Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)
Институт фундаментального образования
Кафедра «интеллектуальных информационных технологий»

Оценка _____

Руководитель курсового
проектирования__Чернойван В.А__

Члены комиссии_____
_____

Дата защиты_____

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту

по теме: РАЗРАБОТКА КОМПЬЮТЕРНОЙ ИГРЫ "2DFG"
по дисциплине: Объектно-ориентированное программирование

Студенты:
Бабичева Татьяна Юрьевна (ФО-260001) _____
(ФИО) (Подпись)
Мезенцева Александра Денисовна (ФО-260001) _____
(ФИО) (Подпись)
Касимов НурАухатович (ФО-260003) _____
(ФИО) (Подпись)
Зарывных Павел Сергеевич (ФО-260001) _____
(ФИО) (Подпись)
Ренев Денис Андреевич (ФО-260001) _____
(ФИО) (Подпись)

Екатеринбург
2018

СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

На сегодняшний день рынок развлечений является одним из самых активных. С момента начала информационно-технической революции мир стремительно движется в будущее, создавая все более совершенные компьютерные системы, чтобы облегчить жизнь человека, а также занять его досуг.

Можно сказать, что игры – это определенный вид искусства, схожий с другими зрелищными жанрами. Игры могут нести не только развлекательный характер, их функционал достаточно широк: развитие, образование, экономика, психология. Игра может заставить задуматься, эмоционально переживать, поднимать серьезные глобальные или психологические вопросы. С помощью игры можно найти и пути их решения. Другими словами, игры – это современный вид искусства, поклонников у которого достаточно много.

Именно поэтому нами было принято решение заняться созданием игрового проекта.

# 1 ПОСТАНОВКА ЗАДАЧИ

Проект направлен на создание развлекательного продукта для пользователей, которые любят проводить своё свободное время, играя в различные видеоигры.

Целевая аудитория: игроки, которые любят видеоигры в жанре RPG или 2D-платформеры.

Целью нашей работы была разработка 2D-платформера в жанре фэнтези с элементами RPG. Для достижения этой цели необходимо было решить следующие задачи:

1. реализовать движение и взаимодействие с объектами;
2. разработать искусственный интеллект;
3. создать инвентарь и магазин;
4. разработать системы диалогов и квестов;
5. реализовать систему развития персонажа;
6. реализовать сохранение игрового процесса;
7. протестировать и отладить разработанную игру.

Игра реализована с помощью игрового движка Unity, на языке программирования C#.

## 2 ПЛАН РАБОТ

Перед началом работ был составлен план действий. Распределение задач по исполнителям приведено в таблицах ниже.

Таблица 1. План работы Бабичевой Татьяны

| № | Задачи | Сроки |
|---|---|---|
| 1 | Разработать концепцию игры и написать сценарий сюжетных квестов | 07.03 – 14.03 |
| 2 | Создать и анимировать игрового персонажа | 15.03 – 21.03 |
| 3 | Разработать диалоговую систему | 22.03 – 11.04 |
| 4 | Разработать систему искусственного интеллекта для вражеских NPC | 12.04 – 25.04 |
| 5 | Создать игровые уровни | 26.04 – 3.05 |
| 6 | Добавить звуковые и анимационные эффекты | 04.05 – 07.05 |
| 7 | Оптимизировать и протестировать код, устранить максимально возможное количество ошибок | 08.05 – 20.05 |

Таблица 2. План работы Мезенцевой Александры

| № | Задачи | Сроки |
|---|---|---|
| 1 | Разработать концепцию инвентаря персонажа | 07.03 – 14.03 |
| 2 | Разработать интерфейс инвентаря | 15.03 – 21.03 |
| 3 | Разработать реализацию магазина | 22.03 – 11.04 |
| 4 | Разработать интерфейс магазина | 12.04 – 25.04 |
| 5 | Реализовать использование предметов из инвентаря | 26.04 – 3.05 |
| 6 | Протестировать и отладить разработанный функционал | 04.05 – 19.05 |

Таблица 3. План работы Касимова Нура

| № | Задачи | Сроки |
|---|---|---|
| 1 | Разработка общей концепции игрового мира и квестов, написание побочных квестов – примеров | 07.03 – 21.03 |
| 2 | Проектирование квестовой системы | 22.03 – 28.03 |
| 3 | Базовая реализация квестовой системы и первого простейшего квеста | 29.03 – 13.04 |
| 4 | Модернизация квестовой системы: уведомления и взаимодействие с пользователем | 14.04 – 20.04 |
| 5 | Модернизация квестовой системы: услужение квестов | 21.04 – 04.05 |
| 6 | Модернизация квестовой системы: распределение квеста на несколько игровых сцен | 05.05 – 18.05 |
| 7 | Общие улучшения и поправки, исправления ошибок | 18.05 – 28.05 |

Таблица 4. План работы Зарывных Павла

| № | Задачи | Сроки |
|---|---|---|
| 1 | Реализовать систему прокачки | 15.03 – 11.04 |
| 2 | Сделать увеличение количества здоровья | 12.04 – 25.04 |
| 3 | Сделать увеличение силы атаки | 26.04 – 3.05 |
| 4 | Сделать проверку на получение уровня игроком | 04.05 – 07.05 |
| 5 | Реализовать интерфейс системы прокачки | 08.05 – 20.05 |

Таблица 5. План работы Ренева Дениса

| № | Задачи | Сроки |
|---|---|---|
| 1 | Реализовать главное игровое меню | 22.03 – 11.04 |
| 2 | Реализовать меню паузы | 12.04 – 25.04 |
| 3 | Разработать экономическую систему | 26.04 – 7.05 |
| 4 | Разработать сохранение и загрузку | 08.05 – 20.05 |

# 3 ОПИСАНИЕ РЕАЛИЗАЦИИ ПРОЕКТА

## 3.1 ОПИСАНИЕ РАБОТЫ БАБИЧЕВОЙ ТАТЬЯНЫ

### 3.1.1 РАЗРАБОТКА КОНЦЕПЦИИ ИГРЫ И НАПИСАНИЕ СЦЕНАРИЯ СЮЖЕТНЫХ КВЕСТОВ

Перед тем, как начать разрабатывать игру, необходимо было продумать её концепцию и сюжет, который будет главной составляющей данной игры.

Таким образом, был решено создать 2D платформер с элементами RPG, одними из которых являются: система прокачки, инвентарь, магазин, а также диалоговая и квестовая системы.

Примечание: в дальнейшем «квестом» является один из основных этапов игры, представляющий собой интерактивную историю с главным героем, управляемым игроком.

В качестве времени действия было выбрано Средневековье. Для разнообразия игрового процесса было решено добавить элементы фэнтези, которые могут быть реализованы за пределами демонстрационной версии.

На основе данной концепции был написан сценарий сюжетных квестов для демонстрационной версии игры, в котором используются все перечисленные выше системы.

### 3.1.2 СОЗДАНИЕ И АНИМИРОВАНИЕ ПЕРСОНАЖА

Внешний вид персонажа был разработан в соответствии с тематикой Средневековья, на которую делался упор при разработке концепции игры. За анимирование игрового персонажа отвечает класс CharacterAnimationController.

Данный класс отвечает за анимацию стойки на месте, поворотов на 180 градусов, ходьбы, прыжка, атаки, получения урона и смерти персонажа.

Для того, чтобы следить за перемещением персонажа по карте, используется класс CameraFollow, который также позволяет избегать перемещения главной камеры за пределы игровой карты.

### 3.1.3 РАЗРАБОТКА ДИАЛОГОВОЙ СИСТЕМЫ

Диалоговая система является основой квестовой системы, т.к. она позволяет выводить различные сообщения на экран (например, элементы обучения, диалоги между персонажами, различные уведомления и т.д.). Данные сообщения являются объектами класса Dialogs, который содержит в себе имя говорящего и фразы, сказанные им. При помощи класса DialogTrigger происходит запуск диалога, во время которого персонаж обездвижен. Класс DialogManager отвечает за отображение сообщений, переключение между ними, а также за завершение диалога. Интерфейс показан на рисунке 1 в Приложении А1.

### 3.1.4 РАЗРАБОТКА СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА ДЛЯ ВРАЖЕСКИХ NPC

Для усложнения игры было принято решение добавить враждебных по отношению к игроку персонажей. Для анимирования таких персонажей используется класс Enemy, в котором прописаны функции, отвечающие за передвижение вражеского NPC по карте, повороты на 180 градусов, произведение дальней и ближней атаки, получение урона и смерть. Для обнаружения и потери цели используется метод Raycasting() в классе EnemySight.

Вражеский персонаж имеет несколько состояний, между которыми он может переходить с помощью интерфейса IEnemyStates. Данными состояниями являются:

- патрулирование (класс PatrolState);
- остановка между патрулями (класс IdleState);
- дальняя атака (класс TargetInRangeState совместно с методом ThrowObject из класса Enemy, отвечающим за инициализацию объектов, использующихся для проведения дальней атаки, например, стрел);
- ближний бой (класс MeleeAttackState).

На рисунках 2, 3 Приложения А1 продемонстрирована дальняя атака вражеского персонажа и его смерть.

### 3.1.5 СОЗДАНИЕ ИГРОВЫХ УРОВНЕЙ

Для разнообразия игрового процесса было решено создать различные локации, различающиеся по своему назначению: леса, для получения эстетического наслаждения от видов природы, пещеры и замки для изучения и поиска секретов и сокровищ, города и деревни, для взаимодействия с различными NPC и т.д.

Для перехода между сценами был реализован класс SwitchScene, который также используется в квестовой системе для определения того, можно ли покинуть сцену, имея определённый квест.

### 3.1.6 ДОБАВИТЬ ЗВУКОВЫЕ И АНИМАЦИОННЫЕ ЭФФЕКТЫ

Для дополнения игры звуковым сопровождением был использован аудио-менеджер, который управляет фоновой музыкой на уровнях и звуковыми эффектами, а также способен регулировать их громкость. С ним можно ознакомиться в классе AudioManager.

### 3.1.7 ОПТИМИЗИРОВАТЬ И ПРОТЕСТИРОВАТЬ КОД, УСТРАНИТЬ МАКСИМАЛЬНО ВОЗМОЖНОЕ КОЛИЧЕСТВО ОШИБОК

В целях оптимизации графики была добавлена функция OnBecameInvisible() в классе Throws для оружия дальней атаки, которая позволяет уничтожать объекты за пределами карты для того, чтобы при создании дополнительных экземпляров объектов они не заполняли память и не нагружали систему.

Для игрового персонажа и вражеских NPC были исправлены ошибки анимации атаки, смерти и прыжков в классах DamageAnimationFix и EnemyDamageAnimationFix.

Для того, чтобы вражеский и игровой персонаж не толкали на сцене друг друга, но сталкивались с коллизией сцен (или любыми другими), был создан класс IgnoreCharactersCollision, позволяющий игнорировать коллизию выбранного объекта.

Также была проведена оптимизация движения вражеского персонажа в функции Move() класса Enemy, в которой умножение на deltaTime нужно, чтобы

не возникало быстрого вычисления скорости передвижения AI на процессорах с большей тактовой частотой и медленного на процессорах с меньшей (в противном случае на разных процессорах скорость AI будет разной).

## 3.2 ОПИСАНИЕ РАБОТЫ МЕЗЕНЦЕВОЙ АЛЕКСАНДРЫ

Мезенцевой А.Д. была поручена разработка взаимодействия персонажа с игровыми предметами. Это взаимодействие принимает две основные формы: инвентарь, т.е. движимое имущество персонажа, и магазин, т.е. потенциальное имущество персонажа. Проще говоря, нужно было добиться того, чтобы персонаж мог подбирать, покупать и использовать предметы.

### 3.2.1 РАЗРАБОТКА ИНВЕНТАРЯ

Прежде чем говорить об инвентаре, следовало определиться с тем, что такое предмет и для чего он предназначен. Поскольку предполагалось, что персонаж сражается с врагами, было очевидно, что понадобятся такие предметы как доспехи и оружие. Поскольку кроме драк в игре мало что есть (это не плохо, это особенность жанра), все остальные предметы можно считать второстепенными, принципиально равнозначными.

Из этих соображений в игре введено подразделение всех предметов на восемь типов. Первая группа типов – оружие: одноручное, двуручное и метательное. Вторая группа – доспехи: шлемы, панцири, щиты и плащи. При этом понятно, что плащ практически не защищает, но уж больно красиво смотрится. Восьмой тип – квестовые предметы, т.е. все остальные предметы, которые попадаются по ходу игры.

В программной реализации это разделение воплощено при помощи перечислимого типа ItemType. Всякие предмет обладает такими характеристиками: тип, цена, вес, атака, защита, название и картинка. При этом показатель атака отличается от нуля только у предметов, которые относятся к типу оружие. Аналогично, защита отличается от нуля только у доспехов.

Для инвентаря центральными являются две идеи: единичность и вытеснение. Под единичностью понимается следующее: персонаж может иметь

только один предмет каждого типа. С этой идеей тесно связано вытеснение: если персонаж получает предмет, нарушающий единичность, то в инвентаре сохраняется более полезный из двух. Таким образом, если герой находит хорошее оружие, он выбрасывает плохое.

Технически инвентарь реализован как словарь, в котором ключами являются типы, а значениями предметы. Это позволяет реализовать единичность, но не вытеснение. Чтобы реализовать вытеснение, понадобилось написать класс Inventory как обертку словаря и написать методы для добавления, удаления и использования предметов. Для того чтобы при вытеснении предметы можно было сравнить, класс Item реализует интерфейс IComparable. При этом оружие сравнивается по показателю атака, доспехи по показателю защиты, а все остальные предметы по цене. Отношения классов проиллюстрированы на рисунке 11 Приложения А2.

Был разработан пользовательский интерфейс инвентаря, позволяющий просмотреть содержимое, удалить предмет или использовать его. В основе реализации интерфейса лежат стандартные классы Unity, понадобилось только написать вспомогательный класс InventorySlot, при помощи которого организовано взаимодействие элементов управления в одном слоте инвентаря. Внешний вид пользовательского интерфейса можно увидеть на рисунке 4 Приложения А1.

### 3.2.2 РАЗРАБОТКА МАГАЗИНА

Основной принцип устройства магазина схож с устройством инвентаря, разница заключается в том, что предметов одного типа в магазине может быть много. Поэтому класс Shop реализован как обертка словаря с методами выбора предмета и покупки. Ключами в словаре выступают типы предметов, а значениями – наборы предметов.

При реализации пользовательского интерфейса оказалось нужным запоминать для каждого типа предметов, какой предмет в наборе просматривался пользователем последним. Поэтому был разработан вложенный класс

Shop.Category, означающий категорию товаров в магазине с указанием текущего. Экземпляры этого класса и хранятся в словаре. Взаимоотношения классов, реализующих магазин, можно увидеть на рисунке 12 в Приложении А2.

Пользовательский интерфейс реализован при помощи стандартных компонентов Unity, его внешний вид показан на рисунке 5 Приложения А1.

## 3.3 ОПИСАНИЕ РАБОТЫ КАСИМОВА НУРА

Основной задачей в проекте являлась разработка одной из важнейших частей игры – квестовой системы. Она должна быть универсальной и реализовывать все возможные квесты предусмотренные концепцией. Кроме того она должна взаимодействовать практически со всеми другими системами, реализованными при разработке игры.

### 3.3.1 ОБЩАЯ КОНЦЕПЦИЯ ИГРОВОГО МИРА И КВЕСТОВ

Игровой мир состоит из локаций, каждая из которых в свою очередь разбита на небольшие сцены. Задачей игрока является пройти от стартовой сцены до конечной, для этого ему придётся выполнять квесты – некоторые препятствия для перехода на следующую сцену. Для выполнения текущего квеста игроку может быть необходимо вернуться на пройденные ранее сцены.

Прежде чем приступить к разработке квестовой системы, решено было написать несколько сценариев квестов максимально различного типа, на основе которых были сформированы требования к квестовой системе:

1. Универсальность: система должна создавать условия для прохождения квестов всех типов, предусмотренных игровым сценарием.
2. Преждевременное выполнение этапа квеста должно быть невозможно.
3. Уведомления игрока о стадиях прохождения квеста является обязательным.
4. Возможность отслеживать предметы инвентаря и уровень прокачки игрового персонажа.

### 3.3.2 БАЗОВАЯ РЕАЛИЗАЦИЯ КВЕСТОВОЙ СИСТЕМЫ

В целом, с точки зрения реализации, квест – это точки на игровом мире, которые могут быть выстроены в определенную иерархию и должны быть пройдены игроком для завершения квеста. Под пройденной точкой подразумевается точка, с которой успешно взаимодействовал игровой персонаж, причем для успешного выполнения родительской точки необходимо прохождение дочерних.

В базовой реализации, квестом являлся класс Quest, позже переименованный в QuestInfo, и содержавший основную информацию о квесте:

1. IsComplete – равен true если квест завершён и false если ещё нет.
2. Name – название квеста.
3. Краткое и полное описание в полях ShortDescription и FullDescription соответственно.
4. Points – массив точек QuestPoint, которые должны быть пройдены.

Кроме того, существует метод TryComplete(), проверяющий выполнение всех точек и обновляющий статус. В свою очередь точкой квеста является класс QuestPointкоторая в базовой реализации имела единственное свойство IsComplete сообщавшая статус квеста (true–пройден, false–нет), а также поле childPoints хранящая дочерние точки. За создание экземпляра класса Questотвечал класс QuestManager, позже переменный в StartQuestPoint. Текущие и завершенные квесты хранятся в ассоциативных массивах (словарях) CurrentQuestsи CompletedQuests соответственно.

### 3.3.3 МОДЕРНИЗАЦИЯ КВЕСТОВОЙ СИСТЕМЫ

В ходе разработки игры и по мере готовности других систем, квестовая система постоянно модернизировалась и развивалась. Это происходило в несколько этапов.

На первом этапе была реализована выдача квестов сопровождающаяся диалогом. Для этого была использована диалоговая система нашей разработки и в StartQuestPoint были внесены соответствующие изменения. Позже были

добавлены уведомления, предназначенные для сообщения игроку о ходе выполнения квеста после взаимодействия с той или иной точкой. Пример уведомления показан на Рисунке 9 Приложения А1.

На втором этапе началась реализация более сложных квестов. После нескольких изменений в конечном варианте в QuestPoint были введены так называемые требования, которые должны быть выполнены, чтобы взаимодействие прошло успешно. Всего было реализовано три вида требований:

1. RequirementToInventory – требование к наличию того или иного предмета в инвентаре игрового персонажа.
2. RequirementToSkill – требование к уровню прокачки персонажа.
3. ChildQuestPoints – требование к выполнению дочерних точек. Как было описано ранее, изначально дочерние точки хранились в поле childPoints, однако на этом этапе они были вынесены в требование.

Все эти классы реализуют интерфейс IRequirement требующий наличия свойства Notificationи метода IsComplete() проверяющая выполнение требования и в зависимости от удовлетворенности условий, возвращает trueили false, а класс QuestPoint при взаимодействии отныне проверяет наличие требований и их выполнение.

Помимо этого, был реализован класс QuestActivators и структура Activators, использующаяся в квестовой системе для задания условий активации взаимодействия с точкой квеста, например, нажатие клавиши “E” или движение влево.

На завершающем этапе была решена задача распределения квеста на несколько игровых сцен. Для этого введён класс QuestManager, который при каждой загрузке сцены подготавливает точки, тех квестов, которые находятся в массиве CurrentQuests у класса PlayerQuests и переносятся с цены на сцену с помощью системы сохранения. Теперь нет необходимости указывать все точки квеста в QuestInfo, достаточно указать лишь конечный, который должен находится на той же сцене, где происходит выдача квеста, и она же будет

являться корневой. BQuestPoint добавлено свойство QuestName по которому точка будет находиться и задаваться соответствующий квест в поле Quest.

## 3.4 ОПИСАНИЕ РАБОТЫ ЗАРЫВНЫХ ПАВЛА

### 3.4.1 РЕАЛИЗОВАТЬ СИСТЕМУ ПРОКАЧКИ

Для реализации данной задачи я разбил ее на подзадачи:

- сделать полностью функциональный пользовательский интерфейс для системы прокачки,
- сделать увеличение количества здоровья,
- сделать увеличение силы атаки,
- сделать увеличение количества интеллекта,
- силы и здоровья,
- сделать проверку на получения уровня игроком.

### 3.4.2 СДЕЛАТЬ УВЕЛИЧЕНИЕ КОЛИЧЕСТВА ЗДОРОВЬЯ

Для решения данной задачи в классе UpdateSystem написан метод FunctionHels(), который подсчитывает максимальное количество здоровья персонажа. В метод отправляются переменная health. В ней лежит количество очков, которые по мере развития персонажа игроком увеличиваются. Каждое очко health добавляет игроку 25 единиц здоровья.

### 3.4.3 СДЕЛАТЬ УВЕЛИЧЕНИЕ СИЛЫ АТАКИ

Для решения данной задачи в классе UpdateSystem написан метод DamagePhysical(), который подсчитывает количество урона игрока. В метод приходит значение силы персонажа strength и урон от оружия в руке персонажа. Сила персонажа умножается на коэффициент и добавляется урон оружия персонажа.

### 3.4.4 СДЕЛАТЬ ПРОВЕРКУ НА ПОЛУЧЕНИЯ УРОВНЯ ИГРОКОМ

Для решения данной задачи в классе UpdateSystem написан методFunctionPoint(). В нем происходит подсчет остаточного опыта для достижения уровня игроком и проверяется, набрал ли игрок достаточное

количество очков опыта для достижения нового уровня. В метод приходит опыт, который игрок получил за квесты или за уничтожения противника. В переменную endExp записывается остаточный опыт. В переменную scaleexp записывается текущий опыт игрока. В переменную scaleexpfull записывается количество опыта, которое должен набрать игрок. При условии, что scaleexp больше или равен scaleexpfull то игрок получит очки прокачки Point в количестве 5 очков, scaleexpfull значения переменной возрастают, а значения переменной scaleexp равно нулю или значению, на которое превысило переменную scaleexpfull.

### 3.4.5 СДЕЛАТЬ ИНТЕРФЕЙС СИСТЕМЫ ПРОКАЧКИ

Для решения данной задачи в классе CharacterAnimationController в методе Update() была написана проверка на нажатие клавиши K. Был создан класс WindowPumping. В нем написаны методы ShowWindow() и ConcealWindow(). Метод ShowWindow() отвечает за проигрывание анимации, чтобы окно системы прокачки появилось на экране.

Метод ConcealWindow() отвечает за проигрывание анимации, чтобы окно системы прокачки исчезло с экрана. Также был создан класс Button. В нем написаны методы Health(), Strength(), Magic(), Intellegnce() и Exit(). Каждый метод привязан к кнопке окна системы прокачки. При нажатии на кнопку окна системы прокачки будет срабатывать привязанный к ней метод. Был создан класс ConclusionParameters и в нем написан метод DataOutput(), который выводит характеристики персонажа, количество оставшиеся очков прокачки и количество опыта, оставшегося до нового уровня, на экран окна системы прокачки. Чтобы вызвать окно системы прокачки, игроку нужно нажать на клавишу K. После вызовется метод ShowWindow() из класса WindowPumping, который выведет окно системы прокачки на экран игрока. Для повышения характеристик персонажа в окне системы прокачки реализованы кнопки, к которым привязаны методы Health(), Strength(), Magic(), Intellegnce() и Exit() из класса Button.

Чтобы выйти из окна системы прокачки, игроку нужно нажать на кнопку выхода, тогда вызовется метод ConcealWindow() из класса WindowPumping и оно

уйдет с экрана персонажа. Также в окне есть текстовые подсказки, какая кнопка за что отвечает. С помощью метода DataOutput() в классе ConclusionParameters вы водится характеристики персонажа.

### 3.4.6 СДЕЛАТЬ УВЕЛИЧЕНИЕ ИНТЕЛЛЕКТА, СИЛЫ И ЗДОРОВЬЯ

Для решения этой задачи в классе Button написаны методы Health(), Strength(), Magic(), Intellegnce(). Если игрок нажимает на одну из кнопок, к которой привязаны эти методы в окне системы прокачки, то вызывается метод PumpingSkills() из класса SystemPumping. В методе происходит проверка на наличие очков прокачки: если они есть, то происходит добавления очков к определённой переменной Health, Strength, Magic или Intellegnce в зависимости от того, на какую кнопку нажал пользователь. После добавления очков из переменной Point вычитается одно очко прокачки и все изменения выводятся на экран с помощью метода DataOutput() в классе ConclusionParameters.

## 3.5 ОПИСАНИЕ РАБОТЫ РЕНЕВА ДЕНИСА

### 3.5.1 РЕАЛИЗАЦИЯ ГЛАВНОГО ИГРОВОГО МЕНЮ

Для того, чтобы пользователь всегда мог по желанию начать новую игру или продолжить играть в уже сохранённую, было решено сделать главное меню, описанное в классе UI_Manager. В данном классе используются следующие методы:

- NewGame() используется для создания новой игры;
- Load()используется для загрузки последней сохранённой игры;
- Exit() отвечает за выход из игры.

Интерфейс главного меню демонстрирует рисунок 7 в Приложении А1.

### 3.5.2 РЕАЛИЗАЦИЯ МЕНЮ ПАУЗЫ

Чтобы пользователь мог в любое время остановить, сохранить или загрузить игру, было решено добавить меню паузы, описанное в классе PauseMenu.В данном классе используются следующие методы:

- Update() отвечает за проверку запуска паузы, а также за блокировку движений персонажа и всего игрового мира, если пауза активна;
- Resume() используется для продолжения игры, т.е. для деактивации паузы;
- MainMenu() позволяет игроку вернуться в главное меню;
- Exit() отвечает за выход из игры.

Интерфейс меню паузы продемонстрирован на Рисунке 8 в Приложении А1.

### 3.5.3 РАЗРАБОТКА ЭКОНОМИЧЕСКОЙ СИСТЕМЫ

В целях логичного прохождения игры, следовало продумать её экономику: покупку/продажу предметов, получение/использование очков прокачки и т.д.

Для этого определённым образом был выработан способ баланса в игре, который определяет:

- сколько монет может выпасть в классе MoneyGeneration, отвечающем за генерацию монет в сундуках и золота, выпадающего из побежденных врагов;
- сколько будет стоить оружие и броня в магазинах в зависимости от её характеристик;
- в каких частях игры можно купить особую броню и по какой цене;
- сколько предметов можно переносить с собой в инвентаре;
- сколько очков прокачки или опыта можно получить за выполнение заданий, убийство врагов и т.д.

### 3.5.4 РАЗРАБОТКА СОХРАНИЯ И ЗАГРУЗКИ

Поскольку наша игра рассчитана на многочасовое прохождение, для нас было важно создать систему сохранений, которая позволит игроку не потерять свой прогресс. Данные функции должны быть расположены в главном меню и меню паузы для удобства пользователя. Также для быстрого сохранения и загрузки можно настроить горячие клавиши.

Для того, чтобы из главного меню или меню паузы можно было загрузить сохранённую игру или сохранить прогресс, было решено создать соответствующие функции. Для взаимодействия с данными функциями были

созданы такие элементы интерфейса, как кнопки Load и Save (Приложение1. Рисунок 8).

В меню паузы, при нажатии на Save, сохраняются характеристики персонажа в файл «player.sav». Запись в файл выполняет метод SavePlayer() в классе SaveLoadManager.

С помощью [Serializable] происходит сериализация для того, чтобы объект можно было воссоздать при необходимости.

При старте игры, пользователь может загрузить ранее сохраненную игру. Для этого в меню паузы нажимаем клавишу Load. Происходит загрузка данных при помощи метода Load()  в классе SystemPumping из файла «player.sav».

# ЗАКЛЮЧЕНИЕ

В результате хорошо спланированных и своевременно произведенных работ дружного коллектива единомышленников удалось с минимальными потерями разработать играбельный прототип приложения. Игрок может водить персонажа по миру, разговаривать с мирными жителями, побеждать врагов, добывать и покупать предметы, повышать характеристики.

Поставленные задачи в основном решены в срок. Программный код написан, откомментирован, отлажен, оттестирован и задокументирован. По итогам работы составлен отчет и подготовлено публичное выступление с презентацией.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Документация Unity [Электронный ресурс] – Режим доступа: https://docs.unity3d.com/ru/530/Manual/
2. Nystrom R. Game Programming Patterns [Электронный ресурс] – Режим доступа: http://gameprogrammingpatterns.com/
3. Хокинг Д. Unity в действии. Мульти платформенная разработка на C# / Хокинг Джозеф. – СПб. Питер, 2016 г. – 336 с.

ПРИЛОЖЕНИЕ А1. СКРИНШОТЫ


Рисунок 1. Диалоговая система

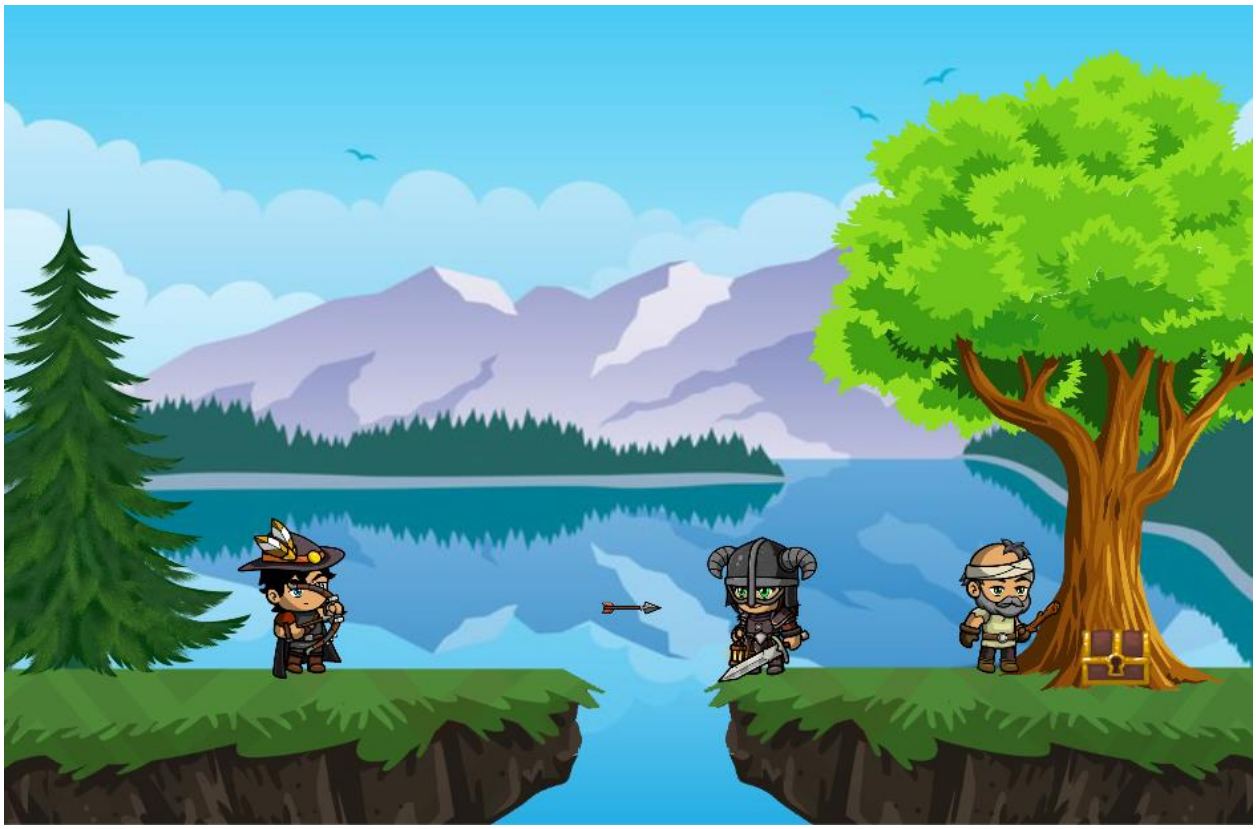

Рисунок 2. Дальняя атака, производимая вражеским NPC

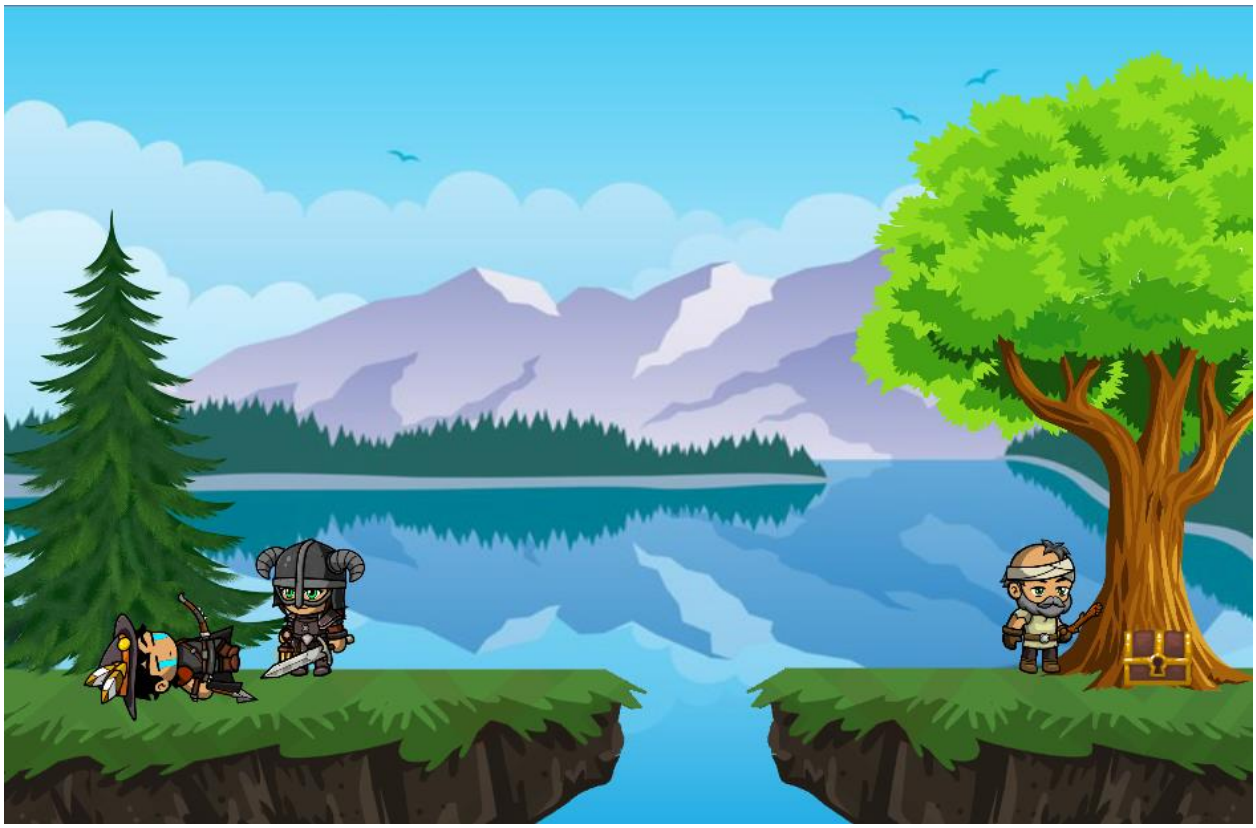

Рисунок 3. Враг убит

Рисунок 4. Интерфейс инвентаря

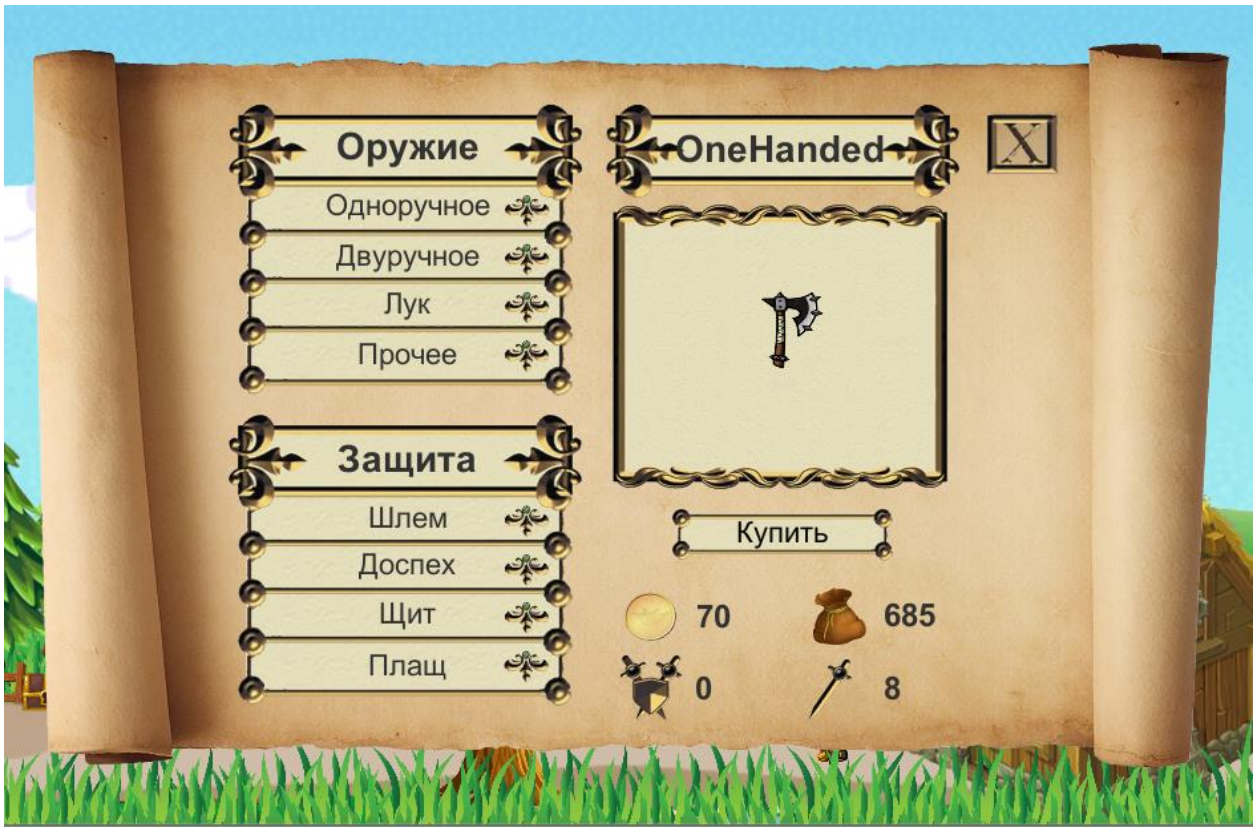


Рисунок 5. Интерфейс магазина


Рисунок 6. Интерфейс системы прокачки


Рисунок 7. Игровое меню


Рисунок 8. Меню паузы

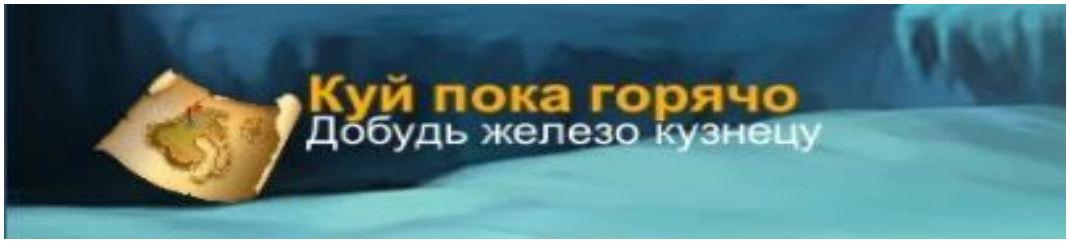


Рисунок 9. Пример квестового уведомления

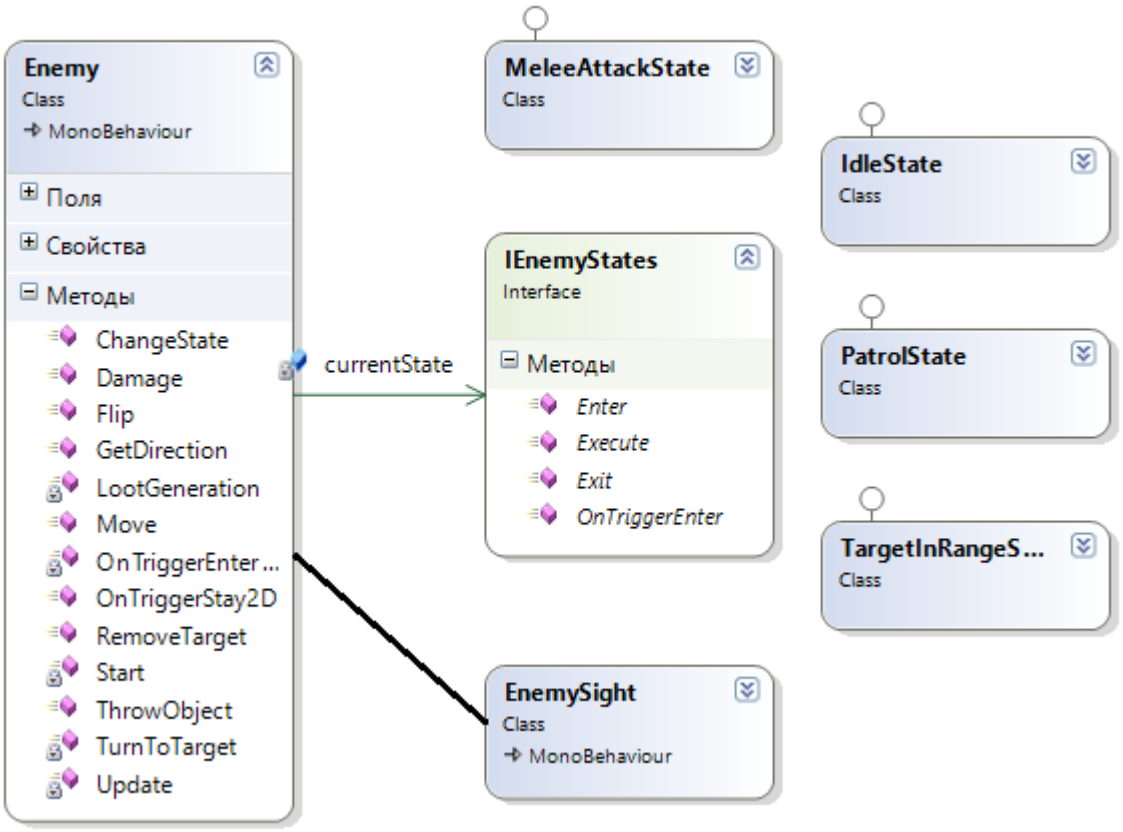


Рисунок 10. Диаграмма классов искусственного интеллекта

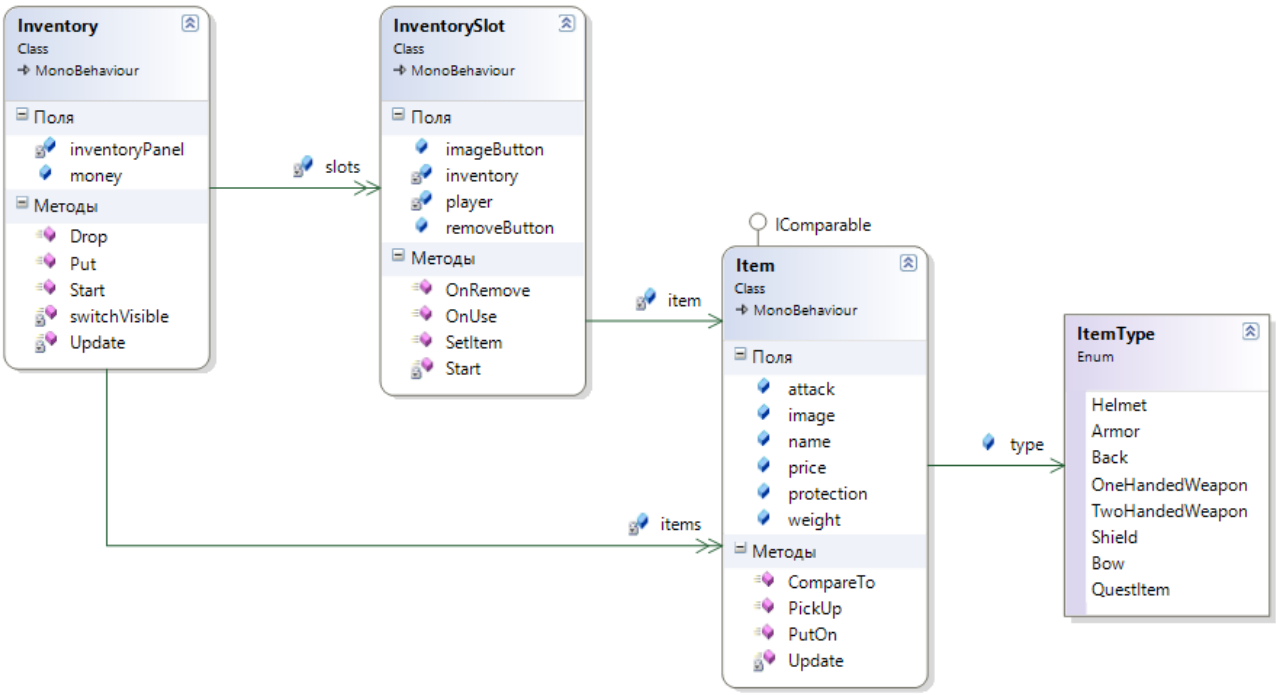


Рисунок 11. Диаграмма классов инвентаря

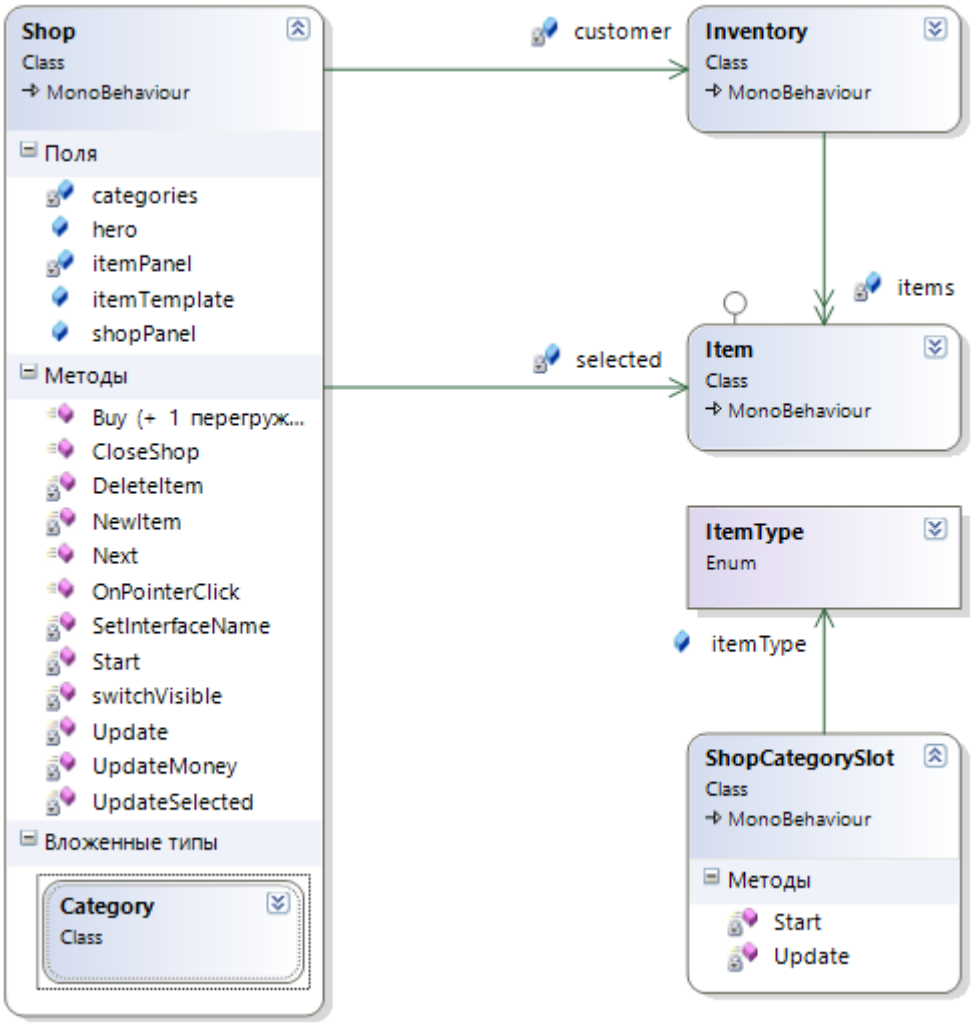


Рисунок 12. Диаграмма классов магазина

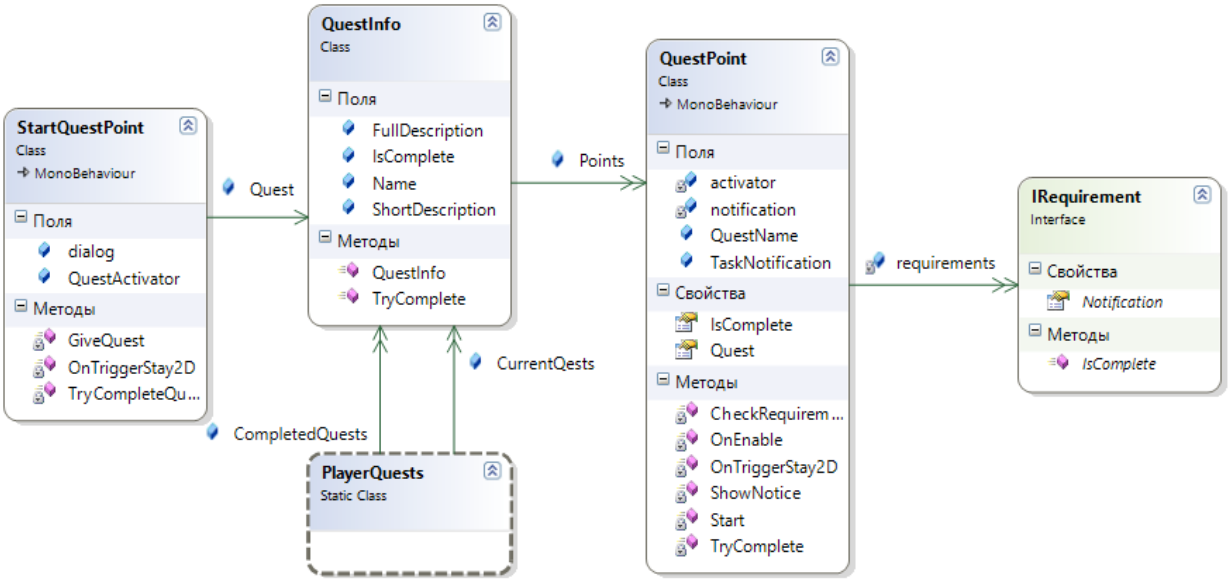


Рисунок 13. Диаграмма классов квестовой системы

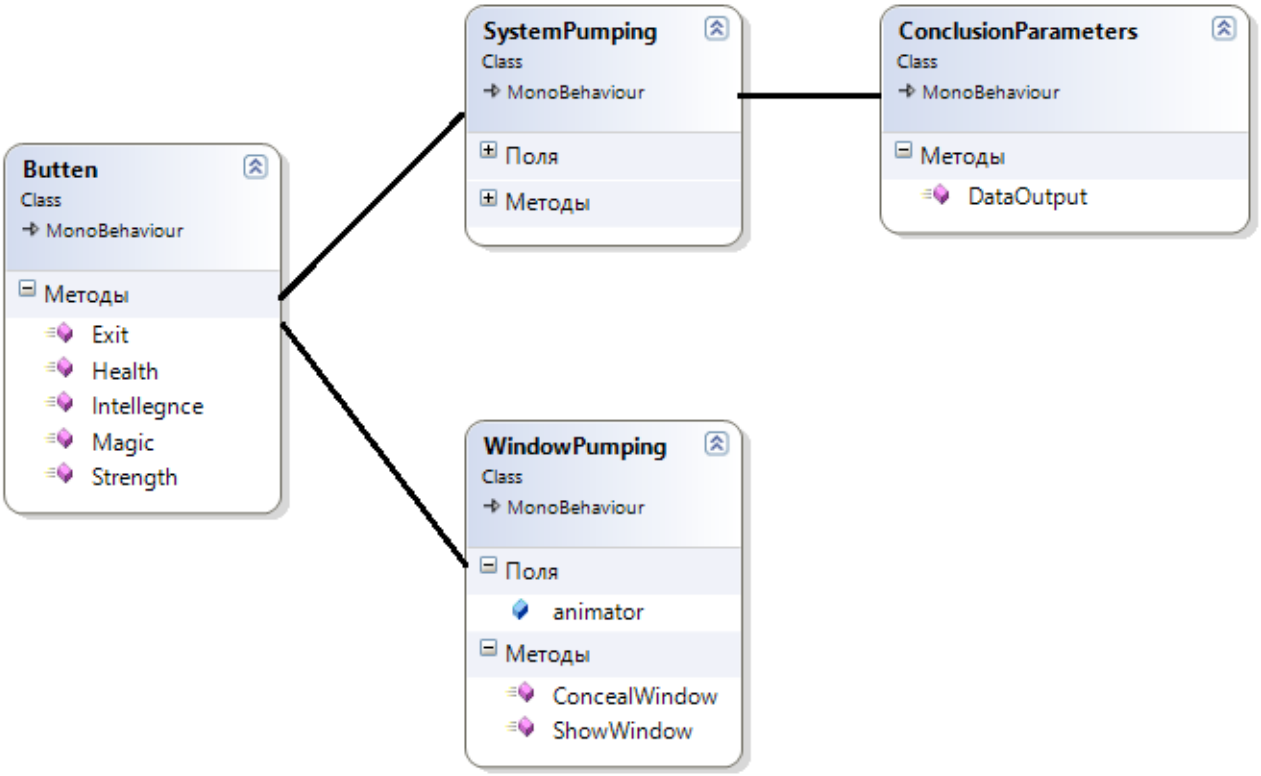


Рисунок 14. Диаграмма классов системы прокачки

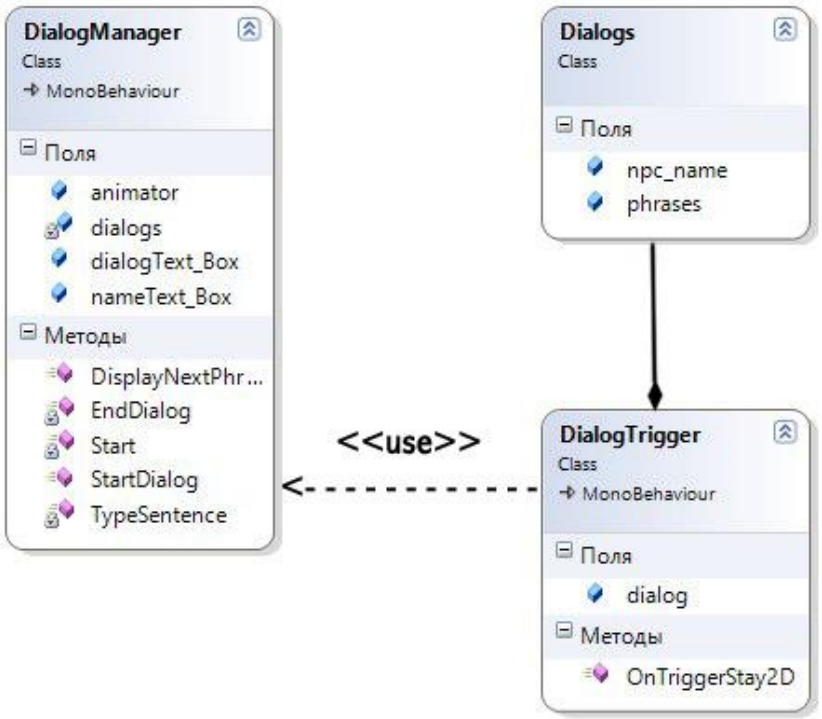


Рисунок 15. Диаграмма классов диалоговой системы