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) Изучить основные принципы библиотеки treading.

2) Разработать концептуальную модель движка для создания компьютерных ролевых игр.

3) Спроектировать программную систему создания игры.

4) Сконструировать и протестировать программную систему движка.

3.2. Входные данные и требуемые результаты для программы:

1) Входными данными для программной системы являются: данные справочников комплектующих, конфигураций, ПО, критериев качества SLA, ИТ-услуг, департаментов компании; технические данные ИТ-ресурсов; данные входящих заявок на ИТ-ресурсы; данные запросов поставщикам на комплектующие.

2) Выходными данными для программной системы являются: сформированные заявки на обслуживание ИТ-ресурсов; сформированные запросы на закупку комплектующих; сведения о выполненных работах по заявкам; статусы заявок; выходные отчеты (инфографика) – по качеству услуг, по состоянию ИТ-ресурсов, по деятельности ИТ-отдела, по стоимости обслуживания ИТ-ресурсов, воронка заявок.

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 85 страницам. Работа содержит 13 иллюстраций, 1 таблицу, 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 85 pages. The work contains 13 illustrations, 1 table, 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 11 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 игр с 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. Исходя из этого, основную цель предлагается рассмотреть в разрезе двух групп подцелей.

Задачами данной разработки являются:

– проектирование интерфейса;
– разработка архитектуры приложения;
– проектирование игровых сценариев;
– реализация взаимодействия приложения с пользователем;
– реализация графики приложения;

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 Пример игры

– ролевая игра моделирует все основные механики Dungeons and Dragons, в которой игрок управляет персонажем, который бродит по одноуровневому подземелью, собирая сокровища и убивая монстров. Подземелье визуализируется в двухмерном виде сверху с использованием экранной графики персонажей и управляется с помощью команд с мыши. Подземелье имеет фиксированную планировку, но встречи с монстрами и сокровища генерируются заданным образом.

– 1. Цель игры: Основная цель игры заключается в исследовании мира, выполнении заданий и квестов, сражении с врагами и развитии своего персонажа. Игра также имеет главный сюжет, который игрок может прогрессировать, следуя определенным событиям и заданиям

– 2. Боевая система: Бои могут происходить в режиме реального времени. Игрок может управлять группой персонажей и давать им команды в бою. В бою игрок может использовать различные атаки, заклинания и способности своего персонажа для победы над врагами

– 3. Персонажи: Игрок может создать своего уникального персонажа, выбрав класс, расу, навыки и характеристики. Каждый класс имеет свои особенности и специализации, определяющие стиль игры и возможности персонажа. Персонажи могут повышать уровень, получать новые навыки и способности, улучшать характеристики и собирать экипировки

– 4. Исследование мира: Игрок может свободно перемещаться по миру игры, исследуя различные локации и взаимодействуя с окружающими объектами. Во время исследования игрок может встретить неигровых персонажей (NPC), с которыми можно общаться, получать задания и информацию о мире.

– 5. Прогрессия и развитие: Игрок может зарабатывать опыт и повышать уровень своего персонажа. Повышение уровня позволяет персонажу получать новые навыки, улучшать характеристики и получать новые способности. Игрок также может собирать и улучшать экипировку для своего персонажа, чтобы повысить его силу и выживаемость.

– 6. Задания и квесты: Игрок может выполнять различные задания и квесты, предлагаемые неигровыми персонажами. Задания могут включать поиск предметов, убийство определенных врагов, решение головоломок и т.д. За выполнение заданий игрок может получать награды, опыт и продвигаться в сюжете игры.

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 Интерфейс пользователя

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.

2.7 Моделирование вариантов использования

На основании анализа предметной области в программе должны быть реализованы следующие прецеденты:

1. Создание персонажа.
2. Создание зоны.
3. Создание объекта.
4. Удаление объекта.
5. Создание сценария.
6. Удаление сценария.

Таким образом, на рисунке 2.2сформированы следующие действия пользователя и их последствия.

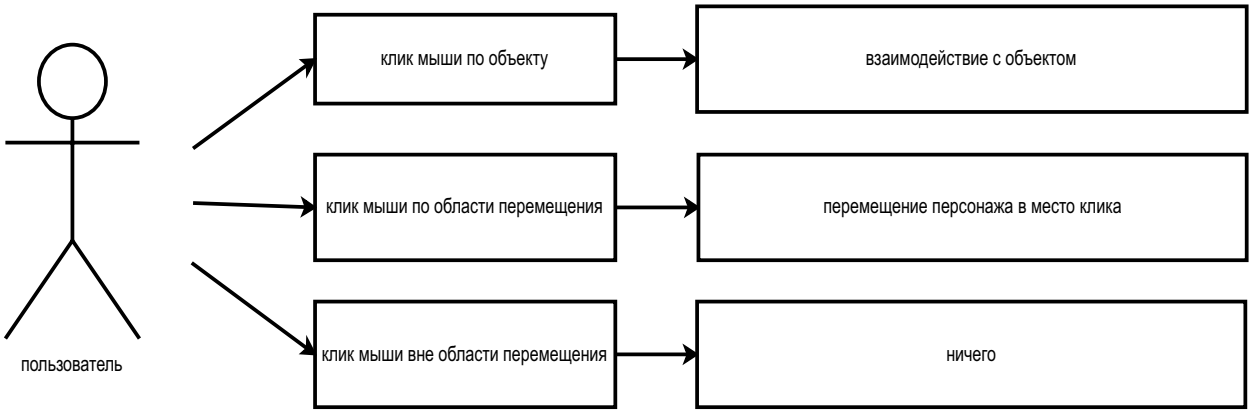


Рисунок 2.2 – Шаблон интерфейса игры

2.8 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.

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++.

3.2.3 Использование библиотеки Tkinter и реализация таймеров 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:

– Создание различных виджетов: кнопки, метки, поля ввода, списки и многое другое.

– Управление компоновкой виджетов с использованием менеджеров компоновки (например, grid, pack, place).

– Обработка событий, таких как щелчок мыши, нажатие клавиш и другие.

– Возможность создания различных диалоговых окон, таких как окна предупреждений, информационные окна и окна запроса.

– Поддержка многопоточности для обновления интерфейса из различных потоков выполнения.

3.2.3.3 Реализация таймеров на Python

Для реализации таймеров на Python можно использовать модуль time или threading. Вот пример использования модуля time для создания простого таймера:

import time

def countdown(t): while t > 0: mins, secs = divmod(t, 60) timeformat = ':02d::02d'.format(mins, secs) print(timeformat, end='") time.sleep(1) t -= 1 print('Таймер завершен!')

t = 10 countdown(t)

Этот код создает простой обратный отсчет таймера с использованием функции countdown. Он выводит оставшееся время в формате MM:CC и уменьшает его на 1 каждую секунду, используя функцию time.sleep(1). Когда время истекает, выводится сообщение о завершении таймера.

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 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 персо-

нажу 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 может 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): 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): ”’ Класс игровой зоны Ruins

”’ 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 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): ”’ Сценарий для движения бугая

:param step_x: шаг движения x :param step_y: шаг движения y ”’ 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)

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()

3.3.1 Пример клиентского кода игры

3.3.1.1 Создание классов персонажей/предметов

Клиент создает модуль содержащий методы модуля RPGGame, например BaldursGateGame. В этом модуле клиент создаем мир игры, с помощью new_actor.

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()

3.3.1.2 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:

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): ''' Класс Adnd_actor содержащий основные механики взаимодействия с другими персонажами

:param x: координата x :param y: координата y :param z: координата z ''' super().__init__(x, y, z, **params) self.on_click = self.click

def click(self): ''' вызывается при клике на персонажа

''' 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): ''' совершает атаку по actor

:param actor: персонаж, которого атакуют ''' actor.hp -= self.damage def update(self): ''' обновляет состояние персонажа

''' super().update() if self.hp <= 0: self.stop_move() self.set_state('death')

3.3.1.3 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. Добавляем 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.

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): ''' Класс игровой зоны Ruins

''' 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
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): ”’ Сценарий для движения бугая

:param step_x: шаг движения x :param step_y: шаг движения y ”’ 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)

модуль bggame: from ruins import * import time import random

3.3.1.4 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.

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): ”’ Создает портал в новую зону

:param x: координата x портала :param y: координата y портала :param width: ширина портала :param height: высота портала :param area: имя зоны куда будет переход :param team_x: местоположение команды в новой зоне :param team_y: местоположение команды в новой зоне ”’ 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): ”’ Проверяет находится ли персонаж внутри портала

:param actor: проверяемый персонаж ”’ 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): ”’ Класс игровой зоны Ruins

”’ 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
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)

3.3.1.5 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: 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): ”’ Класс Adnd_actor содержащий основные механики взаимодействия с другими персонажами

:param x: координата x :param y: координата y :param z: координата z ”’ super().__init__(x, y, z, **params) self.on_click = self.click

def click(self): ”’ вызывается при клике на персонажа

”’ 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): ”’ совершает атаку по actor

:param actor: персонаж, которого атакуют ”’ actor.hp -= self.damage def update(self): ”’ обновляет состояние персонажа

”’ super().update() if self.hp <= 0: self.stop_move() self.set_state(’death’)

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): ”’ Класс игровой зоны Ruins

”’ 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
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): ”’ Сценарий для движения бугая

:param step_x: шаг движения x :param step_y: шаг движения y ”’ 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)

def ai(self, actor): ”’ скрипт противников

:param step_x: размер шага x до персонажа игрока :param step_y: размер шага x до персонажа игрока :param actor: персонаж против-ник ”’ 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”) Game.game.start_script(self.walk, ”grunt 50, 50, actor)

else: actor.is_attack = False time.sleep(2)

3.3.1.6 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)

модуль sprite

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

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

модуль graphics

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 = [] 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) sprite.x = x sprite.y = y self.sprites.append(sprite) self.sprites.sort(key=lambda sprite: sprite.z)

def update(self): ”’ Перерисовывает все спрайты

”’ 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): ”’ 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()

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): ”’ 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()

модуль baldursgame ”’пользовательский модуль”’

from ruins import * from village 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.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()

3.4 Архитектура платформы для создания ролевых игр

3.4.1 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. Основными 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).

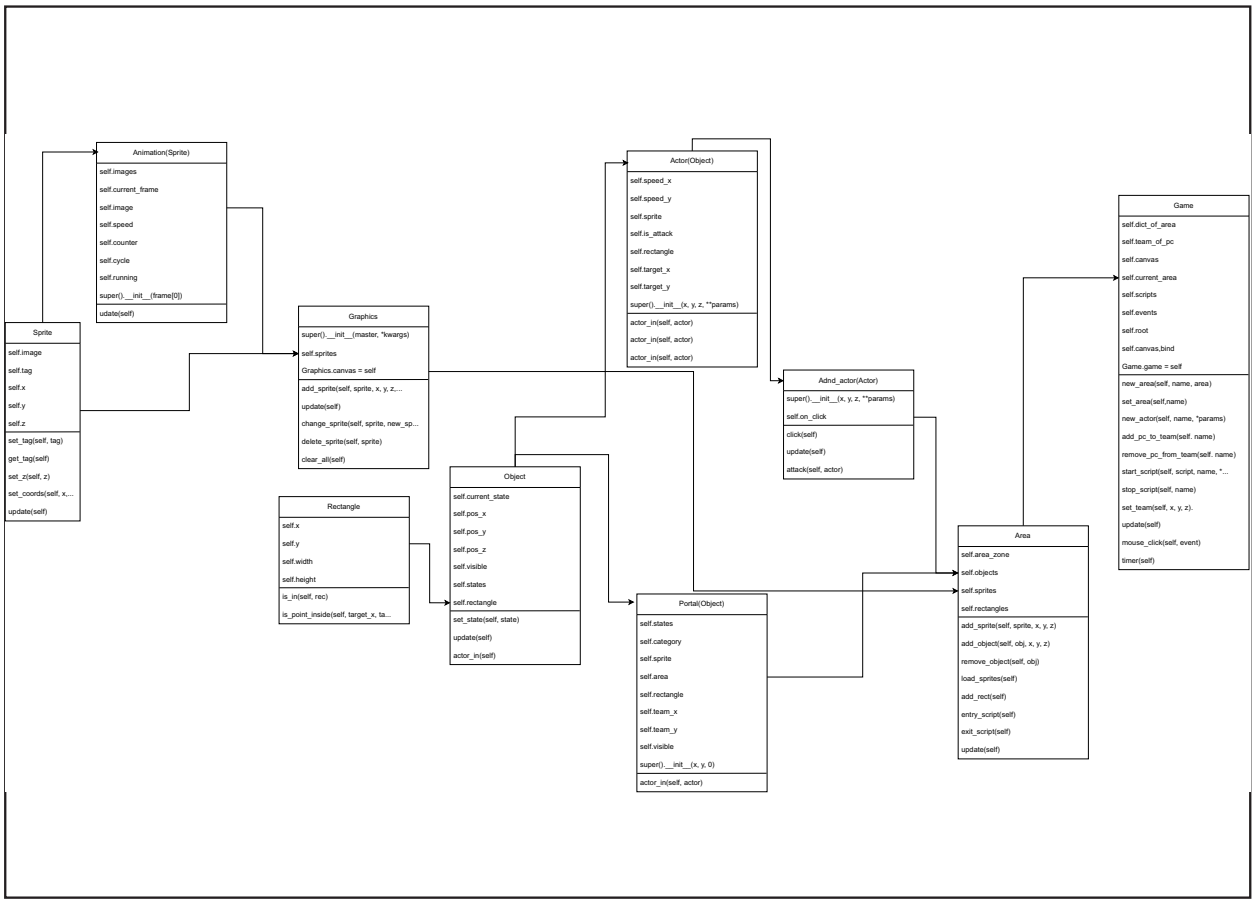


Рисунок 3.1 – Диаграмма компонентов

3.4.1.1 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.

3.4.2 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.

3.4.2.1 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:

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) self.sprites.append(sprite) self.sprites.sort(key=lambda sprite: sprite.z)

3.4.3 Реализация зон

Зоны в программе представляют собой различные игровые области или уровни. Каждая зона реализована через класс Area, который содержит спрайты и объекты, принадлежащие этой зоне. Зоны могут содержать свои собственные скрипты для входа и выхода из зоны (entry_script и exit_script), а также метод update, который обновляет состояние всех объектов в зоне.

3.4.4 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.

3.4.5 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.

3.4.6 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:

def is_point_inside(self, target_x, target_y): return (self.x <= target_x <= self.x + self.width) and (self.y <= target_y <= self.y + self.height) 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).

4 Рабочий проект

4.1 Классы, используемые при разработке приложения

Можно выделить следующий список классов и их методов, использованных при разработке приложения (таблица 4.1). Пример таблицы с уменьшенным межстрочным интервалом.

Таблица 4.1 – Описание классов платформы, используемых в приложении

Название класса	Модуль, к которому относится класс	Описание класса	Методы
1	2	3	4
sprite	rpg	Sprite – Инициализация класса Sprite для работы с изображениями на холсте Canvas.	set_tag(self, tag) Устанавливает тег для спрайта. set_z(self, z) Устанавливает z-координату спрайта. get_tag(self) Возвращает тег спрайта. set_coords(self, new_x, new_y) Обновляет координаты спрайта. update(self) Обновляет анимацию спрайта, если она у него есть.
animation	rpg	Animation – Класс анимации спрайта	update(self) Меняет текущее изображение в списке изображений.

Продолжение таблицы 4.1

1	2	3	4
graphics	rpg	Graphics – Класс с методами для работы со спрайтами	add_sprite(self, sprite, x, y, z, **kwargs) Добавляет спрайт на Canvas. update(self) Перерисовывает все спрайты. change_sprite(self, sprite, new_sprite) Меняет спрайт на новый в Canvas. delete_sprite(self, sprite) Удаляет спрайт с Canvas. clear_all(self) Удаляет все спрайты с Canvas.
rectangle	rpg	Rectangle – Класс прямоугольника, используемый для перемещения	is_in(self, rect) Проверяет, входит ли прямоугольник self в прямоугольник rect. is_point_inside(self, target_x, target_y) Проверяет, входит ли точка (x, y) в данный прямоугольник.
object	rpg	Object – Класс объекта, который будет изменяться методами игровой системы и методами графической системы	set_state(self, state_name) Меняет текущее состояние объекта. actor_in(self, actor) Вызывается когда actor входит внутрь объекта. update(self) Этот метод будет изменён в классах наследниках от object.

Продолжение таблицы 4.1

1	2	3	4
portal	rpg	Portal – Класс портала, используемый для перемещения команды персонажей в новую зону	actor_in(self, actor) Проверяет находится ли персонаж внутри портала.
actor	rpg	Actor – Класс Actor для работы с персонажем	update(self) Изменяет координаты и состояние персонажа. search_position(self, new_x, new_y) Изменяет направление движения у персонажа. stop_move(self) Останавливает движение персонажа.
adnd_actor	rpg	Adnd_actor – Класс Adnd_actor содержащий основные механики взаимодействия с другими персонажами	click(self) Вызывается при клике на персонажа. attack(self, actor) Совершает атаку по actor. update(self) Обновляет состояние персонажа.

Продолжение таблицы 4.1

1	2	3	4
area	rpg	Area – Класс Area, содержащий все поля и методы используемые в каждой зоне	add_sprite(self, sprite, x, y, z) Добавляет спрайт в зону. add_object(self, obj, x, y, z) Добавляет объект в зону. remove_object(self, obj) Удаляет объект из зоны. load_sprites(self) Загружает все спрайты зоны. add_rect(self, rec) Добавляет прямоугольник в зону. entry_script(self) Запускается, когда команда входит в зону exit_script(self) Запускается, когда команда выходит из зоны update(self) Изменяет и проверяет изменение всех объектов в зоне.

Продолжение таблицы 4.1

1	2	3	4
game	rpg	Game – Класс системы управления игрой	new_area(self, name, area) Добавляет новую зону в список. set_area(self, name) Устанавливает текущую зону, загружает графику зоны. new_actor(self, name, **params) Создаёт класс, потомок от Actor и создаёт поле из параметров, и установление их в начальные значения. add_pc_to_team(self, pc) Добавляет персонажа в команду. remove_pc_from_team(self, pc) Удаляет персонажа из команды. start_script(self, script_function, script_name, *args) Запускает сценарий в отдельном потоке с возможностью остановки и передачи аргументов. stop_script(self, script_name) Останавливает сценарий по имени. set_team(self, x, y, z) Устанавливает координаты персонажей команды. update(self) Вызывается в таймере для обновления всех переменных в текущей зоне. mouse_left_click(self,

Продолжение таблицы 4.1

1	2	3	4
village	рабочая система	Village – Класс зоны Village	__init__(self) Инициализирует все поля и методы внутри конкретной зоны.
footman	рабочая система	Footman – Класс наследник от Adnd_actor	__init__(self, x, y, z) Инициализирует все поля и методы внутри конкретного экземпляра класса Footman.
grunt	рабочая система	Grunt – Класс наследник от Adnd_actor	__init__(self, x, y, z) Инициализирует все поля и методы внутри конкретного экземпляра класса Grunt.
mage	рабочая система	Mage – Класс наследник от Adnd_actor	__init__(self, x, y, z) Инициализирует все поля и методы внутри конкретного экземпляра класса Mage.
ruins	рабочая система	Ruins – Класс зоны Ruins	__init__(self) Инициализирует все поля и методы внутри конкретной зоны. walk(self, step_x, step_y, actor) Сценарий для движения персонажа. ai(self, actor) Сценарий для персонажей противников.
bggame	рабочая система	BaldursGame – Класс игры BaldursGame	__init__(self, x, y, z) Инициализирует все поля и методы внутри конкретной игры.
main	рабочая система	Main – Класс Main	методы отсутствуют

4.2 Модульное тестирование разработанного приложения

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

4.3 Системное тестирование разработанного приложения

На рисунке 4.2 представлен пример работы программы.

Рисунок 4.2 – Пример работы программы с одним персонажем внутри одной, игровой зоны Village

На рисунке 4.3 представлен пример анимации персонажа.

Рисунок 4.3 – Анимация передвижения персонажа mage

На рисунке 4.4 представлен пример движения персонажа.

Рисунок 4.4 – Передвижение персонажа mage

На рисунке 4.5 представлен пример невозможности выхода за границу зоны.

Рисунок 4.5 – Персонаж mage, не может выйти за пределы видимой зоны Village

На рисунке 4.6 представлен пример перехода персонажа из зоны.

Рисунок 4.6 – Персонаж mage, переходит из зоны Village в зону Ruins

На рисунке 4.7 представлен пример установки новой зоны.

Рисунок 4.7 – Пример работы программы с тремя персонажами внутри одной, игровой зоны Ruins

На рисунке 4.8 представлен пример работы сценария движения персонажа.

Рисунок 4.8 – Пример работы сценария walk(50, 50, self.footman), игровой зоны Ruins

На рисунке 4.9 представлен пример работы сценария поведения персонажа противника.

Рисунок 4.9 – Пример работы сценария ai(self.grunt), игровой зоны Ruins

На рисунке 4.10 представлен пример работы метода click персонажа.

Рисунок 4.10 – 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, 2001. – 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.

```
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))

  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'''
```

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 15
				Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-02б	

Рисунок А.1 – Сведения о ВКРБ

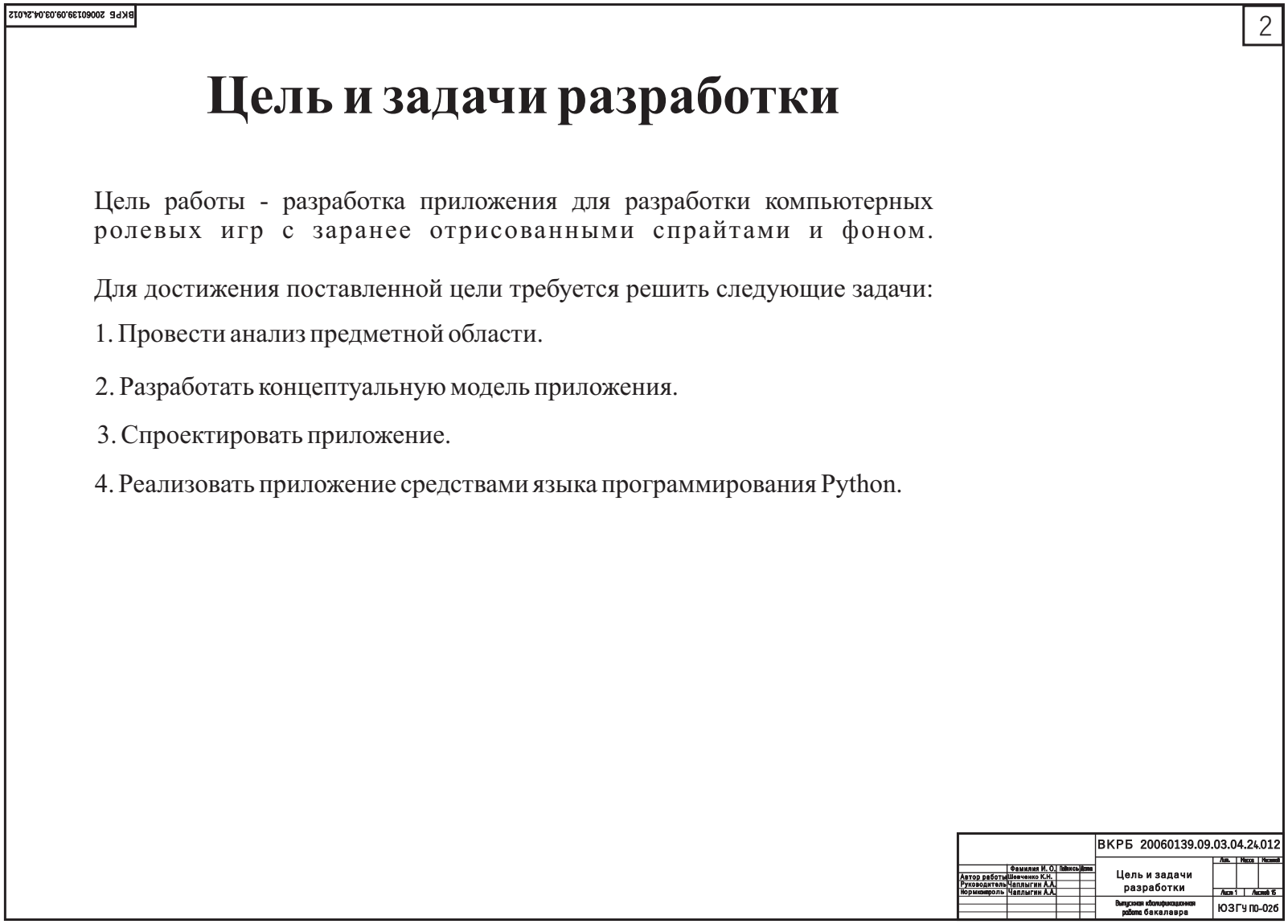
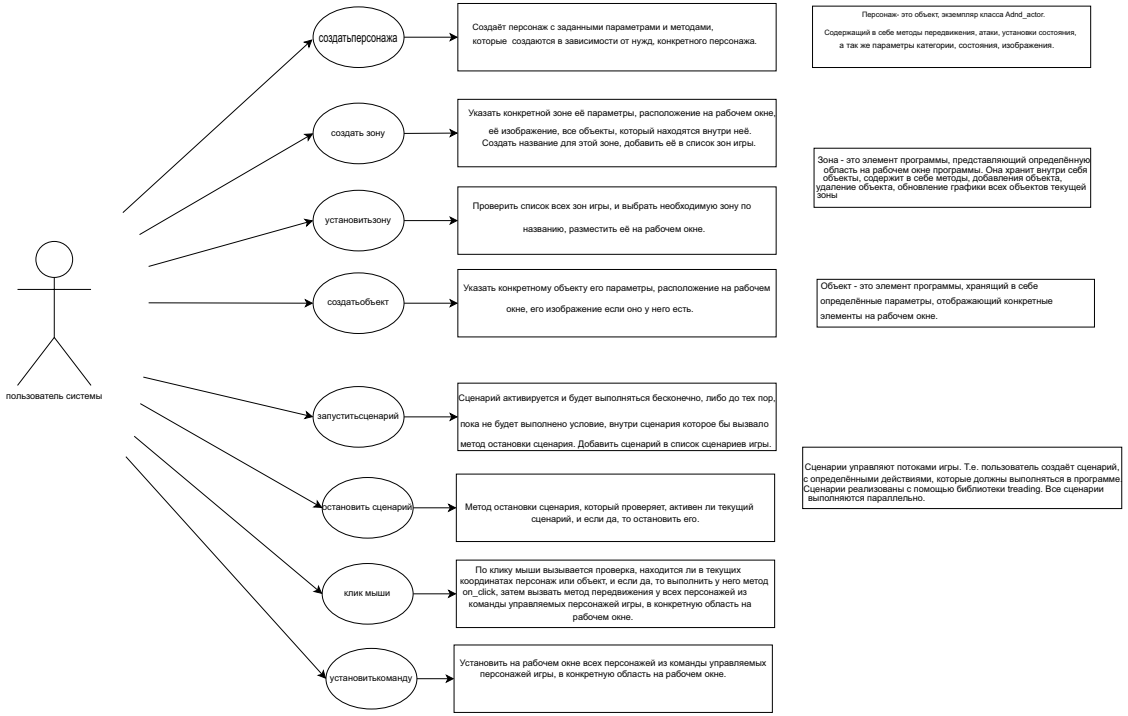


Рисунок А.2 – Цель и задачи разработки

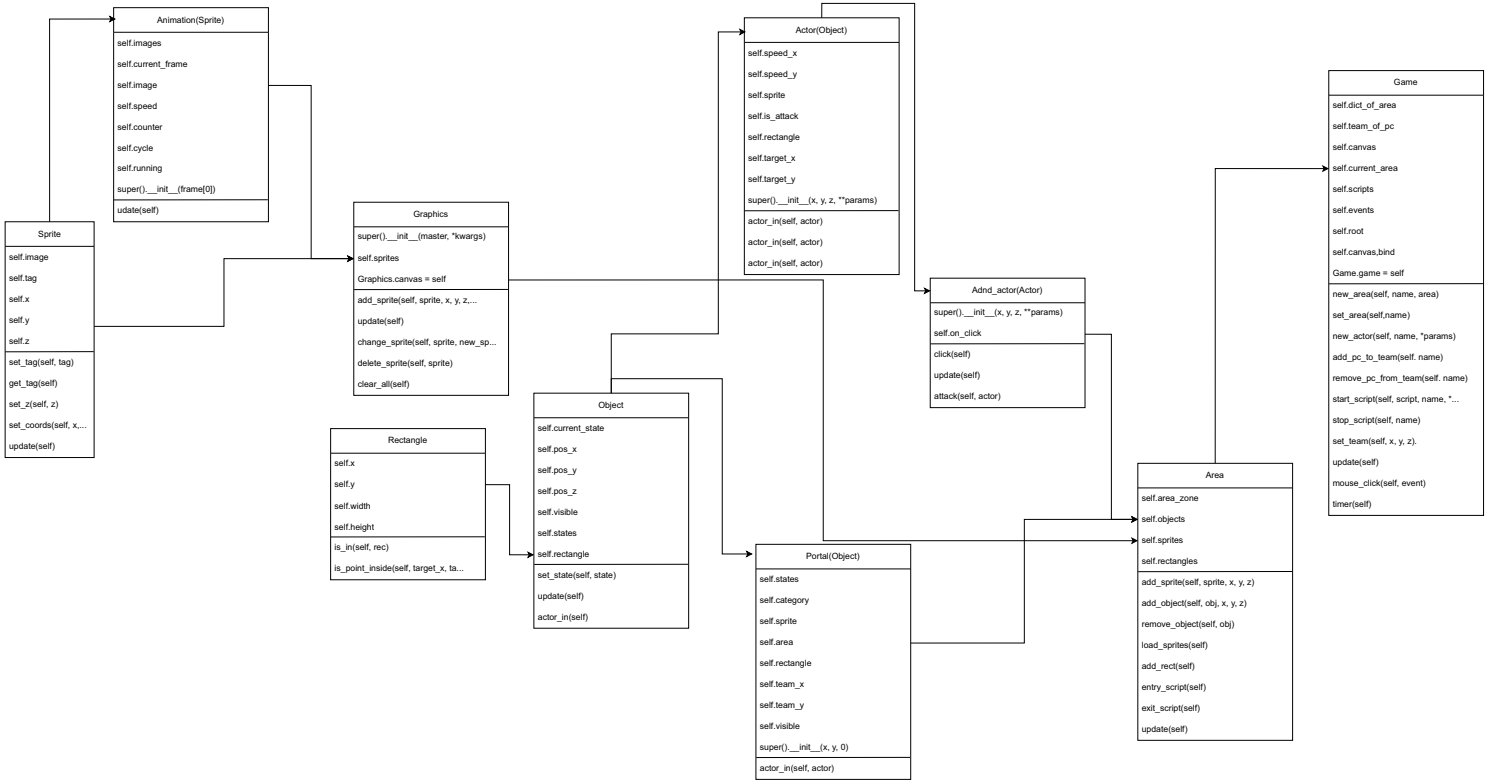
Концептуальная модель



				ВКРБ 20060139.09.03.04.24.012		
	Фамилия И. О.	Подпись	Дата	Концептуальная модель	Лит.	Масса	Масштаб
Автор работы	Шевченко К.Н.						
Руководитель	Чаплыгин А.А.				Лист 1	Листов 15
Нормоконтроль	Чаплыгин А.А.			Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-026	

Рисунок А.3 – Концептуальная модель приложения

Диаграмма классов программы



				ВКРБ 20060139.09.03.04.24.012		
	Фамилия И. О.	Подпись	Дата	Диаграмма классов программы	Лит.	Масса / Масштаб
Автор работы	Шевченко К.Н.					
Руководитель	Чаплыгин А.А.				Лист 1	Листов 15
Нормоконтроль	Чаплыгин А.А.			Выпускная квалификационная работа бакалавра	ЮЗГУ ПО-026	

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

main.tex

```
\input{setup.tex}

% Режим шаблона (должен быть включен один из трех)
\ВКРtrue
%\Практикаtrue
%\Курсоваятrue

\newcommand{\Дисциплина}{<<Проектирование и архитектура программных систем>>} % для курсовой
\newcommand{\КодСпециальности}{09.03.04} % Курсовая
\newcommand{\Специальность}{Программная инженерия} % Курсовая
\newcommand{\Тема}{Платформа для создания компьютерных изометрических ролевых игр} % ВКР Курсовая
\newcommand{\ТемаВтораяСтрока}{с заранее отрисованным двухмерным фоном и спрайтовыми персонажами}
\newcommand{\ГдеПроводитсяПрактика}{Юго-Западном государственном университете} % для практики
\newcommand{\РуководительПрактПредпр}{} % для практики
\newcommand{\ДолжнРуководительПрактПредпр}{директор} % для практики
\newcommand{\РуководительПрактУнивер}{Чаплыгин А. А.} % для практики
\newcommand{\ДолжнРуководительПрактУнивер}{к.т.н. доцент} % для практики
\newcommand{\Автор}{К. Н. Шевченко}
\newcommand{\АвторРод}{Шевченко К.Н.}
\newcommand{\АвторПолностьюРод}{Шевченко Клима Николаевича} % для практики
\newcommand{\Шифр}{20-06-0139}
\newcommand{\Курс}{4} % для практики
\newcommand{\Группа}{ПО-026}
\newcommand{\Руководитель}{А. А. Чаплыгин} % для ВКР и курсовой
\newcommand{\Нормоконтроль}{А. А. Чаплыгин} % для ВКР
\newcommand{\ЗавКаф}{А. В. Малышев} % для ВКР
\newcommand{\ДатаПриказа}{«04» апреля 2024~г.} % для ВКР
\newcommand{\НомерПриказа}{1616-с} % для ВКР
\newcommand{\СрокПредоставления}{«11» июня 2024~г.} % для ВКР, курсового

\begin{document}
\maketitle
\ifПрактика{}\else{
    \input{ЛистЗадания}
    \input{Реферат}}\fi
\tableofcontents
\input{Обозначения}
\ifПрактика{}\else{\input{Введение}}\fi
\input{Анализ}
\input{ТехЗадание}
\input{ТехПроект}
\ifПрактика{}\else{
    \input{РабочийПроект}
    \input{Заключение}
}\fi
```

```
\input{СписокИсточников}
\ifВКР{\input{Плакаты}}\fi
\ifПрактика{}\else{\input{Код}}\fi
\end{document}
```

ТехПроект.tex

```
\section{Технический проект}
\subsection{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.

\subsection{Обоснование выбора технологии проектирования}

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

\subsubsection{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.

\subsubsection{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 интерпретируемый 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++.

\subsubsection{Использование библиотеки Tkinter и реализация таймеров на
    Python}

\paragraph{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.

```

```
\paragraph{Возможности Tkinter}
Вот некоторые из основных возможностей, предоставляемых библиотекой Tkinter:

\begin{itemize}
  \item Создание различных виджетов: кнопки, метки, поля ввода, списки и
      многое другое.
  \item Управление компоновкой виджетов с использованием менеджеров
      компоновки (например, grid, pack, place).
  \item Обработка событий, таких как щелчок мыши, нажатие клавиш и другие.
  \item Возможность создания различных диалоговых окон, таких как окна
      предупреждений, информационные окна и окна запроса.
  \item Поддержка многопоточности для обновления интерфейса из различных
      потоков выполнения.
\end{itemize}

\paragraph{Реализация таймеров на Python}
Для реализации таймеров на Python можно использовать модуль \texttt{time} или
    \texttt{threading}. Вот пример использования модуля \texttt{time} для
    создания простого таймера:

  import time

  def countdown(t):
  while t > 0:
  mins, secs = divmod(t, 60)
  timeformat = '{:02d}:{:02d}'.format(mins, secs)
  print(timeformat, end='\r')
  time.sleep(1)
  t -= 1
  print('Таймер завершен!')


  t = 10
  countdown(t)

Этот код создает простой обратный отсчет таймера с использованием функции \
    texttt{countdown}. Он выводит оставшееся время в формате MM:CC и уменьшает
    его на 1 каждую секунду, используя функцию \texttt{time.sleep(1)}. Когда
    время истекает, выводится сообщение о завершении таймера.

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

\subsection{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 может 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 y 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):

```
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):
```

```
'''
Класс игровой зоны Ruins

'''
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):
'''
Сценарий для движения бугая

:param step\_x: шаг движения x
:param step\_y: шаг движения y
'''
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)

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

```
141
142 class BaldursGame(Game):
143 def \_\_init\_\_(self, canvas, window, **params):
144 '''
145 Класс конкретной игры для демонстрации
146
147 :param canvas: класс графической системы
148 :param window: окно на которое будет выводится игра
149 '''
150 super().\_\_init\_\_(canvas, window, **params)
151 from mage import Mage
152 self.add\_pc\_to\_team(Mage(0, 0, 0))
153 self.new\_area('Ruins', Ruins())
154 self.set\_area('Ruins')
155 self.set\_team(500, 300, 100)
156 self.timer()
157
158 \subsubsection{Пример клиентского кода игры}
159 \paragraph{Создание классов персонажей/предметов}
160 Клиент создает модуль содержащий методы модуля RPGGame, например
      BaldursGateGame. В этом модуле клиент создаем мир игры, с помощью new\_
      actor.
161
162 модуль bggame:
163 from ruins import *
164 import time
165 import random
166
167 class BaldursGame(Game):
168 def \_\_init\_\_(self, canvas, window, **params):
169 '''
170 Класс конкретной игры для демонстрации
171
172 :param canvas: класс графической системы
173 :param window: окно на которое будет выводится игра
174 '''
175 super().\_\_init\_\_(canvas, window, **params)
176 from mage import Mage
177 self.add\_pc\_to\_team(Mage(0, 0, 0))
178 self.new\_area('Ruins', Ruins())
179 self.set\_area('Ruins')
180 self.set\_team(500, 300, 100)
181 self.timer()
182
183 \paragraph{Задание правил атаки}
184 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.
185 Пример:
186
187 модуль adnd\_actor:
188 from math import sqrt
189 from rpg.actor import Actor
190 from rpg.animation import Animation
```

```
191 import rpg.game
192 import time
193
194 class Adnd\_actor(Actor):
195
196 ATTACK\_RANGE = 50
197
198 def \_\_init\_\_(self, x, y, z, **params):
199 '''
200 Класс Adnd\_actor содержащий основные механики взаимодействия с другими
        персонажами
201
202 :param x: координата x
203 :param y: координата y
204 :param z: координата z
205 '''
206 super().\_\_init\_\_(x, y, z, **params)
207 self.on\_click = self.click
208
209 def click(self):
210 '''
211 вызывается при клике на персонажа
212
213 '''
214 pc = rpg.game.Game.game.team\_of\_pc[0]
215 if pc == self:
216 return
217 dx = pc.pos\_x - self.pos\_x
218 dy = pc.pos\_y - self.pos\_y
219 dist = sqrt(dx * dx + dy * dy)
220 if dist <= self.ATTACK\_RANGE:
221 pc.is\_attack = True
222 pc.attack(self)
223 time.sleep(0.125)
224 if self.hp <=0:
225 pc.is\_attack = False
226
227 def attack(self, actor):
228 '''
229 совершает атаку по actor
230
231 :param actor: персонаж, которого атакуют
232 '''
233 actor.hp -= self.damage
234 def update(self):
235 '''
236 обновляет состояние персонажа
237
238 '''
239 super().update()
240 if self.hp <= 0:
241 self.stop\_move()
242 self.set\_state('death')
243
```

```
244
245 \paragraph{Создание зон, заполнение их персонажами/объектами}
246 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.
      Добавляем 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.
247
248 import random
249 from math import sqrt
250 import time
251 from rpg.area import *
252 from rpg.sprite import *
253 from rpg.rectangle import *
254 from rpg.game import Game
255 from rpg.portal import Portal
256
257 class Ruins(Area):
258 def \_\_init\_\_(self):
259 '''
260 Класс игровой зоны Ruins
261
262 '''
263 super().\_\_init\_\_()
264 self.add\_sprite(Sprite('images/fon3.png'), 590, 400, 0)
265 self.add\_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.
      width(), height=Sprite('images/fon3.png').image.height()))
266 from grunt import Grunt
267 self.grunt = Grunt(0,0,0)
268 from footman import Footman
269 self.footman = Footman(0,0,0)
270 self.add\_object(self.footman, 120, 120, 1)
271 self.add\_object(self.grunt, 500, 185, 1)
272 p = Portal(400, 400, 200, 200, 'Village', 480, 100)
273 self.add\_object(p, p.pos\_x, p.pos\_y, 100)
274 Game.game.start\_script(self.ai, "ai", self.grunt)
275 Game.game.start\_script(self.walk\_two, "footman", 50, 50)
276
277
278 def walk(self, step\_x, step\_y, actor):
279 '''
280 Сценарий для движения бугая
281
282 :param step\_x: шаг движения x
283 :param step\_y: шаг движения y
284 '''
```

```
285 if actor.hp <= 0:
286 Game.game.stop\_script("grunt")
287 new\_x = 200
288 new\_y = 200
289 actor.is\_attack = False
290 direction = random.choice(["up", "down", "left", "right"])
291 if direction == "up":
292 new\_y -= step\_y
293 new\_x = step\_x
294 elif direction == "down":
295 new\_y += step\_y
296 new\_x = step\_x
297 elif direction == "left":
298 new\_y = step\_y
299 new\_x -= step\_x
300 elif direction == "right":
301 new\_y = step\_y
302 new\_x += step\_x
303
304 actor.search\_position(new\_x, new\_y)
305
306 time.sleep(2)
307
308 модуль bggame:
309 from ruins import *
310 import time
311 import random
312
313 \paragraph{Пример сценариев: переход между зонами}
314 Глобальные сценарии находятся в классе игры (BGGame), мы подключаем их как :
315 Area.set\_enter\_script(script)
316 Как происходит переход команды между зонами.
317 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.
318
319 from rpg.object import Object
320 from rpg.game import Game
321 from rpg.rectangle import Rectangle
322
323 class Portal(Object):
324 def \_\_init\_\_(self, x, y, width, height, area, team\_x, team\_y):
325 '''
326 Создает портал в новую зону
327
328 :param x: координата x портала
329 :param y: координата y портала
330 :param width: ширина портала
331 :param height: высота портала
332 :param area: имя зоны куда будет переход
333 :param team\_x: местоположение команды в новой зоне
334 :param team\_y: местоположение команды в новой зоне
335 '''
336 self.states = None
```

```
337 self.sprite = None
338 self.category = 'portal'
339 super().\_\_init\_\_(x, y, 0)
340 self.rectangle = Rectangle(x, y, width, height)
341 self.area = area
342 self.team\_x = team\_x
343 self.team\_y = team\_y
344 self.visible = False
345
346 def actor\_in(self, actor):
347 '''
348 Проверяет находится ли персонаж внутри портала
349
350 :param actor: проверяемый персонаж
351 '''
352 if actor.category == "pc":
353 Game.game.set\_area(self.area)
354 Game.game.set\_team(self.team\_x, self.team\_y, 100)
355 actor.stop\_move()
356
357 модуль ruins
358 import random
359 from math import sqrt
360 import time
361 from rpg.area import *
362 from rpg.sprite import *
363 from rpg.rectangle import *
364 from rpg.game import Game
365 from rpg.portal import Portal
366
367 class Ruins(Area):
368 def \_\_init\_\_(self):
369 '''
370 Класс игровой зоны Ruins
371
372 '''
373 super().\_\_init\_\_()
374 self.add\_sprite(Sprite('images/fon3.png'), 590, 400, 0)
375 self.add\_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.
      width(), height=Sprite('images/fon3.png').image.height()))
376 from grunt import Grunt
377 self.grunt = Grunt(0,0,0)
378 from footman import Footman
379 self.footman = Footman(0,0,0)
380 self.add\_object(self.footman, 120, 120, 1)
381 self.add\_object(self.grunt, 500, 185, 1)
382 p = Portal(400, 400, 200, 200, 'Village', 480, 100)
383
384 \paragraph{Как будет идти бой}
385 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:

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):
'''
Класс Adnd\_actor содержащий основные механики взаимодействия с другими персонажами

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

def click(self):
'''
вызывается при клике на персонажа

'''
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):
'''
совершает атаку по actor

```

```
:param actor: персонаж, которого атакуют
'''
actor.hp -= self.damage
def update(self):
'''
обновляет состояние персонажа

'''
super().update()
if self.hp <= 0:
self.stop\_move()
self.set\_state('death')

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):
'''
Класс игровой зоны Ruins

'''
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):
'''
Сценарий для движения бугая

:param step\_x: шаг движения x
:param step\_y: шаг движения y
'''
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)

def ai(self, actor):
'''
скрипт противников

:param step\_x: размер шага x до персонажа игрока
:param step\_y: размер шага x до персонажа игрока
:param actor: персонаж противник
'''
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")
Game.game.start\_script(self.walk, "grunt", 50, 50, actor)

else:
actor.is\_attack = False
time.sleep(2)

\paragraph{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)

```
537
538 модуль sprite
539
540 import tkinter as tk
541 class Sprite:
542
543 def \_\_init\_\_(self, image):
544 '''
545 Класс спрайта для работы с изображениями на Canvas
546
547 :param image: адресс изображения который
548 '''
549 self.image = tk.PhotoImage(file=image)
550 self.tag = None
551 self.x = 0
552 self.y = 0
553 self.z = 0
554
555 def set\_tag(self, tag):
556 '''
557 Устанавливает тег спрайта
558
559 :param tag: тег спрайта
560 '''
561 self.tag = tag
562
563 def set\_z(self, z):
564 '''
565 Устанавливает z-координату спрайта
566
567 :param z: координата z
568 '''
569 self.z = z
570
571 def get\_tag(self):
572 '''
573 Возвращает тег спрайта
574
575 '''
576 return self.tag
577
578 def set\_coords(self, new\_x, new\_y):
579 '''
580 Обновляет координаты спрайта
581
582 :param new\_x: координата x
583 :param new\_y: координата y
584 '''
585 if self.tag:
586 self.x = new\_x
587 self.y = new\_y
```

```
def update(self):
'''
Обновляет анимацию спрайта

'''
pass

модуль graphics

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 = []
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)
sprite.x = x
sprite.y = y
self.sprites.append(sprite)
self.sprites.sort(key=lambda sprite: sprite.z)

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

'''
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):
'''
```

```
642 Меняет  спрайт на новый.
643
644 :param sprite: экземпляр спрайта
645 :param new\_sprite: новый спрайт
646 '''
647 old\_sprite\_pos = None
648 for i, s in enumerate(self.sprites):
649 if s.get\_tag() == sprite.get\_tag():
650 old\_sprite\_pos = i
651 break
652
653 if old\_sprite\_pos is not None:
654 old\_tag = sprite.get\_tag()
655
656 self.sprites[old\_sprite\_pos] = new\_sprite
657 new\_sprite.set\_tag(old\_tag)
658
659 new\_sprite.set\_tag(old\_tag)
660 new\_sprite.set\_z(sprite.z)
661
662 self.tag\_raise(old\_tag)
663 self.coords(old\_tag, sprite.x, sprite.y)
664 self.itemconfig(old\_tag, image=new\_sprite.image)
665
666 def delete\_sprite(self, sprite):
667 '''
668 Удаляет спрайт с Canvas.
669
670 :param sprite: экземпляр спрайта
671 :return:
672 '''
673 self.delete(sprite.get\_tag())
674 self.sprites.remove(sprite)
675
676 def clear\_all(self):
677 '''
678 Удаляет все спрайты с Canvas
679
680 '''
681 for sprite in self.sprites:
682 self.delete(sprite.get\_tag())
683 self.sprites.clear()
684
685 модуль baldursgame '''пользовательский модуль'''
686
687 from ruins import *
688 from village import *
689 import time
690 import random
691
692 class BaldursGame(Game):
693 def \_\_init\_\_(self, canvas, window, **params):
694 '''
695 Класс конкретной игры для демонстрации
```

```
696
697 :param canvas: класс графической системы
698 :param window: окно на которое будет выводится игра
699 '''
700 super().\_\_init\_\_(canvas, window, **params)
701 from mage import Mage
702 self.add\_pc\_to\_team(Mage(0, 0, 0))
703 self.new\_area('Ruins', Ruins())
704 self.new\_area('Village', Village())
705 self.set\_area('Ruins')
706 self.set\_team(500, 300, 100)
707 self.timer()
708
709 модуль main
710 from bggame import *
711
712 root = tk.Tk()
713 root.geometry('1500x1500')
714
715 exit\_button = tk.Button(root, text="Exit", fg="red", command=root.destroy)
716 canvas = Graphics(root, width=1500, height=1500)
717 Graphics.canvas = canvas
718
719 BaldursGame(canvas, root)
720
721 canvas.place(height = 1500, width =1500)
722 BaldursGame.timer
723
724 root.mainloop()
725
726 \subsection{Архитектура платформы для создания ролевых игр}
727 \subsubsection{Диаграмма компонентов классов}
728 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. Основными
    графическими элементами диаграммы компонентов являются компоненты,
    интерфейсы, а также зависимости между ними. На рисунке \ref{diagram.eps:
    image} 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).
729 \begin{figure}[ht]
730   \center{\includegraphics[width=1\linewidth]{diagram}}
731   \caption{Диаграмма компонентов}
732   \label{diagram:image}
733 \end{figure}
734 \paragraph{Описание классов}
735 \begin{enumerate}
736   \item[Graphics] - класс, управляющий спрайтами. Содержит в себе:
737   \begin{itemize}
738     \item self.sprites - список спрайтов;
739     \item add\_sprite(self, sprite x, y, z, image) - добавляет в список
        спрайт, сохраняет координаты;
```

```
740     \item change\_sprite(self, sprite new\_sprite) - меняет местами спрайты в
            списке.
741     \item delete\_sprite(self, sprite) - удаляет спрайт из списка.
742     \item clear\_all(self) - очищает список спрайтов.
743     \item update(self) - добавляет все спрайты из списка на форму.
744   \end{itemize}
745   \item[Sprite] - класс, хранящий в себе изображение игровых объектов image.
746   \begin{itemize}
747     \item self.x - координата x.
748     \item self.y - координата y.
749     \item self.z - координата z.
750     \item self.tag - уникальный номер спрайта.
751     \item self.image - изображение.
752     \item set\_tag(self, tag) - устанавливает tag спрайту.
753     \item get\_tag(self) - возвращает tag спрайта.
754     \item set\_z(self, z) - устанавливает z координату.
755     \item set\_coords(self, new\_x, new\_y) - устанавливает новые координаты.
756     \item update(self) - ничего не делает.
757   \end{itemize}
758   \item[Animation] - класс, хранящий в себе список изображений игровых
        объектов image. Потомок класса Sprite.
759   \begin{itemize}
760     \item self.current\_frame - текущий кадр.
761     \item self.images  - Загрузка всех кадров анимации.
762     \item self.image - Установка начального изображения.
763     \item self.speed - скорость анимации.
764     \item self.counter - счётчик кадров.
765     \item self.cycle - проверка на то что должна ли быть анимация циклично
            или нет.
766     \item self.running - проверка проигрывается ли сейчас анимация.
767     \item update(self) - обновляет кадр в анимации.
768   \end{itemize}
769   \item[Rectangle] - абстрактный класс прямоугольника.
770   \begin{itemize}
771     \item self.pos\_x - координата x.
772     \item self.pos\_y - координата y.
773     \item self.width - ширина.
774     \item self.height - высота.
775     \item is\_in(self, rect) - функция проверки нахождения одного
            прямоугольника в другом.
776     \item is\_point\_inside(self, target\_x, target\_y) - функция проверки
            точки в пределах прямоугольника.
777   \end{itemize}
778   \item[Object] - класс, от которого наследуются классы Adnd\_Actor, Actor,
        Portal.
779   \begin{itemize}
780     \item self.pos\_x - координата x.
781     \item self.pos\_y - координата y.
782     \item self.pos\_z - координата z.
783     \item self.current\_state - текущее состояние.
784     \item self.visible - видимость портала.
785     \item self.on\_click - функция клика по объекту.
786     \item self.rectangle - прямоугольник объекта.
787     \item set\_state(self, state\_name) -устанавливает состояние.
```

```
788   \item actor\_in(self, actor) - ничего не делает.
789   \item update(self) - ничего не делает.
790 \end{itemize}
791 \item[Portal] - класс объекта для перехода между зонами. Потомок класса
      Object.
792 \begin{itemize}
793   \item self.states - состояние портала.
794   \item self.sprite - спрайт портала.
795   \item self.category - категория.
796   \item self.rectangle = - прямоугольник портала.
797   \item self.area - зона в которую ведёт портал.
798   \item self.team\_x - координата x в  которую нужно разместить команду.
799   \item self.team\_y - координата y в  которую нужно разместить команду.
800   \item self.visible - видимость портала.
801   \item actor\_in(actor) - события, которые произойдут, когда персонаж
        окажется внутри прямоугольника портала.
802 \end{itemize}
803 \item[Actor] - класс персонажа, содержащий внутри себя основные поля и
      методы для перемещения по рабочему окну.
804 \begin{itemize}
805   \item self.sprite - спрайт персонажа.
806   \item self.speed\_x - значение скорости x.
807   \item self.speed\_y - значение скорости y.
808   \item self.target\_x - координата x в которую должен прийти персонаж.
809   \item self.target\_y - координата y в которую должен прийти персонаж.
810   \item self.rectangle - прямоугольник персонажа.
811   \item self.is\_attack - атакует ли сейчас персонаж.
812   \item update(self) - функция обновления координат и состояния персонажа.
813   \item search\_position(self, new\_x, new\_y) - поиск координат в которые
        нужно двигаться персонажу.
814   \item stop\_move(self) - остановка движения персонажа.
815 \end{itemize}
816 \item[Adnd\_Actor] - класс персонажа, содержащий методы связанные с
      взаимодействием с другими персонажами. Является наследником Actor.
817 \begin{itemize}
818   \item self.on\_click событие при клике на персонажа
819   \item update(self) - функция обновления координат и состояния персонажа.
820   \item click(self) - функция вызывается при клике по персонажу.
821   \item attack(self, actor) -  функция атаки персонажа по другому персонажу
        .
822 \end{itemize}
823 \item[Area] - зона, в которой находятся персонажи и объекты. Содержит
      следующие поля и методы:
824 \begin{itemize}
825   \item self.area\_zone - параметр определяющий особенности конкретной зоны
        .
826   \item self.objects - список, хранящий в себе множество объектов.
827   \item self.sprites - список фоновых спрайтов.
828   \item self.rectangles - прямоугольник зоны.
829   \item add\_sprite(self, sprite, x, y, z) - функция добавляет спрайт в
        зону.
830   \item add\_object(self, obj, x, y, z) - функция добавляет объект в зону.
831   \item remove\_object(self, obj) - функция удаляет объект из зоны.
832   \item load\_sprites(self) - функция загружает все спрайты зоны.
```

```
833     \item add\_rect(self, rec) - функция добавляет прямоугольник в зону.
834     \item entry\_script(self) - функция запускается, когда команда входит в
            зону.
835     \item exit\_script(self) - функция запускается, когда команда выходит из
            зоны.
836     \item update(self) - функция изменяет и проверяет изменение всех объектов
             в зоне.
837   \end{itemize}
838   \item[Game] - абстрактный класс, управляющий игрой. Имеет следующие поля и
        методы:
839   \begin{itemize}
840     \item self.rpg\_dict\_of\_area - словарь, хранящий в себе множество
            экземпляров класса Area.
841     \item self.team\_of\_pc - список, хранящий в себе имена экземпляров
            класса Actor с параметром category = "pc".
842     \item self.canvas - графика.
843     \item self.root - окно для графики.
844     \item self.current\_area - параметр хранящий, текущую зону.
845     \item self.scripts - словарь для хранения запущенных сценариев.
846     \item self.events - словарь для хранения запущенных event`ов сценариев.
847     \item self.canvas.bind("<Button-1>", self.mouse\_left.\_click) -
            обработка клика мыши по рабочему окну.
848     \item new\_area(self, name, area) - функция добавляет новую зону в список
            .
849     \item set\_area(self, name) - функция устанавливает текущую зону,
            загружает графику зоны.
850     \item new\_actor(self, name, **params) - функция создаёт класс, потомок
            от Actor и создаёт поле из параметров, и установление их в начальные
            значения.
851     \item add\_pc\_to\_team(self, pc) - функция добавляет персонажа в команду
            .
852     \item remove\_pc\_from\_team(self, pc) - функция удаляет персонажа из
            команды.
853     \item start\_script(self, script\_function, script\_name, *args) -
            функция запускает сценарий в отдельном потоке с возможностью остановки
             и передачи аргументов.
854     \item stop\_script(self, script\_name) - функция останавливает сценарий
            по имени.
855     \item set\_team(self, x, y, z) - функция устанавливает координаты
            персонажей команды.
856     \item update(self) - функция вызывается в таймере для обновления всех
            переменных в текущей зоне.
857     \item mouse\_left\_click(self, event) - функция обрабатывает клик мыши.
858     \item timer(self) - функция должна вызывать метод update постоянно.
859   \end{itemize}
860 \end{enumerate}
861
862 \subsubsection{Реализация графической подсистемы}
863 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.
864 \paragraph{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:

```
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)
self.sprites.append(sprite)
self.sprites.sort(key=lambda sprite: sprite.z)
\subsubsection{Реализация зон}
```

Зоны в программе представляют собой различные игровые области или уровни. Каждая зона реализована через класс Area, который содержит спрайты и объекты, принадлежащие этой зоне. Зоны могут содержать свои собственные скрипты для входа и выхода из зоны (entry_script и exit_script), а также метод update, который обновляет состояние всех объектов в зоне.

```
\subsubsection{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.

```
\subsubsection{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.

```
\subsubsection{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:

```
def is\_point\_inside(self, target\_x, target\_y):
return (self.x <= target\_x <= self.x + self.width) and \
(self.y <= target\_y <= self.y + self.height)
```

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).

Место для диска