

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-026

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 «25» 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 компаний; 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 102 страницам. Работа содержит 10 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: движок, 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 движках, графике.

## ABSTRACT

The volume of work is 102 pages. The work contains 10 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: commercial website, System, CMS, Bitrix, Joomla, additive technologies, 3D printers, services, services, informatization, automation, information technology, web form, Apache, classes, database, component, module, entity, information block, method, content editor, administrator, user, web site.

The object of the research is the analysis of information technologies for the development of a production company's website.

The object of the development is the website of a company engaged in the production of 3D printers, the production of equipment for the creation of powders, software development and the organization of additive manufacturing centers.

The purpose of the final qualifying work is to attract customers, increase orders, inform about products and services by creating a company website.

In the process of creating the site, the main entities were identified by creating information blocks, classes and methods of modules were used to ensure work with the entities of the subject area, as well as the correct operation of the website, sections containing information about the company, its activities, products and services were developed, a service for ordering 3D parts was developed.

## СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>12</b>
1 Анализ предметной области	14
1.1 История первых игр что стали прародителями RPG-жанра	14
1.2 Первые популярные RPG-игры	15
1.3 Япония и её JRPG	16
1.4 Популярные RPG студии SSI	17
2 Техническое задание	19
2.1 Основание для разработки	19
2.2 Цель и назначение разработки	19
2.3 Требования пользователя к движку	19
2.4 Правила игры	19
2.5 Особенности Dungeons and Dragons	21
2.6 Игровой мир Фаэрун	23
2.7 монстры мира Фаэрун	24
2.8 Интерфейс пользователя	26
2.9 Пример игры	27
2.10 Моделирование вариантов использования	28
2.11 Требования к оформлению документации	28
3 Технический проект	29
3.1 Общая характеристика организации решения задачи	29
3.2 Обоснование выбора технологии проектирования	29
3.2.1 Описание используемых технологий и языков программирования	29
3.2.2 Язык программирования Python	29
3.2.3 Язык программирования Python	30
3.2.3.1 Достоинства языка Python	30
3.2.3.2 Недостатки языка Python	30
3.2.4 Использование библиотеки Tkinter и реализация таймеров на Python	31
3.2.4.1 Введение	31
3.2.4.2 Возможности Tkinter	31

3.2.4.3 Реализация таймеров на Python	31
3.2.4.4 Заключение	32
3.3 Описание платформы для создания RPG игр	32
3.3.1 Пример клиентского кода игры	34
3.3.1.1 Создание классов персонажей/предметов	34
3.3.1.2 Задание правил атаки	35
3.3.1.3 Создание зон, заполнение их персонажами/объектами	35
3.3.1.4 Пример сценариев: переход между зонами	37
3.3.1.5 Как будет идти бой	38
3.3.1.6 Соединение движка и окон tkinter	40
3.4 Модули и классы	42
3.5 Game	42
3.5.1 Описание модуля	42
3.5.1.1 Конструктор и поля модуля	42
3.5.2 Методы	43
3.5.2.1 New_area	43
3.5.2.2 Set_area	43
3.5.2.3 New_actor	43
3.5.2.4 Start_script	43
3.5.2.5 Stop_script	43
3.5.2.6 Add_pc_to_team	44
3.5.2.7 Remove_pc_from_team	44
3.5.2.8 Set_team	44
3.5.2.9 Update	44
3.5.2.10 Mouse_left_click	44
3.5.2.11 Timer	44
3.6 Area	44
3.6.1 Описание модуля	44
3.6.1.1 Конструктор и поля модуля	44
3.6.2 Методы	45
3.6.2.1 Add_sprite	45

3.6.2.2 Add_object	45
3.6.2.3 Remove_object	45
3.6.2.4 Load_sprites	45
3.6.2.5 Add_rect	45
3.6.2.6 Entry_script	45
3.6.2.7 Exit_script	46
3.6.2.8 Update	46
3.7 Sprite	46
3.7.1 Описание модуля	46
3.7.1.1 Конструктор и поля модуля	46
3.7.2 Методы	46
3.7.2.1 Set_tag	46
3.7.2.2 Set_z	47
3.7.2.3 Get_tag	47
3.7.2.4 Set_coords	47
3.7.2.5 Update	47
3.8 Graphics	47
3.8.1 Описание модуля	47
3.8.1.1 Конструктор и поля модуля	47
3.8.2 Методы	47
3.8.2.1 Add_sprite	47
3.8.2.2 Update	47
3.8.2.3 Change_sprite	48
3.8.2.4 Delete_sprite	48
3.8.2.5 Clear_all	48
3.9 Animation(Sprite)	48
3.9.1 Описание модуля	48
3.9.1.1 Конструктор и поля модуля	48
3.9.2 Методы	49
3.9.2.1 Update:	49
3.10 Object	49

3.10.1 Описание модуля	49
3.10.1.1 Конструктор и поля модуля	49
3.10.2 Методы	50
3.10.2.1 Set_state	50
3.10.2.2 Actor_in	50
3.10.2.3 Update	50
3.11 Actor(Object)	50
3.11.1 Описание модуля	50
3.11.1.1 Конструктор и поля модуля	50
3.11.2 Методы	51
3.11.2.1 update	51
3.11.2.2 Search_position	51
3.11.2.3 Stop_move	51
3.12 Adnd_actor(Actor)	51
3.12.1 Описание модуля	51
3.12.1.1 Конструктор и поля модуля	51
3.12.2 Методы	51
3.12.2.1 Click	51
3.12.2.2 Attack	52
3.12.2.3 Update	52
3.13 Rectangle	52
3.13.1 Описание модуля	52
3.13.1.1 Конструктор и поля модуля	52
3.13.2 Методы	52
3.13.2.1 Is_in	52
3.13.2.2 Is_point_inside	52
3.14 Portal(Object)	53
3.14.1 Описание модуля	53
3.14.1.1 Конструктор и поля модуля	53
3.14.2 Методы	53
3.14.2.1 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	66
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	66
ПРИЛОЖЕНИЕ А Представление графического материала	69
ПРИЛОЖЕНИЕ Б Фрагменты исходного кода программы	78
На отдельных листах (CD-RW в прикрепленном конверте)	102
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 12 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.

*В 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 первоходцами 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 всеми же настольками. Если за игровым столом в компании друзей подробности приключения и игровой мир в деталях рисовало воображение игроков, и лишь оно ограничивало пределы игры, то скучная презентация этих ранних видеоигровых экспериментов и близко не давала такого опыта. Тем более речь 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 отыгрываться 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 текстоцентрических игр: от RPG с объемными диалогами до интерактивных сериалов и даже визуальных новелл. Она могла бы быть стандартной адвентурой 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 издаатель. Смешные по сегодняшним меркам продажи в 30 тысяч копий обрекли *Akalabeth* на сиквел, а ее автора — на профессию игрового разработчика. Так началась многолетняя история одной величайших игровых серий прошлого — *Ultima*.

Благодаря развитию технологий, большему бюджету и поддержке издателя Гэрриот сумел в кратчайшие сроки значительно улучшить техническую составляющую игры — вышедшая спустя год *Ultima* обзавелась тайловой графикой, а для управления персонажем больше не нужно было вводить текстовые команды — до-

статочно 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 сдобрали обширной системой магии, среди которой было место как прямо атакующим заклинаниям, так и различным дебаффам. А еще разработка Sir-Tech была беспощадно сложной: подобно Rogue в случае смерти партии игроку ничего не оставалось, кроме как начать с нуля

### 1.3 Япония и её JRPG

В 1986 году отобранная по конкурсу компанией Enix команда молодых и амбициозных японских технарей во главе с Юдзи Хории разработала и выпустила первую в истории JRPG под названием Dragon Quest. Именно эта игра сформировала основные правила поджанра на десятилетия вперед: вид сверху, более-менее свободное исследование огромного мира, состоящего из квадратных тайлов, случайные встречи, пошаговый бой, отдельное окно для сражений с изображением противника и

списком возможных действий, а также большой акцент на линейное повествование с неизменными тропами: древнее зло, магические артефакты, спасение принцессы... Здесь же любители RPG впервые столкнулись с около-анимешной эстетикой, за которую отвечал специально привлеченный в качестве художника известный мангака Акира Торияма.

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

## 1.4 Популярные RPG студии SSI

Главным же поставщиком RPG на грани десятилетий стала компания SSI. В 1988 году ее президент Джоэл Биллингс ввязался в крупнейшую авантюру своей жизни: в жесточайшей конкуренции за огромные деньги выкупил официальную лицензию на создание игр по обновленной редакции легендарной настольки Advanced Dungeons and Dragons. В следующие пять лет SSI выпустила целых 12 компьютерных ролевых игр, вошедших в историю под общим именем Gold Box. Откровенно говоря, большая их часть не изобретала велосипеда. Они лишь довели знакомую жанровую схему предшественниц до совершенства и сопроводили ее достаточным количеством оригинального контента — врагов, квестов, оружия, элементов окружения. Из важных деталей стоит отметить возможность избежать сражения с врагом путем дипломатии (для этого необходимо было выбрать правильный тон разговора)

и функцию быстрого перемещения с помощью раскинувшейся по игровому миру сети телепортов. Лицензия DnD распространялась и на использование различных сеттингов настольки, 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 стablyно 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 настоящий переворот в фанатских кругах и предопределила формирование сообщества моддеров. Не менее важным событием стал выход в 1991-м Neverwinter Nights. Сейчас эту игру затмил другой релиз под таким же названием, случившийся уже в следующем веке, но в истории индустрии она останется навсегда.

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

## **2 Техническое задание**

### **2.1 Основание для разработки**

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

### **2.2 Цель и назначение разработки**

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

Данный программный продукт предназначен для демонстрации практических навыков, полученных в течение обучения. Исходя из этого, основную цель предлагается рассмотреть в разрезе двух групп подцелей.

Задачами данной разработки являются:

- создание персонажа игрока.
- реализация сражения персонажа игрока с монстрами.
- реализация передвижения персонажа игрока по подземелью.

### **2.3 Требования пользователя к движку**

движок должен включать в себя:

- генерацию локаций.
- генерацию персонажа.
- генерацию монстров.

### **2.4 Правила 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 двух-

мерном виде сверху с использованием экранной графики персонажей и управляет-ся 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, следя определенным событиям и заданиям

– 2. Боевая система: Боя 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 экипировку

– 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 развиваются.

Основные элементы ролевой системы DnD включают:

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

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

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

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

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

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

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

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

– 2. Характеристики:

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

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

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

\* Телосложение - характеристика влияющая на 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, сра-

жение 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 Игровой мир Фаэрун

Континент включает в себя самые разнообразные территории. Помимо береговых линий на западе и на юге, основной особенностью континента является Море Падающих Звезд. Это несимметричное море, которое орошает внутренние земли и соединяет западные и восточные регионы Фаэруна, а также является главным торговым маршрутом для многих наций. Регион дикой местности, тяжелых погодных условий, орд орков и диких варварских племен, В основном регион называют просто "Север" также имея в виду северную часть Побережья Мечей. Побережье Мечей пролегает вдоль Моря Мечей, от северной границы Амна до Моря Движущегося Льда, преимущественно занимается городами-государствами, использующими море в торговых целях. Границами региона обычно считают города Невервинтер на севере и Врата Балдура на юге, но и земли к северу и югу от них, не находящиеся под контролем каких-либо более влиятельных сил, часто тоже включаются в карты Побережья Мечей. Регион Севера при этом, являясь более широкой географической областью, включает в себя всё к северу от Амна, и разделяется на два основных региона: Западное Сердцеzemье и Дикую Границу. Западное Сердцеzemье включает в себя узкую полосу цивилизации между Горами Заката и Морем Мечей, и к северу, от Тролльих Гор и Облачных Пиков до Торгового Пути. К Дикуй Границе относится весь остальной Север, состоящий из совсем незаселенных либо скучно заселенных земель, не включая крупные города и различные мелкие поселения, находящиеся в

их непосредственной сфере влияния. Большинство поселений, наций и государств Севера могут быть отнесены к одной из пяти категорий: члены Альянса Лордов, дварфские крепости, островные государства, независимые королевства, разбросанные по побережью и глубины Подземья. По большей части это дикие земли и неисследованные земли, лежащие между большой пустыней Анаурох на востоке и крупным регионом Побережья Мечей на западе, южной границей которых считается Высокая Пустошь.

## 2.7 монстры мира Фаэрун

- Скелет
- тип существа: нежить, размер: средний.
- показатель опасности 1/4(50 опыта).
- класс доспеха 12, здоровье 13 единиц.
- скорость 30 футов.
- характеристики: СИЛ 10(+0) ЛОВ 14(+2) ТЕЛ 15(+2) ИНТ 6(-2) МУД 8(-1) ХАР 5(-3).

- уязвимость к урону: дробящий.
- иммунитет к урону: яд.
- чувства: пассивное восприятие 9.
- 1.1 Действия:
  - Короткий меч. Рукопашная атака оружием: +4 к попаданию, досягаемость 5 футов, одна цель. Попадание: Колющий урон 5 (1к6 + 2).

- Людоед
- тип существа: гуманоид, размер: средний.
- показатель опасности 1/2(100 опыта).
- класс доспеха 14, здоровье 22 единиц.
- скорость 30 футов.
- характеристики: СИЛ 15(+2) ЛОВ 10(+0) ТЕЛ 13(+1) ИНТ 7(-2) МУД 12(+1) ХАР 7(-2).

- иммунитет к урону: яд.
- чувства: пассивное восприятие 11.

- 2.1 Действия: Мультиатака. Людоящер совершает две рукопашные атаки.
  - Укус. Рукопашная атака оружием: +4 к попаданию, досягаемость 5 футов, одна цель. Попадание: Колющий урон 5 (1к6 + 2).
    - Бурый медведь
    - тип существа: Зверь, размер: Большой.
    - показатель опасности 1(200 опыта).
    - класс доспеха 11, здоровье 34 единиц.
    - скорость 40 футов.
    - характеристики: СИЛ 19(+4) ЛОВ 10(+0) ТЕЛ 16(+3) ИНТ 2(-4) МУД 13(+1)
- ХАР 7(-2).
- чувства: пассивное восприятие 11.
  - 3.1 Действия: Мультиатака. Медведь совершает две атаки: одну укусом, и одну когтями.
    - Укус. Рукопашная атака оружием: +6 к попаданию, досягаемость 5 футов, одна цель. Попадание: Колющий урон 8 (1к8 + 4).
    - Когти. Рукопашная атака оружием: +6 к попаданию, досягаемость 5 футов, одна цель. Попадание: Рубящий урон 11 (2к6 + 4).
- Огр
  - тип существа: монстр, размер: Большой.
  - показатель опасности 2(450 опыта).
  - класс доспеха 11, здоровье 59 единиц.
  - скорость 40 футов.
  - характеристики: СИЛ 19(+4) ЛОВ 8(-1) ТЕЛ 16(+3) ИНТ 5(-3) МУД 7(-2)
- ХАР 7(-2).
- чувства: тёмное зрение 60 футов, пассивное восприятие 8.
  - 4.1 Действия:
    - Палица. Рукопашная атака оружием: +6 к попаданию, досягаемость 5 футов, одна цель. Попадание: 13 (2к8 + 4) дробящего урона.

## **2.8 Интерфейс пользователя**

- 1. Генерация персонажа. Пользователь видит перед собой окно с кнопкой "Создать персонажа". Кликнув на неё, на экране появится лейбл с текстом "Выбирайте расу" и 3 кнопки с названиями: "Человек" "Эльф" "Дварф". После
  - 2. Интерфейс пользователя будет состоять из окна, на котором будет отображаться персонаж и локация, на которой он находится. В левом нижнем углу окна будет находиться красная полоска отображающая здоровье персонажа. Справа от неё будет находиться кнопка с названием "Инвентарь" нажав на которую откроется новое окно "Инвентаря". В нём будет список предметов которыми сейчас владеет персонаж. Справа от кнопки "инвентаря" будет находиться кнопка "Заклинания". Кликнув на неё открывается окно, в котором будет представлен список всех известных на данный момент заклинаний, а так же количество ячеек, которые персонаж может израсходовать, на применение заклинаний. Справа от кнопки "Заклинания" будет находиться кнопка "Особенности и черты". Кликнув на неё открывается окно, в котором в виде списка будут перечислены все особенности расы и черты класса выбранные пользователем, а так же уровень персонажа. Справа от клавиши "Особенности и черты" будут находиться 4 кнопки: "Взаимодействовать" "Атаковать" "Сотворить заклинание" "Добавить".
  - 3. Пользователь будет мышкой водить по видимым объектам, на экране. Наводя мышь на объект и нажав ЛКМ, пользователь сможет с ним взаимодействовать. Если это объект окружения, то пользователь может нажать на кнопку "Взаимодействовать" и откроется окно взаимодействия с объектом. Если это монстр, то пользователь может нажать на кнопку "Атаковать" или "Сотворить заклинание". В случае "Атака" откроется окно инвентаря, и пользователь должен будет выбрать каким оружием он совершил атаку. В случае "Сотворить заклинание" откроется окно "Заклинания" и пользователю будет нужно выбрать одно из списка заклинаний. Если это НПС, то пользователь может нажать кнопку "Взаимодействовать" и откроется окно взаимодействия с НПС.

## 2.9 Пример игры

Игрок создаёт своего персонажа, выбирая один из возможных классов воин, волшебник, следопыт и т.д. Затем он выбирает расу своему персонажу: человек, dwarf, эльф и т.д. Так игрок создал своего человека воина 1 уровня. После этого персонаж появляется в стартовой локации. В ней находится NPC: человек-страж по имени Даниэль, гном-торговец Владимир. Игрок может подойти к Даниэлю и кликнуть по нему, чтобы начать диалог, на экран выводится сообщение со следующим текстом: "Исследователь подземелий, мы ждали тебя, это одно из многих подземелий Ацецера. Этот могучий лич хранит в этом месте множество могучих артефактов и горы несметных сокровищ. По приказу короля Персиала Дэ Ролло третьего, мы созываем искателей приключений, чтобы очистить это место от бродящих по нему монстров. Искатель здесь ты можешь отдохнуть, перед входом в подземелье, так же ты можешь покупать и продавать драгоценности и вещи у Владимира. Вот он". После этого у игрока появляются квесты: 1) "пообщаться с Владимиром"2) "Первые изучения". Игрок передвигает персонажа к гному, кликает по нему, на экран выводится сообщение со следующим текстом:"Здравствуй, дорогой друг, я Владимир местный торговец, у меня ты можешь купить снаряжение или продать предметы найденные в подземелье". После этого квест "Пообщаться с Владимиром" будет выполнен. Персонаж получит за это 25 опыта. После игрок нажмёт на левой кнопкой мыши на иконку входа в подземелье. Персонаж игрока перенесётся внутрь подземелья. В этой локации будет пустая комната с двумя проходами, игрок подойдёт к проходу номер 1, персонаж переместился в новую локацию(комнату). В ней находится скелет, начинается боевая ситуация. Игровая система бросает инициативу за скелета и персонажа игрока, первым ходит персонаж игрока. Игрок перемещает персонажа левым щелчком мыши на 6 клеток к скелету, а затем нажимает на кнопку атака мечом. Боевая система делает скрытый бросок на атаку мечом по скелету, попадание, боевая система делает бросок на урон длинным мечом, 11 рубящего урона, здоровье скелета падает до нуля. Боевая ситуация заканчивается. Игрок нажимает левой кнопкой мыши на кости побеждённого скелета. К персонажу в инвентарь, добавляется "ржавый короткий меч". После этого игрок замечает 2 прохода

в этой комнате, проход номер 1, тот через который игрок прошёл в эту комнату, проход номер 2 ведёт в неизвестную комнату. Так же квест "Первые изучения" считается выполненным. Персонаж игрока получает 50 опыта за победу над скелетом, и ещё 100 опыта за выполнение квеста. Игрок перемещает персонажа вправо, а затем кликает на проход номер 1, после чего перемещается в первую комнату. Затем подходит к проходу номер 2, кликает по нему и выходит из подземелья. Когда он оказывается на стартовой локации, игрок кликает по Даниэлю и тот выдаёт ему квест "Убейте трёх скелетов в подземелье".

## **2.10 Моделирование вариантов использования**

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

1. Создание персонажа.
2. Боевая система.
3. Развитие персонажа.
4. Исследование мира.

## **2.11 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 интерпретируемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потреб-

ление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как C или C++.

### **3.2.3 Язык программирования Python**

#### **3.2.3.1 Достоинства языка Python**

- Простота и читаемость кода: Python использует простой и чистый синтаксис, что делает код легким для понимания и обслуживания.
- Многофункциональность: Python подходит для создания различных типов приложений, включая веб-приложения, настольные приложения, научные вычисления, обработку данных и многое другое
- Большой выбор библиотек: Python имеет огромное сообщество разработчиков, что приводит к большому количеству библиотек и модулей для различных задач. Например, для машинного обучения есть библиотека TensorFlow, для веб-разработки - Django, для анализа данных - Pandas и многое другое.
- Кроссплатформенность: Python работает на различных операционных системах, таких как Windows, macOS, Linux и другие.
- Быстрая разработка: Python позволяет быстро создавать прототипы и тестировать идеи благодаря своей простоте и мощности.

#### **3.2.3.2 Недостатки языка Python**

- Низкая производительность: Python может быть медленнее других языков программирования, таких как C++ или Java, особенно при выполнении вычислительно сложных операций.
- Глобальная блокировка интерпретатора: из-за глобальной блокировки GIL (Global Interpreter Lock) в Python, многопоточные приложения могут испытывать проблемы с параллельным выполнением кода.
- Не самый подходящий для мобильной разработки: Python не является первым выбором для мобильной разработки из-за ограниченной поддержки на мобильных платформах.

- Не все библиотеки могут быть на Python: Так как Python находится в постоянном развитии, не все библиотеки могут быть доступны на этом языке.
- Меньшая поддержка для некоторых областей разработки, таких как игровая разработка или высокопроизводительные вычисления.

### **3.2.4 Использование библиотеки Tkinter и реализация таймеров на Python**

#### **3.2.4.1 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.

#### **3.2.4.2 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.

#### **3.2.4.3 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.

### 3.2.4.4 Заключение

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

## 3.3 Описание платформы для создания RPG игр

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

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

А вообще сценарии и глобальные переменные могут быть без классов, а просто в модулях, так проще, чтобы к ним был доступ из всех комнат. Тогда и функции движка должны быть доступны везде (то есть во всех сценариях). Например делаем модуль руины (ruins):

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

```
class Ruins(Area): def __init__(self): """
    Класс игровой зоны Ruins
    """
    super().__init__()
    self.add_sprite(Sprite('images/fon3.png'), 590, 400, 0)
    self.add_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.width(),
```

```
height=Sprite('images/fon3.png').image.height())) from grunt import Grunt self.grunt = Grunt(0,0,0) from footman import Footman self.footman = Footman(0,0,0) self.add_object(self.footman, 120, 120, 1) self.add_object(self.grunt, 500, 185, 1) p = Portal(400, 400, 200, 200, 'Village', 480, 100) self.add_object(p, p.pos_x, p.pos_y, 100) Game.game.start_script(self.ai, "ai self.grunt") Game.game.start_script(self.walk_two, "footman 50, 50)
```

```
def walk(self, step_x, step_y, actor): """ Сценарий для движения бугая
:param step_x: шаг движения x :param step_y: шаг движения y """
if actor.hp <= 0: Game.game.stop_script("grunt") new_x = 200 new_y = 200 actor.is_attack = False direction = random.choice(["up " "down " "left " "right"]) if direction == "up": new_y -= step_y new_x = step_x elif direction == "down": new_y += step_y new_x = step_x elif direction == "left": new_y = step_y new_x -= step_x elif direction == "right": new_y = step_y new_x += step_x
actor.search_position(new_x, new_y)
time.sleep(2)
```

```
модуль bggame: from ruins import * import time import random
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 выводится игра """
super().__init__(canvas, window, **params) from
mage import Mage self.add_pc_to_team(Mage(0, 0, 0)) self.new_area('Ruins', Ruins())
self.set_area('Ruins') self.set_team(500, 300, 100) self.timer()
```

### 3.3.1 Пример клиентского кода игры

#### 3.3.1.1 Создание классов персонажей/предметов

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

```
модуль bggame: from ruins import * import time import random
class BaldursGame(Game): def __init__(self, canvas, window, **params): """ Класс
конкретной игры для демонстрации
```

```

:param canvas: класс графической системы :param window: окно на которое будет выводится игра """
super().__init__(canvas, window, **params)
from mage import Mage
self.add_pc_to_team(Mage(0, 0, 0))
self.new_area('Ruins', Ruins())
self.set_area('Ruins')
self.set_team(500, 300, 100)
self.timer()

```

### 3.3.1.2 Задание правил атаки

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

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

```

class Adnd_actor(Actor):
    ATTACK_RANGE = 50
    def __init__(self, x, y, z, **params):
        """ Класс Adnd_actor содержащий основные механики взаимодействия с другими персонажами
        :param x: координата x :param y: координата y :param z: координата z """
        super().__init__(x, y, z, **params)
        self.on_click = self.click
    def click(self):
        """ вызывается при клике на персонажа
        """
        pc = rpg.game.Game.game.team_of_pc[0]
        if pc == self:
            return
        dx = pc.pos_x - self.pos_x
        dy = pc.pos_y - self.pos_y
        dist = sqrt(dx * dx + dy * dy)
        if dist <= self.ATTACK_RANGE:
            pc.is_attack = True
            pc.attack(self)
            time.sleep(0.125)
            if self.hp <= 0:
                pc.is_attack = False
    def attack(self, actor):
        """ совершает атаку по actor
        :param actor: персонаж, которого атакуют """
        actor.hp -= self.damage
    def update(self):
        """ обновляет состояние персонажа
        """
        super().update()
        if self.hp <= 0:
            self.stop_move()
            self.set_state('death')

```

### 3.3.1.3 Создание зон, заполнение их персонажами/объектами

Мир также состоит из зон (Area). Каждая зона включает в себя графику, персонажи и предметы и сценарии взаимодействия. Исключение составляет команда PC, которая может перемещаться из зоны в зону (это мы программируем у клиента). Как

программируются зоны. Если нужны локальные переменные (состояние локальных событий), то тогда нужно создавать класс своей зоны как наследник от Area. Или же просто использовать класс Area. Добавляем зону в игру new\_area(name, area). Переключаем зону - set\_area(name). Так же требуется задать область движения, её проще сделать как совокупность прямоугольников, за которые персонажи не могут выйти. Эти прямоугольники должны касаться друг друга, но не пересекаться. Тогда алгоритм проверки выхода несложный: выход за пределы области только тогда, когда прямоугольник персонажа пересек сторону (одну или две) одного из прямоугольников области, эта сторона не является касательной.

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

```
class Ruins(Area): def __init__(self): """ Класс игровой зоны Ruins
    """ super().__init__() self.add_sprite(Sprite('images/fon3.png'), 590, 400, 0)
self.add_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.width(),
height=Sprite('images/fon3.png').image.height())) from grunt import Grunt self.grunt =
Grunt(0,0,0) from footman import Footman self.footman = Footman(0,0,0)
self.add_object(self.footman, 120, 120, 1) self.add_object(self.grunt, 500, 185, 1) p =
Portal(400, 400, 200, 200, 'Village', 480, 100) self.add_object(p, p.pos_x, p.pos_y, 100)
Game.game.start_script(self.ai, "ai self.grunt") Game.game.start_script(self.walk_two,
"footman 50, 50)
```

```
def walk(self, step_x, step_y, actor): """ Сценарий для движения бугая
:param step_x: шаг движения x :param step_y: шаг движения y """
if actor.hp <= 0: Game.game.stop_script("grunt") new_x = 200 new_y = 200 actor.is_attack =
False direction = random.choice(["up "down "left "right"]) if direction == "up": new_y -= step_y new_x = step_x elif direction == "down": new_y += step_y new_x = step_x elif direction == "left": new_y = step_y new_x -= step_x elif direction == "right": new_y = step_y new_x += step_x
actor.search_position(new_x, new_y)
time.sleep(2)
```

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

### 3.3.1.4 Пример сценариев: переход между зонами

Глобальные сценарии находятся в классе игры (BGGame), мы подключаем их как : Area.set\_enter\_script(script) Как происходит переход команды между зонами. В зоне определяем объект портал, по клику мыши когда персонаж заходит внутрь портала срабатывает self.actor\_in(self, actor). При создании портала, мы указываем кудаи в какую зону разместить команду персонажей.

```
from rpg.object import Object from rpg.game import Game from rpg.rectangle
import Rectangle
```

```
class Portal(Object): def __init__(self, x, y, width, height, area, team_x, team_y): ""
```

Создает портал в новую зону

```
:param x: координата x портала :param y: координата y портала :param width:
ширина портала :param height: высота портала :param area: имя зоны куда будет пе-
реход :param team_x: местоположение команды в новой зоне :param team_y: мест-
положение 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.height())) from grunt import Grunt self.grunt
= Grunt(0,0,0) from footman import Footman self.footman = Footman(0,0,0)
self.add_object(self.footman, 120, 120, 1) self.add_object(self.grunt, 500, 185, 1) p
= Portal(400, 400, 200, 200, 'Village', 480, 100)
```

### 3.3.1.5 Как будет идти бой

Бой будет совершаться с помощью сценариев. У класса Adnd\_actor есть метод attack(self, actor), который уменьшает текущее количество здоровья у actor. В модуле game существуют методы start\_script(script, name), stop\_script(name). С помощь сценариев возможно запускать параллельные потоки. В конкретную зону будет добавляться сценарий 'ai', в который передаётся конкретный персонаж. В этом сценарии указывается поведение противника, Что он должен сближаться с персонажем игрока, и когда расстояние до атаки будет достаточным, чтобы её совершить, будет вызван метод actor.attack. Для того, чтобы пользователь мог атаковать персонажа, у каждого экземпляра класса adnd\_actor есть метод click(self), который вызывает проверку условия, если персонаж близко к персонажу игрока, хранящемуся в rpg.game.Game.team\_of\_pc, то вызвать у pc=rpg.game.Game.team\_of\_pc[0], attack(self)/ Пример: модуль adnd\_actor: from math import sqrt from rpg.actor import Actor from rpg.animation import Animation import rpg.game import time

```
class Adnd_actor(Actor):
    ATTACK_RANGE = 50
    def __init__(self, x, y, z, **params):
        """ Класс Adnd_actor содержащий основные
        механики взаимодействия с другими персонажами
        :param x: координата x :param y: координата y :param z: координата z """
        super().__init__(x, y, z, **params)
        self.on_click = self.click
    def click(self):
        """ вызывается при клике на персонажа
        """
        pc = rpg.game.Game.team_of_pc[0]
        if pc == self:
            return
        dx = pc.pos_x - self.pos_x
        dy = pc.pos_y - self.pos_y
        dist = sqrt(dx * dx + dy * dy)
        if dist <= self.ATTACK_RANGE:
            pc.is_attack = True
            pc.attack(self)
            time.sleep(0.125)
            if self.hp <= 0:
                pc.is_attack = False
    def attack(self, actor):
        """ совершает атаку по actor
        :param actor: персонаж, которого атакуют """
        actor.hp -= self.damage
    def update(self):
        """ обновляет состояние персонажа
        """
        super().update()
        if self.hp <= 0:
            self.stop_move()
            self.set_state('death')
```

```

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

class Ruins(Area):
    def __init__(self):
        """ Класс игровой зоны Ruins """
        super().__init__()
        self.add_sprite(Sprite('images/fon3.png'), 590, 400, 0)
        self.add_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.width(),
                               height=Sprite('images/fon3.png').image.height()))
        from grunt import Grunt
        self.grunt = Grunt(0,0,0)
        from footman import Footman
        self.footman = Footman(0,0,0)
        self.add_object(self.footman, 120, 120, 1)
        self.add_object(self.grunt, 500, 185, 1)
        p = Portal(400, 400, 200, 200, 'Village', 480, 100)
        self.add_object(p, p.pos_x, p.pos_y, 100)
        Game.game.start_script(self.ai, "ai self.grunt")
        Game.game.start_script(self.walk_two, "footman 50, 50")

    def walk(self, step_x, step_y, actor):
        """ Сценарий для движения бугая """
        :param step_x: шаг движения x :param step_y: шаг движения y """
        if actor.hp <= 0:
            Game.game.stop_script("grunt")
            new_x = 200
            new_y = 200
            actor.is_attack = False
            direction = random.choice(["up", "down", "left", "right"])
            if direction == "up":
                new_y -= step_y
                new_x = step_x
            elif direction == "down":
                new_y += step_y
                new_x = step_x
            elif direction == "left":
                new_y = step_y
                new_x -= step_x
            elif direction == "right":
                new_y = step_y
                new_x += step_x
            actor.search_position(new_x, new_y)

    def ai(self, actor):
        """ скрипт противников """
        :param step_x: размер шага x до персонажа игрока :param step_y: размер ша-
        га x до персонажа игрока :param actor: персонаж противник """
        if actor.hp <= 0:
            Game.game.stop_script("ai")
            import rpg.game
            pc = rpg.game.Game.game.team_of_pc[0]
            new_x = pc.pos_x
            new_y = pc.pos_y
            actor.search_position(new_x, new_y)
            dx = pc.pos_x - actor.pos_x
            dy = pc.pos_y - actor.pos_y
            dist = sqrt(dx * dx + dy * dy)
            if dist <= actor.ATTACK_RANGE:
                actor.is_attack = True
                actor.attack(pc)
                time.sleep(1)
                if pc.hp <= 0:
                    actor.update()
            Game.game.stop_script("ai")
            Game.game.start_script(self.walk, "grunt 50, 50, actor")
        else:
            actor.is_attack = False
            time.sleep(2)

```

### 3.3.1.6 Соединение движка и окон tkinter

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

модуль `sprite`

```
import tkinter as tk class Sprite:
```

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

```
        :param image: адресс изображения который """ self.image =  
tk.PhotoImage(file=image) self.tag = None self.x = 0 self.y = 0 self.z = 0
```

```
        def set_tag(self, tag): """ Устанавливает тег спрайта
```

```
        :param tag: тег спрайта """ self.tag = tag
```

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

```
        :param z: координата z """ self.z = z
```

```
        def get_tag(self): """ Возвращает тег спрайта
```

```
        """ return self.tag
```

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

```
        :param new_x: координата x :param new_y: координата y """ if self.tag: self.x =  
new_x self.y = new_y def update(self): """ Обновляет анимацию спрайта
```

```
        """ pass
```

модуль `graphics`

```
import tkinter as tk
```

```
class Graphics(tk.Canvas): canvas = None def __init__(self, master, **kwargs): """
```

Класс с методами для работы со спрайтами

```
        """ super().__init__(master, **kwargs) self.sprites = [] Graphics.canvas = self
```

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

```
        :param sprite: спрайт :param x: координата x :param y: координата y :param z:  
координата z :param kwargs: параметры относящиеся к конкретному изображению
```

```

в tkinter """ tag = self.create_image(x, y, image=sprite.image, anchor='center', **kwargs)
sprite.set_tag(tag) sprite.set_z(z) sprite.x = x sprite.y = y self.sprites.append(sprite)
self.sprites.sort(key=lambda sprite: sprite.z)

def update(self): """ Перерисовывает все спрайты
    """ for sprite in self.sprites: sprite.update() self.tag_raise(sprite.get_tag())
self.coords(sprite.get_tag(), sprite.x, sprite.y) self.itemconfig(sprite.get_tag(),
image=sprite.image)

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

if old_sprite_pos is not None: old_tag = sprite.get_tag()
self.sprites[old_sprite_pos] = new_sprite new_sprite.set_tag(old_tag)
new_sprite.set_tag(old_tag) new_sprite.set_z(sprite.z)
self.tag_raise(old_tag) self.coords(old_tag, sprite.x, sprite.y)
self.itemconfig(old_tag, image=new_sprite.image)

def delete_sprite(self, sprite): """ Удаляет спрайт с Canvas.
:param sprite: экземпляр спрайта :return: """
self.delete(sprite.get_tag())
self.sprites.remove(sprite)

def clear_all(self): """ Удаляет все спрайты с Canvas
""" for sprite in self.sprites: self.delete(sprite.get_tag())
self.sprites.clear()

модуль baldursgame """пользовательский модуль"""
from ruins import * from village import * import time import random

class BaldursGame(Game):
    def __init__(self, canvas, window, **params): """ Класс
конкретной игры для демонстрации

:param canvas: класс графической системы :param window: окно на кото-
рое будет выводится игра """
super().__init__(canvas, window, **params)
from mage import Mage
self.add_pc_to_team(Mage(0, 0, 0))
self.new_area('Ruins', Ruins())
self.new_area('Village', Village())
self.set_area('Ruins')
self.set_team(500, 300, 100)
self.timer()

модуль main from bggame import *

```

```

root = tk.Tk() root.geometry('1500x1500')
exit_button = tk.Button(root, text="Exit fg="red command=root.destroy) canvas =
Graphics(root, width=1500, height=1500) Graphics.canvas = canvas
BaldursGame(canvas, root)
canvas.place(height = 1500, width =1500) BaldursGame.timer
root.mainloop()

```

### **3.4 Модули и классы**

### **3.5 Game**

#### **3.5.1 Описание модуля**

##### **3.5.1.1 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

### **3.5.2 Методы**

#### **3.5.2.1 New\_area**

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

#### **3.5.2.2 Set\_area**

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

#### **3.5.2.3 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.

#### **3.5.2.4 Start\_script**

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

#### **3.5.2.5 Stop\_script**

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

### **3.5.2.6 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.

### **3.5.2.7 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.

### **3.5.2.8 Set\_team**

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

### **3.5.2.9 Update**

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

### **3.5.2.10 Mouse\_left\_click**

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

### **3.5.2.11 Timer**

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

## **3.6 Area**

### **3.6.1 Описание модуля**

#### **3.6.1.1 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

### **3.6.2 Методы**

#### **3.6.2.1 Add\_sprite**

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

#### **3.6.2.2 Add\_object**

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

#### **3.6.2.3 Remove\_object**

def remove\_object(self, obj) Описание метода: Удаляет объект из зоны.

#### **3.6.2.4 Load\_sprites**

def load\_sprites(self) Описание метода: Загружает все спрайты зоны.

#### **3.6.2.5 Add\_rect**

def add\_rect(self, rec) Описание метода: Добавляет прямоугольник в зону.

#### **3.6.2.6 Entry\_script**

def entry\_script(self)) Описание метода: Запускается, когда команда входит в зону.

### **3.6.2.7 Exit\_script**

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

### **3.6.2.8 Update**

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

## **3.7 Sprite**

### **3.7.1 Описание модуля**

#### **3.7.1.1 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.

### **3.7.2 Методы**

#### **3.7.2.1 Set\_tag**

`def set_tag(self, tag)`

### **3.7.2.2 Set\_z**

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

### **3.7.2.3 Get\_tag**

```
def get_tag(self)
```

### **3.7.2.4 Set\_coords**

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

### **3.7.2.5 Update**

```
def update(self)
```

## **3.8 Graphics**

### **3.8.1 Описание модуля**

#### **3.8.1.1 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

### **3.8.2 Методы**

#### **3.8.2.1 Add\_sprite**

```
def add_sprite(self, sprite, x, y, z, **kwargs)
```

Описание метода: Добавляет спрайт на Canvas

#### **3.8.2.2 Update**

```
def update(self)
```

Описание метода: Перерисовывает все спрайты

### **3.8.2.3 Change\_sprite**

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

### **3.8.2.4 Delete\_sprite**

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

### **3.8.2.5 Clear\_all**

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

## **3.9 Animation(Sprite)**

### **3.9.1 Описание модуля**

#### **3.9.1.1 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`

## 3.9.2 Методы

### 3.9.2.1 Update:

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

## 3.10 Object

### 3.10.1 Описание модуля

#### 3.10.1.1 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
```

## **3.10.2 Методы**

### **3.10.2.1 Set\_state**

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

### **3.10.2.2 Actor\_in**

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

### **3.10.2.3 Update**

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

## **3.11 Actor(Object)**

### **3.11.1 Описание модуля**

#### **3.11.1.1 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

### **3.11.2 Методы**

#### **3.11.2.1 update**

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

#### **3.11.2.2 Search\_position**

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

#### **3.11.2.3 Stop\_move**

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

### **3.12 Adnd\_actor(Actor)**

#### **3.12.1 Описание модуля**

##### **3.12.1.1 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
```

#### **3.12.2 Методы**

##### **3.12.2.1 Click**

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

### **3.12.2.2 Attack**

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

### **3.12.2.3 Update**

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

## **3.13 Rectangle**

### **3.13.1 Описание модуля**

#### **3.13.1.1 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

### **3.13.2 Методы**

#### **3.13.2.1 Is\_in**

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

#### **3.13.2.2 Is\_point\_inside**

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

## 3.14 Portal(Object)

### 3.14.1 Описание 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

### 3.14.2 Методы

#### 3.14.2.1 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 Bitrix, 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.	<code>set_tag(self, tag)</code> Устанавливает тег для спрайта. <code>set_z(self, z)</code> Устанавливает z-координату спрайта. <code>get_tag(self)</code> Возвращает тег спрайта. <code>set_coords(self, new_x, new_y)</code> Обновляет координаты спрайта. <code>update(self)</code> Обновляет анимацию спрайта, если она у него есть.
animation	rpg	Animation – Класс анимации спрайта	<code>update(self)</code> Меняет текущее изображение в списке изображений.

## Продолжение таблицы 4.1

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

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

## Продолжение таблицы 4.1

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

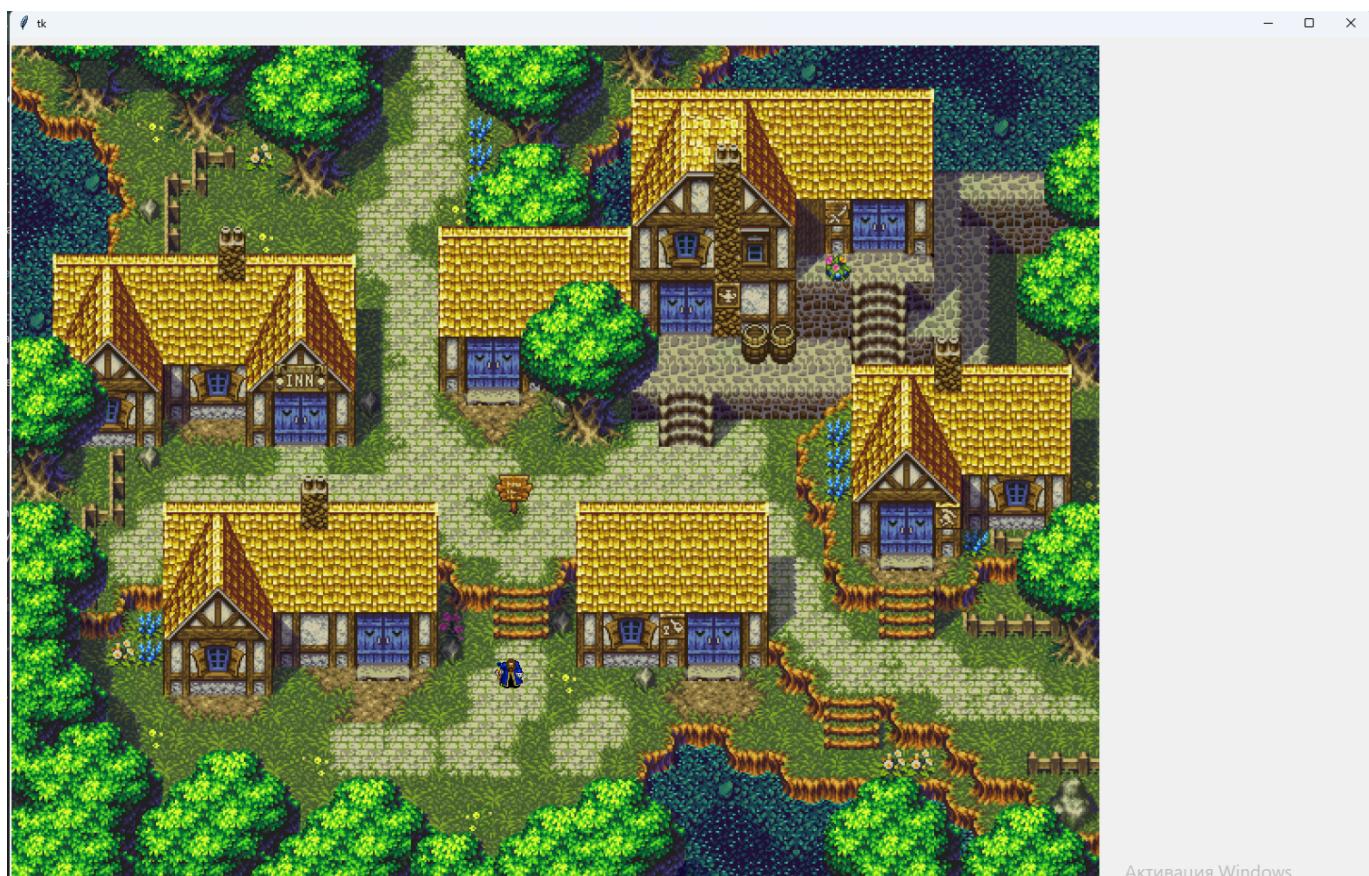


Рисунок 4.2 – Пример работы программы с одним персонажем внутри одной, игровой зоны Village

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.

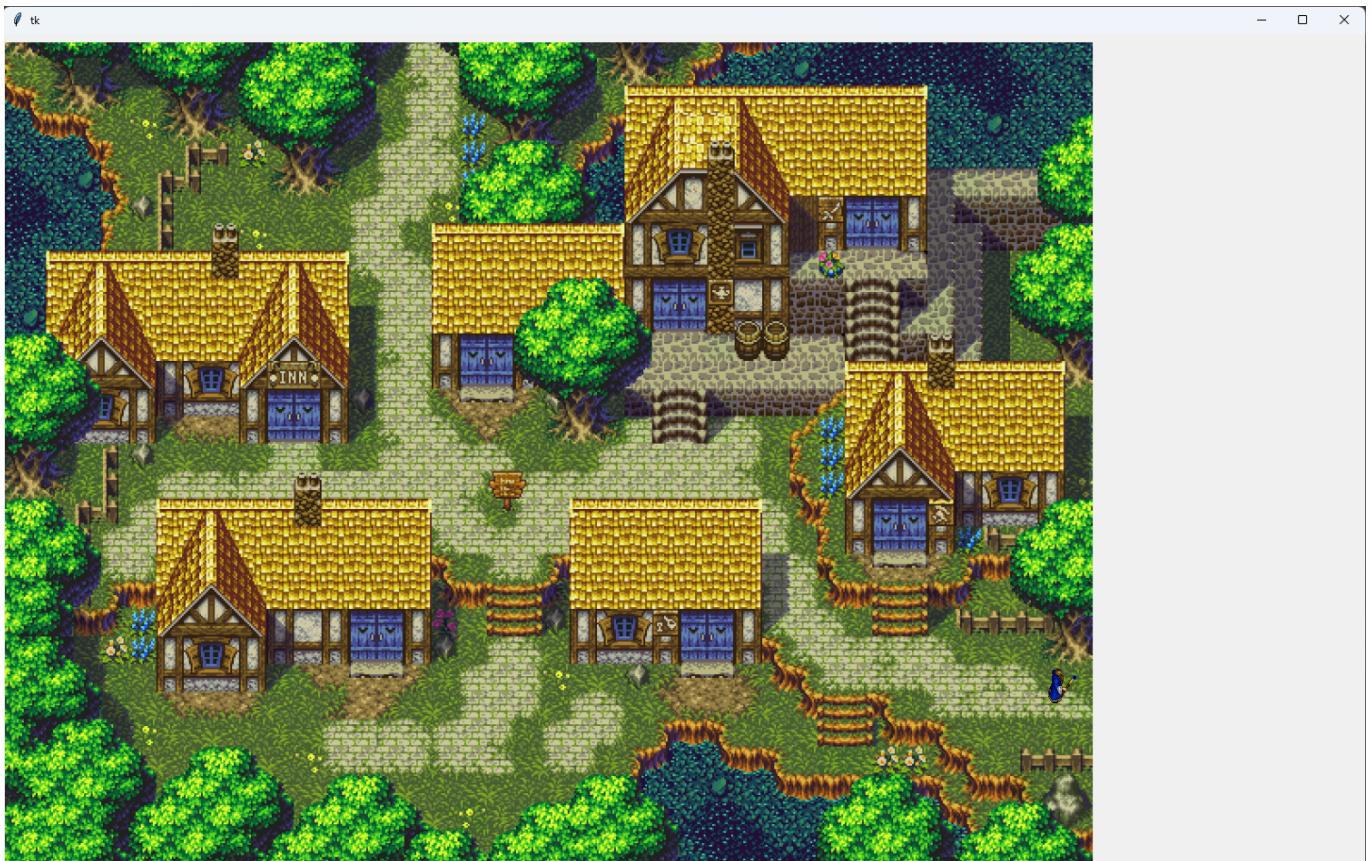


Рисунок 4.5 – Персонаж mage, не может выйти за пределы видимой зоны Village

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 : Бомбара, 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, 3, 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 Климент Николаевич

	ВКРБ 20060139.09.03.04.24.012		
	Сведения о ВКРБ		
Фамилия И. О. Инициалы	Логин	Пароль	Адм. №код / Исполнитель
Шемченко К.Н.			
Руководитель	Члены комиссии		
Коршунова Т.А.	Челебагян А.А.		
			Логин 1 Альянс 5
			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.

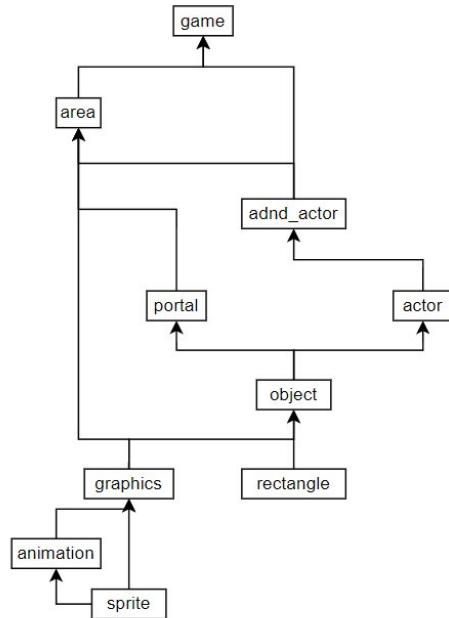
Для достижения поставленной цели требуется решить следующие задачи:

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

	ВКРБ 20060139.09.03.04.24.01
Фамилия И. О. Фамилия/Имя Шевченко К.Н.	Лиц. №паспорт Лиц. 1 Альбина
Автор работы/Руководитель/Координатор Чаплинин А.А.	Цель и задачи разработки
Наименование Чаплинин А.А.	Высокая специализированная работа бакалавра
	ЮЗГУ по-02

## Рисунок А.2 – Цель и задачи разработки

## Концептуальная модель



### Рисунок А.3 – Концептуальная модель приложения

## Диаграмма классов программы



Рисунок А.4 – Диаграмма классов

## Модель работы сценариев

В Python потоки — это легковесные процессы, которые могут выполняться параллельно. В библиотеке `threading` потоки управляются операционной системой, которая решает, когда и как долго каждый поток будет выполняться. Это называется планированием потоков, и оно обычно происходит без вмешательства программиста.

Однако, можно создать модель, которая иллюстрирует переключение между потоками в Python. Представим, что у нас есть три потока: A, B и C. Каждый поток выполняет функцию `worker`, которая занимает определенное время. Планировщик ОС может переключаться между потоками, например, после выполнения каждой инструкции или при блокировке операции ввода-вывода.

Время | Поток А | Поток В | Поток С

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

БКРБ 20060139.09.03.04.21012		
Студент: Чеплыгин К.Н. Автор работ: Чеплыгин К.Н. Руководитель: Чеплыгин А.А. Нормировщик: Чеплыгин А.А.	Модель работы сценариев	Лин. Номе. Номеф
		Лист 1 / Лист 6
		Выпускная квалификационная работа бакалавра
		ЮЗГУ №-026

Рисунок А.5 – Модель работы сценариев

## Модульное тестирование платформы

def test_inside(self): """Тест: прямоугольник внутри другого""" rect_outside = Rectangle(0, 0, 6, 6) self.assertTrue(rect.is_in(rect_outside))	Пройдет успешно
def test_outside(self): """Тест: прямоугольник снаружи другого""" rect_inside = Rectangle(2, 2, 2, 2) self.assertFalse(rect.is_in(rect_inside))	Пройдет успешно
def test_apart(self): """Тест: прямоугольник отдельно от другого""" rect_apart = Rectangle(6, 6, 2, 2) self.assertFalse(rect.is_in(rect_apart))	Пройдет успешно
def test_touching_left(self): """Тест: прямоугольник касается слева""" touching_left = Rectangle(0, 2, 1, 1) self.assertFalse(rect.is_in(touching_left))	Пройдет успешно
def test_touching_right(self): """Тест: прямоугольник касается справа""" touching_right = Rectangle(5, 2, 1, 1) self.assertFalse(rect.is_in(touching_right))	Пройдет успешно
def test_touching_top(self): """Тест: прямоугольник касается сверху""" touching_top = Rectangle(2, 5, 1, 1) self.assertFalse(rect.is_in(touching_top))	Пройдет успешно
def test_touching_bottom(self): """Тест: прямоугольник касается снизу""" touching_bottom = Rectangle(2, 0, 1, 1) self.assertFalse(rect.is_in(touching_bottom))	Пройдет успешно
def test_intersect_left(self): """Тест: пересечение прямоугольника слева""" intersect_left = Rectangle(0, 2, 3, 2) self.assertTrue(rect.is_in(intersect_left))	Пройдет успешно
def test_intersect_right(self): """Тест: пересечение прямоугольника справа""" intersect_right = Rectangle(3, 2, 3, 2) self.assertTrue(rect.is_in(intersect_right))	Пройдет успешно
def test_intersect_top(self): """Тест: пересечение прямоугольника сверху""" intersect_top = Rectangle(2, 3, 2, 3) self.assertTrue(rect.is_in(intersect_top))	Пройдет успешно
def test_intersect_bottom(self): """Тест: пересечение прямоугольника снизу""" intersect_bottom = Rectangle(2, 0, 2, 3) self.assertTrue(rect.is_in(intersect_bottom))	Пройдет успешно
def test_is_point_inside(self): rect = Rectangle(0, 0, 10, 10) """Тест: точка внутри прямоугольника""" self.assertTrue(rect.is_point_inside(5, 5))	Пройдет успешно

ВКРБ 20060139.09.03.04.24.012	
Фамилия И. О. / Имя И. Био.	Англ. Name / Name
Смирнова Н. О. / Наташа Смирнова Автор работ/учебного К.н. Руководитель Чеплыгин А.А. Нормировщик Чеплыгин А.А.	Модульное тестирование платформы
	Лист 1 / Лист 6
	Выпускная квалификационная работа 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.

		БКРБ 20060139.09.03.04.24.012		
		Лист	Номер	
Оценка: К.О / Печать/Факс		Заключение		
Автор работ/дипломчика К.Н. Руководитель Чеплыгин А.А. Нормоконтроль Чеплыгин А.А.		<input type="checkbox"/> Акт 1 <input type="checkbox"/> Акт 2		
		Выпускная квалификационная работа бакалавра ЮЗГУ №-026		

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

```
46 \input{СписокИсточников}
47 \ifБКР{\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 достичь
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 C или C++.
19
20 \subsubsection{Язык программирования Python}
21
22 \paragraph{Достоинства языка Python}
23 \begin{itemize}
24 \item Простота и читаемость кода: Python использует простой и чистый синтаксис,
что делает код легким для понимания и обслуживания.
25 \item Многофункциональность: Python подходит для создания различных типов
приложений, включая веб-приложения, настольные приложения, научные вычисления,
обработку данных и многое другое
26 \item Большой выбор библиотек: Python имеет огромное сообщество разработчиков, что
приводит к большому количеству библиотек и модулей для различных задач.
```

Например, для машинного обучения есть библиотека TensorFlow, для веб-разработки - Django, для анализа данных - Pandas и многое другое.

27 \item Кроссплатформенность: Python работает на различных операционных системах, таких как Windows, macOS, Linux и другие.

28 \item Быстрая разработка: Python позволяет быстро создавать прототипы и тестировать идеи благодаря своей простоте и мощности.

29 \end{itemize}

30

31 \paragraph{Недостатки языка Python}

32

33 \begin{itemize}

34 \item Низкая производительность: Python может быть медленнее других языков программирования, таких как C++ или Java, особенно при выполнении вычислительно сложных операций.

35 \item Глобальная блокировка интерпретатора: из-за глобальной блокировки GIL (Global Interpreter Lock) в Python, многопоточные приложения могут испытывать проблемы с параллельным выполнением кода.

36 \item Не самый подходящий для мобильной разработки: Python не является первым выбором для мобильной разработки из-за ограниченной поддержки на мобильных платформах.

37 \item Не все библиотеки могут быть на Python: Так как Python находится в постоянном развитии, не все библиотеки могут быть доступны на этом языке.

38 \item Меньшая поддержка для некоторых областей разработки, таких как игровая разработка или высокопроизводительные вычисления.

39 \end{itemize}

40

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

42

43 \paragraph{Введение}

44 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.

45

46 \paragraph{Возможности Tkinter}

47 Вот некоторые из основных возможностей, предоставляемых библиотекой Tkinter:

48

49 \begin{itemize}

50 \item Создание различных виджетов: кнопки, метки, поля ввода, списки и многое другое.

51 \item Управление компоновкой виджетов с использованием менеджеров компоновки (например, grid, pack, place).

52 \item Обработка событий, таких как щелчок мыши, нажатие клавиш и другие.

53 \item Возможность создания различных диалоговых окон, таких как окна предупреждений, информационные окна и окна запроса.

54 \item Поддержка многопоточности для обновления интерфейса из различных потоков выполнения.

55 \end{itemize}

56

57 \paragraph{Реализация таймеров на Python}

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

59

```

60 import time
61
62 def countdown(t):
63     while t > 0:
64         mins, secs = divmod(t, 60)
65         timeformat = '{:02d}:{:02d}'.format(mins, secs)
66         print(timeformat, end='\r')
67         time.sleep(1)
68         t -= 1
69     print('Таймер завершен!')
70
71
72 t = 10
73 countdown(t)
74
```

75 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.

76

77 \paragraph{Заключение}

78 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.

79

80 \subsection{Описание платформы для создания RPG игр}

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

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

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

84

85 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 (BGGGame), мы подключаем их как :

86 `Area.set\_enter\_script(script)`

87 В зону мы добавляем персонажей и предметы как `add\_object(x,y, obj)` - z не нужно, так как слой можно определить по у координате.

88 В конкретную зону мы добавляем сценарий для взаимодействия как: `Game.game.start\_script(script, name)`

89 Как происходит переход команды между зонами.

90 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.

91 Сценарии - это потоки которые запускаются параллельно (метод `RPGGame.start\script(script)`). Сценарий может быть остановлен (`stop\script(name)`).

92 Таким образом, мир будет интерактивным.

93 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.

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

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

96

97 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`):

```

98 import random
99 from math import sqrt
100 import time
101 from rpg.area import *
102 from rpg.sprite import *
103 from rpg.rectangle import *
104 from rpg.game import Game
105 from rpg.portal import Portal
106
107 class Ruins(Area):
108     def __init__(self):
109         """
110     Класс игровой зоны Ruins
111
112     """
113     super().__init__()
114     self.add_sprite(Sprite('images/fon3.png'), 590, 400, 0)
115     self.add_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.width(),
116                           height=Sprite('images/fon3.png').image.height()))
117     from grunt import Grunt
118     self.grunt = Grunt(0,0,0)
119     from footman import Footman
120     self.footman = Footman(0,0,0)
121     self.add_object(self.footman, 120, 120, 1)
122     self.add_object(self.grunt, 500, 185, 1)
123     p = Portal(400, 400, 200, 200, 'Village', 480, 100)
124     self.add_object(p, p.pos_x, p.pos_y, 100)
125     Game.game.start_script(self.ai, "ai", self.grunt)
126     Game.game.start_script(self.walk_two, "footman", 50, 50)
127
128     def walk(self, step_x, step_y, actor):
129         """
130     Сценарий для движения бугая
  
```

```

131
132 :param step\_x: шаг движения x
133 :param step\_y: шаг движения y
134 '''
135 if actor.hp <= 0:
136 Game.game.stop\_script("grunt")
137 new\_x = 200
138 new\_y = 200
139 actor.is\_attack = False
140 direction = random.choice(["up", "down", "left", "right"])
141 if direction == "up":
142 new\_y -= step\_y
143 new\_x = step\_x
144 elif direction == "down":
145 new\_y += step\_y
146 new\_x = step\_x
147 elif direction == "left":
148 new\_y = step\_y
149 new\_x -= step\_x
150 elif direction == "right":
151 new\_y = step\_y
152 new\_x += step\_x
153
154 actor.search\_position(new\_x, new\_y)
155
156 time.sleep(2)
157
158 модуль bggame:
159 from ruins import *
160 import time
161 import random
162
163 class BaldursGame(Game):
164 def \_\_init\_\_(self, canvas, window, **params):
165 '''
166 Класс конкретной игры для демонстрации
167
168 :param canvas: класс графической системы
169 :param window: окно на которое будет выводится игра
170 '''
171 super().\_\_init\_\_(canvas, window, **params)
172 from mage import Mage
173 self.add\_pc\_to\_team(Mage(0, 0, 0))
174 self.new\_area('Ruins', Ruins())
175 self.set\_area('Ruins')
176 self.set\_team(500, 300, 100)
177 self.timer()
178
179 \subsubsection{Пример клиентского кода игры}
180 \paragraph{Создание классов персонажей/предметов}
181 Клиент создает модуль содержащий методы модуля RPGGame, например BaldursGateGame. В
этом модуле клиент создаем мир игры, с помощью new\_actor.
182
183 модуль bggame:

```

```

184 from ruins import *
185 import time
186 import random
187
188 class BaldursGame(Game):
189     def __init__(self, canvas, window, **params):
190         """
191         Класс конкретной игры для демонстрации
192
193         :param canvas: класс графической системы
194         :param window: окно на которое будет выводится игра
195         """
196         super().__init__(canvas, window, **params)
197         from mage import Mage
198         self.add_pc_to_team(Mage(0, 0, 0))
199         self.new_area('Ruins', Ruins())
200         self.set_area('Ruins')
201         self.set_team(500, 300, 100)
202         self.timer()
203
204     \paragraph{Задание правил атаки}
205     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.
206     Пример:
207
208     модуль adnd_actor:
209     from math import sqrt
210     from rpg.actor import Actor
211     from rpg.animation import Animation
212     import rpg.game
213     import time
214
215     class Adnd_actor(Actor):
216
217         ATTACK_RANGE = 50
218
219         def __init__(self, x, y, z, **params):
220             """
221             Класс Adnd_actor содержащий основные механики взаимодействия с другими персонажами
222
223             :param x: координата x
224             :param y: координата y
225             :param z: координата z
226             """
227         super().__init__(x, y, z, **params)
228         self.on_click = self.click
229
230     def click(self):
231         """
232             вызывается при клике на персонажа
233
234             """
235         pc = rpg.game.Game.game.team_of_pc[0]

```

```

236 if pc == self:
237     return
238 dx = pc.pos\_x - self.pos\_x
239 dy = pc.pos\_y - self.pos\_y
240 dist = sqrt(dx * dx + dy * dy)
241 if dist <= self.ATTACK\_RANGE:
242     pc.is\_attack = True
243     pc.attack(self)
244     time.sleep(0.125)
245     if self.hp <=0:
246         pc.is\_attack = False
247
248 def attack(self, actor):
249     """
250     совершает атаку по actor
251
252     :param actor: персонаж, которого атакуют
253     """
254     actor.hp -= self.damage
255     def update(self):
256         """
257         обновляет состояние персонажа
258
259         """
260     super().update()
261     if self.hp <= 0:
262         self.stop\_move()
263         self.set\_state('death')
264
265
266 \paragraph{Создание зон, заполнение их персонажами/объектами}
267 Мир также состоит из зон (Area). Каждая зона включает в себя графику, персонажи и предметы и сценарии взаимодействия. Исключение составляет команда РС, 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.

268
269 import random
270 from math import sqrt
271 import time
272 from rpg.area import *
273 from rpg.sprite import *
274 from rpg.rectangle import *
275 from rpg.game import Game
276 from rpg.portal import Portal
277
278 class Ruins(Area):

```

```

279 def __init__(self):
280     """
281     Класс игровой зоны Ruins
282     """
283
284     super().__init__()
285     self.add_sprite(Sprite('images/fon3.png'), 590, 400, 0)
286     self.add_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.width(),
287                           height=Sprite('images/fon3.png').image.height()))
288     from grunt import Grunt
289     self.grunt = Grunt(0,0,0)
290     from footman import Footman
291     self.footman = Footman(0,0,0)
292     self.add_object(self.footman, 120, 120, 1)
293     self.add_object(self.grunt, 500, 185, 1)
294     p = Portal(400, 400, 200, 200, 'Village', 480, 100)
295     self.add_object(p, p.pos_x, p.pos_y, 100)
296     Game.game.start_script(self.ai, "ai", self.grunt)
297     Game.game.start_script(self.walk_two, "footman", 50, 50)
298
299 def walk(self, step_x, step_y, actor):
300     """
301     Сценарий для движения бугая
302
303     :param step_x: шаг движения x
304     :param step_y: шаг движения y
305     """
306
307     if actor.hp <= 0:
308         Game.game.stop_script("grunt")
309         new_x = 200
310         new_y = 200
311         actor.is_attack = False
312         direction = random.choice(["up", "down", "left", "right"])
313         if direction == "up":
314             new_y -= step_y
315             new_x = step_x
316         elif direction == "down":
317             new_y += step_y
318             new_x = step_x
319         elif direction == "left":
320             new_y = step_y
321             new_x -= step_x
322         elif direction == "right":
323             new_y = step_y
324             new_x += step_x
325
326         actor.search_position(new_x, new_y)
327
328         time.sleep(2)
329
330     модуль bggame:
331     from ruins import *
332     import time

```

```

332 import random
333
334 \paragraph{Пример сценариев: переход между зонами}
335 Глобальные сценарии находятся в классе игры (BGGame), мы подключаем их как :
336 Area.set\_enter\_script(script)
337 Как происходит переход команды между зонами.
338 В зоне определяем объект портал, по клику мыши когда персонаж заходит внутрь портала
    срабатывает self.actor\_in(self, actor). При создании портала, мы указываем
    кудаи в какую зону разместить команду персонажей.

339
340 from rpg.object import Object
341 from rpg.game import Game
342 from rpg.rectangle import Rectangle
343
344 class Portal(Object):
345     def \_\_init\_\_(self, x, y, width, height, area, team\_x, team\_y):
346         """
347     Создает портал в новую зону
348
349     :param x: координата x портала
350     :param y: координата y портала
351     :param width: ширина портала
352     :param height: высота портала
353     :param area: имя зоны куда будет переход
354     :param team\_x: местоположение команды в новой зоне
355     :param team\_y: местоположение команды в новой зоне
356     """
357     self.states = None
358     self.sprite = None
359     self.category = 'portal'
360     super().\_\_init\_\_(x, y, 0)
361     self.rectangle = Rectangle(x, y, width, height)
362     self.area = area
363     self.team\_x = team\_x
364     self.team\_y = team\_y
365     self.visible = False
366
367     def actor\_in(self, actor):
368         """
369     Проверяет находится ли персонаж внутри портала
370
371     :param actor: проверяемый персонаж
372     """
373     if actor.category == "pc":
374         Game.game.set\_area(self.area)
375         Game.game.set\_team(self.team\_x, self.team\_y, 100)
376         actor.stop\_move()
377
378     модуль ruins
379     import random
380     from math import sqrt
381     import time
382     from rpg.area import *
383     from rpg.sprite import *

```

```

384 from rpg.rectangle import *
385 from rpg.game import Game
386 from rpg.portal import Portal
387
388 class Ruins(Area):
389     def __init__(self):
390         """
391     Класс игровой зоны Ruins
392
393     """
394     super().__init__()
395     self.add_sprite(Sprite('images/fon3.png'), 590, 400, 0)
396     self.add_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.width(),
397                           height=Sprite('images/fon3.png').image.height()))
398     from grunt import Grunt
399     self.grunt = Grunt(0,0,0)
400     from footman import Footman
401     self.footman = Footman(0,0,0)
402     self.add_object(self.footman, 120, 120, 1)
403     self.add_object(self.grunt, 500, 185, 1)
404     p = Portal(400, 400, 200, 200, 'Village', 480, 100)
405
406 \paragraph{Как будет идти бой}
407 Бой будет совершаться с помощью сценариев. У класса Adnd\_actor есть метод attack(  

    self, actor), который уменьшает текущее количество здоровья у actor. В модуле  

    game существуют методы start\_script(script, name), stop\_script(name). С помощь  

    сценариев возможно запускать параллельные потоки. В конкретную зону будет  

    добавляться сценарий 'ai', в который передаётся конкретный персонаж. В этом  

    сценарии указывается поведение противника, Что он должен сближаться с персонажем  

    игрока, и когда расстояние до атаки будет достаточным, чтобы её совершить, будет  

    вызван метод actor.attack. Для того, чтобы пользователь мог атаковать персонажа,  

    у каждого экземпляра класса adnd\_actor есть метод click(self), который вызывает  

    проверку условия, если персонаж близко к персонажу игрока, хранящемуся в rpg.game  

    .Game.team\_of\_pc, то вызвать у pc=rpg.game.Game.team\_of\_pc[0], attack(self)/
408 Пример:
409 модуль adnd\_actor:
410 from math import sqrt
411 from rpg.actor import Actor
412 from rpg.animation import Animation
413 import rpg.game
414 import time
415
416 class Adnd\_actor(Actor):
417     ATTACK\_RANGE = 50
418
419     def __init__(self, x, y, z, **params):
420         """
421     Класс Adnd\_actor содержащий основные механики взаимодействия с другими персонажами
422
423     :param x: координата x
424     :param y: координата y
425     :param z: координата z
426     """

```

```

427 super().__init__(x, y, z, **params)
428 self.on_click = self.click
429
430 def click(self):
431     """
432     вызывается при клике на персонажа
433
434     """
435 pc = rpg.game.Game.game.team_of_pc[0]
436 if pc == self:
437     return
438 dx = pc.pos_x - self.pos_x
439 dy = pc.pos_y - self.pos_y
440 dist = sqrt(dx * dx + dy * dy)
441 if dist <= self.ATTACK_RANGE:
442     pc.is_attack = True
443     pc.attack(self)
444     time.sleep(0.125)
445     if self.hp <=0:
446         pc.is_attack = False
447
448 def attack(self, actor):
449     """
450     совершает атаку по actor
451
452     :param actor: персонаж, которого атакуют
453     """
454     actor.hp -= self.damage
455 def update(self):
456     """
457     обновляет состояние персонажа
458
459     """
460     super().update()
461     if self.hp <= 0:
462         self.stop_move()
463         self.set_state('death')
464
465 Модуль ruins
466 import random
467 from math import sqrt
468 import time
469 from rpg.area import *
470 from rpg.sprite import *
471 from rpg.rectangle import *
472 from rpg.game import Game
473 from rpg.portal import Portal
474
475 class Ruins(Area):
476     def __init__(self):
477         """
478     Класс игровой зоны Ruins
479
480     """

```

```

481 super().__init__()
482 self.add_sprite(Sprite('images/fon3.png'), 590, 400, 0)
483 self.add_rect(Rectangle(x=0, y=0, width=Sprite('images/fon3.png').image.width(),
484 height=Sprite('images/fon3.png').image.height()))
485 from grunt import Grunt
486 self.grunt = Grunt(0,0,0)
487 from footman import Footman
488 self.footman = Footman(0,0,0)
489 self.add_object(self.footman, 120, 120, 1)
490 self.add_object(self.grunt, 500, 185, 1)
491 p = Portal(400, 400, 200, 200, 'Village', 480, 100)
492 self.add_object(p, p.pos_x, p.pos_y, 100)
493 Game.game.start_script(self.ai, "ai", self.grunt)
494 Game.game.start_script(self.walk_two, "footman", 50, 50)
495
496 def walk(self, step_x, step_y, actor):
497     """
498     Сценарий для движения бугая
499
500     :param step_x: шаг движения x
501     :param step_y: шаг движения y
502     """
503     if actor.hp <= 0:
504         Game.game.stop_script("grunt")
505     new_x = 200
506     new_y = 200
507     actor.is_attack = False
508     direction = random.choice(["up", "down", "left", "right"])
509     if direction == "up":
510         new_y -= step_y
511         new_x = step_x
512     elif direction == "down":
513         new_y += step_y
514         new_x = step_x
515     elif direction == "left":
516         new_y = step_y
517         new_x -= step_x
518     elif direction == "right":
519         new_y = step_y
520         new_x += step_x
521
522     actor.search_position(new_x, new_y)
523
524 def ai(self, actor):
525     """
526     скрипт противников
527
528     :param step_x: размер шага x до персонажа игрока
529     :param step_y: размер шага x до персонажа игрока
530     :param actor: персонаж противник
531     """
532     if actor.hp <= 0:
533         Game.game.stop_script("ai")

```

```

534 import rpg.game
535 pc = rpg.game.Game.game.team\_of\_pc[0]
536 new\_x = pc.pos\_x
537 new\_y = pc.pos\_y
538
539 actor.search\_position(new\_x, new\_y)
540 dx = pc.pos\_x - actor.pos\_x
541 dy = pc.pos\_y - actor.pos\_y
542 dist = sqrt(dx * dx + dy * dy)
543 if dist <= actor.ATTACK\_RANGE:
544 actor.is\_attack = True
545 actor.attack(pc)
546 time.sleep(1)
547 if pc.hp <=0:
548 actor.update()
549 Game.game.stop\_script("ai")
550 Game.game.start\_script(self.walk, "grunt", 50, 50, actor)
551
552 else:
553 actor.is\_attack = False
554 time.sleep(2)
555
556 \paragraph{Соединение движка и окон tkinter}
557 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)

558
559 Модуль sprite
560
561 import tkinter as tk
562 class Sprite:
563
564     def \_\_init\_\_(self, image):
565         """
566         Класс спрайта для работы с изображениями на Canvas
567
568         :param image: адресс изображения который
569         """
570         self.image = tk.PhotoImage(file=image)
571         self.tag = None
572         self.x = 0
573         self.y = 0
574         self.z = 0
575
576     def set\_tag(self, tag):
577         """
578         Устанавливает тег спрайта
579
580         :param tag: тег спрайта
581         """
582         self.tag = tag
583
584     def set\_z(self, z):

```

```

585 """
586 Устанавливает z-координату спрайта
587
588 :param z: координата z
589 """
590 self.z = z
591
592 def get_tag(self):
593 """
594 Возвращает тег спрайта
595
596 """
597 return self.tag
598
599 def set_coords(self, new_x, new_y):
600 """
601 Обновляет координаты спрайта
602
603 :param new_x: координата x
604 :param new_y: координата y
605 """
606 if self.tag:
607     self.x = new_x
608     self.y = new_y
609 def update(self):
610 """
611 Обновляет анимацию спрайта
612
613 """
614 pass
615
616 модуль graphics
617
618 import tkinter as tk
619
620 class Graphics(tk.Canvas):
621     canvas = None
622     def __init__(self, master, **kwargs):
623 """
624 Класс с методами для работы со спрайтами
625
626 """
627     super().__init__(master, **kwargs)
628     self.sprites = []
629     Graphics.canvas = self
630
631     def add_sprite(self, sprite, x, y, z, **kwargs):
632 """
633 Добавляет спрайт на Canvas
634
635 :param sprite: спрайт
636 :param x: координата x
637 :param y: координата y
638 :param z: координата z

```

```

639 :param kwargs: параметры относящиеся к конкретному изображению в tkinter
640 """
641 tag = self.create_image(x, y, image=sprite.image, anchor='center', **kwargs)
642 sprite.set_tag(tag)
643 sprite.set_z(z)
644 sprite.x = x
645 sprite.y = y
646 self.sprites.append(sprite)
647 self.sprites.sort(key=lambda sprite: sprite.z)

648
649 def update(self):
650 """
651 Перерисовывает все спрайты
652 """
653 for sprite in self.sprites:
654     sprite.update()
655     self.tag_raise(sprite.get_tag())
656     self.coords(sprite.get_tag(), sprite.x, sprite.y)
657     self.itemconfig(sprite.get_tag(), image=sprite.image)

658
659
660
661 def change_sprite(self, sprite, new_sprite):
662 """
663 Меняет спрайт на новый.
664
665 :param sprite: экземпляр спрайта
666 :param new_sprite: новый спрайт
667 """
668 old_sprite_pos = None
669 for i, s in enumerate(self.sprites):
670     if s.get_tag() == sprite.get_tag():
671         old_sprite_pos = i
672         break
673
674 if old_sprite_pos is not None:
675     old_tag = sprite.get_tag()
676
677     self.sprites[old_sprite_pos] = new_sprite
678     new_sprite.set_tag(old_tag)

679     new_sprite.set_tag(old_tag)
680     new_sprite.set_z(sprite.z)

681
682     self.tag_raise(old_tag)
683     self.coords(old_tag, sprite.x, sprite.y)
684     self.itemconfig(old_tag, image=new_sprite.image)

685
686
687 def delete_sprite(self, sprite):
688 """
689 Удаляет спрайт с Canvas.
690
691 :param sprite: экземпляр спрайта
692 :return:

```

```

693 """
694 self.delete(sprite.get\_tag())
695 self.sprites.remove(sprite)
696
697 def clear\_all(self):
698 """
699 Удаляет все спрайты с Canvas
700 """
701 for sprite in self.sprites:
702 self.delete(sprite.get\_tag())
703 self.sprites.clear()
704
705 Модуль baldursgame '''пользовательский модуль'''
706
707 from ruins import *
708 from village import *
709 import time
710 import random
711
712 class BaldursGame(Game):
713     def \_\_init\_\_(self, canvas, window, **params):
714         """
715 Класс конкретной игры для демонстрации
716
717 :param canvas: класс графической системы
718 :param window: окно на которое будет выводится игра
719 """
720 super().\_\_init\_\_(canvas, window, **params)
721 from mage import Mage
722 self.add\_pc\_to\_team(Mage(0, 0, 0))
723 self.new\_area('Ruins', Ruins())
724 self.new\_area('Village', Village())
725 self.set\_area('Ruins')
726 self.set\_team(500, 300, 100)
727 self.timer()
728
729 Модуль main
730 from bggame import *
731
732 root = tk.Tk()
733 root.geometry('1500x1500')
734
735 exit\_button = tk.Button(root, text="Exit", fg="red", command=root.destroy)
736 canvas = Graphics(root, width=1500, height=1500)
737 Graphics.canvas = canvas
738
739 BaldursGame(canvas, root)
740
741 canvas.place(height = 1500, width =1500)
742 BaldursGame.timer()
743
744 root.mainloop()
745
746

```

```

747 \subsection{Модули и классы}
748 \subsection{Game}
749 \subsubsection{Описание модуля}
750 \paragraph{Конструктор и поля модуля}
751 def __init__(self, canvas, window, **params)
752 \begin{itemize}
753     \item self.rpg_dict_of_area = {}
754     \item словарь, хранящий в себе множество экземпляров класса Area, {number - ключ :
755         name Area - значение}
756     \item self.team_of_pc = []
757     \item список, хранящий в себе имена экземпляров класса Actor с параметром category
758         = "pc"
759     \item self.canvas = canvas
760     \item графика
761     \item self.root = window
762     \item окно для графики
763     \item self.current_area = None
764     \item параметр хранящий, текущую зону
765     \item self.scripts = {}
766     \item Словарь для хранения запущенных сценариев
767     \item self.events = {}
768     \item Словарь для хранения запущенных event`ов сценариев
769     \item self.canvas.bind("<Button-1>", self.mouse_left_click)
770     \item обработчик клика
771     \item Game.game = self
772     \item экземпляр игры, для обращения к нему напрямую
773 \end{itemize}
774 \subsubsection{Методы}
775 \paragraph{New_area}
776 def new_area(self, name, area)
777 Описание метода: Добавляет новую зону в список.
778 \paragraph{Set_area}
779 def set_area(self, name)
780 Описание метода: Устанавливает текущую зону, загружает графику зоны.
781 \paragraph{New_actor}
782 def new_actor(self, name, **params)
783 Описание метода: метод отвечающий за создание класса, потомка от Actor и создание
784     поля из параметров, и установление их в начальные значения.
785 \paragraph{Start_script}
786 def start_script(self, script_function, script_name, *args)
787 Описание метода: Запускает сценарий в отдельном потоке с возможностью остановки и
788     передачи аргументов.
789 \paragraph{Stop_script}
790 def stop_script(self, script_name)
791 Описание метода: Останавливает сценарий по имени.
792 \paragraph{Add_pc_to_team}
793 def add_pc_to_team(self, pc)
794 Описание метода: метод отвечающий за добавление имени экземпляра класса Actor с
795     параметром category = "pc" в список team_of_pc, хранящий имена всех игровых
796     персонажей.
797 \paragraph{Remove_pc_from_team}
798 def remove_pc_from_team(self, pc)

```

794 Описание метода: метод отвечающий за удаление имени экземпляра класса Actor с параметром category = "pc" в список team\\_of\\_pc, хранящий имена всех игровых персонажей.

795 \paragraph{Set\\_team}

796 def set\\_team(self, x, y, z)

797 Описание метода: Устанавливает координаты персонажей команды.

798 \paragraph{Update}

799 def update(self)

800 Описание метода: Вызывается в таймере для обновления всех переменных в текущей зоне.

801 \paragraph{Mouse\\_left\\_click}

802 def mouse\\_left\\_click(self, event)

803 Описание метода: обрабатывает клик мыши.

804 \paragraph{Timer}

805 def timer(self)

806 Описание метода: Таймер должен вызывать метод update постоянно.

807

808

809 \subsection{Area}

810 \subsubsection{Описание модуля}

811 \paragraph{Конструктор и поля модуля}

812 def \\_\\_init\\_\\_(self, \*\*params)

813 \begin{itemize}

814 \item self.area\\_zone = params

815 \item параметр определяющий особенности конкретной зоны

816 \item self.objects = []

817 \item список, хранящий в себе множество экземпляров классов Item

818 \item self.sprites = []

819 \item список фоновых спрайтов

820 \item self.rectangles = None

821 \item список, хранящий в себе множество прямоугольников

822 \end{itemize}

823 \subsubsection{Методы}

824 \paragraph{Add\\_sprite}

825 def add\\_sprite(self, sprite, x, y, z)

826 Описание метода: Добавляет спрайт в зону.

827 \paragraph{Add\\_object}

828 def add\\_object(self, obj, x, y, z)

829 Описание метода: Добавляет объект в зону.

830 \paragraph{Remove\\_object}

831 def remove\\_object(self, obj)

832 Описание метода: Удаляет объект из зоны.

833 \paragraph{Load\\_sprites}

834 def load\\_sprites(self)

835 Описание метода: Загружает все спрайты зоны.

836 \paragraph{Add\\_rect}

837 def add\\_rect(self, rec)

838 Описание метода: Добавляет прямоугольник в зону.

839 \paragraph{Entry\\_script}

840 def entry\\_script(self)

841 Описание метода: Запускается, когда команда входит в зону.

842 \paragraph{Exit\\_script}

843 def exit\\_script(self)

844 Описание метода: Запускается, когда команда выходит из зоны.

845 \paragraph{Update}

```

846 def update(self)
847 Описание метода: Изменяет и проверяет изменение всех персонажей в зоне.
848
849 \subsection{Sprite}
850 \subsubsection{Описание модуля}
851 \paragraph{Конструктор и поля модуля}
852 def __init__(self, image)
853 \begin{itemize}
854     \item self.spr\image = image
855     \item Описание параметра: параметр хранит изображение конкретного экземпляра
856         класса Sprite.
857     \item self.tag = None
858     \item Описание параметра: параметр хранит числовое значение обозначающее
859         расположение конкретного экземпляра класса Sprite.
860     \item self.spr\x = x
861     \item Описание параметра: параметр хранит числовое значение обозначающее
862         расположение конкретного экземпляра класса Sprite.
863     \item self.spr\y = y
864     \item Описание параметра: параметр хранит числовое значение обозначающее
865         расположение конкретного экземпляра класса Sprite.
866     \item self.spr\z = z
867     \item Описание параметра: параметр хранит числовое значение обозначающее
868         расположение конкретного экземпляра класса Sprite.
869 \end{itemize}
870 \subsubsection{Методы}
871 \paragraph{Set\_tag}
872 def set_tag(self, tag)
873 \paragraph{Set\_z}
874 def set_z(self, z)
875 \paragraph{Get\_tag}
876 def get_tag(self)
877 \paragraph{Set\_coords}
878 def set_coords(self, new\x, new\y)
879 \paragraph{Update}
880 def update(self)
881 \subsection{Graphics}
882 \subsubsection{Описание модуля}
883 \paragraph{Конструктор и поля модуля}
884 def __init__(self, master, **kwargs)
885 \begin{itemize}
886     \item super().__init__(master, **kwargs)
887     \item self.sprites = []
888     \item список спрайтов
889     \item Graphics.canvas = self
890     \item параметр для работы с графикой, где к ней нужно обращаться напрямую
891 \end{itemize}
892 \subsubsection{Методы}
893 \paragraph{Add\_sprite}
894 def add_sprite(self, sprite, x, y, z, **kwargs)
895 Описание метода: Добавляет спрайт на Canvas
896 \paragraph{Update}
897 def update(self)
898 Описание метода: Перерисовывает все спрайты
899 \paragraph{Change\_sprite}
900 def change_sprite(self, sprite, new_sprite)
901 Описание метода: Меняет спрайт на новый

```

```

896 \paragraph{Delete\_sprite}
897 def delete_sprite(self, sprite)
898 Описание метода: Удаляет спрайт с Canvas
899 \paragraph{Clear\_all}
900 def clear_all(self)
901 Описание метода: Удаляет все спрайты с Canvas
902
903 \subsection{Animation(Sprite)}
904 \subsubsection{Описание модуля}
905 \paragraph{Конструктор и поля модуля}
906 def __init__(self, frames, cycle=True)
907 \begin{itemize}
908     \item super().__init__(frames[0])
909     \item self.images = frames
910     \item Описание параметра: """список кадров"""
911     \item self.current_frame = 0
912     \item self.images = [tk.PhotoImage(file=frame) for frame in frames]
913     \item Загрузка всех кадров анимации
914     \item self.image = self.images[0]
915     \item Установка начального изображения
916     \item self.speed = 3
917     \item Описание параметра: """скорость анимации"""
918     \item self.counter = self.speed
919     \item self.cycle = cycle
920     \item self.running = True
921 \end{itemize}
922 \subsubsection{Методы}
923 \paragraph{Update:}
924 def update(self):
925     Меняет текущее изображение в списке изображений.
926
927 \subsection{Object}
928 \subsubsection{Описание модуля}
929 \paragraph{Конструктор и поля модуля}
930 def __init__(self, x, y, z, **params)
931 \begin{itemize}
932     \item self.pos_x = x
933     \item координата x
934     \item self.pos_y = y
935     \item координата y
936     \item self.pos_z = z
937     \item координата z
938     \item self.current_state = None
939     \item текущее состояние
940     \item self.visible = True
941     \item видим ли объект
942     \item self.on_click = lambda x : x
943     \item возможно ли кликнуть по объекту
944     \item if self.states is not None:
945         self.set_state(next(iter(self.states)))
946     \item установка первого состояния если потребуется
947     \item self.rectangle = Rectangle(x, y, 10, 10)
948     \item прямоугольник объекта
949 \end{itemize}

```

```

950 \subsubsection{Методы}
951 \paragraph{Set\_state}
952 def set_state(self, state_name):
953 Описание метода: меняет текущее состояние объекта
954 \paragraph{Actor\_in}
955 def actor_in(self, actor):
956 Описание метода: Вызывается когда actor входит внутрь объекта
957 \paragraph{Update}
958 def update(self)
959 Описание метода: ничего не делает
960
961 \subsection{Actor(Object)}
962 \subsubsection{Описание модуля}
963 \paragraph{Конструктор и поля модуля}
964 def __init__(self, x, y, z, **params)
965 \begin{itemize}
966     \item self.sprite = self.states[next(iter(self.states))]
967     \item параметр хранящий спрайт
968     \item super().__init__(x, y, z, **params)
969     \item self.speed_x = 0
970     \item значение скорости x
971     \item self.speed_y = 0
972     \item значение скорости y
973     \item self.target_x = 0
974     \item координата x в которую будет двигаться персонаж
975     \item self.target_y = 0
976     \item координата y в которую будет двигаться персонаж
977     \item self.rectangle = Rectangle(self.pos_x, self.pos_y, self.sprite.image.width
978                                     (), self.sprite.image.height())
979     \item прямоугольник персонажа
980     \item self.is_attack = False
981     \item состояние - атакует ли персонаж сейчас
982 \end{itemize}
983 \subsubsection{Методы}
984 \paragraph{update}
985 def update(self)
986 Описание метода: изменяет координаты и состояние персонажа.
987 \paragraph{Search\_position}
988 def search_position(self, new_x, new_y)
989 Описание метода: Изменяет направление движения у персонажа.
990 \paragraph{Stop\_move}
991 def stop_move(self):
992 Описание метода: Останавливает движение персонажа.
993
994 \subsection{Adnd\_actor(Actor)}
995 \subsubsection{Описание модуля}
996 \paragraph{Конструктор и поля модуля}
997 def __init__(self, x, y, z, **params)
998 \begin{itemize}
999     \item super().__init__(x, y, z, **params)
1000     \item self.on_click = self.click
1001     \item событие по клику на персонажа
1002 \end{itemize}
1003 \subsubsection{Методы}

```

```

1003 \paragraph{Click}
1004 def click(self)
1005 Описание метода: вызывается при клике на персонажа.
1006 \paragraph{Attack}
1007 def attack(self, actor)
1008 Описание метода: совершает атаку по actor.
1009 \paragraph{Update}
1010 def update(self)
1011 Описание метода: обновляет состояние персонажа.

1012
1013 \subsection{Rectangle}
1014 \subsubsection{Описание модуля}
1015 \paragraph{Конструктор и поля модуля}
1016 def __init__(self, x, y, width, height)
1017 \begin{itemize}
1018     \item self.x = x
1019     \item координата x прямоугольника
1020     \item self.y = y
1021     \item координата y прямоугольника
1022     \item self.width = width
1023     \item ширина прямоугольника
1024     \item self.height = height
1025     \item высота прямоугольника
1026 \end{itemize}
1027 \subsubsection{Методы}
1028 \paragraph{Is\_in}
1029 def is_in(self, rect)
1030 Описание метода: Проверяет, входит ли прямоугольник self в прямоугольник rect
1031 \paragraph{Is\_point\_inside}
1032 def is_point_inside(self, target_x, target_y)
1033 Описание метода: Проверяет, входит ли точка (x, y) в данный прямоугольник

1034
1035 \subsection{Portal(Object)}
1036 \subsubsection{Описание модуля}
1037 \paragraph{Конструктор и поля модуля}
1038 def __init__(self, x, y, width, height, area, team_x, team_y)
1039 \begin{itemize}
1040     \item self.states = None
1041     \item состояние портала
1042     \item self.sprite = None
1043     \item спрайт портала
1044     \item self.category = 'portal'
1045     \item категория портала
1046     \item super().__init__(x, y, 0)
1047     \item self.rectangle = Rectangle(x, y, width, height)
1048     \item прямоугольник портала
1049     \item self.area = area
1050     \item текущая зона
1051     \item self.team_x = team_x
1052     \item координата x в новой зоне в которую установят команду
1053     \item self.team_y = team_y
1054     \item координата y в новой зоне в которую установят команду
1055     \item self.visible = False
1056     \item видимость портала

```

```
1057 \end{itemize}
1058 \subsubsection{Методы}
1059 \paragraph{Actor\_in}
1060 def actor_in(self, actor)
1061 Описание метода: Проверяет находится ли персонаж внутри портала
```

**Место для диска**