Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ПО ПРОГРАММЕ БАКАЛАВРИАТА

09.03.04 Программная инженерия

(код, наименование ОПОП ВО: направление подготовки, направленность (профиль))

«Разработка программно-информационных систем»

Платформа для создания компьютерных изометрических ролевых игр с заранее отрисованным двухмерным фоном и спрайтовыми персонажами

(название темы)

Дипломный проект

(вид ВКР: дипломная работа или дипломный проект)

Автор ВКР		К. Н. Шевченко
	(подпись, дата)	(инициалы, фамилия)
Группа ПО-02б		
Руководитель ВКР		А. А. Чаплыгин
	(подпись, дата)	(инициалы, фамилия)
Нормоконтроль		А. А. Чаплыгин
	(подпись, дата)	(инициалы, фамилия)
ВКР допущена к защите:		
Заведующий кафедрой		А. В. Малышев
	(подпись, дата)	(инициалы, фамилия)

Курск 2024 г.

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

УТВЕРЖДАЮ:

Заведующий кафедрой

_____

(подпись, инициалы, фамилия)

«____» _____ 20____ г.

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ ПО ПРОГРАММЕ БАКАЛАВРИАТА

Студента Шевченко К.Н., шифр 20-06-0139, группа ПО-02б

1. Тема «Платформа для создания компьютерных изометрических ролевых игр с заранее отрисованным двухмерным фоном и спрайтовыми персонажами» утверждена приказом ректора ЮЗГУ от «04» апреля 2024 г. № 1616-с.

2. Срок предоставления работы к защите «11» июня 2024 г.

3. Исходные данные для создания программной системы:

3.1. Перечень решаемых задач:

1) Анализ существующих ролевых игр.
2) Разработка концептуальной модели ролевых игр.
3) Проектирование программной системы для создания ролевых игр.
4) Реализация программной системы для создания ролевых игр.
5) Тестирование разработанной системы.

3.2. Входные данные и требуемые результаты для программы:

1) Входными данными для программной системы являются: данные конфигураций, ПО, критериев качества SLA, ИТ-услуг, информация о языке Python, спрайтовые изображения.

2) Выходными данными для программной системы являются: приложение для разработки компьютерных ролевых игр.

4. Содержание работы (по разделам):

4.1. Введение.

4.1. Анализ предметной области.

4.2. Техническое задание: основание для разработки, назначение разработки, требования к программной системе, требования к оформлению документации.

4.3. Технический проект: общие сведения о программной системе, проект данных программной системы, проектирование архитектуры программной системы, проектирование пользовательского интерфейса программной системы.

4.4. Рабочий проект: спецификация компонентов и классов программной системы, тестирование программной системы, сборка компонентов программной системы.

4.5. Заключение.

4.6. Список использованных источников.

5. Перечень графического материала:

Лист 1. Сведения о ВКРБ.

Лист 2. Цель и задачи разработки.

Лист 3. Концептуальная модель приложения.

Лист 4. Диаграмма классов.

Лист 5. Модель работы сценариев.

Лист 6. Модульное тестирование платформы.

Лист 7. Заключение.

Руководитель ВКР		А. А. Чаплыгин
	(подпись, дата)	(инициалы, фамилия)
Задание принял к исполнению		К. Н. Шевченко
	(подпись, дата)	(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 86 страницам. Работа содержит 28 иллюстраций, 17 таблиц, 12 библиографических источников и 7 листов графического материала. Количество приложений – 2. Графический материал представлен в приложении А. Фрагменты исходного кода представлены в приложении Б.

Перечень ключевых слов: платформа, система, игра, РПГ, Python, сценарии, скрипты, многопоточность, изображения, информатизация, автоматизация, информационные технологии, спрайт, программное обеспечение, классы, обработка клика мыши, подсистема, компонент, модуль, сущность, информационный блок, метод, разработчик, геймдизайнер, пользователь.

Объектом разработки является платформа для создания компьютерных изометрических ролевых игр с заранее отрисованным двумерным фоном и спрайтовыми персонажами.

Целью выпускной квалификационной работы является популяризация рпг игр.

В процессе создания приложения были выделены основные сущности путем создания информационных блоков, использованы классы и методы модулей, обеспечивающие работу с сущностями предметной области, а также корректную работу приложения для разработки рпг-игр, разработаны разделы, содержащие информацию о рпг-играх, игровых платформах для создания игр, графике, языке программирования Python, используемых библиотеках tkinter, treading.

ABSTRACT

The volume of work is 86 pages. The work contains 28 illustrations, 17 tables, 12 bibliographic sources and 7 sheets of graphic material. The number of applications is 2. The graphic material is presented in annex A. The layout of the site, including the connection of components, is presented in annex B.

List of keywords: platform, system, game, RPG, Python, scenarios, scripts, multithreading, images, information, automation, information technology, sprite, software, classes, mouse click processing, subsystem, component, module, entity, information block , method, developer, game designer, user.

The object of development is a platform for creating computer isometric role-playing games with pre-rendered two-dimensional backgrounds and sprite characters.

The purpose of the final qualifying work is to popularize RPG games.

In the process of creating the application, the main entities were identified by creating information blocks, classes and methods of modules were used to ensure work with entities of the subject area, as well as the correct operation of the application for developing RPG games, sections were developed containing information about RPG games, gaming platforms for game creation, graphics, Python programming language, tkinter, treading libraries used.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИС – информационная система.

ИТ – информационные технологии.

КТС – комплекс технических средств.

ПО – программное обеспечение.

РП – рабочий проект.

ТЗ – техническое задание.

ТП – технический проект.

РПГ – ролевая пользовательская игра.

ВВЕДЕНИЕ

С развитием цифровых технологий и увеличением вычислительной мощности персональных компьютеров, появилась возможность создания сложных и многофункциональных программных продуктов, в том числе и для развлекательной индустрии. Одним из таких направлений является разработка компьютерных ролевых игр (RPG), которые погружают пользователя в виртуальные миры с заранее отрисованными фонами и спрайтами. Эти элементы игры не только создают уникальную атмосферу и мир, но и являются ключевыми компонентами в структуре игрового процесса.

Как и аддитивные технологии, которые кардинально изменили подход к проектированию и производству, платформы для создания RPG представляют собой инновационный инструмент, который позволяет разработчикам с минимальными затратами времени и ресурсов создавать захватывающие игры. Это стало возможным благодаря использованию готовых ассетов, таких как фоны и спрайты, а также благодаря гибким инструментам для их интеграции и анимации.

Таким образом, платформы для создания RPG игр с заранее отрисованным фоном и спрайтами являются частью более широкого тренда цифровизации и автоматизации, который охватывает многие отрасли, включая развлекательную индустрию. Они позволяют разработчикам сосредоточиться на творческом процессе, минимизируя технические аспекты реализации проекта.

Цель настоящей работы – разработка приложения для разработки компьютерных ролевых игр с заранее отрисованными спрайтами и фоном. Для достижения поставленной цели необходимо решить следующие задачи:

– провести анализ предметной области;

– разработать концептуальную модель приложения;

– спроектировать приложение;

– реализовать приложение средствами языка программирования python.

Структура и объем работы. Отчет состоит из введения, 4 разделов основной части, заключения, списка использованных источников, 2 приложений. Текст выпускной квалификационной работы равен 10 страницам.

Во введении сформулирована цель работы, поставлены задачи разработки, описана структура работы, приведено краткое содержание каждого из разделов.

В первом разделе на стадии описания технической характеристики предметной области приводится сбор информации о деятельности компании, для которой осуществляется разработка сайта.

Во втором разделе на стадии технического задания приводятся требования к разрабатываемому приложению.

В третьем разделе на стадии технического проектирования представлены проектные решения для приложения.

В четвертом разделе приводится список классов и их методов, использованных при разработке сайта, производится тестирование разработанного приложения.

В заключении излагаются основные результаты работы, полученные в ходе разработки.

В приложении А представлен графический материал. В приложении Б представлены фрагменты исходного кода.

1 Анализ предметной области

1.1 История первых игр что стали прародителями RPG-жанра

Разговор о самых первых компьютерных ролевых играх требует двух важных оговорок. В середине 70-х компьютеры еще не были персональными и представляли собой огромные машины, занимавшие порой отдельные помещения, и были оборудованы подключенными в единую систему терминалами. Доступ к ним был у немногих избранных, а единственными из них, кому могла прийти в голову делать для этих компьютеров игры, были студенты технических университетов. Соответственно, ни у одной из созданных этими первопроходцами игр не было никаких шансов на коммерческий релиз.

Сейчас уже сложно установить, какой была первая видеоигра, которую можно было бы отнести к жанру RPG. Многие из них безнадежно сгинули в пучине истории. Например, теоретически претендующая на почетное первенство игра под названием m199h, созданная в 1974-м в Университете Иллинойса почти сразу после выхода первой редакции DnD, была попросту удалена кем-то из преподавателей — компьютеры ведь созданы для обучения, а не для игрушек. Зато вот появившаяся примерно тогда же The Dungeon сохранилась до наших дней. Она также известна как pedit5 — это название исполняемого файла, который юный разработчик Расти Рутерфорд замаскировал под учебный. От удаления смекалочка игру не спасла, но исходный код уцелел, и сыграть в нее можно даже сегодня.

Привыкшие к современным RPG геймеры от увиденного могут испытать культурный шок. Но даже по меркам середины 70-х эти игры казались примитивными. Причем не в сравнении с другими жанрами видеоигр, а в сравнении со все теми же настолками. Если за игровым столом в компании друзей подробности приключения и игровой мир в деталях рисовало воображение игроков, и лишь оно ограничивало пределы игры, то скудная презентация этих ранних видеоигровых экспериментов и близко не давала такого опыта. Тем более речи не шло ни о каком серьезном отыгрыше роли и глубоком нарративе, к которым нас приучили вышедшие многим позже шедевры

жанра. Чтобы называться компьютерной RPG, в те годы игре достаточно было обладать какой-никакой системой прокачки, да давать возможность отыгрывать в бою воина или мага.

В том, что касается сюжета, диалогов и повествования в целом для жанра гораздо больше сделала игра, которую даже в 1976 году никому не пришло бы в голову назвать ролевой — Colossal Cave Adventure. По сути это прабабушка всех текстоцентричных игр: от RPG с объемными диалогами до интерактивных сериалов и даже визуальных новелл. Она могла бы быть стандартной адвенчурой про исследователя пещер, пытающегося найти сокровища в лабиринте, каких было немало. Вот только ее создатель Уилл Кроутер решил полностью отказаться от графики, забив экран монитора детальным описанием окружения, и тем самым не только вновь отдал бремя проработки деталей на откуп фантазии игрока, но и легитимировал текстовый нарратив для всех будущих разработчиков.

1.2 Первые популярные RPG-игры

Akalabeth стала основой всех будущих dungeon crawler — игр с упором на исследование подземелий. Сохранив геймплейную основу ранних RPG — зачистку подземелий, классы и прокачку — она впервые объединила вид от первого лица при прохождении уровня и вид сверху при перемещении по миру. В игре присутствовала и механика провизии, за объемом которой нужно было постоянно следить, и проработанная система заклинаний, применение которых вызывало подчас совершенно неожиданные последствия. Фантазии, смелости и амбиций автору было не занимать. Последнее особенно подчеркивает существование в мире игры персонажа по имени Lord British, от которого игрок и получал все задания. Разработка Гэрриота оказалась настолько нетривиальной, что ей заинтересовался крупный издатель. Смешные по сегодняшним меркам продажи в 30 тысяч копий обрекли Akalabeth на сиквел, а ее автора — на профессию игрового разработчика. Так началась многолетняя история одной величайших игровых серий прошлого — Ultima.

Благодаря развитию технологий, большему бюджету и поддержке издателя Гэрриот сумел в кратчайшие сроки значительно улучшить техническую составляющую игры — вышедшая спустя год Ultima обзавелась тайловой графикой, а для управления персонажем больше не нужно было вводить текстовые команды — достаточно было нажатия на кнопки со стрелочками. Но больше всего аудиторию поразил небывалый размах приключения: мало того, что игровой мир стал куда более объемным, а благодаря современной графике выглядел реальнее, чем когда-либо, так еще и повествование охватывало аж три временные эпохи.

Между тем игры Гэрриота обрели достойную конкуренцию в лице не менее значительной для жанра серии Wizardry. Созданная в 1981 году командой Sir-Tech Software в лице Эндрю Гринберга и Роберта Вудхеда, она не хватала звезд с неба ни в плане графики, ни в плане сюжета, зато геймплейно была глубже и проработаннее любой другой CRPG. Если Гэрриот ориентировался на посиделки в DnD и старался перенести на экран волшебный антураж, рисуемый воображением, то разработчики Wizardry ставили себе цель вывести на новый уровень игры с мейнфреймов, в которые залипали в студенческие годы. Для них на первом месте была механика. Весь игровой мир изображался в маленьком квадратике в углу экрана, большую же его часть заполняла важная для прохождения информация — очки здоровья и классы бойцов, список заклинаний, данные о противнике. При создании каждого из шести играбельных персонажей можно было не только выбрать расу, класс и распределить очки характеристик, но и прописать героям мировоззрение, влияющее на дальнейшую прокачку. Таким образом Wizardry еще и стала первой партийной RPG в истории, так что корни Baldur’s Gate, Icewind Dale и даже Divinity: Original Sin растут именно отсюда. Боевую систему сдобрили обширной системой магии, среди которой было место как прямо атакующим заклинаниям, так и различным дебаффам. А еще разработка Sir-Tech была беспощадно сложной: подобно Rogue в случае смерти партии игроку ничего не оставалось, кроме как начать с нуля

1.3 Япония и её JRPG

В 1986 году отобранная по конкурсу компанией Enix команда молодых и амбициозных японских технарей во главе с Юдзи Хории разработала и выпустила первую в истории JRPG под названием Dragon Quest. Именно эта игра сформировала основные правила поджанра на десятилетия вперед: вид сверху, более-менее свободное исследование огромного мира, состоящего из квадратных тайлов, случайные встречи, пошаговый бой, отдельное окно для сражений с изображением противника и списком возможных действий, а также большой акцент на линейное повествование с неизменными тропами: древнее зло, магические артефакты, спасение принцессы... Здесь же любители RPG впервые столкнулись с около-анимешной эстетикой, за которую отвечал специально привлеченный в качестве художника известный мангака Акира Торияма.

На старте 1987 года компания Square, обреченная в будущем стать второй (или первой?) половинкой Enix, выпустила на японский рынок игру, с которой началась история длиною в жизнь. И если Dragon Quest изобрела жанр, то синонимом JRPG стало имя Final Fantasy. И ведь, казалось бы, на первый взгляд игра Хиронобу Сакагучи не сильно отличалась от своей предшественницы из Enix. С геймплейной точки зрения ключевым изменением стала система классов — игрок мог по желанию сделать любого из четверки героев воином, вором, монахом или магом одной из школ. Но главное, чем брала Final Fantasy, — небывалой амбициозностью во всем. В ее мире присутствовали и элементы стимпанка, и научная фантастика, и петля времени, которую бравым героям необходимо было разомкнуть… Постановка также была яркой и необычной для своего времени: например, представляющую игру заставку и титры игрок видел лишь после выполнения первого квеста — прием, активно взятый на вооружение современными разработчиками.

1.4 Популярные RPG студии SSI

Главным же поставщиком RPG на грани десятилетий стала компания SSI. В 1988 году ее президент Джоэл Биллингс ввязался в крупнейшую авантюру своей жизни: в жесточайшей конкуренции за огромные деньги выкупил официальную лицензию на создание игр по обновленной редакции легендарной настолки Advanced Dungeons and Dragons. В следующие пять лет SSI выпустила целых 12 компьютерных ролевых игр, вошедших в историю под общим именем Gold Box. Откровенно говоря, большая их часть не изобретала велосипеда. Они лишь довели знакомую жанровую схему предшественниц до совершенства и сопроводили ее достаточным количеством оригинального контента — врагов, квестов, оружия, элементов окружения. Из важных деталей стоит отметить возможность избежать сражения с врагом путем дипломатии (для этого необходимо было выбрать правильный тон разговора) и функцию быстрого перемещения с помощью раскинувшейся по игровому миру сети телепортов. Лицензия DnD распространялась и на использование различных сеттингов настолки, поэтому местом действия игр могли стать как «Забытые Королевства», так и вселенная «Драконьего Копья». Первоисточник даровал разработчикам не только готовую механику, но и проработанную мифологию. Такой мощный фундамент позволял стабильно выпускать новинки раз в несколько месяцев. Наладив потоковое производство, SSI превратила создание ролевых игр в индустрию. Вскоре каталог компании пополнили и игры сторонних студий, разработанные по драгоценной лицензии, в числе которых была, например, популярная трилогия Eye of the Beholder от Westwood Studios.

Два релиза из коллекции Gold Box заслуживают отдельного внимания. Во-первых, это выпущенная в 1993 году Forgotten Realms: Unlimited Adventures, которая технически являлась не игрой, а набором инструментов для создания собственных приключений, основанных на ADnD. Некоторые безумные традиционалисты от мира ролевых игр до сих пор пользуются этой программой для разработки нового контента, а в 90-е она устроила настоя-

щий переворот в фанатских кругах и предопределила формирование сообщества моддеров. Не менее важным событием стал выход в 1991-м Neverwinter Nights. Сейчас эту игру затмил другой релиз под таким же названием, случившийся уже в следующем веке, но в истории индустрии она останется навсегда.

Она не выделялась на фоне других игр SSI ни внешним видом, ни ролевой системой, но один важный нюанс делал ее особенной: Neverwinter Nights стала первой полноценной графической MMORPG. Ее серверы вмещали до 50 игроков одновременно, общая же аудитория исчислялась сотнями тысяч. Фанаты объединялись в гильдии, вступали в виртуальные конфликты и проводили в онлайне массовые сходки. Интерес к Neverwinter Nights не увядал вплоть до ее закрытия в 1997 году, а ее влияние на дальнейшее развитие индустрии неоценимо.

2 Техническое задание

2.1 Основание для разработки

Основанием для разработки является задание на выпускную квалификационную работу бакалавра «Платформа для создания компьютерных изометрических ролевых игрс заранее отрисованным двухмерным фоном и спрайтовыми персонажами».

2.2 Цель и назначение разработки

Основной задачей выпускной квалификационной работы является разработка платформы для создания компьютерных изометрических ролевых игр с заранее отрисованным двумерным фоном и спрайтовыми персонажами для продвижения популярности рпг-игр».

Данный программный продукт предназначен для демонстрации практических навыков, полученных в течение обучения.

Задачами данной разработки являются:

1. Анализ существующих ролевых игр.
2. Разработка концептуальной модели ролевых игр.
3. Проектирование программной системы для создания ролевых игр.
4. Реализация программной системы для создания ролевых игр.
5. Тестирование разработанной системы.

2.3 Требования пользователя к платформе

платформа должна включать в себя:

– создание зон;
– создание объектов;
– создание персонажей;
– добавление объектов в зону;
– удаление объектов из зоны;
– реализацию сценариев.

Композиция шаблона игры, созданной на движке, представлена на рисунке 2.1.

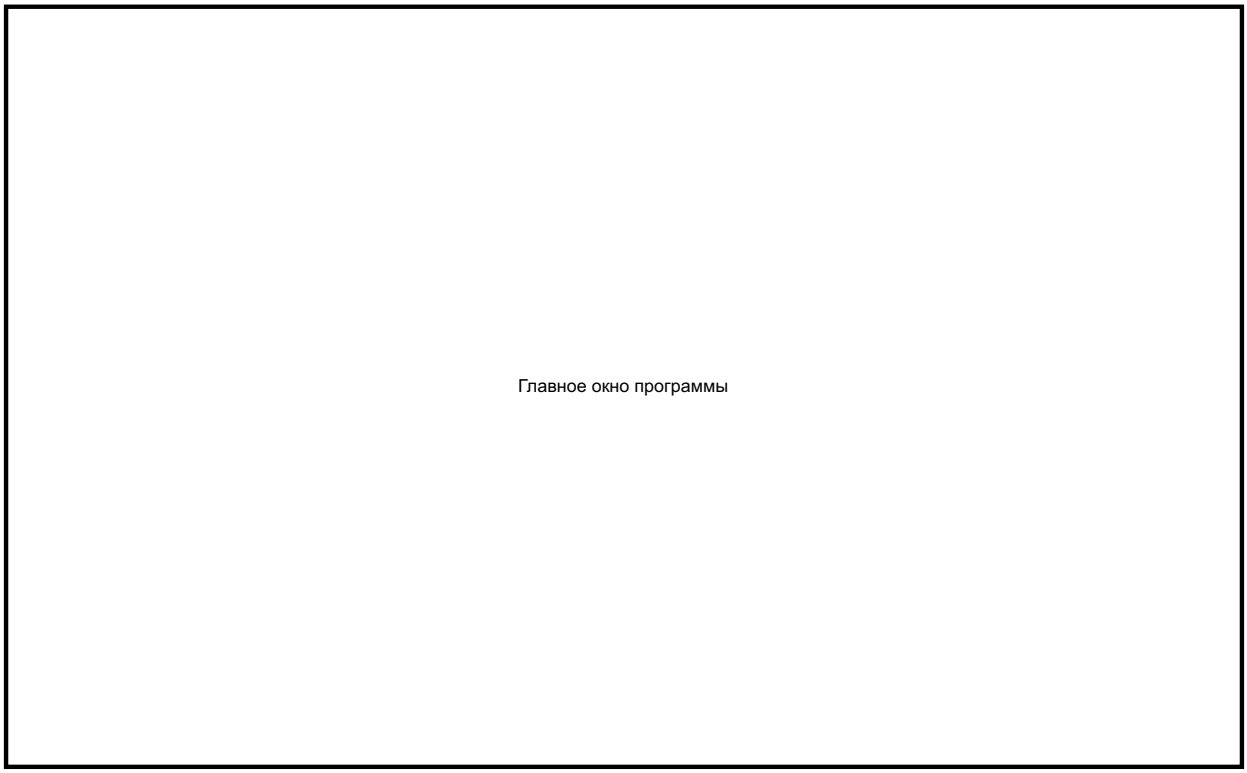

Рисунок 2.1 – Композиция шаблона интерфейса игры

2.4 Описание платформы для создания RPG игр

Клиент создает модуль содержащий методы модуля RPGGame, например bggame. В этом модуле мы создаем мир игры, с помощью new_actor. Мы можем вызывать их много раз с разными параметрами, или загрузить параметры для этих функций из файла. После чего у нас есть персонажи и предметы. Мир также состоит из зон (Area). Каждая зона включает в себя графику, персонажи и предметы и сценарии взаимодействия. Исключение составляет команда PC, которая может перемещаться из зоны в зону (это мы программируем у клиента). Команду мы тоже определяем стартовую и впоследствии можем менять (add_actor_to_team, remove_actor_from_team). Каждому персонажу и объекту может соответствовать пользовательский сценарий (он активируется при нажатии мышкой на объект). Сценарий может включать диалог, взятие предмета, добавление персонажа в команду, квест и т.д. Зона тоже мо-

жет содержать сценарий, который запускается когда команда попадает в зону. Клиентский класс (BGGame) также содержит глобальные переменные, определяющие ситуации в игре (например квесты). Локальные переменные могут быть в зоне.

Как программируются зоны. Если нужны локальные переменные (состояние локальных событий), то тогда нужно создавать класс своей зоны как наследник от Area. Или же просто использовать класс Area. Добавляем зону в игру new_area(name, area). Переключаем зону - set_area(name). Глобальные сценарии находятся в классе игры (BGGame), мы подключаем их как : Area.set_enter_script(script) В зону мы добавляем персонажей и предметы как add_object(x,y, obj) - z не нужно, так как слой можно определить по у координате. В конкретную зону мы добавляем сценарий для взаимодействия как: Game.game.start_script(script, name) Как происходит переход команды между зонами. В зоне определяем объект дверь, по клику мыши она может открываться и закрываться (меняется состояние объекта). Назначаем сценарий walk_script(script), который срабатывает когда кто-то из команды пересекает объект. В этом сценарии мы меняем зону на нужную (set_area), и устанавливаем команду в нужную позицию (set_team). В другой зоне делается аналогично, только переход и позиция будут другими. Сценарии - это потоки которые запускаются параллельно (метод RPGGame.start_script(script)). Сценарий может быть остановлен (stop_script(name)). Таким образом, мир будет интерактивным. Как связано окно и графика с игрой. В окне мы делаем таймер, который вызывает метод update нашей игры (BGGame). Этот метод выполняет все действия объектов в игре за 1 кадр времени. Также в таймере вызываем Graphics.update(), который обновляет графику игры. Все объекты (Actor, Item) должны иметь состояния (как минимум одно). Каждое состояние связано с спрайтом (или анимацией). То есть переключение состояния меняет графику объекта.

А вообще сценарии и глобальные переменные могут быть без классов, а просто в модулях, так проще, чтобы к ним был доступ из всех комнат. Тогда

и функции движка должны быть доступны везде (то есть во всех сценариях). Например делаем модуль руины (ruins) представлен на рисунке 2.2:

```
import random
from math import sqrt
import time
from rpg.area import *
from rpg.sprite import *
from rpg.rectangle import *
from rpg.game import Game
from rpg.portal import Portal

class Ruins(Area):
  def \_\_init\_\_(self):
    super().\_\_init\_\_()
    self.add\_sprite(Sprite('images/fon3.png'), 590, 400, 0)
    self.add\_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').
       image.width(), height=Sprite('images/fon3.png').image.height()))
    from grunt import Grunt
    self.grunt = Grunt(0,0,0)
    from footman import Footman
    self.footman = Footman(0,0,0)
    self.add\_object(self.footman, 120, 120, 1)
    self.add\_object(self.grunt, 500, 185, 1)
    p = Portal(400, 400, 200, 200, 'Village', 480, 100)
    self.add\_object(p, p.pos\_x, p.pos\_y, 100)
    Game.game.start\_script(self.ai, "ai", self.grunt)
    Game.game.start\_script(self.walk\_two, "footman", 50, 50)
  def walk(self, step\_x, step\_y, actor):
    if actor.hp <= 0:\\
      Game.game.stop\_script("grunt")
      new\_x = 200
      new\_y = 200
      actor.is\_attack = False
    direction = random.choice(["up", "down", "left", "right"])
    if direction == "up":
      new\_y -= step\_y
      new\_x = step\_x
    elif direction == "down":
      new\_y += step\_y
      new\_x = step\_x
    elif direction == "left":
      new\_y = step\_y
    new\_x -= step\_x
    elif direction == "right":
    new\_y = step\_y
    new\_x += step\_x

    actor.search\_position(new\_x, new\_y)

    time.sleep(2)
```

Рисунок 2.2 – Пример создания игровой зоны Ruins

продолжение на рисунке 2.3:

```
модуль bggame:
from ruins import *
import time
import random

class BaldursGame(Game):
  def \_\_init\_\_(self, canvas, window, **params):
  '''
  Класс конкретной игры для демонстрации

  :param canvas: класс графической системы
  :param window: окно на которое будет выводится игра
  '''
  super().\_\_init\_\_(canvas, window, **params)
  from mage import Mage
  self.add\_pc\_to\_team(Mage(0, 0, 0))
  self.new\_area('Ruins', Ruins())
  self.set\_area('Ruins')
  self.set\_team(500, 300, 100)
  self.timer()
```

Рисунок 2.3 – Пример создания игровой зоны Ruins

2.4.1 Пример клиентского кода игры

2.4.1.1 Создание классов персонажей/предметов

Клиент создает модуль содержащий методы модуля RPGGame, например BaldursGateGame. В этом модуле клиент создает мир игры, с помощью new_actor. Пример представлен на рисунке 2.4:

```
    модуль bggame:
    from ruins import *
    import time
    import random

    class BaldursGame(Game):
      def \_\_init\_\_(self, canvas, window, **params):
      '''
      Класс конкретной игры для демонстрации

      :param canvas: класс графической системы
      :param window: окно на которое будет выводится игра
      '''
      super().\_\_init\_\_(canvas, window, **params)
      from mage import Mage
      self.add\_pc\_to\_team(Mage(0, 0, 0))
      self.new\_area('Ruins', Ruins())
      self.set\_area('Ruins')
      self.set\_team(500, 300, 100)
      self.timer()
```

Рисунок 2.4 – Пример создания персонажа Mage

2.4.1.2 Задание правил атаки

Пользователь создаёт класс ADnDActor, наследник от класса Actor в своём модуле bggame, в нём он прописывает свои правила по которым происходит атака. То есть Actor.attack(self, actor), где actor - кого атакуют. Пример представлен на рисунке 2.5:

```
модуль adnd\_actor:
from math import sqrt
from rpg.actor import Actor
from rpg.animation import Animation
import rpg.game
import time

class Adnd\_actor(Actor):

  ATTACK\_RANGE = 50

  def \_\_init\_\_(self, x, y, z, **params):
    super().\_\_init\_\_(x, y, z, **params)
    self.on\_click = self.click

  def click(self):
    pc = rpg.game.Game.game.team\_of\_pc[0]
    if pc == self:
      return
    dx = pc.pos\_x - self.pos\_x
    dy = pc.pos\_y - self.pos\_y
    dist = sqrt(dx * dx + dy * dy)
    if dist <= self.ATTACK\_RANGE:
      pc.is\_attack = True
      pc.attack(self)
      time.sleep(0.125)
    if self.hp <=0:
      pc.is\_attack = False

  def attack(self, actor):
    actor.hp -= self.damage
  def update(self):
    super().update()
    if self.hp <= 0:
      self.stop\_move()
      self.set\_state('death')
```

Рисунок 2.5 – Пример задания правил атаки персонажа

2.4.1.3 Создание зон, заполнение их персонажами/объектами

Мир также состоит из зон (Area). Каждая зона включает в себя графику, персонажи и предметы и сценарии взаимодействия. Исключение составляет команда PC, которая может перемещаться из зоны в зону (это мы программируем у клиента). Как программируются зоны. Если нужны локальные переменные (состояние локальных событий), то тогда нужно создавать класс своей зоны как наследник от Area. Или же просто использовать класс Area. До-

бавляем зону в игру new_area(name, area). Переключаем зону - set_area(name). Так же требуется задать область движения, её проще сделать как совокупность прямоугольников, за которые персонажи не могут выйти. Эти прямоугольники должны касаться друг друга, но не пересекаться. Тогда алгоритм проверки выхода несложный: выход за пределы области только тогда, когда прямоугольник персонажа пересек сторону (одну или две) одного из прямоугольников области, эта сторона не является касательной. Пример представлен на рисунке 2.6:

```
    from rpg.game import Game
    from rpg.portal import Portal

    class Ruins(Area):
      def \_\_init\_\_(self):
        super().\_\_init\_\_()
        self.add\_sprite(Sprite('images/fon3.png'), 590, 400, 0)
        self.add\_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').
           image.width(),\\ height=Sprite('images/fon3.png').image.height()))
        from grunt import Grunt
        self.grunt = Grunt(0,0,0)
        from footman import Footman
        self.footman = Footman(0,0,0)
        self.add\_object(self.footman, 120, 120, 1)
        self.add\_object(self.grunt, 500, 185, 1)
        p = Portal(400, 400, 200, 200, 'Village', 480, 100)
        self.add\_object(p, p.pos\_x, p.pos\_y, 100)
        Game.game.start\_script(self.ai, "ai", self.grunt)
        Game.game.start\_script(self.walk\_two, "footman", 50, 50)
      def walk(self, step\_x, step\_y, actor):
        if actor.hp <= 0:
          Game.game.stop\_script("grunt")
        new\_x = 200
        new\_y = 200
        actor.is\_attack = False
        direction = random.choice(["up", "down", "left", "right"])
        if direction == "up":
          new\_y -= step\_y
          new\_x = step\_x
        elif direction == "down":
          new\_y += step\_y
          new\_x = step\_x
        elif direction == "left":
          new\_y = step\_y
          new\_x -= step\_x
        elif direction == "right":
          new\_y = step\_y
          new\_x += step\_x
        actor.search\_position(new\_x, new\_y)
        time.sleep(2)
```

Рисунок 2.6 – Пример создания зоны Ruins

2.4.1.4 Пример сценариев: переход между зонами

Глобальные сценарии находятся в классе игры (BGGame), мы подключаем их как : Area.set_enter_script(script)

Как происходит переход команды между зонами. В зоне определяем объект портал, по клику мыши когда персонаж заходит внутрь портала срабатывает

self.actor_in(self, actor). При создании портала, мы указываем кудаи в какую зону разместить команду персонажей. Пример представлен на рисунке 2.7:

```
    from rpg.object import Object
    from rpg.game import Game
    from rpg.rectangle import Rectangle
    class Portal(Object):
      def \_\_init\_\_(self, x, y, width, height, area, team\_x, team\_y):
        self.states = None
        self.sprite = None
        self.category = 'portal'
        super().\_\_init\_\_(x, y, 0)
        self.rectangle = Rectangle(x, y, width, height)
        self.area = area
        self.team\_x = team\_x
        self.team\_y = team\_y
        self.visible = False

      def actor\_in(self, actor):
        if actor.category == "pc":
          Game.game.set\_area(self.area)
          Game.game.set\_team(self.team\_x, self.team\_y, 100)
        actor.stop\_move()
    модуль ruins
    import random
    from math import sqrt
    import time
    from rpg.area import *
    from rpg.sprite import *
    from rpg.rectangle import *
    from rpg.game import Game
    from rpg.portal import Portal
    class Ruins(Area):
      def \_\_init\_\_(self):
        super().\_\_init\_\_()
        self.add\_sprite(Sprite('images/fon3.png'), 590, 400, 0)
        self.add\_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').
           image.width(), height=Sprite('images/fon3.png').image.height()))
        from grunt import Grunt
        self.grunt = Grunt(0,0,0)
        from footman import Footman
        self.footman = Footman(0,0,0)
        self.add\_object(self.footman, 120, 120, 1)
        self.add\_object(self.grunt, 500, 185, 1)
        p = Portal(400, 400, 200, 200, 'Village', 480, 100)
```

Рисунок 2.7 – Пример перехода между зонами

2.4.1.5 Как будет идти бой

Бой будет совершаться с помощью сценариев. У класса Adnd_actor есть метод attack(self, actor), который уменьшает текущее количество здоровья у actor. В модуле game существуют методы start_script(script, name), stop_script(name). С помощь сценариев возможно запускать параллельные потоки. В конкретную зону будет добавляться сценарий ’ai’, в который передаётся конкретный персонаж. В этом сценарии указывается поведение противника, Что он должен сближаться с персонажем игрока, и когда расстояние до атаки будет достаточным, чтобы её совершить, будет вызван метод actor.attack. Для того, чтобы пользователь мог атаковать персонажа, у каждого экземпляра класса adnd_actor есть метод click(self), который вызывает проверку условия, если персонаж близко к персонажу игрока, хранящемуся в rpg.game.Game.team_of_pc, то вызвать у pc=rpg.game.Game.team_of_pc[0], attack(self)/ Пример представлен на рисунке 2.8:

```
модуль adnd\_actor:
from math import sqrt
from rpg.actor import Actor
from rpg.animation import Animation
import rpg.game
import time
class Adnd\_actor(Actor):
  ATTACK\_RANGE = 50
  def \_\_init\_\_(self, x, y, z, **params):
    super().\_\_init\_\_(x, y, z, **params)
    self.on\_click = self.click
  def click(self):
    pc = rpg.game.Game.game.team\_of\_pc[0]
    if pc == self:
      return
    dx = pc.pos\_x - self.pos\_x
    dy = pc.pos\_y - self.pos\_y
    dist = sqrt(dx * dx + dy * dy)
    if dist <= self.ATTACK\_RANGE:
      pc.is\_attack = True
        pc.attack(self)
        time.sleep(0.125)
    if self.hp <=0:
      pc.is\_attack = False
  def attack(self, actor):
    actor.hp -= self.damage
  def update(self):
    super().update()
    if self.hp <= 0:
      self.stop\_move()
      self.set\_state('death')
```

Рисунок 2.8 – Пример задания боя между персонажами

Продолжение на рисунке 2.9:

```
    модуль ruins
    import random
    from math import sqrt
    import time
    from rpg.area import *
    from rpg.sprite import *
    from rpg.rectangle import *
    from rpg.game import Game
    from rpg.portal import Portal
    class Ruins(Area):
      def \_\_init\_\_(self):
        super().\_\_init\_\_()
        self.add\_sprite(Sprite('images/fon3.png'), 590, 400, 0)
        self.add\_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').
           image.width(), height=Sprite('images/fon3.png').image.height()))
        from grunt import Grunt
        self.grunt = Grunt(0,0,0)
        from footman import Footman
        self.footman = Footman(0,0,0)
        self.add\_object(self.footman, 120, 120, 1)
        self.add\_object(self.grunt, 500, 185, 1)
        p = Portal(400, 400, 200, 200, 'Village', 480, 100)
        self.add\_object(p, p.pos\_x, p.pos\_y, 100)
        Game.game.start\_script(self.ai, "ai", self.grunt)
        Game.game.start\_script(self.walk\_two, "footman", 50, 50)
      def ai(self, actor):
        if actor.hp <= 0:
          Game.game.stop\_script("ai")
        import rpg.game
        pc = rpg.game.Game.game.team\_of\_pc[0]
        new\_x = pc.pos\_x
        new\_y = pc.pos\_y

        actor.search\_position(new\_x, new\_y)
        dx = pc.pos\_x - actor.pos\_x
        dy = pc.pos\_y - actor.pos\_y
        dist = sqrt(dx * dx + dy * dy)
        if dist <= actor.ATTACK\_RANGE:
          actor.is\_attack = True
          actor.attack(pc)
          time.sleep(1)
          if pc.hp <=0:
            actor.update()
            Game.game.stop\_script("ai")
        else:
          actor.is\_attack = False
        time.sleep(2)
```

Рисунок 2.9 – Пример задания боя между персонажами

2.5 Особенности Dungeons and Dragons

– Dungeons and Dragons (DnD) - это настольная ролевая игра, в которой игроки сотрудничают вместе, чтобы создать историю в фантастическом мире. В DnD один игрок выступает в роли Мастера игры (Мастера подземелий), который рассказывает и контролирует мир, а остальные игроки играют за своих персонажей, которых они создают и развивают.

Основные элементы ролевой системы DnD включают:

– 1. Классы и расы: Классы представляют различные роли и специализации персонажей, такие как воин, маг, жрец. Каждый класс имеет свои уникальные способности и навыки.

∗ Особенности воина: воин специализируется на ближнем бою, может использовать все виды оружия, может носить все доспехи и щиты, не способен накладывать заклинания, его кость здоровья 10-гранный кубик (D10).

∗ Особенности мага: маг специализируется на дальнем бою, может использовать только боевые посохи и короткие мечи, не может носить доспехи , способен накладывать заклинания, наносящие большое количество урона, его кость здоровья 6-гранный кубик (D6).

∗ Особенности жреца: жрец специализируется на ближнем бою, может использовать простое оружия, может носить лёгкие,средние доспехи и щиты, способен накладывать заклинания, исцеляющие его, его кость здоровья 8-гранный кубик (D8).

– Расы определяют происхождение персонажа и дают особые характеристики и способности. Примеры рас включают эльфов, дварфов, людей.

∗ Особенности человека: человек на старте получает +1 ко всем характеристикам, его размер средний.

∗ Особенности эльфа: эльф получает +2 к ловкости и +1 к мудрости, его размер средний, у эльфа есть тёмное зрение в радиусе 30 футов.

∗ Особенности дварфа: дварф получает +2 к силе и +2 к телосложению, его размер маленький, у дварфа есть тёмное зрение в радиусе 30 футов.

– 2. Характеристики:

∗ Характеристики определяют физические и умственные способности персонажа, такие как сила, ловкость, телосложение, интеллект, мудрость, харизма. Они влияют на способности и успех персонажа в различных ситуациях.

∗ Сила - характеристика влияющая на броски атак рукопашным оружием, а так же на проверки навыков: атлетика.

∗ Ловкость - характеристика влияющая на броски атак совершаемых стрелковым оружием, на класс доспеха персонажа, а так же на проверки навыков: акробатика, ловкость рук, скрытность.

∗ Телосложение - характеристика влияющая на колличество здоровья персонажа.

∗ Интеллект - характеристика влияющая на броски атак совершённых заклинаниями волшебника, а так же на проверки навыков: магия, история, природа, расследование, религия.

∗ Мудрость - характеристика влияющая на броски атак совершённых заклинаниями жреца, а так же на проверки навыков: восприятие, выживание, проницательность, уход за животными, медицина.

∗ Харизма - характеристика влияющая на общение с не игровыми персонажами, а так же на проверки навыков: выступление, убеждение, обман, запугивание.

– 3. Навыки:

∗ Навыки представляют специализации персонажа в определенных областях, таких как взлом замков, обращение с оружием, магия и т.д. Навыки могут быть использованы для выполнения действий и решения задач

– 4. Броски костей:

∗ Игра DnD использует различные виды игровых костей для случайной генерации результатов. Например, для определения успеха атаки или проверки навыка игрок может бросить 20-гранный кубик (D20) и добавить соответствующие модификаторы.

– 5. Приключения и задания:

∗ Мастер игры создает историю, включающую задания и приключения, которые игроки выполняют. Задания могут включать исследование подземелий, сражение с монстрами, решение головоломок и взаимодействие с неигровыми персонажами.

– 6: Прогрессия и опыт:

∗ Персонажи получают опыт за выполнение заданий и сражение с врагами. Зарабатывая опыт, персонажи повышают уровень, получают новые способности и становятся сильнее.

– 7. Магия:

∗ DnD имеет разветвленную систему магии, позволяющую персонажам использовать заклинания различных уровней и школ. Магические заклинания могут влиять на бой, лечение, обнаружение и другие аспекты игры.

2.6 Пример игры

Создаётся рабочее окно tkinter, на нём пользователь видит текущую зону, из зоны current_area, так же все объекты, находящиеся в ней, и всех персонажей из команды персонажей, текущей игры. Пользователь может взаимодействовать с окном с помощью мыши. Левым кликом мыши по окну вызывает метод mouse_click у текущей игры. который вызывает проверку находится ли в координатах, в которых был совершён клик, какой-либо персонаж или объект, и если есть, то вызвать метод on_click. Если персонажа в данных координатах нет, то вызвать у всех персонажей с полем category == ”pc”метод search_position(x,y), который указывает координаты движения, которые должны прийти персонажи. Так же работают все сценарии, конкретной зоны. они работают до тех пор, пока не будет вызвано условие останавливающее, конкретный сценарий. На рисунке 2.10 сформированы следующие действия пользователя и их последствия.

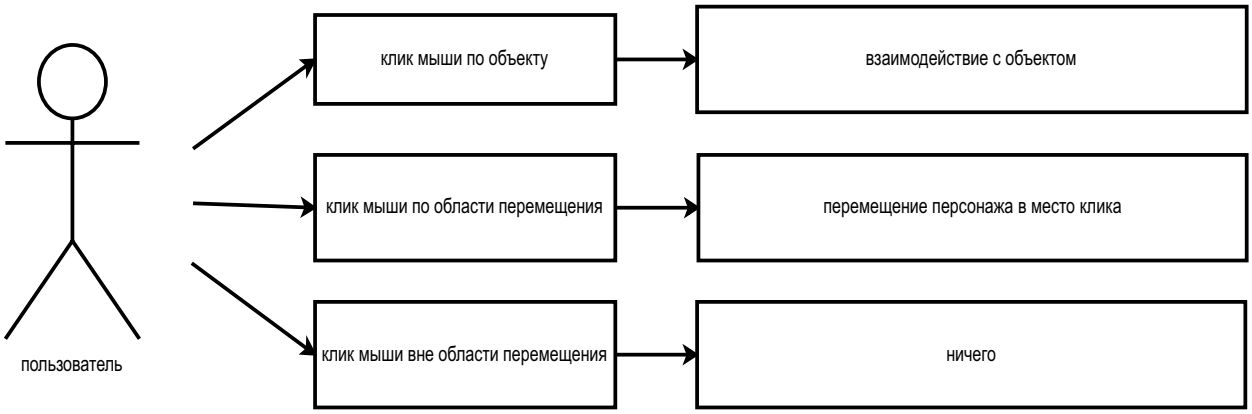

Рисунок 2.10 – Шаблон интерфейса игры

2.7 Требования к оформлению документации

Разработка программной документации и программного изделия должна производиться согласно ГОСТ 19.102-77 и ГОСТ 34.601-90. Единая система программной документации. Программная документация должна включать в себя:

– техническое задание;

– технический проект;

– рабочий проект.

3 Технический проект

3.1 Общая характеристика организации решения задачи

Необходимо спроектировать и разработать приложение, который должен способствовать популяризации ролевых игр.

Приложение представляет собой набор взаимосвязанных различных окон, которые сгруппированы по разделам, содержащие текстовую, графическую информацию. Приложение располагается на компьютере.

3.2 Обоснование выбора технологии проектирования

На сегодняшний день информационный рынок, поставляющий программные решения в выбранной сфере, предлагает множество продуктов, позволяющих достигнуть поставленной цели – разработки приложения.

3.2.1 Описание используемых технологий и языков программирования

В процессе разработки приложения используются программные средства и языки программирования. Каждое программное средство и каждый язык программирования применяется для круга задач, при решении которых они необходимы.

3.2.2 Язык программирования Python

Python – высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпрети-

руемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как C или C++.

Python — это язык программирования, который широко используется в интернет-приложениях, разработке программного обеспечения, науке о данных и машинном обучении (ML). Разработчики используют Python, потому что он эффективен, прост в изучении и работает на разных платформах. Программы на языке Python можно скачать бесплатно, они совместимы со всеми типами систем и повышают скорость разработки.

3.2.3 Использование библиотеки Tkinter на Python

3.2.3.1 Введение

Библиотека Tkinter - это стандартная библиотека Python для создания графического пользовательского интерфейса (GUI). Она обладает широкими возможностями для создания разнообразных приложений с использованием различных виджетов, таких как кнопки, поля ввода, метки и многое другое.

3.2.3.2 Возможности Tkinter

Вот некоторые из основных возможностей, предоставляемых библиотекой Tkinter:

1. Создание различных виджетов: кнопки, метки, поля ввода, списки и многое другое.
2. Управление компоновкой виджетов с использованием менеджеров компоновки (например, grid, pack, place).
3. Обработка событий, таких как щелчок мыши, нажатие клавиш и другие.
4. Возможность создания различных диалоговых окон, таких как окна предупреждений, информационные окна и окна запроса.

5. Поддержка многопоточности для обновления интерфейса из различных потоков выполнения.

3.2.3.3 Соединение движка и окон tkinter

Модуль graphics содержит в себе библиотеку tkinter . Класс Graphics внутри модуля является наследником tk.Canvas. Этот класс взаимодействует с окном root = tk.TK() в программном модуле пользователя. Модуль sprite тоже взаимодействует с tkinter. Изображение для спрайта берётся с помощью метода tk.PhotoImage(file=name) Пример представлен на рисунке 3.1:

```
    модуль sprite

    import tkinter as tk
    class Sprite:
      def \_\_init\_\_(self, image):
        self.image = tk.PhotoImage(file=image)
        self.tag = None
        self.x = 0
        self.y = 0
        self.z = 0
        def set\_tag(self, tag):
        self.tag = tag

      def set\_z(self, z):
        self.z = z

      def get\_tag(self):
        return self.tag
      def set\_coords(self, new\_x, new\_y):
        if self.tag:
        self.x = new\_x
        self.y = new\_y
      def update(self):
        pass
```

Рисунок 3.1 – Пример использования tkinter в программе

Продолжение на рисунке 3.2:

```
    import tkinter as tk
    class Graphics(tk.Canvas):
      canvas = None
      def \_\_init\_\_(self, master, **kwargs):
        super().\_\_init\_\_(master, **kwargs)
        self.sprites = []
        Graphics.canvas = self
      def add\_sprite(self, sprite, x, y, z, **kwargs):
        tag = self.create\_image(x, y, image=sprite.image, anchor='center',
           **kwargs)
        sprite.set\_tag(tag)
        sprite.set\_z(z)
        sprite.x = x
        sprite.y = y
        self.sprites.append(sprite)
        self.sprites.sort(key=lambda sprite: sprite.z)
      def update(self):
        for sprite in self.sprites:
          sprite.update()
          self.tag\_raise(sprite.get\_tag())
          self.coords(sprite.get\_tag(), sprite.x, sprite.y)
          self.itemconfig(sprite.get\_tag(), image=sprite.image)
          def change\_sprite(self, sprite, new\_sprite):
          old\_sprite\_pos = None
        for i, s in enumerate(self.sprites):
          if s.get\_tag() == sprite.get\_tag():
            old\_sprite\_pos = i
            break
        if old\_sprite\_pos is not None:
          old\_tag = sprite.get\_tag()

        self.sprites[old\_sprite\_pos] = new\_sprite
        new\_sprite.set\_tag(old\_tag)\
        new\_sprite.set\_tag(old\_tag)
        new\_sprite.set\_z(sprite.z)
        self.tag\_raise(old\_tag)
        self.coords(old\_tag, sprite.x, sprite.y)
        self.itemconfig(old\_tag, image=new\_sprite.image)

      def delete\_sprite(self, sprite):
        self.delete(sprite.get\_tag()
        self.sprites.remove(sprite)
      def clear\_all(self):
        for sprite in self.sprites:
          self.delete(sprite.get\_tag())
          self.sprites.clear()
```

Рисунок 3.2 – Пример использования tkinter в программе

Продолжение на рисунке 3.3:

```
    модуль baldursgame '''пользовательский модуль'''
    from ruins import *
    from village import *
    import time
    import random
    class BaldursGame(Game):
    def \_\_init\_\_(self, canvas, window, **params):
    super().\_\_init\_\_(canvas, window, **params)
    from mage import Mage
    self.add\_pc\_to\_team(Mage(0, 0, 0))
    self.new\_area('Ruins', Ruins())
    self.new\_area('Village', Village())
    self.set\_area('Ruins')
    self.set\_team(500, 300, 100)
    self.timer()
    модуль main
    from bggame import *
    root = tk.Tk()
    root.geometry('1500x1500')
    exit\_button = tk.Button(root, text="Exit", fg="red", command=root.
       destroy)
    canvas = Graphics(root, width=1500, height=1500)
    Graphics.canvas = canvas
    BaldursGame(canvas, root)
    canvas.place(height = 1500, width =1500)
    BaldursGame.timer
    root.mainloop()
```

Рисунок 3.3 – Пример использования tkinter в программе

3.2.3.4 Заключение

Библиотека Tkinter предоставляет мощные инструменты для создания графических пользовательских интерфейсов на языке Python. Реализация таймеров на Python может быть достигнута с помощью модулей time или threading, в зависимости от конкретных требований приложения.

3.3 Архитектура платформы для создания ролевых игр

3.3.1 Диаграмма компонентов классов

Диаграмма компонентов описывает особенности физического представления разрабатываемой системы. Она позволяет определить архитектуру системы, установив зависимости между программными компонентами, в роли которых может выступать как исходный, так и исполняемый код. Ос-

новными графическими элементами диаграммы компонентов являются компоненты, интерфейсы, а также зависимости между ними. На рисунке **??** изображена диаграмма компонентов для проектируемой системы. Она включает в себя основной класс платформы игры Game и производные от него классы, класс Object с наследниками и их параметрами (полями и методами).

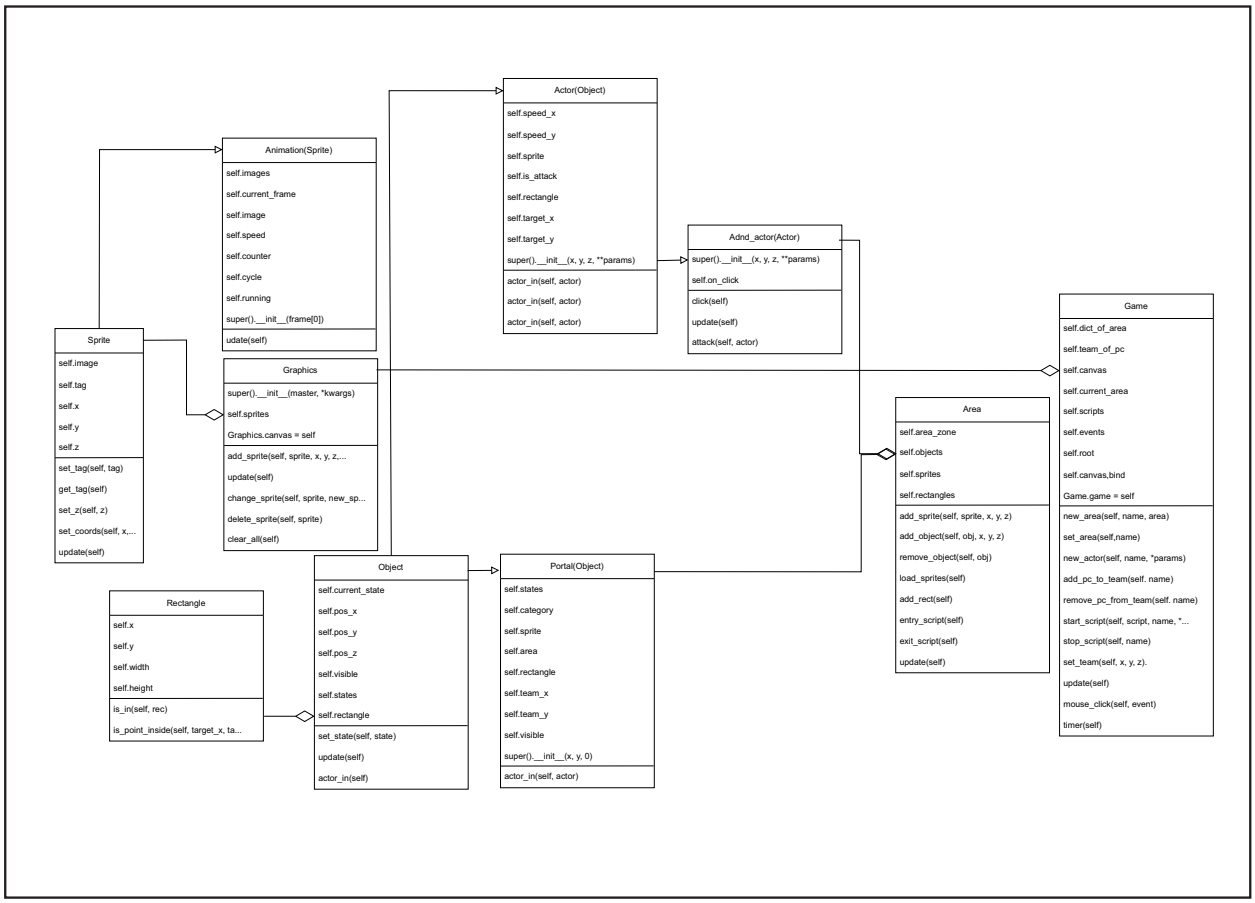

Рисунок 3.4 – Диаграмма компонентов

3.3.1.1 Описание классов

Graphics - класс, управляющий спрайтами. Содержит в себе:

– self.sprites - список спрайтов;

– add_sprite(self, sprite x, y, z, image) - добавляет в список спрайт, сохраняет координаты;

– change_sprite(self, sprite new_sprite) - меняет местами спрайты в списке.

– delete_sprite(self, sprite) - удаляет спрайт из списка.

– clear_all(self) - очищает список спрайтов.

– update(self) - добавляет все спрайты из списка на форму.

Sprite - класс, хранящий в себе изображение игровых объектов image.

– self.x - координата x.

– self.y - координата y.

– self.z - координата z.

– self.tag - уникальный номер спрайта.

– self.image - изображение.

– set_tag(self, tag) - устанавливает tag спрайту.

– get_tag(self) - возвращает tag спрайта.

– set_z(self, z) - устанавливает z координату.

– set_coords(self, new_x, new_y) - устанавливает новые координаты.

– update(self) - ничего не делает.

Animation - класс, хранящий в себе список изображений игровых объектов image. Потомок класса Sprite.

– self.current_frame - текущий кадр.

– self.images - Загрузка всех кадров анимации.

– self.image - Установка начального изображения.

– self.speed - скорость анимации.

– self.counter - счётчик кадров.

– self.cycle - проверка на то что должна ли быть анимация циклично или нет.

– self.running - проверка проигрывается ли сейчас анимация.

– update(self) - обновляет кадр в анимации.

Rectangle - абстрактный класс прямоугольника.

– self.pos_x - координата x.

– self.pos_y - координата y.

– self.width - ширина.

– self.height - высота.

– is_in(self, rect) - функция проверки нахождения одного прямоугольника в другом.

– is_point_inside(self, target_x, target_y) - функция проверки точки в пределах прямоугольника.

Object - класс, от которого наследуются классы Adnd_Actor, Actor, Portal.

– self.pos_x - координата x.

– self.pos_y - координата y.

– self.pos_z - координата z.

– self.current_state - текущее состояние.

– self.visible - видимость портала.

– self.on_click - функция клика по объекту.

– self.rectangle - прямоугольник объекта.

– set_state(self, state_name) -устанавливает состояние.

– actor_in(self, actor) - ничего не делает.

– update(self) - ничего не делает.

Portal - класс объекта для перехода между зонами. Потомок класса Object.

– self.states - состояние портала.

– self.sprite - спрайт портала.

– self.category - категория.

– self.rectangle = - прямоугольник портала.

– self.area - зона в которую ведёт портал.

– self.team_x - координата x в которую нужно разместить команду.

– self.team_y - координата y в которую нужно разместить команду.

– self.visible - видимость портала.

– actor_in(actor) - события, которые произойдут, когда персонаж окажется внутри прямоугольника портала.

Actor - класс персонажа, содержащий внутри себя основные поля и методы для перемещения по рабочему окну.

– self.sprite - спрайт персонажа.

– self.speed_x - значение скорости x.

– self.speed_y - значение скорости y.

– self.target_x - координата x в которую должен прийти персонаж.

– self.target_y - координата y в которую должен прийти персонаж.

– self.rectangle - прямоугольник персонажа.

– self.is_attack - атакует ли сейчас персонаж.

– update(self) - функция обновления координат и состояния персонажа.

– search_position(self, new_x, new_y) - поиск координат в которые нужно двигаться персонажу.

– stop_move(self) - остановка движения персонажа.

Adnd_Actor - класс персонажа, содержащий методы связанные с взаимодействием с другими персонажами. Является наследником Actor.

– self.on_click событие при клике на персонажа

– update(self) - функция обновления координат и состояния персонажа.

– click(self) - функция вызывается при клике по персонажу.

– attack(self, actor) - функция атаки персонажа по другому персонажу.

Area - зона, в которой находятся персонажи и объекты. Содержит следующие поля и методы:

– self.area_zone - параметр определяющий особенности конкретной зоны.

– self.objects - список, хранящий в себе множество объектов.

– self.sprites - список фоновых спрайтов.

– self.rectangles - прямоугольник зоны.

– add_sprite(self, sprite, x, y, z) - функция добавляет спрайт в зону.

– add_object(self, obj, x, y, z) - функция добавляет объект в зону.

– remove_object(self, obj) - функция удаляет объект из зоны.

– load_sprites(self) - функция загружает все спрайты зоны.

– add_rect(self, rec) - функция добавляет прямоугольник в зону.

– entry_script(self) - функция запускается, когда команда входит в зону.

– exit_script(self) - функция запускается, когда команда выходит из зоны.

– update(self) - функция изменяет и проверяет изменение всех объектов в зоне.

Game - абстрактный класс, управляющий игрой. Имеет следующие поля и методы:

– self.rpg_dict_of_area - словарь, хранящий в себе множество экземпляров класса Area.

– self.team_of_pc - список, хранящий в себе имена экземпляров класса Actor с параметром category = ”pc”.

– self.canvas - графика.

– self.root - окно для графики.

– self.current_area - параметр хранящий, текущую зону.

– self.scripts - словарь для хранения запущенных сценариев.

– self.events - словарь для хранения запущенных event‘ов сценариев.

– self.canvas.bind(«Button-1> self.mouse_left._click) - обработка клика мыши по рабочему окну.

– new_area(self, name, area) - функция добавляет новую зону в список.

– set_area(self, name) - функция устанавливает текущую зону, загружает графику зоны.

– new_actor(self, name, **params) - функция создаёт класс, потомок от Actor и создаёт поле из параметров, и установление их в начальные значения.

– add_pc_to_team(self, pc) - функция добавляет персонажа в команду.

– remove_pc_from_team(self, pc) - функция удаляет персонажа из команды.

– start_script(self, script_function, script_name, *args) - функция запускает сценарий в отдельном потоке с возможностью остановки и передачи аргументов.

– stop_script(self, script_name) - функция останавливает сценарий по имени.

– set_team(self, x, y, z) - функция устанавливает координаты персонажей команды.

– update(self) - функция вызывается в таймере для обновления всех переменных в текущей зоне.

– mouse_left_click(self, event) - функция обрабатывает клик мыши.

– timer(self) - функция должна вызывать метод update постоянно.

3.3.2 Реализация графической подсистемы

Графическая подсистема основана на библиотеке tkinter, которая используется для создания графического интерфейса пользователя. В контексте платформы, tkinter используется для отображения и управления спрайтами — графическими объектами, которые представляют персонажей, предметы и другие элементы игры.

3.3.2.1 Система спрайтов

Она реализована через класс Graphics, который расширяет tk.Canvas. Этот класс управляет отображением спрайтов на холсте, их сортировкой по z-координате (что позволяет создать эффект глубины), а также обновлением их позиций. Спрайты могут быть добавлены, перемещены и удалены с холста. Вот пример метода, который добавляет спрайт на холст: Пример на рисунке 3.5:

```
1     def add\_sprite(self, sprite, x, y, z, **kwargs):
2       tag = self.create\_image(x, y, image=sprite.image, anchor='center', **
           kwargs)
3       sprite.set\_tag(tag)
4       sprite.set\_z(z)
5       self.sprites.append(sprite)
6       self.sprites.sort(key=lambda sprite: sprite.z)
```

Рисунок 3.5 – пример добавления спрайта на холст

3.3.3 Реализация зон

Зоны в программе представляют собой различные игровые области. Каждая зона реализована через класс Area, который содержит спрайты и объекты, принадлежащие этой зоне. Класс Area является ключевым элементом в структуре игры, так как он определяет отдельные игровые зоны, которые игрок может исследовать. Каждая зона представляет собой уникальный участок игрового мира со своим набором характеристик и поведения.

Спрайты и объекты: В основе каждой зоны лежат спрайты и объекты. Спрайты — это графические элементы, такие как персонажи, враги или декорации, которые игрок видит на экране. Объекты могут быть как видимыми, так и невидимыми элементами, которые взаимодействуют с игроком или окружением, например, триггеры событий или коллекционные предметы. Скрипты входа и выхода: entry_script и exit_script — это скрипты, которые активируются при входе игрока в зону и при выходе из неё соответственно. Эти скрипты могут использоваться для запуска кат-сцен, начала битв, обновления заданий или любых других событий, которые должны произойти при изменении зоны. Метод update: Метод update вызывается каждый игровой цикл и отвечает за обновление состояния всех спрайтов и объектов в зоне. Это может включать анимацию спрайтов, проверку столкновений, выполнение игровой логики и многое другое. Если в зоне заданы скрипты входа или выхода, метод update также будет их вызывать в соответствующий момент. В целом, класс Area обеспечивает структурированное и модульное построение игрового мира, позволяя разработчикам создавать сложные и интерактивные зоны, которые делают игровой процесс более разнообразным и захватывающим. Это также упрощает управление ресурсами и оптимизацию, так как каждая зона может быть загружена и выгружена независимо от остальных.

3.3.4 Реализация объектов и персонажей

Объекты и персонажи являются ключевыми элементами игрового мира. Они реализованы через классы Object и Adnd_Actor соответственно. Object может представлять любой игровой объект, который может взаимодействовать с игроком или окружением. Adnd_Actor расширяет Object и добавляет дополнительные свойства и методы, специфичные для персонажей, такие как движение, атака и взаимодействие с другими персонажами. Класс Object служит основой для всех элементов в игре, которые могут взаимодействовать с игроком или средой. Это могут быть предметы, такие как ключи, оружие, сундуки с сокровищами, или даже более абстрактные понятия, такие как ловушки или интерактивные точки. Класс Adnd_Actor расширяет Object,

добавляя свойства и методы, которые специфичны для персонажей игры. Это включает в себя движение, атаку, взаимодействие с другими персонажами и возможность реагировать на изменения в игровой среде. Эти классы позволяют разработчикам игр создавать разнообразные и динамичные объекты и персонажи, каждый из которых обладает уникальными характеристиками и способностями. Объекты могут быть простыми и служить одной цели, например, быть предметом для сбора, или же могут быть сложными, выполняя ряд функций в игре. Персонажи, с другой стороны, являются более сложными сущностями, которые могут взаимодействовать с игроком и другими элементами игрового мира на более глубоком уровне, выполняя различные действия, такие как бой, торговля или выполнение заданий.

3.3.5 Реализация сценариев

Сценарии в игре используются для создания интерактивных и динамических событий. Они могут быть реализованы как функции, которые запускаются в отдельных потоках, позволяя игре продолжать обрабатывать другие задачи в фоновом режиме. Класс Game содержит методы start_script и stop_script для управления этими сценариями. Thread – это отдельный поток выполнения. Это означает, что в программе могут работать две и более подпрограммы одновременно. Но разные потоки на самом деле не работают одновременно: это просто кажется. Соблазнительно думать, что в программе работают два (или более) разных процессора, каждый из которых выполняет независимую задачу одновременно. Это почти правильно, но это то, что обеспечивает многопроцессорность (multiprocessing). Запуск потоков (threading) похожа на эту идею, но ваши программы работает только на одном процессоре. Различные задачи, внутри потоков выполняются на одном ядре, а операционная система управляет, когда программа работает с каким потоком. Поскольку потоки выполняется на одном процессоре, они хорошо подходят для ускорения некоторых задач, но не для всех. Задачи, которые требуют значительных вычислений ЦП и тратят мало времени на ожидание внешних со-

бытий, очевидно используя многопоточность не будут выполняться быстрее, вместо этого следует использовать многопроцессорность (multiprocessing).

Архитектура программы при использования многопоточности также может помочь в достижении более чистой архитектуры проекта. Большинство примеров, которые мы рассмотрим в этой статье, не обязательно будут работать быстрее используя потоки. Но использование потоков поможет сделать их архитектуру чище и понятнее. Потоки (Thread) Стандартная библиотека Python предоставляет библиотеку threading, которая содержит необходимые классы для работы с потоками. Основной класс в этой библиотеки Thread. Чтобы запустить отдельный поток, нужно создать экземпляр потока Thread и затем запустить его с помощью метода .start(). Пример на рисунке 3.6:

```
    import threading
    import time
    def thread_function(name):
      logging.info("Thread %s: starting", name)
      time.sleep(2)
      logging.info("Thread %s: finishing", name)
    if __name__ == "__main__":
      format = "%(asctime)s: %(message)s"
      logging.basicConfig(format=format, level=logging.INFO,
      datefmt="%H:%M:%S")
      logging.info("Main    : before creating thread")
      x = threading.Thread(target=thread_function, args=(1,))
      logging.info("Main    : before running thread")
      x.start()
      logging.info("Main    : wait for the thread to finish")
      # x.join()
      logging.info("Main    : all done")
    x = threading.Thread(target=thread_function, args=(1,))
    x.start()
```

Рисунок 3.6 – Пример метода tread.start

Когда создается поток Thread, ему передается функцию и список, содержащий аргументы этой функции. В примере указывается Thread, чтобы он запустил функцию thread_function() и передаем ему 1 в качестве аргумента.

Сама по себе функция thread_function() мало что делает. Она просто выводит некоторые сообщения с промежутком time.sleep() между ними. Демоны потоков. В информатике daemon (демон) – это процесс, который работает в фоновом режиме.

Python потоки имеет особое значение для демонов. Демон потока (или как еще его можно назвать демонический поток) будет остановлен сразу после выхода из программы. Один из способов думать об этих определениях – считать демон потока как потоком, который работает в фоновом режиме, не беспокоясь о его завершении.

Если в программе запущены потоки, которые не являются демонами, то программа будет ожидать завершения этих потоков, прежде чем сможет завершится. Тем не менее, потоки, которые являются демонами, при закрытие программы просто убиваются, в каком бы они состояние ни находились. join() Чтобы указать одному потоку дождаться завершения другого потока, вам нужно вызывать .join(). Если вызвать .join(), этот оператор будет ждать, пока не завершится любой вид потока.

3.3.6 Вычисление пересечения прямоугольников

Для определения столкновений и взаимодействий между объектами используется класс Rectangle. Он содержит методы, такие как is_in, который проверяет, находится ли один прямоугольник внутри другого, и is_point_inside, который проверяет, находится ли точка внутри прямоугольника. Вот пример метода is_point_inside на рисунке 3.7:

```
1    def is\_point\_inside(self, target\_x, target\_y):
2    return (self.x <= target\_x <= self.x + self.width) and
3    (self.y <= target\_y <= self.y + self.height)
4    Этот метод использует логические операторы для проверки, находится ли
        точка (target\_x, target\_y) в пределах прямоугольника, определенного
        координатами (x, y) и размерами (width, height).
```

Рисунок 3.7 – Пример метода is_point_inside класса Rectangle

4 Рабочий проект

4.1 Классы, используемые при разработке приложения

4.1.1 Класс Sprite

Класс Sprite относится к rpg и используется для отображения изображения на холсте.

Описание методов класса Sprite представлено в таблице 4.1.

Таблица 4.1 – Методы класса Sprite

Название метода	Описание метода
1	2
set_tag(self, tag)	Устанавливает тег спрайта, параметр tag: тег спрайта.
set_z(self, z)	Устанавливает z-координату спрайта, параметр z: координата z.
get_tag(self)	Возвращает тег спрайта.
set_coords(self, new_x, new_y)	Обновляет координаты спрайта, параметр new_x: координата x, параметр new_y: координата y.
update(self)	Обновляет анимацию спрайта.

4.1.2 Класс Animation

Класс Animation относится к rpg и предназначен для отображения набора изображений, как анимации на холсте.

Описание методов класса Animation представлено в таблице 4.3.

Таблица 4.3 – Методы класса Animation

Название метода	Описание метода
1	2
update(self)	Меняет текущее изображение в списке изображений.

4.1.3 Класс Graphics

Класс Graphics относится к rpg и предназначен для работы графической системы в платформе.

Описание методов класса Graphics представлено в таблице 4.5.

Таблица 4.5 – Методы класса Graphics

Название метода	Описание метода
1	2
add_sprite(self, sprite, x, y, z, **kwargs)	Добавляет спрайт на Canvas, параметр sprite: спрайт, параметр x: координата x, параметр y: координата y, параметр z: координата z, параметр kwargs: параметры относящиеся к конкретному изображению в tkinter.
update(self)	Перерисовывает все спрайты.
change_sprite(self, sprite, new_sprite)	Меняет спрайт на новый в Canvas, параметр sprite: экземпляр спрайта, параметр new_sprite: новый спрайт.
delete_sprite(self, sprite)	Удаляет спрайт с Canvas, параметр sprite: экземпляр спрайта.
clear_all(self)	Удаляет все спрайты с Canvas.

4.1.4 Класс Rectangle

Класс Rectangle относится к rpg и предназначен для создания прямоугольника, который нужен для объектов и зон.

Описание методов класса Rectangle представлено в таблице 4.7.

Таблица 4.7 – Методы класса Rectangle

Название метода	Описание метода
1	2
is_in(self, rect)	Проверяет, входит ли прямоугольник self в прямоугольник rect, параметр rect: прямоугольник.
is_point_inside(self, target_x, target_y)	Проверяет, входит ли точка (x, y) в данный прямоугольник, параметр target_x: точка x, параметр target_y: точка y.

4.1.5 Класс Object

Класс Object относится к rpg и предназначен для инициализации объектов.

Описание методов класса Object представлено в таблице 4.9.

Таблица 4.9 – Методы класса Object

Название метода	Описание метода
1	2
set_state(self, state_name)	Меняет текущее состояние объекта, параметр state_name: новое состояние.
actor_in(self, actor)	Вызывается когда персонаж входит внутрь объекта, параметр actor: персонаж входящий в объект.
update(self)	Обновляетинформацию об объекте.

4.1.6 Класс Portal

Класс Portal относится к rpg и предназначен для объектов используемых для переходов между зонами.

Описание методов класса Portal представлено в таблице 4.11.

Таблица 4.11 – Методы класса Portal

Название метода	Описание метода
1	2
actor_in(self, actor)	Проверяет находится ли персонаж внутри портала, параметр actor: проверяемый персонаж.

4.1.7 Класс Actor

Класс Actor относится к rpg и предназначен для инициализации персонажей, а так для хранения базовых механик передвижения.

Описание методов класса Actor представлено в таблице 4.13.

Таблица 4.13 – Методы класса Actor

Название метода	Описание метода
1	2
update(self)	Изменяет координаты и состояние персонажа.
search_position(self, new_x, new_y)	Изменяет направление движения у персонажа, параметр new_x: координата новой точки x, параметр new_y: координата новой точки y.
stop_move(self)	Останавливает движение персонажа.

4.1.8 Класс Adnd_actor

Класс Adnd_actor относится к rpg и предназначен для создания персонажа игры с механиками взаимодействия с другими персонажами.

Описание методов класса Adnd_actor представлено в таблице 4.15.

Таблица 4.15 – Методы класса Adnd_actor

Название метода	Описание метода
1	2
click(self)	Вызывается при клике на персонажа.
attack(self, actor)	Совершает атаку по actor, параметр actor: персонаж, которого атакуют.
update(self)	Обновляет состояние персонажа.

4.1.9 Класс Area

Класс Area относится к rpg и предназначен для создания зоны, хранящей в себе определённое количество объектов, изображений.

Описание методов класса Area представлено в таблице 4.17.

Таблица 4.17 – Методы класса Area

Название метода	Описание метода
1	2
add_sprite(self, sprite, x, y, z)	Добавляет спрайт в зону, параметр sprite: экземпляр спрайта, параметр x: коодрината x, параметр y: коодрината y, параметр z: коодрината z.
add_object(self, obj, x, y, z)	Добавляет объект в зону, параметр obj: объект, параметр x: коодрината x, параметр y: коодрината y, параметр z: коодрината z.
remove_object(self, obj)	Удаляет объект из зоны.
load_sprites(self)	Загружает все спрайты зоны.
add_rect(self, rec)	Добавляет прямоугольник в зону, параметр rec: прямоугольник.
entry_script(self)	Запускается, когда команда входит в зону.
exit_script(self)	Запускается, когда команда выходит из зоны.

Продолжение таблицы 4.17

1	2
update(self)	Изменяет и проверяет изменение всех объектов в зоне.

4.1.10 Класс Game

Класс Game относится к rpg и предназначен для создания персонажа игры с механиками взаимодействия с другими персонажами.

Описание методов класса Game представлено в таблице 4.19.

Таблица 4.19 – Методы класса Game

Название метода	Описание метода
1	2
new_area(self, name, area)	Добавляет новую зону в список, параметр name: имя зоны, параметр area: класс area.
set_area(self, name)	Устанавливает текущую зону, загружает графику зоны, параметр name: имя зоны.
new_actor(self, name, **params)	Создаёт класс, потомок от Actor и создаёт поле из параметров, и установление их в начальные значения, параметр name: название нового класса, параметр params: поля нового класса.
add_pc_to_team(self, pc)	Добавляет персонажа в команду, параметр pc: персонаж, которого нужно добавить в команду.
remove_pc_from_team(self, pc)	Удаляет персонажа из команды, параметр pc: персонаж, которого нужно удалить.

Продолжение таблицы 4.19

1	2
start_script(self, script_function, script_name, *args)	Запускает сценарий в отдельном потоке с возможностью остановки и передачи аргументов, параметр script_function: Функция, содержащая код сценария, параметр script_name: Имя сценария, параметр args: Дополнительные аргументы, которые нужно передать в сценарий..
stop_script(self, script_name)	Останавливает сценарий по имени, параметр script_name: имя сценария, который нужно остановить.
set_team(self, x, y, z)	Устанавливает координаты персонажей команды, параметр x: координата x, параметр y: координата y, параметр z: координата z.
update(self)	Вызывается в таймере для обновления всех переменных в текущей зоне.
mouse_left_click(self, event)	Обрабатывает клик мыши, параметр event: клим мыши.
timer(self)	Таймер дожен вызывать метод update постоянно.

4.1.11 Модуль Grunt

Модуль Grunt относится к примеру работы платформы и предназначен для создания персонажа орка.

Описание методов модуля Grunt представлено в таблице 4.21.

Таблица 4.21 – Методы модуля Grunt

Название метода	Описание метода
1	2
new_actor('Grunt', category='enemy', damage=10, hp=10, strange=5, wizdom=10, name='Grunt', states={})	Создаёт персонажа орка.

4.1.12 Модуль Mage

Модуль Mage относится к примеру работы платформы и предназначен для создания персонажа мага.

Описание методов модуля Mage представлено в таблице 4.23.

Таблица 4.23 – Методы модуля Mage

Название метода	Описание метода
1	2
new_actor('Mage', category='pc', damage=10, hp=10, strange=5, wizdom=10, name='Mage', states={})	Создаёт персонажа мага.

4.1.13 Модуль Footman

Модуль Footman относится к примеру работы платформы и предназначен для создания персонажа рыцаря.

Описание методов модуля Grunt представлено в таблице 4.25.

Таблица 4.25 – Методы модуля Footman

Название метода	Описание метода
1	2
new_actor('Footman', category='npc', damage=10, hp=10, strange=5, wizdom=10, name='Footman', states={})	Создаёт персонажа рыцаря.

4.1.14 Модуль Village

Модуль Village относится к примеру работы платформы и предназначен для создания зоны деревня.

Описание методов модуля Village представлено в таблице 4.27.

Таблица 4.27 – Методы модуля Village

Название метода	Описание метода
1	2
__init__(self)	Инициализирует зону Village.

4.1.15 Модуль Ruins

Модуль Ruins относится к примеру работы платформы и предназначен для создания зоны деревня.

Описание методов модуля Ruins представлено в таблице 4.29.

Таблица 4.29 – Методы модуля Ruins

Название метода	Описание метода
1	2
__init__(self)	Инициализирует зону Ruins.
walk(self, step_x, step_y, actor)	Сценарий для движения персонажа, параметр step_x: шаг движения x, параметр step_y: шаг движения y.
ai(self, actor)	Сценарий для противников, параметр step_x: размер шага x до персонажа игрока, параметр step_y: размер шага x до персонажа игрока, параметр actor: персонаж противник.

4.1.16 Модуль bggame

Модуль bggame относится к примеру работы платформы и предназначен для создания экземпляра игры.

Описание методов модуля bggame представлено в таблице 4.31.

Таблица 4.31 – Методы модуля bggame

Название метода	Описание метода
1	2
__init__(self, canvas, window, **params)	Инициализирует игру bggame, параметр canvas: класс графической системы, параметр window: окно на которое будет выводится игра.

4.1.17 Модуль main

Модуль main относится к примеру работы платформы и предназначен для создания экземпляра игры.

Описание методов модуля main представлено в таблице 4.33.

Таблица 4.33 – Методы модуля main

Название метода	Описание метода
1	2
root = tk.Tk()	Создаёт окно tkinter.
root.geometry('1500x1500')	Указывает размер окну tkinter.
canvas = Graphics(root, width=1500, height=1500)	Создание экземпляра класса graphics, который будет взаимодействовать с окном.
Graphics.canvas = canvas	Присваивание canvas в статическое поле.
BaldursGame(canvas, root)	Создание новой игры BaldursGame.
canvas.place(height = 1500, width =1500)	Размещение canvas на окне tkinter.
BaldursGame.timer	Вызов метода timer у BaldursGame.
root.mainloop()	Основной цикл обработки событий.

4.2 Модульное тестирование разработанного приложения

Модульный тест для класса Rectangle из модели данных представлен на рисунке 4.1.

```
import unittest
from rpg.rectangle import Rectangle

class TestRectangle(unittest.TestCase):
  def setUp(self):
    # Прямоугольник для использования в тестах
    self.rect = Rectangle(1, 1, 4, 4)

  def test_inside(self):
    '''Тест: прямоугольник внутри другого'''
    rect_outside = Rectangle(0, 0, 6, 6)
    self.assertTrue(self.rect.is_in(rect_outside))

  def test_outside(self):
    '''Тест: прямоугольник снаружи другого'''
    rect_inside = Rectangle(2, 2, 2, 2)
    self.assertFalse(self.rect.is_in(rect_inside))

  def test_apartside(self):
    '''Тест: прямоугольник отдельно от другого'''
    rect_apart = Rectangle(6, 6, 2, 2)
    self.assertFalse(self.rect.is_in(rect_apart))

  def test_touching_left(self):
    '''Тест: прямоугольник касается слева'''
    touching_left = Rectangle(0, 2, 1, 1)
    self.assertFalse(self.rect.is_in(touching_left))

  def test_touching_right(self):
    '''Тест: прямоугольник касается справа'''
    touching_right = Rectangle(5, 2, 1, 1)
    self.assertFalse(self.rect.is_in(touching_right))

  def test_touching_top(self):
    '''Тест: прямоугольник касается сверху'''
    touching_top = Rectangle(2, 5, 1, 1)
    self.assertFalse(self.rect.is_in(touching_top))

  def test_touching_bottom(self):
    '''Тест: прямоугольник касается снизу'''
    touching_bottom = Rectangle(2, 0, 1, 1)
    self.assertFalse(self.rect.is_in(touching_bottom))
```

Рисунок 4.1 – Модульный тест класса Rectangle

Продолжение модульного теста для класса Rectangle из модели данных представлен на рисунке 4.2.

```
    def test_intersect_left(self):
      '''Тест: пересечение прямоугольника слева'''
      intersect_left = Rectangle(0, 2, 3, 2)
      self.assertTrue(self.rect.is_in(intersect_left))

    def test_intersect_right(self):
      '''Тест: пересечение прямоугольника справа'''
      intersect_right = Rectangle(3, 2, 3, 2)
      self.assertTrue(self.rect.is_in(intersect_right))

    def test_intersect_top(self):
      '''Тест: пересечение прямоугольника сверху'''
      intersect_top = Rectangle(2, 3, 2, 3)
      self.assertTrue(self.rect.is_in(intersect_top))

    def test_intersect_bottom(self):
      '''Тест: пересечение прямоугольника снизу'''
      intersect_bottom = Rectangle(2, 0, 2, 3)
      self.assertTrue(self.rect.is_in(intersect_bottom))

    def test_is_point_inside(self):
      # Создайте прямоугольник
      rect = Rectangle(0, 0, 10, 10)

      # Точка внутри прямоугольника
      self.assertTrue(rect.is_point_inside(5, 5))

      # Точка на границе прямоугольника
      self.assertTrue(rect.is_point_inside(0, 0))
      self.assertTrue(rect.is_point_inside(0, 10))
      self.assertTrue(rect.is_point_inside(10, 0))
      self.assertTrue(rect.is_point_inside(10, 10))

      # Точка вне прямоугольника
      self.assertFalse(rect.is_point_inside(-1, -1))
      self.assertFalse(rect.is_point_inside(11, 11))
      self.assertFalse(rect.is_point_inside(5, -5))
      self.assertFalse(rect.is_point_inside(-5, 5))

if __name__ == '__main__':
unittest.main()
```

Рисунок 4.2 – Модульный тест класса Rectangle

4.3 Системное тестирование разработанного приложения

На рисунке 4.3 представлен пример работы программы.

Рисунок 4.3 – Пример работы программы с одним персонажем внутри одной, игровой зоны Village

На рисунке 4.4 представлен пример анимации персонажа.

Рисунок 4.4 – Анимация передвижения персонажа mage

На рисунке 4.5 представлен пример движения персонажа.

Рисунок 4.5 – Передвижение персонажа mage

На рисунке 4.6 представлен пример невозможности выхода за границу зоны.

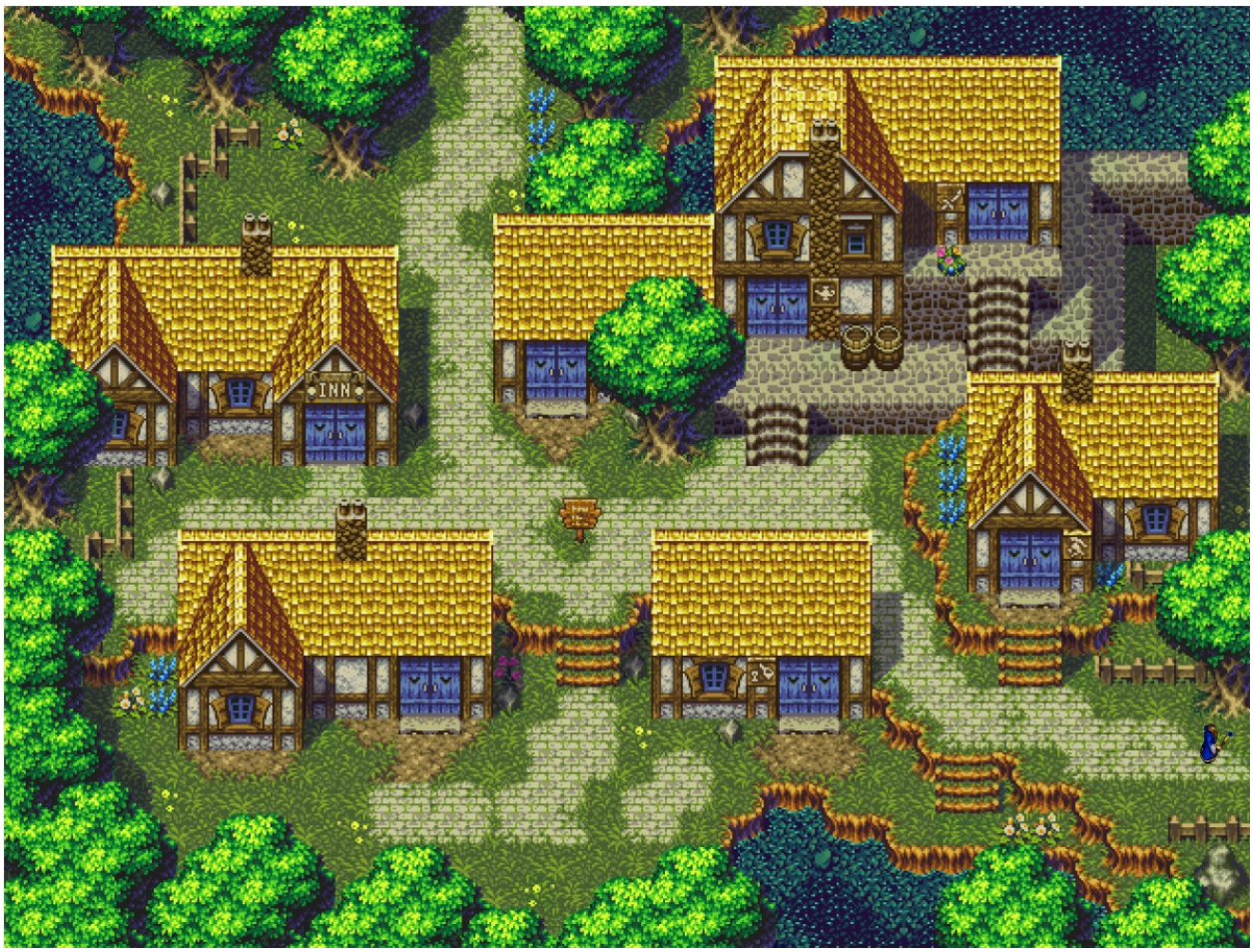

Рисунок 4.6 – Персонаж mage, не может выйти за пределы видимой зоны Village

На рисунке 4.7 представлен пример перехода персонажа из зоны.

Рисунок 4.7 – Персонаж mage, переходит из зоны Village в зону Ruins

На рисунке 4.8 представлен пример установки новой зоны.

Рисунок 4.8 – Пример работы программы с тремя персонажами внутри одной, игровой зоны Ruins

На рисунке 4.9 представлен пример работы сценария движения персонажа.

Рисунок 4.9 – Пример работы сценария walk(50, 50, self.footman), игровой зоны Ruins

На рисунке 4.10 представлен пример работы сценария поведения персонажа противника.

Рисунок 4.10 – Пример работы сценария ai(self.grunt), игровой зоны Ruins

На рисунке 4.11 представлен пример работы метода click персонажа.

Рисунок 4.11 – Вызов метода click, у персонажа Grunt

ЗАКЛЮЧЕНИЕ

В заключение, платформа для создания компьютерных изометрических ролевых игр с заранее отрисованным двумерным фоном и спрайтовыми персонажами представляет собой мощный инструмент, который открывает широкие возможности для разработчиков и дизайнеров. Она позволяет воплощать в жизнь уникальные игровые миры с богатой графикой и детализированными персонажами, сохраняя при этом классическое ощущение и глубину RPG. Эта платформа не только упрощает процесс разработки игр, но и делает его более доступным для широкого круга творческих людей, желающих реализовать свои идеи без необходимости владения сложными навыками программирования. Таким образом, она способствует росту индустрии компьютерных игр и обогащает культурное пространство новыми, захватывающими проектами.

Основные результаты работы:

1. Проведен анализ предметной области.
2. Разработана концептуальная модель приложения. Разработана модель данных системы. Определены требования к системе.
3. Осуществлено проектирование приложения. Разработан пользовательский интерфейс приложения.
4. Реализовано и протестировано приложение. Проведено модульное и системное тестирование.

Все требования, объявленные в техническом задании, были полностью реализованы, все задачи, поставленные в начале разработки проекта, были также решены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Изучаем Python / М. Лутц. – Санкт-Петербург : Диалектика, 2013. – 1648 с. – ISBN 978-5-907144-52-1. – Текст : непосредственный.

2. Изучаем Python. Программирование игр, визуализация данных, веб-приложения / Э. Мэтиз. – Санкт-Петербург : Питер, 2016. – 544 с. – ISBN 978-5-496-02305-4. – Текст : непосредственный.

3. Автоматизация рутинных задач с помощью Python / Э. Свейгарт. – Москва : И.Д. Вильямс, 2016. – 592 с. – ISBN 978-5-8459-20902-4. – Текст : непосредственный.

4. Эл Свейгарт: Учим Python, делая крутые игры / Э. Свейгарт. – Москва : Бомбора, 2021 г. – 416 с. – ISBN 978-5-699-99572-1. – Текст : непосредственный.

5. Программист-прагматик. Путь от подмастерья к мастеру / Э. Хант, Д. Томас. – Санкт-Петербург : Диалектика’, 2020. – 368 с. – ISBN 978-5-907203-32-7. – Текст : непосредственный.

6. Совершенный код / С. Макконнелл. – Москва : Издательство «Русская редакция», 2010. — 896 стр. – ISBN 978-5-7502-0064-1. – Текст : непосредственный.

7. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – Санкт-Петербург : Питер, 2021. – 368 с. – ISBN 5-272-00355-1. – Текст : непосредственный.

8. Рефакторинг. Улучшение существующего кода / Ф. Мартин. – Москва : Диалектика-Вильямс, 2019 – 448 с. – ISBN 978-5-9909445-1-0. – Текст : непосредственный.

9. Роберт Мартин: Чистый код. Создание, анализ и рефакторинг / Р. Мартин. – Санкт-Петербург : Питер, 2020 г, 2016 – 464 с. – ISBN 978-5-4461-0960-9. – Текст : непосредственный.

10. Dungeons & Dragons. Книга игрока / Wizards of the Coast. – Минск : ИП Якосенко А.А., 2014 – 320 с. – ISBN 978-5-6041656-8-3. – Текст : непосредственный.

11. Dungeons & Dragons. Руководство мастера подземелий / Wizards of the Coast. – Минск : ИП Якосенко А.А., 2014 – 320 с. – ISBN 978-5-907170-20-9. – Текст : непосредственный.

12. Dungeons & Dragons. Бестиарий. Энциклопедия чудовищ / Wizards of the Coast. – Минск : ИП Якосенко А.А., 2014 – 400 с. – ISBN 978-0786965618. – Текст : непосредственный.

ПРИЛОЖЕНИЕ А

Представление графического материала

Графический материал, выполненный на отдельных листах, изображен на рисунках А.1–А.7.

ВКРБ 20060139.09.03.04.24.012

1

Сведения о ВКРБ

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПО ПРОГРАММЕ БАКАЛАВРИАТА

«Платформа для создания компьютерных изометрических ролевых игр
с заранее отрисованным двухмерным фоном и спрайтовыми персонажами»

Руководитель ВКРБ
к.т.н, доцент
Чаплыгин Александр Александрович

Автор ВКРБ
студент группы ПО-02б
Шевченко Клим Николаевич

	Фамилия И. О.	Подпись	Дата	ВКРБ 20060139.09.03.04.24.012		
Автор работы	Шевченко К.Н.			Сведения о ВКРБ	Лит.	Масса	Масштаб
Руководитель	Чаплыгин А.А.						
Нормоконтроль	Чаплыгин А.А.				Лист 1	Листов 7
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-02б	

Рисунок А.1 – Сведения о ВКРБ

Цель и задачи разработки

Цель работы - разработка платформы для создания компьютерных изометрических ролевых игр с заранее отрисованным двухмерным фоном и спрайтовыми персонажами.

Для достижения поставленной цели требуется решить следующие задачи:

1. Анализ существующих ролевых игр.
2. Разработать концептуальную модель ролевых игр.
3. Проектирование программной системы для создания ролевых игр.
4. Реализация программной системы для создания ролевых игр.
5. Тестирование разработанной системы.

Рисунок А.2 – Цель и задачи разработки

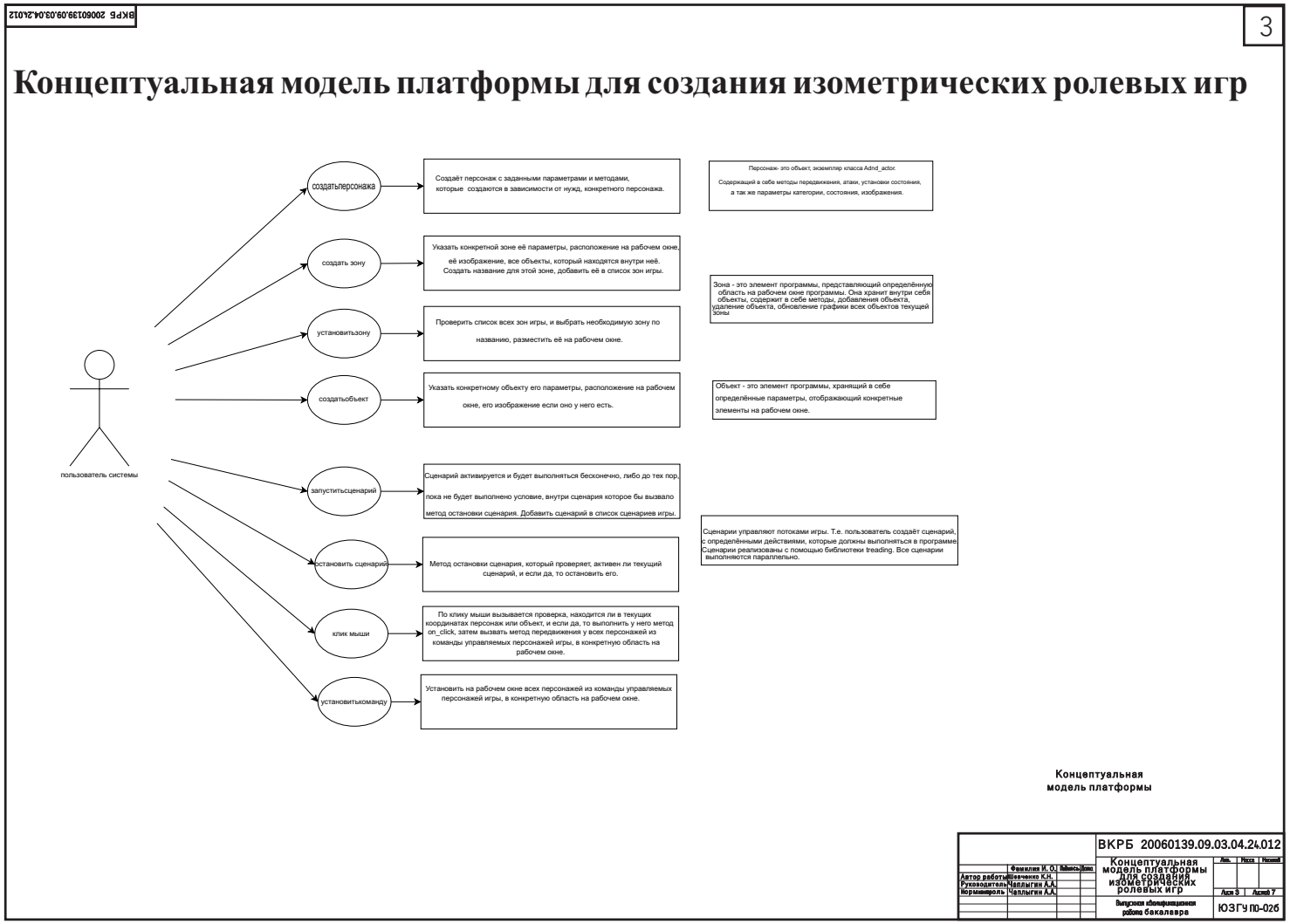

Рисунок А.3 – Концептуальная модель приложения

Архитектура платформы для создания изометрических ролевых игр

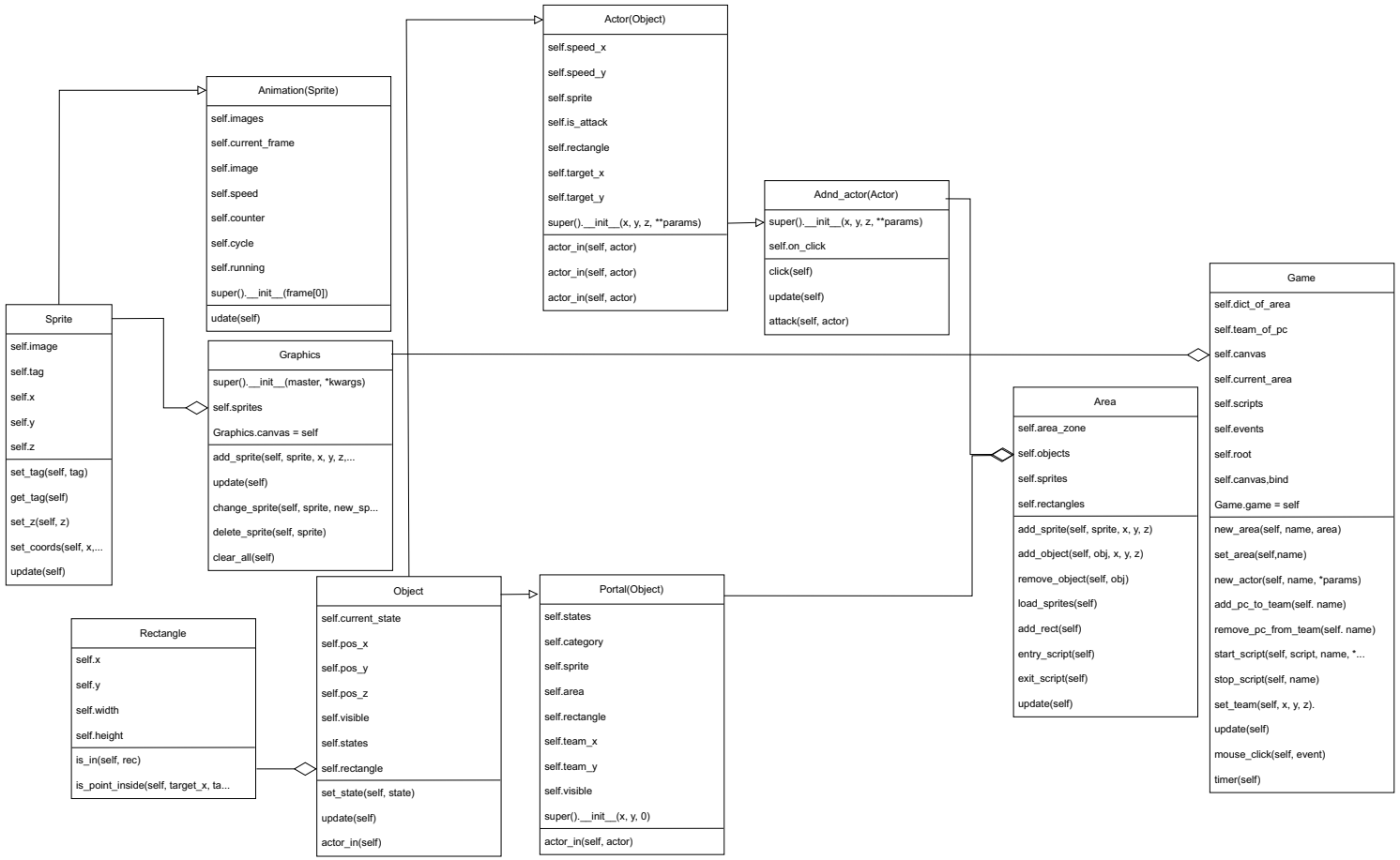

Рисунок А.4 – Диаграмма классов

Модель работы сценариев

В Python потоки — это легковесные процессы, которые могут выполняться параллельно. В библиотеке threading потоки управляются операционной системой, которая решает, когда и как долго каждый поток будет выполняться. Это называется планированием потоков, и оно обычно происходит без вмешательства программиста.
Однако, можно создать модель, которая иллюстрирует переключение между потоками в Python. Представим, что у нас есть три потока: A, B и C. Каждый поток выполняет функцию worker, которая занимает определенное время. Планировщик ОС может переключаться между потоками, например, после выполнения каждой инструкции или при блокировке операции ввода-вывода.

Время	Поток A	Поток B	Поток C
t0	start		
t1		start	
t2			start
t3	work		
t4		work	
t5			work
t6	work		
t7		work	
t8			work
t9	finish		
t10		finish	
t11			finish

В этой модели:
Время t0, t1, t2 — это моменты времени, когда каждый поток начинает работу.
work означает, что поток выполняет свою функцию.
finish означает, что поток завершил свою работу.
Пустые ячейки означают, что поток в данный момент времени не активен

Рисунок А.5 – Модель работы сценариев

Модульное тестирование платформы

функция тестирования	входные данные	ожидаемый результат
Прямоугольник внутри другого прямоугольника	rect=Rectangle(1,1,4,4) rect_outside = Rectangle(0, 0, 6, 6)	вернёт значение True
Прямоугольник снаружи другого прямоугольника	rect=Rectangle(1,1,4,4) rect_inside = Rectangle(2, 2, 2, 2)	вернёт значение False
Прямоугольник отдельно от другого	rect=Rectangle(1,1,4,4) rect_apart = Rectangle(6, 6, 2, 2)	вернёт значение False
Прямоугольник касается слева	rect=Rectangle(1,1,4,4) touching_left = Rectangle(0, 2, 1, 1)	вернёт значение False
Прямоугольник касается справа	rect=Rectangle(1,1,4,4) touching_right = Rectangle(5, 2, 1, 1)	вернёт значение False
Прямоугольник касается сверху	rect=Rectangle(1,1,4,4) touching_top = Rectangle(2, 5, 1, 1)	вернёт значение False
Прямоугольник касается снизу	rect=Rectangle(1,1,4,4) touching_bottom = Rectangle(2, 0, 1, 1)	вернёт значение False
Пересечение прямоугольника слева	rect=Rectangle(1,1,4,4) intersect_left = Rectangle(0, 2, 3, 2)	вернёт значение False
Пересечение прямоугольника справа	rect=Rectangle(1,1,4,4) intersect_right = Rectangle(3, 2, 3, 2)	вернёт значение False
Пересечение прямоугольника сверху	rect=Rectangle(1,1,4,4) intersect_top = Rectangle(2, 3, 2, 3)	вернёт значение False
Пересечение прямоугольника снизу	rect=Rectangle(1,1,4,4) intersect_bottom = Rectangle(2, 0, 2, 3)	вернёт значение False

Рисунок А.6 – Модульное тестирование платформы

Заключение

В заключение, платформа для создания компьютерных изометрических ролевых игр с заранее отрисованным двумерным фоном и спрайтовыми персонажами представляет собой мощный инструмент, который открывает широкие возможности для разработчиков и дизайнеров. Она позволяет воплощать в жизнь уникальные игровые миры с богатой графикой и детализированными персонажами, сохраняя при этом классическое ощущение и глубину RPG. Эта платформа не только упрощает процесс разработки игр, но и делает его более доступным для широкого круга творческих людей, желающих реализовать свои идеи без необходимости владения сложными навыками программирования. Таким образом, она способствует росту индустрии компьютерных игр и обогащает культурное пространство новыми, захватывающими проектами.

Основные результаты работы:

Проведен анализ предметной области.
Разработана концептуальная модель приложения. Разработана модель данных системы. Определены требования к системе.
Осуществлено проектирование приложения. Разработан пользовательский интерфейс приложения.
Реализовано и протестировано приложение. Проведено модульное и системное тестирование.

Все требования, объявленные в техническом задании, были полностью реализованы, все задачи, поставленные в начале разработки проекта, были также решены.

Рисунок А.7 – Заключение

ПРИЛОЖЕНИЕ Б

Фрагменты исходного кода программы

sprite.py

```
import tkinter as tk
class Sprite:

    def __init__(self, image):
        '''
        Класс спрайта для работы с изображениями на Canvas

        :param image: адресс изображения который
        '''
        self.image = tk.PhotoImage(file=image)
        self.tag = None
        self.x = 0
        self.y = 0
        self.z = 0  # z-координата спрайта

    def set_tag(self, tag):
        '''
        Устанавливает тег спрайта

        :param tag: тег спрайта
        '''
        self.tag = tag

    def set_z(self, z):
        '''
        Устанавливает z-координату спрайта

        :param z: координата z
        '''
        self.z = z

    def get_tag(self):
        '''
        Возвращает тег спрайта

        '''
        return self.tag

    def set_coords(self, new_x, new_y):
        '''
        Обновляет координаты спрайта

        :param new_x: координата x
        :param new_y: координата y
        '''
        if self.tag:
            self.x = new_x
            self.y = new_y
    def update(self):
```

```
        '''
        Обновляет анимацию спрайта

        '''
        pass
```

graphics.py

```
import tkinter as tk

class Graphics(tk.Canvas):
    canvas = None
    def __init__(self, master, **kwargs):
        '''
        Класс с методами для работы со спрайтами

        '''
        super().__init__(master, **kwargs)
        self.sprites = []  # список спрайтов
        Graphics.canvas = self

    def add_sprite(self, sprite, x, y, z, **kwargs):
        '''
        Добавляет спрайт на Canvas

        :param sprite: спрайт
        :param x: координата x
        :param y: координата y
        :param z: координата z
        :param kwargs: параметры относящиеся к конкретному изображению в
            tkinter
        '''
        tag = self.create_image(x, y, image=sprite.image, anchor='center', **
            kwargs)
        sprite.set_tag(tag)
        sprite.set_z(z)  # устанавливаем z-координату спрайта
        sprite.x = x
        sprite.y = y
        self.sprites.append(sprite)
        self.sprites.sort(key=lambda sprite: sprite.z)  # сортировка спрайтов
            по z-координате

    def update(self):
        '''
        Перерисовывает все спрайты

        '''
        for sprite in self.sprites:
            sprite.update()
            self.tag_raise(sprite.get_tag())  # перемещаем спрайт на передний
                план
            self.coords(sprite.get_tag(), sprite.x, sprite.y)
            self.itemconfig(sprite.get_tag(), image=sprite.image)


```

```
    def change_sprite(self, sprite, new_sprite):
        '''
        Меняет  спрайт на новый.

        :param sprite: экземпляр спрайта
        :param new_sprite: новый спрайт
        '''
        old_sprite_pos = None
        for i, s in enumerate(self.sprites):
            if s.get_tag() == sprite.get_tag():
                old_sprite_pos = i
                break

        if old_sprite_pos is not None:
            old_tag = sprite.get_tag()

            # Обновляем список спрайтов
            self.sprites[old_sprite_pos] = new_sprite
            new_sprite.set_tag(old_tag)

            # Обновляем тег нового спрайта
            new_sprite.set_tag(old_tag)
            new_sprite.set_z(sprite.z)

            # Перемещаем новый спрайт на передний план и обновляем его
                координаты
            self.tag_raise(old_tag)
            self.coords(old_tag, sprite.x, sprite.y)
            self.itemconfig(old_tag, image=new_sprite.image)

    def delete_sprite(self, sprite):
        '''
        Удаляет спрайт с Canvas.

        :param sprite: экземпляр спрайта
        :return:
        '''
        self.delete(sprite.get_tag())
        self.sprites.remove(sprite)

    def clear_all(self):
        '''
        Удаляет все спрайты с Canvas

        '''
        for sprite in self.sprites:
            self.delete(sprite.get_tag())
        self.sprites.clear()
```

game.py

```
from rpg.area import *
from rpg.graphics import *
from rpg.sprite import *
from rpg.actor import *
```

```
5  from rpg.adnd_actor import Adnd_actor
6  import threading
7
8  class Game():
9      def __init__(self, canvas, window, **params):
10         '''
11         Класс системы управления игрой
12
13         :param canvas: класс графической системы
14         :param window: окно на которое будет выводится игра
15         '''
16         self.rpg_dict_of_area = {} # словарь, хранящий в себе множество
               экземпляров класса Area, {number - ключ : name Area - значение}
17         self.team_of_pc = [] # список, хранящий в себе имена экземпляров
               класса Actor с параметром category = "pc"
18         self.canvas = canvas # графика
19         self.root = window # окно для графики
20         self.current_area = None # параметр хранящий, текущую зону
21         self.scripts = {}  # Словарь для хранения запущенных сценариев
22         self.events = {} # Словарь для хранения запущенных event`ов сценариев
23         self.canvas.bind("<Button-1>", self.mouse_left_click)
24         Game.game = self
25
26     def new_area(self, name, area):
27         '''
28         Добавляет новую зону в список
29
30         :param name: имя зоны
31         :param area: класс area
32         '''
33         self.rpg_dict_of_area[name] = area
34
35     def set_area(self, name):
36         '''
37         Устанавливает текущую зону, загружает графику зоны.
38
39         :param name: имя зоны
40         '''
41         if name in self.rpg_dict_of_area:
42             if self.current_area is not None:
43                 self.current_area.exit_script()
44                 for pc in self.team_of_pc:
45                     self.current_area.remove_object(pc)
46             self.current_area = self.rpg_dict_of_area[name]
47             self.canvas.clear_all()
48             self.current_area.load_sprites()
49             for pc in self.team_of_pc:
50                 self.current_area.add_object(pc, pc.pos_x, pc.pos_y, pc.pos_z
                       )
51
52     def new_actor(self, name, **params):
53         '''
54         Создаёт класс, потомок от Actor и создаёт поле из параметров, и
               установление их в начальные значения.
```

```
55
56          :param name: название нового класса
57          :param params: поля нового класса
58          '''
59          class_attributes = {}
60          for key, value in params.items():
61              class_attributes[key] = value
62          return type(name, (Actor,), class_attributes)
63
64      def add_pc_to_team(self, pc):
65          '''
66          Добавдяет персонажа в команду
67          :param pc: персонаж, которого нужно добавить в команду
68          '''
69          if pc.category == "pc":
70              self.team_of_pc.append(pc)
71          else: print('попытка добавить персонажа в команду не успешна')
72
73      def remove_pc_from_team(self, pc):
74          ''' удаляет персонажа из команды '''
75          if pc.category == "pc":
76              self.team_of_pc.remove(pc)
77          else:
78              print('попытка удалить персонажа из команды не успешна')
79
80      def start_script(self, script_function, script_name, *args):
81          '''
82          Запускает сценарий в отдельном потоке с возможностью остановки и
              передачи аргументов.
83
84          :param: script_function: Функция, содержащая код сценария.
85          :param: script_name: Имя сценария.
86          :param: args: Дополнительные аргументы, которые нужно передать в
              сценарий.
87          '''
88          # Создание потока для сценария
89          e = threading.Event()
90          self.events[script_name] = e
91
92          def func(e, args):
93              while not e.is_set():
94                  script_function(*args)
95
96          # Создание потока для сценария
97          script_thread = threading.Thread(target=func, args=(e, args))
98          script_thread.daemon = True
99          script_thread.start()
100
101         # Добавление потока в словарь активных сценариев
102         self.scripts[script_name] = script_thread
103
104     def stop_script(self, script_name):
105         '''
106         Останавливает сценарий по имени
```

```
107
108         :param: script_name: имя сценария, который нужно остановить
109         '''
110         # Проверка существования сценария
111         if script_name in self.scripts:
112             # Если сценарий существует, прерываем его выполнение
113             self.events[script_name].set()
114             # Убираем сценарий из словаря активных сценариев
115             del self.scripts[script_name]
116             del self.events[script_name]
117             print(f"Сценарий {script_name} остановлен.")
118         else:
119             # Если сценарий не существует
120             print(f"Сценарий {script_name} не существует.")
121
122     def set_team(self, x, y, z):
123         '''
124         Устанавливает координаты персонажей команды
125
126         :param x: координата x
127         :param y: координата y
128         :param z: координата z
129         '''
130         for elememt in self.team_of_pc:
131             elememt.pos_x = x
132             elememt.pos_y = y
133             elememt.pos_z = z
134
135     def update(self):
136         '''
137         Вызывается в таймере для обновления всех переменных в текущей зоне
138
139         '''
140         self.current_area.update()
141         self.canvas.update()
142
143     def mouse_left_click(self, event):
144         '''
145         обрабатывает клик мыши
146
147         :param event:
148         '''
149         for actor in self.current_area.objects:
150             if actor.category == 'enemy':
151                 if not actor.rectangle.is_point_inside(event.x, event.y):
152                     actor.on_click()
153
154         for actor in self.current_area.objects:
155             if actor.category == 'pc':
156                 actor.search_position(event.x, event.y)
157
158     def new_item(self, name, **params):
159         '''
```

```
160         Создаёт класс, потомок от Item и создаёт поле из параметров, и
                установление их в начальные значения.
161
162         :param name: название нового класса
163         :param params: поля нового класса
164         '''
165         '''item_attributes = {}
166         for key, value in params.items():
167             item_attributes[key] = value
168         new_item = type(name, (Item,), item_attributes)
169         # Добавление нового предмета в словарь игры
170         self.items[name] = new_item
171         return new_item'''
172
173     def new_spell(self, name, **params):
174         ''' добавляет новое заклинание '''
175         self
176
177     def timer(self):
178         '''
179         Таймер дожен вызывать метод update постоянно
180
181         '''
182         self.update()
183         self.root.after(50, self.timer)
184
185 def new_actor(self, name, **params):
186         '''
187         Создаёт класс, потомок от Actor и создаёт поле из параметров, и
                установление их в начальные значения.
188
189         :param name: название нового класса
190         :param params: поля нового класса
191         '''
192         class_attributes = {}
193         for key, value in params.items():
194             class_attributes[key] = value
195         return type(name, (Adnd_actor,), class_attributes)
```

Место для диска