Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ОТЧЕТ

о преддипломной (производственной) практике

наименование вида и типа практики

на (в) Юго-Западном государственном университете

наименование предприятия, организации, учреждения

Студента 4курса, группы ПО-02б

курса, группы

Шевченко Клима Николаевича

фамилия, имя, отчество

Руководитель практики от предприятия, организации, учреждения

директор

должность, звание, степень

Оценка _____

_____ _____

фамилия и. о. подпись, дата

Руководитель практики от университета

к.т.н. доцент

должность, звание, степень

Оценка _____

Чаплыгин А. А. _____

фамилия и. о. подпись, дата

Члены комиссии

_____ _____

подпись, дата фамилия и. о.

_____ _____

подпись, дата фамилия и. о.

_____ _____

подпись, дата фамилия и. о.

Курск 2024 г.

СОДЕРЖАНИЕ

1 Анализ предметной области 7
1.1 История первых игр что стали прародителями RPG-жанра 7
1.2 Первые популярные RPG-игры 8
1.3 Япония и её JRPG 9
1.4 Популярные RPG студии SSI 10
2 Техническое задание 12
2.1 Основание для разработки 12
2.2 Цель и назначение разработки 12
2.3 Требования пользователя к движку 12
2.4 Правила игры 13
2.5 Особенности Dungeons and Dragons 14
2.6 Игровой мир Фаэрун 16
2.7 монстры мира Фаэрун 17
2.8 Интерфейс пользователя 19
2.9 Пример игры 20
2.10 Моделирование вариантов использования 21
2.11 Требования к оформлению документации 21
3 Технический проект 22
3.1 Общая характеристика организации решения задачи 22
3.2 Обоснование выбора технологии проектирования 22
3.2.1 Описание используемых технологий и языков программирования 22
3.2.2 Язык программирования Python 22
3.2.3 Язык программирования Python 23
3.2.3.1 Достоинства языка Python 23
3.2.3.2 Недостатки языка Python 23
3.2.4 Использование библиотеки Tkinter и реализация таймеров на Python 24
3.2.4.1 Введение 24
3.2.4.2 Возможности Tkinter 24
3.2.4.3 Реализация таймеров на Python 24

3.2.4.4 Заключение 25
3.3 Описание платформы для создания RPG игр 25
3.3.1 Пример клиентского кода игры 28
3.3.1.1 Создание классов персонажей/предметов 28
3.3.1.2 Задание правил атаки 28
3.3.1.3 Создание заклинаний с их действием 29
3.3.1.4 Создание зон, заполнение их персонажами/объектами 29
3.3.1.5 Пример сценариев: переход между зонами 30
3.3.1.6 Как будет идти бой 30
3.3.1.7 Сценарии с диалогами 32
3.3.1.8 Соединение движка и окон tkinter 33
3.4 Модули и классы 34
3.5 Game 34
3.5.1 Описание модуля 34
3.5.1.1 Конструктор и поля модуля 34
3.5.2 Методы 34
3.5.2.1 New_item 34
3.5.2.2 New_spell 34
3.5.2.3 New_actor 35
3.5.2.4 New_area 35
3.5.2.5 Set_area 35
3.5.2.6 Add_pc_to_team 35
3.5.2.7 Remove_pc_from_team 35
3.5.2.8 Start_script 35
3.5.2.9 Stop_thread 36
3.6 Area 36
3.6.1 Описание модуля 36
3.6.1.1 Конструктор и поля модуля 36
3.6.2 Методы 36
3.6.2.1 Add_sprite 36
3.6.2.2 Add_obj 36

3.6.2.3 Check_obj_sprite 37
3.6.2.4 Set_team 37
3.7 Sprite 37
3.7.1 Описание модуля 37
3.7.1.1 Конструктор и поля модуля 37
3.8 Graphics 38
3.8.1 Описание модуля 38
3.8.1.1 Конструктор и поля модуля 38
3.8.2 Методы 38
3.8.2.1 Change_frame 38
3.8.2.2 Add_anim 38
3.8.2.3 OnTimer 38
3.8.2.4 Draw_all 38
3.8.2.5 Add_sprite 38
3.8.2.6 Update 38
3.9 Animation 39
3.9.1 Описание модуля 39
3.9.1.1 Конструктор и поля модуля 39
3.9.2 Методы 39
3.9.2.1 onTimerAnimation 39
3.9.2.2 Crop 39
3.9.2.3 Get_frame 39
3.9.2.4 Draw_all 39
3.10 Object 40
3.10.1 Описание модуля 40
3.10.1.1 Конструктор и поля модуля 40
3.10.2 Методы 41
3.10.2.1 Move 41
3.10.2.2 Attack 41
3.10.2.3 Die 42
3.10.2.4 Set_click_script 42

3.11 Actor(Object) 42
3.11.1 Описание модуля 42
3.11.1.1 Конструктор и поля модуля 42
3.11.2 Методы 44
3.11.2.1 Readtext 44
3.11.2.2 Open_inventory 44
3.11.2.3 Open_list_spells 44
3.11.2.4 Target_use_spell 44
3.11.2.5 Target_use_item 44
3.12 Spell 45
3.12.1 Описание модуля 45
3.12.1.1 Конструктор и поля модуля 45
3.12.2 Методы 45
3.12.2.1 Cast_spell 45
3.12.2.2 Learn_spell 45
3.12.2.3 Forget_spell 46
3.13 Item 46
3.13.1 Описание модуля 46
3.13.1.1 Конструктор и поля модуля 46
3.13.2 Методы 47
3.13.2.1 Use_item 47
3.13.2.2 Equip_item 47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 47

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИС – информационная система.

ИТ – информационные технологии.

КТС – комплекс технических средств.

ОМТС – отдел материально-технического снабжения.

ПО – программное обеспечение.

РП – рабочий проект.

ТЗ – техническое задание.

ТП – технический проект.

UML (Unified Modelling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

1 Анализ предметной области

1.1 История первых игр что стали прародителями RPG-жанра

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

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

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

было обладать какой-никакой системой прокачки, да давать возможность отыгрывать в бою воина или мага.

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

1.2 Первые популярные RPG-игры

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

Благодаря развитию технологий, большему бюджету и поддержке издателя Гэрриот сумел в кратчайшие сроки значительно улучшить техническую составляющую игры — вышедшая спустя год Ultima обзавелась тайловой графикой, а для управления персонажем больше не нужно было вводить текстовые команды — 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 сдобрили обширной системой магии, среди которой было место как прямо атакующим заклинаниям, так и различным дебаффам. А еще разработка Sir-Tech была беспощадно сложной: подобно Rogue в случае смерти партии игроку ничего не оставалось, кроме как начать с нуля

1.3 Япония и её JRPG

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

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

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

1.4 Популярные RPG студии SSI

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

и функцию быстрого перемещения с помощью раскинувшейся по игровому миру сети телепортов. Лицензия DnD распространялась и на использование различных сеттингов настолки, поэтому местом действия игр могли стать как «Забытые Королевства», так и вселенная «Драконьего Копья». Первоисточник даровал разработчикам не только готовую механику, но и проработанную мифологию. Такой мощный фундамент позволял стабильно выпускать новинки раз в несколько месяцев. Наладив потоковое производство, SSI превратила создание ролевых игр в индустрию. Вскоре каталог компании пополнили и игры сторонних студий, разработанные по драгоценной лицензии, в числе которых была, например, популярная трилогия Eye of the Beholder от Westwood Studios.

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

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 генерируемых локаций;
- реализация основных механик Dungeons and Dragons;
- 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;
- реализацию пошаговой системы боя;
- иммитацию основных механик ролевой игры Dungeons and Dragons.

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 двухмерном виде сверху с использованием экранной графики персонажей и управляется с помощью команд с клавиатуры. Подземелье имеет фиксированную планировку, но встречи с монстрами и сокровища генерируются случайным образом. Они генерируются всякий раз, когда создается новый персонаж, и сохраняются вместе с ним.

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

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

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

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

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

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

2.5 Особенности Dungeons and Dragons

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

Основные элементы ролевой системы DnD включают:

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

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

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

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

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

∗ Особенности человека: человек на старте получает +1 ко всем характеристикам, его базовая скорость передвижения равна 30 футам, его размер средний.

∗ Особенности эльфа: эльф получает +2 к ловкости и +1 к мудрости, его базовая скорость передвижения равна 35 футов, его размер средний, у эльфа есть тёмное зрение в радиусе 30 футов.

∗ Особенности дварфа: дварф получает +2 к силе и +2 к телосложению, его базовая скорость передвижения равна 25 футов, его размер маленький, у дварфа есть тёмное зрение в радиусе 30 футов.

– 2. Характеристики:

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

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

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

∗ Телосложение - характеристика влияющая на колличество здоровья персонажа.

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

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

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

– 3. Навыки:

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

– 4. Броски костей:

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

– 5. Приключения и задания:

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

– 6: Прогрессия и опыт:

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

– 7. Магия:

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

2.6 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 Сердцеземье и Дикую Границу. Западное Сердцеземье 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 Дикой Границе относится весь остальной Север, состоящий из совсем незаселенных либо скудно 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).

- Людоящер
- 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 появятся лейбл с текстом ”Выбирете 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 находится красная полоска отображающая здоровье персонажа. Справа от неё будет находится 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 находится 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 находится кнопка ”Особенности и черты”. Кликнув на неё откроется 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 находится 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 совершит 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.

- 4. Клавишами WASD пользователь будет передвигать игрового персонажа: вверх, влево, вниз, вправо.

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, дварф, 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 выведется сообщение со следующим текстом: ”Исследователь подземелий, мы ждали тебя, это одно из многих подземелий Ацецерака. Этот могучий лич хранит в этом месте множество могучих артефактов и горы несметных сокровищ. По приказу короля Персиваля 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 выведется 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. Когла 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!')

Установка времени для таймера (в секундах) t = 10 countdown(t)

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

3.2.4.4 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 BaldursGateGame. В этом модуле мы создаем мир игры, с помощью new_actor, new_item, new_spell. 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 предмету может соответствовать пользовательский сценарий (он активируется при нажатии мышкой на объект). Сценарий может включать диалог, взятие предмета, добавление персонажа в команду, квест и т.д. Зона тоже может содержать сценарий, который запускается когда команда попадает в зону. Клиентский класс (BGGame) также содержит глобальные переменные, определяющие ситуации в игре (например квесты). Локальные переменные могут быть в зоне.

Как программируются зоны. Если нужны локальные переменные (состояние локальных событий), то тогда нужно создавать класс своей зоны как наследник от Area. Или же просто использовать класс Area. Добавляем зону в игру new_area(name, area). Переключаем зону - set_area(name). Глобальные сценарии находятся в классе игры (BGGame), мы подключаем их как : Area.set_enter_script(script) В зону мы добавляем персонажей и предметы как add_object(x,y, obj) - z не нужно, так как слой можно определить по y координате. На объект мы добавляем сценарий для взаимодействия как: obj.set_click_script(script) Как происходит переход команды между зонами. В зоне определяем объект дверь, по клику мыши она может открываться и закрываться (меняется состояние объекта). Назначаем сценарий set_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_thread). Таким образом, мир будет интерактивным. Как связано окно и графика с игрой. В окне мы делаем таймер, который вызывает метод update нашей игры (BGGame). Этот метод выполняет все действия объектов в игре за 1 кадр времени. Также в таймере вызываем Graphics.update(), который обновляет графику игры. Все объекты (Actor, Item) должны иметь состояния (как минимум одно). Каждое состояние связано с спрайтом (или анимацией). То есть переключение состояния меняет графику объекта.

А вообще сценарии и глобальные переменные могут быть без классов, а просто в модулях, так проще, чтобы к ним был доступ из всех комнат. Тогда и функции движка должны быть доступны везде (то есть во всех сценариях). Например делаем модуль лес (forest): from rpg.game import * from rpg.area import Area forest = Area() forest.add_sprite(0, 0, 0, Sprite(”back.jpg”)) sword = Sword(normal: Sprite(”sword.jpg”)) forest.add_object(10, 10, sword) forest.add_sprite(0, 0, 0, Sprite(”brown.jpg”)) door = Door(normal: Sprite(”door.jpg”)) forest.add_object(50, 50, door) man = ManNPC(left: Animation(”left.jpg”), right: Animation(”right.jpg”)) player = PlayerPC(left:

Animation(”left.jpg”), right: Animation(”right.jpg”)) forest.add_object(100, 100, man) forest.add_object(150, 132, player)

def man_dialog(): ...

man.set_click_script(man_dialog) new_area(”Forest forest)

lair = Area() lair.add_sprite(0, 0, 0, Sprite(”green.jpg”)) knife = Knife(normal: Sprite(”knife.jpg”)) lair.add_object(60, 80, knife) bandit = EnemyNPC(left: Animation(”left.jpg”), right: Animation(”right.jpg”)) player = PlayerPC(left: Animation(”left.jpg”), right: Animation(”right.jpg”)) lair.add_object(100, 100, bandit) lair.add_object(50, 50, player)

new_area(”Bandits lair lair)

модуль bggame: from rpg.game import * teampc =() new_item(”Sword category: ”Weapon attack: 10) new_actor(”ManNPC ADnDActor, category: ”NPC”) new_actor(”Bandit ADnDActor, category: ”enemy”) new_actor(”PlayerPC ADnDActor, category: ”PC”) add_actor_to_team(PlayerPC, teampc)

def walk_script(): if (act.x = door.x && act.y = door.y) set_area(”Bandits lair”) put_team(50, 50)

import forest

set_area(”Forest”) set_team(teampc, forest, 100, 100)

set_walk_script(walk_script)

import lair

Еще надо определиться, как задать правила AD&D. Лучше так: сделать класс ADnDActor (это делает клиент, не движок), где мы перекрываем метод attack, и там программируем формулу расчета повреждения, уменьшаем hp и т.д. То есть Actor.attack(actor), где actor - кого атакуют Тогда надо в методе new_actor указывать еще один параметр - базовый класс, от которого будет наследование. Пример. new_actor(”Knight ADnDActor, category: ”enemy ...) С магией можно сделать так: абстрактный класс Spell, где есть метод cast(actor1, actor2), то есть actor1 делает cast на actor2. он изначально пустой. А в new_spell мы должны указать функцию как именно должен работать Spell (можно прямо как lambda): new_spell(”Fireball level:3, class: [”wizard ”priest”], lambda a1,a2 : a2.hp -= 30) проверку на класс (который требуется) можно изначально сделать в Spell

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, new_item, new_spell.

модуль bggame: from rpg.game import * teampc =() new_item(”Sword category: ”Weapon attack: 10) new_actor(”ManNPC ADnDActor, category: ”NPC”) new_actor(”Bandit ADnDActor, category: ”enemy”) new_actor(”PlayerPC ADnDActor, category: ”PC”) new_spell(”Fireball level:3, class: [”wizard ”priest”], lambda a1,a2 : a2.hp -= 30) pc = PlayerPC() add_actor_to_team(pc)

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(actor), где actor - кого атакуют Тогда надо в методе new_actor указывать еще один параметр - базовый класс, от которого будет наследование. Пример.

модуль bggame: from rpg.game import * from rpg.area import Actor

class ADnDActor(Actor) def __init__(self, **params) ADnD_min_disctans

def attack(actor):

mod_attack = self.str % 10 hit = randrange(1,20) + mod_attack damage = randrange(1, weapon) + mod_attack if (abc(self.x - actor.x) <= weapon_min_distance && abc(self.y - actor.y) <= weapon_min_distance) if (hit >= obj.ac) actor.hp -= damage if actor.hp <= 0) actor.alive = false start_script(”death self.death_script, actor)
stop_script(”death”)

def death_script(actor) actor.status = ”death”actor.add_anim(”death”) actor.status = ”corspe”

new_actor(”Knight ADnDActor, category: ”enemy ...)

3.3.1.3 Создание заклинаний с их действием

Создание заклинаний с их действием. С магией можно сделать так: абстрактный класс Spell, где есть метод cast(actor1, actor2), то есть actor1 делает cast на actor2. он изначально пустой. А в new_spell мы должны указать функцию как именно должен работать Spell (можно прямо как lambda):

модуль bggame: from rpg.game import * from rpg.area import Spell

new_spell(”Fireball level:3, class: [”wizard ”priest”], lambda a1,a2 : a2.hp -= 30)

3.3.1.4 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.

from rpg.game import * from rpg.area import Area forest = Area() forest.add_sprite(0, 0, 0, Sprite(”back.png”)) sword = Sword(normal: Sprite(”sword.jpg”)) forest.add_object(10, 10, sword) forest.add_sprite(0, 0, 0, Sprite(”brown.png”)) door = Door(normal: Sprite(”door.png”)) forest.add_object(50, 50, door) man = ManNPC(left: Animation(”left.png”), right: Animation(”right.png”)) player = PlayerPC(left: Animation(”left.png”), right: Animation(”right.png”)) forest.add_object(100, 100, man) forest.add_object(150, 132, player)

def man_dialog(): ...

man.set_click_script(man_dialog) new_area(”Forest forest)

3.3.1.5 Пример сценариев: переход между зонами

Глобальные сценарии находятся в классе игры (BGGame), мы подключаем их как : Area.set_enter_script(script) На объект мы добавляем сценарий для взаимодействия как: obj.set_click_script(script) Как происходит переход команды между зонами. В зоне определяем объект дверь, по клику мыши она может открываться и закрываться (меняется состояние объекта). Назначаем сценарий set_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. Сценарий перехода - в модуле зоны, и там не нужны проверки координат, потому что сработает движок на пересечение объекта. Нужен какой-то объект (нора, дверь ...) на который назначается walk_script А в самом сценарии просто меняем зону и переставляем команду put_team(50, 50) (команда в движке)

модуль bggame: from rpg.game import *

def walk_script(): if (act.x = door.x && act.y = door.y) set_area(”Bandits lair”) put_team(50, 50)

import forest

set_area(”Forest”) set_team(teampc, forest, 100, 100)

set_walk_script(walk_script)

import lair

3.3.1.6 Как будет идти бой

Боевая система - я предполагаю, что боевая система будет пошаговая. То есть Существует боевая ситуация, которая активируется триггером. В мирном состоянии пользователь управляя персонажем PC может передвигаться свободно, на любые расстояния преодолевая препятствия. Другие NPC тоже будут передвигаться спокойно передвигаться согласно их скриптам. Боевая ситуация активируется только в случае, если PC будет входить в зону видимости NPC category: enemy. Я добавлю NPC ”прямоугольник”размером 60 на 60 пикселей, от точки NPC на карте, который проверяет есть ли в площади этого ”прямоугольника”координаты иг-

рока. В случае когда координаты игрока попадают в этот прямоугольник, то начинается боевая ситуация. Либо сделать по-другому, при попадании в новую зону команды PC, если в этой зоне есть NPC category: enemy, то начинается скрипт set_battle_script(script)”боевая ситуация”начинается сразу же. Боевая ситуация будет представлять из себя следующее: Очередь: все персонажи будут совершать все действия по очереди. Будет список в котором отсортированны по значению параметра agility все персонажи, которые участвуют в боевой ситуации в конкретной зоне. Будет у всех участников флаг, который активен, только когда имя того персонажа, находится вверху списка. Список будет обновляться после совершения любого действия. После совершения действия имя персонажа в списке переносится в конец списка, у персонажа отключается флаг хода, а в начало списка переносится следующий элемент в списке, при смерти персонажа, его имя удаляется из списка. Действия NPC: Я предполагаю, что поведение NPC в боевой ситуации, определяется скриптами написанными пользователем специально для боя. Т.е. пользователь пишет свой battle_script для каждого экземпляра класса ADndActor. Скрипт будет вызывать методы передвижения или атаки в зависимости от того как должен вести себя каждый NPC category: enemy. Т.е. у условного скелета скрип будет вызывать метод skeleton.attack(actor) по возможности, если его нецелесообразно использовать(расстояние не позволяет совершать метод attack), то вызывать skeleton.move(x,y). У условного ящера скрипт будет будет вызывать метод lizardman.move(x,y) чтобы увеличить расстояние, а когда оно будет больше 100 единиц по модулю, то только после этого будет вызываться метод lizardman.attack(actor), потому что ящер стреляет из лука, а это значит расстояние для атаки у него больше, чем у того-же скелета. Таким образом все действия NPC определяются заранее написанными скриптами. Но я пока не представляю, как задать эти скрипты В модуле bggame наверное или в каждом экземпляре класса ADndActor. Я не уверен. Окончание боевой ситуации: бой заканчивается, когда в списке очереди, остаётся одно имя или вообще никаких, скрип перестаёт действовать (stop_thread).

Тут надо делать разные сценарии. Собственно начало боя - это тоже сценарий (зашел в комнату, сработал walk_script на что-то (или enter_script - сценарий,

который запускается при входе команды в зону). Условно этот сценарий называется battle_script. Там мы прописываем правила боя (может достаточно только установить глобальный режим боя и проверять условие окончания боя). Пошаговый, значит в определенный момент мы выключаем управление игрока (обработку мыши, клавиш) - control_off(), а затем опять включаем control_on(). Когда наш ход, мы должны считать очки движения/действий. Это может делать сценарий по событию движения персонажа (move_script - тоже может быть один на всех: и своих и врагов). click_script - сценарий по нажатию на врагов, делает атаку, если позволяет расстояние. Причем, это делается только если режим боя (то есть установлена глобальная переменная). Когда очки закончились (или нажата кнопка конец хода), то мы запускаем enemy_script, который выключает управление игрока и по очереди запускает сценарии ai у врагов. Все сценарии работают параллельно (хотя реально работает только один сценарий в определенный момент времени), и в Питоне они должны передавать управление друг другу, это команда движка yield(есть примеры в Интернете как это сделать).

модуль bggame: from rpg.game import * actors_list =() def battle_script(**params) actors_list = (params) action_points = 2 While (actors_list.len > 1) if (actor_1.agi >= actor_2.agi) yield actor_1.click_script() if (actor_2.agi > actor_1.agi) yield actor_2.enemy_script() stop_script() def click_script(self, actor) self.attack(actor) self.move_script() self.control_off() if (actor.status = ”corspe”) actors_list.remove(actor) def enemy_script(self) if (self.attack(actor) && actor.distance <= self_weapon_min_distance) yield self.attack(actor) if (self.move_script() && actor.distance > self_weapon_min_distance) yield self.move_script() actor.control_on() if (actor.status = ”corspe”) actors_list.remove(actor) def move_script(self) self.move(point)

lair.battle_script(player, bandit)

3.3.1.7 Сценарии с диалогами

NPC category: friend будут при клике на них вызывать сценарий, который будет вызвать окно в котором будет поле, содержащее текст прописанный этим скриптом.

def man_say_text(): ...

man.set_click_script(man_say_text)

3.3.1.8 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: ”Класс спрайта для работы с изображениями на Canvas.”def __init__(self, image): self.image = tk.PhotoImage(file=image) self.tag = None self.x = 0 self.y = 0 self.z = 0

модуль graphics

import tkinter as tk import time class Graphics(tk.Canvas): ”Класс Canvas с дополнительными методами для работы со спрайтами.”def __init__(self, master, **kwargs): super().__init__(master, **kwargs) self.sprites = [] # список спрайтов self.mouse = None

def add_sprite(self, sprite, x, y, z, **kwargs): ”Добавляет спрайт на Canvas. param sprite - экземпляр спрайта, x y z - коодринаты.”tag = self.create_image(x, y, image=sprite.image, anchor=’center’, **kwargs) sprite.set_tag(tag) sprite.set_z(z) # устанавливаем z-координату спрайта sprite.x = x sprite.y = y self.sprites.append(sprite) self.sprites.sort(key=lambda sprite: sprite.z) # сортировка спрайтов по z-координате

модуль baldursgame ”’пользовательский модуль”’

from rpg.sprite import * from rpg.game import * from rpg.area import * from rpg.actor import * import datetime import threading

def timer(): first_game.update() root.after(1000, timer)

root = tk.Tk() root.geometry(’1500x1500’)

exit_button = tk.Button(root, text=”Exit fg=”red command=root.destroy) exit_button.pack()

Создание экземпляра класса graphics, который будет взаимодействовать с окном canvas = Graphics(root, width=1500, height=1500)

Загрузка изображения im1_1 = Sprite(’images/fon1.png’) im1_2 = Sprite(’images/fon2.png’) im2_1 = Sprite(’images/person1.png’) im2_2 = Sprite(’images/person2.png’)

3.4 Модули и классы

3.5 Game

3.5.1 Описание модуля

3.5.1.1 Конструктор и поля модуля

def __init__(self, **params)

- self.rpg_dict_of_area =
- Описание параметра: параметр - словарь, хранящий в себе множество экземпляров класса Area, number - ключ : name Area - значение.
- self.team_of_pc[] = name_pc
- Описание параметра: параметр - список, хранящий в себе имена экземпляров класса Actor с параметром category = ”pc”.

3.5.2 Методы

3.5.2.1 New_item

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

3.5.2.2 New_spell

def new_spell(self, name, **params, self. name, level, type, damage, hp) Описание метода: метод отвечающий за создание класса, потомка от Spell и создание поля из параметров, и установление их в начальные значения.

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 New_area

def new_area(name, area) Описание метода: метод отвечающий за создание класса, потомка от Area и создание поля из параметров, и установление их в начальные значения.

3.5.2.5 Set_area

def set_area(name) Описание метода: метод отвечающий за размещение экземпляра класса Area в поле rpg_dict_of_area класса game.

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 Start_script

def start_script(script) Описание метода: метод отвечающий за активацию скрипта.

3.5.2.9 Stop_thread

def stop_thread(script) Описание метода: метод отвечающий за прекращение действий скрипта.

3.6 Area

3.6.1 Описание модуля

3.6.1.1 Конструктор и поля модуля

def __init__(self, **params)

- self.area_zone = params
- Описание параметра: параметр определяющий особенности конкретной зоны (то есть Лес, скорость всех персонажей, понижена вдвое. Река передвижение по зоне невозможно и т.д.)
- self.area_list_objects = params
- Описание параметра: параметр - список, хранящий в себе множество экземпляров классов Item, Actor.

3.6.2 Методы

3.6.2.1 Add_sprite

def add_sprite (x, y, sprite) Описание метода: метод отвечающий за вызов метода add_sprite у класса Graphics.

3.6.2.2 Add_obj

def add_obj(self, name, obj, **params) Описание метода: метод отвечающий за добавление экземпляров классов Item, Actor, в поле area_list_objects экземпляра класса Area.

3.6.2.3 Check_obj_sprite

def check_obj_sprite(x, y, z, sprite) Описание метода: метод отвечающий за проверку пересекаются ли координаты Sprite‘ов различных объектов в экземпляре класса Area.

3.6.2.4 Set_team

def set_team(x, y) Описание метода: метод отвечающий за установку команды игровых персонажей в конкретной 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.spr_x = x
- Описание параметра: параметр хранит числовое значение обозначающее расположение конкретного экземпляра класса Sprite.
- self.spr_y = y
- Описание параметра: параметр хранит числовое значение обозначающее расположение конкретного экземпляра класса Sprite.
- self.spr_z = z
- Описание параметра: параметр хранит числовое значение обозначающее расположение конкретного экземпляра класса Sprite.

3.8 Graphics

3.8.1 Описание модуля

3.8.1.1 Конструктор и поля модуля

def __init__(self)

- self.anim =
- Описание параметра: список анимаций

3.8.2 Методы

3.8.2.1 Change_frame

def change_frame(self, id, img) Описание метода: Смена кадра анимации.

3.8.2.2 Add_anim

def add_anim(self, a, b) Описание метода: добавить анимацию

3.8.2.3 OnTimer

def onTimer(self) Описание метода: Цикл анимаций

3.8.2.4 Draw_all

def draw_all(self, sprite) Описание метода: отрисовать все спрайты

3.8.2.5 Add_sprite

def add_sprite(self, sprite) Описание метода: метод отвечающий за отображение экземпляра класса Sprite

3.8.2.6 Update

def update(self) Описание метода: метод проверяющий координаты всех объектов, состояния объектов, всех кадров анимаций, а затем вызывает метод draw_all.

3.9 Animation

3.9.1 Описание модуля

3.9.1.1 Конструктор и поля модуля

def __init__(self, name, file_name, n, time)

- self.frame = 0
- Описание параметра: ”индекс кадра”
- self.time = time
- Описание параметра: ”скорость анимации”
- self.name = name
- Описание параметра: ”название анимации”
- self.anim = []
- Описание параметра: ”список кадров”
- self.n = n
- Описание параметра: ”колличество кадров в анимации”
- self.file_name = file_name
- Описание параметра: ”имя файла”

3.9.2 Методы

3.9.2.1 onTimerAnimation

def onTimerAnimation(self) Описание метода: Счетчик кадров анимации.

3.9.2.2 Crop

def crop(self) Описание метода: Разбиение на кадры

3.9.2.3 Get_frame

def get_frame(self) Описание метода: Возвращает текущий кадр

3.9.2.4 Draw_all

def draw_all(self, sprite) Описание метода: отрисовать все спрайты

3.10 Object

3.10.1 Описание модуля

3.10.1.1 Конструктор и поля модуля

def __init__(self, name, hp, ac, speed, bool(alive), bool(angry), damage, category, x, y, z)

- self.obj_name = name

- Описание параметра: параметр хранит название конкретного экземпляра класса Object.

- self.obj_hp = hp

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц жизни у конкретного экземпляра класса Object.

- self.obj_ac = ac

- Описание параметра: параметр хранит числовое значение обозначающее класс доспеха у конкретного экземпляра класса Object.

- self.obj_speed = speed

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц передвижения у конкретного экземпляра класса Object.

- self.obj_alive = alive

- Описание параметра: параметр хранит логическое значение обозначающее статус: жив или мёртв(возможно ли с ним взаимодействие) конкретный экземпляр класса Object.

- self.obj_angry = angry

- Описание параметра: параметр хранит логическое значение обозначающее агрессивно ли настроен конкретный экземпляр класса Object, т.е. может ли он вызывать метод def attack().

- self.obj_damage = damage

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц урона, у конкретного экземпляра класса Object, на которое мо-

жет быть уменьшено колличество единиц жизни(параметр obj_hp) у другого экземпляра класса Object.

- self.obj_category = category

- Описание параметра: параметр хранит текстовое значение обозначающее принадлежность его к определённому наследуемому классу - т.е. у экземпляра класса PlayerCharacter category будет иметь значение ”pc у враждебно настроенного NPC(экземпляр класса Actor) будет иметь значение ”enemy у не враждебно настроенного NPC(экземпляр класса Actor) будет иметь значение ”friend”.

- self.obj_pos_x = x

- Описание параметра: параметр хранит числовое значение обозначающее расположение на экране(или зоны - экземпляра класса Area) по координате x у конкретного экземпляра класса Object.

- self.obj_pos_y = y

- Описание параметра: параметр хранит числовое значение обозначающее расположение на экране(или зоны - экземпляра класса Area) по координате y у конкретного экземпляра класса Object.

- self.obj_pos_z = z

- Описание параметра: параметр хранит числовое значение обозначающее расположение на экране(или зоны - экземпляра класса Area) по координате z у конкретного экземпляра класса Object.

3.10.2 Методы

3.10.2.1 Move

def move(self, x, y, z, speed): Описание метода: метод отвечающий за изменение полей obj_pos_x, obj_pos_y obj_pos_z экземпляра класса Object.

3.10.2.2 Attack

def attack(obj): Описание метода: метод отвечающий за изменение поля obj_hp экземпляра класса Object.

3.10.2.3 Die

def die(self, alive, category) Описание метода: метод отвечающий за изменение полей obj_category, obj_alive экземпляра класса Object.

3.10.2.4 Set_click_script

def set_click_script(script): Описание метода: метод отвечающий за действие скрипта у конкретного объекта.

3.11 Actor(Object)

3.11.1 Описание модуля

3.11.1.1 Конструктор и поля модуля

def __init__(self, name, **params)

- self.act_name = name

- Описание параметра: параметр хранит название конкретного экземпляра класса Actor.

- self.act_loot = loot

- Описание параметра: параметр - список, который хранит в себе некоторое количество экземпляров класса Item.

- self.act_text = text

- Описание параметра: параметр хранит текстовое значение, которое будет выведено на экран методом readtext класса Actor.

- self.act_class = class

- Описание параметра: параметр хранит текстовое значение, обозначающее принадлежность к определённому классу игрового персонажа: ”warriorwizardcleric”.

- self.act_level = level

- Описание параметра: параметр хранит числовое значение обозначающее уровень у конкретного экземпляра класса Actor.

- self.act_race = race

- Описание параметра: параметр хранит текстовое значение, обозначающее принадлежность к определённой расе игрового персонажа: ”humanelfdwarf”.

- self.act_str = str

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц параметра ”сила”игрового персонажа у конкретного экземпляра класса Actor.

- self.act_agi = agi

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц параметра ”ловкость”игрового персонажа у конкретного экземпляра класса Actor.

- self.act_con = con

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц параметра ”телосложение”игрового персонажа у конкретного экземпляра класса Actor.

- self.act_int = int

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц параметра ”интеллект”игрового персонажа у конкретного экземпляра класса Actor.

- self.act_wiz = wiz

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц параметра ”мудрость”игрового персонажа у конкретного экземпляра класса Actor.

- self.act_chr = chr

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц параметра ”харизма”игрового персонажа у конкретного экземпляра класса Actor.

- self.act_inventory = inventory

- Описание параметра: параметр - список, хранящий в себе множество экземпляров классов Item.

- self.act_list_spells = spells

- Описание параметра: параметр - список, хранящий в себе множество экземпляров классов Spell.
- self.act_status_cadr
- Описание параметра: параметр - хранит в себе текущий кадр анимации.

3.11.2 Методы

3.11.2.1 Readtext

def readtext(self, text) Описание метода: метод отвечающий за вывод содержания поля act_text экземпляра класса Actor на экран.

3.11.2.2 Open_inventory

def open_inventory(self, inventory) Описание метода: метод отвечающий за вывод содержимого поля act_inventory на экран.

3.11.2.3 Open_list_spells

def open_list_spells(self, spells): Описание метода: метод отвечающий за вывод содержимого поля act_list_spells на экран.

3.11.2.4 Target_use_spell

def target_use_spell(self, obj =Spell(), tar = Actor()): Описание метода: метод отвечающий за вызов метода cast_spell экземпляра класса Spell.

3.11.2.5 Target_use_item

def target_use_item(self, obj =Item(), tar = Actor()): Описание метода: метод отвечающий за вызов метода use_item экземпляра класса Item.

3.12 Spell

3.12.1 Описание модуля

3.12.1.1 Конструктор и поля модуля

def __init__(self, name, level, type, damage)

- self.spell_name = name

- Описание параметра: параметр хранит название конкретного экземпляра класса Spell.

- self.spell_level = level

- Описание параметра: параметр хранит числовое значение обозначающее уровень у конкретного экземпляра класса Spell.

- self.spell_type = type

- Описание параметра: параметр хранит текстовое значение, обозначающее принадлежность к определённому типу заклинания (урон, лечение, перемещение) у конкретного экземпляра класса Spell.

- self.spell_damage = damage

- Описание параметра: параметр хранит числовое значение обозначающее колличество единиц урона, у конкретного экземпляра класса Spell, на которое может быть уменьшено колличество единиц жизни(параметр obj_hp) у другого экземпляра класса Object.

3.12.2 Методы

3.12.2.1 Cast_spell

def cast_spell(self. actor1, actor2) Описание метода: метод отвечающий за изменение поля obj_hp экземпляра класса Object на число равное значению поля spell_damage экземпляра класса Spell.

3.12.2.2 Learn_spell

def learn_spell(self, name, level, list) Описание метода: метод отвечающий за добавление в поле act_list_spells экземпляра класса Actor нового элемента.

3.12.2.3 Forget_spell

def def forget_spell(self, name, level, list) Описание метода: метод отвечающий за удаление из поля act_list_spells экземпляра класса Actor первого элемента.

3.13 Item

3.13.1 Описание модуля

3.13.1.1 Конструктор и поля модуля

def __init__(self, name, category, type, ability)

- self.it.category = category
- Описание параметра: параметр хранит текстовое значение, обозначающее принадлежность к определённой категории предмета (оружие, доспех, магический предмет, прочее) у конкретного экземпляра класса Item.
- self.it.type = type
- Описание параметра: параметр хранит текстовое значение, обозначающее принадлежность к определённому типу предмета, которое он может занимать в инвентаре (голова, одна рука, две руки, доспех, ноги, обувь, прочее) у конкретного экземпляра класса Item.
- self.it.ability = ability
- Описание параметра: параметр хранит текстовое значение, обозначающее особенность (доспех, изменяет значение параметра obj_ac у конкретного экземпляра класса Object, оружие изменяет значение параметра obj_damage у конкретного экземпляра класса Object, магический предмет добавляет новый экземпляр класса Spell, в список act_list_spells конкретного экземпляра класса Actor, прочее, не изменяет параметров, но содержит текст, который будет выводится на экран, при взаимодействии пользователя с предметом) у конкретного экземпляра класса Item.
- self.it_status_cadr
- Описание параметра: параметр - хранит в себе текущий кадр анимации.

3.13.2 Методы

3.13.2.1 Use_item

def use_item(self, ability, obj) Описание метода: метод отвечающий за изменение полей obj_hp, obj_ac, obj_speed экземпляра класса Object или полей act_str, act_agi экземпляра класса Actor на число равное значению поля it.ability экземпляра класса Item.

3.13.2.2 Equip_item

def equip_item(actor, name, inventory) Описание метода: метод отвечающий за добавление в список act_inventory экземпляра класса Actor элемента с названием экземпляра класса Item.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

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

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

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

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

7. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – Санкт-Петербург : Питер, 2001. – 368 с. – ISBN 5-272-00355-1. – Текст : непосредственный.

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

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

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

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

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