Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

**на итоговый проект «Разработка**

**платформера «Quantum Vigilance»» по ДПП ПП «Основы Gamedev и VR-разработки»**

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Провести исследование и анализ современных платформеров | Акименко Д. И. | 02.05.2025 – 13.05.2025 |
| 2 | Разработать собственную концепию платформера, основываясь анализе | 14.05.2025 – 15.05.2025 |
| 3 | Разработать опытный образец платформера | 15.05.2025 – 30.05.2025 |
| 4 | Сделать выводы исходя из результатов работы | 30.05.2025 – 3.06.2025 |

Руководитель проекта   
преподаватель Осыкин Д.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / «\_\_\_ » \_\_\_\_\_\_\_\_\_ 2025 г.

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

канд. физ.-мат. наук, доцент Козлов Д.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г.

(подпись)

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

Отчет о выполнении группового итогового проекта по ДПП ПП

«Основы Gamedev и VR-разработки»

**«Разработка платформера “Escape”»**

Исполнители:

Акименко Д. И.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта

преподаватель Осыкин Д. А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

# Цель проекта

**Quantum Vigilance** — это короткая иммерсивная перцептивная головоломка, в которой герой-дрон VIGIL-9 исследует кристаллические пещеры Nexus Prime и использует уникальные механики «Phantom Dash» и «Wall Cling» для стабилизации нестабильного квантового реактора. Главная задача игры — донести до игрока ощущение «реальности, материализующейся под взглядом» и предоставить насыщенный, но компактный опыт, сочетающий физику, динамику и экспериментальное мышление.

Конкретные цели:

• Продемонстрировать ключевые игровые механики в ограниченном по времени формате.

• Создать законченный прототип зоны (3–4 сцены), позволяющий пройти от начала до конца за 15 минут.

• Подготовить презентационный материал (видео + слайды) для инвесторов и демонстраций.

# Задачи проекта и исполнители

| Задача | Ответственный |  |
| --- | --- | --- |
| **Дизайн механик**: проработать «Phantom Dash» и «Wall Cling» | Lead Game Designer |  |
| **Прототип уровня**: каркас сцен с placeholder-тайлами | Level Designer / Unity Scripter |  |
| **Программная реализация**: |  |  |
| • Физическая модель игрока и коллизий |  |  |
| • Камера и HUD |  |  |
| • UI-экраны (меню, Messagebox) | Lead Programmer |  |
| **Арты и спрайты (placeholder → финал)** | 2D Artist |  |
| **Звук и музыка**: |  |  |
| • Основной саундтрек |  |  |
| • Звуковые эффекты |  |  |
| • Голосовое сопровождение вступления | Sound Designer |  |
| **Видео-презентация**: запись геймплея, монтаж, озвучка | Video Producer |  |
| **Тестирование и полировка** | QA Engineer |  |

# Актуальность и востребованность проекта

1. **Рынок инди-головоломок**
   * В 2024–2025 гг. наблюдается устойчивый рост интереса к коротким, но глубоким играм формата «bite-sized experience» (по данным IndieDB и Steam Next-fest).
   * Игры длительностью 10–20 мин часто попадают в подборки «Лучшие короткие инди-проект» и получают высокую пользовательскую оценку за уникальные механики.
2. **Уникальность механик**
   * **Phantom Dash** и **Wall Cling** предлагают новое чувство взаимодействия с окружающей средой, вписываясь в тренд «игр-экспериментальных интуитивных взаимодействий» (пример: *Observation*, *Twelve Minutes*).
   * «Материализация под взглядом» резонирует с популярной темой «attention-driven gameplay» (игры, реагирующие на фокус, как в VR-проекте *Keep Talking and Nobody Explodes*).
3. **Целевые аудитории**
   * **Инди-геймеры** (возраст 18–35 лет), ценящие новизну механик.
   * **Куч-платформеры и puzzle-проекты**: сообщество, ищущее креативные, но недлинные игры.
   * **Конференции и фестивали**: жанр подходит для демонстрации на Steam Next-fest, Gamescom Indie Arena и др.
4. **Бизнес-преимущества**
   * Низкие затраты на разработку (простые тайлы-плейсхолдеры, короткий цикл).
   * Быстрый выход на рынок и оперативная обратная связь.
   * Возможность дальнейшего расширения (новые уровни, мультиплеерный режим «соревнование на время»).

# Общие сведения о проделанной работе

### 4.1 Этапы выполнения проекта

1. **Анализ и концептуализация**
   * Сбор референсов по перцептивным головоломкам и играм про роботов.
   * Формулировка ключевой идеи: «Quantum Vigilance» — дрон-ремонтник в квантовых пещерах.
2. **Проектирование архитектуры**
   * Выделение модулей:
     + Main.py (основная логика, цикл, меню),
     + entity.py (физика и поведение игрока),
     + tilemap.py (тайловый движок и загрузка карт),
     + functional\_blocks.py (интерактивные объекты уровня),
     + utils.py (загрузка ресурсов, UI-элементы, анимации),
     + config.py (централизованные константы).
3. **Настройка окружения и прототипирование**
   * Язык Python 3.12, библиотека Pygame.
   * Среда разработки: VS Code с расширением для Pygame и отладкой.
   * Система контроля версий: Git (GitHub).
4. **Реализация ключевых механик**
   * **Phantom Dash**: фазовый рывок сквозь тонкие преграды.
   * **Wall Cling**: цепляние за стены и wall-jump.
   * Интеграция этих механик в класс Player (наследник PhysicsEntity).
5. **Создание тайлового движка и редактора**
   * Тайловая сетка: хранение в dict ключ→{type,variant,pos}.
   * Система коллизий через physics\_tiles\_around().
   * Вспомогательный редактор (editor.py), позволяющий быстро править JSON-карты и offgrid-тайлы.
6. **Разработка UI и HUD**
   * Классы Textbox и Messagebox — отрисовка меню и диалогов с анимацией печати.
   * HUD: единая полоска стамины, таймер в минутах в углу экрана, динамическая подсветка.
7. **Рефакторинг и конфигурация**
   * Вынесение всех «магических чисел», путей и цветов в config.py для быстрой балансировки.
   * Замена жёстких значений на константы: размеры экрана, шрифты, цвета, пороги переключения камеры.
8. **Тестирование и отладка**
   * Локальный play-test прототипа для проверки механик.
   * Добавление консольного вывода позиции игрока для точной настройки камерных порогов.
   * Исправление критических ошибок (несоответствие списка/кортежа при сдвиге камеры).

### 4.2 Выбранные инструменты и обоснование

| Инструмент | Причина выбора |
| --- | --- |
| **Python 3.12 + Pygame** | Быстрое прототипирование 2D-игр, простая работа с графикой и событиями. |
| **VS Code** | Люди знакомы с IDE, есть поддержка подсветки, линтинга и отладки. |
| **Git/GitHub** | Контроль версий, совместная работа и резервное хранение кода. |
| **JSON** | Удобный и читаемый формат для карт; легко редактируется вручную и через editor.py. |
| **PNG/ASEprite** | Стандарт для спрайтов; ASEprite упрощает создание анимаций. |
| **config.py** | Централизованная настройка проекта: цвета, размеры, пути, физические константы. |
| **Custom Animation class** | Простейшая система кадровых анимаций без внешних зависимостей. |

**Архитектурные решения**:

* **OOP-модель** (PhysicsEntity → Player, functional\_block → Jumper\_block/Exit\_block) улучшает читаемость и расширяемость.
* **Модульность**: разделение ответственности между модулями позволяет параллельно дорабатывать UI, механику и карты.
* **Рефакторинг** в config.py ускорил балансировку и снизил риск ошибок «магических чисел».

# Результаты проекта

Результатом проекта стал функциональный прототип игры, который наглядно демонстрирует все задуманные ключевые особенности: мрачную атмосферу, геймплейные механики, звуковое оформление и нестандартный подход к повествованию. Этот прототип позволяет оценить как сильные стороны концепта, такие как инновационное управление через дрон и напряжённый игровой процесс, так и выявить слабые места — баланс сложности, удобство управления или необходимость дополнительных визуальных подсказок.

1. При этом прототип построен с расчётом на масштабирование — его архитектура позволяет легко добавлять новые уровни, механики и контент, что делает его готовым к дальнейшей доработке и полноценному релизу. Таким образом, проект уже сейчас представляет собой устойчивую основу для будущей игры, сохраняя потенциал для **Рабочий прототип**
   * Полноценный цикл из нескольких сцен, проходимый за 15 минут.
   * Две базовые механики реализованы и протестированы: Phantom Dash и Wall Cling.
2. **Инструменты разработки**
   * Встроенный редактор уровней (editor.py) позволяет визуально добавлять/удалять тайлы и функциональные объекты, сохранять JSON.
3. **Гибкая система камеры**
   * Три режима: only\_x, only\_y, free с возможностью динамического переключения через код (без триггеров в карте).
4. **Централизованная конфигурация**
   * config.py содержит все параметры игры: размеры, шрифты, цвета, физические константы, пути к ресурсам.
5. **Презентационный пакет**
   * Слайды с общей концепцией, описанием механик и фрагментами кода.
   * Видео-демо прохождения прототипа, записанное в OBS и смонтированное с аннотациями.
6. **Качество кода и документация**
   * Лаконичный и читаемый код (PEP 8, модульная структура).
   * Инлайновые комментарии и конфиг-файл облегчают дальнейшее расширение проекта.



Рисунок 1.

**Приложение 1.**Программный код

import Scripts.entity as Entity

import Scripts.tilemap as TMap

import sys

import config

from Scripts.utils import Animation, Textbox, image\_load, images\_load, Textbox, Messagebox

import pygame

class Game:

    def \_\_init\_\_(self):

        pygame.init()

        pygame.display.set\_caption("Quantum Vigilance")

        self.game\_font = pygame.font.Font(config.FONT\_PATH , config.FONT\_SIZE)

        self.Screen = pygame.display.set\_mode((config.SCREEN\_WIDTH, config.SCREEN\_HEIGHT))

        self.display = pygame.Surface((config.DISPLAY\_WIDTH , config.DISPLAY\_HEIGHT))

        self.clock = pygame.time.Clock()

        self.timer = config.INITIAL\_TIMER;

        self.time=0;

        self.camera\_shift = [0,0]

        self.assets = {

            'decor': images\_load('tiles/decor'),

            'surface': images\_load('tiles/surface'),

            'glass': images\_load('tiles/glass'),

            'large\_decor': images\_load('tiles/large\_decor'),

            'player':image\_load('entities/player.png'),

            'Jumper': images\_load('functional\_blocks/jumper'),

            'Exit': images\_load('functional\_blocks/Exit'),

            'player/idle':Animation(images\_load("entities/player/idle"),duration=6),

            'player/run':Animation(images\_load("entities/player/run"),duration=4),

            'player/jump':Animation(images\_load("entities/player/jump"),duration=7)

        }

        self.movement = [0, 0]

        self.tilemap = TMap.Tilemap(self, config.TILE\_SIZE)

        self.player = Entity.Player(self,self.tilemap.player\_pos,(self.assets['player'].get\_width(),self.assets['player'].get\_height()))

        self.tilemap.load("data/Maps/map1.json")

        # self.tilemap.camera\_info = 'only\_x'

        self.next\_level= False

        self.player.pos = self.tilemap.player\_pos

        self.level = 1;

        self.display.blit(image\_load("background.png"),(-50,-30))

    def load\_next\_level(self):

        for i in range(0,1800):

            self.Screen.fill(config.COLOR\_BG,(320-i,240-i,i,i))

            pygame.display.update()

        self.level+=1

        try:

            self.tilemap.load("data/Maps/map"+str(self.level)+".json")

            self.player.pos = self.tilemap.player\_pos

            trans\_text = f"[DEBUG] Vertical movement is activated."

            Messagebox(trans\_text, (20, 20)).render(self)

        except:

            EndMassage = Messagebox("[DEBUG] EOF",(20,20))

            EndMassage.render(self)

        self.next\_level =False

    def reload(self):

        Deathbox = Textbox((180,240),"[DEBUG] Fatal error",32,120,280)

        self.player.pos = self.tilemap.player\_pos

        for i in range(0,1800):

            self.Screen.fill(config.COLOR\_BG,(320-i,200-i,i,i))

            Deathbox.render(self.Screen,0)

            pygame.display.update()

        self.player.is\_death = False

        self.tilemap.load("data/Maps/map"+str(self.level)+".json")

    def menu(self):

        menu\_pos = 0

        self.display.blit(self.game\_font.render("Quantum Vigilance",False,config.COLOR\_TEXT\_PRIMARY),(10,10))

        start\_button = Textbox((20,70),"Start")

        # level\_button = Textbox((20,110),"Level\_editor")

        exit\_button = Textbox((20,150),"Exit")

        while True:

            menu\_pos = menu\_pos % 2

            menu\_elements = ["Start","Exit"]

            start\_button.render(self.display,False)

            # level\_button.render(self.display,False)

            exit\_button.render(self.display,False)

            match menu\_pos:

                case 0:

                    start\_button.render(self.display, True)

                # case 1:

                #    level\_button.render(self.display,True)

                case 1:

                    exit\_button .render(self.display, True)

            for event in pygame.event.get():

                match event.type:

                    case pygame.KEYDOWN:

                        match event.key:

                            case pygame.K\_DOWN:

                                menu\_pos+=1

                            case pygame.K\_UP:

                                menu\_pos-=1

                            case pygame.K\_SPACE:

                                match menu\_elements[menu\_pos]:

                                    case "Start":

                                        self.run()

                                    case "Exit":

                                        pygame.quit()

                                        sys.exit()

            self.Screen.blit(pygame.transform.scale(self.display,self.Screen.get\_size()),(0,0))

            pygame.display.update()

    def hud(self):

        if self.timer == 0:

            EndMessage = Messagebox("You fail mission and spaceship was completly breoken!!!",(20,20))

            self.player.is\_death = True

            self.level = 1

            self.timer = config.INITIAL\_TIMER

            EndMessage.render()

        self.time= (self.time+1)%config.TIMER\_CYCLE

        self.timer-=self.time//60

        # новый таймер — в минутах, справа сверху

        minutes = self.timer // 60

        timer\_str = f"Battery: {minutes} cells"

        txt\_surf = self.game\_font.render(timer\_str, False, (100, 200, 220))

        # позиция — правый верхний угол с отступом 10px

        tx = config.DISPLAY\_WIDTH - txt\_surf.get\_width() - 10

        ty = 10

        self.display.blit(txt\_surf, (tx, ty))

        # Стамина: отрисовка заполнения

        # Рисуем единый индикатор: фон + заливка по проценту

        x0, y0 = config.STAMINA\_BAR\_POS

        W, H  = config.STAMINA\_BAR\_WIDTH, config.STAMINA\_BAR\_HEIGHT

        pct   = max(0, min(1, self.player.stamina / config.MAX\_STAMINA))

        # фон

        pygame.draw.rect(self.display, (30, 30, 50), (x0, y0, W, H))

        # заливка — янтарный цвет под тёмную пещеру

        fill\_w = int(W \* pct)

        pygame.draw.rect(self.display, (220, 160,  40), (x0, y0, fill\_w, H))

        # граница

        pygame.draw.rect(self.display, (100,  60, 30), (x0, y0, W, H), 2)

        # подпись

        label = self.game\_font.render("", False, (200, 200, 180)) # Выносливость

        self.display.blit(label, (x0, y0 - H - 4))

        # Бордер полосы

        pygame.draw.rect(

            self.display,

            config.COLOR\_STAMINA\_BORDER,

            (

                config.STAMINA\_BAR\_POS[0],

                config.STAMINA\_BAR\_POS[1],

                config.STAMINA\_BAR\_WIDTH,

                config.STAMINA\_BAR\_HEIGHT

            ),

            1

        )

        # Лейбл «stamina:»

        self.display.blit(

            self.game\_font.render("stamina:", False, config.COLOR\_STAMINA\_BORDER),

            config.STAMINA\_BAR\_POS

        )

    def camera\_load(self):

        print(f"[DEBUG] level={self.level}, x={self.player.pos[0]:.1f}, thresh={config.LEVEL2\_CAMERA\_SWITCH\_X}")

        if self.level == 2 and self.player.pos[0] > config.LEVEL2\_CAMERA\_SWITCH\_X:

            print("[DEBUG] Switching to only\_x")

            self.tilemap.camera\_info = "only\_x"

        match self.tilemap.camera\_info:

            case "only\_x":

                # По X: центрируем на игроке

                self.camera\_shift[0] = int(self.player.pos[0]) - self.display.get\_width() // 2

                if self.camera\_shift[0] < 0:

                    self.camera\_shift[0] = 0

                # По Y: фиксируем для первой карты

                if self.level == 1:

                    self.camera\_shift[1] = config.FIRST\_MAP\_CAMERA\_Y

                # иначе оставляем предыдущий или нулевой

            case "only\_y":

                self.camera\_shift[1] = (int(self.player.pos[1])-self.display.get\_height()//2)

                if self.camera\_shift[1]>0:

                    self.camera\_shift[1]=0

            case "free":

                # Меняем элементы списка, не создаём кортеж

                self.camera\_shift[0] = int(self.player.pos[0]) - self.display.get\_width()//2

                self.camera\_shift[1] = int(self.player.pos[1]) - self.display.get\_height()//2

    def run(self):

        intro = Messagebox(

            "System boot sequence incomplete—quantum cores unstable. \n"

            "You awaken as VIGIL-9, an autonomous repair drone entombed within the crystalline caverns \n"

            "of Nexus Prime. Sensors detect cascading quantum anomalies as the reactor core spirals toward collapse. \n"

            "Only you can stabilize the core and illuminate these pitch-black depths. Act swiftly, VIGIL-9.\n",

            (20,20)

        )

        intro.render(self)

        intro = Messagebox("X - dash \nArrows keys - move \nEscape - menu" ,(20,20))

        intro.render(self)

        while True:

            if self.next\_level:

                self.load\_next\_level()

            if self.player.is\_death:

                self.reload()

            self.display.fill(config.COLOR\_BG)

            self.player.update(self.tilemap,self.movement)

            # Для отладки: выводим позицию игрока в консоль

            print(f"Player pos: x={self.player.pos[0]:.1f}, y={self.player.pos[1]:.1f}")

            self.tilemap.render(self.display,self.camera\_shift)

            self.player.render(self.display,self.camera\_shift)

            self.tilemap.fun\_render(self.display, self.camera\_shift)

            self.hud()

            self.camera\_load()

            for event in pygame.event.get():

                match event.type:

                    case pygame.QUIT:

                        pygame.quit()

                        sys.exit()

                    case pygame.KEYDOWN:

                        match event.key:

                            case pygame.K\_LEFT:

                                self.movement[0] =-1

                            case pygame.K\_RIGHT:

                                self.movement[0] = 1

                            case pygame.K\_SPACE:

                                self.player.jump()

                            case pygame.K\_x:

                                self.player.Phantom\_dash()

                            case pygame.K\_DOWN:

                                self.movement[1] = 1

                            case pygame.K\_r:

                                self.reload()

                            case pygame.K\_ESCAPE:

                                self.menu()

                    case pygame.KEYUP:

                            match event.key:

                                case pygame.K\_LEFT:

                                    self.movement[0] = 0

                                case pygame.K\_RIGHT:

                                    self.movement[0] = 0

            fps = int(self.clock.get\_fps())

            fps\_text = self.game\_font.render(f"FPS: {fps}", False, config.COLOR\_TEXT\_PRIMARY)

            self.display.blit(fps\_text, config.FPS\_TEXT\_POS)

            self.clock.tick(config.FPS)

            self.Screen.blit(pygame.transform.scale(self.display,self.Screen.get\_size()),(0,0))

            pygame.display.update()

Game().menu()

# config.py

# Окно и рендер

SCREEN\_WIDTH = 640

SCREEN\_HEIGHT = 480

DISPLAY\_WIDTH = 320

DISPLAY\_HEIGHT = 240

RENDER\_SCALE = SCREEN\_WIDTH / DISPLAY\_WIDTH  # обычно 2.0

# Тайловая сетка

TILE\_SIZE = 16

# Шрифты

FONT\_PATH = "data/fonts/PublicPixel-rv0pA.ttf"

FONT\_SIZE = 8

# Таймер и FPS

FPS = 60

INITIAL\_TIMER = 15\*60

TIMER\_CYCLE = 61

FPS\_TEXT\_POS = (2, 2)

# Физика

GRAVITY = 0.1

# Механики игрока

MAX\_STAMINA = 360

STAMINA\_BAR\_WIDTH = 100

STAMINA\_BAR\_HEIGHT = 10

STAMINA\_BAR\_POS = (15, 10)

TIMER\_TEXT\_POS = (15, 35)

TIMER\_TEXT\_WIDTH = 150

DASH\_SPEED = 8          # скорость во время даш-движения

DASH\_POWER = 10         # начальная сила даш

DASH\_STAMINA\_COST = 360

JUMP\_VELOCITY = -3

WALL\_JUMP\_VELOCITY\_X = 2

WALL\_JUMP\_STAMINA\_COST = 60

WALL\_GRAB\_THRESHOLD = 4  # air\_time > 4

ANIM\_OFFSET = (-3, -3)   # смещение спрайта при рендере

# Цвета

COLOR\_BG = (10, 12, 25)

COLOR\_TEXT\_PRIMARY = (255, 142, 128)

COLOR\_TEXT\_SECONDARY = (74, 36, 128)

COLOR\_STAMINA\_FILL = (197, 58, 157)

COLOR\_STAMINA\_BORDER = (74, 36, 128)

COLOR\_UI\_TEXT    = (200, 200, 180)  # бежево-серый

COLOR\_STAMINA\_BG = (30,  30, 50)    # чуть светлее фона

COLOR\_STAMINA\_FILL = (220, 160, 40) # янтарный

COLOR\_STAMINA\_BORDER = (100, 60, 30)

COLOR\_TIMER\_TEXT = (100, 200, 220)  # холодный циан

COLOR\_MESSAGEBOX\_BG     = COLOR\_BG           # фон сообщения

COLOR\_MESSAGEBOX\_BORDER = COLOR\_TEXT\_PRIMARY # рамка и текст

# Textbox

TEXTBOX\_DEFAULT\_HEIGHT = 30

TEXTBOX\_DEFAULT\_WIDTH = 100

TEXTBOX\_PADDING = 10

# Messagebox

MESSAGEBOX\_SIZE = (300, 220)

MESSAGEBOX\_CHAR\_WIDTH = 8

MESSAGEBOX\_LINE\_WRAP = 270

MESSAGEBOX\_LINE\_HEIGHT = 12

# Tilemap

NEIGHBOURS = [(-1, 0), (-1, -1), (0, -1), (1, -1), (1, 0),

              (0, 0), (-1, 1), (0, 1), (1, 1)]

PHYSICS\_TILES = {"surface", "Jumper"}

# Редактор

EDITOR\_RENDER\_SCALE = 2.0

SCROLL\_SPEED = 5

GAZE\_RECT\_SIZE = (20, 20)        # размер зоны взгляда

GAZE\_DEBUG\_COLOR = (200, 200, 200)

FIRST\_MAP\_CAMERA\_Y = -250   # пример: сместить камеру на 50 пикселей вверх

LEVEL2\_CAMERA\_SWITCH\_X = 540

import pygame

import config

class PhysicsEntity:

    def \_\_init\_\_(self, game, e\_type, pos, size):

        self.game = game

        self.type = e\_type

        self.pos = list(pos)

        self.size = size

        self.velocity = [0, 0]

        self.collisions = {"up": False,"down": False,"left": False,"right": False}

        self.flip = False

        self.action = ""

        self.anim\_offset = config.ANIM\_OFFSET

        self.set\_action("idle")

    def set\_action(self,action):

        if action != self.action:

            self.action = action

            self.animation = self.game.assets[self.type+'/'+self.action].copy()

    def rect(self):

        return pygame.Rect(self.pos[0],self.pos[1],self.size[0],self.size[1])

    def update(self,tilemap, movement=(0,0), gravity=config.GRAVITY,dash = False):

        self.collisions = {"up": False,"down": False,"left": False,"right": False}

        frame\_movement = (movement[0] + self.velocity[0], movement[1] + self.velocity[1])

        self.pos[0]+=frame\_movement[0]

        entity\_rect = self.rect()

        if not dash:

            for rect in tilemap.physics\_tiles\_around(self.pos):

                if entity\_rect.colliderect(rect):

                    if frame\_movement[0] > 0:

                        entity\_rect.right = rect.left

                        self.collisions["right"] = True

                    if frame\_movement[0] < 0:

                        entity\_rect.left  = rect.right

                        self.collisions["left"] = True

                    self.pos[0] = entity\_rect.x

        if not dash:

            self.pos[1]+=frame\_movement[1]

        entity\_rect = self.rect()

        if not dash:

            for rect in tilemap.physics\_tiles\_around(self.pos):

                if entity\_rect.colliderect(rect):

                    if frame\_movement[1] > 0:

                        entity\_rect.bottom  = rect.top

                        self.collisions["down"] = True

                    if frame\_movement[1] < 0:

                        entity\_rect.top = rect.bottom

                        self.collisions["up"] = True

                    self.pos[1] =entity\_rect.y

        self.velocity[1] = min(5,self.velocity[1]+gravity)

        if self.velocity[0]<0:

            self.velocity[0] = min(0,self.velocity[0]+gravity)

        else:

            self.velocity[0] = max(0,self.velocity[0]-gravity)

        if movement[0]>0:

            self.flip = False

        if movement[0]<0:

            self.flip = True

        if self.collisions["up"] or self.collisions["down"]:

            self.velocity[1] = 0

        self.animation.update()

    def render(self,display,shift = (0,0)):

        display.blit(pygame.transform.flip(self.animation.img(), self.flip, False),(self.pos[0]-shift[0]+self.anim\_offset[0],self.pos[1]-shift[1]+self.anim\_offset[1]))

class Player(PhysicsEntity):

    def \_\_init\_\_(self,game,pos,size):

        super().\_\_init\_\_(game,'player',pos,size)

        self.air\_time = 0

        self.stamina = config.DASH\_STAMINA\_COST

        self.wall\_grap = True

        self.dashing = 0

        self.dash = False

        self.is\_death = False

    def update(self,tilemap,movement=(0,0)):

        super().update(tilemap, movement=movement,dash=self.dash)

        self.air\_time+=1

        if self.wall\_grap:

            self.stamina-=1

        if self.collisions['down']:

            self.air\_time = 0

            self.wall\_grap = False

            self.stamina = config.DASH\_STAMINA\_COST

        if self.air\_time>config.WALL\_GRAB\_THRESHOLD:

            self.set\_action('jump')

        if self.collisions["right"] or self.collisions["left"] and self.air\_time>config.WALL\_GRAB\_THRESHOLD:

            self.wall\_grap =False

            if self.stamina >0:

                self.velocity[1] = 0

                self.wall\_grap = True

        if self.dashing>0:

            self.dashing = max(0,self.dashing-1)

        if self.dashing<0:

            self.dashing = min(0,self.dashing+1)

        if self.dashing!=0:

            self.velocity[0] = abs(self.dashing)/self.dashing\*config.DASH\_SPEED

            if abs(self.dashing) == 1:

                self.velocity[0]\*=0.01

                self.dash = False

                entity\_rect = self.rect()

                for rect in tilemap.physics\_tiles\_around(self.pos):

                    if entity\_rect.colliderect(rect):

                        self.is\_death = True

        elif movement[0]!=0:

            self.set\_action('run')

        else:

            self.set\_action('idle')

    def jump(self):

        if self.wall\_grap == True:

            if self.collisions["right"]:

                self.stamina-=config.WALL\_JUMP\_STAMINA\_COST

                self.velocity[1] = config.JUMP\_VELOCITY

                self.velocity[0] = -config.WALL\_JUMP\_VELOCITY\_X

            if self.collisions["left"]:

                self.stamina-=config.WALL\_JUMP\_STAMINA\_COST

                self.velocity[1] = config.JUMP\_VELOCITY

                self.velocity[0] = config.WALL\_JUMP\_VELOCITY\_X

        else:

            if self.air\_time<5:

                self.velocity[1] = config.JUMP\_VELOCITY

    def Phantom\_dash(self):

        if self.dashing==0 and self.stamina>0:

            self.stamina-=config.DASH\_STAMINA\_COST

            if self.flip:

                self.dashing = -config.DASH\_POWER

            else:

                self.dashing = config.DASH\_POWER

            self.dash =True

from Scripts.utils import Animation, images\_load

import pygame

import config

class functional\_block:

    def \_\_init\_\_(self,type,pos,game,tile\_size=16):

        self.pos = (pos[0]\*tile\_size,pos[1]\*tile\_size)

        self.type = type

        self.tile\_size = tile\_size

        self.game = game

        self.player = self.game.player

        self.animation =  Animation(self.game.assets[self.type].copy())

        self.active = False

    def rect(self):

        return pygame.Rect(self.pos[0],self.pos[1]-2,self.animation.img().get\_width(),self.animation.img().get\_height())

    def render(self):

        self.game.display.blit(self.animation.img(),(self.pos[0]-self.game.camera\_shift[0],self.pos[1]-self.game.camera\_shift[1]))

        if self.active:

            self.game.display.blit(self.animation.img(),(self.pos[0]-self.game.camera\_shift[0],self.pos[1]-self.game.camera\_shift[1]))

            self.animation.update()

class Jumper\_block(functional\_block):

    def \_\_init\_\_(self,pos,game):

        super().\_\_init\_\_("Jumper",pos,game)

    def update(self):

        if self.player.collisions["down"] and self.rect().colliderect(self.player.rect()):

            self.player.velocity[1]= -4.5

            self.active = True

class Exit\_block(functional\_block):

    def \_\_init\_\_(self,pos,game):

        super().\_\_init\_\_("Exit",pos,game)

    def update(self):

        if self.rect().colliderect(self.player.rect()):

            self.game.next\_level = True

    def render(self):

        self.game.display.blit(self.animation.img(),(self.pos[0]-self.game.camera\_shift[0],self.pos[1]-self.game.camera\_shift[1]))

        self.animation.update()

"""

class GazePlatform(functional\_block):

    def \_\_init\_\_(self, pos, game):

        super().\_\_init\_\_("GazePlatform", pos, game)

    def update(self, gaze\_rect):

        # глобальная позиция на экране

        screen\_x = self.pos[0] - self.game.camera\_shift[0]

        screen\_y = self.pos[1] - self.game.camera\_shift[1]

        tile\_rect = pygame.Rect(

            screen\_x, screen\_y,

            config.TILE\_SIZE, config.TILE\_SIZE

        )

        # активируется, только если фокус пересекается с тайлом

        self.active = gaze\_rect.colliderect(tile\_rect)

"""

import json

import pygame

import config

from Scripts.functional\_blocks import functional\_block,Jumper\_block,Exit\_block #,GazePlatform

NEIGBOURS = config.NEIGHBOURS

PHYSICS\_TILES = config.PHYSICS\_TILES

FUNC\_BLOCKS = {"jumper"}

class Tilemap:

    def \_\_init\_\_(self,game,tsize=16):

        self.game = game

        self.tile\_size = tsize;

        self.tilemap = {}

        self.functional\_blocks = []

        self.offgrid\_tiles = []

        self.camera\_info = "only\_x"

        self.player\_pos = (50,50)

    def tiles\_around(self,pos):

        tiles = []

        tile\_loc = (int(pos[0]/self.tile\_size),int(pos[1]/self.tile\_size))

        for ne\_tile in NEIGBOURS:

            check\_loc = str(tile\_loc[0]+ne\_tile[0])+";"+str(tile\_loc[1]+ne\_tile[1])

            if check\_loc in self.tilemap:

                tiles.append(self.tilemap[check\_loc])

        return tiles

    def physics\_tiles\_around(self,pos):

        rects = []

        for tile in self.tiles\_around(pos):

            if tile["type"] in PHYSICS\_TILES:

                rects.append(pygame.Rect(tile["pos"][0]\*self.tile\_size,tile["pos"][1]\*self.tile\_size,self.tile\_size,self.tile\_size))

        return rects

    def save(self,path,camera\_info,player\_pos):

        f = open(path,"w")

        json.dump({"tilemap": self.tilemap,"tile\_size": self.tile\_size, "offgrid": self.offgrid\_tiles, "camera": camera\_info,"player":player\_pos},f)

        f.close()

    def load(self,path):

        self.functional\_blocks.clear()

        f = open(path,"r")

        map\_data = json.load(f)

        f.close()

        self.camera\_info = map\_data["camera"]

        self.player\_pos = map\_data["player"]

        self.tilemap = map\_data["tilemap"]

        self.tile\_size = map\_data["tile\_size"]

        self.offgrid\_tiles = map\_data["offgrid"]

        for tile in self.tilemap:

            match self.tilemap[tile]["type"]:

                case "Jumper":

                    self.functional\_blocks.append(Jumper\_block(self.tilemap[tile]["pos"],self.game))

                case "Exit":

                    self.functional\_blocks.append(Exit\_block(self.tilemap[tile]["pos"],self.game))

                #case "GazePlatform":

                #    self.functional\_blocks.append(GazePlatform(self.tilemap[tile]["pos"], self.game))

    def eload(self,path):

        f = open(path,"r")

        map\_data = json.load(f)

        f.close()

        if "camera" in map\_data:

            self.camera\_info = map\_data["camera"]

        self.tilemap = map\_data["tilemap"]

        self.tile\_size = map\_data["tile\_size"]

        self.offgrid\_tiles = map\_data["offgrid"]

    def render(self,surf,shift=(0,0)):

        for offtile in self.offgrid\_tiles:

            surf.blit(self.game.assets[offtile["type"]][offtile["variant"]],(offtile["pos"][0]-shift[0],offtile["pos"][1]-shift[1]))

        for x in range(shift[0] // self.tile\_size, (shift[0]+surf.get\_width())//self.tile\_size+1):

            for y in range(shift[1] // self.tile\_size, (shift[1]+surf.get\_height())//self.tile\_size+1):

                loc = str(x)+";"+str(y)

                if loc in self.tilemap:

                    tile = self.tilemap[loc]

                    surf.blit(self.game.assets[tile['type']][tile['variant']],(tile['pos'][0]\*self.tile\_size- shift[0],tile['pos'][1]\*self.tile\_size- shift[1]))

    def fun\_render(self, surf, shift):

        for fb in self.functional\_blocks:

            # если это GazePlatform, передаём gaze\_rect

            #if isinstance(fb, GazePlatform):

            #    fb.update(gaze\_rect)

            #else:

            fb.update()

            fb.render()

import pygame

import os

import config

BASE\_IMG\_PATH='data/images/'

def image\_load(path):

    img = pygame.image.load(BASE\_IMG\_PATH+path)

    img.set\_colorkey((0,0,0))

    return img

def images\_load(path):

    images = []

    for img\_name in sorted(os.listdir(BASE\_IMG\_PATH+path)):

        images.append(image\_load(path+"/"+img\_name))

    return images

class Textbox:

    def \_\_init\_\_(self,pos,text,text\_size=config.FONT\_SIZE,height=config.TEXTBOX\_DEFAULT\_HEIGHT,widht=config.TEXTBOX\_DEFAULT\_WIDTH):

        self.font = pygame.font.Font(config.FONT\_PATH,text\_size)

        self.text = self.font.render(text,False,(255,142,128))

        self.active\_text = self.font.render(text,False,(74,36,128))

        self.textbox\_pos = list(pos)

        self.size = (height,widht)

    def render(self,screen,active):

        if active:

            pygame.draw.rect(screen,(255,142,128),(self.textbox\_pos[0]-10,self.textbox\_pos[1]-10,self.size[1],self.size[0]))

            pygame.draw.rect(screen,(74,36,128),(self.textbox\_pos[0]-10,self.textbox\_pos[1]-10,self.size[1],self.size[0]),3)

            screen.blit(self.active\_text,self.textbox\_pos)

        else:

            pygame.draw.rect(screen,(5,31,57),(self.textbox\_pos[0]-10,self.textbox\_pos[1]-10,self.size[1],self.size[0]))

            pygame.draw.rect(screen,(255,142,128),(self.textbox\_pos[0]-10,self.textbox\_pos[1]-10,self.size[1],self.size[0]),3)

            screen.blit(self.text,self.textbox\_pos)

class Messagebox():

    def \_\_init\_\_(self, text, pos):

        self.text = text

        self.size = config.MESSAGEBOX\_SIZE

        self.textbox\_pos = pos

        self.font = pygame.font.Font(config.FONT\_PATH, config.FONT\_SIZE)

    def render(self, game):

        # Фон

        pygame.draw.rect(

            game.display,

            config.COLOR\_MESSAGEBOX\_BG,

            (

                self.textbox\_pos[0] - config.TEXTBOX\_PADDING,

                self.textbox\_pos[1] - config.TEXTBOX\_PADDING,

                self.size[0],

                self.size[1]

            )

        )

        # Рамка

        pygame.draw.rect(

            game.display,

            config.COLOR\_MESSAGEBOX\_BORDER,

            (

                self.textbox\_pos[0] - config.TEXTBOX\_PADDING,

                self.textbox\_pos[1] - config.TEXTBOX\_PADDING,

                self.size[0],

                self.size[1]

            ),

            3

        )

        x\_shift = 0

        y\_shift = 0

        for char in self.text:

            x\_shift += config.MESSAGEBOX\_CHAR\_WIDTH

            if x\_shift > config.MESSAGEBOX\_LINE\_WRAP:

                y\_shift += config.MESSAGEBOX\_LINE\_HEIGHT

                x\_shift = 0

            glyph = self.font.render(char, False, config.COLOR\_MESSAGEBOX\_BORDER)

            game.display.blit(

                glyph,

                (self.textbox\_pos[0] + x\_shift, self.textbox\_pos[1] + y\_shift)

            )

            # Анимация «печати»

            game.Screen.blit(

                pygame.transform.scale(game.display, game.Screen.get\_size()),

                (0, 0)

            )

            pygame.display.update()

        # Ждём пробела

        exit\_dialog = False

        while not exit\_dialog:

            for event in pygame.event.get():

                if event.type == pygame.KEYDOWN and event.key == pygame.K\_SPACE:

                    exit\_dialog = True

class Animation:

    def \_\_init\_\_(self,images, duration=5, loop=True):

        self.images = images

        self.loop = loop

        self.duration = duration

        self.done = False

        self.frame = 0

    def copy(self):

        return Animation(self.images,self.duration,self.loop)

    def update(self):

        if self.loop:

            self.frame=(self.frame +1)%(self.duration\*len(self.images))

        else:

            self.frame = min(self.frame+1,self.duration\*len(self.images)-1)

            if self.frame>= self.duration \* len(self.images)-1:

                self.done = True

    def img(self):

        return self.images[int(self.frame/self.duration)]

import Scripts.tilemap as TMap

import sys

from Scripts.utils import Animation, image\_load, images\_load, Textbox, Messagebox

import pygame

import config

config.EDITOR\_RENDER\_SCALE = 2.0

class Editor:

    def \_\_init\_\_(self):

        pygame.init()

        pygame.display.set\_caption("Editor")

        self.Screen = pygame.display.set\_mode((config.SCREEN\_WIDTH, config.SCREEN\_HEIGHT))

        self.display = pygame.Surface((config.DISPLAY\_WIDTH, config.DISPLAY\_HEIGHT))

        self.clock = pygame.time.Clock()

        self.camera\_shift = [0,0]

        self.assets = {

            'decor':        images\_load('tiles/decor'),

            'glass':        images\_load('tiles/glass'),

            'surface':      images\_load('tiles/surface'),

            'large\_decor':  images\_load('tiles/large\_decor'),

            'Jumper':       images\_load('functional\_blocks/jumper'),

            'Exit':         images\_load('functional\_blocks/Exit'),

            # ← сюда добавляем новые «гейз-объекты»

            #'GazePlatform': images\_load('functional\_blocks/GazePlatform'),

            #'GazeDoor':     images\_load('functional\_blocks/GazeDoor'),

            #'GazeLight':    images\_load('functional\_blocks/GazeLight'),

        }

        self.level = "map2.json"

        self.camera\_info = ["only\_x","only\_y","free"]

        self.camera\_type = 0

        self.tilemap = TMap.Tilemap(self, config.TILE\_SIZE)

        self.movement = [0,0,0,0]

        self.scroll = [0,0]

        self.tile\_list = list(self.assets)

        self.tile\_group = 0

        self.tile\_variant = 0

        self.Clicking = False

        self.right\_clicking = False

        self.shift = False

        self.ongrid = True

        self.player\_position = (50,50)

        self.tilemap.eload("data/Maps/map2.json")

    def ui(self):

        level\_info = Textbox((10,20),self.level)

        camera\_info = Textbox((120,20),self.camera\_info[self.camera\_type])

        level\_info.render(self.display,0)

        camera\_info.render(self.display,0)

    def run(self):

        intro = Messagebox("L - level info edit, O - save level, LKM/RKM - tiles add/delete, alt - offgrid\_tiles",(20,20))

        intro.render(self)

        ui\_enable = True

        level\_edit = False

        level\_select = 0;

        while True:

            self.display.fill(config.COLOR\_BG)

            self.scroll[0]+=(self.movement[1]-self.movement[0])

            self.scroll[1]+=(self.movement[3]-self.movement[2])

            current\_tile\_img = self.assets[self.tile\_list[self.tile\_group]][self.tile\_variant].copy()

            current\_tile\_img.set\_alpha(100)

            render\_scroll = (int(self.scroll[0]),int(self.scroll[1]))

            self.tilemap.render(self.display,shift = render\_scroll)

            if ui\_enable:

                self.ui()

            mouse\_pos = pygame.mouse.get\_pos()

            mouse\_pos = (mouse\_pos[0]/ config.EDITOR\_RENDER\_SCALE, mouse\_pos[1]/config.EDITOR\_RENDER\_SCALE)

            tile\_pos = (int((mouse\_pos[0]+self.scroll[0])//self.tilemap.tile\_size),int((mouse\_pos[1]+self.scroll[1])//self.tilemap.tile\_size))

            if self.Clicking and self.ongrid:

                self.tilemap.tilemap[str(tile\_pos[0])+";"+str(tile\_pos[1])] = {"type": self.tile\_list[self.tile\_group],"variant": self.tile\_variant, "pos": tile\_pos}

            if self.right\_clicking and self.ongrid:

                tile\_loc = str(str(tile\_pos[0])+";"+str(tile\_pos[1]))

                if tile\_loc in self.tilemap.tilemap:

                    del self.tilemap.tilemap[tile\_loc]

            if self.right\_clicking and not self.ongrid:

                for tile in self.tilemap.offgrid\_tiles.copy():

                    tile\_img = self.assets[tile["type"]][tile["variant"]]

                    tile\_r = pygame.Rect(tile["pos"][0]-self.scroll[0],tile["pos"][1]-self.scroll[1],tile\_img.get\_width(),tile\_img.get\_height())

                    if tile\_r.collidepoint(mouse\_pos[0],mouse\_pos[1]):

                        self.tilemap.offgrid\_tiles.remove(tile)

            self.display.blit(current\_tile\_img,mouse\_pos)

            for event in pygame.event.get():

                match event.type:

                    case pygame.QUIT:

                        pygame.quit()

                        sys.exit()

                    case pygame.MOUSEBUTTONDOWN:

                        if event.button == 1:

                                self.Clicking = True

                                if not self.ongrid:

                                    self.tilemap.offgrid\_tiles.append({"type": self.tile\_list[self.tile\_group],"variant": self.tile\_variant, "pos":(mouse\_pos[0]+self.scroll[0],mouse\_pos[1]+self.scroll[1])})

                        if event.button == 3:

                                self.right\_clicking = True

                        if level\_edit:

                            level\_select= level\_select%60

                            if self.shift:

                                if event.button==5:

                                    self.camera\_type+=1

                                    self.camera\_type = self.camera\_type% len(self.camera\_info)

                                if event.button==4:

                                    self.camera\_type-=1

                                    self.camera\_type = self.camera\_type% len(self.camera\_info)

                            elif event.button == 5:

                                level\_select+=1

                                self.level = "map"+ str(level\_select)+ ".json"

                            elif event.button ==4:

                                level\_select-=1

                                self.level = "map"+ str(level\_select)+ ".json"

                        if self.shift:

                            if event.button == 4:

                                self.tile\_variant = (self.tile\_variant - 1) % len(self.assets[self.tile\_list[self.tile\_group]])

                            if event.button == 5:

                                self.tile\_variant = (self.tile\_variant + 1) % len(self.assets[self.tile\_list[self.tile\_group]])

                        else:

                            if event.button == 4:

                                self.tile\_group = (self.tile\_group - 1) % len(self.tile\_list)

                                self.tile\_variant = 0

                            if event.button == 5:

                                self.tile\_group = (self.tile\_group + 1) % len(self.tile\_list)

                                self.tile\_variant = 0

                    case pygame.MOUSEBUTTONUP:

                        if event.button == 1:

                                self.Clicking = False

                        if event.button == 3:

                                self.right\_clicking = False

                    case pygame.KEYDOWN:

                        match event.key:

                            case pygame.K\_a:

                                self.movement[0] = config.SCROLL\_SPEED

                            case pygame.K\_d:

                                self.movement[1] = config.SCROLL\_SPEED

                            case pygame.K\_w:

                                self.movement[2] = config.SCROLL\_SPEED

                            case pygame.K\_s:

                                self.movement[3] = config.SCROLL\_SPEED

                            case pygame.K\_o:

                                self.tilemap.save("data/Maps/"+self.level,self.camera\_info[self.camera\_type],list(self.player\_position))

                            case pygame.K\_p:

                                self.player\_position = (tile\_pos[0]\*self.tilemap.tile\_size,tile\_pos[1]\*self.tilemap.tile\_size)

                            case pygame.K\_LALT:

                                self.ongrid = False

                            case pygame.K\_ESCAPE:

                                import Main

                            case pygame.K\_LSHIFT:

                                self.shift = True

                            case pygame.K\_h:

                                ui\_enable = not ui\_enable

                            case pygame.K\_l:

                                level\_edit = not level\_edit

                    case pygame.KEYUP:

                        match event.key:

                            case pygame.K\_a:

                                self.movement[0] = 0

                            case pygame.K\_d:

                                self.movement[1] = 0

                            case pygame.K\_w:

                                self.movement[2] = 0

                            case pygame.K\_s:

                                self.movement[3] = 0

                            case pygame.K\_LALT:

                                self.ongrid = True

                            case pygame.K\_LSHIFT:

                                self.shift = False

            self.clock.tick(60)

            self.Screen.blit(pygame.transform.scale(self.display,self.Screen.get\_size()),(0,0))

            pygame.display.update()

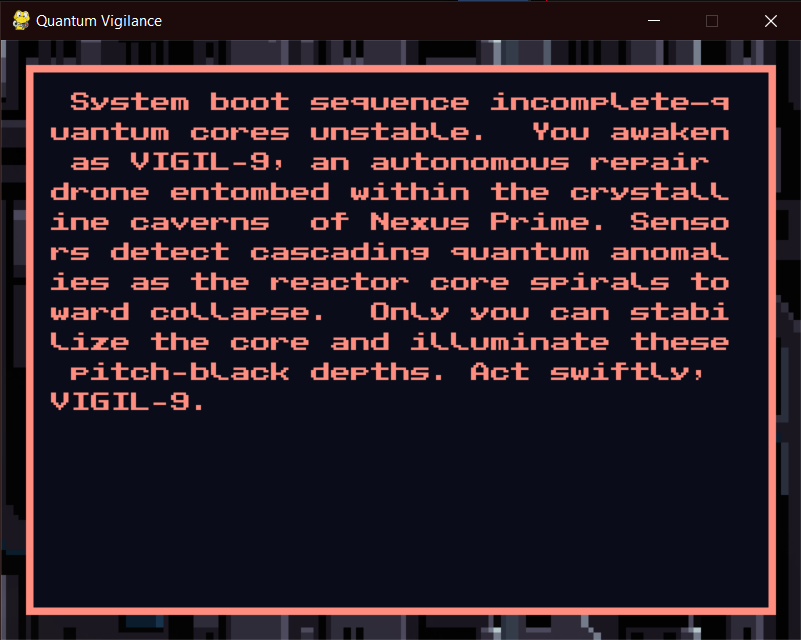
Editor().run()

**Приложение 2.**









**Приложение 3.**

Репозиторий: https://github.com/Agtemiun/Quantum\_Vigilance

Ссылка на презентацию: https://github.com/Agtemiun/Quantum\_Vigilance/blob/main/Report/Presentation.pptx