МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Новосибирский национальный исследовательский государственный университет»

(Новосибирский государственный университет, НГУ)

Структурное подразделение Новосибирского государственного университета – Высший колледж информатики Университета (ВКИ НГУ)

КАФЕДРА ИНФОРМАТИКИ

**РАЗРАБОТКА ДЕСКТОПНОГО ПРИЛОЖЕНИЯ ДЛЯ ОС WINDOWS «СОЗДАНИЕ D&D ПЕРСОНАЖА»**

Квалификация программист

|  |  |
| --- | --- |
| Руководитель | Пауль С.А.  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2023 г. |
| Студент 3 курса  гр. 107в2 | Сидоров А.А.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2023 г. |

Новосибирск

2023

**СОДЕРЖАНИЕ**

**Элементы оглавления не найдены.**

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ**

Программное средство (ПС) – это совокупность программных продуктов, которые выполняют определенные функции или решают конкретные задачи.

Операционная система (ОС) – это программное обеспечение, которое управляет аппаратными и программными ресурсами компьютера или другого устройства.

C Sharp (C#) – это объектно-ориентированный язык программирования общего назначения.

Геймер – это термин, который используется для обозначения человека, увлекающегося компьютерными или видеоиграми.

Кастомизация – это процесс настройки и изменения внешнего вида, характеристик или поведения объекта в игре в соответствии с предпочтениями и желаниями игрока.

Игровой персонаж – это виртуальный персонаж, созданный для взаимодействия с игровым миром.

Игровой мир – это виртуальная среда или область, в которой развивается действие видеоигры.

Компьютерные видеоигры – это форма интерактивного развлечения, предназначенная для запуска и воспроизведения на компьютере.

Dungeons and Dragons (D&D) – это настольная ролевая игра, созданная Гари Гайгексом и Дэйвом Арнесоном.

НРИ – Настольная ролевая игра.

Информационный объект – класс, построенный на основе обобщения различных объектов, процессов и моделей, связанных с информационными описаниями, процессами и технологиями, и отношений между ними.

ВВЕДЕНИЕ

Современный мир проживает эпоху стремительного развития цифровых технологий, в которой компьютерные приложения, а особенно видеоигры, играют важную роль в формировании культуры и самовыражении личности.

С ростом популярности компьютерных игр особенно заметен тренд к кастомизации персонажей. Игроки стремятся не только пройти увлекательные сценарии, но и взаимодействовать с игровым миром через уникальные и персонализированные персонажи. Кастомизация открывает дополнительные возможности для самовыражения, позволяя игрокам воплотить свои творческие идеи и фантазии в виртуальной реальности.

Одним из ключевых аспектов актуальности является индивидуальность геймера в виртуальном пространстве видеоигр. В современном обществе игровая культура стала неотъемлемой частью личной идентичности многих людей. Виртуальные миры видеоигр предоставляют возможность создать уникального персонажа, выразить свою индивидуальность и представить себя таким, каким хочется быть в виртуальной реальности.

Еще одной из актуальных сфер в информационной технологии является разработка приложений для управления персонажами в играх, особенно в контексте популярных настольных ролевых игр, таких как Dungeons & Dragons (D&D). Отслеживание инициативы персонажей в таких играх – это важный этап, которым позволяет отслеживать ход боя.

Актуальность данной работы обусловлена не только удовлетворением запросов любителей настольных ролевых игр, но и более широким влиянием цифровых инноваций на формирование современной культуры. Интеграция технологий в развлекательные процессы становится неотъемлемой частью нашей жизни, и разработка программ, улучшающих игровой опыт, имеет существенное значение для современного общества.

В этом контексте создание десктопного приложения для создания персонажей в настольной ролевой игре Dungeons & Dragons (D&D) не только соответствует техническим трендам, но и обладает высокой социокультурной актуальностью. Создание приложения, позволяющего пользователю отслеживать инициативу персонажей в игре D&D. Это не только предоставляет возможность правильно распределить ресурсы пользователя на более важные аспекты игры.

Данный курсовой проект направлен на разработку десктопного приложения для операционной системы Windows, которое позволит упростить и сделать более увлекательным процесс отслеживания боя в игре D&D.

Однако, за пределами узкоспециализированной темы отслеживания инициативы персонажей, стоит заметить, что разработка подобных программ имеет глубокие корни в общем контексте роста цифровых технологий и их влияния на развлекательную и игровую индустрию.

Данная работа также рассматривает вопросы удобства пользовательского интерфейса, что подчеркивает общую тенденцию к созданию приложений, ориентированных на интуитивное взаимодействие с пользователем.

Наконец, анализ предметной области и обоснование необходимости разработки данного приложения требует рассмотрения существующих подходов и решений в данной области. В ходе выполнения курсовой работы будут проанализированы существующие продукты и определены преимущества предлагаемого приложения.

Цель курсовой работы – разработка десктопного приложения для ОС Windows «Система администрирования для НРИ D&D». Следовательно, чтобы добиться достижения поставленной цели нужно выполнить определённый ряд работ, среди которых можно выделить:

1. Изучение основных аспектов предметной области, осуществление исследования для более полного понимания требований и характеристик, связанных с отслеживанием инициативы.
2. Проведение анализа существующих аналогичных приложений с целью выявления их особенностей, преимуществ и недостатков. Подробное изучение функциональности и интерфейса, с целью выявления наилучших практик и уникальных черт для использования в разрабатываемом приложении.
3. Разработка прототипов пользовательского интерфейса с учетом удобства использования. Исследование современных дизайн-тенденций и применение их в разработке интерфейса для повышения привлекательности и интуитивной понятности.
4. Написание кода на языке программирования C# с использованием технологии WPF. Этот код должен включать в себя логику отслеживания состояния персонажей их порядка по инициативе. Также, в коде будет предусмотрено взаимодействие с пользовательским интерфейсом для отображения этих характеристик.
5. Тестирование и отладка. Планирование и проведение тестирования приложения на различных этапах разработки. Выявление и устранение ошибок и недочетов в функциональности и интерфейсе приложения.
6. Написание руководства пользователя. Подготовка подробного руководства, описывающего процесс использования разработанного приложения. Руководство должно включать в себя шаги по управлением интерфейса. Также, следует предоставить информацию об основных возможностях приложения и способах взаимодействия с интерфейсом.
7. Написание пояснительной записки к курсовой работе. Создание подробной пояснительной записки, в которой освещаются все этапы разработки приложения. Это включает в себя описание выбранных технологий, методов и средств разработки, анализ предметной области, обоснование принятых решений по проектированию интерфейса и функционала, а также результаты тестирования и отладки. Также, в пояснительной записке следует выделить основные преимущества разработанного приложения, анализировать полученный опыт и предлагать возможные направления для дальнейшего улучшения.

Таким образом, создание десктопного приложения для отображения и управление инициативой персонажа в игре D&D является не только техническим проектом, но и вкладом в общий контекст развития цифровых технологий, повышения пользовательского опыта и удовлетворения запросов современного поколения, активно взаимодействующего с цифровым миром. Создание приложения для управления персонажами в D&D выходит за рамки простой технической задачи. Это важный шаг в поддержке индивидуальности и самовыражении геймеров в виртуальных мирах, что в современном мире приобретает значимость и влияет на формирование цифровой культуры.

1 ПОСТАНОВКА ЗАДАЧИ ВКР

Целью данного курсового проекта является разработка десктопного приложения для ОС Windows «Система администрирования для НРИ D&D». Целью разработки является предоставление удобного и быстрого инструмента, который позволит сократить время, затрачиваемое на управление игрой в реальной мире, и снизить барьеры вхождения для новых игроков. Для выполнения поставленной цели требуется выполнить определённый ряд работ.

Опишем следующие задачи, необходимые для достижения цели:

1. Отображение показателей персонажа:
   1. Реализовать логику отображения информации о персонаже при переходе на вкладку «Character».
   2. Реализовать логику отображения значений для характеристик при переходе на вкладку «Stat».
2. Инициализация броска инициативы для вхождения в бой:
   1. При нажатии кнопки «бросок инициативы» реализовать логику, которая произведет бросок инициативы на основе характеристик персонажа.
3. Управление ходом:
   1. При нажатии кнопки «Закончить ход» реализовать логику, которая произведет конец хода персонажа.
4. Обработка ошибок:
   1. Добавить обработку возможных ошибок, таких как некорректные значения характеристик или отсутствие выбора расы/класса.
   2. Вывести сообщения об ошибках или подсказки для пользователя для улучшения опыта использования приложения.
5. Оформление и стиль:
   1. Работать над стилем и оформлением интерфейса для повышения привлекательности и интуитивной понятности.
   2. Разработать логику обновления и отображения элементов интерфейса, чтобы они были интуитивно понятны и легко воспринимаемы.
6. Тестирование и отладка:
   1. Провести тестирование приложения на различных этапах разработки для выявления ошибок и недочетов в функциональности и интерфейсе.
   2. Устранить выявленные ошибки и обеспечить стабильную работу приложения.
7. Разработка руководства пользователя:
   1. Создать руководство пользователя, содержащее подробные инструкции по использованию приложения.

Вышеописанные задачи направлены на успешное выполнение поставленной цели и создание полноценного приложения для создания D&D персонажей на платформе Windows. Разработка приложения нацелена на достижение практических целей, удовлетворяя потребности широкого круга пользователей и содействуя улучшению опыта в мире Dungeons & Dragons.

**1.1 Пользовательские требования**

Пользовательские требования определяют функциональности, которые пользователи ожидают от приложения «Система администрирования для НРИ D&D» с целью достижения своих целей в контексте управления персонажами в игре Dungeons & Dragons.

В эту область также включаются характеристики приложения, существенные для обеспечения удовлетворения потребностей пользователей и обеспечения им приятного и продуктивного взаимодействия с приложением.

1. Возможность выбора расы и класса персонажа:
   1. Пользователь должен иметь возможность выбирать из предоставленного списка различные расы и классы для своего персонажа в соответствии с правилами игры Dungeons & Dragons.
2. Генерация характеристик персонажа:
   1. Приложение должно предоставлять функцию для автоматической генерации характеристик персонажа, включая силу, ловкость, выносливость, интеллект, мудрость и харизму, согласно правилам D&D.
3. Просмотр и редактирование характеристик:
   1. Пользователь должен иметь возможность просматривать и редактировать характеристики своего персонажа, включая базовые значения и модификаторы, для дальнейшего улучшения персонажа.
4. Отображение информации о расе и классе:
   1. Приложение должно показывать подробную информацию о выбранной расе и классе, включая бонусы к характеристикам и другие особенности, чтобы пользователь мог принимать более осознанные решения.
5. Создание персонажа:
   1. Пользователь должен иметь возможность создавать и сохранять своих уникальных персонажей, интегрируя выбор расы, класса и характеристик.
6. Интуитивный пользовательский интерфейс:
   1. Интерфейс приложения должен быть интуитивно понятным как для опытных игроков D&D, так и для новичков, обеспечивая удобство использования и навигации.

Разработка десктопного приложения «Создание D&D персонажа» представляет собой значимый шаг в направлении улучшения пользовательского опыта и облегчения процесса создания персонажей для игроков настольной ролевой игры Dungeons & Dragons (D&D).

**1.2 Системные требования**

В контексте приложения «Создание D&D персонажа», системные требования определяют технические характеристики и условия, которые необходимы для корректной установки, функционирования и использования вашего программного продукта.

Эти требования включают в себя:

1. Платформа. Приложение разработано для операционной системы Windows 8 и больше.
2. Память. Рекомендуемый объем оперативной памяти для стабильной работы приложения – не менее 4 ГБ.
3. Процессор. Процессор с тактовой частотой не менее 2.0 ГГц.
4. Дисковое пространство. Свободное дисковое пространство для установки приложения и сохранения данных – не менее 200 МБ.
5. Разрешение экрана. Рекомендуемое разрешение экрана – не менее 1280x720 пикселей.
6. Надежность. Приложение должно обеспечивать стабильную работу без существенных сбоев и зависаний.
7. Поддержка разрядности. Приложение поддерживает 64-разрядные версии операционной системы Windows.

**1.3 Требования к графическому пользовательскому интерфейсу**

В контексте приложения «Создание D&D персонажа», требования к графическому пользовательскому интерфейсу определяют, каким образом пользователи будут взаимодействовать с приложением. Эти требования включают в себя:

1. Внешний вид интерфейса:
   1. Описание дизайна и стиля интерфейса, который должен быть интуитивно понятным для пользователей D&D, включая опытных игроков и новичков.
   2. Графические элементы (цвета, шрифты, изображения), соответствующие тематике D&D.
2. Навигация:
   1. Обеспечение удобной навигации по приложению, чтобы пользователи могли легко находить нужные функции.
   2. Размещение элементов управления (кнопки, поля ввода) в логичных и интуитивно понятных местах.
3. Взаимодействие:
   1. Определение способов взаимодействия пользователя с приложением (например, клики).
   2. Предоставление обратной связи пользователю при выполнении определенных действий.
4. Доступность функциональности:
   1. Гарантирование, что все функциональные возможности приложения доступны через пользовательский интерфейс.
   2. Понятные и информативные метки, подсказки и инструкции для облегчения понимания возможностей приложения.

Таким образом, пользовательский интерфейс приложения разрабатывается с учетом легкости освоения, что делает его доступным как для опытных игроков, так и для новичков.

**1.4 План-график выполнения ВКР**

Ниже продемонстрирован план-график выполнения курсовой работы (табл.1).

Таблица 1 – План-график выполнения курсовой работы

| Дата | Задача |
| --- | --- |
| 20.11.2023 | Проведение анализа предметной области и изучение требований. |
| 26.11.2023 | Исследование аналогичных приложений и выявление их особенностей. |
| 01.12.2023 | Разработка прототипов пользовательского интерфейса. |
| 07.12.2023 | Начало написания кода на C# с использованием WPF. |
| 10.12.2023 | Внедрение привязок данных между объектами MainVM2 и элементами управления. |
| 20.12.2023 | Тестирование и отладка приложения. |
| 21.12.2023 | Подготовка пояснительной записки и руководства пользователя. |

Таким образом, план-график предполагает последовательное выполнение основных этапов работы над проектом в течение декабря 2023 года. Задачи включают в себя анализ, исследование, разработку, программирование, тестирование и завершение проекта с подготовкой документации.

2 АНАЛИЗ ТРЕБОВАНИЙ И ОПРЕДЕЛЕНИЕ СПЕЦИФИКАЦИЙ

2.1 Описание предметной области задачи ВКР

Современные исследования подтверждают, что компьютерные игры в наше время играют важную роль в жизни многих людей. Игры не только предоставляют развлечение, но и становятся средством самовыражения и даже социализации. В этом контексте, настольные ролевые игры, такие как Dungeons & Dragons (D&D), занимают особенное место, предоставляя игрокам уникальный опыт создания и воплощения собственных персонажей в вымышленном мире.

При создании персонажа в D&D игрок сталкивается с множеством выборов, определяющих уникальные черты его героя. Однако, это может быть сложным и иногда утомительным процессом, особенно для новичков. Исследования показывают, что уровень удовлетворенности игрока от игрового процесса напрямую связан с лёгкостью восприятия и управления характеристиками персонажа.

В предметной области проектируемого приложения важными аспектами являются не только выбор характеристик, расы и класса персонажа, но и создание удобного и интуитивно понятного интерфейса для визуализации и редактирования этих характеристик. Это позволит игрокам наслаждаться процессом создания персонажа, подчеркивая индивидуальность каждого геймера.

Проект ориентирован на решение проблем, с которыми сталкиваются игроки D&D при создании персонажей, и на предоставление им инструмента, который сделает этот процесс более простым и доступным для широкого круга пользователей. Предметная область также включает в себя анализ тенденций в развитии игровой индустрии и растущего интереса к настольным ролевым играм, что подчеркивает актуальность данного проекта.

В рамках данного проекта будут рассмотрены и решены проблемы, связанные с созданием и управлением персонажами в D&D, а также будет предложено удобное приложение, позволяющее игрокам интуитивно взаимодействовать с характеристиками своих персонажей.

***2.1.1 Информационные объекты предметной области и взаимосвязи между ними***

В предметной области создания программного средства, направленного на кастомизацию и индивидуализацию игроков в настольных играх, информационные объекты представляют собой разнообразные элементы, процессы и модели. Эти элементы тесно связаны с описаниями, процессами и технологиями, активно применяемыми в сфере визуальных решений и развития креативного мышления. В рамках предметной области «Создание D&D персонажа» выделяются разнообразные информационные объекты, взаимосвязанные между собой.

Выделим следующие информационные объекты:

1. Графические компоненты:
   1. Визуальные элементы, предоставляющие пользователю возможность кастомизации внешнего вида персонажей.
   2. Примеры: Редакторы внешности, выбор стилей, визуальные ассеты.
2. Процессы кастомизации:
   1. Шаги и этапы, которые пользователь проходит для создания уникального персонажа.
   2. Примеры: Выбор характеристик, определение внешности, настройка особенностей.
3. Технологии визуализации:
   1. Специфические методы и средства для визуализации персонажей и их характеристик.
   2. Примеры: 2D и 3D графика, анимации, техники отображения.
4. Креативные модели:
   1. Модели и инструменты, способствующие развитию креативности у игроков.
   2. Примеры: Генераторы идей, поддержка творческого мышления, кастомизация сюжетных элементов.
5. Интерфейс взаимодействия:
   1. Элементы, обеспечивающие взаимодействие пользователя с программным средством.
   2. Примеры: Удобные панели управления, интуитивные элементы интерфейса.

Эти информационные объекты формируют сложную систему, обеспечивая качественную кастомизацию и индивидуализацию игрового опыта в настольных играх.

***2.1.2 Информационные и функциональные потребности пользователей разрабатываемой ПС (ПМ)***

Исходя из предметной области, пользователи разрабатываемой программной системы (ПС) имеют следующие информационные и функциональные потребности:

1. Информационные потребности:
   1. Получение информации о доступных характеристиках, расах и классах.
   2. Ознакомление с описаниями и уникальными чертами персонажей.
   3. Возможность просмотра изменений в характеристиках при выборе разных рас и классов.
2. Функциональные потребности:
   1. Создание персонажа с выбранными характеристиками, расой и классом.
   2. Визуализация созданного персонажа.
   3. Возможность редактирования характеристик персонажа.

Эти потребности обуславливают функциональность приложения, которая должна быть интуитивной и удовлетворять требованиям разнообразных пользователей.

***2.1.4 Обзор существующих программных реализаций решения задачи***

В настоящее время существует несколько программных решений, направленных на кастомизацию и индивидуализацию игрового опыта в настольных играх.

Ниже представлен обзор двух таких аналогов с указанием их достоинств и недостатков.

Tabletopia – это абсолютно новая виртуальная платформа для настольных игр в электронном формате. Пользователи могут создавать, играть и делиться настольными играми в виртуальной среде.

Платформа обеспечивает возможность играть в разнообразные настольные игры, включая классические настольные ролевые игры, настольные стратегии, настольные карточные игры и многие другие.

Игроки могут взаимодействовать с игрой и другими игроками в реальном времени, создавать собственные правила и настраивать игровой процесс.

Платформа Tabletopia стремится предоставить виртуальное пространство для глобального настольного игрового опыта.

Ниже продемонстрирована виртуальная платформа Tabletopia (рис.1).



Рисунок 1 – Виртуальная платформа Tabletopia

Достоинства:

1. Онлайн-Платформа для настольных игр. Tabletopia предоставляет виртуальную платформу, на которой пользователи могут играть в настольные игры онлайн.
2. Широкий выбор игр. Платформа включает в себя множество настольных игр различных жанров и стилей.
3. Возможность создания собственных игр. Пользователи могут создавать и публиковать свои собственные настольные игры.

Недостатки:

1. Ограничения в кастомизации персонажей. В некоторых играх на Tabletopia может быть ограниченный уровень кастомизации персонажей.
2. Зависимость от интернета. Для использования Tabletopia требуется стабильное интернет-соединение.

Hero Forge – это онлайн-сервис, предоставляющий возможность создания и заказа персонализированных настольных миниатюр для настольных ролевых игр (НРИ) и других игровых систем. Сервис позволяет пользователям индивидуализировать внешний вид своих персонажей, выбирая различные параметры, такие как раса, класс, поза, экипировка и даже фон.

Ниже продемонстрирован онлайн-сервис Hero Forge (рис.2).



Рисунок 2 – Онлайн-сервис Hero Forge

Достоинства:

1. Полная кастомизация внешности. Hero Forge предлагает полную кастомизацию внешности персонажей с возможностью выбора позы, экипировки и деталей внешности.
2. Возможность заказа реальной миниатюры. Пользователи могут заказать физическую миниатюру своего персонажа, созданного в приложении.
3. Широкие варианты рас и классов. Множество опций для выбора расы и класса персонажа.

Недостатки:

1. Отсутствие игровых характеристик. Hero Forge фокусируется в основном на внешности, и ей не присущи функции отслеживания игровых характеристик.
2. Высокая цена физических миниатюр. Заказ физической миниатюры может быть дорогим удовольствием.

D&D Beyond (Dungeons & Dragons Beyond) – это цифровая платформа, предназначенная для поддержки игры в настольную ролевую игру Dungeons & Dragons (D&D). Она предлагает игрокам и ведущим игр современные инструменты для управления персонажами, кампаниями и вселенной D&D. В D&D Beyond пользователи могут создавать, редактировать и хранить персонажей, использовать онлайн-ресурсы правил, а также обмениваться информацией с другими участниками игры. Платформа также предоставляет доступ к цифровым версиям основных книг D&D, что облегчает доступ к правилам и ресурсам для игроков.

Ниже продемонстрирована цифровая платформа D&D Beyond (рис.3).

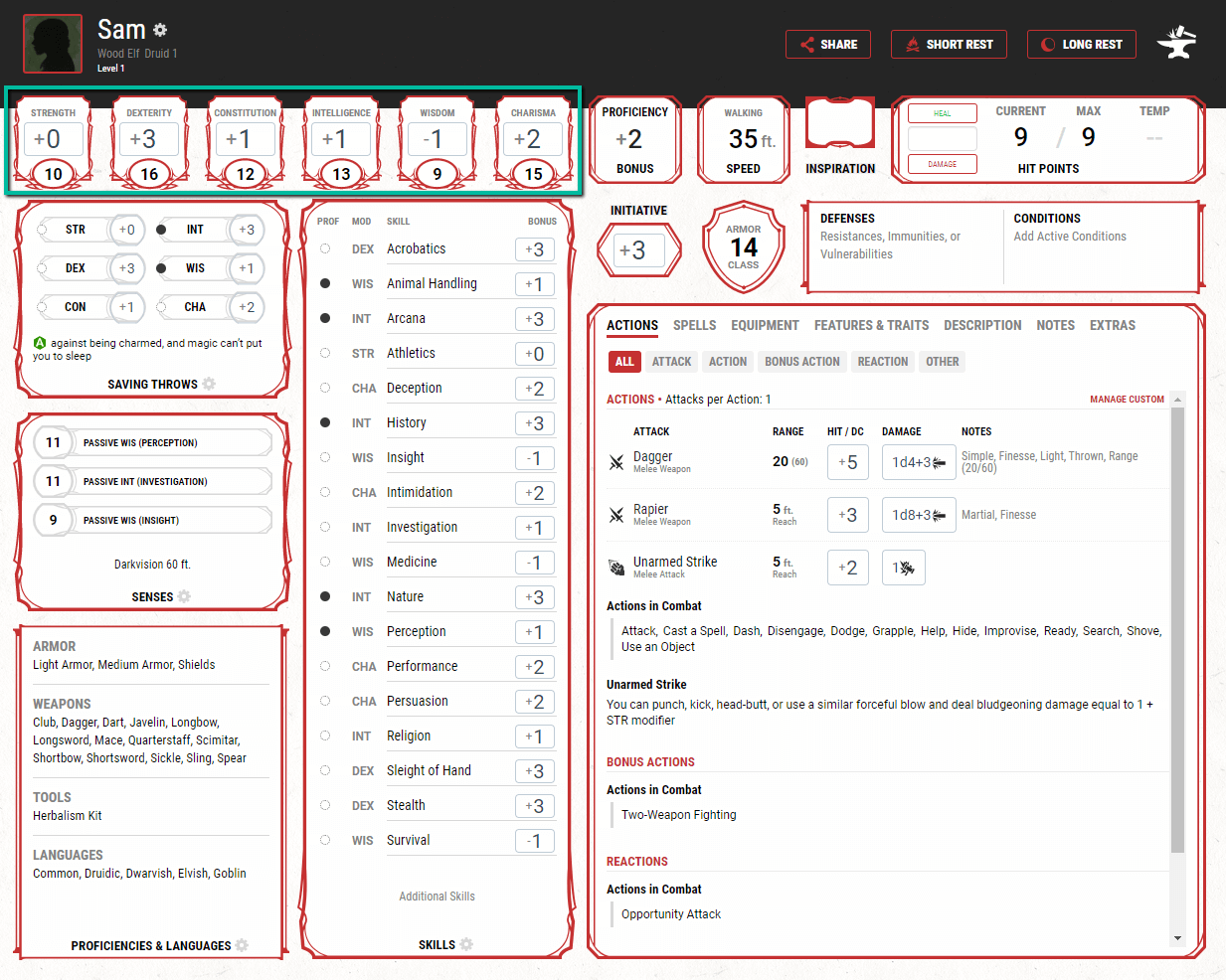


Рисунок 3 – Цифровая платформа D&D Beyond

Достоинства:

1. Обширная база знаний. D&D Beyond предоставляет обширную базу данных по вселенной Dungeons & Dragons, включая расы, классы и предметы.
2. Онлайн-редактор персонажей. Имеется удобный онлайн-редактор персонажей с возможностью кастомизации внешности и выбора характеристик.
3. Удобство взаимодействия. Интеграция с онлайн-играми, обмен персонажами между игроками.

Недостатки:

1. Платные опции. Некоторые функции доступны только по подписке, что может быть неприемлемо для части пользователей.
2. Ограниченные возможности бесплатной версии. Некоторые функциональности ограничены в бесплатной версии приложения.

Существующие программные реализации предлагают различные подходы к кастомизации персонажей в настольных играх. Новое программное средство, разрабатываемое в рамках ВКР, будет уникальным за счет комбинации удобства взаимодействия, обширных функциональных возможностей и отсутствия ограничений в использовании основных функций.

***2.1.5 Концептуальное обоснование разработки***

Концептуальное обоснование разработки программного средства для кастомизации и индивидуализации игрового опыта в настольных играх базируется на тщательном анализе проблем в предметной области. После изучения вышеописанных решений, выявляются определенные ограничения и недостатки, которые могут быть решены через новый подход.

Программное средство, разрабатываемое в данной работе, стремится объединить преимущества кастомизации внешности персонажей. Ключевой идеей является расширение функциональности.

Разработка нового программного средства обоснована потребностью в более гибком, разнообразном и индивидуализированном подходе к созданию и взаимодействию с персонажами.

**2.2 Классы и характеристики пользователей**

В рамках разработки программного средства, направленного на кастомизацию и индивидуализацию игрового опыта в настольных играх, выделяются следующие пользовательские роли:

1. Пользователи:
   1. Предпочтения в стилях кастомизации.
   2. Творческий подход к созданию визуальных элементов.
   3. Опыт в кастомизации и индивидуализации игровых персонажей (новичок, опытный игрок, мастера игры).
2. Задачи:
   1. Создание персонализированных персонажей.
   2. Создание уникальных внешних обликов персонажей.
3. Разработчики:
   1. Навыки программирования и разработки ПО.
   2. Понимание технических аспектов создания приложений.
4. Задачи:
   1. Реализация функциональности программного средства.
   2. Обеспечение интеграции различных компонентов.

Каждая из этих пользовательских ролей обладает своими уникальными характеристиками и выполняет определенные задачи в контексте использования разрабатываемого программного средства. Это разнообразие ролей позволит удовлетворить потребности широкого круга пользователей и сделать программу более привлекательной для различных категорий игроков и создателей игр.

**2.3 Функциональные требования**

Функциональные требования определяют поведение программной системы и набор задач, которые она должна выполнять. В контексте разрабатываемого программного средства, направленного на кастомизацию и индивидуализацию игрового опыта в настольных играх, определяются следующие функциональные требования.

Ниже продемонстрирована Диаграмма вариантов использования (рис.4).

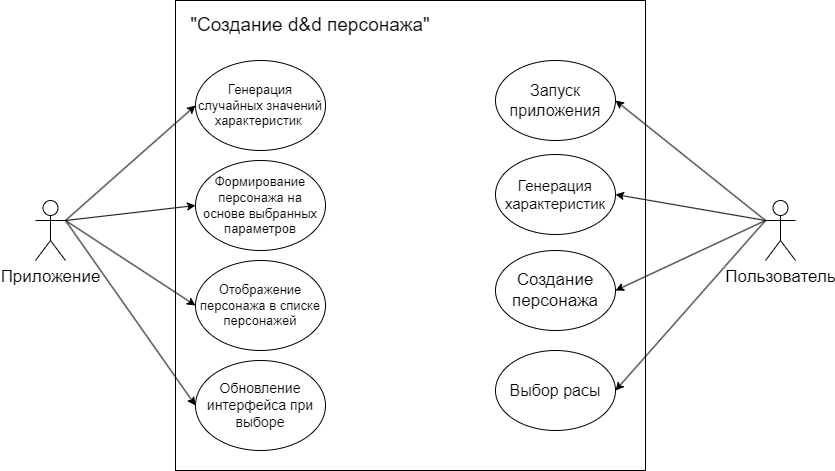


Рисунок 4 – Диаграмма вариантов использования

Актеры:

Пользователь: Лицо, использующее приложение для создания персонажей.

Система: Приложение для создания персонажей.

Варианты использования:

Генерация характеристик:

Описание: Пользователь нажимает кнопку «Сгенерировать характеристики».

Действие системы: Генерация случайных значений характеристик и их обновление в интерфейсе.

Создание персонажа:

Описание: Пользователь нажимает кнопку «Создать персонажа» после выбора расы и класса.

Действие системы: Формирование персонажа на основе выбранных параметров и отображение его в списке персонажей.

Выбор расы:

Описание: Пользователь выбирает расу из списка доступных в MainWindow.

Действие системы: Обновление интерфейса, предоставление дополнительных опций для выбора расы.

Выбор класса:

Описание: Пользователь выбирает класс из списка доступных в MainWindow.

Действие системы: Обновление интерфейса, предоставление дополнительных опций для выбора класса.

Обновление интерфейса при выборе:

Описание: Изменение параметров (расы, класса, характеристик) пользователем.

Действие системы: Автоматическое обновление интерфейса с учетом выбранных параметров.

Выполнение вышеперечисленных функциональных требований играют ключевую роль в обеспечении функциональности и удовлетворении потребностей пользователей в области кастомизации настольных игр. Создание системы поддержки и обеспечения взаимодействия между этими пользователями позволит создать инновационное и удобное в использовании приложение для кастомизации игрового опыта.

***2.3.1 Описание прецедентов***

1. Генерация характеристик:

Пользователь запускает процесс генерации случайных значений характеристик, необходимых для создания персонажа.

Основной сценарий:

* 1. Пользователь открывает приложение и переходит в раздел с характеристиками игрового персонажа.
  2. Пользователь нажимает кнопку «Сгенерировать характеристики».
  3. Система генерирует случайные значения для каждой характеристики и выводит в текстовые поля возможные значения.

1. Создание персонажа:

Пользователь формирует персонажа, выбирая расу, класс и используя сгенерированные характеристики.

Основной сценарий:

* 1. Пользователь выбирает расу из списка.
  2. Пользователь выбирает класс из списка.
  3. Пользователь вводит характеристики из списка.
  4. Пользователь нажимает кнопку «Создать персонажа».
  5. Система создает персонажа, используя выбранные параметры, и отображает его в списке.

1. Выбор расы:

Пользователь выбирает расу для своего персонажа из списка доступных опций.

Основной сценарий:

* 1. Пользователь просматривает доступные расы в списке.
  2. Пользователь выбирает желаемую расу.

1. Выбор класса

Пользователь выбирает класс для своего персонажа из списка доступных опций.

Основной сценарий:

* 1. Пользователь просматривает доступные классы в списке.
  2. Пользователь выбирает желаемый класс.

1. Обновление интерфейса при выборе:

Пользователь видит актуализированный интерфейс при изменении параметров (расы, класса, характеристик).

Основной сценарий:

* 1. Пользователь выбирает расу, класс или генерирует характеристики.
  2. Система автоматически обновляет интерфейс, предоставляя пользователю актуальные опции и информацию.

**2.4** Нефункциональные требования

Нефункциональные требования определяют качественные характеристики системы, не связанные прямо с её функциональностью. Они играют ключевую роль в обеспечении стабильности, безопасности и удовлетворения пользовательских потребностей.

Опишем подробнее нефункциональные требования для разработки десктопного приложения «Создание D&D персонажа»:

1. Производительность: Система должна предоставлять отклик на запросы пользователя в течение не более чем 2 секунд.
2. Переносимость: Приложение должно корректно функционировать на операционных системах Windows.
3. Надежность: Вероятность критических сбоев в системе должна быть минимальна.
4. Интуитивный интерфейс: Пользовательский интерфейс должен быть интуитивно понятным, даже для новых пользователей.
5. Доступность: Система должна соответствовать стандартам доступности, обеспечивая удобство использования для людей с ограниченными возможностями.

Следовательно, нефункциональные требования направлены на создание стабильного, эффективного и удобного в использовании приложения. Они обеспечивают не только работоспособность, но и общепризнанное качество взаимодействия пользователя с приложением, что способствует позитивному опыту и удовлетворенности пользователя.

3 ВЫБОР ПРОГРАММНЫХ СРЕД И СРЕДСТВ РАЗРАБОТКИ

Для разработки десктопного приложения «Создание D&D персонажа» были выбраны следующие программные средства и фреймворки: Visual Studio, C#, WPF, а также XAML. Выбор программных сред разработки оправдан использованием Visual Studio. Это обширное и мощное инструментальное средство, которое обеспечивает высокую производительность разработки, интеграцию с WPF и XAML, а также широкие возможности для проектирования и отладки приложений. Важными факторами при выборе были также богатая документация и поддержка со стороны сообщества разработчиков.

**3.1 Сравнительный анализ имеющихся возможностей по выбору средств разработки**

Visual Studio. Среда разработки, сделанная компанией Microsoft, Visual Studio предоставляет разработчикам широкий набор инструментов и возможностей для написания, отладки, тестирования и развертывания программ.

Достоинства:

1. Обширные возможности разработки.
2. Поддержка C# и XAML с высоким уровнем интеграции.
3. Большое сообщество и обширная документация.

Недостатки:

1. Занимает много места на диске.
2. Может потребовать времени для освоения всех функций.

C# (C Sharp). Язык программирования, разработанный компанией Microsoft.

Достоинства:

1. Объектно-ориентированный подход: Удобство организации кода и взаимодействия с объектами.
2. Интеграция с платформой .NET: Возможность использования библиотек и фреймворков, упрощающих разработку.

Недостатки:

1. Ограничение на платформу Microsoft: Язык ориентирован на платформу Microsoft, что может создать зависимость от её технологий.

XAML (Extensible Application Markup Language). Предоставляет декларативный способ создания графических пользовательских интерфейсов, позволяя разработчикам описывать структуру и внешний вид UI, а также взаимодействие между элементами.

Достоинства:

1. Декларативный подход: Удобство создания и изменения интерфейса без необходимости программирования.
2. Интеграция с WPF: Возможность создания богатого и гибкого пользовательского интерфейса.

Недостатки:

1. Не предоставляет функциональности программирования.
2. XAML не является языком программирования, и для логики приложения все равно требуется использование C# или другого языка.

WPF (Windows Presentation Foundation). WPF является частью платформы .NET и предоставляет разработчикам мощные инструменты для создания интерактивных и стильных приложений.

Достоинства:

1. Гибкий пользовательский интерфейс:
2. Возможность создания сложных и красочных интерфейсов.
3. Интеграция с XAML и C#:
4. Способствует разделению дизайна и логики приложения.

Недостатки:

1. Требует изучения.
2. WPF имеет свои особенности, и разработчику может потребоваться время для их освоения.

**3.2 Характеристика выбранных программных сред и средств**

Visual Studio – это мощная интегрированная среда разработки, предоставляющая все необходимые инструменты для создания WPF-приложений на языке C# с использованием XAML.

C# (C Sharp) – это объектно-ориентированный язык программирования, разработанный для создания приложений на платформе Microsoft. В контексте приложения "Создание D&D персонажа", C# используется для программирования логики приложения, взаимодействия с интерфейсом и обработки данных.

Windows Presentation Foundation (WPF) – это фреймворк для построения приложений с богатым графическим интерфейсом пользователя. Он предоставляет средства для разработки инновационных и стильных приложений под Windows, а также интегрируется с языком C# и языком разметки XAML. Это технология, предоставляющая средства для создания десктопных приложений с богатым и гибким пользовательским интерфейсом. В контексте приложения "Создание D&D персонажа", WPF используется для реализации визуальной части приложения, взаимодействия с XAML и обеспечения пользовательского опыта.

XAML (Extensible Application Markup Language) – это декларативный язык разметки, используемый для создания пользовательского интерфейса в приложениях, разработанных под управлением технологии WPF. В контексте приложения «Создание D&D персонажа», XAML используется для описания структуры и внешнего вида пользовательского интерфейса.

Таким образом, выбор программных сред и средств (Visual Studio, C#, XAML, WPF) обоснован стремлением к созданию десктопного приложения с современным и привлекательным интерфейсом для удобства пользователя. Visual Studio предоставляет всесторонние возможности разработки и интеграции с технологиями, а C#, XAML и WPF обеспечивают эффективные средства для создания функционального и графически привлекательного приложения. Сочетание этих инструментов позволяет достичь баланса между производительностью разработки, мощью функционала и качеством пользовательского интерфейса.

4 **АЛГОРИТМ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ**

**4.1 Этапы реализации ПС (ПМ)**

Для реализации приложения «Создание D&D персонажа» можно выделить следующие этапы:

1. Анализ требований и проектирование:
   1. Изучение бизнес-требований и пользовательских запросов.
   2. Проектирование базы данных для хранения информации о персонажах.
   3. Разработка структуры пользовательского интерфейса в XAML.
   4. Проектирование архитектуры приложения.
2. Настройка окружения разработки:
   1. Установка Visual Studio и необходимых компонентов.
   2. Создание проекта WPF.
3. Разработка интерфейса:
   1. Создание главного окна приложения в XAML.
   2. Добавление элементов управления для выбора расы, класса и генерации характеристик.
   3. Визуализация списка доступных рас и классов.
4. Логика генерации характеристик:
   1. Реализация алгоритма генерации случайных значений для характеристик.
   2. Обновление соответствующих текстовых полей в интерфейсе с новыми значениями.
5. Логика создания персонажа:
   1. Реализация функционала создания персонажа с учетом выбранных параметров.
   2. Отображение созданного персонажа в списке в пользовательском интерфейсе.
6. Выбор расы и класса:
   1. Расширение возможностей выбора расы и класса, обеспечение подробного описания каждой расы и класса.
   2. Улучшение визуального представления рас и классов.
7. Тестирование:
   1. Проведение модульных тестов для проверки корректности работы каждого функционала.
   2. Проведение интеграционных тестов для оценки взаимодействия компонентов.
8. Оптимизация и доработка:
   1. Выявление узких мест и оптимизация производительности.
   2. Доработка пользовательского интерфейса и функционала на основе обратной связи.
9. Документация:
   1. Подготовка технической документации.
   2. Создание руководства пользователя.

Каждый этап включает в себя ряд конкретных задач и шагов, необходимых для успешной реализации приложения.

**4.2 Пользовательский интерфейс ПС (ПМ)**

***4.2.1 Взаимодействие пользователей с ПС (ПМ)***

Алгоритмы взаимодействия:

1. Вход в систему:
   1. Пользователь запускает приложение.
2. Создание персонажа:
   1. Пользователь запускает процесс создания нового персонажа.
   2. Система предоставляет форму с необходимыми полями (характеристики, класс, раса и т.д.).
   3. После ввода данных, система создаёт лист персонажа в отдельном окне.

***4.2.2 Проектирование пользовательских сценариев***

При проектировании пользовательского интерфейса важным этапом является разработка логики диалога с пользователем и определение взаимосвязей форм приложения.

Ниже представлены текстовые описания некоторых ключевых пользовательских сценариев:

1. Создание нового персонажа:
   1. Описание: Пользователь запускает приложение и выбирает опцию «Создать нового персонажа». Затем система предоставляет форму для ввода основных данных, таких как раса, класс и характеристики.
   2. Взаимодействие: Пользователь вводит необходимую информацию и переходит к следующему шагу, выбору класса.
2. Выбор класса персонажа:
   1. Описание: После ввода основных данных, пользователь переходит к выбору класса персонажа из представленного списка. Каждый класс имеет свои уникальные характеристики и способности.
   2. Взаимодействие: Пользователь просматривает доступные классы, выбирает подходящий, и переходит к следующему этапу или редактирует дополнительные параметры.
3. Предпросмотр и завершение:
   1. Описание: После внесения всех необходимых данных и выбора класса, система предоставляет пользователю возможность предпросмотра созданного персонажа. Пользователь может проверить все введенные данные и внести коррективы при необходимости.
   2. Взаимодействие: Пользователь просматривает предпросмотр, уточняет детали, и при удовлетворении завершает создание персонажа.

Проектирование пользовательских сценариев и диалогов важно для обеспечения удобства использования приложения. В результате этих этапов разработки достигается понятность взаимодействия пользователя с системой, а также оптимальный пользовательский опыт при выполнении различных задач. Такие сценарии позволяют пользователям легко и интуитивно взаимодействовать с приложением, создавая и редактируя персонажей в соответствии с правилами D&D.

***4.2.3 Определение операций пользователей***

При проектировании пользовательского интерфейса приложения «Создание D&D персонажа» определены следующие операции, которые пользователи могут выполнять в рамках прецедентов:

1. Создание нового персонажа:

Операции:

* 1. Ввод основных характеристик (раса, класс).
  2. Выбор дополнительных параметров (навыки, способности).
  3. Изменение характеристик, класса или расы.
  4. Подтверждение создания персонажа.

1. Предпросмотр персонажа:

Операции:

* 1. Выбор персонажа для предпросмотра.
  2. Просмотр подробной информации о персонаже.

***4.2.4 Проектирование структуры экранов и схема функционирования экранов***

На основе определенных операций пользователей была разработана структура экрана. Вся информация о персонаже, включая основные и дополнительные характеристики, отображается на единственном экране. Пользователь имеет возможность в интерактивном режиме вводить и изменять данные, а также выполнять различные действия.

На основе определенных операций пользователей и сценариев взаимодействия, была разработана структура экранов системы:

Ввод и редактирование данных:

Пользователь взаимодействует с текстовыми полями и элементами выбора для ввода основных и дополнительных характеристик.

Интерактивные элементы обеспечивают удобство ввода и выбора параметров.

На рисунке 5 схема функционирования экранов приложения «Создание d&d персонажа»

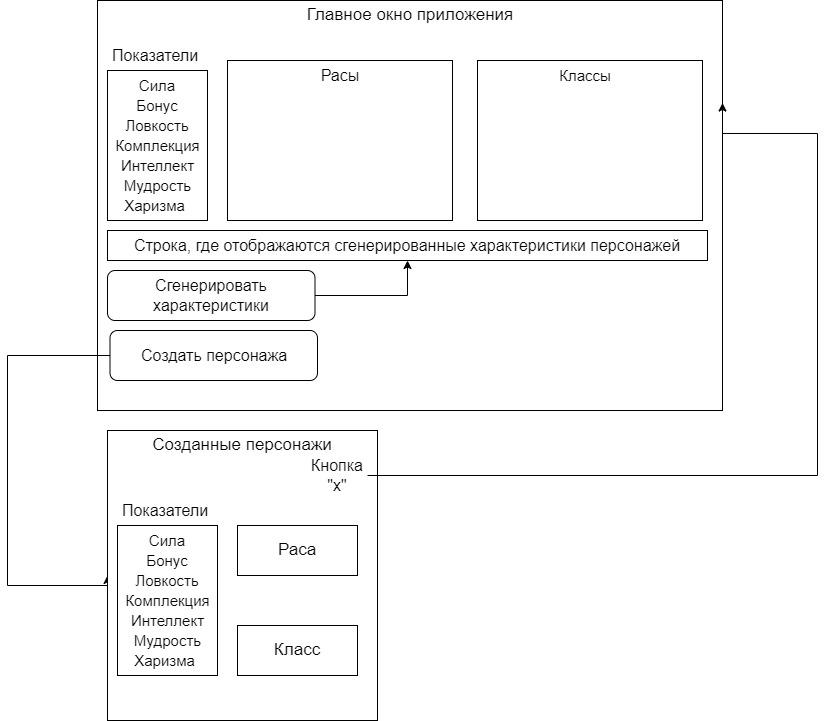


Рисунок 5 – Схема функционирования экранов

Такой подход создает единое пространство, где пользователь может управлять всей информацией о персонаже на одном экране, что способствует удобству использования приложения.

**4.3 Входные, выходные и промежуточные данные**

В данном разделе представлена структура входных, выходных и промежуточных данных приложения «Создание D&D персонажа». Раздел, посвященный структуре данных входа, выхода и промежуточных этапов приложения «Создание D&D персонажа», играет ключевую роль в обеспечении функциональности и эффективности данного программного продукта.

Этот раздел призван подробно описать весь жизненный цикл данных, начиная от ввода пользовательских параметров и заканчивая отображением сгенерированных персонажей.

Разработка структуры данных основывается на концепции удобства использования и оптимизации процесса создания персонажей в рамках Dungeons & Dragons (D&D).

Опишем данную структуру подробнее:

Входные данные:

Пользователь взаимодействует с приложением, вводя различные параметры, такие как выбор расы, класса, характеристик и других важных аспектов персонажа. Ввод осуществляется с использованием предоставляемых интерфейсом элементов управления, что обеспечивает легкость восприятия и интуитивное понимание пользовательского ввода.

Форма для ввода пользовательских параметров, таких как раса, класс, характеристики и прочее.

Пользовательский ввод через элементы управления, предоставляемые интерфейсом приложения.

Промежуточные данные:

На этапе промежуточной обработки осуществляется анализ и преобразование введенных пользователем данных. Здесь происходит подготовка данных для дальнейшей генерации персонажа. Временные данные используются для хранения результатов обработки и обеспечивают целостность процесса генерации.

Процесс обработки введенных данных, их преобразование и подготовка для дальнейшей обработки.

Временные данные, используемые для хранения промежуточных результатов при работе с генерацией персонажей.

Выходные данные:

Сгенерированный персонаж отображается на пользовательском интерфейсе в соответствии с выбранными параметрами. Эти выходные данные предоставляют полную информацию о созданном персонаже, обеспечивая пользователей всей необходимой информацией для дальнейших игровых сессий. Структура файлов, в свою очередь, соответствует определенным схемам данных, обеспечивая правильное сохранение и загрузку информации о персонажах.

Сгенерированный персонаж с учетом выбранных параметров и характеристик.

Отображение информации о персонаже на пользовательском интерфейсе.

Структура файлов должна соответствовать определенным схемам данных, учитывая особенности сохранения и загрузки информации о персонажах.

В общем, данный раздел предоставляет всесторонний обзор процесса работы с данными, отражая ключевые этапы взаимодействия приложения с пользовательским вводом, промежуточной обработки и окончательного формирования выходных данных.

**4.4 Архитектура ПС (ПМ)**

Класс MainVM представляет ViewModel для создания персонажа в приложении. Давайте рассмотрим каждое из свойств и методов класса:

private string str\_start\_box;

Тип: string

Описание: Приватное поле, представляющее значение силы в строковом формате.

public string Str\_start\_box;

Тип: string

Описание: Публичное свойство для доступа к значению силы. Имеет метод уведомления об изменении (OnPropertyChanged).

private string dex\_start\_box;

Тип: string

Описание: Приватное поле, представляющее значение ловкости в строковом формате.

public string Dex\_start\_box;

Тип: string

Описание: Публичное свойство для доступа к значению ловкости. Имеет метод уведомления об изменении.

private string con\_start\_box;

Тип: string

Описание: Приватное поле, представляющее значение телосложения в строковом формате.

public string Con\_start\_box;

Тип: string

Описание: Публичное свойство для доступа к значению телосложения. Имеет метод уведомления об изменении.

private string inte\_start\_box;

Тип: string

Описание: Приватное поле, представляющее значение интеллекта в строковом формате.

public string Inte\_start\_box;

Тип: string

Описание: Публичное свойство для доступа к значению интеллекта. Имеет метод уведомления об изменении.

private string wis\_start\_box;

Тип: string

Описание: Приватное поле, представляющее значение мудрости в строковом формате.

public string Wis\_start\_box;

Тип: string

Описание: Публичное свойство для доступа к значению мудрости. Имеет метод уведомления об изменении.

private string cha\_start\_box;

Тип: string

Описание: Приватное поле, представляющее значение харизмы в строковом формате.

public string Cha\_start\_box;

Тип: string

Описание: Публичное свойство для доступа к значению харизмы. Имеет метод уведомления об изменении.

private Character character\_make;

Тип: Character

Описание: Приватное поле, представляющее объект персонажа в процессе создания.

public Character Character\_make;

Тип: Character

Описание: Публичное свойство для доступа к объекту персонажа. Имеет метод уведомления об изменении.

private Rase selectedRase;

Тип: Rase

Описание: Приватное поле, представляющее выбранную расу персонажа.

public Rase SelectedRase;

Тип: Rase

Описание: Публичное свойство для доступа к выбранной расе персонажа. Имеет метод уведомления об изменении.

private dndClass selectedClass;

Тип: dndClass

Описание: Приватное поле, представляющее выбранный класс персонажа.

public dndClass SelectedClass;

Тип: dndClass

Описание: Публичное свойство для доступа к выбранному классу персонажа. Имеет метод уведомления об изменении.

private string lab;

Тип: string

Описание: Приватное поле, представляющее строку для отображения информации о созданных характеристиках.

public string Lab;

Тип: string

Описание: Публичное свойство для доступа к строке. Имеет метод уведомления об изменении.

private RelayCommand stats\_create;

Тип: RelayCommand

Описание: Приватное поле, представляющее команду для создания случайных характеристик.

public RelayCommand Stats\_create;

Тип: RelayCommand

Описание: Публичное свойство для доступа к команде создания случайных характеристик. Имеет метод уведомления об изменении.

private RelayCommand char\_create;

Тип: RelayCommand

Описание: Приватное поле, представляющее команду для создания персонажа.

public RelayCommand Char\_create;

Тип: RelayCommand

Описание: Публичное свойство для доступа к команде создания персонажа. Имеет метод уведомления об изменении.

Методы:

public MainVM(Character cr);

Тип: Конструктор

Описание: Конструктор класса. Принимает объект персонажа при создании экземпляра класса.

private void OnPropertyChanged(string prop);

Тип: void

Описание: Метод для уведомления об изменении свойства. Реализует интерфейс INotifyPropertyChanged.

private void ExecuteStatsCreate(object obj);

Тип: void

Описание: Метод, выполняющий создание случайных характеристик персонажа.

private void ExecuteCharCreate(object obj);

Тип: void

Описание: Метод, выполняющий создание персонажа на основе выбранных характеристик, расы и класса.

Класс MainVM предназначен для взаимодействия с UI и управления процессом создания персонажа в приложении.

MainVM2 – это класс ViewModel, отвечающий за предоставление данных и управление представлением. Ваша XAML-страница привязана к экземпляру этого класса.

Класс MainVM2 представляет ViewModel в приложении и содержит свойства, которые используются для отображения характеристик персонажа. Давайте рассмотрим каждое из свойств и описываем их:

Character character;

Тип: Character

Описание: Приватное поле, представляющее объект персонажа в приложении.

MainVM2(Character cr);

Тип: Конструктор

Описание: Конструктор класса, принимающий объект типа Character и инициализирующий поле character.

string Str\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление характеристики силы (Str) персонажа.

string Dex\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление характеристики ловкости (Dex) персонажа.

string Con\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление характеристики телосложения (Con) персонажа.

string Inte\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление характеристики интеллекта (Inte) персонажа.

string Wis\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление характеристики мудрости (Wis) персонажа.

string Cha\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление характеристики харизмы (Cha) персонажа.

string Rase\_cha;

Тип: string

Описание: Публичное свойство, возвращающее имя расы (Rase.Name) персонажа.

string Class\_cha;

Тип: string

Описание: Публичное свойство, возвращающее имя класса (Game\_class.Name) персонажа.

string Str\_dise\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление бонуса к броску кубика при проверке силы (Str\_dise) персонажа.

string Dex\_dise\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление бонуса к броску кубика при проверке ловкости (Dex\_dise) персонажа.

string Con\_dise\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление бонуса к броску кубика при проверке телосложения (Con\_dise) персонажа.

string Inte\_dise\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление бонуса к броску кубика при проверке интеллекта (Inte\_dise) персонажа.

string Wis\_dise\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление бонуса к броску кубика при проверке мудрости (Wis\_dise) персонажа.

string Cha\_dise\_cha;

Тип: string

Описание: Публичное свойство, возвращающее строковое представление бонуса к броску кубика при проверке харизмы (Cha\_dise) персонажа.

Этот класс предоставляет свойства для отображения характеристик персонажа и используется в UI для связывания данных.

Char\_Show.xaml – это представление, которое отображает информацию о характере, используя данные из MainVM2. Класс Char\_Show представляет XAML-представление, которое отображает информацию о характере, используя данные из MainVM2. Давайте рассмотрим его подробно:

public partial class Char\_Show : Window

Тип: Класс

Описание: Класс представления окна Char\_Show, который является потомком класса Window в WPF.

public Char\_Show(MainVM2 vm)

Тип: Конструктор

Описание: Конструктор класса Char\_Show, который принимает объект типа MainVM2 в качестве параметра при создании экземпляра класса.

InitializeComponent();

Тип: void

Описание: Метод инициализации компонентов окна. В WPF он генерируется автоматически на основе XAML-файла. Здесь он вызывается при создании окна.

DataContext = vm;

Тип: object

Описание: Устанавливает контекст данных для окна, используя объект MainVM2 в качестве источника данных. Это означает, что элементы управления в XAML могут привязываться к свойствам MainVM2 для отображения информации о характере.

Класс Char\_Show является частью паттерна проектирования MVVM (Model-View-ViewModel), где MainVM2 является ViewModel, предоставляющей данные, а Char\_Show представляет XAML-представление, отвечающее за отображение данных на пользовательском интерфейсе.

Character.cs – класс, представляющий персонажа с характеристиками и связанными с ними методами. Это класс, представляющий персонажа в контексте игры Dungeons & Dragons. Класс содержит характеристики персонажа, а также связанные методы для работы с этими характеристиками. Давайте подробно рассмотрим свойства и методы класса:

Свойства:

Stats: Объект типа stats, представляющий характеристики персонажа, такие как сила, ловкость, телосложение и т. д.

Race: Объект типа Rase, представляющий расу персонажа.

Class: Объект типа dndClass, представляющий класс персонажа.

Дополнительные свойства и методы в контексте использования с MainVM2 (не видны в предоставленном коде):

Str\_cha, Dex\_cha, ..., Cha\_dise\_cha: Свойства, предоставляющие строковые представления характеристик персонажа для отображения в пользовательском интерфейсе.

Rase\_cha и Class\_cha: Свойства, предоставляющие строки с именами расы и класса персонажа соответственно.

Str\_dise\_cha, Dex\_dise\_cha, ..., Cha\_dise\_cha: Свойства, предоставляющие строковые представления бонусов к броскам кубика для соответствующих характеристик.

Этот класс является ключевым элементом для представления и работы с персонажем в контексте приложения Dungeons & Dragons Character Creator.

dndClass.cs – класс, представляющий класс персонажа в Dungeons & Dragons с его характеристиками. Это класс, представляющий класс персонажа в контексте игры Dungeons & Dragons. Класс содержит характеристики класса персонажа, такие как основные характеристики (сила, ловкость, телосложение и т. д.), а также связанные методы. Давайте подробно рассмотрим свойства и методы класса:

Свойства:

Name: Строка, представляющая имя класса персонажа.

StrMain: Целочисленное значение, представляющее основную характеристику «Сила» для данного класса.

DexMain: Целочисленное значение, представляющее основную характеристику «Ловкость» для данного класса.

ConMain: Целочисленное значение, представляющее основную характеристику «Телосложение» для данного класса.

InteMain: Целочисленное значение, представляющее основную характеристику «Интеллект» для данного класса.

WisMain: Целочисленное значение, представляющее основную характеристику «Мудрость» для данного класса.

ChaMain: Целочисленное значение, представляющее основную характеристику «Харизма» для данного класса.

Методы:

ToString(): Переопределенный метод, возвращающий строковое представление класса персонажа. В этом методе могут использоваться различные свойства для формирования полной информации о классе.

Дополнительные методы в контексте использования (не видны в предоставленном коде): Возможны дополнительные методы, предназначенные для выполнения действий, связанных с классом персонажа (например, методы для применения особенностей класса).

Дополнительные свойства и методы в контексте использования с MainVM2 (не видны в предоставленном коде):

StrMain, DexMain, ..., ChaMain: Свойства, предоставляющие целочисленные значения основных характеристик для отображения в пользовательском интерфейсе.

Другие методы или свойства, используемые для формирования информации о классе персонажа в интерфейсе.

Этот класс играет ключевую роль в представлении класса персонажа и взаимодействии с характеристиками персонажа в приложении Dungeons & Dragons Character Creator.

Rase.cs – класс, представляющий расу персонажа с ее характеристиками.

это класс, представляющий расу (рассу) персонажа в контексте игры Dungeons & Dragons. Класс содержит характеристики расы, такие как бонусы к характеристикам, внешний вид расы и дополнительные особенности. Давайте подробно рассмотрим свойства и методы этого класса:

Свойства:

Id: Уникальный идентификатор расы.

Name: Строка, представляющая имя расы.

Appearance: Строка, описывающая внешний вид расы.

Str\_bonus: Целочисленное значение, представляющее бонус к характеристике «Сила» для персонажей данной расы.

Dex\_bonus: Целочисленное значение, представляющее бонус к характеристике «Ловкость» для персонажей данной расы.

Con\_bonus: Целочисленное значение, представляющее бонус к характеристике «Телосложение» для персонажей данной расы.

Inte\_bonus: Целочисленное значение, представляющее бонус к характеристике «Интеллект» для персонажей данной расы.

Wis\_bonus: Целочисленное значение, представляющее бонус к характеристике «Мудрость» для персонажей данной расы.

Cha\_bonus: Целочисленное значение, представляющее бонус к характеристике «Харизма» для персонажей данной расы.

Rase\_bonus: Строка, содержащая описание дополнительных особенностей и бонусов данной расы.

Методы:

ToString(): Переопределенный метод, возвращающий строковое представление расы. В этом методе могут использоваться различные свойства для формирования полной информации о расе.

Дополнительные свойства и методы в контексте использования (не видны в предоставленном коде):

Свойства, предоставляющие бонусы к характеристикам для отображения в пользовательском интерфейсе (Str\_bonus, Dex\_bonus, ..., Cha\_bonus).

Свойства или методы, используемые для формирования информации о расе в интерфейсе.

Этот класс играет ключевую роль в представлении расы персонажа и взаимодействии с характеристиками персонажа в приложении Dungeons & Dragons Character Creator.

start.cs – класс, связанный с начальной настройкой персонажа, выбором расы и класса. Это класс, отвечающий за начальную настройку персонажа, включая выбор расы и класса в контексте игры Dungeons & Dragons. Давайте подробно рассмотрим свойства и методы этого класса:

Свойства:

SelectedRase: Объект типа Rase, представляющий выбранную расу персонажа.

SelectedClass: Объект типа dndClass, представляющий выбранный класс персонажа.

События:

Rase\_Chose: Событие, возникающее при выборе расы и класса. Обработчик этого события выполняет определенные действия, в данном случае, отображение сообщения с использованием MessageBox.

Методы:

Rase\_Chose\_Execute(): Метод, выполняющийся при выборе расы и класса. В данном случае, отображает сообщение с использованием MessageBox, если выбраны и раса, и класс.

Дополнительные свойства и методы в контексте использования (не видны в предоставленном коде):

Свойства и методы, обеспечивающие взаимодействие с выбором расы и класса в пользовательском интерфейсе.

Логика обработки выбора пользователя при создании персонажа.

Этот класс играет важную роль в процессе начальной настройки персонажа, включая выбор его расы и класса, и взаимодействует с другими компонентами приложения Dungeons & Dragons Character Creator для правильной инициализации персонажа перед созданием.

stats.cs – класс, представляющий характеристики персонажа и содержащий вычисляемые свойства для расчета модификаторов характеристик. Это класс, представляющий характеристики персонажа в контексте игры Dungeons & Dragons. Давайте рассмотрим его свойства и вычисляемые свойства более подробно:

Свойства:

Rase: Объект типа Rase, представляющий расу персонажа.

Game\_class: Объект типа dndClass, представляющий класс персонажа.

Характеристики персонажа:

Str\_start, Dex\_start, Con\_start, Inte\_start, Wis\_start, Cha\_start: Исходные значения характеристик, задаваемые при создании персонажа.

Модификаторы характеристик (вычисляемые свойства):

Str, Dex, Con, Inte, Wis, Cha: Модифицированные значения характеристик, учитывающие бонусы от расы и класса.

Str\_dise, Dex\_dise, Con\_dise, Inte\_dise, Wis\_dise, Cha\_dise: Модификаторы для бросков кубика при проверке соответствующих характеристик.

Методы:

OnPropertyChanged(string prop): Метод, вызывающий событие изменения свойства для реализации механизма уведомлений об изменениях.

Дополнительные методы, не видные в предоставленном коде:

Логика расчета модификаторов характеристик.

Взаимодействие с другими частями приложения для корректного обновления и отображения характеристик персонажа.

Этот класс играет важную роль в представлении и обработке характеристик персонажа, включая учет бонусов от расы и класса, а также вычисление модификаторов для использования в игровых механиках.

Notify.cs – базовый класс для реализации уведомлений об изменении свойств. Класс Notify представляет базовую реализацию механизма уведомлений об изменениях свойств, реализуя интерфейс INotifyPropertyChanged. Этот механизм используется для обновления пользовательского интерфейса (UI), когда происходят изменения в свойствах объектов.

Описание методов и свойств:

Свойство PropertyChanged:

Тип: event PropertyChangedEventHandler

Описание: Событие, которое срабатывает при изменении свойств объекта. Подписываясь на это событие, другие части кода могут реагировать на изменения в объекте.

Метод OnPropertyChanged(string prop):

Тип: void

Параметры:

prop - Название изменённого свойства

Описание: Вызывает событие PropertyChanged, уведомляя подписчиков (например, элементы пользовательского интерфейса) о том, что свойство объекта было изменено. Метод принимает название изменённого свойства в качестве аргумента.

Назначение класса:

Класс Notify предоставляет базовую функциональность для реализации паттерна "Издатель-Подписчик" (Observer). При изменении свойств объекта, унаследованного от Notify, объект автоматически уведомляет своих подписчиков (например, элементы UI), что позволяет им обновляться и отражать актуальное состояние объекта. Этот подход часто используется в паттерне проектирования MVVM (Model-View-ViewModel) для обновления пользовательского интерфейса в приложениях WPF и других фреймворках.

Dwarf.cs – статический класс, представляющий расу «Дварф» с ее характеристиками.

Elf.cs – статический класс, представляющий расу «Эльф» с ее характеристиками.

Gearforged.cs – статический класс, представляющий расу «Кованный» с ее характеристиками.

Halfling.cs – статический класс, представляющий расу «Полурослик» с ее характеристиками.

Human.cs – статический класс, представляющий расу «Человек» с ее характеристиками.

Orc.cs – статический класс, представляющий расу «Орк» с ее характеристиками.

Tabaxi.cs – статический класс, представляющий расу «Табакси» с ее характеристиками.

Tiefling.cs – статический класс, представляющий расу «Тифлинг» с ее характеристиками.

Artificer.cs – статический класс, представляющий класс «Изобретатель» с его характеристиками.

Fighter.cs – статический класс, представляющий класс «Воин» с его характеристиками.

Monk.cs – статический класс, представляющий класс «Монах» с его характеристиками.

Paladin.cs – статический класс, представляющий класс «Паладин» с его характеристиками.

Ranger.cs – статический класс, представляющий класс «Следопыт» с его характеристиками.

Давайте представим все описанные выше нами классы в виде диаграммы классов для наглядности (рис.6).

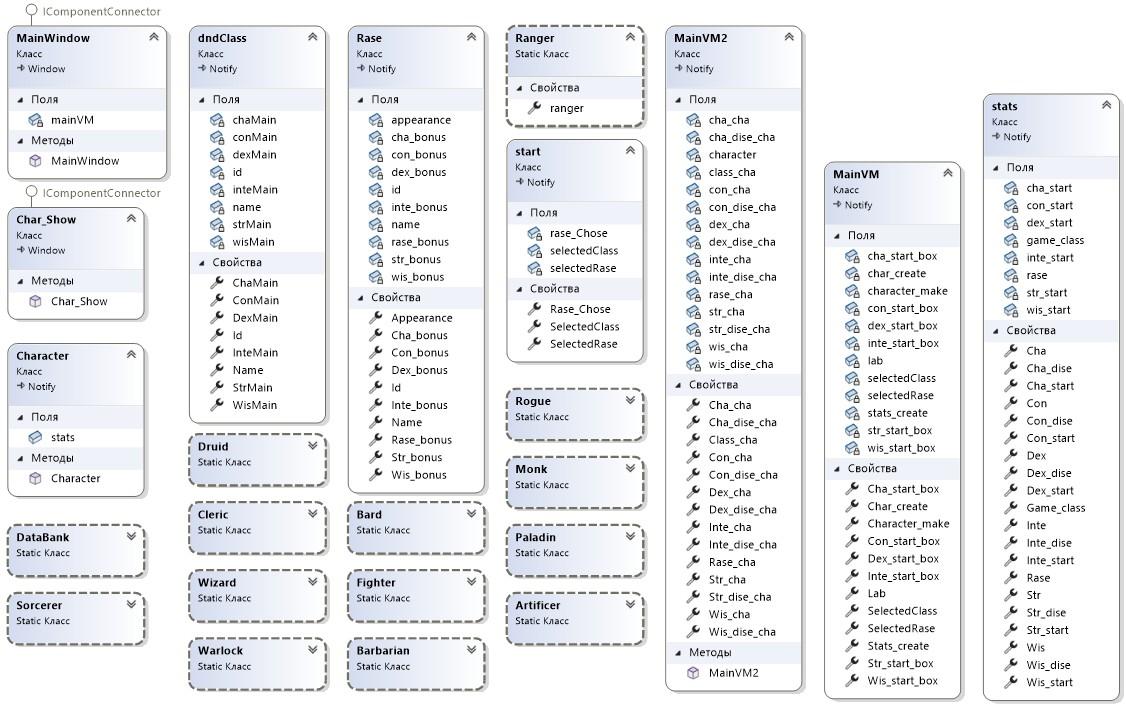


Рисунок 6 – Диаграмма классов

5 ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ

**5.1 План тестирования**

Тестирование является важной частью жизненного цикла разработки программного обеспечения, направленной на обеспечение качества и надежности создаваемой системы.

В данном проекте осуществляется ручное тестирование, так как оно более гибко подстраивается под изменения в коде, а также учитывает человеческий фактор в оценке удобства использования. Ручное тестирование выбрано в силу простоты и маневренности в процессе разработки, что позволяет более гибко реагировать на изменения в функциональности и оперативно вносить коррективы. Также, учитывая ограниченные ресурсы проекта, ручное тестирование обеспечивает достаточный уровень проверки качества.

План тестирования приложения:

1. Проверка генерации характеристик.
   1. Вводим различные параметры и проверяем, что генерация происходит корректно.
   2. Проверка создания персонажа.
   3. Выбор разных параметров (раса, класс) и проверка, что персонаж создается с учетом этих параметров.
   4. Проверка обновления интерфейса.
   5. Изменение параметров и проверка, что интерфейс обновляется соответственно.
2. Интеграционное тестирование:
   1. Проверка взаимодействия компонентов.
   2. Выполнение различных действий и проверка, что компоненты взаимодействуют корректно.
3. Тестирование производительности:
   1. Проверка времени отклика системы.
   2. Оценка времени отклика при различных сценариях использования.
4. Тестирование совместимости:
   1. Проверка работы на операционной системе Windows.
   2. Установка и тестирование на различных версиях Windows.
5. Тестирование удобства использования:
   1. Оценка интуитивности интерфейса.
   2. Проведение тестов с участием различных пользователей для оценки удобства использования.

**5.2 Результаты тестирования**

В ходе тестирования приложения «Создание D&D персонажа» было проведено ручное тестирование.

В результате тестов обнаружена ошибка при вводе символов, отличных от чисел, в поля характеристик.

Интерфейс обновляется в соответствии с выбранными параметрами, и приложение отвечает на действия пользователя в течение заданных временных рамок.

Тестирование также включило проверку работы на операционной системе Windows, обеспечивая совместимость с различными версиями.

**5.3 Оптимизация ПС (ПМ)**

Процесс оптимизации приложения «Создание D&D персонажа» включал тщательный анализ результатов тестирования с целью выявления узких мест и определения возможных улучшений. Оптимизация была направлена на улучшение производительности, устранение возможных ошибок и повышение эффективности работы приложения.

Этот процесс включал идентификацию время затратных операций, оптимизацию алгоритмов, улучшение структуры приложения, оптимизацию интерфейса и проверку вводимых значений.

Результатом стало существенное улучшение общей производительности и отзывчивости приложения, что подтверждено успешным прохождением последующих тестов.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Приложение «Создание D&D персонажа» предоставляет интуитивно понятный интерфейс, где пользователь может задавать характеристики сам или генерировать их, выбирая расу и класс, создавая различных игровых персонажей, тем самым кастомизируя персонажей и повышать свою индивидуальность в игровом мире.

Чтобы войти в приложение, пользователь должен зайти в загрузки или на рабочий стол и найти ярлык с приложением, нажав на него правой кнопкой мыши несколько раз.

Когда приложение запуститься, то пользователь окажется в основном экране приложения с основным его функционалом (рис.7).

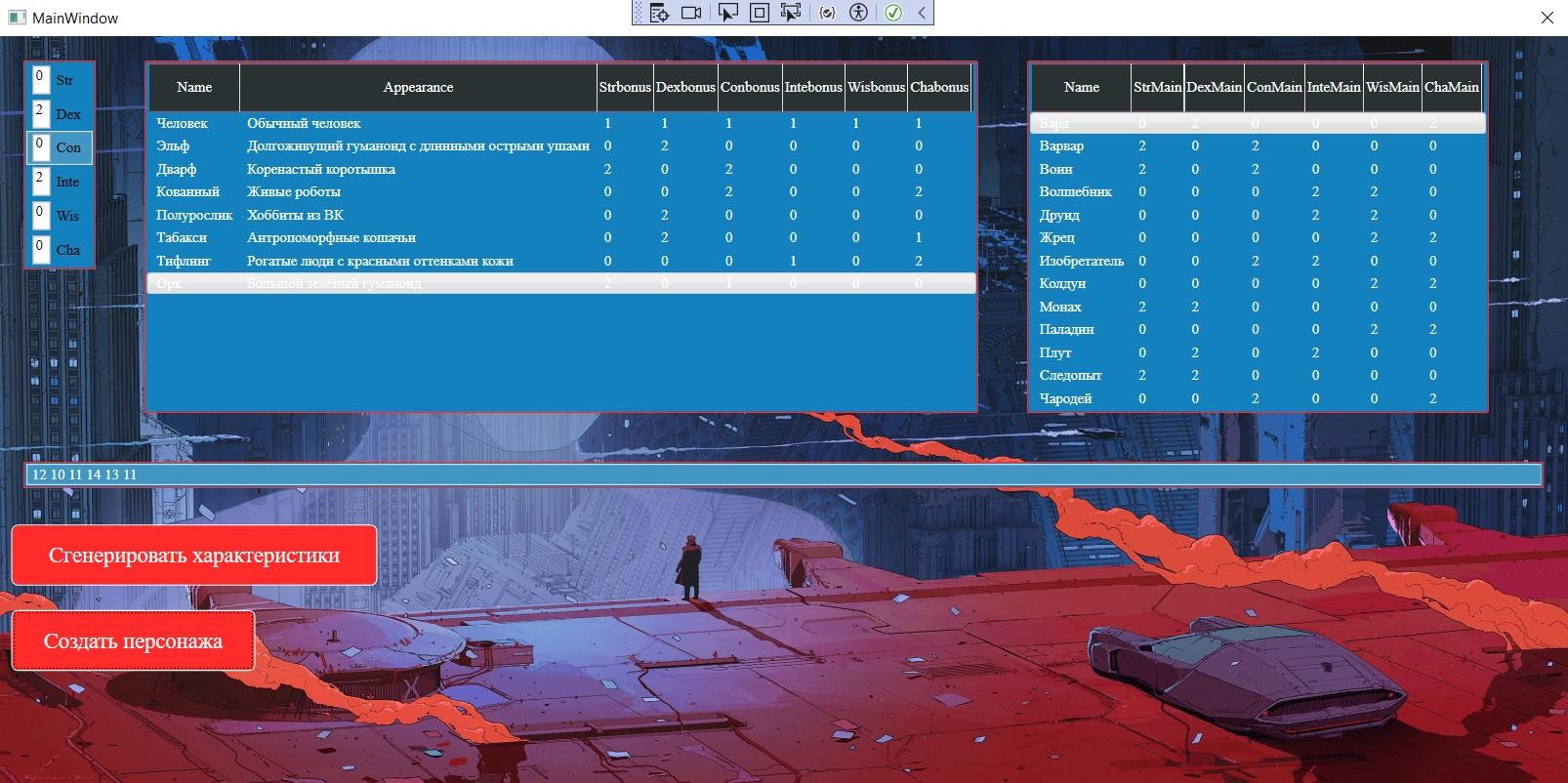


Рисунок 7 – Общий вид главного экрана приложения

Пользователь может сгенерировать значения характеристик, которые будут автоматически заполнены в соответствующих полях. Для этого необходимо нажать кнопку «Сгенерировать характеристики». Значения всех характеристик будут отображаться в отдельном поле (рис.8).



Рисунок 8 – Кнопка «Сгенерировать характеристики»

И тогда в текстовом поле отобразятся сгенерированные характеристика для персонажа (рис.9).



Рисунок 9 – Сгенерированные характеристики

Пользователь может ввести значения характеристик вручную, указав их в соответствующих полях (рис.10).

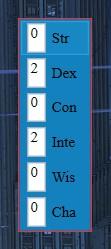


Рисунок 10 – Ручное введение характеристик

Выбор расы и класса. После ввода или генерации характеристик пользователь может выбрать расу и класс для своего персонажа. Для этого предоставлены соответствующие списки, из которых пользователь может выбрать желаемые опции (рис.11).



Рисунок 11 – Расы, классы и характеристики персонажа

В приложении есть так называемые таблицы с классами и расами для персонажей (рис.12-13).

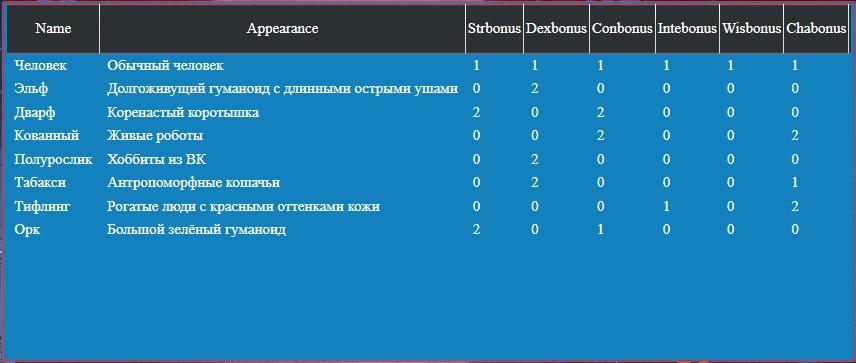


Рисунок 12 – Классы

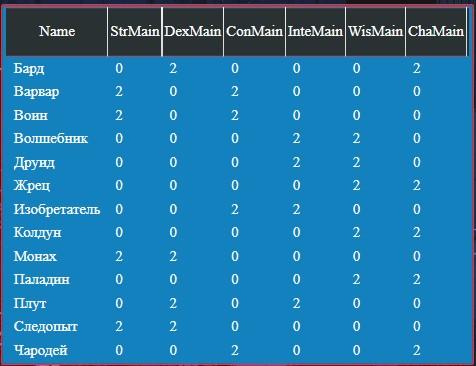


Рисунок 13 – Расы

Перед созданием персонажа убедитесь, что все необходимые характеристики заданы. При возникновении проблем или ошибок ввода, приложение предоставит сообщения с предупреждениями или подсказками. Если характеристики или выбор расы/класса не заданы, приложение предупредит пользователя и не позволит создать персонажа (рис.14).

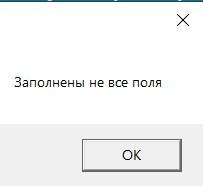
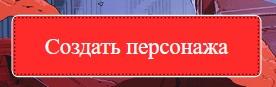


Рисунок 14 – Сообщение об ошибке

После задания характеристик, выбора расы и класса, пользователь может создать персонаж, нажав на кнопку «Создать персонажа» (рис.15).

  
Рисунок 15 – Кнопка «Создать персонажа»

В новом открывшимся окне создастся игровой персонаж с различным набором характеристик (рис.16).



Рисунок 16 – Окно с игровым персонажем

Для выхода из приложения можно воспользоваться стандартным способом закрытия окна.

Приложение «Создание D&D персонажа» предоставляет удобный способ создания уникальных персонажей для игры в Dungeons & Dragons, а его интуитивно понятный интерфейс обеспечивает легкость в использовании.

**ЗАКЛЮЧЕНИЕ**

Было разработано десктопное приложение «Создание D&D персонажа» для операционной системы Windows. Также были успешно реализованы поставленные задачи, полностью выполнив проект. Разработанное программное средство представляет собой десктопное приложение с интуитивно понятным пользовательским интерфейсом, позволяющим генерировать персонажей для игры в Dungeons & Dragons.

В процессе разработки был успешно использован инструментарий Visual Studio, с языками программирования C# и XAML, а также фреймворк WPF. Это позволило нам создать приложение с высокой производительностью, эффективно взаимодействующее с пользовательским интерфейсом, ориентированным на создание персонажей и управление их характеристиками.

Работа над проектом включала в себя проектирование и реализацию функционала генерации характеристик, создание персонажа с учетом выбранных параметров, а также обновление интерфейса в реальном времени при изменении параметров. Пройденное ручное тестирование подтвердило стабильную и корректную работу приложения, а его оптимизация позволила достичь высокой производительности.

Созданное приложение предоставляет удобный и интуитивно понятный способ создания персонажей для игроков Dungeons & Dragons, и представляет собой готовую основу для дальнейших улучшений, добавления новых функций и расширения возможностей. В заключение, данное приложение может служить отличным инструментом для игровых сессий и погружения в мир фэнтези.

Основные преимущества приложения включают в себя повышение эффективности взаимодействия с приложением, сокращение времени подготовки к игре, повышение доступности для новичков и улучшение общего опыта пользователей. Привлекательный пользовательский интерфейс и современный дизайн направлены на увеличение интереса к D&D и привлечение новых участников в игровое сообщество.

Таким образом, разработанное приложение не только решает поставленную задачу, но также предоставляет основу для будущего развития и улучшения функционала, что делает его перспективным инструментом для игровых сессий в мире Dungeons & Dragons.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Wizards of the Coast. URL: https://dnd.wizards.com
2. Игромания. "Что такое Dungeons & Dragons и как в это играть. Краткий справочник для новичков." URL: https://www.igromania.ru/article/31235/Chto\_takoe\_Dungeons\_and\_Dragons\_i\_kak\_v\_yeto\_igrat.\_Kratkiy\_spravochnik\_dlya\_novichkov.html
3. D&D.SU. "Дварф" URL: https://dnd.su/race/78-dwarf/
4. DM Stuff. Каталог рас. URL: http://dm-stuff.ru/race-catalog/
5. Microsoft Docs. "Model-View-ViewModel (MVVM) architecture for .NET MAUI" URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm
6. Metanit. "WPF Tutorial" URL: https://metanit.com/sharp/wpf/22.1.php
7. TV Besedka. "Шаблон MVVM - просто о сложном. Пример на C#" URL: https://tvbesedka.com/article/pattern\_mvvm\_-\_prosto\_o\_slozhnom\_primer\_na\_c\_644
8. WebDelphi. "Первое приложение WPF" URL: https://csharp.webdelphi.ru/pervoe-prilozhenie-wpf/
9. Metanit. "Основы WPF" URL: https://metanit.com/sharp/wpf/1.2.php
10. StackRU. "Как добавить стиль в XAML для окна, созданного из кода с использованием MVVM" URL: https://stackru.com/questions/30763571/kak-dobavit-stil-v-xaml-dlya-okna-sozdannogo-iz-koda-s-mvvm

**ПРИЛОЖЕНИЯ**

**Приложение А: MainWindow.xaml.cs**

using dnd\_char\_create.dataContext.Classes;

using dnd\_char\_create.dataContext.Rases;

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using dnd\_char\_create.dataContext;

using dnd\_char\_create.model.Base;

using System.ComponentModel;

using dnd\_char\_create.VIewModel;

namespace dnd\_char\_create

{

public partial class MainWindow : Window

{

MainVM mainVM = new MainVM();

public MainWindow()

{

InitializeComponent();

DataContext = mainVM;

raseView.ItemsSource = new List<Rase>() { Human.human ,Elf.elf, Dwarf.dwarf, Gearforged.gearforged, Halfling.halfling, Tabaxi.tabaxi, Tiefling.tiefling, Orc.orc};

//Character cha = new Character();

classView.ItemsSource = new List<dndClass>() { Bard.bard, Barbarian.barbarian,Fighter.fighter, Wizard.wizard,Druid.druid,Cleric.cleric,Artificer.artificer,Warlock.warlock,Monk.monk,Paladin.paladin,Rogue.rogue,Ranger.ranger,Sorcerer.sorcerer};

}

}

}

**Приложение Б: MainWindow.xaml**

<Window x:Class="dnd\_char\_create.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:dnd\_char\_create" xmlns:model="clr-namespace:dnd\_char\_create.model" d:DataContext="{d:DesignInstance Type=model:start}"

mc:Ignorable="d"

Title="MainWindow" Height="650" Width="1300" ResizeMode="NoResize">

<Window.Resources>

<!-- Стиль для ListView -->

<Style TargetType="ListView">

<Setter Property="Background" Value="#1381bd"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="BorderBrush" Value="#FF2B2B"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="Margin" Value="20"/>

<Setter Property="FontFamily" Value="Times New Roman"/>

<Style.Triggers>

<!-- Эффект при наведении -->

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="#1381bd"/>

</Trigger>

</Style.Triggers>

</Style>

<!-- Стиль для GridViewColumnHeader -->

<Style TargetType="GridViewColumnHeader">

<Setter Property="Background" Value="#293133"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="BorderBrush" Value="#FF2B2B"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="Height" Value="40"/>

<Setter Property="FontFamily" Value="Times New Roman"/>

<Style.Triggers>

<!-- Эффект при наведении -->

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="#2A363B"/>

</Trigger>

</Style.Triggers>

</Style>

<!-- Стиль для кнопки -->

<Style TargetType="Button">

<Setter Property="BorderBrush" Value="White"/>

<Setter Property="Background" Value="#FF2B2B"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="Margin" Value="10"/>

<Setter Property="Padding" Value="10"/>

<Setter Property="FontSize" Value="18"/>

<Setter Property="FontFamily" Value="Times New Roman"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border Background="{TemplateBinding Background}" BorderBrush="{TemplateBinding BorderBrush}" BorderThickness="{TemplateBinding BorderThickness}" CornerRadius="5">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</Window.Resources>

<Window.Background>

<!-- Добавление фоновой картинки -->

<ImageBrush ImageSource="C:\Users\79526\Downloads\photo\_5463410971784631679\_w.jpg"/>

</Window.Background>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<StackPanel>

<StackPanel Orientation="Horizontal">

<StackPanel Orientation="Vertical">

<ListView Name="charStats">

<StackPanel Orientation="Horizontal">

<TextBox x:Name="str\_start\_box" Width="15" Text="{Binding Str\_start\_box}">

</TextBox>

<Label>Str</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBox x:Name="dex\_start\_box" Width="15" Text="{Binding Dex\_start\_box}"></TextBox>

<Label>Dex</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBox x:Name="con\_start\_box" Width="15" Text="{Binding Con\_start\_box}"></TextBox>

<Label>Con</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBox x:Name="inte\_start\_box" Width="15" Text="{Binding Inte\_start\_box}"></TextBox>

<Label>Inte</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBox x:Name="wis\_start\_box" Width="15" Text="{Binding Wis\_start\_box}"></TextBox>

<Label>Wis</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<TextBox x:Name="cha\_start\_box" Width="15" Text="{Binding Cha\_start\_box}"></TextBox>

<Label>Cha</Label>

</StackPanel>

</ListView>

</StackPanel>

<StackPanel Orientation="Horizontal">

<ListView Name="raseView" SelectedItem="{Binding SelectedRase}">

<ListView.View>

<GridView>

<GridViewColumn DisplayMemberBinding="{Binding Path=Name}">Name</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=Appearance}">Appearance</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=Str\_bonus}">Str\_bonus</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=Dex\_bonus}">Dex\_bonus</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=Con\_bonus}">Con\_bonus</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=Inte\_bonus}">Inte\_bonus</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=Wis\_bonus}">Wis\_bonus</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=Cha\_bonus}">Cha\_bonus</GridViewColumn>

</GridView>

</ListView.View>

</ListView>

<ListView Name="classView" SelectedItem="{Binding SelectedClass}">

<ListView.View>

<GridView>

<GridViewColumn DisplayMemberBinding="{Binding Path=Name}">Name</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=StrMain}">StrMain</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=DexMain}">DexMain</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=ConMain}">ConMain</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=InteMain}">InteMain</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=WisMain}">WisMain</GridViewColumn>

<GridViewColumn DisplayMemberBinding="{Binding Path=ChaMain}">ChaMain</GridViewColumn>

</GridView>

</ListView.View>

</ListView>

</StackPanel>

</StackPanel>

<ListView>

<TextBlock Text="{Binding Lab}"></TextBlock>

</ListView>

<Button Height="50px" Width="300" Content="Сгенерировать характеристики" Command="{Binding Stats\_create}" HorizontalAlignment="Left"/>

<Button Height="50px" Width="200" Content="Создать персонажа" Command="{Binding Char\_create}" HorizontalAlignment="Left" />

</StackPanel>

</Grid>

</Window>

**Приложение В: MainVM.cs**

using dnd\_char\_create.model;

using dnd\_char\_create.model.Base;

using dnd\_char\_create.View;

using Labrary.Model.Base;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

namespace dnd\_char\_create.VIewModel

{

public class MainVM : Notify

{

private string str\_start\_box;

public string Str\_start\_box

{

get { return str\_start\_box; }

set

{

str\_start\_box = value;

OnPropertyChanged(nameof(Str\_start\_box));

}

}

private string dex\_start\_box;

public string Dex\_start\_box

{

get { return dex\_start\_box; }

set

{

dex\_start\_box = value;

OnPropertyChanged(nameof(Dex\_start\_box));

}

}

private string con\_start\_box;

public string Con\_start\_box

{

get { return con\_start\_box; }

set

{

con\_start\_box = value;

OnPropertyChanged(nameof(Con\_start\_box));

}

}

private string inte\_start\_box;

public string Inte\_start\_box

{

get { return inte\_start\_box; }

set

{

inte\_start\_box = value;

OnPropertyChanged(nameof(Inte\_start\_box));

}

}

private string wis\_start\_box;

public string Wis\_start\_box

{

get { return wis\_start\_box; }

set

{

wis\_start\_box = value;

OnPropertyChanged(nameof(Wis\_start\_box));

}

}

private string cha\_start\_box;

public string Cha\_start\_box

{

get { return cha\_start\_box; }

set

{

cha\_start\_box = value;

OnPropertyChanged(nameof(Cha\_start\_box));

}

}

private Character character\_make;

public Character Character\_make

{

get { return character\_make; }

set

{

character\_make = value;

OnPropertyChanged(nameof(Character\_make));

}

}

private Rase selectedRase;

public Rase SelectedRase

{

get { return selectedRase; }

set

{

selectedRase = value;

OnPropertyChanged(nameof(SelectedRase));

}

}

private dndClass selectedClass;

public dndClass SelectedClass

{

get { return selectedClass; }

set

{

selectedClass = value;

OnPropertyChanged(nameof(SelectedClass));

}

}

private string lab;

public string Lab

{

get { return lab; }

set { lab = value; OnPropertyChanged(nameof(Lab)); }

}

private RelayCommand stats\_create;

public RelayCommand Stats\_create

{

get

{

return stats\_create ??

(stats\_create = new RelayCommand(obj =>

{

int[] arr = { 0, 0, 0, 0, 0, 0 };

int min = 7;

int a = 0;

Random rnd = new Random();

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 4; j++)

{

a = rnd.Next(1, 6);

arr[i] += a;

if (a < min) min = a;

}

arr[i] -= min;

min = 7;

}

Lab = arr[0].ToString() + " " + arr[1].ToString() + " " + arr[2].ToString() + " " + arr[3].ToString() + " " + arr[4].ToString() + " " + arr[5].ToString();

}));

}

}

private RelayCommand char\_create;

public RelayCommand Char\_create

{

get

{

return char\_create ??

(char\_create = new RelayCommand(obj =>

{

if ((SelectedRase != null) && (SelectedClass != null) && (Dex\_start\_box != null) && (Con\_start\_box != null) && (Inte\_start\_box != null) && (Wis\_start\_box != null) && (Cha\_start\_box != null) && (Str\_start\_box != null))

{

string check\_cha="1";

bool check\_t\_f = true;

for (int i=0;i<6; i++)

{

if (i == 0)

{

check\_cha = Str\_start\_box;

}

if (i == 1)

{

check\_cha = Dex\_start\_box;

}

if (i == 2)

{

check\_cha = Con\_start\_box;

}

if (i == 3)

{

check\_cha = Inte\_start\_box;

}

if (i == 4)

{

check\_cha = Wis\_start\_box;

}

if (i == 5)

{

check\_cha = Cha\_start\_box;

}

for (int j = 0; j < check\_cha.Length; j++)

{

if (!(check\_cha[j] >= '0' && check\_cha[j] <= '9'))

{

check\_t\_f=false;

break;

}

}

if (!check\_t\_f)

{

break;

}

}

if (check\_t\_f)

{

Character\_make = new Character(Convert.ToInt32(Str\_start\_box), Convert.ToInt32(Dex\_start\_box), Convert.ToInt32(Con\_start\_box), Convert.ToInt32(Inte\_start\_box), Convert.ToInt32(Wis\_start\_box), Convert.ToInt32(Cha\_start\_box), SelectedRase, SelectedClass);

Char\_Show ch = new Char\_Show(Character\_make);

ch.Show();

}

else

{

MessageBox.Show("Поля заполнены некорректно");

}

}

else

{

MessageBox.Show("Заполнены не все поля");

}

}));

}

}

}

}

**Приложение Г: MainVM2.cs**

using dnd\_char\_create.model;

using dnd\_char\_create.model.Base;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.VIewModel

{

public class MainVM2 :Notify

{

Character character;

public MainVM2(Character cr)

{

character = cr;

}

private string str\_cha;

public string Str\_cha

{

get { return character.stats.Str.ToString(); }

}

private string dex\_cha;

public string Dex\_cha

{

get { return character.stats.Dex.ToString(); }

}

private string con\_cha;

public string Con\_cha

{

get { return character.stats.Con.ToString(); }

}

private string inte\_cha;

public string Inte\_cha

{

get { return character.stats.Inte.ToString(); }

}

private string wis\_cha;

public string Wis\_cha

{

get { return character.stats.Wis.ToString(); }

}

private string cha\_cha;

public string Cha\_cha

{

get { return character.stats.Cha.ToString(); }

}

private string rase\_cha;

public string Rase\_cha

{

get { return character.stats.Rase.Name; }

}

private string class\_cha;

public string Class\_cha

{

get { return character.stats.Game\_class.Name; }

}

private string str\_dise\_cha;

public string Str\_dise\_cha

{

get { return character.stats.Str\_dise.ToString(); }

}

private string dex\_dise\_cha;

public string Dex\_dise\_cha

{

get { return character.stats.Dex\_dise.ToString(); }

}

private string con\_dise\_cha;

public string Con\_dise\_cha

{

get { return character.stats.Con\_dise.ToString(); }

}

private string inte\_dise\_cha;

public string Inte\_dise\_cha

{

get { return character.stats.Inte\_dise.ToString(); }

}

private string wis\_dise\_cha;

public string Wis\_dise\_cha

{

get { return character.stats.Wis\_dise.ToString(); }

}

private string cha\_dise\_cha;

public string Cha\_dise\_cha

{

get { return character.stats.Cha\_dise.ToString(); }

}

}

}

**Приложение Д: Char\_Show.xaml**

<Window x:Class="dnd\_char\_create.View.Char\_Show"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:dnd\_char\_create.View" xmlns:viewmodel="clr-namespace:dnd\_char\_create.VIewModel" d:DataContext="{d:DesignInstance Type=viewmodel:MainVM}"

mc:Ignorable="d"

Title="MainWindow" Height="400" Width="300" ResizeMode="NoResize">

<Window.Resources>

<!-- Стиль для ListView -->

<Style TargetType="ListView">

<Setter Property="Background" Value="#8a3333"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="BorderBrush" Value="#FF2B2B"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="Margin" Value="20"/>

<Setter Property="FontFamily" Value="Times New Roman"/>

<Style.Triggers>

<!-- Эффект при наведении -->

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="#8a3333"/>

</Trigger>

</Style.Triggers>

</Style>

<!-- Стиль для GridViewColumnHeader -->

<Style TargetType="GridViewColumnHeader">

<Setter Property="Background" Value="#293133"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="BorderBrush" Value="#FF2B2B"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="Height" Value="40"/>

<Setter Property="FontFamily" Value="Times New Roman"/>

<Style.Triggers>

<!-- Эффект при наведении -->

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="#2A363B"/>

</Trigger>

</Style.Triggers>

</Style>

<!-- Стиль для кнопки -->

<Style TargetType="Button">

<Setter Property="BorderBrush" Value="White"/>

<Setter Property="Background" Value="#FF2B2B"/>

<Setter Property="Foreground" Value="White"/>

<Setter Property="Margin" Value="10"/>

<Setter Property="Padding" Value="10"/>

<Setter Property="FontSize" Value="18"/>

<Setter Property="FontFamily" Value="Times New Roman"/>

<Setter Property="BorderThickness" Value="1"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border Background="{TemplateBinding Background}" BorderBrush="{TemplateBinding BorderBrush}" BorderThickness="{TemplateBinding BorderThickness}" CornerRadius="5">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</Window.Resources>

<Window.Background>

<!-- Добавление фоновой картинки -->

<ImageBrush ImageSource="C:\Users\79526\OneDrive\Рабочий стол\Для лекций по истории\dnd\_char\_create\ac2ea54e87f2d81783143d8b5c4e1e26.jpg"/>

</Window.Background>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<StackPanel Orientation="Horizontal">

<ListView Name="charStats">

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Str\_cha}"/>

<Label>Str</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Str\_dise\_cha}"/>

<Label>StrDise</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Dex\_cha}"/>

<Label>Dex</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Dex\_dise\_cha}"/>

<Label>DexDise</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Con\_cha}"/>

<Label>Con</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Con\_dise\_cha}"/>

<Label>ConDise</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Inte\_cha}"/>

<Label>Inte</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Inte\_dise\_cha}"/>

<Label>InteDise</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Wis\_cha}"/>

<Label>Wis</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Cha\_cha}"/>

<Label>Cha</Label>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Label Content="{Binding Cha\_dise\_cha}"/>

<Label>ChaDise</Label>

</StackPanel>

</ListView>

<StackPanel Orientation="Vertical">

<ListView>

<Label Content="{Binding Rase\_cha}"></Label>

</ListView>

<ListView>

<Label Content="{Binding Class\_cha}"></Label>

</ListView>

</StackPanel>

</StackPanel>

</Grid>

</Window>

**Приложение Е: Character. cs**

using dnd\_char\_create.dataContext.Classes;

using dnd\_char\_create.dataContext.Rases;

using dnd\_char\_create.model.Base;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection.Metadata;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.model

{

public class Character:Notify

{

public stats stats;

public Character(int str\_start, int dex\_start, int con\_start, int inte\_start, int wis\_start, int cha\_start, Rase rase, dndClass dndClass)

{

stats = new stats()

{

Str\_start = str\_start,

Dex\_start = dex\_start,

Con\_start = con\_start,

Inte\_start = inte\_start,

Wis\_start = wis\_start,

Cha\_start = cha\_start,

Rase = rase,

Game\_class = dndClass,

};

}

}

}

**Приложение Ж: dndClass. cs**

using System;

using System.Collections.ObjectModel;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using dnd\_char\_create.model.Base;

namespace dnd\_char\_create.model

{

public class dndClass : Notify

{

private int id;

public int Id { get { return id; } set { id = value; OnPropertyChanged("Id"); } }

private string name;

public string Name { get { return name; } set { name = value; OnPropertyChanged("Name"); } }

private int strMain;

public int StrMain { get { return strMain; } set { strMain = value; OnPropertyChanged("StrMain"); } }

private int dexMain;

public int DexMain { get { return dexMain; } set { dexMain = value; OnPropertyChanged("DexMain"); } }

private int conMain;

public int ConMain { get { return conMain; } set { conMain = value; OnPropertyChanged("ConMain"); } }

private int inteMain;

public int InteMain { get { return inteMain; } set { inteMain = value; OnPropertyChanged("InteMain"); } }

private int wisMain;

public int WisMain { get { return wisMain; } set { wisMain = value; OnPropertyChanged("WisMain"); } }

private int chaMain;

public int ChaMain { get { return chaMain; } set { chaMain = value; OnPropertyChanged("chaMain"); } }

}

}

**Приложение З: Rase. cs**

using System;

using System.Collections.ObjectModel;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using dnd\_char\_create.model.Base;

namespace dnd\_char\_create.model

{

public class Rase : Notify

{

private int id;

public int Id { get { return id; } set { id = value; OnPropertyChanged("Id"); } }

private string name;

public string Name { get { return name; } set { name = value; OnPropertyChanged("Name"); } }

private string appearance;

public string Appearance { get { return appearance; } set { appearance = value; OnPropertyChanged("Appearance"); } }

private int str\_bonus;

public int Str\_bonus { get { return str\_bonus; } set { str\_bonus = value; OnPropertyChanged("Str\_bonus"); } }

private int dex\_bonus;

public int Dex\_bonus { get { return dex\_bonus; } set { dex\_bonus = value; OnPropertyChanged("Dex\_bonus"); } }

private int con\_bonus;

public int Con\_bonus { get { return con\_bonus; } set { con\_bonus = value; OnPropertyChanged("Con\_bonus"); } }

private int inte\_bonus;

public int Inte\_bonus { get { return inte\_bonus; } set { inte\_bonus = value; OnPropertyChanged("Inte\_bonus"); } }

private int wis\_bonus;

public int Wis\_bonus { get { return wis\_bonus; } set { wis\_bonus = value; OnPropertyChanged("Wis\_bonus"); } }

private int cha\_bonus;

public int Cha\_bonus { get { return cha\_bonus; } set { cha\_bonus = value; OnPropertyChanged("Cha\_bonus"); } }

private string rase\_bonus;

public string Rase\_bonus { get { return rase\_bonus; } set { rase\_bonus = value; OnPropertyChanged("Cha\_bonus"); } }

}

}

**Приложение И: start. cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using dnd\_char\_create.model.Base;

using Labrary.Model.Base;

using System.Collections.ObjectModel;

using System.Windows;

namespace dnd\_char\_create.model

{

public class start :Notify

{

private Rase selectedRase;

public Rase SelectedRase { get { return selectedRase; } set { selectedRase = value; OnPropertyChanged("SelectedRase"); } }

private dndClass selectedClass;

public dndClass SelectedClass { get { return selectedClass; } set { selectedClass = value; OnPropertyChanged("SelectedClass"); } }

private RelayCommand rase\_Chose;

public RelayCommand Rase\_Chose

{ get

{

return rase\_Chose ??

(rase\_Chose = new RelayCommand(obj =>

{

if ((SelectedRase != null )&& (SelectedClass!= null))

{

MessageBox.Show("");

}

}));

} }

}

}

**Приложение К: stats. cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

using dnd\_char\_create.model.Base;

namespace dnd\_char\_create.model

{

public class stats:Notify

{

private Rase rase;

public Rase Rase { get { return rase; } set { rase = value;OnPropertyChanged("Rase"); } }

private dndClass game\_class;

public dndClass Game\_class { get { return game\_class; } set { game\_class = value; OnPropertyChanged("Rase"); } }

private int str\_start;

public int Str\_start { get { return str\_start; } set { str\_start = value; OnPropertyChanged("Str\_start"); } }

public int Str => rase.Str\_bonus + str\_start;

public int Str\_dise => game\_class.StrMain + (int)((Str-10) / 2);

private int dex\_start;

public int Dex\_start { get { return dex\_start; } set { dex\_start = value; OnPropertyChanged("Dex\_start"); } }

public int Dex => rase.Dex\_bonus + dex\_start;

public int Dex\_dise => game\_class.DexMain + (int)((Dex - 10) / 2);

private int con\_start;

public int Con\_start { get { return con\_start; } set { con\_start = value; OnPropertyChanged("Con\_start"); } }

public int Con => rase.Con\_bonus + con\_start;

public int Con\_dise => game\_class.ConMain + (int)((Con - 10) / 2);

private int inte\_start;

public int Inte\_start { get { return inte\_start; } set { inte\_start = value; OnPropertyChanged("Inte\_start"); } }

public int Inte => rase.Inte\_bonus + inte\_start;

public int Inte\_dise => game\_class.InteMain + (int)((Inte - 10) / 2);

private int wis\_start;

public int Wis\_start { get { return wis\_start; } set { wis\_start = value; OnPropertyChanged("Wis\_start"); } }

public int Wis => rase.Wis\_bonus + wis\_start;

public int Wis\_dise => game\_class.WisMain + (int)((Wis - 10) / 2);

private int cha\_start;

public int Cha\_start { get { return cha\_start; } set { cha\_start = value; OnPropertyChanged("Cha\_start"); } }

public int Cha => rase.Cha\_bonus + cha\_start;

public int Cha\_dise => game\_class.ChaMain + (int)((Cha - 10) / 2);

}

}

**Приложение Л: Dwarf. cs**

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Rases

{

public static class Dwarf

{

public static Rase dwarf { get; set; } = new Rase()

{

Id = 3,

Name = "Дварф",

Appearance = "Коренастый коротышка",

Str\_bonus = 2,

Dex\_bonus = 0,

Con\_bonus = 2,

Inte\_bonus = 0,

Wis\_bonus = 0,

Cha\_bonus = 0,

Rase\_bonus = "Тёмное зрение, Дварфская устойчивость, Боевая тренировка, Знание камня, Языки: общий и дварфийский"

}

;

}

}

**Приложение М: Elf. cs**

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Rases

{

public static class Elf

{

public static Rase elf { get; set; } = new Rase()

{

Id = 2,

Name = "Эльф",

Appearance = "Долгоживущий гуманоид с длинными острыми ушами",

Str\_bonus = 0,

Dex\_bonus = 2,

Con\_bonus = 0,

Inte\_bonus = 0,

Wis\_bonus = 0,

Cha\_bonus = 0,

Rase\_bonus = "Тёмное зрение, Обострённые чувства, Наследие фей, Транс, Языки: общий и любой на выбор"

}

;

}

}

**Приложение Н: Gearforged. cs**

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Rases

{

public static class Gearforged

{

public static Rase gearforged { get; set; } = new Rase()

{

Id = 5,

Name = "Кованный",

Appearance = "Живые роботы",

Str\_bonus = 0,

Dex\_bonus = 0,

Con\_bonus = 2,

Inte\_bonus = 0,

Wis\_bonus = 0,

Cha\_bonus = 2,

Rase\_bonus = "Механическое тело, Стальная плоть, Прочная конструкция, Языки: общий и C#"

}

;

}

}

**Приложение О: Halfling. cs**

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Rases

{

public static class Halfling

{

public static Rase halfling { get; set; } = new Rase()

{

Id = 7,

Name = "Полурослик",

Appearance = "Хоббиты из ВК",

Str\_bonus = 0,

Dex\_bonus = 2,

Con\_bonus = 0,

Inte\_bonus = 0,

Wis\_bonus = 0,

Cha\_bonus = 0,

Rase\_bonus = "Везучий, Храбрый, Проворство полуросликов, Языки: общий и язык полуросликов"

}

;

}

}

**Приложение П: Human. cs**

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Rases

{

public static class Human

{

public static Rase human { get; set; } = new Rase()

{

Id= 1,

Name="Человек",

Appearance="Обычный человек",

Str\_bonus=1,

Dex\_bonus=1,

Con\_bonus=1,

Inte\_bonus=1,

Wis\_bonus=1,

Cha\_bonus=1,

Rase\_bonus="Языки: общий и любой на выбор"

};

}

}

**Приложение Р: Orc. cs**

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Rases

{

public static class Orc

{

public static Rase orc { get; set; } = new Rase()

{

Id = 8,

Name = "Орк",

Appearance = "Большой зелёный гуманоид",

Str\_bonus = 2,

Dex\_bonus = 0,

Con\_bonus = 1,

Inte\_bonus = 0,

Wis\_bonus = 0,

Cha\_bonus = 0,

Rase\_bonus = "Тёмное зрение, Агрессивный, Чутьё дикаря, Мощное телосложение, Языки: общий и орочий"

}

;

}

}

**Приложение С: Tabaxi. cs**

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Rases

{

public static class Tabaxi

{

public static Rase tabaxi { get; set; } = new Rase()

{

Id = 4,

Name = "Табакси",

Appearance = "Антропоморфные кошачьи",

Str\_bonus = 0,

Dex\_bonus = 2,

Con\_bonus = 0,

Inte\_bonus = 0,

Wis\_bonus = 0,

Cha\_bonus = 1,

Rase\_bonus = "Тёмное зрение, Кошачье проворство, Кошачьи когти, Кошачьи способности, Языки: общий и любой на выбор"

}

;

}

}

**Приложение Т: Tiefling. cs**

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Rases

{

public static class Tiefling

{

public static Rase tiefling { get; set; } = new Rase()

{

Id = 6,

Name = "Тифлинг",

Appearance = "Рогатые люди с красными оттенками кожи",

Str\_bonus = 0,

Dex\_bonus = 0,

Con\_bonus = 0,

Inte\_bonus = 1,

Wis\_bonus = 0,

Cha\_bonus = 2,

Rase\_bonus = "Тёмное зрение, Адское сопротивление, Дьявольское наследие, Языки: общий и инфернальный"

}

;

}

}

**Приложение У: Notify. cs**

using System;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.model.Base

{

public class Notify : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged(string prop)

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

}

}

**Приложение Ф: Classes**

Скрипт – Artificer

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Artificer

{

public static dndClass artificer { get; set; } = new dndClass()

{

Id = 7,

Name = "Изобретатель",

StrMain = 0,

DexMain = 0,

ConMain = 2,

InteMain = 2,

WisMain = 0,

ChaMain = 0,

}

;

}

}

Скрипт – Barbarian

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Barbarian

{

public static dndClass barbarian { get; set; } = new dndClass()

{

Id = 2,

Name = "Варвар",

StrMain = 2,

DexMain = 0,

ConMain = 2,

InteMain = 0,

WisMain = 0,

ChaMain = 0,

}

;

}

}

Скрипт – Bard

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Bard

{

public static dndClass bard { get; set; } = new dndClass()

{

Id = 1,

Name = "Бард",

StrMain = 0,

DexMain = 2,

ConMain = 0,

InteMain = 0,

WisMain = 0,

ChaMain = 2,

}

;

}

}

Скрипт – Cleric

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Cleric

{

public static dndClass cleric { get; set; } = new dndClass()

{

Id = 6,

Name = "Жрец",

StrMain = 0,

DexMain = 0,

ConMain = 0,

InteMain = 0,

WisMain = 2,

ChaMain = 2,

}

;

}

}

Скрипт – Druid

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Druid

{

public static dndClass druid { get; set; } = new dndClass()

{

Id = 5,

Name = "Друид",

StrMain = 0,

DexMain = 0,

ConMain = 0,

InteMain = 2,

WisMain = 2,

ChaMain = 0,

}

;

}

}

Скрипт – Fighter

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Fighter

{

public static dndClass fighter { get; set; } = new dndClass()

{

Id = 3,

Name = "Воин",

StrMain = 2,

DexMain = 0,

ConMain = 2,

InteMain = 0,

WisMain = 0,

ChaMain = 0,

}

;

}

}

Скрипт – Monk

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Monk

{

public static dndClass monk { get; set; } = new dndClass()

{

Id = 9,

Name = "Монах",

StrMain = 2,

DexMain = 2,

ConMain = 0,

InteMain = 0,

WisMain = 0,

ChaMain = 0,

}

;

}

}

Скрипт – Paladin

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Paladin

{

public static dndClass paladin { get; set; } = new dndClass()

{

Id = 10,

Name = "Паладин",

StrMain = 0,

DexMain = 0,

ConMain = 0,

InteMain = 0,

WisMain = 2,

ChaMain = 2,

}

;

}

}

Скрипт – Ranger

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Ranger

{

public static dndClass ranger { get; set; } = new dndClass()

{

Id = 12,

Name = "Следопыт",

StrMain = 2,

DexMain = 2,

ConMain = 0,

InteMain = 0,

WisMain = 0,

ChaMain = 0,

}

;

}

}

Скрипт – Sorcerer

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Sorcerer

{

public static dndClass sorcerer { get; set; } = new dndClass()

{

Id = 13,

Name = "Чародей",

StrMain = 0,

DexMain = 0,

ConMain = 2,

InteMain = 0,

WisMain = 0,

ChaMain = 2,

}

;

}

}

Скрипт – Warlock

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Warlock

{

public static dndClass warlock { get; set; } = new dndClass()

{

Id = 8,

Name = "Колдун",

StrMain = 0,

DexMain = 0,

ConMain = 0,

InteMain = 0,

WisMain = 2,

ChaMain = 2,

}

;

}

}

Скрипт – Wizard

using dnd\_char\_create.model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dnd\_char\_create.dataContext.Classes

{

public static class Wizard

{

public static dndClass wizard { get; set; } = new dndClass()

{

Id = 4,

Name = "Волшебник",

StrMain = 0,

DexMain = 0,

ConMain = 0,

InteMain = 2,

WisMain = 2,

ChaMain = 0,

}

;

}

}