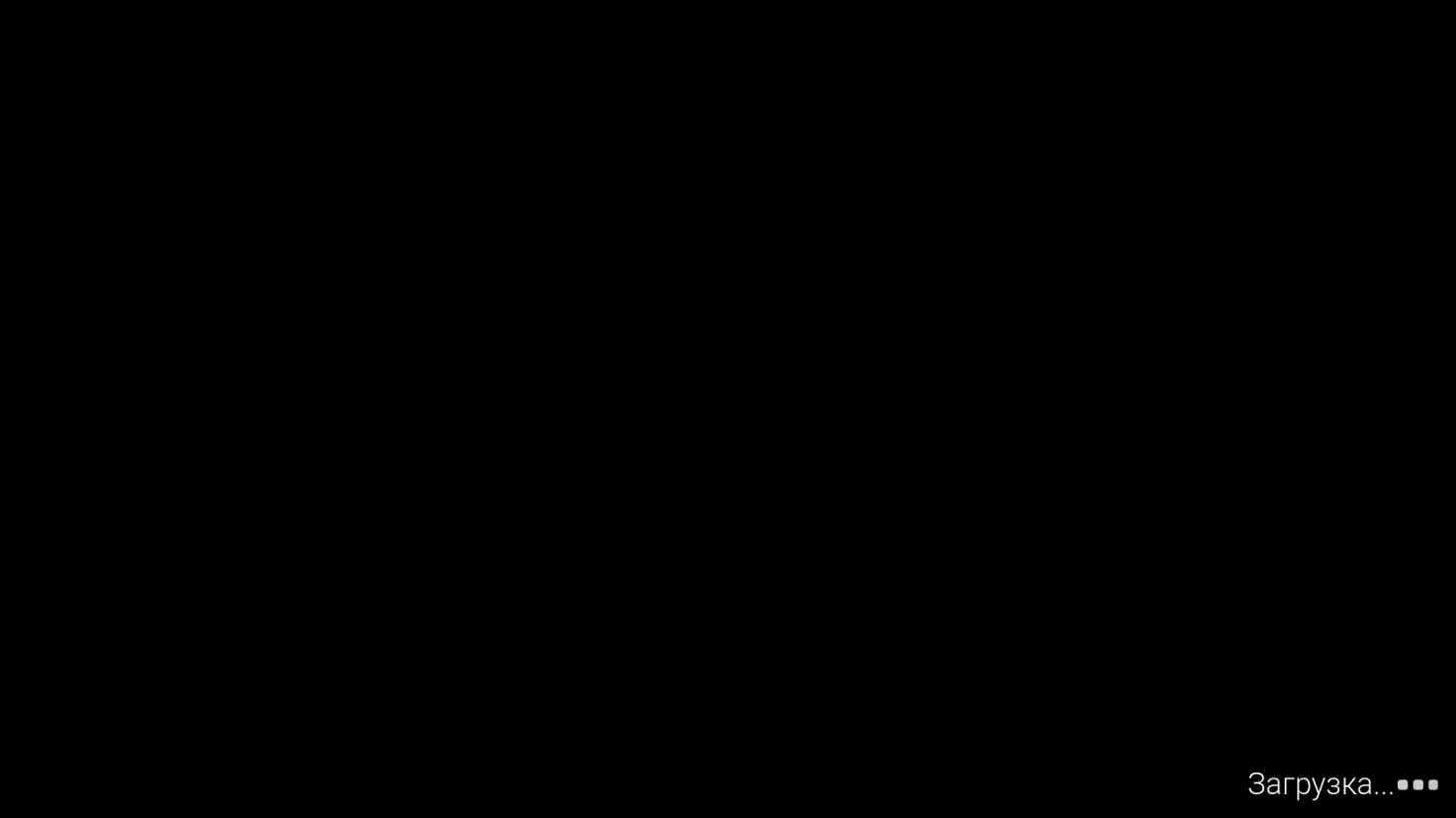


Рисунок Б.2 – Загрузочный экран



Рисунок Б.3 – Загруженная игра



Рисунок Б.4 – Предмет на земле

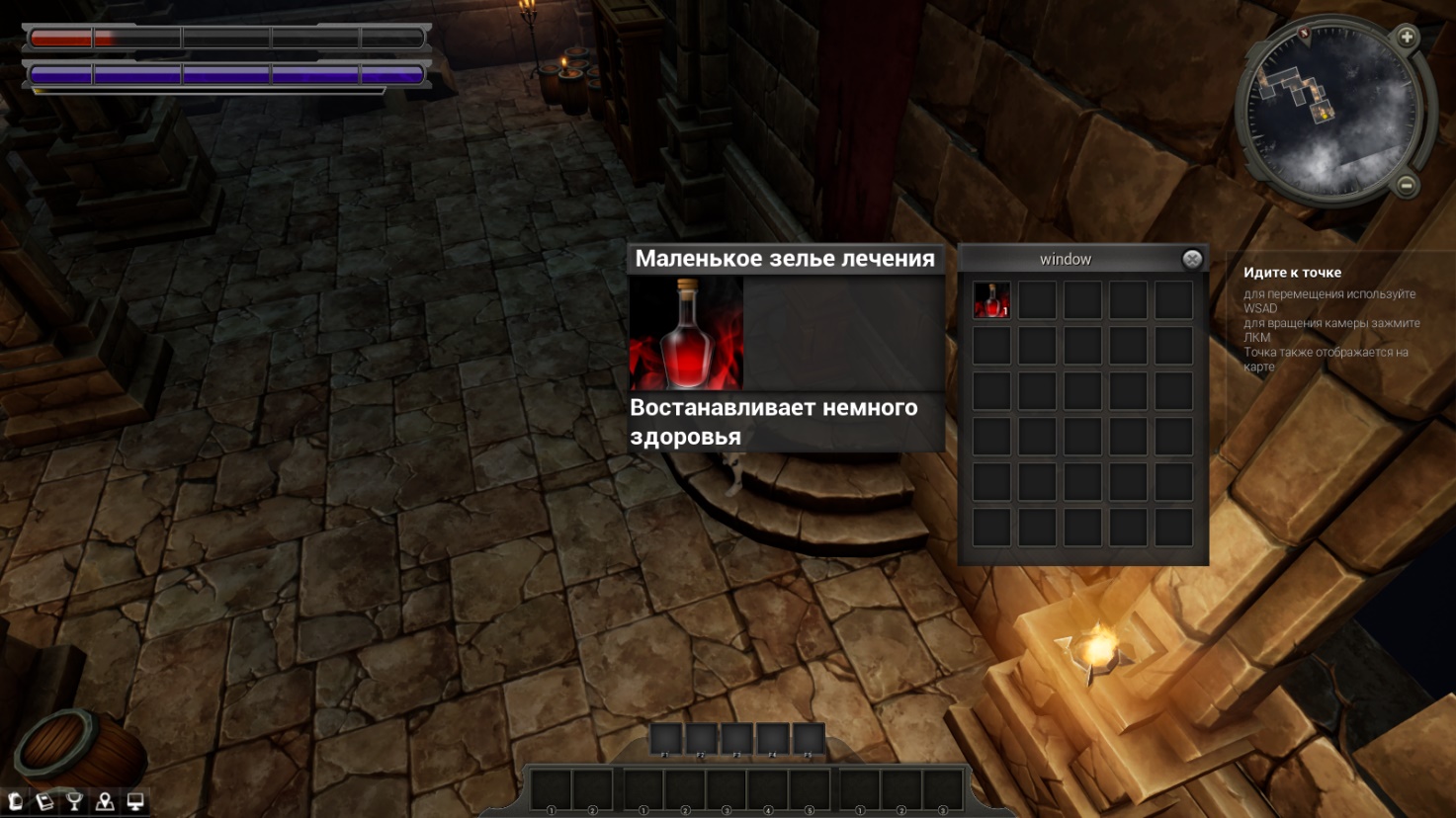


Рисунок Б.5 – Предмет в инвентаре



Рисунок Б.6 – Персонаж рядом с колонной



Рисунок Б.7 - Персонаж за колонной