

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

(подпись, дата)

К. Н. Шевченко

(инициалы, фамилия)

Группа ПО-026

Руководитель ВКР

(подпись, дата)

А. А. Чаплыгин

(инициалы, фамилия)

Нормоконтроль

(подпись, дата)

А. А. Чаплыгин

(инициалы, фамилия)

ВКР допущена к защите:

Заведующий кафедрой

(подпись, дата)

А. В. Малышев

(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

(подпись, дата)

А. А. Чаплыгин

(инициалы, фамилия)

Задание принял к исполнению

(подпись, дата)

К. Н. Шевченко

(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 100 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 100 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

ВВЕДЕНИЕ	13
1 Анализ предметной области	15
1.1 История первых игр что стали прародителями RPG-жанра	15
1.2 Первые популярные RPG-игры	16
1.3 Япония и её JRPG	18
1.4 Популярные RPG студии SSI	19
2 Техническое задание	21
2.1 Основание для разработки	21
2.2 Цель и назначение разработки	21
2.3 Требования пользователя к платформе	21
2.4 Пример игры	22
2.5 Особенности Dungeons and Dragons	23
2.6 Интерфейс пользователя	26
2.7 Моделирование вариантов использования	26
2.8 Требования к оформлению документации	27
3 Технический проект	28
3.1 Общая характеристика организации решения задачи	28
3.2 Обоснование выбора технологии проектирования	28
3.2.1 Описание используемых технологий и языков программирования	28
3.2.2 Язык программирования Python	28
3.2.3 Использование библиотеки Tkinter и реализация таймеров на Python	29
3.2.3.1 Введение	29
3.2.3.2 Возможности Tkinter	29
3.2.3.3 Реализация таймеров на Python	30
3.2.3.4 Заключение	30
3.3 Описание платформы для создания RPG игр	30
3.3.1 Пример клиентского кода игры	33

3.3.1.1	Создание классов персонажей/предметов	33
3.3.1.2	Задание правил атаки	33
3.3.1.3	Создание зон, заполнение их персонажами/объектами	34
3.3.1.4	Пример сценариев: переход между зонами	35
3.3.1.5	Как будет идти бой	36
3.3.1.6	Соединение движка и окон tkinter	38
3.4	Модули и классы	41
3.5	Диаграмма компонентов классов	41
3.6	Game	42
3.6.1	Описание модуля	42
3.6.1.1	Конструктор и поля модуля	42
3.6.2	Методы	42
3.6.2.1	New_area	42
3.6.2.2	Set_area	43
3.6.2.3	New_actor	43
3.6.2.4	Start_script	43
3.6.2.5	Stop_script	43
3.6.2.6	Add_pc_to_team	43
3.6.2.7	Remove_pc_from_team	43
3.6.2.8	Set_team	44
3.6.2.9	Update	44
3.6.2.10	Mouse_left_click	44
3.6.2.11	Timer	44
3.7	Area	44
3.7.1	Описание модуля	44
3.7.1.1	Конструктор и поля модуля	44
3.7.2	Методы	45
3.7.2.1	Add_sprite	45
3.7.2.2	Add_object	45
3.7.2.3	Remove_object	45
3.7.2.4	Load_sprites	45

3.7.2.5 Add_rect	45
3.7.2.6 Entry_script	45
3.7.2.7 Exit_script	45
3.7.2.8 Update	45
3.8 Sprite	46
3.8.1 Описание модуля	46
3.8.1.1 Конструктор и поля модуля	46
3.8.2 Методы	46
3.8.2.1 Set_tag	46
3.8.2.2 Set_z	46
3.8.2.3 Get_tag	46
3.8.2.4 Set_coords	47
3.8.2.5 Update	47
3.9 Graphics	47
3.9.1 Описание модуля	47
3.9.1.1 Конструктор и поля модуля	47
3.9.2 Методы	47
3.9.2.1 Add_sprite	47
3.9.2.2 Update	47
3.9.2.3 Change_sprite	47
3.9.2.4 Delete_sprite	48
3.9.2.5 Clear_all	48
3.10 Animation(Sprite)	48
3.10.1 Описание модуля	48
3.10.1.1 Конструктор и поля модуля	48
3.10.2 Методы	48
3.10.2.1 Update:	48
3.11 Object	49
3.11.1 Описание модуля	49
3.11.1.1 Конструктор и поля модуля	49
3.11.2 Методы	49

3.11.2.1 Set_state	49
3.11.2.2 Actor_in	49
3.11.2.3 Update	50
3.12 Actor(Object)	50
3.12.1 Описание модуля	50
3.12.1.1 Конструктор и поля модуля	50
3.12.2 Методы	50
3.12.2.1 update	50
3.12.2.2 Search_position	51
3.12.2.3 Stop_move	51
3.13 Adnd_actor(Actor)	51
3.13.1 Описание модуля	51
3.13.1.1 Конструктор и поля модуля	51
3.13.2 Методы	51
3.13.2.1 Click	51
3.13.2.2 Attack	51
3.13.2.3 Update	51
3.14 Rectangle	52
3.14.1 Описание модуля	52
3.14.1.1 Конструктор и поля модуля	52
3.14.2 Методы	52
3.14.2.1 Is_in	52
3.14.2.2 Is_point_inside	52
3.15 Portal(Object)	52
3.15.1 Описание модуля	52
3.15.1.1 Конструктор и поля модуля	52
3.15.2 Методы	53
3.15.2.1 Actor_in	53
4 Рабочий проект	54
4.1 Классы, используемые при разработке приложения	54
4.2 Модульное тестирование разработанного приложения	60

4.3 Системное тестирование разработанного приложения	60
ЗАКЛЮЧЕНИЕ	65
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	65
ПРИЛОЖЕНИЕ А Представление графического материала	68
ПРИЛОЖЕНИЕ Б Фрагменты исходного кода программы	77
На отдельных листах (CD-RW в прикрепленном конверте)	100
Сведения о ВКРБ (Графический материал / Сведения о ВКРБ.png)	Лист 1
Цель и задачи разработки (Графический материал / Цель и задачи разработки.png)	Лист 2
Концептуальная модель приложения (Графический материал / Концептуальная модель приложения.png)	Лист 3
Диаграмма классов (Графический материал / Диаграмма классов.png)	Лист 4
Модель работы сценариев (Графический материал / Модель работы сценариев.png)	Лист 5
Модульное тестирование платформы (Графический материал / Модульное тестирование платформы.png)	Лист 6
Заключение (Графический материал / Заключение.png)	Лист 7

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 13 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, большому 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 Популярныe 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 сопровождали 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 онлайн 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 оружие, 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, dwarфов, 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 dwarфа: dwarф получает +2 к силе и +2 к телосложению, его размер маленький, у dwarфа 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 количество 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 2.2 сформированы следующие действия пользователя и их последствия.

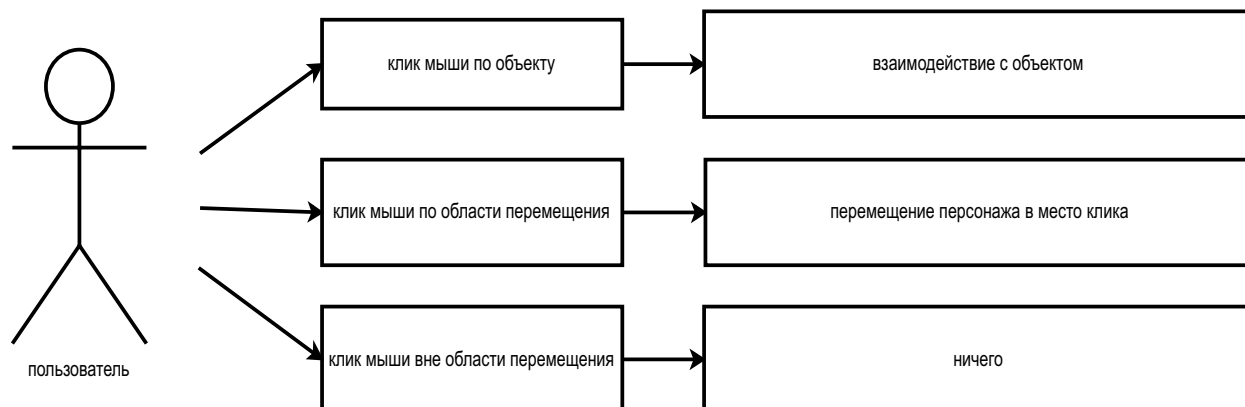


Рисунок 2.2 – Шаблон интерфейса игры

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 С или С++.

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 ММ:СС 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 `bgame`. 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 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
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 кото-
рое будет выводиться 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 кото-
рое будет выводиться 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 ос-
новные 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 куда и 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 помощью 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 ос-
        новные 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.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 y 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 конкретно-
му 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: ок-
но на которое будет выводиться 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 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).

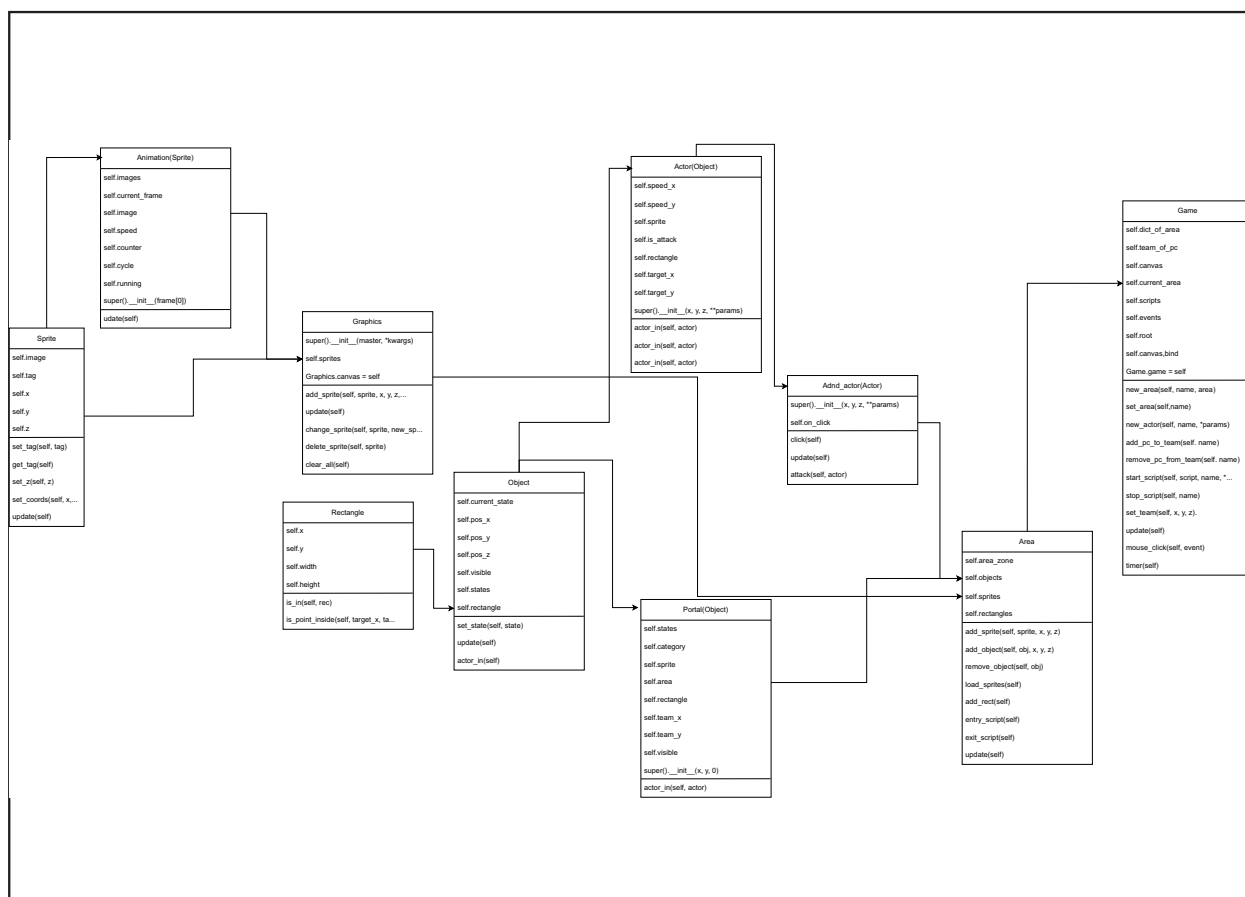


Рисунок 3.1 – Диаграмма компонентов

3.5 Модули и классы

3.6 Game

3.6.1 Описание модуля

3.6.1.1 Конструктор и поля модуля

```
def __init__(self, canvas, window, **params)
```

- self.rpg_dict_of_area =

- словарь, хранящий в себе множество экземпляров класса Area, number - ключ : name Area - значение

- self.team_of_pc = []

- список, хранящий в себе имена экземпляров класса Actor с параметром category = "pc"

- self.canvas = canvas

- графика

- self.root = window

- окно для графики

- self.current_area = None

- 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)

- обработчик клика

- Game.game = self

- экземпляр игры, для обращения к нему напрямую

3.6.2 Методы

3.6.2.1 New_area

`def new_area(self, name, area)` Описание метода: Добавляет новую зону в список.

3.6.2.2 Set_area

`def set_area(self, name)` Описание метода: Устанавливает текущую зону, загружает графику зоны.

3.6.2.3 New_actor

`def new_actor(self, name, **params)` Описание метода: метод отвечающий за создание класса, потомка от `Actor` и создание поля из параметров, и установление их в начальные значения.

3.6.2.4 Start_script

`def start_script(self, script_function, script_name, *args)` Описание метода: Запускает сценарий в отдельном потоке с возможностью остановки и передачи аргументов.

3.6.2.5 Stop_script

`def stop_script(self, script_name)` Описание метода: Останавливает сценарий по имени.

3.6.2.6 Add_pc_to_team

`def add_pc_to_team(self, pc)` Описание метода: метод отвечающий за добавление имени экземпляра класса `Actor` с параметром `category = "pc"` в список `team_of_pc`, хранящий имена всех игровых персонажей.

3.6.2.7 Remove_pc_from_team

`def remove_pc_from_team(self, pc)` Описание метода: метод отвечающий за удаление имени экземпляра класса Actor с параметром category = "pc" в список team_of_pc, хранящий имена всех игровых персонажей.

3.6.2.8 Set_team

`def set_team(self, x, y, z)` Описание метода: Устанавливает координаты персонажей команды.

3.6.2.9 Update

`def update(self)` Описание метода: Вызывается в таймере для обновления всех переменных в текущей зоне.

3.6.2.10 Mouse_left_click

`def mouse_left_click(self, event)` Описание метода: обрабатывает клик мыши.

3.6.2.11 Timer

`def timer(self)` Описание метода: Таймер должен вызывать метод update постоянно.

3.7 Area

3.7.1 Описание модуля

3.7.1.1 Конструктор и поля модуля

`def __init__(self, **params)`

- `self.area_zone = params`
- параметр определяющий особенности конкретной зоны
- `self.objects = []`
- список, хранящий в себе множество экземпляров классов Item

- self.sprites = []
- список фоновых спрайтов
- self.rectangles = None
- список, хранящий в себе множество прямоугольников

3.7.2 Методы

3.7.2.1 Add_sprite

def add_sprite(self, sprite, x, y, z) Описание метода: Добавляет спрайт в зону.

3.7.2.2 Add_object

def add_object(self, obj, x, y, z) Описание метода: Добавляет объект в зону.

3.7.2.3 Remove_object

def remove_object(self, obj) Описание метода: Удаляет объект из зоны.

3.7.2.4 Load_sprites

def load_sprites(self) Описание метода: Загружает все спрайты зоны.

3.7.2.5 Add_rect

def add_rect(self, rec) Описание метода: Добавляет прямоугольник в зону.

3.7.2.6 Entry_script

def entry_script(self) Описание метода: Запускается, когда команда входит в зону.

3.7.2.7 Exit_script

`def exit_script(self)` Описание метода: Запускается, когда команда выходит из зоны.

3.7.2.8 Update

`def update(self)` Описание метода: Изменяет и проверяет изменение всех персонажей в зоне.

3.8 Sprite

3.8.1 Описание модуля

3.8.1.1 Конструктор и поля модуля

`def __init__(self, image)`

- `self.spr_image = image`

- Описание параметра: параметр хранит изображение конкретного экземпляра класса `Sprite`.

- `self.tag = None`

- `self.spr_x = x`

- Описание параметра: параметр хранит числовое значение обозначающее расположение конкретного экземпляра класса `Sprite`.

- `self.spr_y = y`

- Описание параметра: параметр хранит числовое значение обозначающее расположение конкретного экземпляра класса `Sprite`.

- `self.spr_z = z`

- Описание параметра: параметр хранит числовое значение обозначающее расположение конкретного экземпляра класса `Sprite`.

3.8.2 Методы

3.8.2.1 Set_tag

`def set_tag(self, tag)`

3.8.2.2 Set_z

```
def set_z(self, z)
```

3.8.2.3 Get_tag

```
def get_tag(self)
```

3.8.2.4 Set_coords

```
def set_coords(self, new_x, new_y)
```

3.8.2.5 Update

```
def update(self)
```

3.9 Graphics

3.9.1 Описание модуля

3.9.1.1 Конструктор и поля модуля

```
def __init__(self, master, **kwargs)
```

```
– super().__init__(master, **kwargs)
```

```
– self.sprites = []
```

```
– список спрайтов
```

```
– Graphics.canvas = self
```

```
– параметр для работы с графикой, где к ней нужно обращаться напрямую
```

3.9.2 Методы

3.9.2.1 Add_sprite

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

3.9.2.2 Update

`def update(self)` Описание метода: Перерисовывает все спрайты

3.9.2.3 Change_sprite

`def change_sprite(self, sprite, new_sprite)` Описание метода: Меняет спрайт на новый

3.9.2.4 Delete_sprite

`def delete_sprite(self, sprite)` Описание метода: Удаляет спрайт с Canvas

3.9.2.5 Clear_all

`def clear_all(self)` Описание метода: Удаляет все спрайты с Canvas

3.10 Animation(Sprite)

3.10.1 Описание модуля

3.10.1.1 Конструктор и поля модуля

```
def __init__(self, frames, cycle=True)
– super().__init__(frames[0])
– self.images = frames
– Описание параметра: "список кадров"
– self.current_frame = 0
– self.images = [tk.PhotoImage(file=frame) for frame in frames]
– Загрузка всех кадров анимации
– self.image = self.images[0]
– Установка начального изображения
– self.speed = 3
– Описание параметра: "скорость анимации"
– self.counter = self.speed
– self.cycle = cycle
```


- self.running = True

3.10.2 Методы

3.10.2.1 Update:

def update(self): Меняет текущее изображение в списке изображений.

3.11 Object

3.11.1 Описание модуля

3.11.1.1 Конструктор и поля модуля

def __init__(self, x, y, z, **params)

- self.pos_x = x
- координата x
- self.pos_y = y
- координата y
- self.pos_z = z
- координата z
- self.current_state = None
- текущее состояние
- self.visible = True
- видим ли объект
- self.on_click = lambda x : x
- возможно ли кликнуть по объекту
- if self.states is not None: self.set_state(next(iter(self.states)))
- установка первого состояния если потребуется
- self.rectangle = Rectangle(x, y, 10, 10)
- прямоугольник объекта

3.11.2 Методы

3.11.2.1 Set_state

`def set_state(self, state_name):` Описание метода: меняет текущее состояние объекта

3.11.2.2 Actor_in

`def actor_in(self, actor):` Описание метода: Вызывается когда actor входит внутрь объекта

3.11.2.3 Update

`def update(self)` Описание метода: ничего не делает

3.12 Actor(Object)

3.12.1 Описание модуля

3.12.1.1 Конструктор и поля модуля

```
def __init__(self, x, y, z, **params)
– self.sprite = self.states[next(iter(self.states))]
– параметр хранящий спрайт
– super().__init__(x, y, z, **params)
– self.speed_x = 0
– значение скорости x
– self.speed_y = 0
– значение скорости y
– self.target_x = 0
– координата x в которую будет двигаться персонаж
– self.target_y = 0
– координата y в которую будет двигаться персонаж
– self.rectangle = Rectangle(self.pos_x, self.pos_y,
self.sprite.image.width(), self.sprite.image.height())
```

- прямоугольник персонажа
- self.is_attack = False
- состояние - атакует ли персонаж сейчас

3.12.2 Методы

3.12.2.1 update

def update(self) Описание метода: изменяет координаты и состояние персонажа.

3.12.2.2 Search_position

def search_position(self, new_x, new_y) Описание метода: Изменяет направление движения у персонажа.

3.12.2.3 Stop_move

def stop_move(self): Описание метода: Останавливает движение персонажа.

3.13 Adnd_actor(Actor)

3.13.1 Описание модуля

3.13.1.1 Конструктор и поля модуля

```
def __init__(self, x, y, z, **params)
– super().__init__(x, y, z, **params)
– self.on_click = self.click
– событие по клику на персонажа
```

3.13.2 Методы

3.13.2.1 Click

def click(self) Описание метода: вызывается при клике на персонажа.

3.13.2.2 Attack

`def attack(self, actor)` Описание метода: совершает атаку по actor.

3.13.2.3 Update

`def update(self)` Описание метода: обновляет состояние персонажа.

3.14 Rectangle

3.14.1 Описание модуля

3.14.1.1 Конструктор и поля модуля

`def __init__(self, x, y, width, height)`

- `self.x = x`
- координата x прямоугольника
- `self.y = y`
- координата y прямоугольника
- `self.width = width`
- ширина прямоугольника
- `self.height = height`
- высота прямоугольника

3.14.2 Методы

3.14.2.1 Is_in

`def is_in(self, rect)` Описание метода: Проверяет, входит ли прямоугольник `self` в прямоугольник `rect`

3.14.2.2 Is_point_inside

`def is_point_inside(self, target_x, target_y)` Описание метода: Проверяет, входит ли точка (x, y) в данный прямоугольник

3.15 Portal(Object)

3.15.1 Описание модуля

3.15.1.1 Конструктор и поля модуля

`def __init__(self, x, y, width, height, area, team_x, team_y)`

- `self.states = None`
- состояние портала
- `self.sprite = None`
- спрайт портала
- `self.category = 'portal'`
- категория портала
- `super().__init__(x, y, 0)`
- `self.rectangle = Rectangle(x, y, width, height)`
- прямоугольник портала
- `self.area = area`
- текущая зона
- `self.team_x = team_x`
- координата x в новой зоне в которую установят команду
- `self.team_y = team_y`
- координата y в новой зоне в которую установят команду
- `self.visible = False`
- видимость портала

3.15.2 Методы

3.15.2.1 Actor_in

`def actor_in(self, actor)` Описание метода: Проверяет находится ли персонаж внутри портала

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

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	<p>add_sprite(self, sprite, x, y, z) Добавляет спрайт в зону.</p> <p>add_object(self, obj, x, y, z) Добавляет объект в зону.</p> <p>remove_object(self, obj) Удаляет объект из зоны.</p> <p>load_sprites(self) Загружает все спрайты зоны.</p> <p>add_rect(self, rec) Добавляет прямоугольник в зону.</p> <p>entry_script(self) Запускается, когда команда входит в зону</p> <p>exit_script(self) Запускается, когда команда выходит из зоны</p> <p>update(self) Изменяет и проверяет изменение всех объектов в зоне.</p>

Продолжение таблицы 4.1

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

Продолжение таблицы 4.1

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

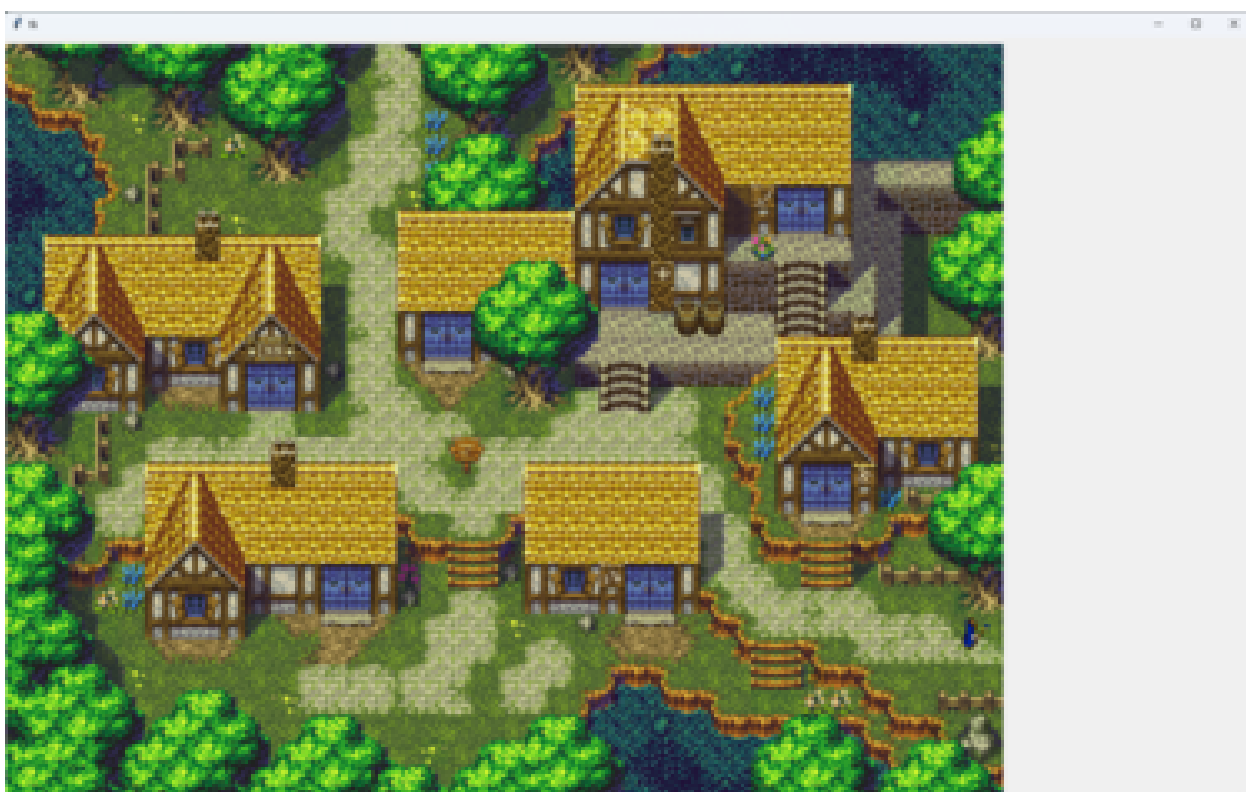


Рисунок 4.5 – Персонаж mage, не может выйти за пределы видимой зоны
Village

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.

```

1 import unittest
2 from rpg.rectangle import Rectangle
3 class TestRectangle(unittest.TestCase):
4     def setUp(self):
5         # Прямоугольник для использования в тестах
6         self.rect = Rectangle(1, 1, 4, 4)
7
8     def test_inside(self):
9         '''Тест: прямоугольник внутри другого'''
10        rect_outside = Rectangle(0, 0, 6, 6)
11        self.assertTrue(self.rect.is_in(rect_outside))
12
13    def test_outside(self):
14        '''Тест: прямоугольник снаружи другого'''
15        rect_inside = Rectangle(2, 2, 2, 2)
16        self.assertFalse(self.rect.is_in(rect_inside))
17
18    def test_apartside(self):
19        '''Тест: прямоугольник отдельно от другого'''
20        rect_apart = Rectangle(6, 6, 2, 2)
21        self.assertFalse(self.rect.is_in(rect_apart))
22
23    def test_touching_left(self):
24        '''Тест: прямоугольник касается слева'''
25        touching_left = Rectangle(0, 2, 1, 1)
26        self.assertFalse(self.rect.is_in(touching_left))
27
28    def test_touching_right(self):
29        '''Тест: прямоугольник касается справа'''
30        touching_right = Rectangle(5, 2, 1, 1)
31        self.assertFalse(self.rect.is_in(touching_right))
32
33    def test_touching_top(self):
34        '''Тест: прямоугольник касается сверху'''
35        touching_top = Rectangle(2, 5, 1, 1)
36        self.assertFalse(self.rect.is_in(touching_top))
37
38    def test_touching_bottom(self):
39        '''Тест: прямоугольник касается снизу'''
40        touching_bottom = Rectangle(2, 0, 1, 1)
41        self.assertFalse(self.rect.is_in(touching_bottom))
42
43    def test_intersect_left(self):
44        '''Тест: пересечение прямоугольника слева'''
45        intersect_left = Rectangle(0, 2, 3, 2)
46        self.assertTrue(self.rect.is_in(intersect_left))
47
48    def test_intersect_right(self):
49        '''Тест: пересечение прямоугольника справа'''
50        intersect_right = Rectangle(3, 2, 3, 2)
51        self.assertTrue(self.rect.is_in(intersect_right))
52
53    def test_intersect_top(self):
54        '''Тест: пересечение прямоугольника сверху'''
55        intersect_top = Rectangle(2, 3, 2, 3)
56        self.assertTrue(self.rect.is_in(intersect_top))
57
58    def test_intersect_bottom(self):
59        '''Тест: пересечение прямоугольника снизу'''
60        intersect_bottom = Rectangle(2, 2, 2, 2)

```

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-026
Шевченко Клим Николаевич

ВКРБ 20060139.09.03.04.24.012			Лист	Из	Листов
Сведения о ВКРБ			Лист	Из	Листов
Автор работы	Шевченко К.Н.		Лист	Из	Листов
Руководитель	Чаплыгин А.А.		Лист	Из	Листов
Мужинский	Чаплыгин А.А.		Лист	Из	Листов
Выпускная квалификационная работа бакалавра			ЮЗГУ ПО-026		

Рисунок А.1 – Сведения о ВКРБ

Цель и задачи разработки

Цель работы - разработка приложения для разработки компьютерных ролевых игр с заранее отрисованными спрайтами и фоном.

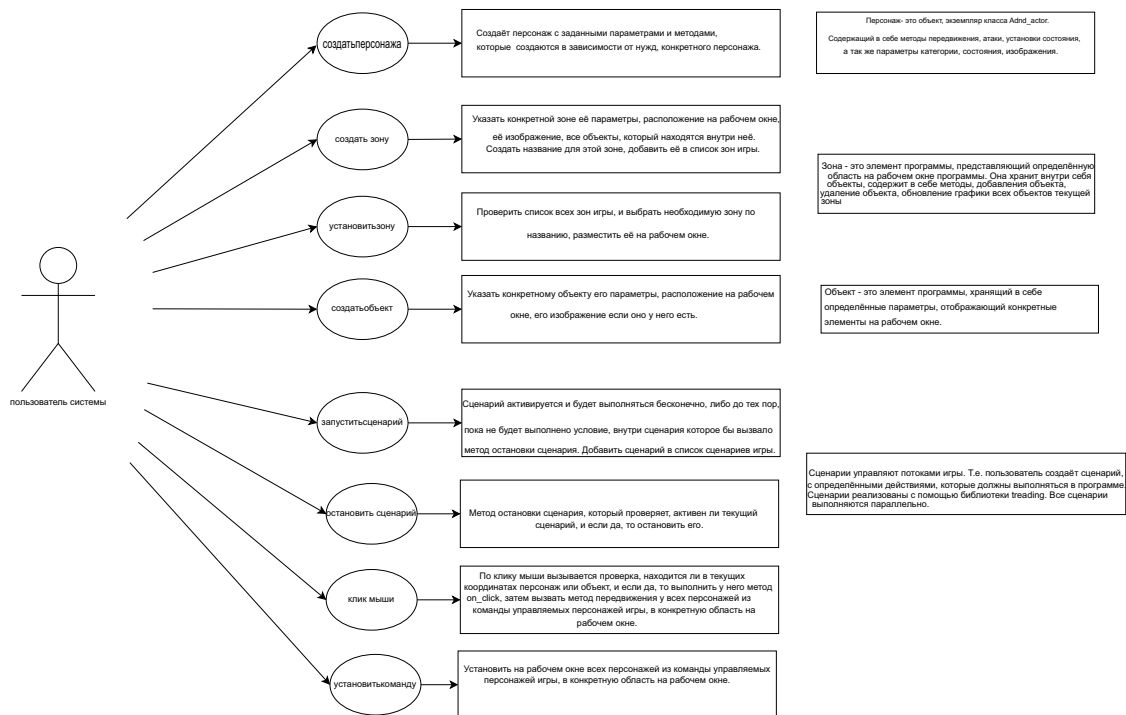
Для достижения поставленной цели требуется решить следующие задачи:

1. Провести анализ предметной области.
2. Разработать концептуальную модель приложения.
3. Спроектировать приложение.
4. Реализовать приложение средствами языка программирования Python.

ВКРБ 20060139.09.03.04.24.012		Лит.	Норм.	Исход.
Автор работы	Фамилия И. О.			
Руководитель	Иванченко К.С.			
Модератор	Чалыгин А.А.			
Цель и задачи разработки		Лит.	Норм.	Исход.
Выпуск квалификационной работы бакалавра		Лит.	Норм.	Исход.
		ЮЗГУ ПО-026		

Рисунок А.2 – Цель и задачи разработки

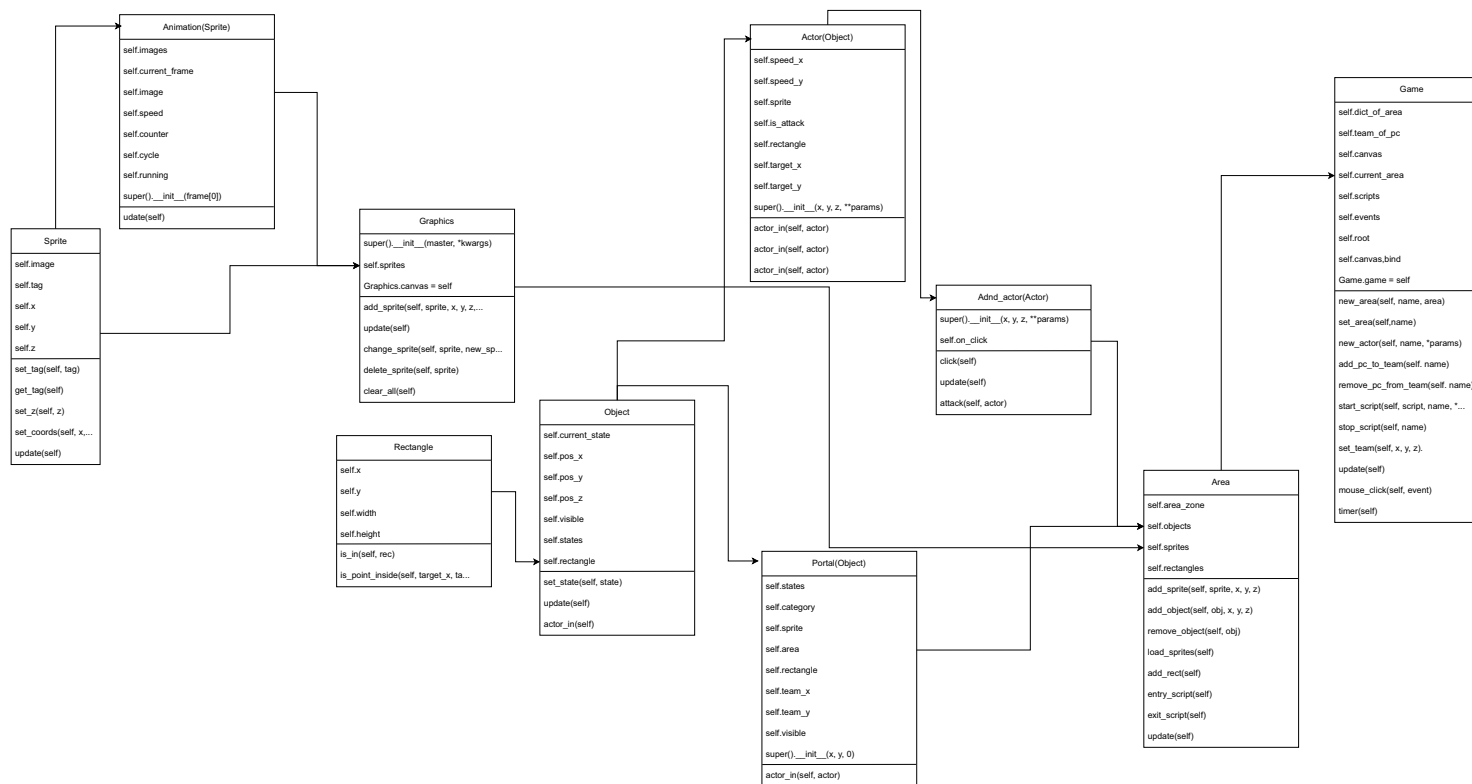
Концептуальная модель



ВКРБ 20060139.09.03.04.24.012			
Концептуальная модель		Лит.	Норм.
Автор работы	Иванченко К.Ю.	Акт 1	Лист 6
Руководитель	Чалыгина А.А.		
Мужинский	Чалыгина А.А.		
Выпуск квалификационной работы бакалавра		ЮЗГУ ПО-026	

Рисунок А.3 – Концептуальная модель приложения

Диаграмма классов программы



ВКРБ 20060139.09.03.04.24.012			
Фамилия И. О.	Имя	Фамилия И. О.	Имя
Автор работы	Иванов И.И.	Руководитель	Петров П.П.
Мужской пол	Человек А.А.	Женский пол	Человек Б.Б.
Выполнил		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: А, В и С. 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 А | Поток В | Поток С

```

-----
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

ВКРБ 20060139.09.03.04.24.012			Лит.	Норм.	Нормат.
Фамилия И. О.	Инициалы		Модель работы сценариев		
Автор работы	Иванов И.И.				
Рецензент	Петров П.П.		Выпуск квалификационной работы бакалавра		
Модератор	Сидоров С.С.				
			Лист 1	Лист 6	
			ЮЗГУ ПО-026		

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

ВКРБ 20060139.09.03.04.24.012			
Фамилия И. О.	Имя	Фамилия	Имя
Автор работы	Иванов И.И.	Модульное тестирование платформы	Лист 1
Руководитель	Петров П.П.		Лист 6
Мужской пол	Человек А.А.	Выпуск квалификационной работы бакалавра	ЮЗГУ ПО-026

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.

ВКРБ 20060139.09.03.04.24.012			Лит.	Норм.	Нормат.
Фамилия И. О.	Инициалы				
Автор работы	Иванов И. С.				
Руководитель	Петров П. А.				
Модератор	Сидоров С. А.				
Заключение			Лит.	Норм.	Нормат.
Выпуск квалификационной работы бакалавра			Лит.	Норм.	Нормат.
			Лит.	Норм.	Нормат.

Рисунок А.7 – Заключение

ПРИЛОЖЕНИЕ Б

Фрагменты исходного кода программы

main.tex

```
1 \input{setup.tex}
2
3 % Режим шаблона (должен быть включен один из трех)
4 \BKPtrue
5 %\Практикаtrue
6 %\Курсоваяtrue
7
8 \newcommand{\Дисциплина}{<<Проектирование и архитектура программных систем>>}
9   % для курсовой
10 \newcommand{\КодСпециальности}{09.03.04} % Курсовая
11 \newcommand{\Специальность}{Программная инженерия} % Курсовая
12 \newcommand{\Тема}{Платформа для создания компьютерных изометрических ролевых
13   игр} % ВКР Курсовая
14 \newcommand{\ТемаВтораяСтрока}{с заранее отрисованным двухмерным фоном и
15   спрайтовыми персонажами}
16 \newcommand{\ГдеПроводитсяПрактика}{Юго-Западном государственном университете
17   } % для практики
18 \newcommand{\РуководительПрактПредпр}{ } % для практики
19 \newcommand{\ДолжнРуководительПрактПредпр}{директор} % для практики
20 \newcommand{\РуководительПрактУнивер}{Чаплыгин А. А.} % для практики
21 \newcommand{\ДолжнРуководительПрактУнивер}{к.т.н. доцент} % для практики
22 \newcommand{\Автор}{К. Н. Шевченко}
23 \newcommand{\АвторРод}{Шевченко К.Н.}
24 \newcommand{\АвторПолностьюРод}{Шевченко Клима Николаевича} % для практики
25 \newcommand{\Шифр}{20-06-0139}
26 \newcommand{\Курс}{4} % для практики
27 \newcommand{\Группа}{ПО-026}
28 \newcommand{\Руководитель}{А. А. Чаплыгин} % для ВКР и курсовой
29 \newcommand{\Нормоконтроль}{А. А. Чаплыгин} % для ВКР
30 \newcommand{\ЗавКаф}{А. В. Малышев} % для ВКР
31 \newcommand{\ДатаПриказа}{«04» апреля 2024~г.} % для ВКР
32 \newcommand{\НомерПриказа}{1616-с} % для ВКР
33 \newcommand{\СрокПредоставления}{«11» июня 2024~г.} % для ВКР, курсового
34
35 \begin{document}
36 \maketitle
37 \ifПрактика{}\else{
38   \input{ЛистЗадания}
39   \input{Реферат}}\fi
40 \tableofcontents
41 \input{Обозначения}
42 \ifПрактика{}\else{\input{Введение}}\fi
43 \input{Анализ}
44 \input{ТехЗадание}
45 \input{ТехПроект}
46 \ifПрактика{}\else{
47   \input{РабочийПроект}
48   \input{Заключение}
49 }\fi
```

```

46 \input{СписокИсточников}
47 \ifBKP{\input{Плакаты}}\fi
48 \ifПрактика{}\else{\input{Код}}\fi
49 \end{document}

```

TexПроект.tex

```

1 \section{Технический проект}
2 \subsection{Общая характеристика организации решения задачи}
3
4 Необходимо спроектировать и разработать приложение, который должен
   способствовать популяризации ролевых игр.
5
6 Приложение представляет собой набор взаимосвязанных различных окон, которые
   сгруппированы по разделам, содержащие текстовую, графическую информацию.
   Приложение располагается на компьютере.
7
8 \subsection{Обоснование выбора технологии проектирования}
9
10 На сегодняшний день информационный рынок, поставляющий программные решения в
   выбранной сфере, предлагает множество продуктов, позволяющих достигнуть
   поставленной цели – разработки приложения.
11
12 \subsubsection{Описание используемых технологий и языков программирования}
13
14 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.
15
16 \subsubsection{Язык программирования Python}
17
18 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 С или С++.
19
20 \subsubsection{Использование библиотеки Tkinter и реализация таймеров на
   Python}
21
22 \paragraph{Введение}
23 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.
24

```

```

25 \paragraph{Возможности Tkinter}
26 Вот некоторые из основных возможностей, предоставляемых библиотекой Tkinter:
27
28 \begin{itemize}
29 \item Создание различных виджетов: кнопки, метки, поля ввода, списки и
    многое другое.
30 \item Управление компоновкой виджетов с использованием менеджеров
    компоновки (например, grid, pack, place).
31 \item Обработка событий, таких как щелчок мыши, нажатие клавиш и другие.
32 \item Возможность создания различных диалоговых окон, таких как окна
    предупреждений, информационные окна и окна запроса.
33 \item Поддержка многопоточности для обновления интерфейса из различных
    потоков выполнения.
34 \end{itemize}
35
36 \paragraph{Реализация таймеров на Python}
37 Для реализации таймеров на Python можно использовать модуль \texttt{time} или
    \texttt{threading}. Вот пример использования модуля \texttt{time} для
    создания простого таймера:
38
39 import time
40
41 def countdown(t):
42     while t > 0:
43         mins, secs = divmod(t, 60)
44         timeformat = '{:02d}:{:02d}'.format(mins, secs)
45         print(timeformat, end='\r')
46         time.sleep(1)
47         t -= 1
48         print('Таймер завершен!')
49
50
51 t = 10
52 countdown(t)
53
54 Этот код создает простой обратный отсчет таймера с использованием функции \
    \texttt{countdown}. Он выводит оставшееся время в формате ММ:СС и уменьшает
    его на 1 каждую секунду, используя функцию \texttt{time.sleep(1)}. Когда
    время истекает, выводится сообщение о завершении таймера.
55
56 \paragraph{Заключение}
57 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.
58
59 \subsection{Описание платформы для создания RPG игр}
60 Клиент создает модуль содержащий методы модуля RPGGame, например bgame. 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.

61 Зона тоже может содержать сценарий, который запускается когда команда попадает в зону.

62 Клиентский класс (BGGame) также содержит глобальные переменные, определяющие ситуации в игре (например квесты). Локальные переменные могут быть в зоне.

63

64 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 :

65 Area.set_enter_script(script)

66 В зону мы добавляем персонажей и предметы как add_object(x,y, obj) - z не нужно, так как слой можно определить по y координате.

67 В конкретную зону мы добавляем сценарий для взаимодействия как: Game.game.start_script(script, name)

68 Как происходит переход команды между зонами.

69 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.

70 Сценарии - это потоки которые запускаются параллельно (метод RPGGame.start_script(script)). Сценарий может быть остановлен (stop_script(name)).

71 Таким образом, мир будет интерактивным.

72 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.

73 Также в таймере вызываем Graphics.update(), который обновляет графику игры.

74 Все объекты (Actor, Item) должны иметь состояния (как минимум одно). Каждое состояние связано с спрайтом (или анимацией). То есть переключение состояния меняет графику объекта.

75

76 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):

77 import random

78 from math import sqrt

79 import time

80 from rpg.area import *

81 from rpg.sprite import *

82 from rpg.rectangle import *

83 from rpg.game import Game

84 from rpg.portal import Portal

85

86 class Ruins(Area):

87 def __init__(self):


```

88     '''
89     Класс игровой зоны Ruins
90
91     '''
92     super().__init__()
93     self.add_sprite(Sprite('images/fon3.png'), 590, 400, 0)
94     self.add_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.
95         width(), height=Sprite('images/fon3.png').image.height()))
96     from grunt import Grunt
97     self.grunt = Grunt(0,0,0)
98     from footman import Footman
99     self.footman = Footman(0,0,0)
100    self.add_object(self.footman, 120, 120, 1)
101    self.add_object(self.grunt, 500, 185, 1)
102    p = Portal(400, 400, 200, 200, 'Village', 480, 100)
103    self.add_object(p, p.pos_x, p.pos_y, 100)
104    Game.game.start_script(self.ai, "ai", self.grunt)
105    Game.game.start_script(self.walk_two, "footman", 50, 50)
106
107    def walk(self, step_x, step_y, actor):
108        '''
109        Сценарий для движения бугая
110
111        :param step_x: шаг движения x
112        :param step_y: шаг движения y
113        '''
114        if actor.hp <= 0:
115            Game.game.stop_script("grunt")
116            new_x = 200
117            new_y = 200
118            actor.is_attack = False
119            direction = random.choice(["up", "down", "left", "right"])
120            if direction == "up":
121                new_y -= step_y
122                new_x = step_x
123            elif direction == "down":
124                new_y += step_y
125                new_x = step_x
126            elif direction == "left":
127                new_y = step_y
128                new_x -= step_x
129            elif direction == "right":
130                new_y = step_y
131                new_x += step_x
132
133            actor.search_position(new_x, new_y)
134
135            time.sleep(2)
136
137    модуль bggame:
138    from ruins import *
139    import time
140    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 выводиться 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 \subsection{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 выводиться 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
201             персонажами
202
203         :param x: координата x
204         :param y: координата y
205         :param z: координата z
206         '''
207         super().__init__(x, y, z, **params)
208         self.on_click = self.click
209
210     def click(self):
211         '''
212         вызываете при клике на персонажа
213
214         '''
215         pc = rpg.game.Game.game.team_of_pc[0]
216         if pc == self:
217             return
218         dx = pc.pos_x - self.pos_x
219         dy = pc.pos_y - self.pos_y
220         dist = sqrt(dx * dx + dy * dy)
221         if dist <= self.ATTACK_RANGE:
222             pc.is_attack = True
223             pc.attack(self)
224             time.sleep(0.125)
225             if self.hp <= 0:
226                 pc.is_attack = False
227
228     def attack(self, actor):
229         '''
230         совершает атаку по actor
231
232         :param actor: персонаж, которого атакуют
233         '''
234         actor.hp -= self.damage
235
236     def update(self):
237         '''
238         обновляет состояние персонажа
239
240         '''
241         super().update()
242         if self.hp <= 0:
243             self.stop_move()
244             self.set_state('death')

```

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 куда и 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 помощью 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)`/

```

386 Пример:
387 модуль adnd_actor:
388 from math import sqrt
389 from rpg.actor import Actor
390 from rpg.animation import Animation
391 import rpg.game
392 import time
393
394 class Adnd_actor(Actor):
395
396     ATTACK_RANGE = 50
397
398     def __init__(self, x, y, z, **params):
399         '''
400         Класс Adnd_actor содержащий основные механики взаимодействия с другими
401         персонажами
402
403         :param x: координата x
404         :param y: координата y
405         :param z: координата z
406         '''
407         super().__init__(x, y, z, **params)
408         self.on_click = self.click
409
410     def click(self):
411         '''
412         вызывается при клике на персонажа
413
414         '''
415         pc = rpg.game.Game.team_of_pc[0]
416         if pc == self:
417             return
418         dx = pc.pos_x - self.pos_x
419         dy = pc.pos_y - self.pos_y
420         dist = sqrt(dx * dx + dy * dy)
421         if dist <= self.ATTACK_RANGE:
422             pc.is_attack = True
423             pc.attack(self)
424             time.sleep(0.125)
425             if self.hp <= 0:
426                 pc.is_attack = False
427
428     def attack(self, actor):
429         '''
430         совершает атаку по actor

```

```

431 :param actor: персонаж, которого атакуют
432 '''
433 actor.hp -= self.damage
434 def update(self):
435     '''
436     обновляет состояние персонажа
437
438     '''
439     super().update()
440     if self.hp <= 0:
441         self.stop\_move()
442         self.set\_state('death')
443
444     модуль ruins
445     import random
446     from math import sqrt
447     import time
448     from rpg.area import *
449     from rpg.sprite import *
450     from rpg.rectangle import *
451     from rpg.game import Game
452     from rpg.portal import Portal
453
454     class Ruins(Area):
455         def \_\_init\_\_(self):
456             '''
457             Класс игровой зоны Ruins
458
459             '''
460             super().\_\_init\_\_()
461             self.add\_sprite(Sprite('images/fon3.png'), 590, 400, 0)
462             self.add\_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.
463                 width(), height=Sprite('images/fon3.png').image.height()))
464             from grunt import Grunt
465             self.grunt = Grunt(0,0,0)
466             from footman import Footman
467             self.footman = Footman(0,0,0)
468             self.add\_object(self.footman, 120, 120, 1)
469             self.add\_object(self.grunt, 500, 185, 1)
470             p = Portal(400, 400, 200, 200, 'Village', 480, 100)
471             self.add\_object(p, p.pos\_x, p.pos\_y, 100)
472             Game.game.start\_script(self.ai, "ai", self.grunt)
473             Game.game.start\_script(self.walk\_two, "footman", 50, 50)
474
475         def walk(self, step\_x, step\_y, actor):
476             '''
477             Сценарий для движения бугая
478
479             :param step\_x: шаг движения x
480             :param step\_y: шаг движения y
481             '''
482             if actor.hp <= 0:
483                 Game.game.stop\_script("grunt")

```



```

484 new\_x = 200
485 new\_y = 200
486 actor.is\_attack = False
487 direction = random.choice(["up", "down", "left", "right"])
488 if direction == "up":
489     new\_y -= step\_y
490     new\_x = step\_x
491 elif direction == "down":
492     new\_y += step\_y
493     new\_x = step\_x
494 elif direction == "left":
495     new\_y = step\_y
496     new\_x -= step\_x
497 elif direction == "right":
498     new\_y = step\_y
499     new\_x += step\_x
500
501 actor.search\_position(new\_x, new\_y)
502
503 def ai(self, actor):
504     '''
505     скрипт противников
506
507     :param step\_x: размер шага x до персонажа игрока
508     :param step\_y: размер шага y до персонажа игрока
509     :param actor: персонаж противник
510     '''
511     if actor.hp <= 0:
512         Game.game.stop\_script("ai")
513     import rpg.game
514     pc = rpg.game.Game.game.team\_of\_pc[0]
515     new\_x = pc.pos\_x
516     new\_y = pc.pos\_y
517
518     actor.search\_position(new\_x, new\_y)
519     dx = pc.pos\_x - actor.pos\_x
520     dy = pc.pos\_y - actor.pos\_y
521     dist = sqrt(dx * dx + dy * dy)
522     if dist <= actor.ATTACK\_RANGE:
523         actor.is\_attack = True
524         actor.attack(pc)
525         time.sleep(1)
526         if pc.hp <= 0:
527             actor.update()
528         Game.game.stop\_script("ai")
529         Game.game.start\_script(self.walk, "grunt", 50, 50, actor)
530
531     else:
532         actor.is\_attack = False
533         time.sleep(2)
534
535 \paragraph{Соединение движка и окон tkinter}
536 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
```

```

588 def update(self):
589     '''
590     Обновляет анимацию спрайта
591     '''
592     pass
593
594
595 модуль graphics
596
597 import tkinter as tk
598
599 class Graphics(tk.Canvas):
600     canvas = None
601     def __init__(self, master, **kwargs):
602         '''
603         Класс с методами для работы со спрайтами
604         '''
605         super().__init__(master, **kwargs)
606         self.sprites = []
607         Graphics.canvas = self
608
609     def add_sprite(self, sprite, x, y, z, **kwargs):
610         '''
611         Добавляет спрайт на Canvas
612         :param sprite: спрайт
613         :param x: координата x
614         :param y: координата y
615         :param z: координата z
616         :param kwargs: параметры относящиеся к конкретному изображению в tkinter
617         '''
618         tag = self.create_image(x, y, image=sprite.image, anchor='center', **kwargs)
619         sprite.set_tag(tag)
620         sprite.set_z(z)
621         sprite.x = x
622         sprite.y = y
623         self.sprites.append(sprite)
624         self.sprites.sort(key=lambda sprite: sprite.z)
625
626     def update(self):
627         '''
628         Перерисовывает все спрайты
629         '''
630         for sprite in self.sprites:
631             sprite.update()
632             self.tag_raise(sprite.get_tag())
633             self.coords(sprite.get_tag(), sprite.x, sprite.y)
634             self.itemconfig(sprite.get_tag(), image=sprite.image)
635
636     def change_sprite(self, sprite, new_sprite):
637         '''
638
639
640
641

```

```

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 выводиться 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 bgame 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{Диаграмма компонентов классов}
727
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 \subsection{Модули и классы}
735 \subsection{Game}
736 \subsubsection{Описание модуля}
737 \paragraph{Конструктор и поля модуля}
738 def __init__(self, canvas, window, **params)
739 \begin{itemize}
740 \item self.rpg_dict_of_area = {}

```

```

741 \item словарь, хранящий в себе множество экземпляров класса Area, {number -
      ключ : name Area - значение}
742 \item self.team\_of\_pc = []
743 \item список, хранящий в себе имена экземпляров класса Actor с параметром
      category = "pc"
744 \item self.canvas = canvas
745 \item графика
746 \item self.root = window
747 \item окно для графики
748 \item self.current\_area = None
749 \item параметр хранящий, текущую зону
750 \item self.scripts = {}
751 \item Словарь для хранения запущенных сценариев
752 \item self.events = {}
753 \item Словарь для хранения запущенных event`ов сценариев
754 \item self.canvas.bind("<Button-1>", self.mouse\_left\_click)
755 \item обработчик клика
756 \item Game.game = self
757 \item экземпляр игры, для обращения к нему напрямую
758 \end{itemize}
759 \subsubsection{Методы}
760 \paragraph{New\_area}
761 def new\_area(self, name, area)
762 Описание метода: Добавляет новую зону в список.
763 \paragraph{Set\_area}
764 def set\_area(self, name)
765 Описание метода: Устанавливает текущую зону, загружает графику зоны.
766 \paragraph{New\_actor}
767 def new\_actor(self, name, **params)
768 Описание метода: метод отвечающий за создание класса, потомка от Actor и
      создание поля из параметров, и установление их в начальные значения.
769 \paragraph{Start\_script}
770 def start\_script(self, script\_function, script\_name, *args)
771 Описание метода: Запускает сценарий в отдельном потоке с возможностью
      остановки и передачи аргументов.
772 \paragraph{Stop\_script}
773 def stop\_script(self, script\_name)
774 Описание метода: Останавливает сценарий по имени.
775 \paragraph{Add\_pc\_to\_team}
776 def add\_pc\_to\_team(self, pc)
777 Описание метода: метод отвечающий за добавление имени экземпляра класса Actor
      с параметром category = "pc" в список team\_of\_pc, хранящий имена всех
      игровых персонажей.
778 \paragraph{Remove\_pc\_from\_team}
779 def remove\_pc\_from\_team(self, pc)
780 Описание метода: метод отвечающий за удаление имени экземпляра класса Actor с
      параметром category = "pc" в список team\_of\_pc, хранящий имена всех
      игровых персонажей.
781 \paragraph{Set\_team}
782 def set\_team(self, x, y, z)
783 Описание метода: Устанавливает координаты персонажей команды.
784 \paragraph{Update}
785 def update(self)

```

```

786 Описание метода: Вызывается в таймере для обновления всех переменных в
      текущей зоне.
787 \paragraph{Mouse\_left\_click}
788 def mouse\_left\_click(self, event)
789 Описание метода: обрабатывает клик мыши.
790 \paragraph{Timer}
791 def timer(self)
792 Описание метода: Таймер должен вызывать метод update постоянно.
793
794
795 \subsection{Area}
796 \subsubsection{Описание модуля}
797 \paragraph{Конструктор и поля модуля}
798 def \_\_init\_\_(self, **params)
799 \begin{itemize}
800   \item self.area\_zone = params
801   \item параметр определяющий особенности конкретной зоны
802   \item self.objects = []
803   \item список, хранящий в себе множество экземпляров классов Item
804   \item self.sprites = []
805   \item список фоновых спрайтов
806   \item self.rectangles = None
807   \item список, хранящий в себе множество прямоугольников
808 \end{itemize}
809 \subsubsection{Методы}
810 \paragraph{Add\_sprite}
811 def add\_sprite(self, sprite, x, y, z)
812 Описание метода: Добавляет спрайт в зону.
813 \paragraph{Add\_object}
814 def add\_object(self, obj, x, y, z)
815 Описание метода: Добавляет объект в зону.
816 \paragraph{Remove\_object}
817 def remove\_object(self, obj)
818 Описание метода: Удаляет объект из зоны.
819 \paragraph{Load\_sprites}
820 def load\_sprites(self)
821 Описание метода: Загружает все спрайты зоны.
822 \paragraph{Add\_rect}
823 def add\_rect(self, rec)
824 Описание метода: Добавляет прямоугольник в зону.
825 \paragraph{Entry\_script}
826 def entry\_script(self)
827 Описание метода: Запускается, когда команда входит в зону.
828 \paragraph{Exit\_script}
829 def exit\_script(self)
830 Описание метода: Запускается, когда команда выходит из зоны.
831 \paragraph{Update}
832 def update(self)
833 Описание метода: Изменяет и проверяет изменение всех персонажей в зоне.
834
835 \subsection{Sprite}
836 \subsubsection{Описание модуля}
837 \paragraph{Конструктор и поля модуля}
838 def \_\_init\_\_(self, image)

```

```

839 \begin{itemize}
840   \item self.spr\_image = image
841   \item Описание параметра: параметр хранит изображение конкретного
      экземпляра класса Sprite.
842   \item self.tag = None
843   \item self.spr\_x = x
844   \item Описание параметра: параметр хранит числовое значение обозначающее
      расположение конкретного экземпляра класса Sprite.
845   \item self.spr\_y = y
846   \item Описание параметра: параметр хранит числовое значение обозначающее
      расположение конкретного экземпляра класса Sprite.
847   \item self.spr\_z = z
848   \item Описание параметра: параметр хранит числовое значение обозначающее
      расположение конкретного экземпляра класса Sprite.
849 \end{itemize}
850 \subsubsection{Методы}
851 \paragraph{Set\_tag}
852 def set\_tag(self, tag)
853 \paragraph{Set\_z}
854 def set\_z(self, z)
855 \paragraph{Get\_tag}
856 def get\_tag(self)
857 \paragraph{Set\_coords}
858 def set\_coords(self, new\_x, new\_y)
859 \paragraph{Update}
860 def update(self)
861 \subsection{Graphics}
862 \subsubsection{Описание модуля}
863 \paragraph{Конструктор и поля модуля}
864 def \_\_init\_\_(self, master, **kwargs)
865 \begin{itemize}
866   \item super().\_\_init\_\_(master, **kwargs)
867   \item self.sprites = []
868   \item список спрайтов
869   \item Graphics.canvas = self
870   \item параметр для работы с графикой, где к ней нужно обращаться напрямую
871 \end{itemize}
872 \subsubsection{Методы}
873 \paragraph{Add\_sprite}
874 def add\_sprite(self, sprite, x, y, z, **kwargs)
875 Описание метода: Добавляет спрайт на Canvas
876 \paragraph{Update}
877 def update(self)
878 Описание метода: Перерисовывает все спрайты
879 \paragraph{Change\_sprite}
880 def change\_sprite(self, sprite, new\_sprite)
881 Описание метода: Меняет спрайт на новый
882 \paragraph{Delete\_sprite}
883 def delete\_sprite(self, sprite)
884 Описание метода: Удаляет спрайт с Canvas
885 \paragraph{Clear\_all}
886 def clear\_all(self)
887 Описание метода: Удаляет все спрайты с Canvas
888

```



```

889 \subsection{Animation(Sprite)}
890 \subsubsection{Описание модуля}
891 \paragraph{Конструктор и поля модуля}
892 def \_\_init\_\_(self, frames, cycle=True)
893 \begin{itemize}
894 \item super().\_\_init\_\_(frames[0])
895 \item self.images = frames
896 \item Описание параметра: """"список кадров""""
897 \item self.current\_frame = 0
898 \item self.images = [tk.PhotoImage(file=frame) for frame in frames]
899 \item Загрузка всех кадров анимации
900 \item self.image = self.images[0]
901 \item Установка начального изображения
902 \item self.speed = 3
903 \item Описание параметра: """"скорость анимации""""
904 \item self.counter = self.speed
905 \item self.cycle = cycle
906 \item self.running = True
907 \end{itemize}
908 \subsubsection{Методы}
909 \paragraph{Update:}
910 def update(self):
911 Меняет текущее изображение в списке изображений.
912
913 \subsection{Object}
914 \subsubsection{Описание модуля}
915 \paragraph{Конструктор и поля модуля}
916 def \_\_init\_\_(self, x, y, z, **params)
917 \begin{itemize}
918 \item self.pos\_x = x
919 \item координата x
920 \item self.pos\_y = y
921 \item координата y
922 \item self.pos\_z = z
923 \item координата z
924 \item self.current\_state = None
925 \item текущее состояние
926 \item self.visible = True
927 \item видим ли объект
928 \item self.on\_click = lambda x : x
929 \item возможно ли кликнуть по объекту
930 \item if self.states is not None:
931 self.set\_state(next(iter(self.states)))
932 \item установка первого состояния если потребуется
933 \item self.rectangle = Rectangle(x, y, 10, 10)
934 \item прямоугольник объекта
935 \end{itemize}
936 \subsubsection{Методы}
937 \paragraph{Set\_state}
938 def set\_state(self, state\_name):
939 Описание метода: меняет текущее состояние объекта
940 \paragraph{Actor\_in}
941 def actor\_in(self, actor):
942 Описание метода:Вызывается когда actor входит внутрь объекта

```

```

943 \paragraph{Update}
944 def update(self)
945 Описание метода: ничего не делает
946
947 \subsection{Actor(Object)}
948 \subsubsection{Описание модуля}
949 \paragraph{Конструктор и поля модуля}
950 def \_\_init\_\_(self, x, y, z, **params)
951 \begin{itemize}
952 \item self.sprite = self.states[next(iter(self.states))]
953 \item параметр хранящий спрайт
954 \item super().\_\_init\_\_(x, y, z, **params)
955 \item self.speed\_x = 0
956 \item значение скорости x
957 \item self.speed\_y = 0
958 \item значение скорости y
959 \item self.target\_x = 0
960 \item координата x в которую будет двигаться персонаж
961 \item self.target\_y = 0
962 \item координата y в которую будет двигаться персонаж
963 \item self.rectangle = Rectangle(self.pos\_x, self.pos\_y, self.sprite.
    image.width(), self.sprite.image.height())
964 \item прямоугольник персонажа
965 \item self.is\_attack = False
966 \item состояние - атакует ли персонаж сейчас
967 \end{itemize}
968 \subsubsection{Методы}
969 \paragraph{update}
970 def update(self)
971 Описание метода: изменяет координаты и состояние персонажа.
972 \paragraph{Search\_position}
973 def search\_position(self, new\_x, new\_y)
974 Описание метода: Изменяет направление движения у персонажа.
975 \paragraph{Stop\_move}
976 def stop\_move(self):
977 Описание метода: Останавливает движение персонажа.
978
979 \subsection{Adnd\_actor(Actor)}
980 \subsubsection{Описание модуля}
981 \paragraph{Конструктор и поля модуля}
982 def \_\_init\_\_(self, x, y, z, **params)
983 \begin{itemize}
984 \item super().\_\_init\_\_(x, y, z, **params)
985 \item self.on\_click = self.click
986 \item событие по клику на персонажа
987 \end{itemize}
988 \subsubsection{Методы}
989 \paragraph{Click}
990 def click(self)
991 Описание метода: вызывается при клике на персонажа.
992 \paragraph{Attack}
993 def attack(self, actor)
994 Описание метода: совершает атаку по actor.
995 \paragraph{Update}

```

```

996 def update(self)
997 Описание метода: обновляет состояние персонажа.
998
999 \subsection{Rectangle}
1000 \subsubsection{Описание модуля}
1001 \paragraph{Конструктор и поля модуля}
1002 def \_\_init\_\_(self, x, y, width, height)
1003 \begin{itemize}
1004 \item self.x = x
1005 \item координата x прямоугольника
1006 \item self.y = y
1007 \item координата y прямоугольника
1008 \item self.width = width
1009 \item ширина прямоугольника
1010 \item self.height = height
1011 \item высота прямоугольника
1012 \end{itemize}
1013 \subsubsection{Методы}
1014 \paragraph{Is\_in}
1015 def is\_in(self, rect)
1016 Описание метода: Проверяет, входит ли прямоугольник self в прямоугольник rect
1017 \paragraph{Is\_point\_inside}
1018 def is\_point\_inside(self, target\_x, target\_y)
1019 Описание метода: Проверяет, входит ли точка (x, y) в данный прямоугольник
1020
1021 \subsection{Portal(Object)}
1022 \subsubsection{Описание модуля}
1023 \paragraph{Конструктор и поля модуля}
1024 def \_\_init\_\_(self, x, y, width, height, area, team\_x, team\_y)
1025 \begin{itemize}
1026 \item self.states = None
1027 \item состояние портала
1028 \item self.sprite = None
1029 \item спрайт портала
1030 \item self.category = 'portal'
1031 \item категория портала
1032 \item super().\_\_init\_\_(x, y, 0)
1033 \item self.rectangle = Rectangle(x, y, width, height)
1034 \item прямоугольник портала
1035 \item self.area = area
1036 \item текущая зона
1037 \item self.team\_x = team\_x
1038 \item координата x в новой зоне в которую установят команду
1039 \item self.team\_y = team\_y
1040 \item координата y в новой зоне в которую установят команду
1041 \item self.visible = False
1042 \item видимость портала
1043 \end{itemize}
1044 \subsubsection{Методы}
1045 \paragraph{Actor\_in}
1046 def actor\_in(self, actor)
1047 Описание метода: Проверяет находится ли персонаж внутри портала

```

Место для диска