

Geekbrains

Дипломный проект

"Создания игры для Windows на языке Python с использованием библиотеки Pygame и применением возможностей нейросетей"

Студент: Горошков М.Н.

Программа "Цифровые профессии. Разработчик. Программист"

Санкт-Петербург

2023 г.

Содержание

1.	Введение.....	1
2.	Основная часть.....	3
	I. Обзор библиотеки Pygame.....	3
	II. Создание проекта в среде PyCharm.....	6
	III. Создание окна и отображение графики.....	8
	IV. Работа с изображениями и звуком.....	10
	V. Создание игровых объектов.....	15
	VI. Создание меню и интерфейса пользователя.....	24
	VII. Реализация игровой логики и управление объектами.....	29
	VIII. Применение нейросетевых моделей.....	41
	IX. Сборка приложения.....	45
3.	Заключение.....	48
4.	Список используемой литературы и ресурсов.....	50
5.	Приложения:.....	51
	I. Примеры промптов и результатов в Stable Diffusion 1.5.....	52
	II. Примеры промптов и ответов в чат-помощнике Bing (GPT-4)...	59
	III. Исходный код игры.....	63

1. Введение

В данном проекте будет рассмотрена разработка игры в жанре **platformer** с элементами **rpg** на языке программирования **Python**. Жанр platformer/rpg сочетает в себе элементы двух популярных жанров игр: платформеров и rpg. Платформеры - это игры, в которых игрок управляет персонажем, который должен преодолевать препятствия и сражаться с врагами на пути к цели. RPG - это аббревиатура на английском языке, которую можно расшифровать как Role-Playing Game, что переводится буквально как «ролевая игра». Ролевая игра - это жанр компьютерных игр, основанный на элементах игрового процесса традиционных настольных ролевых игр. В таких играх игроки могут выбирать своих персонажей и управлять ими в вымышленном мире. Игроки могут взаимодействовать с другими персонажами и окружающей средой, выполнять задания и развивать своих персонажей. В ходе работы будет рассмотрено создание игровых объектов, реализация игровой логики и управление объектами, а также создание меню и интерфейса пользователя.

Целью данного проекта является более глубокое изучение возможностей языка Python, так как изучение Python может помочь мне улучшить свои навыки программирования и открыть для себя новые возможности. Python широко используется в различных областях, таких как веб-разработка, научные вычисления, анализ данных и машинное обучение. Python также является одним из самых популярных языков программирования в мире. В ходе дальнейшего обучения после программы "Разработчик" по направлению "Data science" Python будет основным языком программирования для меня.

Data science также может быть полезен для разработки игр в нескольких аспектах. Например, вы можете использовать анализ данных для определения того, какие элементы игры наиболее популярны среди пользователей, и использовать эту информацию для улучшения игрового процесса и увеличения удовлетворенности пользователей. Также вы можете использовать машинное

обучение для создания более интеллектуальных противников в игре. Например, вы можете использовать алгоритмы машинного обучения для обучения ИИ противника на основе поведения пользователя и создания более реалистичного оппонента.

В данном же проекте я буду использовать возможности современных диффузных нейросетевых моделей для создания изображений для визуальной части игры, а также языковых моделей для написания сюжета.

2. Основная часть

1. Обзор библиотеки Pygame

Pygame - это библиотека модулей для языка программирования Python, созданная для разработки 2D игр. Pygame может называться фреймворком. Она предоставляет разработчикам множество инструментов для создания игр, включая графический движок, звуковой движок и механизмы управления событиями. Pygame позволяет создавать игры на Python с использованием простых и понятных конструкций языка программирования Python. Библиотека Pygame имеет открытый исходный код и доступна для бесплатного скачивания и использования. Я хочу создать игру, которая будет увлекательной и интересной для пользователей.

Pygame добавляет функциональность поверх превосходной библиотеки SDL. Это позволяет создавать полнофункциональные игры и мультимедийные программы на языке python. Pygame очень портативна и работает практически на любой платформе и операционной системе. Сама Pygame была скачана миллионы раз. Pygame бесплатна. Выпущенный по лицензии LGPL, вы можете создавать с его помощью игры с открытым исходным кодом, бесплатные, условно-бесплатные и коммерческие.

Многоядерные процессоры можно легко использовать. С распространенными двухъядерными процессорами и дешевыми 8-ядерными процессорами, доступными в настольных системах, что позволяет использовать многоядерные процессоры. Использует оптимизированный C и ассемблерный код для основных функций. Код на C часто в 10-20 раз быстрее кода на python, а ассемблерный код легко может быть в 100 и более раз быстрее кода на python.

Поставляется со многими операционными системами. Не нужно возиться с установкой его вне менеджера пакетов вашей операционной

системы. Поставляется с установщиками (и деинсталляторами) бинарных pos-систем для Windows или MacOSX. Для установки Rугame не требуются инструменты настройки даже с stypes.

Действительно портативна. Поддерживает Linux (ругame поставляется с большинством основных дистрибутивов Linux), Windows (95, 98, ME, 2000, XP, Vista, 64-разрядные Windows и т.д.), Windows CE, BeOS, macOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, и QNX. Код содержит поддержку AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS и OS/2, но они официально не поддерживаются. Вы можете использовать его на портативных устройствах, игровых консолях и компьютере "Один ноутбук на ребенка" (OLPC).

Rугame прост и удобен в использовании. Дети и взрослые создают стрелялки с помощью ругame. Ругame используется в проекте OLPC и преподается на курсах эссе для маленьких детей и студентов колледжей. Он также используется людьми, которые впервые программировали на z80 assembler или c64 basic.

Было опубликовано много игр. Включая финалистов Фестиваля инди-игр, финалистов Австралийского фестиваля игр, популярное условно-бесплатное программное обеспечение, мультимедийные проекты и игры с открытым исходным кодом. Более 660 проектов были опубликованы на веб-сайтах ругame, таких как: требуется список. Еще много игр было выпущено с SDL (на котором основана ругame), так что вы можете быть уверены, что большая часть из них была хорошо протестирована миллионами пользователей.

Вы управляете своим основным циклом. Вы вызываете функции ругame, они не вызывают ваши функции. Это дает вам больший контроль при использовании других библиотек и для различных типов программ.

Не требует графического интерфейса для использования всех функций. Вы можете использовать rugame из командной строки, если хотите использовать его только для обработки изображений, ввода с джойстика или воспроизведения звуков.

Небольшой объем кода библиотеки. В нем нет сотен тысяч строк кода для вещей, которые вы все равно не будете использовать. Ядро сохранено простым, а дополнительные функции, такие как библиотеки графического интерфейса и эффекты, разрабатываются отдельно за пределами rugame.

Rugame модульный. Вы можете использовать фрагменты rugame отдельно. Хотите использовать другую звуковую библиотеку? Это прекрасно. Многие из основных модулей могут быть инициализированы и использованы отдельно.

Я буду использовать в проекте множество инструментов, предоставляемых Rugame, для создания графического движка, звукового движка и механизмов управления событиями.

II. Создание проекта в среде PyCharm

PyCharm — это кроссплатформенная интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains на основе IntelliJ IDEA. Предоставляет пользователю комплекс средств для написания кода и визуальный отладчик.

Продукт доступен в двух версиях: PyCharm Community Edition — бесплатная версия, находится под лицензией Apache License, и PyCharm Professional Edition — расширенная версия продукта, обладающая дополнительной функциональностью, является проприетарным ПО. В данном проекте используется бесплатная версия - PyCharm Community Edition.

Все, что вы делаете в PyCharm, выполняется в контексте проекта. Он служит основой для поддержки кодирования, рефакторинга, согласованности стиля кодирования и т.д. Создадим проект и для этого выберем пункт New Project. Далее нам откроется окно для настройки проекта. В поле Location необходимо указать путь к проекту. В моем случае проект будет помещаться в папку Diplom. Собственно название папки и будет названием проекта. Кроме пути к проекту все остальные настройки оставим по умолчанию и нажмем на кнопку Create для создания проекта. После этого будет создан пустой проект. В центре среды будет открыт файл main.py с некоторым содержимым по умолчанию.

При создании проекта автоматически создается и запускается виртуальная среда. Виртуальная среда - это изолированное окружение Python, которое позволяет управлять зависимостями и настройками конкретного проекта независимо от других проектов. Виртуальная среда создается для того, чтобы изолировать проект от других проектов и от системных библиотек. Это позволяет иметь разные версии библиотек для разных проектов.

Для установки Pyspace в нашу виртуальную среду можно использовать инструмент pip (который используется в Python для установки пакетов). Вот

команда для установки Pygame, которая вводится во внутреннем терминале PyCharm:

```
pip install pygame
```

III. Создание окна и отображение графики

Для создания окна и отображения графики в pygame необходимо выполнить следующие шаги:

Импортировать библиотеку pygame и инициализировать ее.

```
import pygame
```

```
pygame.init()
```

Создать окно с помощью метода

```
pygame.display.set_mode()
```

В качестве аргументов передаются размеры окна в пикселях.

```
screen = pygame.display.set_mode((width, height))
```

Отобразить окно на экране

```
pygame.display.flip()
```

Для отображения графики на окне необходимо использовать методы рисования pygame. Например, чтобы нарисовать простой прямоугольник на окне, можно использовать следующий код:

```
pygame.draw.rect(screen, color, (x,y,width,height))
```

Где screen - это поверхность для рисования (окно), color - цвет прямоугольника в формате RGB или RGBA, (x,y) - координаты верхнего левого угла прямоугольника, width и height - ширина и высота прямоугольника соответственно.

pygame.display.set_caption("заголовок") - это метод, который устанавливает заголовок окна Pygame. Заголовок окна - это текст, который отображается в верхней части окна. Например, в моей игре заголовок "Escape from Zefira"

`pygame.display.set_icon()` - это метод, который устанавливает значок окна Pygame. Значок окна - это изображение, которое отображается в верхней левой части окна и в списке задач операционной системы. Например, если вы хотите установить значок окна "icon.png", вы можете использовать следующий код:

```
icon = pygame.image.load("icon.png")
```

```
pygame.display.set_icon(icon)
```

IV. Работа с изображениями и звуком

Для работы с изображениями в Pygame используется модуль `pygame.image`. С помощью этого модуля вы можете загружать изображения и изменять их размеры.

Для изображений в игре создадим в нашем проекте отдельную папку 'images'.

`pygame.image.load()` - это метод, который загружает изображение из файла и возвращает объект `Surface`. Вы можете использовать этот объект для отображения изображения на экране.

Например, для загрузки фона стартового экрана я использовал использовать следующий код:

```
start_bg = pygame.image.load('images/start_screen.png').convert_alpha()
```

`convert_alpha()` - это метод, который изменяет формат пикселей изображения на формат с альфа-каналом. Это позволяет использовать прозрачность в изображении.

Что-бы отобразить изображение на экране в точке (x, y), вы можете использовать следующий код:

`screen.blit(image, (x, y))` - это метод `blit()`, который копирует содержимое одной поверхности на другую поверхность.

Метод `subsurface()` в Pygame используется для создания новой поверхности, которая ссылается на родительскую поверхность. Например, если у вас есть изображение `image`, и вы хотите получить подповерхность изображения в прямоугольной области (x, y, width, height), вы можете использовать следующий код:

```
subsurface = image.subsurface(pygame.Rect(x, y, width, height))
```

Таким образом можно использовать spritesheets. Spritesheets - это изображения, которые содержат несколько спрайтов (небольших изображений), которые могут быть использованы в качестве анимации. В Pygame вы можете загрузить спрайты из спрайт-листа в объект PygAnimation без необходимости нарезки листа на отдельные файлы изображений. Имя файла спрайт-листа передается в функцию `getImagesFromSpriteSheet()`, которая возвращает список объектов `pygame.Surface`. Так создаются анимации персонажа.

Пример создания анимации из таблицы спрайтов:

```
spritesheet = pygame.image.load('images/flash_new.png')

walk_frame_width = 43

walk_frame_height = 78

walk_frame_count = 12

walk_pos = [(10 + i * (walk_frame_width + 1), 98) for i in
range(walk_frame_count)]

walk_right = [spritesheet.subsurface(pygame.Rect(pos, (walk_frame_width,
walk_frame_height))) for pos in walk_pos]

walk_left = [pygame.transform.flip(spritesheet.subsurface(pygame.Rect(pos,
(walk_frame_width, walk_frame_height))), True, False) for pos in walk_pos]
```

Этот код загружает изображение и разбивает его на кадры анимации. Он создает список кадров для движения вправо и влево. Каждый кадр имеет ширину 43 и высоту 78. Всего есть 12 кадров. Переменная `walk_pos` содержит координаты каждого кадра на изображении. Затем он создает список кадров для движения вправо и влево, используя метод `subsurface()` для получения каждого кадра из изображения.

Для воспроизведения анимации внутри игрового цикла можно использовать такой код:

```
screen.blit(walk_right[walk_anim_count], (player_x, player_y))

right_orient = True

player_x += walk_speed

walk_anim_timer += 1

if walk_anim_count == 11 and walk_anim_timer == 3:

    walk_anim_count = 0

    walk_anim_timer = 0

elif walk_anim_timer == 3:

    walk_anim_count += 1

    walk_anim_timer = 0
```

Этот код использует метод `blit()` для отображения изображения на экране. Он отображает текущий кадр анимации для движения вправо с помощью переменной `walk_armed_anim_count`, которая изменяется когда переменная `walk_anim_timer == 3`. Переменная `walk_anim_timer` меняется с каждым проходом цикла. В данном случае число 3 показывает, сколько пройдет кадров до смены следующего кадра в игре. Координаты отображения на экране задаются переменными `player_x` и `player_y`.

Стоит отметить, что данный проект не является коммерческим, а я не имею навыков создания анимации. Поэтому я использовал уже готовые анимации персонажей из существующих игр. Источником анимаций является сайт <https://www.spritters-resource.com>. Обработка изображений и

анимаций, корректировка цветовой гаммы и прочие эффекты производились с помощью графического редактора Aseprite.

Aseprite - это редактор изображений для рисования и анимации в стиле пиксельного искусства. Он предназначен для создания спрайтов и анимации в играх. Он работает на Windows, macOS и Linux и содержит различные инструменты для редактирования изображений и анимации, такие как слои, кадры, поддержка tilemap, интерфейс командной строки, скрипты Lua и многое другое.

Элементы интерфейса, а также инвентаря главного героя, спрайты различных платформ и стен - вот неполный список того, что мне удалось сделать с помощью редактора Aseprite.

Для работы со звуком в Pygame используется модуль `pygame.mixer`. С помощью этого модуля вы можете загружать звуковые файлы и проигрывать их.

Для звуков и музыки в игре создадим в нашем проекте отдельную папку 'sounds'.

`pygame.mixer.Sound()` - это класс для загрузки звуковых файлов в Pygame. Он принимает имя файла или объект файла, в этом проекте используются файлы формата MP3. Затем вы можете использовать метод `play()` для воспроизведения звука.

Он может принимать необязательный аргумент `loops`, который указывает количество повторений звука. Если значение `loops` равно -1, то звук будет воспроизводиться бесконечно. Вы также можете использовать метод `set_volume()` для установки громкости звука.

Пример:

```
intro_sound = pygame.mixer.Sound('sounds/Overrated.mp3')
```

```
intro_sound.play(loops=-1)
```

Еще некоторые из возможностей `pygame.mixer`:

- Загрузка звуковых файлов в форматах WAV и OGG.
- Воспроизведение звуковых файлов с различными параметрами, такими как громкость и панорамирование.
- Управление каналами воспроизведения для создания сложных звуковых эффектов.
- Поддержка многопоточности для одновременного воспроизведения нескольких звуков

V. Создание игровых объектов

Создаем главного героя.

Загрузив изображение для персонажа,

```
player_stay_right = spritesheet.subsurface(pygame.Rect((0,6), (43, 78)))
```

назначаем его нашему герою:

```
player = player_stay_right
```

```
player_rect = player.get_rect(topleft=(player_x, player_y))
```

Этот код использует метод `get_rect()` для получения прямоугольника, который описывает границы изображения. Метод `get_rect()` возвращает объект `pygame.Rect`, который содержит координаты и размеры прямоугольника. Координаты прямоугольника задаются параметром `topleft`, который устанавливает координаты верхнего левого угла прямоугольника.

Эти параметры относятся к главному герою игры:

`walk_speed` - скорость передвижения героя без оружия

`walk_armed_speed` - скорость передвижения героя с оружием

`player_x` - координата x героя на экране

`player_y` - координата y героя на экране

`hp` - текущее количество здоровья героя

`hp_max` - максимальное количество здоровья героя

`player_damage` - урон, наносимый героем врагам

`experience` - количество опыта, полученного героем

level - уровень героя

Для создания врагов в игре , количество и параметры которых будут различаться удобно использовать классы, например:

```
class Soldier(pygame.sprite.Sprite):

    def __init__(self, x, y, bg_x, bg_y, lev):

        super().__init__()

        self.lev = lev

        self.x = x

        self.y = y

        self.bg_x = bg_x

        self.bg_y = bg_y

        self.alive = True

        self.damage = 20 * self.lev

        self.hp_max = 120 * self.lev

        self.hp = 120 * self.lev

        self.exp = 50 * self.lev

        self.soldier_stay_right = soldier_sheet.subsurface(pygame.Rect((10,2),
(76, 78)))

        self.soldier_stay_left =
pygame.transform.flip(soldier_sheet.subsurface(pygame.Rect((3,2), (57, 78))), True,
False)

    def decrease_hp(self, player_damage):
```

```
self.hp -= player_damage
```

```
if self.hp < 0:
```

```
    self.hp = 0
```

```
if self.hp > self.hp_max:
```

```
    self.hp = self.hp_max
```

Этот класс представляет врага в игре. Он наследуется от класса `pygame.sprite.Sprite`. Конструктор класса принимает следующие параметры:

`x` - координата `x` врага на экране

`y` - координата `y` врага на экране

`bg_x` - номер уровня, на котором будет отображаться враг, по горизонтали

`bg_y` - номер уровня, на котором будет отображаться враг, по вертикали

`lev` - уровень врага

Класс имеет следующие атрибуты:

`alive` - флаг, указывающий, жив ли враг

`damage` - урон, наносимый врагом игроку

`hp_max` - максимальное количество здоровья врага

`hp` - текущее количество здоровья врага

`exp` - количество опыта, получаемого за убийство врага

Класс имеет методы:

`decrease_hp()` - уменьшает количество здоровья врага на заданное количество

Также класс имеет два атрибута:

`soldier_stay_right` - изображение спрайта врага, когда он стоит и смотрит направо

soldier_stay_left - изображение спрайта врага, когда он стоит и смотрит налево

Далее можно создать экземпляр данного класса:

```
soldier_1 = Soldier(400, 22, 4, 1, 1)
```

Затем создается список `soldiers_full_list`, который будет содержать всех врагов данного класса в игре:

```
soldiers_full_list = [soldier_1]
```

Также создается пустой список врагов, участвующих в игре в данный момент, он будет меняться:

```
soldiers_list_in_game = []
```

Создание платформ в игре

Для платформ удобно будет создать отдельный класс:

```
class Platform(pygame.sprite.Sprite):  
  
    def __init__(self, x, y, width, height, color):  
  
        super().__init__()   
  
        self.image = pygame.Surface((width, height))  
  
        self.image.fill(color)  
  
        self.rect = self.image.get_rect()  
  
        self.rect.x = x  
  
        self.rect.y = y  
  
        self.width = width
```

```
self.height = height
```

Он содержит следующие атрибуты:

- x: координата x платформы на экране.
- y: координата y платформы на экране.
- width: ширина платформы.
- height: высота платформы.
- color: цвет платформы.

Конструктор класса инициализирует атрибуты объекта и создает поверхность `pygame.Surface` для отображения платформы на экране. Он также устанавливает начальные координаты платформы и ее размер.

Так как в игре буде 8 уровней по горизонтали и 2 по вертикали, создадим соответствующий двумерный массив с платформами:

```
platforms = [[pygame.sprite.Group() for j in range(2)] for i in range(8)]
```

Пример создания экземпляра класса:

```
platforms[0][0].add(Platform(120, 340, 510, 20, 'red'))
```

По тому-же принципу создаются стены в игре. А вот для создания уступов по краям платформ, на которые будет забираться главный герой используем цикл:

```
for i in range(8):  
  
    for j in range(2):  
  
        for platform in platforms[i][j]:  
  
            if platform.rect.x != 0:
```

```

        ledges_left[i][j].add(Platform(platform.rect.x-30,
platform.rect.y, 30, 20, 'green'))

        if platform.rect.x + platform.width <= 580:

            ledges_right[i][j].add(Platform(platform.rect.x +
platform.width, platform.rect.y, 30, 20, 'green'))

        try:

            for platform in platforms[i][j-1]:

                if platform.rect.x != 0 and platform.rect.y == 340:

                    ledges_left[i][j].add(Platform(platform.rect.x-30, -20, 30,
20, 'green'))

                    if platform.rect.x + platform.width <= 580 and platform.rect.y
== 340:

                        ledges_right[i][j].add(Platform(platform.rect.x +
platform.width, -20, 30, 20, 'green'))

        except:

            pass

```

В этом цикле также учтено, что уступы находящиеся на краю экрана должны быть видны на разных уровнях.

В основном цикле игры также можно пописать цикл, который заполнит платформы или стены нужным спрайтом с шагом 30 пикселей:

```

for wall in walls[bg_x][bg_y]:

    for i in range(wall.height // 30):

        screen.blit(tile_2, (wall.rect.x, wall.rect.y + i * 30))

```

Создание инвентаря героя в игре

В инвентаре героя будут находиться предметы, для которых создаем отдельный класс:

```
class Item(pygame.sprite.Sprite):  
  
    def init(self, name, image, x, y, bg_x, bg_y):  
  
        super().init()  
  
        self.name = name  
  
        self.image = image  
  
        self.x = x self.y = y  
  
        self.bg_x = bg_x  
  
        self.bg_y = bg_y  
  
        self.rect = self.image.get_rect(topleft=(x, y))
```

Класс Item - это класс `pygame.sprite.Sprite`, который представляет объект в игре. Он содержит следующие атрибуты:

- `name`: имя объекта.
- `image`: изображение объекта.
- `x`: координата x объекта на экране.
- `y`: координата y объекта на экране.
- `bg_x`: координата x фона на экране.
- `bg_y`: координата y фона на экране.

Конструктор класса инициализирует атрибуты объекта и создает поверхность `pygame.Surface` для отображения объекта на экране. Он также устанавливает начальные координаты объекта и его размер.

Создаем экземпляры класса, предварительно загрузив для них изображения:

```
drug = pygame.image.load('images/drug.png').convert_alpha()

key = pygame.image.load('images/card-key.png').convert_alpha()

drug_1 = Item('drug', drug, 150, 84, 0, 0)

key_1 = Item('key', key, 590, 84, 1, 0)
```

Помещаем экземпляры в общий список предметов в игре:

```
items_list_start = [drug_1, drug_2, drug_3, drug_4, drug_5, key_1, key_2,
key_3, key_4]
```

Создаем инвентарь героя в виде списка, который изначально пуст:

```
inventory_list = []
```

Для отображения инвентаря на экране можно использовать цикл. Также отобразим специальную рамку, которая будет показывать текущий выбранный предмет в инвентаре:

```
for i, item in enumerate(inventory_list):

    screen.blit(item.image, (590 - i * 20, 5))
```

```
screen.blit(border, (border_x, border_y))
```

Опишем также логику движения рамки по инвентарю:

```
if keys[pygame.K_m] and not is_delay:

    is_delay = True

    if len(inventory_list) > 1:

        if border_x == 588 - (len(inventory_list)-1) * 20:

            border_x = 588

            border_pos = 0
```


else:

border_x -= 20

border_pos += 1

Переменная `is_delay` необходима, чтобы создать задержку при нажатии клавиши, иначе нажатие может дублироваться при каждом проходе цикла. Величину задержки установим в 10 кадров:

if `is_delay`:

`delay_timer` += 1

if `delay_timer` == 10:

`is_delay` = False

`delay_timer` = 0

VI. Создание меню и интерфейса пользователя

Важным элементом интерфейса пользователя является обработка и отображение текстовой информации. Первым делом для этого нужно загрузить и подключить шрифты. Создадим в нашем проекте отдельную папку 'fonts'. На сайте <https://fonts.google.com/> выберем и скачаем подходящий шрифт и поместим его в нашу папку.

`pygame.font.Font()` - это функция Pygame, которая создает новый объект шрифта из файла. Модуль шрифтов позволяет рендерить шрифты TrueType в новом объекте Surface. Каждый символ UCS-2 (от 'u0001' до 'uFFFF') может быть отображен с помощью этого модуля¹².

В приведенном коде

```
label_0 = pygame.font.Font('fonts/Pangolin-Regular.ttf', 30)
```

Создается новый объект шрифта размером 30 из файла Pangolin-Regular.ttf.

Теперь можно создать текстовый объект данного шрифта. Например, для названия игры на стартовом экране используем такой код:

```
name_label = label_0.render("Escape from Zefira", False, (209, 151, 27))
```

В приведенном коде создает новый объект Surface с текстом "Escape from Zefira" и цветом (209, 151, 27) с помощью метода `render()` объекта шрифта `label_0`.

Если подразумевается взаимодействие пользователя с текстовым объектом, создается объект поверхности:

```
restart_label_rect = restart_label.get_rect(topleft=(240, 90))
```

В данном проекте я захотел, чтобы кнопка старта игры переливалась разными цветами, для этого создаем лист с несколькими объектами разного цвета:

```
start_label = [label.render("Start game", False, (150 + i * 20, 151, 27)) for i
in range(5)]
```

Далее в цикле стартового экрана будем перебирать эти объекты с помощью переменной `start_label_count` и отображать на экране:

```
start_label_rect = start_label[start_label_count].get_rect(topleft=(240, 90))
```

```
screen.blit(start_label[start_label_count], start_label_rect)
```

```
start_label_count += 1
```

```
if start_label_count == 5:
```

```
    start_label_count = 0
```

Нажатие на кнопку старта реализуем так:

```
mouse = pygame.mouse.get_pos()
```

```
if start_label_rect.collidepoint(mouse) and pygame.mouse.get_pressed()[0]:
```

```
    intro = False
```

```
    running = True
```

```
    scene = True
```

Метод `pygame.mouse.get_pos()` возвращает кортеж из двух значений - координаты `x` и `y` курсора мыши. Позиция относительно верхнего левого угла экрана.

Метод `collidepoint()` класса `pygame.Rect` проверяет, находится ли точка внутри прямоугольника. Метод принимает координаты точки и возвращает `True`, если точка находится внутри прямоугольника, иначе - `False`. Для использования метода необходимо создать объект класса `pygame.Rect`.

Метод `pygame.mouse.get_pressed()` возвращает кортеж, представляющий состояние всех кнопок мыши. Кортеж содержит 3 значения, каждое из которых представляет состояние левой, средней и правой кнопок мыши соответственно. Значение равно 1, если кнопка нажата, и 0, если она не нажата.

Нажатие на кнопку старта отключает флаг `intro = False`, прерывая цикл стартового меню и включает флаг основного цикла игры `running = True` и флаг сцены перед первым уровнем.

По тому же принципу создается кнопка, ведущая в меню управления в игре. В самом меню помимо описания кнопок управления есть кнопка возврата в стартовое меню:

```
back_start_menu_label_rect = back_start_menu_label.get_rect(topleft=(210, 300))
```

```
screen.blit(back_start_menu_label, back_start_menu_label_rect)
```

```
if back_start_menu_label_rect.collidepoint(mouse) and  
pygame.mouse.get_pressed()[0]:
```

```
    controls_menu = False
```

Отображение текста в сценах между уровнями реализована более интересным способом - с визуальными и звуковыми эффектами.

Для смены сцен будем использовать переменную `scene_count`. По завершении игроком уровня она будет увеличиваться на единицу. Для каждой сцены создается список строк отображаемого текста и загружается картинка фона:

```
scene_1_text = ['Главный герой игры, Алекс, молодой ученый, работающий ',  
                'над экспериментом в области телепортации на секретной',  
                'лабораторной базе.Во время одного из экспериментов',
```

'происходит неожиданная неудача, и Алекс оказывается',

'телепортированным в другой инопланетный мир']

```
bg_scene_1 = pygame.image.load('images/bg_scene_1.png').convert_alpha()
```

Для отображения текста с имитацией побуквенной печати текста на экран, как на печатной машинке с звуковым эффектом нажатия клавиши используем следующий код:

```
scene_label =  
label_2.render(eval(f'scene_{scene_count}_text')[scene_line][:scene_label_count],  
False, (209, 151, 27))
```

```
if scene_label_count == len(eval(f'scene_{scene_count}_text')[scene_line]):
```

```
    scene_label_count = 1
```

```
    scene_line += 1
```

```
key_sound.play()
```

```
screen.blit(scene_label, (50, scene_y))
```

```
scene_label_count += 1
```

По завершении печати текста сцена завершается с задержкой 2 секунды:

```
if scene_line == len(eval(f'scene_{scene_count}_text')):
```

```
    scene = False
```

```
    gameplay = True
```

```
    running = True
```

```
    clock.tick(0.5)
```

```
    scene_label_count = 1
```

```
scene_line = 0
```

Пятая сцена в игре является финальной, после нее программа завершается. Для этого в основном цикле должно быть прописано условие:

```
if scene_count == 5:
```

```
    running = False
```

```
    pygame.quit()
```

По тем же шаблонам можно создать экран для рестарта игры в случае гибели персонажа:

```
screen.blit(lose_label, (260, 20))
```

```
screen.blit(restart_label, restart_label_rect)
```

```
mouse = pygame.mouse.get_pos()
```

```
if restart_label_rect.collidepoint(mouse) and pygame.mouse.get_pressed()[0]:
```

```
    gameplay = True
```

VII. Реализация игровой логики и управление объектами

Для контроля за временем в игре используется `pygame.time.Clock()`.

`pygame.time.Clock()` - это объект, который помогает нам убедиться, что наша программа работает с определенной максимальной частотой кадров в секунду. Этот объект `Clock` будет гарантировать, что наши игровые программы не работают слишком быстро, вставляя небольшие паузы на каждой итерации игрового цикла. Если бы у нас не было этих пауз, наша игровая программа работала бы так быстро, как это может делать компьютер.

Установить количество кадров в секунду (fps) можно поместив в цикл:

```
clock.tick(fps)
```

Основной цикл игры в Pygame - это бесконечный цикл, который обрабатывает события и обновляет экран. В своей основе он выглядит так:

```
while running:
```

```
    for event in pygame.event.get():
```

```
        if event.type == pygame.QUIT:
```

```
            pygame.quit()
```

```
    # Обновление экрана
```

```
    pygame.display.update()
```

Пока переменная `running = True`, игра продолжается. После обработки событий мы обновляем экран с помощью функции `pygame.display.update()`. Эта функция перерисовывает экран и отображает все изменения, которые были внесены в последнем кадре.

Внутри цикла будет описанный ранее подцикл, который будет активен во время сцен между уровнями игры:

while scene:

.....

Также главной частью цикла является условие

if gameplay and not scene:

.....

которое отвечает за активацию основного геймплея игры если сцены между уровнями не активны.

Главный герой игры будет выполнять отдельные действия. Для их фиксации используем различные булевы значения (флаги):

- is_busy: флаг, указывающий на то, что главный герой занят и не может выполнять другие действия.
- is_fall: флаг, указывающий на то, что главный герой падает.
- is_jump: флаг, указывающий на то, что главный герой прыгает.
- is_jump_up: флаг, указывающий на то, что главный герой прыгает вверх.
- is_jump_down: флаг, указывающий на то, что главный герой прыгает вниз.
- is_arm: флаг, указывающий на то, что главный герой достает оружие.
- is_armed: флаг, указывающий на то, что у главной герой вооружен и может стрелять.
- is_shoot: флаг, указывающий на то, что главный герой стреляет.
- on_platform: флаг, указывающий на то, что главный герой находится на платформе.
- is_climb: флаг, указывающий на то, что главный герой карабкается вверх.
- is_climb_down: флаг, указывающий на то, что главный герой спускается вниз.
- is_wounded: флаг, указывающий на то, что главный герой ранен и падает.

- `is_getup`: флаг, указывающий на то, что главный герой встает после падения.
- `is_sit_down`: флаг, указывающий на то, что главный герой садится.
- `is_sit_up`: флаг, указывающий на то, что главный герой встает из сидячего положения.
- `is_sit`: флаг, указывающий на то, что главный герой сидит.
- `is_somersault`: флаг, указывающий на то, что главный герой делает кувырок.
- `is_sit_turn`: флаг, указывающий на то, что главный герой поворачивается в сидячем положении.
- `right_orient`: флаг ориентации персонажа.

Геймплей активен, пока у персонажа положительный уровень здоровья

```
if hp <= 0:
```

```
    gameplay = False
```

Также в процессе геймплея герой накапливает опыт. После набора определенного опыта увеличивается уровень персонажа, его здоровье и наносим им ущерб врагам

```
if experience >= (level**2) * 100:
```

```
    experience -= (level**2) * 100
```

```
    level += 1
```

```
    player_damage *= 1.2
```

```
    hp_max *= 1.1
```

```
    hp *= 1.1
```

Здоровье, уровень персонажа и накопленный опыт отображаются в верхней части экрана с помощью незамысловатых циклов:

```
for i in range(int(hp/10)):

    screen.blit(hp_line_icon, (45 + i * 5, 5))

level_label = label_3.render(f"Level: {level}", False, (175, 50, 50))

screen.blit(level_label, (200, 5))

screen.blit(exp_line_icon, (270, 9))

for i in range(int((experience/((level**2) * 100))*10)):

    screen.blit(exp_icon, (270 + i * 10, 9))
```

В процессе игры надо отслеживать: находится ли герой на платформе. Если не находится, то он должен падать. Если до этого он падал с большой высоты, необходимо уменьшить его здоровье. И в конце необходимо отцентрировать положение героя на высоту спрайта, которая отличается для сидячего и стоячего положения:

```
for platform in platforms[bg_x][bg_y]:

    if player_rect.colliderect(platform.rect):

        if falling_high > 8:

            is_getup = True

            hp -= 25

            falling_high = 0

        if is_fall:

            if right_orient:
```

```

        player_x += 25

    if not right_orient:

        player_x -= 25

    is_fall = False

    falling_high = 0

    is_busy = False

    if not is_sit:

        player_y = platform.rect.top - 77

    else:

        player_y = platform.rect.top - 49

    falling_anim_count = 0

    on_platform = True

```

Переменная `falling_high` рассчитывается в цикле падения персонажа:

```

if not on_platform and not is_jump_up and not is_jump_down and not is_jump:

    is_fall = True

    is_busy = True

    is_somersault = False

    if right_orient:

        screen.blit(falling_right[falling_anim_count], (player_x, player_y))

    if not right_orient:

```

```

        screen.blit(falling_left[falling_anim_count], (player_x, player_y))

    falling_anim_timer += 1

    if falling_anim_count == 7 and falling_anim_timer == 3:

        falling_high += 1

        falling_anim_timer = 0

        player_y += 52

    elif falling_anim_timer == 3:

        falling_high += 1

        falling_anim_count += 1

        falling_anim_timer = 0

        player_y += 0.5 * falling_anim_count ** 2

```

Как можно заметить переменная `player_y` меняется квадратически, что имитирует ускорение свободного падения. Также на время падения включаются флаги `is_fall` и `is_busy`, что говорит о том, что герой не может выполнять другие действия (например, стрелять). Если до этого герой производил кувырок `is_somersault`, эта операция прерывается, флаг становится `is_somersault = False`. Соответственно, для левостороннего и правостороннего положения во время падения проигрывается определенная анимация в координатах героя.

Чтобы персонаж реагировал на нажатия клавиш будем использовать модуль `pygame.key`. Он содержит функции для работы с клавиатурой. Когда кнопки на клавиатуре нажимаются и отпускаются, события `pygame.KEYDOWN` и `pygame.KEYUP` вставляются в очередь событий. Атрибут `key` - это

целочисленный идентификатор, представляющий каждую клавишу на клавиатуре

Например, для перемещения персонажа влево -

```
elif keys[pygame.K_LEFT] and not is_busy and not is_sit and not is_armed:
```

```
    screen.blit(walk_left[walk_anim_count], (player_x, player_y))
```

```
    right_orient = False
```

```
    player_x -= walk_speed
```

где walk_speed - скорость перемещения персонажа.

При этом, если персонаж в положении сидя и не может идти, то он может совершить кувырок, но при этом совершить разворот, если находится в противоположном положении:

```
elif keys[pygame.K_LEFT] and not is_busy and is_sit and not is_sit_turn:
```

```
    if right_orient:
```

```
        is_busy = True
```

```
        is_sit_turn = True
```

```
    else:
```

```
        is_busy = True
```

```
        is_somersault = True
```

Отдельного внимания заслуживает логика осуществления стрельбы героем. Стрельба происходит, если предварительно персонаж достал пистолет (флаг is_armed = True) и он ничем не занят:

```
elif keys[pygame.K_b] and is_armed and not is_busy:
```

```
if right_orient:
```

```
    bullets_right.append(bullet.get_rect(topleft=(player_x, player_y)))
```

```
else:
```

```
    bullets_left.append(bullet.get_rect(topleft=(player_x, player_y)))
```

```
is_shoot = True
```

```
is_busy = True
```

Здесь `bullets_right` и `bullets_left` списки для пуль летящих в правую и левую сторону соответственно. После включения флага, проигрывается звук выстрела и соответствующая анимация, флаги снимаются

```
elif is_armed and is_shoot:
```

```
    if shoot_anim_count == 2:
```

```
        blaster_sound.play()
```

```
        .....
```

```
    is_shoot = False
```

```
    is_busy = False
```

Сами пули с каждым обновлением экрана меняют свою координату:

```
if bullets_right:
```

```
    for (i, el) in enumerate(bullets_right):
```

```
        screen.blit(bullet, (el.x, el.y))
```

```
        el.x += 15
```

Логика попадания пуль в различные объекты и их уничтожение выглядит примерно так:

```
if bullets_right:

    for (i, el) in enumerate(bullets_right):

        for wall in walls[bg_x][bg_y]:

            if el.collidect(wall.rect):

                try:

                    bullets_right.pop(i)

                except:

                    pass

        if el.x < 0 or el.x > screen_width:

            try:

                bullets_right.pop(i)

            except:

                pass
```

Не многим отличается логика попадания пули во врага:

```
if archers_list_in_game:

    for (index, archer) in enumerate(archers_list_in_game):

        if archer.alive and

el.collidect(archer.archer_stay_left.get_rect(topleft=(archer.x, archer.y))):

            archer.decrease_hp(player_damage)
```

```

if archer.hp <= 0:

    archer.hp = archer.hp_max

    experience += archer.exp

    archer.alive = False

try:

    bullets_right.pop(i)

except:

    pass

```

Перебрав всех врагов в списке, проверяем их на коллизию с объектом пули с помощью метода `.colliderect()`, если соприкосновение происходит, убавляем здоровье врага. Если здоровье меньше или равно нулю, меняем для данного экземпляра врага флаг `archer.alive` (живой) на `False`.

По той же логике происходит взаимодействие героя со снарядами, выпускаемыми врагами.

Смена уровня игры происходит, когда герой пересекает границы экрана, например:

```

if player_y < 0 and bg_y != 0:

    bg_y -= 1

    player_y = player_y + screen_height

```

Также смена локации также происходит, когда герой использует ключ из инвентаря напротив двери:

```

for i, item in enumerate(inventory_list):

```



```

if item.name == 'key':

    for door in doors[bg_x][bg_y]:

        if player_rect.colliderect(door.rect):

            if scene_count < 4:

                bg_x += 1

                scene = True

                scene_count += 1

                key_card_sound.play()

                player_x = 30

                inventory_list.pop(i)

```

При этом проигрывается звук открывания двери `key_card_sound` и включается флаг проигрывания сцены между уровнями `scene`. Использованный ключ удаляется из инвентаря `inventory_list.pop(i)`

Похожим образом реализовано использование игроком лекарства:

```

for i, item in enumerate(inventory_list):

    if border_pos == i:

        if item.name == 'drug':

            hp += 50

            if hp > hp_max:

                hp = hp_max

            sip_sound.play()

```

```
inventory_list.pop(i)
```

В этом пункте я описал только основные механики для реализации игровой логики, сам код игры имеет чуть более сложную и витиеватую структуру.

VIII. Применение нейросетевых моделей

Работа с графикой

Создание высококачественных изображений на основе текстовых описаний - сложная задача. Это требует глубокого понимания основного значения текста и способности генерировать изображение, соответствующее этому значению. В последние годы диффузионные модели стали мощным инструментом для решения этой проблемы. Эти модели позволяют невероятно легко создавать высококачественные изображения в различных стилях, используя всего несколько слов.

Stable Diffusion - это программное обеспечение, создающее изображения по текстовым описаниям с открытым исходным кодом. Оно основано на нейросети, которая была разработана компанией Stability AI под руководством Эмада Мостака. Stable Diffusion был выпущен в августе 2022 года и использует технологию диффузии для генерации реалистичных изображений. Сообщество разработчиков с открытым исходным кодом было очень активным с момента его выпуска. За короткий промежуток времени сообщество выпустило несколько стабильных диффузионных моделей с открытым исходным кодом, доработанных для различных наборов данных, стилизованных под художественные. Можно свободно использовать эти модели и создавать новые изображения, используя эти стили. Они могут варьироваться от японского аниме и футуристических роботов до киберпанковских миров.

Полная архитектура стабильной диффузии состоит из трех моделей:

- Кодировщик текста, который принимает текстовое приглашение.
- Преобразование текстовых подсказок в машиночитаемые векторы.
- U-Net

- Это диффузионная модель, отвечающая за генерацию изображений.
- Вариационный автоэнкодер, состоящий из модели кодера и декодера.
 - Кодировщик используется для уменьшения размеров изображения.
 - Диффузионная модель UNet работает в этом меньшем измерении.
 - Затем декодер отвечает за улучшение / восстановление изображения, сгенерированного диффузионной моделью, до его первоначального размера.

Кроме того, если у вас есть вычислительные ресурсы, вы также можете настроить стабильную диффузию для запуска в вашей системе, следуя документации из открытых репозиториях.

В своем проекте для генерации задних фонов к игре я использовал Stable Diffusion версии 1.5 в онлайн-сервисе <https://playgroundai.com/>.

Используя диффузионные модели, мы можем генерировать изображения как условно, так и безоговорочно. Безусловная генерация изображений просто означает, что модель преобразует шум в любую “случайную репрезентативную выборку данных”. Процесс генерации не контролируется и не управляется, и модель может генерировать изображение любого характера. Условная генерация изображений - это когда модели предоставляется дополнительная информация с помощью текста (text2img) или меток классов (как в CGANs). Это случай управляемой или управляемой генерации изображений. Предоставляя дополнительную информацию, мы ожидаем, что модель будет генерировать определенные наборы изображений.

Запрос в нейросеть осуществляется с помощью промптов. Промпт - это запрос, подсказка, или инструкция – те вводные данные, которые вы набираете, когда общаетесь с нейросетью. От правильного промпта, то есть корректного запроса, зависит то, насколько релевантной будет информация на

выходе. Примеры используемых мной промптов и результатов представлены в приложении 2.

Работа с текстом

Сюжет в играх может быть важным элементом игрового процесса. Он может помочь игрокам лучше понять мир игры и создать более глубокую связь с персонажами. Однако, сюжет не всегда является необходимым элементом игры. Некоторые игры могут быть очень успешными без сильного сюжета. Например, в некоторых играх сюжет может быть простым и линейным, но все еще быть увлекательным и интересным. В других играх сюжет может быть более сложным и многоуровневым, что может привести к более глубокому погружению в мир игры. В целом, важность сюжета зависит от жанра игры и предпочтений игрока.

Для написания сюжета я использовал возможности современных языковых моделей. Языковая модель состоит из нейронной сети со множеством параметров (обычно миллиарды весовых коэффициентов и более), обученной на большом количестве неразмеченного текста с использованием обучения без учителя. Она может использоваться для генерации текста, перевода текста на другой язык и других задач. Языковые модели генерируют вероятности путём обучения на корпусе текстов на одном или нескольких языках. Языковые модели - это большие нейронные сети, которые обучаются предсказывать следующее слово (или часть слова) в тексте с учетом предыдущего контекста. Языковые модели используются в различных областях, таких как машинный перевод, распознавание речи и генерация текста.

В Microsoft Bing Chat, который я использовал, встроена передовая языковая модель GPT-4. GPT-4 (Generative Pre-trained Transformer 4) — мультимодальная большая языковая модель, созданная OpenAI, четвёртая в серии GPT. Она была выпущена 14 марта 2023 года.

Примеры используемых мной промптов и ответы представлены в приложении 3.

IX. Сборка приложения

Для сборки приложения будем использовать PyInstaller.

PyInstaller - это инструмент для создания автономных (standalone) исполняемых файлов Python для Windows, Linux и Mac OS X, которые можно запускать без установки Python или дополнительных библиотек. Он упаковывает все необходимые зависимости, библиотеки и модули в один исполняемый файл, который может быть запущен на другой машине без необходимости установки Python и всех необходимых модулей. PyInstaller поддерживает многие сторонние библиотеки и позволяет оптимизировать размер результирующего файла.

Для упаковки проекта с множеством файлов и зависимостей в pyinstaller необходимо выполнить следующие шаги:

1. Необходимо установить PyInstaller, используя следующую команду в командной строке:

```
pip install pyinstaller
```

2. Создать файл `spec` для проекта, который перечисляет все файлы и зависимости, на которые имеется ссылка в вашем коде. Для создания файла `spec` используется команда:

```
pyinstaller --name=main main.py
```

3. Заменить `main` на название проекта, а `main.py` на имя файла, который должен быть запущен, чтобы начать работу проекта. PyInstaller автоматически выполнит анализ кода и определит все зависимости и файлы, которые должны быть включены в пакет.

4. Открыть файл `spec` в редакторе кода. В этом файле можно добавить дополнительные ресурсы, на которые проект ссылается, в раздел `datas`:

```
a = Analysis(['main.py'],
```

```
pathex=['/path/to/my/project'],  
  
binaries=[],  
  
datas=(['/path/to/my/project/data/', 'data/']),  
  
hiddenimports=[],  
  
hookspath=[],  
  
runtime_hooks=[],  
  
excludes=[],  
  
win_no_prefer_redirects=False,  
  
win_private_assemblies=False,  
  
cipher=None,  
  
noarchive=False)
```

В этом примере мы добавляем папку `data/` в наш проект и помещаем ее в папку, созданную PyInstaller для упаковки наших нужных файлов. Ресурсы, на которые ссылается проект можно также добавить вручную после сборки (не подходит для сборки в один файл).

Запустить PyInstaller с использованием файла `spec` и дополнительной командой `--onefile`, чтобы создать единственный исполняемый файл:

```
pyinstaller main.spec --onefile
```

После завершения процесса упаковки файл `.exe` будет находиться в папке `dist` нашего проекта. Можно убедиться, что пакет содержит все зависимости и ресурсы, используя следующую команду:

```
pyinstaller --name=main --add-data "data/*;data/" main.py
```


Здесь мы добавляем папку `data/` при помощи `--add-data` параметра в наш пакет.

После этих операций в зависимости от использования дополнительной команды `--onefile` или нет в папке `dist` появится либо приложение собранное в один исполняемой, либо исполняемый файл с дополнительными ресурсами.

3. Заключение

В данном проекте была создана игра для операционной системе Windows. Игра написана на языке Python версии 3.10 с применением свободно распространяемой библиотеки для создания игр Pygame. Жанр - платформер с элементами ролевой игры.

В процессе создания игры я использовал интегрированную среду разработки (IDE) PyCharm Community Edition 2023, укрепил навыки создания проекта и настройки виртуальной среды окружения. Рассмотрел процесс установки дополнительных библиотек в виртуальную среду.

Также мною были рассмотрены многочисленные нюансы работы с библиотекой Pygame. Был описан процесс создания окна игры, вывода информации на экран. Я научился загружать в проект изображения, обрабатывать их и выводить на экран, описал, как можно использовать таблицы спрайтов и составлять из них анимации. В процессе работы с графикой я также ознакомился с графическим редактором Aseprite. Мною был описан процесс загрузки и использования звуковых файлов, создания звуковых эффектов, которые оживили игру и сделали ее более увлекательной.

В проекте я рассказал, как можно создавать различные игровые объекты в игре, применяя методы объектно-ориентированного подхода, как объединять объекты в группы и проводить над ними различные действия. Таким образом были воссозданы такие объекты окружения как стены, платформы, уступы. Были сделаны различные типы врагов в игре, и предусмотрены механизмы их взаимодействия с героем и миром. Был проработан главный персонаж, обладающий различными способами передвижения и взаимодействия с окружением. Создан инвентарь героя, в который могут входить различные предметы.

В игре было реализовано стартовое меню с интерфейсом взаимодействия игрока и игры. Подача сюжета реализована с некоторыми визуальными и звуковыми эффектами.

Также я рассмотрел как можно применять современные нейросети для облегчения работы при создании игры. Диффузная модель Stable Diffusion 1.5 нарисовала для игры бэкграунды, а чат-помощник Bing придумал сюжет. В ходе дальнейшего обучения после программы "Разработчик" по направлению "Data science" этот опыт точно не будет лишним для меня.

Для создания исполняемых файлов приложения был использован такой инструмент как PyInstaller. Мною рассмотрены основные моменты работы с ним.

В процессе создания проекта мои навыки программирования на языке Python прогрессировали. В личном плане ко мне пришло понимание того, что при наличии достаточного времени и других ресурсов возможно реализовать программный продукт любой сложности. Но при этом для создания большого проекта высокого уровня качества конечно нужна коллективная работа узкоспециализированных специалистов. Например, для создания игры помимо программистов нужны художники, аниматоры, геймдизайнеры, тестировщики и т.д.

Конечно данный проект - учебный, он не является коммерческим, но в дальнейшем я хотел бы его улучшать: добавлять новые уровни и виды врагов с развитым 'интеллектом', добавлять новые механики, и заменить анимации персонажей на уникальные. После этого проект имеет возможность стать коммерческим.

4. Список используемой литературы и ресурсов

1. Документация по языку Python (<https://www.python.org/doc/>).
2. Документация по библиотеке Pygame (<https://www.pygame.org/docs/>).
3. Библиотека Pygame. Статья (<https://habr.com/ru/articles/588605/>).
4. Документация по Pyinstaller (<https://pyinstaller.org/>)
5. Сайт, посвященный сбору, архивированию и оценке материалов из видеоигр (<https://www.sprisers-resource.com/>)
6. Шрифты для отображения текста (<https://fonts.google.com/>)
7. Введение в диффузионные модели для генерации изображений – полное руководство. Статья (<https://habr.com/ru/articles/713076/>)
8. Большая языковая модель. Статья (<https://ru.wikipedia.org/>)

5. Приложения

III. Исходный код игры представлен в репозитории по адресу

<https://github.com/MikeGoroshkov/Diploma>

следующими файлами:

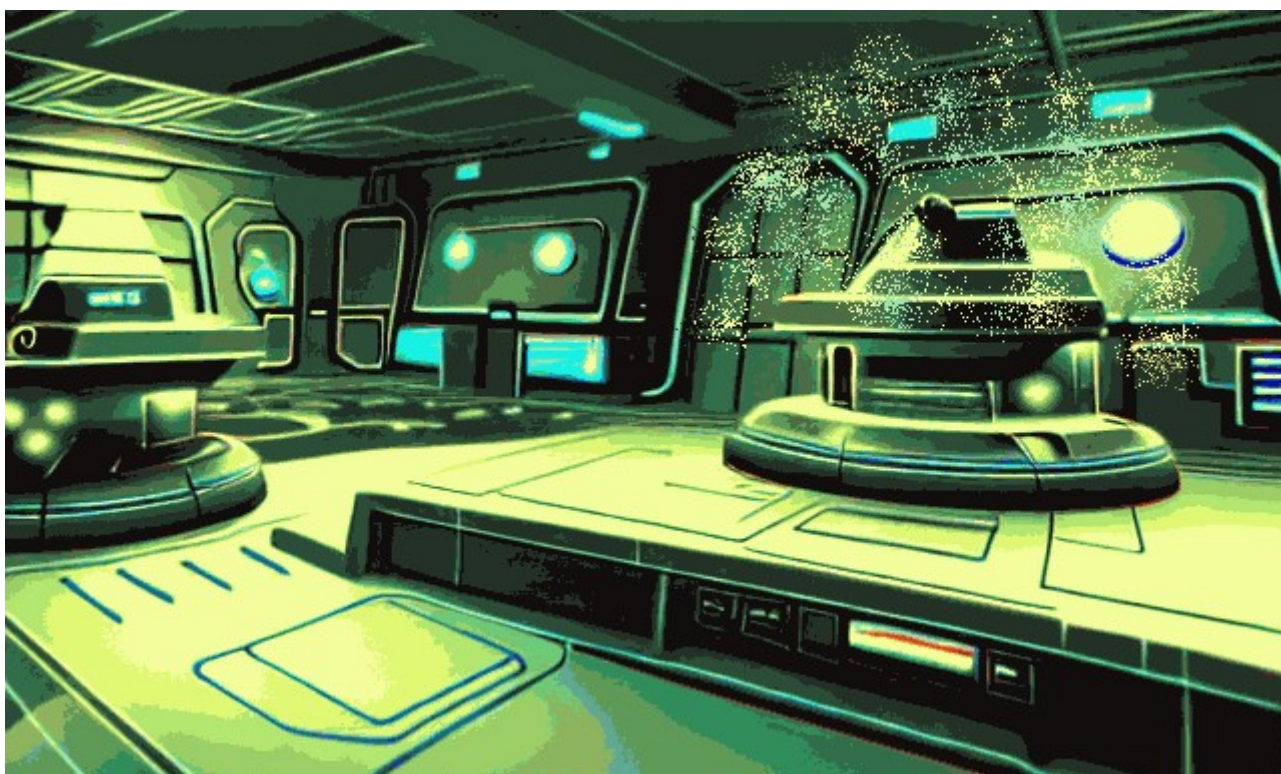
- main.py
- interface.py
- inventory.py
- hero.py
- levels.py
- enemies.py
- sounds.py

Приложение 1. Примеры промптов и результатов в Stable Diffusion 1.5

science-fiction-a-man-in-a-purple-jacket-with-a-blaster-on-the-background-of-the-city-of-the-future



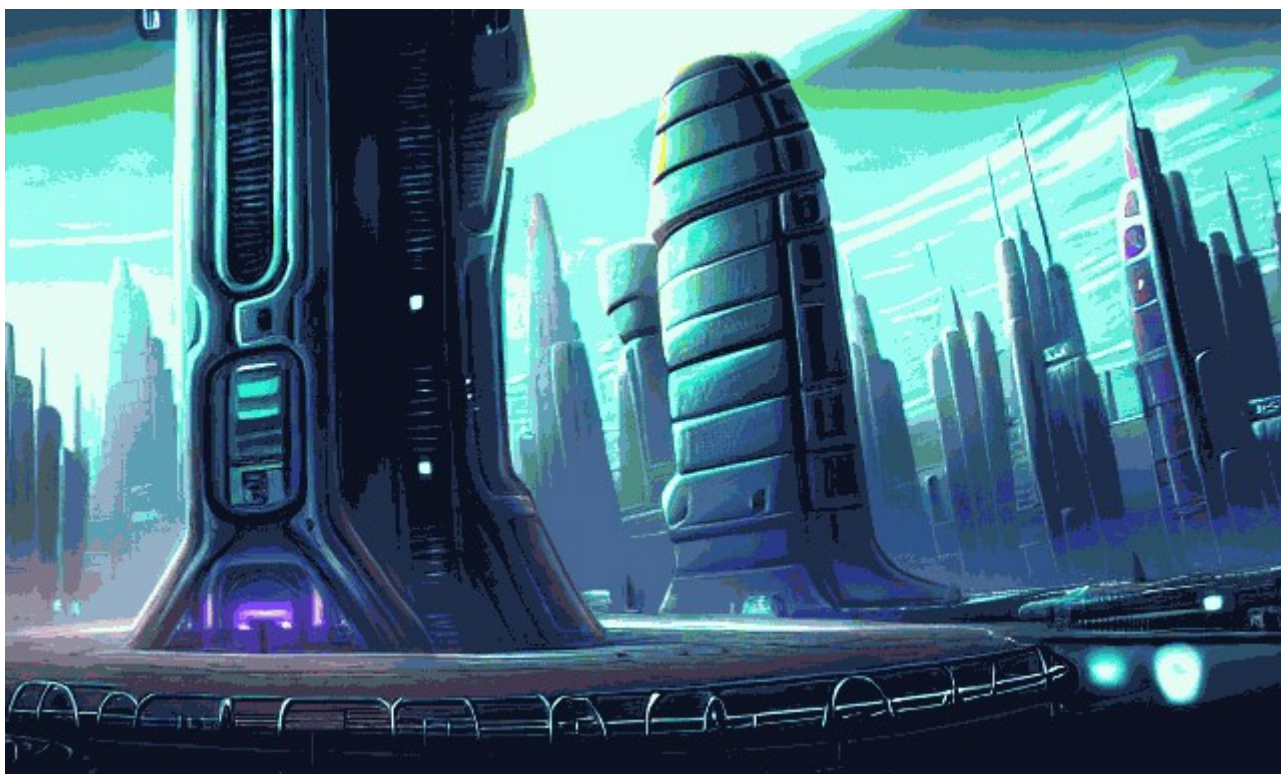
aliens-physcis-laboratory-background-for-sci-fi-game



cartoon-sci-fi-alien-planet-background



aliens-city-far-away-background-for-game

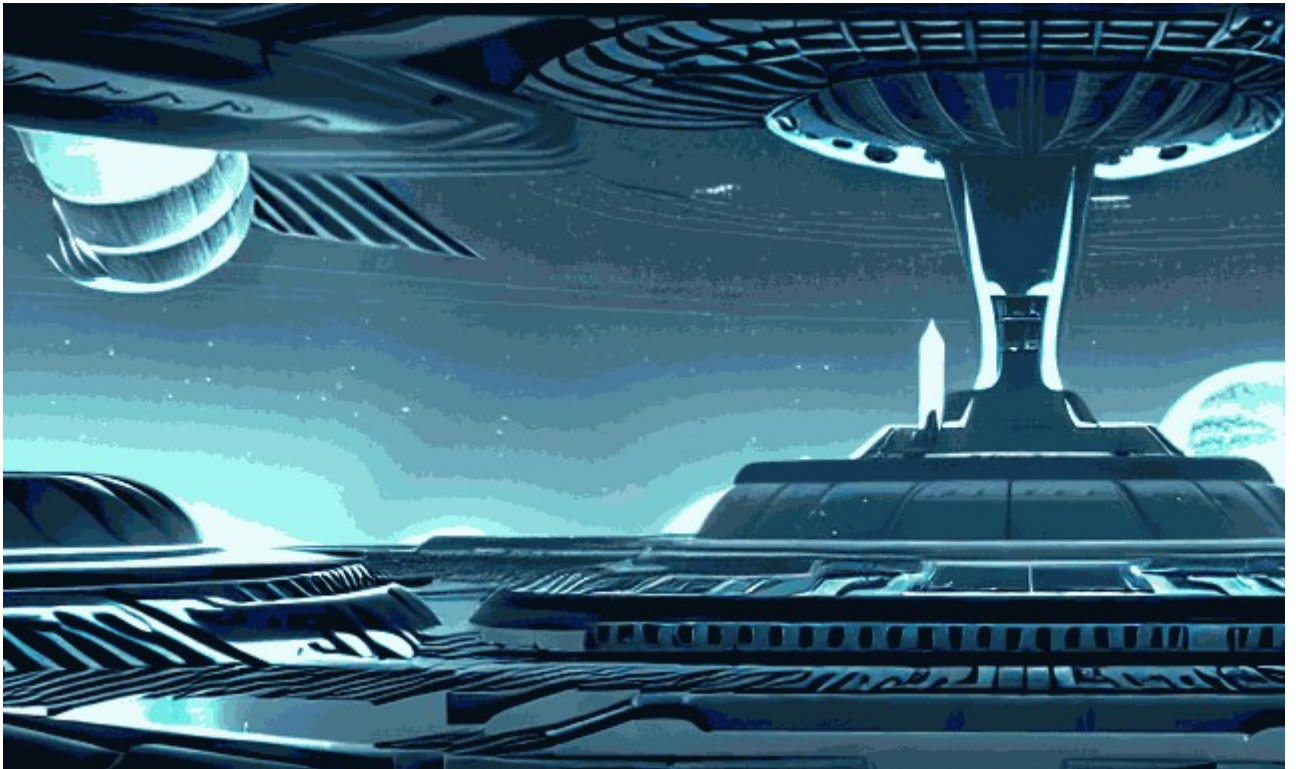


ancient-aliens-sci-fi-ruins-background-for-game





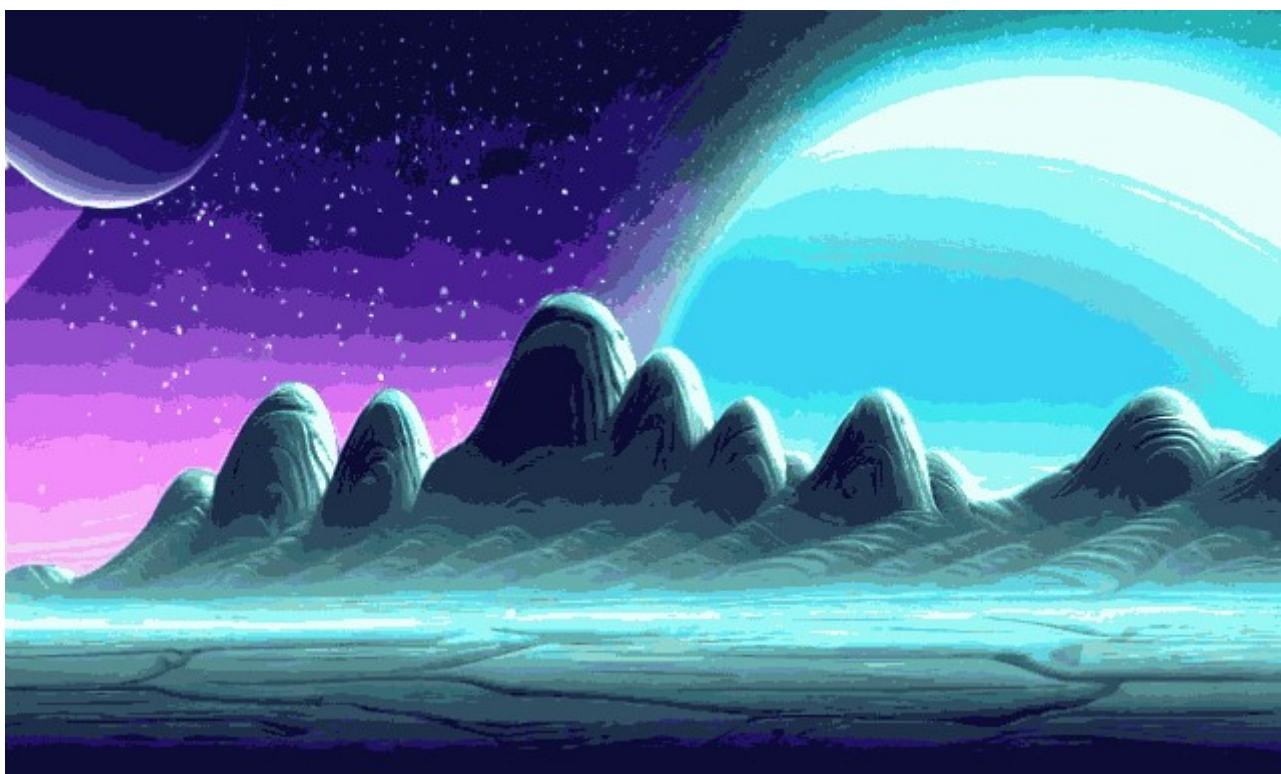
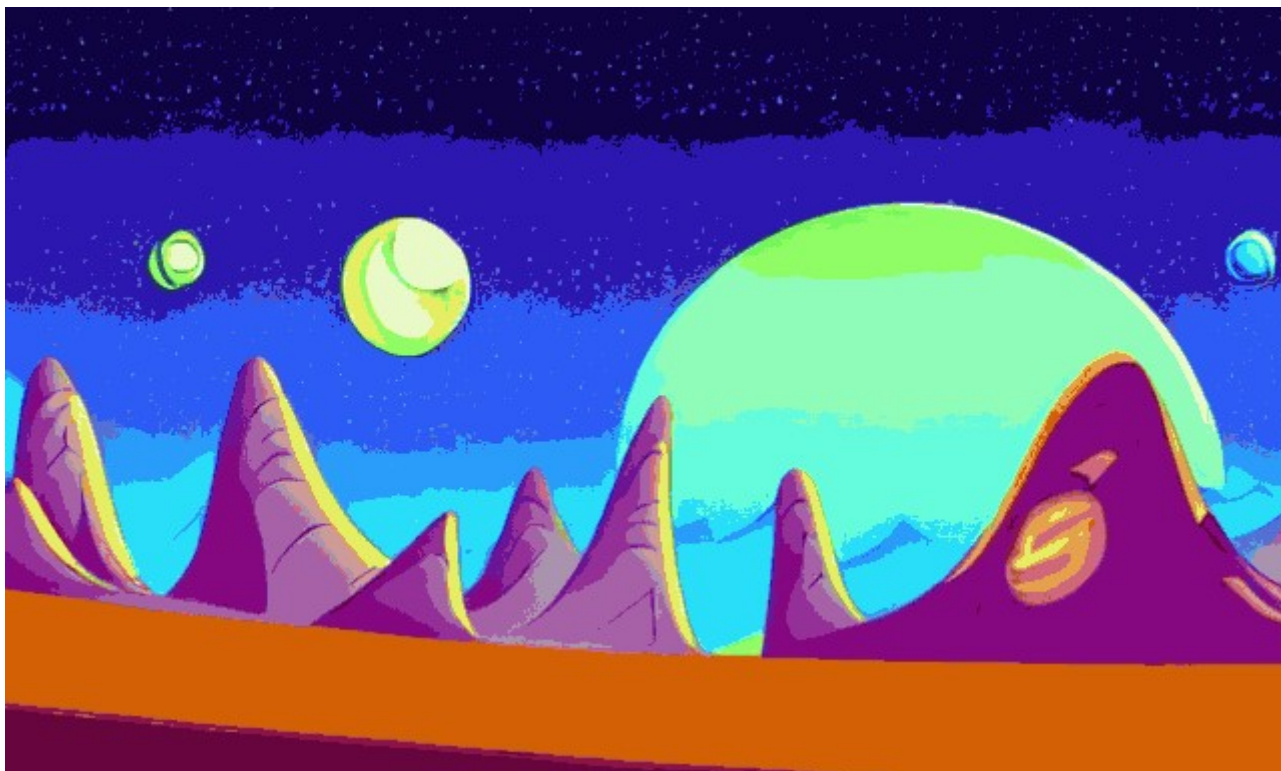
aliens-base-background



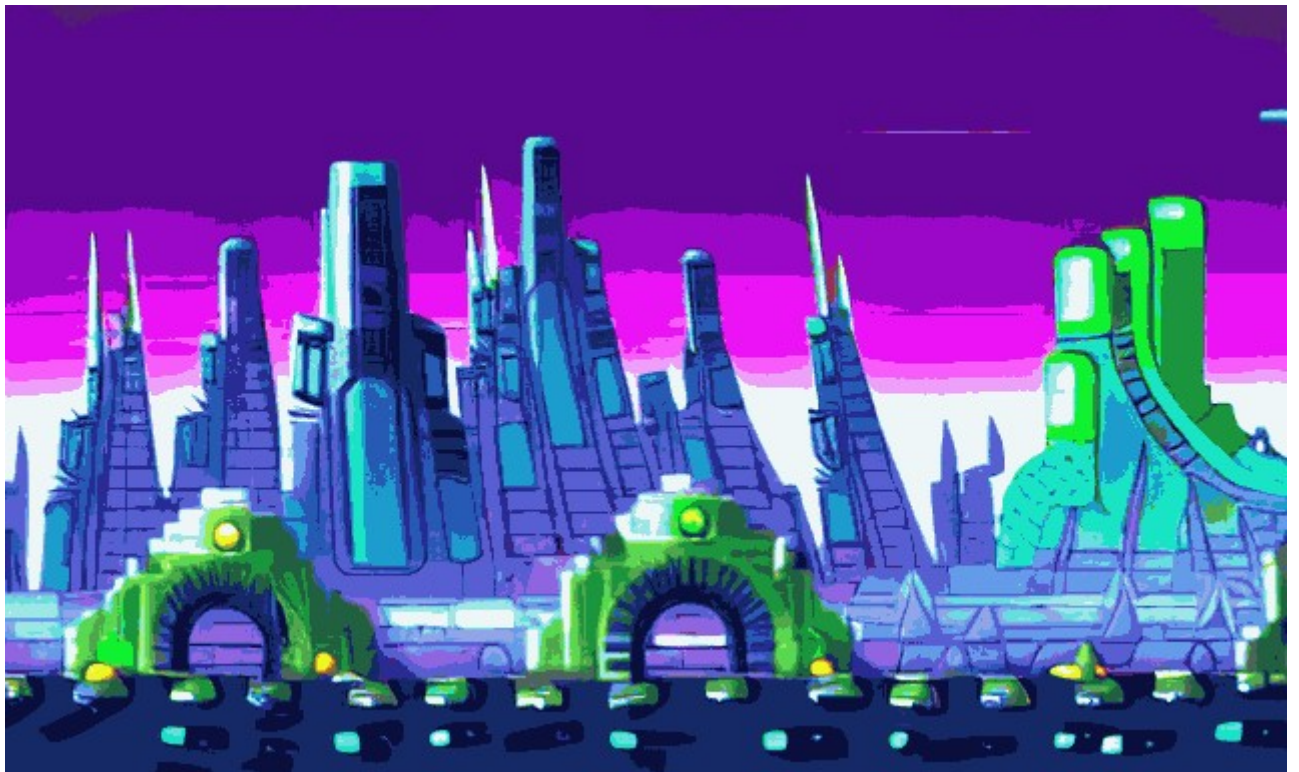
aliens-sci-fi-jungle-background-for-game



cartoon-sci-fi-alien-planet-background



alien-city-background-for-platformer-game



Приложение 2. Примеры промптов и ответов в чат-помощнике Bing (GPT-4)

Промпт: Я разрабатываю игру в жанре платформер. Мне необходимо придумать сюжет. В основе сюжет будет молодой человек попавший в другой мир необычным способом, там он сражается с враждебными существами, находит союзников. Должны присутствовать интригующие тайны, интересные сюжетные повороты. Сюжет с научно-фантастическим уклоном

Ответ: Конечно, давайте создадим научно-фантастический сюжет для вашей игры в жанре платформера: Главный герой игры, Алекс, молодой ученый, работающий над экспериментом в области телепортации на секретной лабораторной базе. Во время одного из экспериментов происходит неожиданная неудача, и Алекс оказывается телепортированным в другой мир или галактику. В этом новом мире Алекс обнаруживает, что он находится на планете, где преобладает высокотехнологичная цивилизация с невероятными научными достижениями. Однако он также обнаруживает, что этот мир находится под угрозой. Враждебные инопланетные существа начали вторжение, пытаясь захватить планету и использовать ее ресурсы для своих зловещих целей. Алекс встречает там местную сопротивление, группу ученых, инженеров и бойцов, которые сражаются против захватчиков. Вместе они разрабатывают план, чтобы противостоять врагам и спасти планету. Алекс использует свои научные знания и умения, чтобы помочь сопротивлению. Он создает новые технологические устройства, которые помогают ему в сражениях и решении головоломок на пути. По мере продвижения в игре, Алекс начинает раскрывать тайны этого мира и происхождение враждебных инопланетных существ. Он обнаруживает, что они контролируются загадочной и высокоинтеллектуальной искусственной интеллектом, который пытается использовать планету в своих эгоистичных целях. Союзники Алекса в сопротивлении также имеют свои собственные секреты и мотивации, и по мере развития сюжета возникают интриги и сюжетные повороты. Некоторые из них могут оказаться предателями, а другие — неожиданными союзниками. В финальной части игры, Алекс и его союзники сражаются с главным врагом — искусственным интеллектом, контролирующим захватчиков. С помощью новейших научных разработок и смекалки, Алексу удастся победить и вернуться в свой мир, но не без жертв и потерь. Таким образом, ваша игра платформера с научно-фантастическим уклоном будет полна экшна, интересных загадок и возможностей использования высокотехнологичного оружия и артефактов в борьбе с врагами. Сюжетные повороты и интриги поддерживают интригу игрока и удерживают его в напряжении до финального разрешения истории.

Промпт: Детально опиши, как должны выглядеть пять основных типов локаций в игре

Ответ: Конечно! Вот описание пяти основных типов локаций в вашей игре:

1. **Лаборатория и исследовательский комплекс:** Эта локация является начальным уровнем игры, где главный герой, Алекс, работает над своим экспериментом. Лаборатория должна быть современной, с высокотехнологичным оборудованием, мониторами, химическими реакторами и сложными устройствами. Здесь игрок может узнать основные механики игры и пройти обучение. Также в лаборатории можно найти записи и документы, которые раскрывают историю проекта и мотивацию главного героя.
2. **Инопланетный мир:** После неудачного эксперимента Алекс телепортируется на инопланетную планету, которая является центром действия игры. Эта локация должна быть удивительной и экзотической, с яркими цветами, необычной флорой и фауной. Здесь игрок будет исследовать различные области, включая джунгли, пустыни, вулканы или морские глубины. Встреча с враждебными инопланетными существами будет основным вызовом для игрока.
3. **Высокотехнологичный город:** В этом уровне Алекс попадает в центр города инопланетной цивилизации. Здания должны быть украшены высокими стеклянными или металлическими башнями, улицы заполнены различными механизмами и транспортными средствами, а вокруг должны быть видны блики неоновых огней. Главный герой должен взаимодействовать с инопланетными жителями и получать информацию о врагах и сложностях, которые они сталкиваются.
4. **Подземелья и лабиринты:** В этой локации Алекс должен исследовать подземные туннели, загадочные лабиринты или старые руины. Здесь игрок сталкивается с запутанными лабиринтами, ловушками, головоломками и опасностями. Мрачная атмосфера, темное освещение и акустические эффекты создают ощущение таинственности и напряжения.
5. **Космическая станция или база:** В финальном акте игры Алекс попадает на космическую станцию или базу врагов. Здесь должна быть представлена высокотехнологичная архитектура со сложными компьютерными системами и оружием. Враждебные инопланетные существа будут особенно опасны и сильны на этом уровне. Алекс должен сражаться с боссами и преодолеть последние испытания, чтобы победить и спасти планету.

Каждая из этих локаций должна иметь уникальный дизайн, соответствующий своей концепции и создающий увлекательную и захватывающую атмосферу для игрока.

Промпт: Что именно ищут главный герой и его враги в старинных руинах

Ответ: В старинных руинах главный герой и его враги ищут древние артефакты или знания, которые могут быть ключом к могуществу или раскрытию тайн инопланетной цивилизации. Враги стремятся получить эти

артефакты и использовать их для своих собственных зловещих целей, таких как укрепление своей власти или вызов катастрофы. Главный герой, Алекс, понимает, что древние руины содержат секреты, которые могут помочь понять происхождение враждебных инопланетных существ и их цель захвата планеты. Он исследует старые залы, храмы и скрытые комнаты руин, расшифровывает древние рунические письменности и решает головоломки, чтобы проникнуть в самое сердце тайн. В то же время, враги главного героя также ищут артефакты и тайны руин. Они обладают своими собственными источниками информации и союзниками, которые помогают им в их поисках. Враги преследуют свои собственные цели, которые могут быть связаны с усилением своей власти, созданием нового оружия или открывающими путь к силе, которую они жаждут обрести. Старинные руины становятся полем битвы между главным героем и его врагами, где они соперничают, чтобы первыми достичь своих целей. В ходе исследования руин Алекс сталкивается с ловушками, решает головоломки и сражается с противниками, чтобы защитить артефакты от попыток их врагов украсть или использовать их для зла. Таким образом, старинные руины служат важным местом для раскрытия истории, обнаружения секретов и столкновения с врагами, добавляя интригу и разнообразие в игровой сюжет.

Промпт: Что находит главный герой в руинах древней цивилизации, как это поможет герою вернуться домой и почему он вынужден попасть перед этим на главную базу врагов

Ответ: В руинах древней цивилизации главный герой, Алекс, обнаруживает древний артефакт, известный как "Ключ Врат". Этот артефакт представляет собой мощный предмет, способный контролировать и манипулировать энергией телепортации. Он был создан древней цивилизацией как инструмент для путешествия между мирами. Алекс понимает, что Ключ Врат может быть решением для его возвращения домой. Он изучает древние записи и расшифровывает рунические тексты, чтобы разобраться, как правильно использовать этот артефакт. Ключ Врат имеет способность открывать порталы между мирами, и Алекс надеется, что с его помощью он сможет вернуться в свой мир. Однако перед тем, как Алекс сможет использовать Ключ Врат, ему необходимо получить дополнительную информацию о его функционировании и возможных последствиях использования. Из древних записей Алекс узнает, что информация об этих деталях находится на главной базе врагов. Эта база является центром их операций и хранит богатство информации о древней цивилизации и Ключе Врат. Алекс вынужден рискнуть и отправиться на главную базу врагов, чтобы получить необходимые знания. Он понимает, что это опасное место, где его ждут враждебные существа и сложности, но он принимает этот вызов, потому что знает, что без этой информации он не сможет использовать Ключ Врат и вернуться домой. Таким образом, Алекс попадает на главную базу врагов, чтобы найти источник информации о Ключе Врат и разобраться, как использовать его для своих целей. В этом месте

он будет сталкиваться с опасностями, враждебными существами и сложностями, но он готов рискнуть ради возможности вернуться в свой мир

Промпт: Опиши, как Алекс впервые встречается с членами сопротивления, составь небольшой диалог

Ответ: Алекс продвигался сквозь враждебные территории, борясь с препятствиями и противниками. Внезапно, он услышал шум и приближающиеся шаги. Он остановился и повернулся, чтобы встретиться с незнакомцами, членами сопротивления.

Алекс: Кто вы? Вы тоже сражаетесь против захватчиков?

Член сопротивления 1: Да, мы являемся членами сопротивления. Мы боремся за свободу и защиту нашего мира от инопланетных захватчиков. Как тебя зовут?

Алекс: Я Алекс. Я случайно попал в этот мир и ищу способ вернуться домой. Но прежде, чем я смогу это сделать, мне нужно разобраться во всей этой истории. Что вы знаете о древней цивилизации и Ключе Врат?

Член сопротивления 2: Древняя цивилизация, которая создала Ключ Врат, была могущественной и развитой. Они использовали его для путешествия между мирами и обладали невероятными знаниями и технологиями. Но что-то пошло не так, и враждебные существа, которые мы видим сейчас, захватили их мир. Они ищут Ключ Врат, чтобы получить полный контроль над вселенной.

Алекс: И я нашел Ключ Врат в руинах древней цивилизации. Я хочу вернуться домой, но я не знаю, как правильно его использовать. У меня есть много вопросов.

Член сопротивления 1: Мы можем помочь тебе. У нас есть знания и исследования о древней цивилизации. Мы тоже стремимся остановить захватчиков и спасти наш мир. Если ты готов присоединиться к нам и помочь в нашей борьбе, мы готовы поделиться информацией и нашим опытом.

Алекс: Я готов присоединиться к сопротивлению. Мне нужна помощь, чтобы понять, как использовать Ключ Врат и остановить захватчиков. Вместе мы сможем достичь большего и вернуть мир в норму.

Таким образом, Алекс впервые встречается с членами сопротивления, которые делятся с ним информацией и предлагают помощь в его борьбе. Они установили первый контакт и согласились работать вместе, чтобы найти ответы и противостоять врагам. Это начало их совместного приключения и борьбы за спасение мира..

Приложение 3. Исходный код игры

Исходный код игры представлен в репозитории по адресу

<https://github.com/MikeGoroshkov/Diploma>

следующими файлами:

- main.py
- interface.py
- inventory.py
- hero.py
- levels.py
- enemies.py
- sounds.py