

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
«Вычислительная техника»
(наименование)

Кафедра
«Системы автоматизированного проектирования»
(наименование)

Направление подготовки

09.03.01 «Информатика и вычислительная техника»
(код и наименование)

Профиль

«Системы автоматизированного проектирования»
(наименование)

БАКАЛАВРСКАЯ РАБОТА

на тему

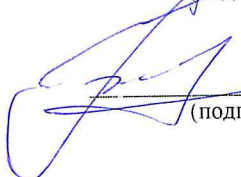
Разработка инвентаря игрового персонажа и системы
лояльности при взаимодействии с неигровыми персонажами игры

Студент

 23.05.17
(подпись, дата)

Шабалова К.Ю.
(фамилия, инициалы)

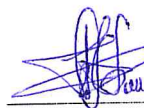
Руководитель

 23.05.17
(подпись, дата)

Ришард Н.А.
(фамилия, инициалы)

Нормоконтролёр

информ. каф. САПР ПГУ
(должность, место работы)

 26.05.17
(подпись, дата)

Федоткин С.С.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой


(подпись)

Березинский В.М.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

(фамилия, инициалы)

Пенза 2017

Реферат

Пояснительная записка содержит 67 листов, 16 рисунков, 13 источников, 7 приложение.

UNITY 3D, ИГРОВОЙ ИНВЕНТАРЬ, ВНУТРИИГРОВОЙ ЦЕНОВОЙ БАЛАНС, МОДЕЛИРОВАНИЕ 3D ОБЪЕКТОВ, КОМПЬЮТЕРНАЯ ИГРА.

Объектами разработки является система инвентаря, система лояльности, а так же модели статических объектов.

Целью ВКР являлось создание ряда элементов, необходимых для дальнейшей разработки игры:

Система инвентаря - представляет собой вызывающееся по нажатию клавиши хранилище игровых предметов, позволяющее добавлять, удалять перемещать, разделять предметы. В разрабатываемой системе, в отличии от рассмотренных аналогов, учтен новый критерий - редкость предмета, который учитывается при разработке системы лояльности игрока.

Система лояльности - предназначена для адаптации игрового мира в зависимости от действий игрового персонажа и учитывает такие критерии как расчет цены предмета, включающий значимость игрового персонажа для неигрового персонажа, учет фракции, к которой тот относится и т.д. Дополнительно производится расчет ценности противника в единицах игровой валюты, которая используется при расчете баланса игровых предметов и персонажей.

Статические объекты - 3Д модели для создания игрового окружения.

					ВКР 090301.18 17 81 01		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Шабалова К.Ю.		1			
Провер.		Финогеев А. А.					
Н. Контр		Бождай А. С.		26.05.17			
Утверд.		Бершадский А. М.					
Разработка инвентаря игрового персонажа и системы лояльности при взаимодействии с неигровыми персонажами игры. Пояснительная записка					Лит.	Лист	Листов
					ФВТ, каф. САПР, группа 13ВВ2		

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	10
1.1 Сравнение с аналогами	10
1.1.1 Система хранения игровых предметов персонажа	10
1.1.2 Обзор аналогов системы лояльности	14
1.2 Выводы по главе	15
1.2.1 Итоги анализа по системе инвентаря	15
1.2.2 Итоги анализа по системе лояльности	16
2 ПРОЕКТИРОВАНИЕ	18
2.1 Проектирование инвентаря	18
2.2 Проектирование системы лояльности	18
2.3 Определение визуальной направленности моделей	25
2.3.1 Моделирование	25
2.3.2 Текстурирование объекта	26
3 РАЗРАБОТКА СИСТЕМЫ	28
3.1 Принципы взаимодействия пользователя с системой	28
3.2 Описание ключевых классов	28
3.3 Описание алгоритмов	31
3.4 Используемое программное обеспечение	34
3.4.1 Microsoft visual studio 2017	34
3.4.2 Unity 3D 5.6.0	34
3.5 Контрольный пример	35
ЗАКЛЮЧЕНИЕ	41
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ	42

ПРИЛОЖЕНИЕ А1	44
ПРИЛОЖЕНИЕ Б1	45
ПРИЛОЖЕНИЕ В1	46
ПРИЛОЖЕНИЕ В2	58
ПРИЛОЖЕНИЕ В3	61
ПРИЛОЖЕНИЕ В4	66
ПРИЛОЖЕНИЕ В5	67

ВВЕДЕНИЕ

Тема выпускной квалификационной работы посвящена разработке системы хранения предметов игрового персонажа, а также системы лояльности, которая представляется как алгоритм расчета цены для приобретения персонажем предмета в игровом аналоге магазина. Так же дополнительно были созданы статические объекты окружения.

Данная ВКР является частью одного игрового проекта жанра RPG. Данный жанр довольно популярен за счет своей ролевой системы, описывающей способы создания, изменения и развития персонажей, повышающие их эффективность в игре. 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, а также применения того или иного снаряжения. Таким образом игры жанра RPG позволяют игроку самому создавать собственный стиль игры и тренировать аналитические способности. Игры такого жанра так же динамичны и часто требуют быстроты реакции, что так же положительно воздействует на работу мозга.

Одной из главной составляющей игр такого жанра является система хранения игровых предметов, так как позволяет осуществлять планирование дальнейшего развития игрового персонажа при сборе той или иной экипировки и предметов. В различных играх инвентарь может представляться по-разному, однако основа всегда остается неизменной: игрок может вызывать инвентарь в любой момент времени во время игры, само хранилище представляется может представляться либо списком, либо массивом ячеек, в которых хранятся те или иные предметы.

Система лояльности, представляемая как ценообразование в игре, встречается все чаще и чаще. Она способствует к большему изучению игры и таким образом разработчик удерживает интерес игрока.

Так же для поддержания интереса к игре необходима графическая часть. Создание окружения в игре настолько же важно, как и сама механика и сюжет. Однако необходимо учитывать жанр игры и цели, для которых будут использоваться модели.

Цель работы: Разработать систему хранения игровых предметов, систему лояльности и создать статическое окружение.

В связи с поставленной целью решаются следующие задачи:

- Выполнить сравнительный анализ систем лояльностей и систем хранения предметов персонажа;
- Определить положительные моменты и недостатки аналогов;
- Предложить варианты устранения недостатков и определить дальнейшие задачи разработки систем;
- Создать статические модели окружения;
- Реализовать составленные алгоритмы на языке программирования C#;
- Обеспечить визуальное представление алгоритмов и варианты их прохождения.

В ВКР были использованы слова, которые относятся к данной предметной области. Далее будут даны основные термины и их определения.

Видеоигра (компьютерная игра) – это «компьютерная программа, служащая для организации игрового процесса, связи с партнёрами по игре, или сама выступающая в качестве партнёра».

(Статический игровой) объект – это не игровой элемент игры, которым заполняется игровое окружение.

(Игровой) персонаж – это игровой элемент игры, управляемый пользователем.

Неигровой персонаж (NPC) (от англ. Non-Player Character) – это персонаж, управляемый программно, служащий в качестве реализации системы заданий и игровых магазинов.

(Игровой) предмет — игровое снаряжение, необходимое персонажу для выполнения задания или для удовлетворения его потребностей — экипировка, расходные материалы и т. п.

Инвентарь — это система хранения предметов игрового персонажа.

Low-poly – трехмерная модель с малым количеством полигонов.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Моя выпускная квалификационная работа направлена на создание игровой экономики посредством рассмотрения системы лояльности и системы хранения предметов, входящих в состав имущества персонажа, в дальнейшем называемая «инвентарем». Данные системы необходимы при разработке каждой игры жанра RPG и в каждой из них реализованы по-разному, поэтому важно рассмотреть уже существующие аналоги и выявить положительные качества их реализованных систем.

1.1 Сравнение с аналогами

Аналогами, рассмотренными для данной работы, являются такие игры как «Diablo II», «Grim Down», «The Elder Scrolls III: Morrowind», «Ведьмак 3. Дикая охота.».

Данные игры относятся к одному жанру — RPG, что в дальнейшем позволит нам провести сравнение этих игр по заданным критериям.

1.1.1 Система хранения игровых предметов персонажа.

1.1.1.1 Diablo II.

Diablo II — компьютерная игра в жанре Action/RPG, была выпущена компанией Blizzard Entertainment для Windows и Mac OS в 2000 году [1].

Плюсы системы инвентаря:

- Добавление реалистичности в игровой процесс путем привязки реального размера предмета к количеству ячеек, занимаемым им.
- Использование системы дополнительного инвентаря для расходных предметов.

Недостатки системы инвентаря:

- Небольшой размер склада, без возможности его расширить.
- Сортировка отсутствует

Ниже представлен инвентарь игры Diablo II (Рисунок 1).



Рисунок 1 - Инвентарь Diablo II

1.1.1.2 Grim Dawn.

Grim Dawn — компьютерная игра жанра Action/RPG, разработанная компанией Crate Entertainment и выпущенная в 2016 году [2].

Плюсы системы инвентаря:

- Дополнительный инвентарь
- Расходные предметы складировются по несколько штук и занимают одну ячейку инвентаря.
- Добавление реалистичности в игровой процесс путем привязки реального размера предмета к количеству ячеек, занимаемым им.

Недостатки системы инвентаря:

- Сортировка отсутствует.
- Переключение между основным инвентарем и дополнительным происходит вручную посредством нажатия кнопки мыши.

Ниже представлен инвентарь игры Grim Dawn (Рисунок 2).



Рисунок 2 - Инвентарь Grim Dawn

1.1.1.3 The Elder Scrolls III: Morrowind.

The Elder Scrolls III: Morrowind – компьютерная ролевая игра с открытым миром в жанре фэнтези, разработанная Bethesda Game Studios и изданная Bethesda Softworks и Ubisoft в 2002 году [3].

Плюсы системы инвентаря:

- Динамически расширяемый инвентарь
- Расходные предметы складировются по несколько штук и занимают одну ячейку инвентаря
- Для каждого предмета (кроме расходных) используется одна ячейка инвентаря
- Имеется сортировка

Недостатки системы инвентаря:

- При долгой игре возникают проблемы с поиском предмета в инвентаре

Ниже представлен инвентарь игры The Elder Scrolls III: Morrowind (Рисунок 3).



Рисунок 3 - Инвентарь The Elder Scrolls III: Morrowin

1.1.1.4 Ведьмак 3: Дикая Охота.

Ведьмак 3: Дикая Охота — мультиплатформенная компьютерная ролевая игра, разработанная польской студией CD Projekt RED, выпущенная в 2015 году для Windows, PlayStation 4 и Xbox One [4].

Плюсы системы инвентаря:

- Динамически расширяемый инвентарь
- Расходные предметы складировются по несколько штук и занимают одну ячейку инвентаря.
- Добавление реалистичности в игровой процесс путем привязки реального размера предмета к количеству ячеек, занимаемым им.
- Использование визуального разграничения предметов по типу.

Недостатки системы инвентаря:

- При долгой игре возникают проблемы с поиском предмета в инвентаре.

Ниже представлен инвентарь игры Ведьмак 3: Дикая Охота (Рисунок 4).



Рисунок 4 - Инвентарь Ведьмак 3: Дикая охота

Игроки чаще отдают предпочтение целостному инвентарю фиксированного размера, но с возможностью использования дополнительного инвентаря, обширному хранилищу, а также формированию пачек из расходных предметов одного типа. Многим игрокам не нравятся системы, где количество ячеек, занимаемых предметом, так как это, по их мнению, это усложняет процесс оптимизации пространства инвентаря.

1.1.2 Обзор аналогов системы лояльности

В The Elder Scrolls III: Morrowind соотношение цен на купленный/проданный товар и размер максимально возможной наценки/скидки зависит, прежде всего, от трех параметров: вашего уровня Торговли, процента Отношения торговца к вам и процента Усталости. Изначально торговец предлагает только половину стоимости предмета, а продает за гораздо большую цену, но благодаря развитому навыку торговли предложение становится более выгодным. Развивать навык можно меняя предложение цены — немного увеличивать при продаже и

уменьшать при покупке. Навык торговли также определяет, насколько сильно можно изменить предложенную торговцем цену [7].

Стоит учитывать, что если торговцу цена покажется слишком высокой при продаже или слишком низкой при покупке — он отклонит предложение, а его отношение герою при этом немного упадёт.

Торговля прекрасно сочетается с навыком «Красноречие», так как, повысив расположение торговца, можно уговорить его на более выгодную сделку.

В игре «Ведьмак 3. Дикая охота.» система лояльности является более продуманной и зависит от множества критериев: первоначальная лояльность, первоначальная ценность предмета, процент его использования, из каких материалов мог быть создан данный предмет, стоимости этих компонентов, на какие ресурсы можно было разложить этот предмет и уже их цена и т. д. Так же влияние количества операций купле-продажи на дальнейшее ценообразование. Однако и эта система имеет ряд недостатков: из-за обилия критериев требуется определенная часть вычислительных ресурсов ПК для обработки и хранения данных, также чтобы решить проблему повышения стоимости со временем, в игру были добавлены ремонтные наборы, кроме того, для того, чтобы создать возможность для производства, в игре предусмотрено потенциально неограниченное количество инвентаря, которое может запутать игроков [8].

1.2 Выводы по главе

Проведя сравнительный анализ можно сформулировать задачи для каждой из рассматриваемых систем.

1.2.1 Итоги анализа по системе инвентаря

1. Создать инвентарь оптимального фиксированного размера.
2. Не использовать привязку предмета определенного типа к количеству ячеек инвентаря.

3. Формировать пачки для расходных предметов.
4. Добавить новые возможности в систему хранения игровых предметов.

В качестве дополнительной функции по улучшению инвентаря было предложено создать визуальное разграничение по редкости предмета. В разрабатываемой игре предметы будут иметь 3 категории редкости: обычный, редкий и легендарный. Данная система позволит пользователю более просто осуществлять поиск предметов в инвентаре, а также уменьшит количество случайных продаж дорогих вещей.

Таким образом создадим итоговую таблицу (Таблица 1).

1.2.2 Итоги анализа по системе лояльности

Так же задачи были выделены и для системы лояльности:

1. Сделать систему лояльности, которая будет использовать достаточное количество критериев, благодаря которым экономика в игре будет более динамичной, но не затрачивающая большие вычислительные мощности ПК.
2. Добавить собственные критерии, отличающие разрабатываемую систему в игре от аналогов.

Таблица 1 – Сравнительный анализ аналогов и разрабатываемой системы

Критерии оценки	Diablo 2	Grim Dawn	The Elder Scroll III: Morrowind	Ведьмак 3: Дикая охота	Разрабатываемая система
Размер занимаемых ячеек не зависит от размера предмета	-	-	+	-	+
Создание инвентаря с ячейками	+	+	-	+	+
Использование дополнительного инвентаря в качестве расширения основного	-	+	-	-	+
Большой размер склада	-	-	+	+	+
Создание связок из расходных предметов	-	+	+	+	+
Разграничение предметов по типу редкости	-	-	+	+	+

2 ПРОЕКТИРОВАНИЕ

Постановка задачи: разработать систему хранения игровых предметов и систему лояльности. Определить набор моделей для игрового пространства.

Для проектирования будем использовать язык графического описания UML.

2.1 Проектирование инвентаря

Перед тем как реализовывать саму систему необходимо спроектировать ее компоненты и выделить общую схему их взаимодействий. Для начала выделим для данной системы участников и ее возможности. Для этого спроектируем диаграмму прецедентов или вариантов использования, которая представлена в Приложении А1. Прецеденты - это не зависящее от реализации высокоуровневое представление того, что пользователь ожидает от системы, т.е. описание функциональности системы. Действующее лицо - это все, что взаимодействует с системой. Данные составляющие определяют сферу применения создаваемой системы. При этом прецеденты описывают все то, что происходит внутри системы, а действующие лица - то, что происходит снаружи [10]. В диаграмме действующие лица обозначаются в виде значков фигур, а варианты использования - в виде овалов [6].

В данном случае действующим лицом является игрок, а прецедентами — все возможные использования инвентаря.

2.2 Проектирование системы лояльности

Ценовой также, как и игровой, баланс обычно ищется методом проб и ошибок. Необходимо множество тестеров (в случае самостоятельных разработчиков, игра выставляется в свободное тестирование), а потом долго-долго идет подгонка и настройка всех параметров. Однако несмотря на обилие теоретических разработок, рассчитать точный ценовой и игровой баланс удастся редко. Но для того, чтобы его поиск не занял годы,

надо бы с самого начала сотворить нечто, издали напоминающее баланс [11].

Система лояльности будет представлять из себя расчетную формулу покупки и продажи игрового предмета игроком, а также первоначальный расчет получения игровой валюты с противника и после выполнения заданий. После этапа анализа были выделены задачи для разрабатываемой системы (см. пункт 1.1.2). Одной из задач было создание критериев, которые отличали бы создаваемую систему от других. В связи с этим было предложено ввести такие критерии как: лояльность фракции, лояльность конкретного не игрового персонажа и редкость игрового предмета. Так же для составления формулы нам понадобятся дополнительные критерии — первоначальная цена предмета, шкала лояльности фракции, шкала лояльности не игрового персонажа в локации и начальная цена предмета.

Для начала необходимо определить первоначальную цену [9]. Выберем некое число как «среднее» количество игровой валюты за противника А. Данную величину отметим как С. То же делаем для противника типа В. Далее рассчитаем вероятность победы противника типа А, то есть смоделируем бой между А и В. Бой проводился 100 раз и у нас получилось определенное количество успешных исходов для противника А и В, обозначим их E_A и E_B .

Тогда новая цена будет выглядеть так:

$$C_A = f(E_A) * C_B, \quad (1)$$

где C_A – цена за противника А;

C_B – цена за противника В;

$f(E_A)$ – функция от вероятности.

$$f(E_A) = E_A/E_B, \quad (2)$$

если ни одно из чисел не равно 0; если же оно равно 0, то заменим его на маленькую величину = 0,01.

При моделировании были использованы противники с такими характеристиками:

Противник класса «воин»:

```
mob.enemyChar.maxHealth = Random.Range(20, 35) * 10; //Здоровье объекта  
mob.enemyChar.damage = Random.Range(10, 20); //Урон, наносимый объектом
```

Противник класса «маг»:

```
mob.enemyChar.maxHealth = Random.Range(10, 20) * 10; //Здоровье объекта  
mob.enemyChar.damage = Random.Range(20, 30); //Урон, наносимый объектом
```

Количество успешных исходов для воина составило 58, количество успешных исходов для мага — 42. Таким образом мы получили потенциальную стоимость противника класса «воин» $C_B = 1,38 * C_M$

За среднюю цену мы приняли значение в 300 е.и.в. (единиц игровой валюты), получив таким образом цену за воина в 414 е.и.в., и цену за мага в 216 е.и.в.

По такому принципу мы получаем стоимость всех противников.

Для того чтобы рассчитать первоначальную стоимость предмета необходимо произвести такие же расчеты, только с использованием этого предмета непосредственно при моделировании боя. Так противник типа «воин» был экипирован мечом, прибавляющий его урон на 5-10. После моделирования боя были получены следующие результаты: количество успешных исходов для воина — 72, количество успешных исходов для мага - 28; $C_B = 2,57 * C_M$. Таким образом цена воина возросла до 771 е.и.в. Теперь вычитаем из получившегося значения стоимость самого противника и получим первоначальную стоимость предмета в 357 е.и.в.

Далее рассчитаем продажу и покупку предмета с учетом лояльности к персонажу фракций и не игровых персонажей.

Первоначальная цена в дальнейшей формуле используется в е.и.в. и считается за 100%.

Начальная цена — это сумма первоначальной цены и цена редкости предмета. Так же переводится в проценты.

Редкость предмета. В ВКР представлено 3 вида редкости предмета: обычный, редкий и легендарный. Предмет может выпадать не всегда. Если же он выпал, то автоматически сначала приравнивается к обычному. Далее

этот предмет может с 30-ти процентной вероятностью стать редким и с 5-ти процентной вероятностью легендарным. Редкость дает прибавку к первоначальной цене в процентах: Обычный — 0%, редкий — 70%, легендарный — 95% от первоначальной стоимости предмета. То есть если первоначальная цена предмета равна 200 е.и.в., то если предмет оказывается легендарным, его начальная стоимость становится равна 390 е.и.в. Цена редкости предмета в данном примере составила 190 е.и.в..

Репутация с фракцией — это числовое значение значимости персонажа для той или иной фракции. Минимальным значением является 0 о.р.ф. (очки репутации фракции), а максимальным — 100 о.р.ф. Для персонажа, первоначально выбравшему определенную фракцию, будет начислено 30 о.р.ф. Очки репутации фракции можно зарабатывать, выполняя задания фракции и задания не игровых персонажей, принадлежащих к этой фракции. За задание фракции начисляется по 5 о.р.ф., а за задания NPC — 2о.р.ф. Полностью заполненная шкала репутации фракции дает 30-ти процентную скидку при покупке или продаже предмета торговцу, принадлежащему к данной фракции.

Репутация с неигровым персонажем в одной взятой локации — это числовое значение значимости персонажа для той или иной локации. Минимальным значением является 0 о.р.нп. (очки репутации неигрового персонажа), а максимальным — 100 о.р.нп. Очки репутации фракции можно зарабатывать, выполняя задания не игровых персонажей в одной конкретной локации. За задание начисляется по 10 о.р.нп. Полностью заполненная шкала репутации с неигровым персонажем дает 20-ти процентную скидку при покупке или продаже предмета торговцу, находящемуся в данной локации.

Конечная цена предмета — цена, полученная на выходе.

Таким образом составим формулы покупки и продажи предметов персонажем:

Покупка предмета у не игрового персонажа:

(3)

P_e – стоимость покупки предмета у неигрового персонажа в е.и.в.;

P_f – первоначальная стоимость предмета в е.и.в.;

P_r – процент редкости предмета;

R_{fr} – количество очков репутации фракции в процентах;

R_{npc} – количество очков репутации неигрового персонажа в процентах.

Продажа предмета не игровому персонажу:

(4)

P_s – стоимость продажи предмета неигровому персонажу в е.и.в.

Далее приведем примеры для предметов с одинаковой стоимостью, но разной редкостью. Первоначальная цена будет равна 200 е.и.в. Редкость и репутация с фракцией и NPC будет меняться. Пример расчета стоимости покупки предмета у неигрового персонажа приведен в Таблице 2. Пример расчета продажи предмета неигровому персонажу приведен в Таблице 3.

Таблица 2 – Пример расчета стоимости покупки предмета у NPC

	Количество очков репутации фракции в %	Количество очков репутации NPC в %	Первоначальная стоимость предмета - const = 100%	Первоначальная стоимость предмета в е.и.в.	% редкости предмета	Стоимость покупки
Предмет №1 Редкость «Обычный»	0	0	100	200	0	200
	6	20	100	200	0	148
	30	0	100	200	0	140
	30	20	100	200	0	100
Предмет №2 Редкость «Редкий»	0	0	100	200	70	340
	6	20	100	200	70	288
	30	0	100	200	70	280
	30	20	100	200	70	240
Предмет №1 Редкость «Легендарный»	0	0	100	200	95	390
	6	20	100	200	95	338
	30	0	100	200	95	330
	30	20	100	200	95	290

Таблица 3 – Пример расчета стоимости продажи предмета NPC

	Количество очков репутации фракции в %	Количество очков репутации NPC в %	Первоначальная стоимость предмета - const = 100%	Первоначальная стоимость предмета в е.и.в.	% редкости предмета	Стоимость продажи
Предмет №1 Редкость «Обычный»	0	0	100	200	0	200
	6	20	100	200	0	252
	30	0	100	200	0	260
	30	20	100	200	0	300
Предмет №2 Редкость «Редкий»	0	0	100	200	70	340
	6	20	100	200	70	392
	30	0	100	200	70	400
	30	20	100	200	70	440
Предмет №1 Редкость «Легендарный»	0	0	100	200	95	390
	6	20	100	200	95	442
	30	0	100	200	95	450
	30	20	100	200	95	490

2.3 Определение визуальной направленности моделей

В данной ВКР разрабатываются части игры жанра RPG, где установка камеры находится наиболее отдалено в сравнении с другими жанрами. Следовательно, нет необходимости моделировать высокополигональные модели, так как они лишь незначительно улучшат графику (иногда их влияние по сравнению с малополигональными аналогами совсем незаметно), однако технические требования сильно возрастут [13]. Таким образом было принято решение использовать Low-poly моделирование, то есть создавать модели, которые состоят из минимального числа полигонов, при этом их достаточное количество для визуального восприятия получаемого объекта.

Для ВКР были созданы такие объекты как: 5 видов домов, 2 вида бочек, ящик, 2 вида колодцев, 2 вида надгробия, 3 вида балок, 2 вида телег.

Наиболее сложным в моделировании были дома. Далее будет представлен процесс создания одного из них.

2.3.1 Моделирование

Для начала создаем простой примитив «параллелепипед» с параметрами длины, ширины и высоты. Далее мы конвертируем его в Editable poly (редактируемая полигональная сетка) и применим инструмент «Секущая плоскость» для разделения нашего параллелепипеда на дополнительные полигоны для создания балок. При помощи инструмента «Выдавливание» создадим их. Затем создадим примитивами «плоскость» стены, чуть углубленные в балки и соединим конструкцию инструментом attach. Создадим лестницу при помощи добавления линий инструментом cut и выделив полигон примените инструмент slice plane создадим дополнительные горизонтальные линии. Далее инструментом extrude выдавим ступеньки. Таким же способом, используя инструменты slice plane и extrude создадим окна и дверь у нашего здания. Дополнительные опорные балки и доски на окнах делали при помощи простых примитивов

box, присоединяя далее инструментом attach. Второй этаж создается по тем же принципам. Далее создадим сложный примитив L-Ext и зададим ему количество сегментов сторон 3, так как крыша должна быть чуть заужена. При помощи обычного перемещения граней создадим нужную нам форму. Таким образом наш дом готов. Фонарь и вывеска создавались по похожим принципам. В качестве еще одного украшения был смоделирован кувшин. Для его создания использовался сплайн. Была создана граница половины кувшина и при помощи модификатора Lathe (модификатор для создания объектов путем вращения сплайна вокруг центральной оси) мы получили готовый кувшин. На Рисунке 5 представлена готовая модель.

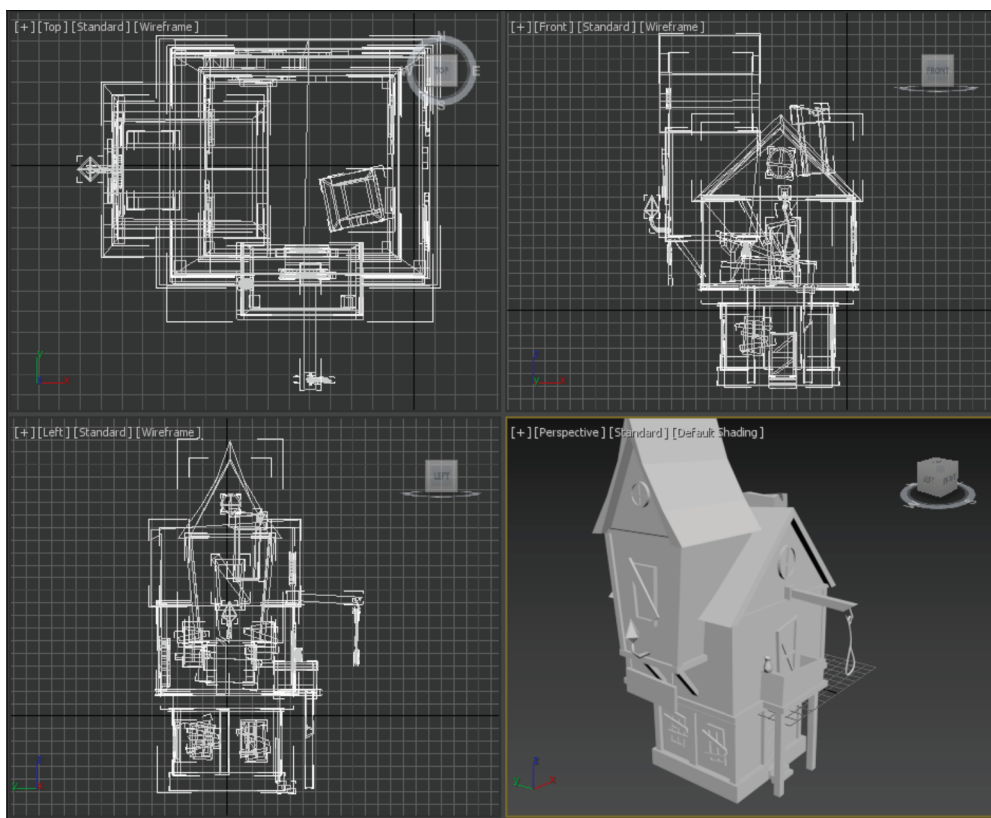


Рисунок 5 – Модель дома

2.3.2 Текстурирование объекта

Для примера возьмем еще один моделируемый для этой ВКР объект — колодец. Для начала необходимо создать развертку для всех элементов колодца — ведро, веревка, ручка, стойки, основание колодца и балка крепления. К каждой части применяет модификатор Unwrap UVW и далее

открываем UV Editor. Далее корректируем развертку при помощи инструментов break и relax:custom. Затем применяем рендер (Рисунок 6) и рисуем необходимую нам текстуру. Получившуюся модель показана на Рисунке 7.

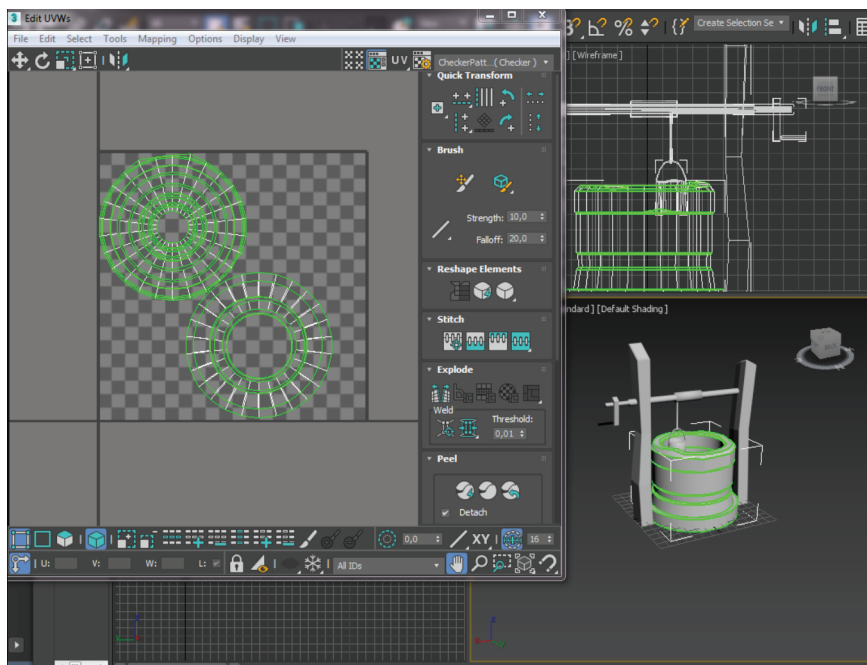


Рисунок 6 – Интерфейс Unwrap UVW



Рисунок – 7 Результат наложения текстур

3 РАЗРАБОТКА СИСТЕМЫ

Разработка системы инвентаря включает в себя создание классов и методов, а также описание алгоритмов работы программы.

3.1 Принципы взаимодействия пользователя с системой

Все взаимодействие происходит с использованием нажатия на символные клавиши, клавиши — модификаторы, а также использование клавиш мыши.

После загрузки прототипа, пользователь может вызвать инвентарь нажатием на клавишу I(Ш). Инвентарь можно передвигать на удобное пользователю положение экрана путем зажатия левой кнопки мыши и передвижение ее на необходимый участок экрана.

Для демонстрации данной системы был выделен генератор игровых предметов. При взаимодействии путем клика мышкой на генератор пользователь добавляет предмет в инвентарь. Каждый предмет занимает одну ячейку инвентаря. Однако предметы, относящиеся к классу «расходуемые» могут складироваться вместе. Количество таких предметов в связке обозначено цифрой в правом нижнем углу ячейки. Посредством нажатия клавиши — модификатора «Shift» и клику по расходуемому предмету, можно провести разделение связки на более мелкие. Так же при одиночном клике на игровой предмет и дальнейшему клику на пустую ячейку можно перенести выбранный предмет в пустую ячейку. Если же мы кликнули не на пустую ячейку, а на занятую, то предметы меняются местами. Для удаления предмета из инвентаря необходимо кликнуть по предмету и затем кликнуть по области вне границы инвентаря. Такой предмет удалится безвозвратно.

3.2 Описание ключевых классов

Данные системы реализованы на языке программирования C#. Он является объектно-ориентированным языком программирования и

содержит инструментарий, позволяющий создавать и использовать различные классы для создания качественного кода, что и было использовано для написания разрабатываемых систем.

3.2.1 Class Inventory.cs.

Данный класс является классом — экземпляром инвентарей и их частного функционала. Листинг программы представлен в Приложении В1.

Метод «Update» служит для постоянного обновления структуры игры. В ней содержатся описание всех важных алгоритмов, касаемых инвентаря, а также проверка на захват объекта.

Метод «Open» реализует открытие и закрытие инвентаря по нажатию клавиши.

Метод «ShowToolTip» создаст всплывающую подсказку при наведении на предмет.

Метод «HideToolTip» прячет подсказку о предмете после передвижения курсора в другую область.

Метод «CreateLayout» служит для инициализации инвентаря и установки слотов в нужных позициях.

Метод «Additem» реализует добавление предмета в слот инвентаря.

Метод «MoveInventory» реализует изменение положение инвентаря на экране.

Метод «PlaceEmpty» размещает предмет в новую ячейку инвентаря, если не было найдено среди расходуемых предметов места в связке или предмет являлся экипировкой.

Метод «MoveItem» реализует перемещение предметов в пространстве инвентаря.

Метод «CreateHoverIcon» создает «зависаемый» объект. При перемещении клике на предмет он «зависает» на мышке, чтобы пользователь видел какой именно объект он собирается переместить в другую ячейку.

Метод «PutItemBack» возвращает «зависимый» объект в ячейку, если действие не было совершено с этим объектом.

Метод «SplitStack» реализует разделение предметов, находящихся в связке.

Метод «ChangeStackText» изменяет значение связки при выполнении действий над ней.

Метод «MergeStacks» - если в связке имеется еще место, то данный метод позволяет объединить 2 связки.

Метод «FadeOut» создает плавное затухание при закрытии инвентаря.

Метод «FadeIn» создает плавное затухание при открытии инвентаря.

3.2.2 Class InventoryManager.cs.

Данный класс создан для чтения из XML и добавление предметов в ItemContainer. Здесь реализуется получение типов предметов из XML и самих предметов. Листинг программы представлен в Приложении В2.

3.2.3 Class Slot.cs.

Данный класс создан для работы с предметами в ячейках и хранения информации о каждой. Листинг программы представлен в Приложении В3.

Метод «Start» инициализирует и форматирует ячейки в инвентаре.

Метод «AddItem» - при добавлении предмета в инвентарь уменьшает количество пустых слотов и помещает предмет в выданный слот.

Метод «AddItems» меняет местами 2 предмета в инвентаре.

Метод «ChangeSprite» реализует подсветку предмета при наведении на него в инвентаре.

Метод «UseItem» изменяет состояние ячейки при использовании предмета, либо изменяет количество предметов в связке.

Метод «ClearSlot» очищает ячейку от всех предметов.

Метод «RemoveItems» удаляет несколько предметов.

Метод «RemoveItem» удаляет один предмет.

Метод «OnPointerClick» реализует интерфейс для работы с инвентарем по клику.

Метод «SwapItems» реализует замену предметов.

Метод «Save» - сохраняет предметы и их положение в инвентаря, а потом загружает их

3.2.4 Class Interactable.cs.

Данный класс создан для представления реализации процесса взаимодействия игрового персонажа с окружающей средой. Листинг программы представлен в Приложении В4.

3.2.5 Class PickUpItem.cs.

Данный класс наследуется от Interactable и привязывается к любому предмету, который можно поднять и положить в инвентарь. Листинг программы представлен в Приложении В5.

Для наглядной демонстрации взаимодействия между классами была создана диаграмма классов (Приложение Б1).

3.3 Описание алгоритмов

Пользователь имеет 8 типов взаимодействия с инвентарем и предметами, находящимися там.

1. Вызов инвентаря по нажатию клавиши
2. Перемещение инвентаря по игровому пространству
3. Добавление предмета в инвентарь
4. Удаление предмета из инвентаря
5. Перемещение предмета между ячейками инвентаря
6. Получение информации о предмете при наведении на него
7. Разделение связок расходуемых предметов
8. Объединение связок расходуемых предметов

При запуске игры у нас вызывается единожды функция Start, а затем происходит вызов Update каждый игровой фрейм.

В Inventory хранятся все уникальные для инвентаря переменные и функции. InventoryManager все общие переменные – slotPrefab, iconPrefab,

hoverPrefab, itemObject, hoverObject, tooltipObject, sizeTextObject, visualTextObject, Canvas, stackText, selectStackSize, eventSystem. Slot – делает масштабирование слотов и в нем же мы полчаем ссылку на родительский объект canvas (устанавливаем размер текста, изменяем максимальный и минимальный размера текста в зависимости от разрешения).

Функция Update содержится в скриптах Inventory и Interactable.

В Inventory – если мы взяли какой то предмет, и выкидываем его за пределы инвентаря, то мы очищаем слот, если у нас есть объект на мышке и инвентарь не закрыт (Canvas = 1), то мы присоединяем объект, который мы перемещаем цепляем его иконку на мышку.

В Interactable – если происходит взаимодействие с одним и тем же предметом, то взаимодействие происходит один раз, а не каждый фрейм, так же проверяем дистанцию между позицией персонажа и точки назначения.

На слотах и инвентаре висят события OnClick . как только мы совершаем клик по слоту вызывается функция OnPointerClick в классе Slot. Если мы нажимаем правую кнопку мыши и у нас нет hover объекта и альфа >0 , то вызываем UseItem класса Slot. Если слот, на который мы кликнули, не пустой, то мы удаляем верхний элемент стека Items и вызываем метод Use из класса Item script и, если количество предметов больше 1, текст перезаписывается (количество предметов), если нет - строка становится пустой. И если слот становится пустым, мы меняем спрайт на спрайт пустого слота и добавляем в список пустых слотов наш слот.

Далее OnPointerClick проверяет, если была нажата левая кнопка мыши с кнопкой Shift, а также слот не является пустым и не существует hover объекта, то мы на позиции мышки устанавливаем панель StackSizeBack. Захватываем позицию мышки, активируем панель и переносим нашу панель (SelectStackSize) на эти координаты. Дальше

вызываем метод `StackSizeInfo`, передавая ему информацию о количестве предметов.

В `SelectSizeInfo` мы выключаем `TooltipObject`, которая отвечает за подсказки, чтобы она не мешала, устанавливаем `SplitAmount` в 0, присваиваем максимальному значению счетчика то значение, которое мы получили при вызове функции и записываем в текст панельки переменную `SplitAmount`, которая изменяется при нажатии кнопок левой и правой стрелочки, если `SplitAmount` меньше 0, то ты переприсваиваем 0. И если `SplitAmount` больше `maxStackCount`, то мы присваиваем значение `SplitAmount` `maxStackCount` и перезаписываем текст нашего стака - в `stackText` записываем переменную `SplitAmount`.

`MoveItem` из класса `Inventory`. В `clicked` будет храниться слот, по которому мы кликнули. Если есть `CanvasGroup` или альфа > 0 (объект не прозрачен), то в объект `Clicked` класса `InventoryManager` запишем тот объект, по которому мы кликнули. Если `MoveSlot` не пустой, то ты временному слоту присваиваем компонент слота, по которому мы кликнули. Если временный слот пуст, то мы вызываем функцию `AddItems`, которая переносит переменную из `MoveSlot` во временный слот тем самым очищая `MoveSlot`. Если временный слот не пуст и имена предметов во временном слоте и в `Move Slot` совпадают и связка предметов временного слота не полна, то мы вызываем метод объединения связок `MergeStacks`.

Функция `MergeStacks`. Передаем первым параметром `MoveSlot`, а вторым - `tmp` - временный слот.

`MergeStacks (Slot source, Slot destination)` создаем переменную `max`, в которой будет храниться значение максимального количества предметов, которое можно хранить в слоте `destination` минус реальное количество предметов в связке. Далее мы вычисляем количество предметов, которое можно перенести, чтобы в связках не оказалось больше предметов, чем в

них может быть. Далее в цикле `i..count` вызываем функцию `destination.AddItem(source.RemoveItems())`.

`AddItem`: в самом методе проверяем, если слот не пуст, то мы декрементируем количество пустых слотов и добавляем предмет в связку. Если количество предметов в связке больше одного, то мы в `stackTex.text` записываем это количество и вызываем метод `ChangeSprite`, передавая ему параметры `item.spriteNeutral`, `item.spriteHighlighted`.

3.4 Используемое программное обеспечение

3.4.1 Microsoft visual studio 2017

В качестве среды программирования мной была выбрана Microsoft Visual Studio 2017. Данный программный продукт включает в себя интегрированную среду разработки программного обеспечения и ряд других инструментальных средств.

Visual Studio построена на архитектуре, поддерживающей возможность использования встраиваемых дополнений от сторонних разработчиков, что позволяет расширять возможности среды разработки. Так же она поддерживает такие языки программирования как Си, C++ и C#.

3.4.2 Unity 3D 5.6.0.

Unity - это инструмент для разработки двух- и трехмерных приложений и игр, работающий под операционной системой Windows, Linux и OS X. Созданные с помощью Unity приложения работают под операционными системами Windows, OS X, Windows Phone, Android, Apple iOS, Linux, а также на игровых приставках Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One [5].

Приложения, созданные с помощью Unity, поддерживают DirectX и OpenGL. Активно движок используется как крупными разработчиками (Blizzard, EA, QuartSoft, Ubisoft), так и

разработчиками Indie-игр в силу наличия бесплатной версии, удобного интерфейса и простоты работы с движком.

Одной из причин выбора данного игрового движка послужил язык программирования, поддерживаемый им. C# является языком высокого уровня и имеет мягкий порог вхождения. Сегодня данный язык активно используется в различных IT сферах, что позволяет найти много дополнительной обучающей информации по нему [12]. C# поддерживается не только Microsoft, но и различными сторонними сообществами, что свидетельствует о его популярности.

3.5 Контрольный пример

В соответствии с темой ВКР, была создана система хранения игровых предметов. Были реализованы следующие операции:

1. Вызов инвентаря по нажатию клавиши.
2. Перемещение инвентаря по игровому пространству.
3. Добавление предмета в инвентарь.
4. Удаление предмета из инвентаря.
5. Перемещение предмета между ячейками инвентаря.
 - 5.1. Перемещение в пустую ячейку.
 - 5.2. Перемещение в уже занятую ячейку.
6. Разделение связок расходуемых предметов.
7. Объединение связок расходуемых предметов.
8. Получение информации о предмете при наведении на него

Пример работы с спроектированной и разработанной системой представлен ниже:

- 1) Сначала запускаем .exe – файл.
- 2) Нажимаем на клавишу I(Ш), которой вызываем инвентарь (Рисунок 8).



Рисунок 8 - Инвентарь

3) При нажатии левой кнопки мыши (ЛКМ) на фон инвентаря появляется возможность перемещения инвентаря по игровому пространству (Рисунок 9).

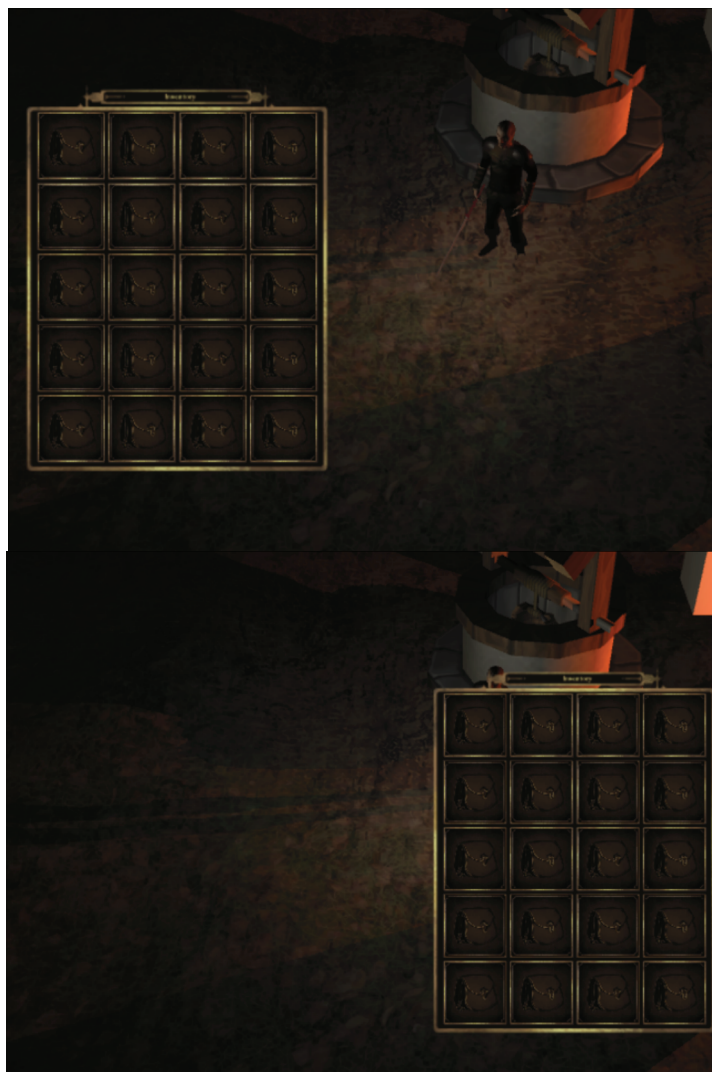


Рисунок 9 - Перемещение инвентаря

4) При нажатии ЛКМ по генератору игровых объектов (синий куб) в наш инвентарь добавляется игровой предмет (Рисунок 10).



Рисунок 10 - Добавление предмета в ячейку инвентаря по нажатию на генератор (синий куб)

5) При нажатии ЛКМ на предмет, а затем на область, находящуюся вне рабочей области, мы удаляем предмет (Рисунок 11).



Рисунок 11 - Удаление предмета из инвентаря

6) При нажатии ЛКМ на предмет, а затем на пустую ячейку, происходит перенос предмета в эту ячейку (Рисунок 12).



Рисунок 12 - Перемещение предмета в свободную ячейку инвентаря

7) При нажатии ЛКМ на предмет, а затем на ячейку с другим предметом, происходит смена предметов в этих ячейках (Рисунок 13).



Рисунок 13 - Перемещение предмета в ячейку с другим предметом.

8) При нажатии правой кнопки мыши (ПКМ) + SHIFT по связке предметов происходит вызов окна с выбором количества предметов в этой связке. Количество регулируется нажатиями на стрелки этого окна (Рисунок 14).



Рисунок 14 - Разделение связок расходуемых предметов

9) При нажатии ЛКМ по расходуемому предмету и дальнейшему нажатию на предметы с таким же именем происходит объединение этих предметов в одну связку (Рисунок 15).



Рисунок 15 - Объединение расходуемых предметов в связку

10) При наведении курсора на предмет появляется информация о предмете (Рисунок 16).



Рисунок 16 - Получение информации о предмете

Таким образом были продемонстрированы основные функции разработанной системы хранения игровых предметов.

ЗАКЛЮЧЕНИЕ

При выполнении выпускной квалификационной работы была разработана система хранения игровых предметов персонажа, которая предлагала пользователю интерфейс для работы с предметами – хранение, добавление, удаление, перемещение и получение информации об их характеристиках. При рассмотрении аналогов было выявлено, что не все из них удовлетворяют определенным требованиям игроков. Была составлена таблица, которая показывала, какие критерии пользователи ценят больше всего в системе инвентаря (сравнительный анализ представлен в Таблице 1). Разработанная система была создана с учетом всех предъявленных качеств.

Другой разработанной системой была система лояльности, которая представляла из себя расчет стоимости предмета с учетом лояльности неигровых персонажей по отношению к игровому. При проектировании была выведена формула, применимая к расчету цены при продаже и покупке игровым персонажем предметов у NPC. В ходе работы были продемонстрированы принципы расчета ценности противников, что так же понадобится при дальнейшем развитии проекта.

В дополнении были созданы модели окружения игры и были продемонстрированы навыки создания низкополигональных моделей.

Требования технического задания были выполнены в полном объеме и в установленные сроки.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Blizzard entertainment [Электронный ресурс]. Режим доступа: eu.blizzard.com/ru-ru/games/d2/index.html - Заглавие с экрана. – (Дата обращения: 14.02.2017).
2. Grim dawn [Электронный ресурс]. Режим доступа: www.grimdawn.com/index.php - Заглавие с экрана. – (Дата обращения: 14.02.2017).
3. The Elder Scrolls III: Morrowind [Электронный ресурс]. Режим доступа: <https://elderscrolls.bethesda.net> - Заглавие с экрана. – (Дата обращения: 14.02.2017).
4. The Witcher: Wild Hunt [Электронный ресурс]. Режим доступа: thewitcher.com/en/witcher3 - Заглавие с экрана. – (Дата обращения: 14.02.2017).
5. Unity – Руководство: Создание и Использование Скриптов [Электронный ресурс]. Режим доступа: <https://docs.unity3d.com/ru/530/Manual/CreatingAndUsingScripts.html> - Заглавие с экрана. – (Дата обращения: 18.04.2017).
6. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. 2-е издание. М.: ДМК Пресс, 2006 - 496с.
7. Искусство торговли [Электронный ресурс]. Режим доступа: tesmorr.narod.ru/sovet/torg.html - Заглавие с экрана. – (Дата обращения: 15.02.2017).
8. Как устроена экономика популярной игры «Ведьмак 3: Дикая охота» [Электронный ресурс]. Режим доступа: <https://vc.ru/p/witcher-economy> - Заглавие с экрана. – (Дата обращения: 14.02.2017).
9. Как рассчитать игровой баланс [Электронный ресурс]. Режим доступа: www.lki.ru/text.php?id=216 - Заглавие с экрана. – (Дата обращения: 02.05.2017).

10. Макурина Т.А., Гудков П.А. Объектно-ориентированное программирование. Методические указания к курсовому проектированию. Пенза: Изд-во ПГУ, 2005 - 67с.

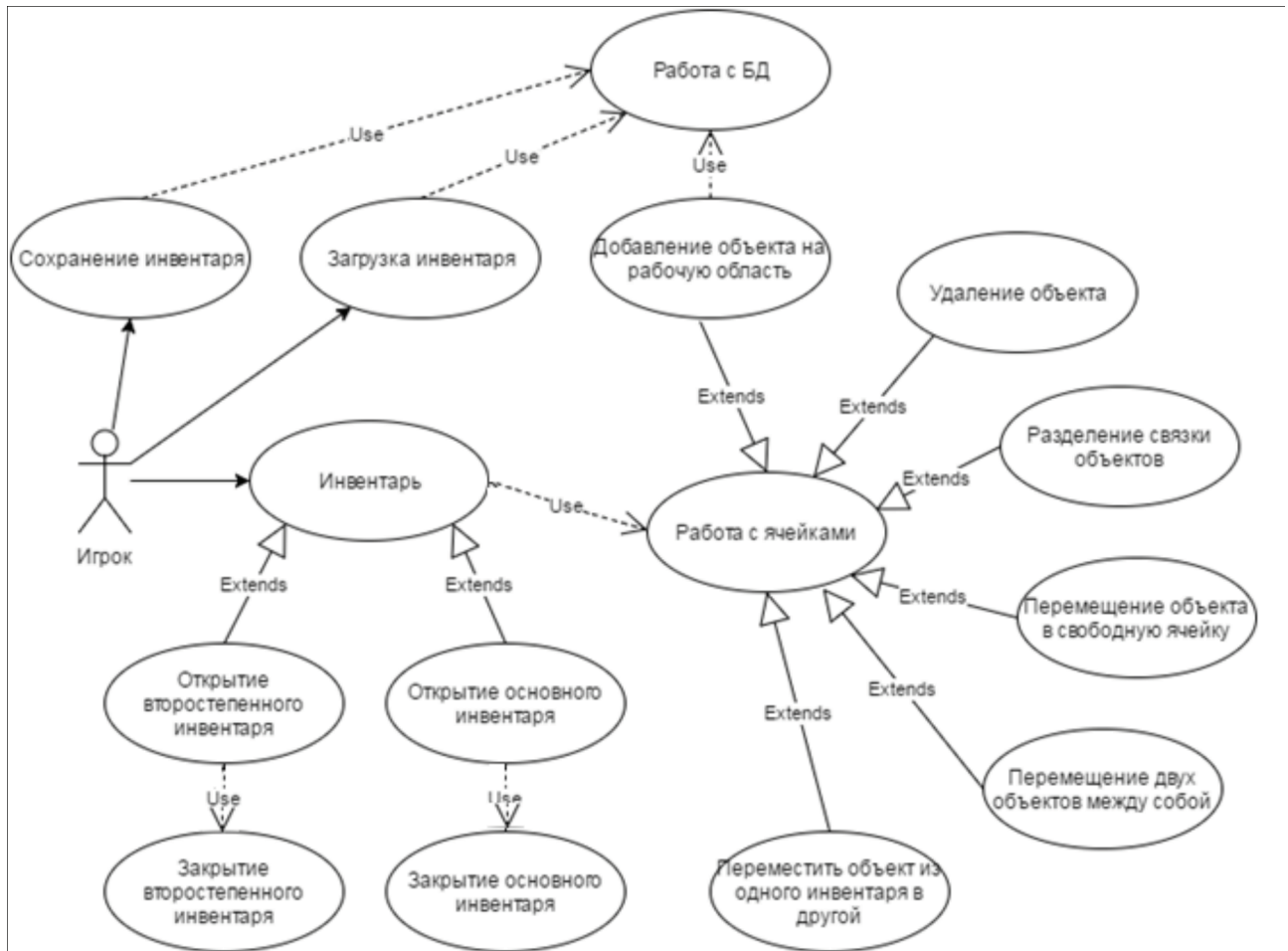
11. Основы игрового баланса: Детерминированные, недетерминированные и разрешимые игры [Электронный ресурс]. Режим доступа: <https://vc.ru/p/game-balance> - Заглавие с экрана. – (Дата обращения: 18.04.2017).

12. Особенности бесплатного кроссплатформенного движка Unity3d [Электронный ресурс]. Режим доступа: <https://gamesmaker.ru/3d-game-engines/unity3d/osobennosti-besplatnogo-krossplatformennogo-dvizhka-unity3d/> - Заглавие с экрана. – (Дата обращения: 01.05.2017).

13. Сыркин Ю. И. Краткое учебное пособие по курсу 3ds Max. СПб.: Международная школа дизайна, 2016 - 29с.

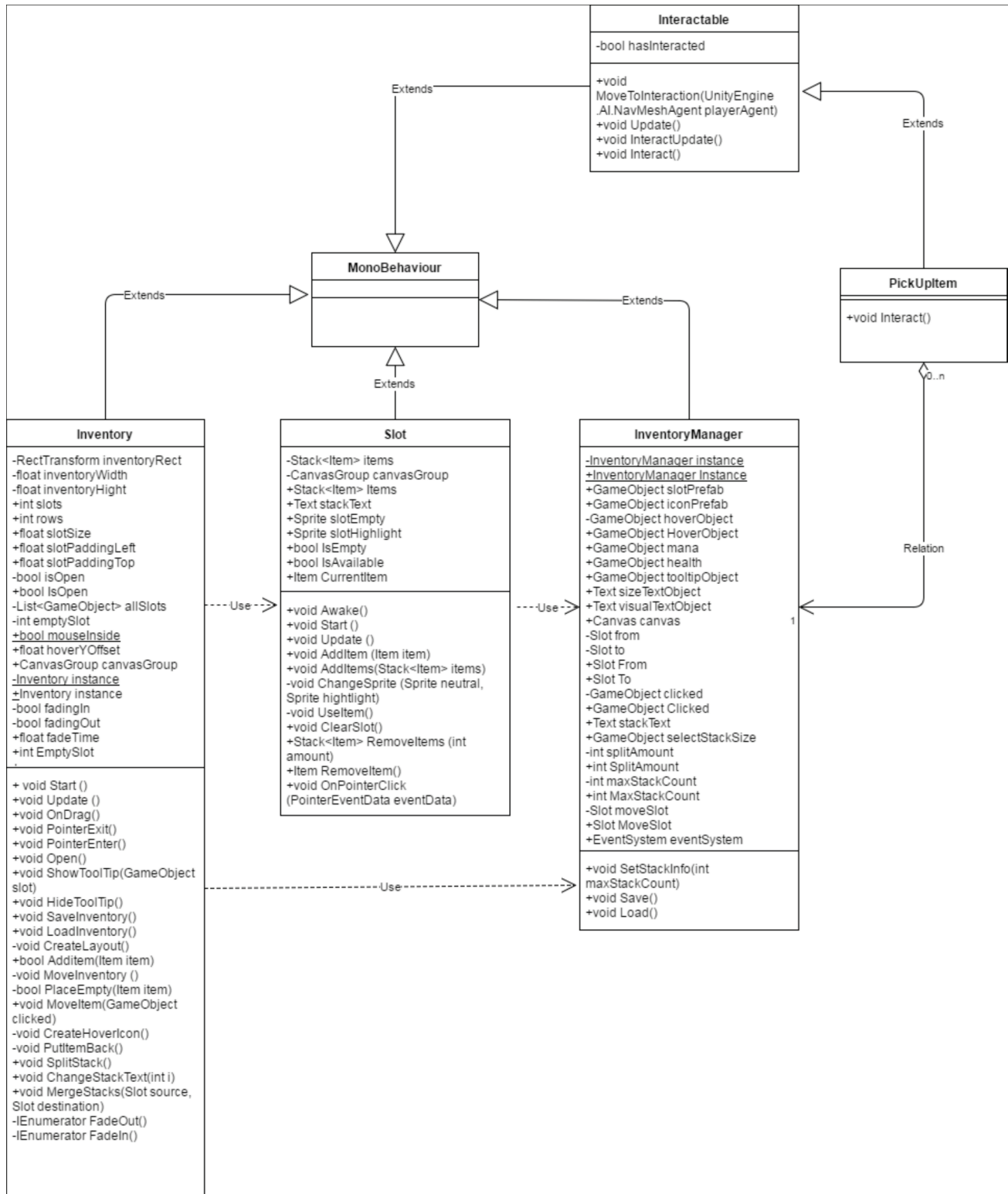
ПРИЛОЖЕНИЕ А1

Диаграмма прецедентов



ПРИЛОЖЕНИЕ Б1

Диаграмма классов



ПРИЛОЖЕНИЕ В1

Листинг программы. Inventory.cs

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

public class Inventory : MonoBehaviour
{
    // класс, содержащий все афинные преобразования инвентаря
    private RectTransform inventoryRect;

    private float inventoryWidth, inventoryHight;

    public int slots;

    public int rows;

    public float slotSize;

    public float slotPaddingLeft, slotPaddingTop;

    private bool isOpen;

    public bool IsOpen
    {
        get{ return isOpen;}
        set{ isOpen = value;}
    }

    private List<GameObject> allSlots;

    private int emptySlot;

    public static bool mouseInside = false;

    private float hoverYOffset;

    public CanvasGroup canvasGroup;

    private bool fadingIn;

    private bool fadingOut;

    public float fadeTime;

    public int EmptySlot {
        get { return emptySlot; }
        set { emptySlot = value;}
    }
}
```

```

// Use this for initialization
void Start ()
{
    isOpen = false;
    CreateLayout ();

    //создаем ссылку на конкретный слот
    InventoryManager.Instance.MoveSlot = GameObject.Find
("moveSlot").GetComponent<Slot> ();
}

// Update is called once per frame
void Update ()
{
    if(Input.GetMouseButtonUp(0))
    {
        if(!mouseInside && InventoryManager.Instance.From != null)
        {
            InventoryManager.Instance.From.GetComponent<Image>
().color = Color.white;
            InventoryManager.Instance.From.ClearSlot ();
            Destroy (GameObject.Find ("Hover"));
            InventoryManager.Instance.To = null;
            InventoryManager.Instance.From = null;

            Debug.Log ("i've made emptyslot="+);
        }
        else
        if(!InventoryManager.Instance.eventSystem.IsPointerOverGameObject(-1) &&
!InventoryManager.Instance.MoveSlot.IsEmpty)
        {
            InventoryManager.Instance.MoveSlot.ClearSlot ();
            Destroy(GameObject.Find("Hover"));
        }
    }
    // проверяем, есть ли объект на мышке
    if (InventoryManager.Instance.HoverObject != null &&
canvasGroup.alpha == 1)
    {
        //позиция в которой будет находиться наша икона все время
        существования
        Vector2 position;
        // если вкратце - берем координаты с монитора и переносим их в
        локальные координаты canvas, чтобы можно было с ней взаимодействовать

        RectTransformUtility.ScreenPointToLocalPointInRectangle(InventoryManager.Instance.canvas.transform as RectTransform, Input.mousePosition, InventoryManager.Instance.canvas.worldCamera, out position);
        // немного смещаем дердащий предмет по оси Y, чтобы проще было
        его положить в другой слот инвентаря
        position.Set (position.x, position.y - hoverYOffset);

        InventoryManager.Instance.HoverObject.transform.position =
InventoryManager.Instance.canvas.transform.TransformPoint (position);
    }
}

```



```

public void OnDrag()
{
    MoveInventory ();
}

public void PointerExit()
{
    mouseInside = false;
}

public void PointerEnter()
{
    if (canvasGroup.alpha > 0)
    {
        mouseInside = true;
    }
}

public void Open()
{
    if (canvasGroup.alpha > 0)
    {
        StartCoroutine ("FadeOut");
        PutItemBack ();
        HideToolTip ();
        isOpen = false;
    }
    else
    {
        canvasGroup.blocksRaycasts = true;
        gameObject.SetActive (true);
        StartCoroutine ("FadeIn");
        isOpen = true;
    }
}

public void DisableInventory()
{
    gameObject.SetActive (false);
}

public void EnableInventory()
{
    gameObject.SetActive (true);
}

public virtual void ShowToolTip(GameObject slot)
{
    //сохраняем ссылку слота на который указывает мышка
    Slot tmpSlot = slot.GetComponent<Slot> ();
    if (slot.GetComponentInParent<Inventory>().isOpen && !tmpSlot.IsEmpty
    && InventoryManager.Instance.HoverObject == null &&
    !InventoryManager.Instance.selectStackSize.activeSelf)
    {
        InventoryManager.Instance.visualTextObject.text =
tmpSlot.CurrentItem.GetTooltip ();
    }
}

```

```

        InventoryManager.Instance.sizeTextObject.text =
InventoryManager.Instance.visualTextObject.text;
        InventoryManager.Instance.tooltipObject.SetActive (true);

        float xPos = slot.transform.position.x + slotPaddingLeft;
        float yPos = slot.transform.position.y -
slot.GetComponent<RectTransform> ().sizeDelta.y - slotPaddingTop;

        InventoryManager.Instance.tooltipObject.transform.position =
new Vector2 (xPos, yPos);
    }

    public void HideToolTip()
    {
        InventoryManager.Instance.tooltipObject.SetActive (false);
    }

    public void SaveInventory()
    {
        string content = string.Empty;

        for (int i = 0; i < allSlots.Count; i++)
        {
            Slot tmp = allSlots [i].GetComponent<Slot> ();

            if (!tmp.IsEmpty) {
                // пример: "0-MANA-3;1-HEALTH-2;"
                content += i + "-" +
tmp.CurrentItem.Item.ItemName.ToString () + "-" + tmp.Items.Count.ToString () +
";";
            }
        }

        PlayerPrefs.SetString (gameObject.name + "content", content);
        PlayerPrefs.SetInt (gameObject.name + "slots", slots);
        PlayerPrefs.SetInt (gameObject.name + "rows", rows);
        PlayerPrefs.SetFloat (gameObject.name + "slotPaddingLeft",
slotPaddingLeft);
        PlayerPrefs.SetFloat (gameObject.name + "slotPaddingTop",
slotPaddingTop);
        PlayerPrefs.SetFloat (gameObject.name + "slotSize", slotSize);
        PlayerPrefs.SetFloat (gameObject.name + "xPos",
inventoryRect.position.x);
        PlayerPrefs.SetFloat (gameObject.name + "yPos",
inventoryRect.position.y);

        PlayerPrefs.Save ();
    }

    public void LoadInventory()
    {
        string content = PlayerPrefs.GetString (gameObject.name + "content");

        slots = PlayerPrefs.GetInt (gameObject.name + "slots");
        rows = PlayerPrefs.GetInt (gameObject.name + "rows");
        slotPaddingLeft = PlayerPrefs.GetFloat (gameObject.name +
"slotPaddingLeft");
        slotPaddingTop = PlayerPrefs.GetFloat (gameObject.name +
"slotPaddingTop");
        slotSize = PlayerPrefs.GetFloat (gameObject.name + "slotSize");

```

```

        inventoryRect.position = new Vector3
(PlayerPrefs.GetFloat(gameObject.name + "xPos"),
PlayerPrefs.GetFloat(gameObject.name + "yPos"), inventoryRect.position.z);

        CreateLayout ();
        // делим информацию о слотах на отдельные значения
        string[] splitContent = content.Split(';');

        for(int x = 0; x < splitContent.Length-1; x++)
        {
            string[] splitValues = splitContent [x].Split ('-');

            int index = Int32.Parse (splitValues[0]);

            //ItemType type = (ItemType)Enum.Parse (typeof(ItemType),
splitValues [1]);
            //получаем имя предмета которое пытаемся загрузить из
сохраненной игры
            string itemName = splitValues[1];
            // количество предметов, просто смотреть по контенту в
сохранениях
            int amount = Int32.Parse (splitValues[2]);

            Item tmp = null;

            for(int i = 0; i < amount; i++)
            {
                GameObject loadedItem = Instantiate
(InventoryManager.Instance.itemObject);
                // если код ничего не находит похожего по списку
предметов - перескакивает на следующий if
                if (tmp == null)
                {
                    // ищем в контейнере используемого предметы с
именем, всякий раз когда мы находим предмет с именем, мы ищем такой предмет в
списке имеющихся

                    tmp =
InventoryManager.Instance.ItemContainer.Consumeables.Find(item => item.ItemName ==
itemName);
                }
                if (tmp == null)
                {
                    tmp =
InventoryManager.Instance.ItemContainer.Equipment.Find(item => item.ItemName ==
itemName);
                }
                if (tmp == null)
                {
                    tmp =
InventoryManager.Instance.ItemContainer.Weapons.Find(item => item.ItemName ==
itemName);
                }

                loadedItem.AddComponent<ItemScript> ();
                loadedItem.GetComponent<ItemScript> ().Item = tmp;
                allSlots [index].GetComponent<Slot> ().AddItem
(loadedItem.GetComponent<ItemScript> ());

                Destroy (loadedItem);
            }
        }
    }

```

```

    }
}

//инициализация инвентаря и установка слотов в нужных позициях
public virtual void CreateLayout()
{
    if(allSlots != null)
    {
        foreach (GameObject go in allSlots)
        {
            Destroy (go);
        }
    }

    allSlots = new List<GameObject> ();
    // устанавливаем смещение в позицию нашего hoverObject
    hoverYOffset = slotSize * 0.01f;

    emptySlot = slots;
    // вычисление размеров инвентаря
    inventoryWidth = (slots / rows) * (slotSize + slotPaddingLeft) +
slotPaddingLeft + slotSize / 4;

    inventoryHight = rows * (slotSize + slotPaddingTop) + slotPaddingTop
+ slotSize / 2;

    inventoryRect = GetComponent<RectTransform> ();

    inventoryRect.SetSizeWithCurrentAnchors
(RectTransform.Axis.Horizontal, inventoryWidth);
    inventoryRect.SetSizeWithCurrentAnchors (RectTransform.Axis.Vertical,
inventoryHight);

    int columns = slots / rows;

    //позиционируем слоты
    for (int y = 0; y < rows; y++)
    {
        for (int x = 0; x < columns; x++)
        {
            //получаем префаб слота
            GameObject newSlot = (GameObject)Instantiate
(InventoryManager.Instance.slotPrefab);
            //получаем размер слота
            RectTransform slotRect =
newSlot.GetComponent<RectTransform> ();
            //переименовываем создаваемый объект в слот
            newSlot.name = "Slot";
            // делаем его потомком инвентаря на котором висит этот
скрипт
            newSlot.transform.SetParent (this.transform.parent);
            //задаем позицию слота
            slotRect.localPosition = inventoryRect.localPosition +
new Vector3 (slotPaddingLeft * (x + 1) + (slotSize * x) + 5f, -slotPaddingTop * (y
+ 1) - (slotSize * y)- slotSize / 2.5f);

            slotRect.SetSizeWithCurrentAnchors
(RectTransform.Axis.Horizontal, slotSize *
InventoryManager.Instance.canvas.scaleFactor);

```

```

        slotRect.SetSizeWithCurrentAnchors
(RectTransform.Axis.Vertical, slotSize *
InventoryManager.Instance.canvas.scaleFactor );

        newSlot.transform.SetParent (this.transform);
        allSlots.Add (newSlot);
    }
}
// добавляем предмет в инвентарь (ваш К.0.)
public bool Additem(ItemScript item)
{
    Debug.Log("im in additem");
    Debug.Log (item.Item.MaxSize);
    //если в предмете только одна связка - закидываем предмет туда, в
ином случае проверяем, есть ли уже такой предмет в инвентаре и не полна ли эта
связка
    if (item.Item.MaxSize == 1) {
        return PlaceEmpty (item);
        //return true;
    }
    else
    {
        foreach (GameObject slot in allSlots)
        {
            Slot tmp = slot.GetComponent<Slot> ();

            if (!tmp.IsEmpty)
            {
                if (tmp.CurrentItem.Item.ItemName ==
item.Item.ItemName && tmp.IsAvailable)
                {
                    tmp.AddItem (item);
                    //emptySlot--;
                    return true;
                }
            }
        }
        //сюда он доходит только в том случае, если весь цикл просмотра
инвентаря прошел, и не нашел совпадений на предмет или же стака аналогов полны
        if (emptySlot > 0)
        {
            return PlaceEmpty (item);
        }
    }

    return false;
}

private void MoveInventory ()
{
    Vector2 mousePos;
    // устанавливаем инвентарь по центру мышки
    RectTransformUtility.ScreenPointToLocalPointInRectangle
(InventoryManager.Instance.canvas.transform as RectTransform, new
Vector3(Input.mousePosition.x -(inventoryRect.sizeDelta.x/2 *
InventoryManager.Instance.canvas.scaleFactor), Input.mousePosition.y +
(inventoryRect.sizeDelta.y/2*InventoryManager.Instance.canvas.scaleFactor)),
InventoryManager.Instance.canvas.worldCamera, out mousePos);

```

```

        transform.position =
InventoryManager.Instance.canvas.transform.TransformPoint (mousePos);
    }

    private bool PlaceEmpty(ItemScript item)
    {
        Debug.Log ("PlaceEmpty");
        if (emptySlot > 0)
        {
            foreach (GameObject slot in allSlots)
            {
                Slot tmp = slot.GetComponent<Slot> ();
                if (tmp.IsEmpty)
                {
                    tmp.AddItem (item);
                    //emptySlot--;
                    return true;
                }
            }
        }

        return false;
    }

    public void MoveItem(GameObject clicked)
    {
        CanvasGroup cg = clicked.transform.parent.GetComponent<CanvasGroup>
();

        if (cg != null && cg.alpha > 0 ||
clicked.transform.parent.parent.GetComponent<CanvasGroup>().alpha > 0)
        {
            InventoryManager.Instance.Clicked = clicked;

            if (!InventoryManager.Instance.MoveSlot.IsEmpty)
            {
                Slot tmp = clicked.GetComponent<Slot> ();

                if (tmp.IsEmpty)
                {
                    tmp.AddItems
(InventoryManager.Instance.MoveSlot.Items);
                    InventoryManager.Instance.MoveSlot.Items.Clear ();
                    Destroy (GameObject.Find ("Hover"));
                }
                else if (!tmp.IsEmpty &&
InventoryManager.Instance.MoveSlot.CurrentItem.Item.ItemName ==
tmp.CurrentItem.Item.ItemName && tmp.IsAvailable)
                {
                    MergeStacks (InventoryManager.Instance.MoveSlot,
tmp);
                }
            }

            else if (InventoryManager.Instance.From == null &&
clicked.transform.parent.GetComponent<Inventory>().isOpen &&
!Input.GetKey(KeyCode.LeftShift)) {

```



```

        //проверка на то, что слот куда мы кликнули не пустой
        if (!clicked.GetComponent<Slot> ().IsEmpty &&
!GameObject.Find("Hover"))
        {
            // получаем связь с предметом на который кликнули
            InventoryManager.Instance.From =
clicked.GetComponent<Slot> ();
            InventoryManager.Instance.From.GetComponent<Image>
().color = Color.gray;

            CreateHoverIcon ();
        }
    }
    else if (InventoryManager.Instance.To == null &&
!Input.GetKey(KeyCode.LeftShift))
    {
        InventoryManager.Instance.To =
clicked.GetComponent<Slot> ();
        Destroy (GameObject.Find("Hover"));
    }
    if (InventoryManager.Instance.To != null &&
InventoryManager.Instance.From != null)
    {
        //создаем временный стек для смены ну как в методе
пузырька крч
        if (!InventoryManager.Instance.To.IsEmpty &&
InventoryManager.Instance.From.CurrentItem.Item.ItemName ==
InventoryManager.Instance.To.CurrentItem.Item.ItemName &&
InventoryManager.Instance.To.IsAvailable)
        {
            MergeStacks (InventoryManager.Instance.From,
InventoryManager.Instance.To);
        }
        else
        {
            Slot.SwapItems (InventoryManager.Instance.From,
InventoryManager.Instance.To);
        }

        InventoryManager.Instance.From.GetComponent<Image>
().color = Color.white;
        InventoryManager.Instance.To = null;
        InventoryManager.Instance.From = null;
        Destroy (GameObject.Find ("Hover"));
    }
}

private void CreateHoverIcon()
{
    // каст префаба иконки в наш объект (для зависания иконки на мышке во
время хватания объекта)
    InventoryManager.Instance.HoverObject = (GameObject)Instantiate
(InventoryManager.Instance.iconPrefab);
    // получаем спрайт из слота на который мы кликнули и запикиваем его в
hoverobject
    InventoryManager.Instance.HoverObject.GetComponent<Image> ().sprite =
InventoryManager.Instance.Clicked.GetComponent<Image> ().sprite;
    //это просто для ясности

```

```

InventoryManager.Instance.HoverObject.name = "Hover";
// создание связи между трансформациями hover и clicked
RectTransform hoverTransform =
InventoryManager.Instance.HoverObject.GetComponent<RectTransform> ();
RectTransform clickedTransform =
InventoryManager.Instance.Clicked.GetComponent<RectTransform> ();

hoverTransform.SetSizeWithCurrentAnchors
(RectTransform.Axis.Horizontal, clickedTransform.sizeDelta.x);
hoverTransform.SetSizeWithCurrentAnchors
(RectTransform.Axis.Vertical, clickedTransform.sizeDelta.y);
//нужно удостовериться, что объект который зависает на мышке -
потомок нашего UI элемента, ну Canvas
InventoryManager.Instance.HoverObject.transform.SetParent
(GameObject.Find ("Canvas").transform, true);
//проверяем не накосячили ли с размерами, берем стандартные
InventoryManager.Instance.HoverObject.transform.localScale =
InventoryManager.Instance.Clicked.gameObject.transform.localScale;

InventoryManager.Instance.HoverObject.transform.GetChild
(0).GetComponent<Text> ().text = InventoryManager.Instance.MoveSlot.Items.Count >
1 ? InventoryManager.Instance.MoveSlot.Items.Count.ToString () : string.Empty;
}

private void PutItemBack()
{
    if (InventoryManager.Instance.From != null)
    {
        Destroy (GameObject.Find ("Hover"));
        InventoryManager.Instance.From.GetComponent<Image> ().color =
Color.white;
        InventoryManager.Instance.From = null;
    }
    else if (!InventoryManager.Instance.MoveSlot.IsEmpty)
    {
        Destroy (GameObject.Find("Hover"));

        foreach (ItemScript item in
InventoryManager.Instance.MoveSlot.Items)
        {
            InventoryManager.Instance.Clicked.GetComponent<Slot>
().AddItem (item);
        }

        InventoryManager.Instance.MoveSlot.ClearSlot ();
    }

    InventoryManager.Instance.selectStackSize.SetActive (false);
}

public void SplitStack()
{
    // когда жмем кнопку OK - панель разделения исчезает
    InventoryManager.Instance.selectStackSize.SetActive (false);
    if (InventoryManager.Instance.SplitAmount ==
InventoryManager.Instance.MaxStackCount)
    {
        MoveItem (InventoryManager.Instance.Clicked);
    }
}

```

```

        else if (InventoryManager.Instance.SplitAmount > 0)
        {
            InventoryManager.Instance.MoveSlot.Items =
InventoryManager.Instance.Clicked.GetComponent<Slot> ().RemoveItems
(InventoryManager.Instance.SplitAmount);
            CreateHoverIcon ();
        }
    }

    public void ChangeStackText(int i)
    {
        InventoryManager.Instance.SplitAmount += i;

        if(InventoryManager.Instance.SplitAmount < 0)
        {
            InventoryManager.Instance.SplitAmount = 0;
        }
        if (InventoryManager.Instance.SplitAmount >
InventoryManager.Instance.MaxStackCount)
        {
            InventoryManager.Instance.SplitAmount =
InventoryManager.Instance.MaxStackCount;
        }

        InventoryManager.Instance.stackText.text =
InventoryManager.Instance.SplitAmount.ToString ();
    }

    public void MergeStacks(Slot source, Slot destination)
    {
        int max = destination.CurrentItem.Item.MaxSize -
destination.Items.Count;
        // вычисляем количество переметов которые необходимо перенести, чтобы
в стаках не оказалось больше предметов чем в них может быть
        int count = source.Items.Count < max ? source.Items.Count : max;

        for (int i = 0; i < count; i++)
        {
            //берем по одному предмету из руки и добавляем в стак
            destination.AddItem (source.RemoveItem ());

            InventoryManager.Instance.HoverObject.transform.GetChild
(0).GetComponent<Text> ().text =
InventoryManager.Instance.MoveSlot.Items.Count.ToString();
        }
        if(source.Items.Count == 0)
        {
            source.ClearSlot ();
            Destroy (GameObject.Find ("Hover"));
        }
    }

    private IEnumerator FadeOut()
    {
        fadingOut = true;
        fadingIn = false;
        StopCoroutine ("FadeIn");

        float startAlpha = canvasGroup.alpha;

        float rate = 1.0f / fadeTime;

```

```

float progress = 0.0f;

while (progress < 1.0)
{
    canvasGroup.alpha = Mathf.Lerp (startAlpha, 0, progress);

    progress += rate * Time.deltaTime;

    yield return null;
}

canvasGroup.alpha = 0;
fadingOut = false;
//gameObject.SetActive (false);
canvasGroup.blocksRaycasts = false;
}

private IEnumerator FadeIn()
{
    fadingOut = false;
    fadingIn = true;
    StopCoroutine ("FadeOut");

    float startAlpha = canvasGroup.alpha;

    float rate = 1.0f / fadeTime;

    float progress = 0.0f;

    while (progress < 1.0)
    {
        canvasGroup.alpha = Mathf.Lerp (startAlpha, 1, progress);
        progress += rate * Time.deltaTime;

        yield return null;
    }
    canvasGroup.alpha = 1;
    fadingIn = false;
}
}

```

ПРИЛОЖЕНИЕ В2

Листинг программы. InventoryManager.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using System;
using System.Xml.Serialization;
using System.IO;

public class InventoryManager : MonoBehaviour {

    private static InventoryManager instance;
    //singleton pattern
    public static InventoryManager Instance
    {
        get
        {
            if(instance == null)
            {
                instance = FindObjectOfType<InventoryManager> ();
            }
            return instance;
        }
    }
    //в InventoryManager хранятся переменные, общие для всех
    инвентарей/сумок/трейдов и т.д.
    public GameObject slotPrefab;

    public GameObject iconPrefab;

    public GameObject itemObject;

    private  GameObject hoverObject;

    public GameObject HoverObject
    {
        get { return hoverObject;}
        set { hoverObject = value;}
    }

    public GameObject tooltipObject;
    public Text sizeTextObject;
    public Text visualTextObject;

    public Canvas canvas;

    private Slot from, to;

    public Slot From
    {
        get{ return from;}
        set { from = value;}
    }

    public Slot To
    {
```

```

        get{ return to;}
        set{ to = value;}
    }

    private GameObject clicked;

    public GameObject Clicked
    {
        get{ return clicked;}
        set{ clicked = value;}
    }

    public Text stackText;

    public GameObject selectStackSize;

    private int splitAmount;

    public int SplitAmount
    {
        get{ return splitAmount;}
        set{ splitAmount = value;}
    }

    private int maxStackCount;

    public int MaxStackCount {
        get { return maxStackCount;}
        set { maxStackCount = value;}
    }

    private Slot moveSlot;

    public Slot MoveSlot {
        get { return moveSlot;}
        set { moveSlot = value;}
    }

    public EventSystem eventSystem;

    private ItemContainer itemContainer = new ItemContainer();

    public ItemContainer ItemContainer {
        get {
            return itemContainer;
        }
        set {
            itemContainer = value;
        }
    }

    public void Start()
    {
        // получаем типы предметов, сами предметы из XML
        Type[] itemTypes = { typeof(Equipment), typeof(Weapon),
typeof(Consumable) };
        XmlSerializer serializer = new XmlSerializer (typeof(ItemContainer),
itemTypes);
        TextReader textReader = new StreamReader
(Application.streamingAssetsPath + "/Items.xml");
        itemContainer = (ItemContainer)serializer.Deserialize (textReader);
    }

```

```

        textReader.Close ();
    }

    public void SetStackInfo(int maxStackCount)
    {
        selectStackSize.SetActive (true);
        tooltipObject.SetActive (false);
        splitAmount = 0;
        this.maxStackCount = maxStackCount;
        stackText.text = splitAmount.ToString ();
    }

    public void Save()
    {
        GameObject[] inventories = GameObject.FindGameObjectsWithTag
("Inventory");

        foreach (GameObject inventory in inventories)
        {
            inventory.GetComponent<Inventory> ().SaveInventory ();
        }
    }

    public void Load()
    {
        GameObject[] inventories = GameObject.FindGameObjectsWithTag
("Inventory");

        foreach (GameObject inventory in inventories)
        {
            inventory.GetComponent<Inventory> ().LoadInventory ();
        }
    }
}

```


ПРИЛОЖЕНИЕ В3

Листинг программы. Slot.cs

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

public class Slot : MonoBehaviour, IPointerClickHandler
{
    //симулятор стэка в ACM
    private Stack<ItemScript> items;

    private CanvasGroup canvasGroup;

    public ItemType canContain;

    public Stack<ItemScript> Items
    {
        get { return items;}
        set { items = value;}
    }
    // тут будет храниться само количество стэков предмета
    public Text stackText;
    // просто спрайты на ресет
    public Sprite slotEmpty;

    public Sprite slotHighlight;
    // проверяем, свободен ли слот и можно ли в него что-то поместить
    public bool IsEmpty
    {
        get { return items.Count == 0; }
    }
    // проверяем, можно ли туда еще чего-нибудь запихнуть или слот полон как мой
    живот после универской пиццы
    public bool IsAvailable
    {
        get { return CurrentItem.Item.MaxSize > items.Count; }
    }
    // получаем тип предмета в слоте (peek - получение самого верхнего элемента
    в стеке)
    public ItemScript CurrentItem
    {
        get { return items.Peek ();}
    }

    void Awake()
    {
        items = new Stack<ItemScript> ();
    }

    // Use this for initialization
    void Start ()
    {
        RectTransform slotRect = GetComponent<RectTransform> ();
        RectTransform textRect = stackText.GetComponent<RectTransform> ();
        // больше форматирования)
    }
}
```

```

        int textScaleFactor = (int)(slotRect.sizeDelta.x * 0.60);
        stackText.resizeTextMaxSize = textScaleFactor;
        stackText.resizeTextMinSize = textScaleFactor;

        textRect.SetSizeWithCurrentAnchors (RectTransform.Axis.Horizontal,
slotRect.sizeDelta.x-10f);
        textRect.SetSizeWithCurrentAnchors (RectTransform.Axis.Vertical,
slotRect.sizeDelta.y+50f);

        if (transform.parent != null)
        {
            canvasGroup = transform.parent.GetComponent<CanvasGroup> ();
        }

    }

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

    }

    public void AddItem (ItemScript item)
    {
        if (IsEmpty)
        {
            transform.parent.GetComponent<Inventory> ().EmptySlot--;
        }

        items.Push (item);

        if (items.Count > 1)
        {
            stackText.text = items.Count.ToString ();
        }

        ChangeSprite (item.spriteNeutral, item.spriteHighlighted);
    }
    // меняем местами 2 предмета в инвентаре
    public void AddItems(Stack<ItemScript> items)
    {
        // непосредственная замена стака
        this.items = new Stack<ItemScript> (items);
        // замена счетчика стака
        stackText.text = items.Count > 1 ? items.Count.ToString () :
string.Empty;
        // смена спрайтов
        ChangeSprite (CurrentItem.spriteNeutral,
CurrentItem.spriteHighlighted);
    }

    private void ChangeSprite (Sprite neutral, Sprite highlight)
    {
        GetComponent<Image> ().sprite = neutral;

        SpriteState st = new SpriteState ();

        st.highlightedSprite = highlight;
        st.pressedSprite = neutral;
    }

```

```

        GetComponent<Button> ().spriteState = st;
    }

    private void UseItem()
    {
        if (!IsEmpty)
        {
            // удаляем верхний элемент стека items и вызываем метод Use от
туда же
            items.Peek ().Use (this);
            // если количество предметов больше одного - текст
перезаписывает количество предметов, если нет - строка становится пустой, аналог
if else только покороче
            stackText.text = items.Count > 1 ? items.Count.ToString () :
string.Empty;

            if (IsEmpty)
            {
                ChangeSprite (slotEmpty, slotHighlight);
                transform.parent.GetComponent<Inventory> ().EmptySlot++;
                //Inventory.EmptySlot++;
            }
        }
    }
    // очищает слот от всего что в нем есть, опять ваш КО
    public void ClearSlot()
    {
        items.Clear ();
        ChangeSprite (slotEmpty, slotHighlight);
        stackText.text = string.Empty;
        if (transform.parent != null)
        {
            transform.parent.GetComponent<Inventory> ().EmptySlot++;
        }
    }

    public Stack<ItemScript> RemoveItems (int amount)
    {
        Stack<ItemScript> tmp = new Stack<ItemScript> ();

        for (int i = 0; i < amount; i++)
        {
            tmp.Push (items.Pop ());
        }

        stackText.text = items.Count > 1 ? items.Count.ToString () :
string.Empty;

        return tmp;
    }

    public ItemScript RemoveItem()
    {
        ItemScript tmp;

        tmp = items.Pop ();

        stackText.text = items.Count > 1 ? items.Count.ToString () :
string.Empty;

        return tmp;
    }

```

```

    }

    // интерфейс для работы с инвентарем по клику
    public void OnPointerClick (PointerEventData eventData)
    {
        if (eventData.button == PointerEventData.InputButton.Right &&
!GameObject.Find ("Hover") && canvasGroup.alpha > 0) {
            UseItem ();
        } else if (eventData.button == PointerEventData.InputButton.Left &&
Input.GetKey (KeyCode.LeftShift) && !IsEmpty && !GameObject.Find ("Hover"))
        {
            Vector2 position;

            RectTransformUtility.ScreenPointToLocalPointInRectangle
(InventoryManager.Instance.canvas.transform as RectTransform, Input.mousePosition,
InventoryManager.Instance.canvas.worldCamera, out position);
            // активируем размер стака
            InventoryManager.Instance.selectStackSize.SetActive (true);

            InventoryManager.Instance.selectStackSize.transform.position =
InventoryManager.Instance.canvas.transform.TransformPoint (position);

            InventoryManager.Instance.SetStackInfo (items.Count);
        }
    }

    public static void SwapItems(Slot from, Slot to)
    {
        ItemType movingType = from.CurrentItem.Item.ItemType;

        if (to != null && from != null)
        {
            bool calcStats = from.transform.parent ==
CharacterPanel.Instance.transform || to.transform.parent ==
CharacterPanel.Instance.transform;

            if(movingType == ItemType.TWOHAND &&
CharacterPanel.Instance.OffHandSlot.IsEmpty || movingType == ItemType.MAINHAND)
            {
                movingType = ItemType.GENERICWEAPON;
            }
            if (to.canContain == ItemType.GENERIC || movingType ==
to.canContain)
            {
                if (movingType != ItemType.OFFHAND ||
(CharacterPanel.Instance.WeaponSlot.IsEmpty ||
CharacterPanel.Instance.WeaponSlot.CurrentItem.Item.ItemType != ItemType.TWOHAND))
                {
                    Stack<ItemScript> tmpTo = new Stack<ItemScript>
(to.Items);

                    to.AddItems (from.Items);
                    if (tmpTo.Count == 0)
                    {
                        to.transform.parent.GetComponent<Inventory>
().EmptySlot--;
                        from.ClearSlot ();
                    }
                }
            }
        }
    }

```

```

        }
        else
        {
            from.AddItem (tmpTo);
        }
    }
}
if(calcStats)
{
    CharacterPanel.Instance.CalcStats ();
}
}
}
}

```

ПРИЛОЖЕНИЕ В4

Листинг программы. Interactable.cs

```
using UnityEngine;
using System.Collections;

public class Interactable : MonoBehaviour
{
    public UnityEngine.AI.NavMeshAgent playerAgent;
    private bool hasInteracted;

    public virtual void MoveToInteraction(UnityEngine.AI.NavMeshAgent
playerAgent)
    {
        hasInteracted = false;
        this.playerAgent = playerAgent;
        playerAgent.destination = this.transform.position;

    }

    void Update()
    {
        if (! hasInteracted && playerAgent != null &&
!playerAgent.pathPending)
        {
            if(playerAgent.remainingDistance <=
playerAgent.stoppingDistance)
            {
                Interact ();
                hasInteracted = true;
            }
        }

        InteractUpdate ();
    }

    public virtual void InteractUpdate()
    {
    }

    public virtual void Interact()
    {
        Debug.Log ("Interact with the base class");
        //inventory.Additem (other.GetComponent<Item> ());
    }
}
```

ПРИЛОЖЕНИЕ В5

Листинг программы. PickupItem.cs

```
using UnityEngine;
using System.Collections;

public class PickupItem : Interactable
{
    public WorldInteraction worldInteraction;

    public override void Interact()
    {
        Debug.Log ("Interact with item");

        worldInteraction.AddItemsToInventory();

        //DestroyObject (worldInteraction.interactedObject);

        //newObject.GetComponent<WorldInteraction> ().AddItemsToInventory();
    }
}
```