import os

import json

import time

import pygame

import sys

# Константы

WIDTH, HEIGHT = 1440, 810 # Увеличенные размеры окна

INITIAL\_POINTS = 100 # Начальное количество очков

# Цвета

WHITE = (255, 255, 255)

class VirtualVirusApp:

def \_\_init\_\_(self):

*"""Инициализирует игровой класс.*

*Устанавливает параметры игры, загружает изображения, иконки и*

*начальные параметры кастомизации. Инициализирует статистику игрока*

*и запускает основной игровой цикл.*

*Загружает следующие ресурсы:*

*- Изображения для иконок состояния (Здоровье, Голод, Счастье, Чистота, Сон).*

*- Изображения для кастомизации (шляпы, платья, обувь).*

*- Фоновое изображение и изображение персонажа.*

*Инициализирует кнопки для взаимодействия с игроком:*

*- Лечить*

*- Кормить*

*- Играть*

*- Очистить*

*- Спать*

*"""*

pygame.init()

self.screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Мой виртуальный вирус")

self.timer = 0

self.last\_update\_time = time.time()

# Добавьте начальные параметры кастомизации

self.current\_hat = 0

self.current\_dress = 0

self.current\_shoes = 0

self.load\_customization() # Загружаем кастомизацию при запуске

self.buttons = {

"Лечить": lambda: self.update\_stat("Здоровье", 5),

"Кормить": lambda: self.update\_stat("Голод", 5),

"Играть": lambda: self.update\_stat("Счастье", 5),

"Очистить": lambda: self.update\_stat("Чистота", 5),

"Спать": lambda: self.update\_stat("Сон", 5),

}

# Инициализация иконок

self.icons = {

"Здоровье": {

"good": pygame.transform.scale(pygame.image.load("health\_good.png"), (70, 70)),

"middle": pygame.transform.scale(pygame.image.load("health\_middle.png"), (70, 70)),

"bad": pygame.transform.scale(pygame.image.load("health\_bad.png"), (70, 70)),

},

"Голод": {

"good": pygame.transform.scale(pygame.image.load("hunger\_good.png"), (70, 70)),

"middle": pygame.transform.scale(pygame.image.load("hunger\_middle.png"), (70, 70)),

"bad": pygame.transform.scale(pygame.image.load("hunger\_bad.png"), (70, 70)),

},

"Счастье": {

"good": pygame.transform.scale(pygame.image.load("happiness\_good.png"), (70, 70)),

"middle": pygame.transform.scale(pygame.image.load("happiness\_middle.png"), (70, 70)),

"bad": pygame.transform.scale(pygame.image.load("happiness\_bad.png"), (70, 70)),

},

"Чистота": {

"good": pygame.transform.scale(pygame.image.load("clean\_good.png"), (70, 70)),

"middle": pygame.transform.scale(pygame.image.load("clean\_middle.png"), (70, 70)),

"bad": pygame.transform.scale(pygame.image.load("clean\_bad.png"), (70, 70)),

},

"Сон": {

"good": pygame.transform.scale(pygame.image.load("sleep\_good.png"), (70, 70)),

"middle": pygame.transform.scale(pygame.image.load("sleep\_middle.png"), (70, 70)),

"bad": pygame.transform.scale(pygame.image.load("sleep\_bad.png"), (70, 70)),

},

}

# Загрузка изображений для кастомизации

self.hat\_images = [

self.load\_image('assets/hat.png', (500, 750)),

self.load\_image('assets/cap.png', (500, 750)),

self.load\_image('assets/beret.png', (500, 750)),

self.load\_image('assets/glasses.png', (500, 750)),

self.load\_image('assets/shapka.png', (500, 750)),

self.load\_image('assets/empty.png', (500, 750)),

]

self.dress\_images = [

self.load\_image('assets/dress.png', (500, 750)),

self.load\_image('assets/kostum.png', (500, 750)),

self.load\_image('assets/pijama.png', (500, 750)),

self.load\_image('assets/t-shirt.png', (500, 750)),

self.load\_image('assets/uniform.png', (500, 750)),

self.load\_image('assets/empty.png', (500, 750)),

]

self.shoe\_images = [

self.load\_image('assets/shoes.png', (500, 750)),

self.load\_image('assets/boots.png', (500, 750)),

self.load\_image('assets/kedy.png', (500, 750)),

self.load\_image('assets/snickers.png', (500, 750)),

self.load\_image('assets/slippers.png', (500, 750)),

self.load\_image('assets/empty.png', (500, 750)),

]

# Остальная инициализация

self.background\_image = self.load\_image("fon.png", (WIDTH, HEIGHT))

self.avatar\_image = self.load\_image("avatarka.png", (70, 70))

self.character\_image = self.load\_image("assets/char.png", (500, 750))

self.stats = {

"Здоровье": 50,

"Голод": 50,

"Счастье": 50,

"Чистота": 50,

"Сон": 50,

}

self.player\_name = "Игрок"

self.player\_level = 1

self.player\_points = INITIAL\_POINTS

self.font = pygame.font.Font(None, 36)

self.load\_data()

self.run()

def run(self):

*"""Запускает основной игровой цикл.*

*Этот метод отвечает за отображение экрана приветствия и*

*дальнейшее управление игровым процессом. Он может включать*

*обработку событий, обновление состояния игры и отрисовку*

*элементов на экране.*

*В текущей реализации метод вызывает функцию для показа*

*экрана приветствия.*

*"""*

self.show\_welcome\_screen() # Показать экран приветствия

def run\_game(self):

*"""Запускает основной игровой цикл игры.*

*Этот метод отвечает за обработку событий, обновление состояния*

*игры и отрисовку элементов на экране. Он выполняет следующие*

*действия:*

*1. Обрабатывает события, такие как выход из игры.*

*2. Очищает экран.*

*3. Отрисовывает кастомизированного персонажа.*

*4. Обновляет экран для отображения изменений.*

*Цикл продолжается до тех пор, пока переменная `running`*

*остается истинной. При получении события выхода из игры*

*происходит завершение работы Pygame и выход из программы.*

*"""*

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

# Очистка экрана

self.screen.fill((255, 255, 255))

# Отрисовка кастомизированного персонажа

self.draw\_customized\_character()

pygame.display.flip()

def main\_game\_loop(self):

*"""Основной игровой цикл.*

*Этот метод отвечает за непрерывное выполнение игры, включая*

*обработку событий, обновление состояния игры и отрисовку*

*элементов на экране. Цикл выполняется бесконечно, пока*

*игра активна.*

*В процессе выполнения метода:*

*1. Обрабатываются события, такие как нажатия клавиш и выход из игры.*

*2. Каждые 60 секунд обновляются значения состояний (например, здоровье, голод и т.д.).*

*3. Отрисовываются все элементы на экране.*

*4. Обновляется окно для отображения изменений.*

*Этот метод должен вызываться для запуска игрового процесса.*

*"""*

while True:

self.handle\_events() # Обработка событий

# Проверка времени для уменьшения статов

current\_time = time.time()

if current\_time - self.last\_update\_time >= 60: # Каждые 60 секунд

self.update\_stats\_over\_time() # Уменьшаем значения состояний

self.last\_update\_time = current\_time # Обновляем время последнего обновления

self.draw() # Отрисовка элементов на экране

pygame.display.flip() # Обновление окна

def load\_customization(self):

*"""Загружает данные кастомизации из файла.*

*Этот метод проверяет наличие файла "customization\_data.json".*

*Если файл существует, он загружает данные кастомизации*

*(шляпа, платье и обувь) и обновляет соответствующие атрибуты*

*текущего объекта.*

*Атрибуты, которые обновляются:*

*- current\_hat: Индекс текущей шляпы (по умолчанию 0).*

*- current\_dress: Индекс текущего платья (по умолчанию 0).*

*- current\_shoes: Индекс текущей обуви (по умолчанию 0).*

*Если файл не существует, значения атрибутов остаются без изменений.*

*"""*

if os.path.exists("customization\_data.json"):

with open("customization\_data.json", "r") as f:

customization\_data = json.load(f)

self.current\_hat = customization\_data.get("hat", 0)

self.current\_dress = customization\_data.get("dress", 0)

self.current\_shoes = customization\_data.get("shoes", 0)

def show\_welcome\_screen(self):

*"""Отображает экран приветствия.*

*Этот метод проверяет, существует ли аккаунт игрока.*

*Если имя игрока равно "Игрок", вызывается метод для*

*создания нового аккаунта. Если аккаунт уже существует,*

*вызывается метод для отображения экрана существующего аккаунта.*

*В зависимости от состояния аккаунта, метод выполняет одно из следующих действий:*

*- Вызывает метод `create\_account\_screen()` для создания нового аккаунта.*

*- Вызывает метод `existing\_account\_screen()` для отображения информации о существующем аккаунте.*

*"""*

if self.player\_name == "Игрок": # Если аккаунта нет

self.create\_account\_screen()

else: # Если аккаунт уже существует

self.existing\_account\_screen()

def open\_customization(self):

*"""Открывает экран кастомизации.*

*Этот метод создает экземпляр класса `CustomizationScreen` и*

*запускает его, позволяя пользователю настраивать внешний вид*

*персонажа.*

*В процессе выполнения метода:*

*- Создается новый экран кастомизации, передавая текущий объект*

*в качестве параметра.*

*- Запускается метод `run()` для отображения экрана кастомизации*

*и обработки взаимодействия пользователя.*

*"""*

customization\_screen = CustomizationScreen(self)

customization\_screen.run()

def update\_stats\_over\_time(self):

*"""Уменьшает значения состояний, кроме здоровья, на 1 каждую минуту.*

*Этот метод отвечает за обновление статистики игрока, уменьшая*

*значения состояний (например, голод, счастье, чистота и сон)*

*на 1 каждую минуту. Значение здоровья не изменяется.*

*В процессе выполнения метода:*

*- Для каждого состояния в словаре `self.stats`, кроме "Здоровье",*

*значение уменьшается на 1.*

*- Значение состояния не опускается ниже 0.*

*"""*

for stat in self.stats:

if stat != "Здоровье":

self.stats[stat] = max(self.stats[stat] - 1, 0) # Уменьшаем значение, не опуская ниже 0

def create\_account\_screen(self):

*"""Отображает экран создания аккаунта.*

*Этот метод отвечает за отображение интерфейса для создания*

*нового аккаунта игрока. Он позволяет пользователю ввести*

*имя питомца и перейти к основной игре после успешного*

*создания аккаунта.*

*В процессе выполнения метода:*

*- Отображается текст приветствия и инструкции по вводу имени.*

*- Пользователь может вводить имя питомца, которое сохраняется*

*при нажатии клавиши Enter.*

*- Если имя пустое, нажатие Enter игнорируется.*

*- Пользователь может выйти из игры, нажав на кнопку "Выйти из игры".*

*После успешного ввода имени питомца, метод сохраняет данные*

*и запускает основной игровой цикл.*

*"""*

input\_box = pygame.Rect(WIDTH // 2 - 100, HEIGHT // 2 - 20, 200, 40)

color\_inactive = pygame.Color('lightskyblue3')

color\_active = pygame.Color('dodgerblue2')

color = color\_inactive

active = False

text = ''

welcome\_text = self.font.render("Добро пожаловать в игру!", True, (0, 0, 0))

start\_text = self.font.render("Введите имя вашего питомца:", True, (0, 0, 0))

exit\_text = self.font.render("Выйти из игры", True, (0, 0, 0))

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.MOUSEBUTTONDOWN:

if input\_box.collidepoint(event.pos):

active = not active

else:

active = False

if event.type == pygame.KEYDOWN:

if active:

if event.key == pygame.K\_RETURN:

if text: # Если имя не пустое

self.player\_name = text

self.stats = {stat: 50 for stat in self.stats} # Начальные значения для питомца

self.save\_data() # Сохраняем данные после создания питомца

self.main\_game\_loop() # Переход к основной игре

return # Выходим из окна создания аккаунта

else:

continue # Игнорируем нажатие Enter, если имя пустое

elif event.key == pygame.K\_BACKSPACE:

text = text[:-1]

else:

text += event.unicode

# Отображаем экран создания аккаунта

self.screen.fill(WHITE)

self.screen.blit(self.background\_image, (0, 0))

self.screen.blit(welcome\_text, (WIDTH // 2 - welcome\_text.get\_width() // 2, HEIGHT // 2 - 100))

self.screen.blit(start\_text, (WIDTH // 2 - start\_text.get\_width() // 2, HEIGHT // 2 - 50))

txt\_surface = self.font.render(text, True, color)

width = max(200, txt\_surface.get\_width() + 10)

input\_box.w = width

self.screen.blit(txt\_surface, (input\_box.x + 5, input\_box.y + 5))

pygame.draw.rect(self.screen, color, input\_box, 2)

# Отображаем кнопку "Выйти из игры"

exit\_button\_rect = pygame.Rect(WIDTH // 2 - 100, HEIGHT // 2 + 100, 200, 40)

pygame.draw.rect(self.screen, (205, 231, 238), exit\_button\_rect)

self.screen.blit(exit\_text, (exit\_button\_rect.x + 5, exit\_button\_rect.y + 5))

# Проверка нажатия кнопки "Выйти из игры"

if exit\_button\_rect.collidepoint(pygame.mouse.get\_pos()) and pygame.mouse.get\_pressed()[0]:

self.exit\_game() # Вызов метода выхода из игры

pygame.display.flip()

def existing\_account\_screen(self):

*"""*

*Отображает экран для существующего аккаунта, приветствуя игрока и предоставляя*

*возможность продолжить игру или выйти. Также включает кнопку для удаления аккаунта.*

*Этот метод создает графический интерфейс с текстом приветствия, уровнем игрока,*

*инструкциями по продолжению игры и выходу. Он также отображает кнопку для удаления*

*аккаунта и обрабатывает соответствующие события.*

*Основные функции:*

*- Отображение приветственного текста с именем игрока и уровнем.*

*- Обработка нажатий клавиш для продолжения игры или выхода.*

*- Отображение кнопки "Удалить аккаунт" и обработка нажатия на нее.*

*Примечания:*

*- Для продолжения игры игрок должен нажать клавишу 'C'.*

*- Для выхода из игры игрок должен нажать клавишу 'ESC'.*

*- Удаление аккаунта происходит при нажатии на кнопку "Удалить аккаунт".*

*Args:*

*self: Ссылка на экземпляр класса, содержащего информацию о состоянии игры,*

*включая имя игрока, уровень, шрифт и экран.*

*Raises:*

*SystemExit: Вызывается при закрытии окна или выходе из игры.*

*"""*

welcome\_text = self.font.render(f"Добро пожаловать, {self.player\_name}!", True, (0, 0, 0))

level\_text = self.font.render(f"Уровень: {self.player\_level}", True, (0, 0, 0))

continue\_text = self.font.render("Нажмите 'C' для продолжения", True, (0, 0, 0))

exit\_text = self.font.render("Выйти из игры", True, (0, 0, 0))

# Кнопка "Удалить аккаунт"

delete\_button\_rect = pygame.Rect(WIDTH // 2 - 100, HEIGHT // 2 + 150, 200, 40)

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_c: # Нажатие 'C' для продолжения игры

self.main\_game\_loop() # Переход к основной игре

return # Выходим из окна приветствия

elif event.key == pygame.K\_ESCAPE: # Нажатие 'ESC' для выхода

self.exit\_game()

# Отображаем экран приветствия

self.screen.fill(WHITE)

self.screen.blit(self.background\_image, (0, 0))

self.screen.blit(welcome\_text, (WIDTH // 2 - welcome\_text.get\_width() // 2, HEIGHT // 2 - 100))

self.screen.blit(level\_text, (WIDTH // 2 - level\_text.get\_width() // 2, HEIGHT // 2 - 50))

self.screen.blit(continue\_text, (WIDTH // 2 - continue\_text.get\_width() // 2, HEIGHT // 2 + 50))

self.screen.blit(exit\_text, (WIDTH // 2 - exit\_text.get\_width() // 2, HEIGHT // 2 + 100))

# Отображаем кнопку "Удалить аккаунт"

pygame.draw.rect(self.screen, (205, 231, 238), delete\_button\_rect)

delete\_button\_text = self.font.render("Удалить аккаунт", True, (5, 82, 123))

self.screen.blit(delete\_button\_text, (delete\_button\_rect.x + 5, delete\_button\_rect.y + 5))

# Проверка нажатия кнопки "Удалить аккаунт"

if delete\_button\_rect.collidepoint(pygame.mouse.get\_pos()) and pygame.mouse.get\_pressed()[0]:

self.delete\_account() # Удаление аккаунта

pygame.display.flip() # Обновляем экран

def add\_points(self, points):

*"""*

*Добавляет очки к текущему значению очков игрока.*

*Этот метод увеличивает количество очков игрока на заданное значение и*

*сохраняет обновленные данные.*

*Args:*

*points (int): Количество очков, которое нужно добавить к текущему значению.*

*Raises:*

*ValueError: Если переданное значение очков не является целым числом или*

*меньше нуля.*

*"""*

self.player\_points += points

self.save\_data() # Сохраняем данные после добавления очков

def load\_customization(self):

*"""*

*Загружает кастомизацию игрока из файла.*

*Этот метод читает данные кастомизации из файла `customization\_data.json` и*

*обновляет текущие параметры игрока, такие как шляпа, платье и обувь. Если файл*

*не найден, используются значения по умолчанию.*

*Raises:*

*FileNotFoundError: Если файл `customization\_data.json` не найден.*

*"""*

try:

with open("customization\_data.json", "r") as f:

customization\_data = json.load(f)

self.current\_hat = customization\_data.get("hat", 0)

self.current\_dress = customization\_data.get("dress", 0)

self.current\_shoes = customization\_data.get("shoes", 0)

except FileNotFoundError:

print("Файл кастомизации не найден. Используются значения по умолчанию.")

def handle\_events(self):

*"""*

*Обрабатывает события Pygame.*

*Этот метод проверяет события, происходящие в игре, и выполняет соответствующие*

*действия. В частности, он обрабатывает событие выхода из игры.*

*Raises:*

*SystemExit: Вызывается при закрытии окна игры.*

*"""*

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

def draw(self):

*"""*

*Отображает графику на экране.*

*Этот метод отвечает за отрисовку всех элементов интерфейса, включая фон,*

*аватар игрока, текстовые данные (имя, уровень, очки), кастомизированного*

*персонажа, иконки состояния, кнопки и шкалы. Он также обрабатывает*

*взаимодействие с кнопками.*

*Примечания:*

*- Функция обновляет экран после отрисовки всех элементов.*

*- Обрабатывает нажатия кнопок и вызывает соответствующие действия.*

*"""*

# Отображаем фон

self.screen.blit(self.background\_image, (0, 0))

# Рисуем аватар в левом верхнем углу

self.screen.blit(self.avatar\_image, (20, 20)) # Положение аватара

name\_text = self.font.render(self.player\_name, True, (0, 0, 0))

level\_text = self.font.render(f"Уровень: {self.player\_level}", True, (0, 0, 0))

points\_text = self.font.render(f"Очки: {self.player\_points}", True, (0, 0, 0))

self.screen.blit(name\_text, (100, 20)) # Положение имени

self.screen.blit(level\_text, (100, 50)) # Положение уровня

self.screen.blit(points\_text, (100, 80)) # Положение очков

# Рисуем кастомизированного персонажа

self.draw\_customized\_character()

# Рисуем иконки и шкалы по верхней грани окна

icon\_start\_x = 300 # Начальная позиция для иконок состояния

icon\_spacing = (WIDTH - icon\_start\_x) // len(self.stats) # Расстояние между иконками

for i, (stat, value) in enumerate(self.stats.items()):

if value >= 68:

icon = self.icons[stat]["good"]

elif value >= 34:

icon = self.icons[stat]["middle"]

else:

icon = self.icons[stat]["bad"]

icon\_x = icon\_start\_x + i \* icon\_spacing

self.screen.blit(icon, (icon\_x, 20))

pygame.draw.rect(self.screen, (0, 0, 0), (icon\_x, 100, 100, 20), 2) # Увеличенная шкала

pygame.draw.rect(self.screen, (0, 255, 0), (icon\_x, 100, value, 20)) # Увеличенная шкала

# Отображаем значение

value\_text = self.font.render(str(value), True, (0, 0, 0))

self.screen.blit(value\_text, (icon\_x + 110, 98))

# Подпись к иконке

label\_text = self.font.render(stat.capitalize(), True, (0, 0, 0))

self.screen.blit(label\_text, (icon\_x, 120))

# Рисуем кнопки состояния внизу экрана

button\_width = 110

button\_height = 40

button\_spacing = (WIDTH - 50 \* 2 - button\_width \* len(self.buttons)) // (len(self.buttons) - 1)

for i, (text, action) in enumerate(self.buttons.items()):

button\_x = icon\_start\_x + i \* button\_spacing # Расположение кнопок прямо под иконками

button\_rect = pygame.Rect(button\_x, HEIGHT - 45, button\_width, button\_height)

pygame.draw.rect(self.screen, (205, 231, 238), button\_rect) # Измененный цвет кнопки

button\_text = self.font.render(text, True, (5, 82, 123)) # Измененный цвет текста

self.screen.blit(button\_text, (button\_rect.x + 5, button\_rect.y + 5))

# Проверка нажатия кнопки

if button\_rect.collidepoint(pygame.mouse.get\_pos()):

if pygame.mouse.get\_pressed()[0]: # Если кнопка мыши нажата

action() # Вызываем действие, связанное с кнопкой

pygame.time.delay(200) # Задержка, чтобы избежать многократного нажатия

break # Выходим из цикла, чтобы избежать повторного срабатывания

# Рисуем кнопки под аватаром

button\_width = 150

button\_height = 40

button\_y = 130 # Положение кнопок под аватаром

# Кнопка "Выйти из игры"

exit\_button\_rect = pygame.Rect(20, button\_y, 200, button\_height)

pygame.draw.rect(self.screen, (205, 231, 238), exit\_button\_rect)

exit\_button\_text = self.font.render("Выйти из игры", True, (5, 82, 123))

self.screen.blit(exit\_button\_text, (exit\_button\_rect.x + 5, exit\_button\_rect.y + 5))

# Кнопка "Удалить аккаунт"

delete\_button\_rect = pygame.Rect(20, button\_y + button\_height + 10, 230, button\_height)

pygame.draw.rect(self.screen, (205, 231, 238), delete\_button\_rect)

delete\_button\_text = self.font.render("Удалить аккаунт", True, (5, 82, 123))

self.screen.blit(delete\_button\_text, (delete\_button\_rect.x + 5, delete\_button\_rect.y + 5))

# Кнопка "Кастомизация" рядом с кнопкой "Удалить аккаунт"

customization\_button\_rect = pygame.Rect(20, button\_y + (button\_height + 10) \* 2, 230, button\_height)

pygame.draw.rect(self.screen, (205, 231, 238), customization\_button\_rect)

customization\_button\_text = self.font.render("Кастомизация", True, (5, 82, 123))

self.screen.blit(customization\_button\_text, (customization\_button\_rect.x + 5, customization\_button\_rect.y + 5))

# Кнопка "Мини-игры"

mini\_games\_button\_rect = pygame.Rect(20, button\_y + (button\_height + 10) \* 3, 230, button\_height)

pygame.draw.rect(self.screen, (205, 231, 238), mini\_games\_button\_rect)

mini\_games\_button\_text = self.font.render("Мини-игры", True, (5, 82, 123))

self.screen.blit(mini\_games\_button\_text, (mini\_games\_button\_rect.x + 5, mini\_games\_button\_rect.y + 5))

# Проверка нажатия кнопки "Мини-игры"

if mini\_games\_button\_rect.collidepoint(pygame.mouse.get\_pos()) and pygame.mouse.get\_pressed()[0]:

self.open\_mini\_games\_menu() # Открываем меню мини-игр

# Проверка нажатия кнопок

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

if exit\_button\_rect.collidepoint(mouse\_pos) and pygame.mouse.get\_pressed()[0]:

self.exit\_game() # Выход из игры

elif delete\_button\_rect.collidepoint(mouse\_pos) and pygame.mouse.get\_pressed()[0]:

self.delete\_account() # Удаление аккаунта

elif customization\_button\_rect.collidepoint(mouse\_pos) and pygame.mouse.get\_pressed()[0]:

self.open\_customization() # Открытие кастомизации

pygame.display.flip() # Обновляем экран

def draw\_customized\_character(self):

*"""*

*Отрисовывает кастомизированного персонажа на экране.*

*Этот метод отвечает за отображение персонажа с учетом текущих элементов кастомизации,*

*таких как шляпа, платье и обувь. Персонаж рисуется в центре экрана, а элементы*

*кастомизации позиционируются относительно его тела.*

*Примечания:*

*- Позиционирование элементов кастомизации осуществляется с учетом их размеров.*

*- Если элемент кастомизации отсутствует, он не отображается.*

*"""*

base\_x = (WIDTH - self.character\_image.get\_width()) // 2

base\_y = (HEIGHT - self.character\_image.get\_height()) // 2 + 60

# Рисуем персонажа в центре экрана

character\_x = (WIDTH - self.character\_image.get\_width()) // 2

character\_y = (HEIGHT - self.character\_image.get\_height()) // 2 + 60

self.screen.blit(self.character\_image, (character\_x, character\_y))

# Отображаем текущие элементы кастомизации

hat\_image = self.hat\_images[self.current\_hat]

dress\_image = self.dress\_images[self.current\_dress]

shoes\_image = self.shoe\_images[self.current\_shoes]

# Позиционирование одежды

if dress\_image:

self.screen.blit(dress\_image, (base\_x, base\_y)) # Платье на уровне тела

# Позиционирование головного убора

if hat\_image:

self.screen.blit(hat\_image, (base\_x, base\_y - hat\_image.get\_height() + 750)) # Поднимаем шляпу над головой

# Позиционирование обуви

if shoes\_image:

self.screen.blit(shoes\_image, (

base\_x, base\_y + self.character\_image.get\_height() - shoes\_image.get\_height())) # Обувь на уровне ног

def open\_mini\_games\_menu(self):

*"""*

*Открывает меню мини-игр.*

*Этот метод запускает меню мини-игр, позволяя игроку выбрать и начать*

*одну из доступных мини-игр.*

*"""*

run\_menu(self) # Запускаем меню мини-игр

def update\_stat(self, stat, value):

*"""*

*Обновляет значение указанной шкалы состояния.*

*Этот метод увеличивает значение состояния игрока (например, здоровья,*

*энергии и т.д.) на заданное количество, если у игрока достаточно очков.*

*Если текущее значение уже равно 100, обновление не происходит.*

*Args:*

*stat (str): Название состояния, которое нужно обновить.*

*value (int): Значение, на которое нужно увеличить состояние.*

*Примечания:*

*- Стоимость повышения состояния составляет 5 очков.*

*- Если для достижения 100 требуется меньше очков, чем стоимость,*

*состояние будет увеличено до 100, и будут потрачены только*

*необходимые очки.*

*- Данные сохраняются после обновления состояния.*

*"""*

cost = 5 # Стоимость повышения состояния

current\_value = self.stats[stat]

# Если текущее значение уже 100, ничего не делаем

if current\_value >= 100:

return

# Вычисляем, сколько нужно для достижения 100

needed = 100 - current\_value

# Проверяем, достаточно ли очков для увеличения

if self.player\_points >= cost:

if needed < cost: # Если нужно меньше очков, чем стоимость

self.stats[stat] = 100 # Увеличиваем до 100

self.player\_points -= needed # Тратим только нужное количество очков

else:

self.stats[stat] = min(current\_value + value, 101) # Увеличиваем значение, не превышая 100

self.player\_points -= cost # Уменьшаем количество очков

self.last\_update\_time = time.time() # Обновляем время последнего обновления

self.save\_data() # Сохраняем данные после обновления

def save\_data(self):

*"""*

*Сохраняет данные игрока в файл.*

*Этот метод создает словарь с текущими данными игрока, включая имя, уровень,*

*очки, статистику и время последнего обновления, и записывает его в файл*

*`player\_data.json`.*

*"""*

data = {

"player\_name": self.player\_name,

"player\_level": self.player\_level,

"player\_points": self.player\_points,

"stats": self.stats,

"last\_update\_time": time.time(),

}

with open("player\_data.json", "w") as f:

json.dump(data, f)

def exit\_game(self):

*"""*

*Выходит из игры с сохранением данных.*

*Этот метод сохраняет текущие данные игрока, закрывает Pygame и завершает*

*выполнение программы.*

*"""*

self.save\_data() # Сохраняем данные перед выходом

pygame.quit()

sys.exit()

def delete\_account(self):

*"""*

*Удаляет аккаунт и данные игрока.*

*Этот метод удаляет файл с данными игрока, сбрасывает имя, уровень, очки*

*и статистику, а также отображает экран приветствия снова.*

*"""*

if os.path.exists("player\_data.json"):

os.remove("player\_data.json") # Удаляем файл с данными

self.player\_name = "Игрок" # Сбрасываем имя

self.player\_level = 1 # Сбрасываем уровень

self.player\_points = INITIAL\_POINTS # Сбрасываем очки

self.stats = {stat: 0 for stat in self.stats} # Сбрасываем статистику

self.show\_welcome\_screen() # Показать экран приветствия снова

self.last\_update\_time = time.time()

def apply\_time\_effect(self, elapsed\_minutes):

*"""*

*Уменьшает значения состояний в зависимости от прошедшего времени.*

*Этот метод обновляет значения состояний игрока, уменьшая их на количество*

*минут, прошедших с последнего обновления. Состояние "Здоровье" не*

*уменьшается.*

*Args:*

*elapsed\_minutes (int): Количество минут, прошедших с последнего обновления.*

*Примечания:*

*- Значения состояний не могут быть меньше 0.*

*"""*

for stat in self.stats:

if stat != "Здоровье": # Например, здоровье не уменьшается

self.stats[stat] = max(self.stats[stat] - elapsed\_minutes, 0)

def show\_main\_screen(self):

*"""*

*Показывает главный экран с персонажем.*

*Этот метод запускает основной игровой цикл, который отвечает за*

*отображение главного экрана игры и управление игровым процессом.*

*"""*

self.main\_game\_loop() # Запускаем основной игровой цикл

def load\_data(self):

*"""Загрузка данных игрока из файла.*

*Эта функция загружает данные игрока из файла `player\_data.json`.*

*Если файл существует, данные считываются и присваиваются соответствующим*

*атрибутам игрока. Если файл не найден, вызывается экран создания аккаунта.*

*Атрибуты, загружаемые из файла:*

*player\_name (str): Имя игрока. По умолчанию "Игрок".*

*player\_level (int): Уровень игрока. По умолчанию 1.*

*player\_points (int): Очки игрока. По умолчанию значение INITIAL\_POINTS.*

*stats (dict): Статистика игрока. По умолчанию все статусы равны 50.*

*last\_update\_time (float): Время последнего обновления в формате timestamp.*

*Если с момента последнего обновления прошло время,*

*применяется эффект времени через метод `apply\_time\_effect`.*

*Если файл не существует, вызывается метод `create\_account\_screen`.*

*Raises:*

*FileNotFoundError: Если файл `player\_data.json` не найден (необязательно,*

*так как обрабатывается условием if).*

*"""*

if os.path.exists("player\_data.json"):

with open("player\_data.json", "r") as f:

data = json.load(f)

self.player\_name = data.get("player\_name", "Игрок")

self.player\_level = data.get("player\_level", 1)

self.player\_points = data.get("player\_points", INITIAL\_POINTS)

self.stats = data.get("stats", {stat: 50 for stat in self.stats})

last\_update\_time = data.get("last\_update\_time", time.time())

# Вычисляем, сколько времени прошло с последнего обновления

current\_time = time.time()

elapsed\_minutes = int((current\_time - last\_update\_time) / 60)

if elapsed\_minutes > 0:

self.apply\_time\_effect(elapsed\_minutes)

else:

self.create\_account\_screen()

def load\_image(self, filename, size=None):

*"""Загрузка изображения с обработкой ошибок.*

*Эта функция загружает изображение из указанного файла.*

*Если указаны размеры, изображение будет масштабировано до этих размеров.*

*В случае возникновения ошибки при загрузке изображения,*

*выводится сообщение об ошибке, и функция возвращает None.*

*Args:*

*filename (str): Путь к файлу изображения, который необходимо загрузить.*

*size (tuple, optional): Кортеж из двух целых чисел (ширина, высота)*

*для изменения размера изображения. Если не указано, изображение*

*загружается в оригинальном размере.*

*Returns:*

*pygame.Surface or None: Загруженное изображение в виде объекта*

*pygame.Surface, или None, если загрузка не удалась.*

*Raises:*

*pygame.error: Исключение, возникающее при ошибке загрузки изображения,*

*которое обрабатывается внутри функции.*

*"""*

try:

image = pygame.image.load(filename)

if size:

image = pygame.transform.scale(image, size)

return image

except pygame.error as e:

print(f"Не удалось загрузить изображение {filename}: {e}")

return None # Возвращаем None, если изображение не удалось загрузить

class CustomizationScreen:

def \_\_init\_\_(self, app):

# Устанавливаем текущую кастомизацию из VirtualVirusApp

self.current\_hat = app.current\_hat

self.current\_dress = app.current\_dress

self.current\_shoes = app.current\_shoes

self.app = app

self.screen\_width, self.screen\_height = 1440, 810

self.char\_image = pygame.image.load('assets/char.png')

self.background\_image = pygame.image.load('assets/backgr.png')

self.right\_arrow = pygame.image.load('assets/right\_arrow.png')

self.left\_arrow = pygame.image.load('assets/left\_arrow.png')

self.background\_image = pygame.transform.scale(self.background\_image, (self.screen\_width, self.screen\_height))

self.hats = [

pygame.image.load('assets/hat.png'),

pygame.image.load('assets/cap.png'),

pygame.image.load('assets/beret.png'),

pygame.image.load('assets/glasses.png'),

pygame.image.load('assets/shapka.png'),

pygame.image.load('assets/empty.png'),

]

self.dresses = [

pygame.image.load('assets/dress.png'),

pygame.image.load('assets/kostum.png'),

pygame.image.load('assets/pijama.png'),

pygame.image.load('assets/t-shirt.png'),

pygame.image.load('assets/uniform.png'),

pygame.image.load('assets/empty.png'),

]

self.shoes = [

pygame.image.load('assets/shoes.png'),

pygame.image.load('assets/boots.png'),

pygame.image.load('assets/kedy.png'),

pygame.image.load('assets/snickers.png'),

pygame.image.load('assets/slippers.png'),

pygame.image.load('assets/empty.png'),

]

self.current\_hat = 0

self.current\_dress = 0

self.current\_shoes = 0

self.char\_image = pygame.transform.scale(self.char\_image, (400, 600))

self.hats = [pygame.transform.scale(hat, (400, 600)) for hat in self.hats]

self.dresses = [pygame.transform.scale(dress, (400, 600)) for dress in self.dresses]

self.shoes = [pygame.transform.scale(shoe, (400, 600)) for shoe in self.shoes]

self.button\_width = 80

self.button\_height = 80

self.button\_left\_hat = pygame.Rect(400, self.screen\_height // 2 - 200, self.button\_width, self.button\_height)

self.button\_left\_dress = pygame.Rect(400, self.screen\_height // 2 - 50, self.button\_width, self.button\_height)

self.button\_left\_shoes = pygame.Rect(400, self.screen\_height // 2 + 100, self.button\_width, self.button\_height)

self.button\_right\_hat = pygame.Rect(self.screen\_width - 530, self.screen\_height // 2 - 200, self.button\_width, self.button\_height)

self.button\_right\_dress = pygame.Rect(self.screen\_width - 530, self.screen\_height // 2 - 50, self.button\_width, self.button\_height)

self.button\_right\_shoes = pygame.Rect(self.screen\_width - 530, self.screen\_height // 2 + 100, self.button\_width, self.button\_height)

# Кнопка выхода

self.exit\_button\_rect = pygame.Rect(self.screen\_width // 2 - 100, self.screen\_height - 100, 200, 40)

# Кнопка сохранения

self.save\_button\_rect = pygame.Rect(self.screen\_width // 2 - 100, self.screen\_height - 150, 200, 40)

def draw\_char(self):

base\_x, base\_y = 500, 150

self.app.screen.blit(self.char\_image, (base\_x, base\_y))

self.app.screen.blit(self.hats[self.current\_hat], (base\_x, base\_y))

self.app.screen.blit(self.dresses[self.current\_dress], (base\_x, base\_y))

self.app.screen.blit(self.shoes[self.current\_shoes], (base\_x, base\_y))

def draw\_buttons(self):

button\_color = (200, 200, 200)

button\_hover\_color = (170, 170, 170)

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

def draw\_button(rect, text):

if rect.collidepoint(mouse\_pos):

pygame.draw.rect(self.app.screen, button\_hover\_color, rect)

else:

pygame.draw.rect(self.app.screen, button\_color, rect)

text\_surface = self.app.font.render(text, True, (0, 0, 0))

self.app.screen.blit(text\_surface, (rect.x + 5, rect.y + 5))

draw\_button(self.button\_left\_hat, "<")

draw\_button(self.button\_left\_dress, "<")

draw\_button(self.button\_left\_shoes, "<")

draw\_button(self.button\_right\_hat, ">")

draw\_button(self.button\_right\_dress, ">")

draw\_button(self.button\_right\_shoes, ">")

# Отрисовка кнопки выхода

draw\_button(self.exit\_button\_rect, "Выход")

# Отрисовка кнопки сохранения

draw\_button(self.save\_button\_rect, "Сохранить")

def save\_customization(self):

*"""Сохранить текущую кастомизацию и обновить данные в VirtualVirusApp."""*

customization\_data = {

"hat": self.current\_hat,

"dress": self.current\_dress,

"shoes": self.current\_shoes,

}

with open("customization\_data.json", "w") as f:

json.dump(customization\_data, f)

# Обновление данных в VirtualVirusApp

self.app.current\_hat = self.current\_hat

self.app.current\_dress = self.current\_dress

self.app.current\_shoes = self.current\_shoes

print("Кастомизация сохранена!")

def run(self):

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.MOUSEBUTTONDOWN:

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

if self.button\_left\_hat.collidepoint(mouse\_pos):

self.current\_hat = (self.current\_hat - 1) % len(self.hats)

elif self.button\_left\_dress.collidepoint(mouse\_pos):

self.current\_dress = (self.current\_dress - 1) % len(self.dresses)

elif self.button\_left\_shoes.collidepoint(mouse\_pos):

self.current\_shoes = (self.current\_shoes - 1) % len(self.shoes)

if self.button\_right\_hat.collidepoint(mouse\_pos):

self.current\_hat = (self.current\_hat + 1) % len(self.hats)

elif self.button\_right\_dress.collidepoint(mouse\_pos):

self.current\_dress = (self.current\_dress + 1) % len(self.dresses)

elif self.button\_right\_shoes.collidepoint(mouse\_pos):

self.current\_shoes = (self.current\_shoes + 1) % len(self.shoes)

# Проверка нажатия кнопки выхода

if self.exit\_button\_rect.collidepoint(mouse\_pos):

return # Возвращаемся в основной экран игры

# Проверка нажатия кнопки сохранения

if self.save\_button\_rect.collidepoint(mouse\_pos):

self.save\_customization() # Сохранить кастомизацию

# Отрисовка фона и элементов

self.app.screen.blit(self.background\_image, (0, 0))

self.draw\_char()

self.draw\_buttons()

pygame.display.flip()

def save\_money(money):

*"""Сохраняет количество денег в файл.*

*Записывает переданное количество денег в файл "money.txt".*

*Если файл не существует, он будет создан.*

*Args:*

*money (int): Количество денег, которое нужно сохранить.*

*Returns:*

*None*

*"""*

with open("money.txt", "w") as file:

file.write(str(money))

def load\_money():

*"""Загружает количество денег из файла.*

*Читает количество денег из файла "money.txt". Если файл не найден,*

*возвращает 0.*

*Returns:*

*int: Количество денег, загруженное из файла, или 0, если файл не найден.*

*"""*

try:

with open("money.txt", "r") as file:

money = int(file.read())

return money

except FileNotFoundError:

return 0 # Если файл не найден, возвращаем 0

# Классы для игровых объектов

class Strawberry:

*"""*

*Инициализирует объект клубники.*

*Args:*

*x (int): Начальная координата x.*

*y (int): Начальная координата y.*

*"""*

def \_\_init\_\_(self, x, y):

*"""Перемещает клубнику вниз по экрану."""*

self.image = pygame.image.load('strawberry.png')

self.rect = self.image.get\_rect(topleft=(x, y))

self.visible = True

self.speed = 5

def move(self):

*"""Отображает клубнику на заданной поверхности.*

*Args:*

*surface (pygame.Surface): Поверхность, на которой будет нарисована клубника.*

*"""*

if self.visible:

self.rect.y += self.speed

if self.rect.y > 810:

self.rect.y = 0

self.rect.x = random.randint(0, 1440 - self.rect.width)

def draw(self, surface):

*"""Отображает клубнику на заданной поверхности.*

*Args:*

*surface (pygame.Surface): Поверхность, на которой будет нарисована клубника.*

*"""*

if self.visible:

surface.blit(self.image, self.rect.topleft)

class Mushroom:

*"""*

*Класс, представляющий гриб в игре.*

*Args:*

*x (int): Начальная координата x.*

*y (int): Начальная координата y.*

*"""*

def \_\_init\_\_(self, x, y):

*"""Инициализация объекта гриба и его начального положения."""*

self.image = pygame.image.load('mashroom.jpg')

self.rect = self.image.get\_rect(topleft=(x, y))

self.visible = True

self.speed = 5

def move(self):

*"""Перемещает гриб вниз по экрану.*

*Если гриб выходит за пределы экрана, он возвращается в верхнюю часть с новой случайной координатой x.*

*"""*

if self.visible:

self.rect.y += self.speed

if self.rect.y > screen\_height:

self.rect.y = 0

self.rect.x = random.randint(0, screen\_width - self.rect.width)

def draw(self, surface):

*"""Отображает гриб на заданной поверхности.*

*Args:*

*surface (pygame.Surface): Поверхность, на которой будет нарисован гриб.*

*"""*

if self.visible:

surface.blit(self.image, self.rect.topleft)

class Oclock:

*"""*

*Класс, представляющий часы в игре.*

*Args:*

*x (int): Начальная координата x.*

*y (int): Начальная координата y.*

*"""*

def \_\_init\_\_(self, x, y):

*"""Инициализация объекта часов и его начального положения."""*

self.image = pygame.image.load('oclock.png')

self.rect = self.image.get\_rect(topleft=(x, y))

self.visible = True

self.speed = 1

def move(self):

*"""Перемещает часы вниз по экрану.*

*Если часы выходят за пределы экрана, они возвращаются в верхнюю часть с новой случайной координатой x.*

*"""*

if self.visible:

self.rect.y += self.speed

if self.rect.y > screen\_height:

self.rect.y = 0

self.rect.x = random.randint(0, screen\_width - self.rect.width)

def draw(self, surface):

*"""Отображает часы на заданной поверхности.*

*Args:*

*surface (pygame.Surface): Поверхность, на которой будут нарисованы часы.*

*"""*

if self.visible:

surface.blit(self.image, self.rect.topleft)

class Player:

*"""*

*Класс, представляющий игрока в игре.*

*Этот класс управляет изображением игрока, его положением и движением.*

*"""*

def \_\_init\_\_(self):

*"""Инициализация объекта игрока и его начального положения."""*

self.image = pygame.image.load('player\_down.png')

self.rect = self.image.get\_rect(topleft=(character\_x, character\_y))

self.speed = 10

def move(self):

*"""Перемещает игрока влево или вправо в зависимости от нажатых клавиш."""*

keys = pygame.key.get\_pressed()

if keys[pygame.K\_LEFT]:

self.rect.x -= self.speed

if keys[pygame.K\_RIGHT]:

self.rect.x += self.speed

if self.rect.x < 0:

self.rect.x = 0

if self.rect.x > screen\_width - self.rect.width:

self.rect.x = screen\_width - self.rect.width

def draw(self, surface):

*"""Отображает игрока на заданной поверхности.*

*Args:*

*surface (pygame.Surface): Поверхность, на которой будет нарисован игрок.*

*"""*

surface.blit(self.image, self.rect.topleft)

# Функция для создания новых объектов

def create\_new\_strawberry():

*"""Создает новый объект Strawberry со случайной координатой x.*

*Координата y устанавливается в 0. Координата x генерируется случайным образом*

*в пределах ширины экрана минус ширина объекта Strawberry.*

*Returns:*

*Strawberry: Новый объект Strawberry со случайной координатой x.*

*"""*

return Strawberry(random.randint(0, screen\_width - 30), 0)

def create\_new\_mushroom():

*"""Создает новый объект Mushroom со случайной координатой x.*

*Координата y устанавливается в 0. Координата x генерируется случайным образом*

*в пределах ширины экрана минус ширина объекта Mushroom.*

*Returns:*

*Mushroom: Новый объект Mushroom со случайной координатой x.*

*"""*

return Mushroom(random.randint(0, screen\_width - 30), 0)

def create\_new\_oclock():

*"""Создает новый объект Oclock с случайной координатой x.*

*Координата y устанавливается в 0. Координата x генерируется случайным образом*

*в пределах ширины экрана минус ширина объекта Oclock.*

*Returns:*

*Oclock: Новый объект Oclock с случайной координатой x.*

*"""*

return Oclock(random.randint(0, screen\_width - 30), 0)

# Основная функция игры

def run\_game(app):

*"""Запускает игру "Падающая клубника".*

*Инициализирует Pygame, устанавливает размеры окна, загружает фон и*

*настраивает игровую логику, включая создание объектов, обработку событий,*

*движение персонажа и проверку столкновений. Игра продолжается в течение*

*30 секунд, после чего отображается итоговый счет и возможность вернуться в меню.*

*В процессе игры игрок может собирать клубнику для увеличения счета,*

*избегать столкновений с грибами, которые уменьшают счет, и собирать часы,*

*которые добавляют время.*

*Returns:*

*None*

*"""*

global screen\_width, screen\_height, character\_x, character\_y

pygame.init()

# Установка размера окна

screen\_width, screen\_height = 1440, 810

screen = pygame.display.set\_mode((screen\_width, screen\_height))

# Загрузка и изменение размера фона

background = pygame.image.load('forest.png')

background = pygame.transform.scale(background, (screen\_width, screen\_height)) # Изменяем размер фона

pygame.display.set\_caption("Игра падающая клубника")

# Установка времени

duration = 30000 # 30 секунд в миллисекундах

start\_time = pygame.time.get\_ticks() # Инициализация времени

# Настройка анимации/смены кадров

clock = pygame.time.Clock()

FPS = 30

# Счет

score = 0

score\_screen\_text = pygame.font.Font(None, 74)

BLACK = (0, 0, 0)

# Итог

end\_screen\_text = pygame.font.Font(None, 90)

text\_surface\_menu = score\_screen\_text.render("Обратно в меню", True, (148, 0, 211))

text\_rect\_menu = text\_surface\_menu.get\_rect(center=(screen\_width // 2, screen\_height // 2 + 100))

# Настройка положения и движения

speed = 50

character\_y = screen\_height / 2 + 150

character\_x = (screen\_width / 2) - 10

button\_click\_menu = False # Если кнопка нажата

# Создание объектов

strawberry = create\_new\_strawberry()

mushroom = create\_new\_mushroom()

oclock = create\_new\_oclock()

player = Player()

running = True

while running:

# Обработка событий

clock.tick(FPS)

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

elif event.type == pygame.MOUSEBUTTONDOWN:

if event.button == 1: # Левый клик мыши

if text\_rect\_menu.collidepoint(event.pos): # Проверяем, попадает ли курсор в область текста

button\_click\_menu = True # Устанавливаем состояние нажатия кнопки

if button\_click\_menu:

current\_money = load\_money() # Загружаем деньги из файла

total\_points = score \* 5 + current\_money

save\_money(total\_points) # Сохраняем деньги

app.add\_points(score \* 5) # Добавляем очки к текущему значению

running = False # Выход из игры, чтобы вернуться в меню

run\_menu(app) # Возвращаемся в меню

break

# Движение персонажа

player.move()

# Проверка на столкновение

if player.rect.colliderect(oclock.rect) and oclock.visible:

oclock.visible = False # Скрыть часы при столкновении

duration += 5000

oclock = create\_new\_oclock() # Создать новые часы

if player.rect.colliderect(mushroom.rect) and mushroom.visible:

mushroom.visible = False # Скрыть гриб при столкновении

if score > 0:

score -= 1

mushroom = create\_new\_mushroom() # Создать новую гриб

if player.rect.colliderect(strawberry.rect) and strawberry.visible:

strawberry.visible = False # Скрыть клубнику при столкновении

score += 1

strawberry = create\_new\_strawberry() # Создать новую клубнику

# Проверяем, прошло ли 30 секунд

current\_time = pygame.time.get\_ticks()

screen.fill((0, 0, 0))

# Инициализация текстов

text\_score = score\_screen\_text.render(str(score), True, (0, 0, 0)) # Счет на экране

text\_rect\_score = text\_score.get\_rect(center=(100, 100))

text\_time = score\_screen\_text.render(f'{(duration - (current\_time - start\_time)) // 1000} сек', True, (0, 0, 0)) # Счетчик времени

text\_rect\_time = text\_time.get\_rect(center=(300, 100))

text\_end = score\_screen\_text.render('Время вышло', True, (0, 0, 0)) # Итог

text\_rect\_end = text\_end.get\_rect(center=(screen\_width / 2, screen\_height / 2 - 200))

text\_score\_end = score\_screen\_text.render(f'Ваш счёт: {score \* 5}', True, (0, 0, 0)) # Счёт в конце игры

text\_score\_end = score\_screen\_text.render(f'Ваш счёт: {score \* 5}', True, (0, 0, 0)) # Счёт в конце игры

text\_rect\_score\_end = text\_score\_end.get\_rect(center=(screen\_width / 2, screen\_height / 2 - 100))

# Прорисовка заднего фона

screen.blit(background, (0, 0))

if current\_time - start\_time < duration:

screen.blit(text\_score, text\_rect\_score)

screen.blit(text\_time, text\_rect\_time)

oclock.move()

oclock.draw(screen)

strawberry.move()

strawberry.draw(screen)

mushroom.move()

mushroom.draw(screen)

player.draw(screen)

else:

text\_money = score\_screen\_text.render(f'Количество денег: {str(score \* 5 + load\_money())}', True,

(0, 0, 0)) # Счет на экране

text\_rect\_money = text\_money.get\_rect(center=(screen\_width / 2, screen\_height / 2))

screen.blit(text\_money, text\_rect\_money)

screen.blit(text\_end, text\_rect\_end)

screen.blit(text\_score\_end, text\_rect\_score\_end)

screen.blit(text\_surface\_menu, text\_rect\_menu)

# Сохранение денег при нажатии кнопки

if button\_click\_menu:

current\_money = load\_money() # Загружаем деньги из файла

save\_money(score \* 5 + current\_money)

running = False # Выход из игры, чтобы вернуться в меню

run\_menu(app) # Возвращаемся в меню

break

pygame.display.flip() # Обновляем экран

pygame.quit() # Закрываем Pygame

import pygame

import random

# Инициализация Pygame

pygame.init()

# Устанавливаем размеры окна и плиток

width, height = 400, 400

tile\_size = width // 4

fps = 30

font\_plays = pygame.font.Font(None, 30) # Используем стандартный шрифт

text\_surface\_menu = font\_plays.render("Меню", True, (148, 0, 211)) # Создаем поверхность текста

text\_rect\_menu = text\_surface\_menu.get\_rect(center=(35, 13)) # Получаем прямоугольник текста

# Определяем цвета

white = (255, 255, 255)

black = (0, 0, 0)

gray = (200, 200, 200)

green = (1, 55, 32)

# Функция для создания доски

def create\_board():

*"""Создает игровое поле для игры в 15.*

*Генерирует случайное расположение 15 плиток и одной пустой клетки.*

*Плитки представлены числами от 1 до 15, а пустая клетка обозначается нулем.*

*Возвращает двумерный список, представляющий игровое поле с 4 строками.*

*Returns:*

*list: Двумерный список размером 4x4, содержащий числа от 1 до 15 и 0.*

*"""*

numbers = list(range(1, 16)) + [0] # 15 плиток и одна пустая клетка

random.shuffle(numbers)

return [numbers[i:i + 4] for i in range(0, 16, 4)] # Создаем 4 строки

# Функция для отрисовки доски

def draw\_board(board):

*"""Отображает игровое поле на экране.*

*Рисует 4 строки и 4 столбца плиток на экране, представляя игровое поле*

*для игры в пятнашки. Плитки отображаются в сером цвете, а пустая клетка — в белом.*

*Числа на плитках отображаются черным цветом.*

*Args:*

*board (list): Двумерный список размером 4x4, содержащий числа от 1 до 15 и 0,*

*где 0 представляет пустую клетку.*

*Returns:*

*None*

*"""*

for i in range(4): # 4 строки

for j in range(4): # 4 столбца

number = board[i][j]

rect = pygame.Rect(j \* tile\_size, i \* tile\_size, tile\_size, tile\_size)

if number == 0:

pygame.draw.rect(screen, white, rect) # Пустая клетка

else:

pygame.draw.rect(screen, gray, rect) # Плитка

font = pygame.font.Font(None, 74)

text = font.render(str(number), True, black)

screen.blit(text, (j \* tile\_size + tile\_size // 4, i \* tile\_size + tile\_size // 4))

pygame.draw.rect(screen, black, rect, 1) # Граница плитки

# Функция для перемещения плитки

def move\_tile(board, row, col):

*"""Перемещает плитку на игровом поле.*

*Если плитка, находящаяся в позиции (row, col), соседствует с пустой клеткой,*

*плитка перемещается на место пустой клетки. Функция обновляет состояние*

*игрового поля и возвращает True, если перемещение успешно, и False в противном случае.*

*Args:*

*board (list): Двумерный список размером 4x4, представляющий игровое поле,*

*где 0 обозначает пустую клетку.*

*row (int): Индекс строки плитки, которую нужно переместить.*

*col (int): Индекс столбца плитки, которую нужно переместить.*

*Returns:*

*bool: True, если перемещение успешно, иначе False.*

*"""*

empty\_row, empty\_col = find\_empty\_tile(board)

if (abs(empty\_row - row) == 1 and empty\_col == col) or (abs(empty\_col - col) == 1 and empty\_row == row):

board[empty\_row][empty\_col], board[row][col] = board[row][col], board[empty\_row][empty\_col]

return True

return False

# Функция для нахождения пустой плитки

def find\_empty\_tile(board):

*"""Находит координаты пустой клетки на игровом поле.*

*Ищет пустую клетку, обозначенную нулем, в двумерном списке, представляющем*

*игровое поле. Возвращает индексы строки и столбца пустой клетки. Если пустая*

*клетка не найдена, возвращает None (что не должно происходить, если доска корректна).*

*Args:*

*board (list): Двумерный список размером 4x4, представляющий игровое поле,*

*где 0 обозначает пустую клетку.*

*Returns:*

*tuple: Кортеж из двух целых чисел (row, col), представляющий координаты*

*пустой клетки, или None, если пустая клетка не найдена.*

*"""*

for i in range(4):

for j in range(4):

if board[i][j] == 0:

return i, j

return None # Это не должно происходить, если доска корректна

# Функция для проверки выигрыша

def check\_win(board):

*"""Проверяет, выиграл ли игрок.*

*Сравнивает текущее состояние игрового поля с правильным порядком плиток.*

*Если плитки расположены в порядке от 1 до 15, а пустая клетка находится*

*в последнем месте, функция возвращает True. В противном случае возвращает False.*

*Args:*

*board (list): Двумерный список размером 4x4, представляющий игровое поле,*

*где 0 обозначает пустую клетку.*

*Returns:*

*bool: True, если игрок выиграл (плитки расположены в правильном порядке),*

*иначе False.*

*"""*

correct\_order = list(range(1, 16)) + [0]

flat\_board = [num for row in board for num in row]

return flat\_board == correct\_order

# Основная функция игры

def run\_game\_pyatnashki(app):

*"""Запускает игру "Пятнашки".*

*Инициализирует Pygame, создает игровое окно и запускает основной игровой цикл.*

*Обрабатывает события, такие как нажатия мыши для перемещения плиток. Проверяет,*

*выиграл ли игрок, и отображает сообщение о победе. Позволяет вернуться в меню*

*и сохраняет деньги при выигрыше.*

*Returns:*

*None*

*"""*

global screen # Делаем экран глобальным, чтобы использовать его в других функциях

screen = pygame.display.set\_mode((width, height))

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

clock = pygame.time.Clock()

board = create\_board()

running = True

button\_pressed\_menu = False

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

elif event.type == pygame.MOUSEBUTTONDOWN:

x, y = event.pos

col, row = x // tile\_size, y // tile\_size

if 0 <= col < 4 and 0 <= row < 4: # Проверяем, что клик внутри границ

move\_tile(board, row, col)

if event.button == 1: # Левый клик мыши

if text\_rect\_menu.collidepoint(event.pos): # Проверяем, попадает ли курсор в область текста

button\_pressed\_menu = True

if button\_pressed\_menu:

if check\_win(board):

save\_money(load\_money() + 300)

app.add\_points(300) # Добавляем 300 очков к текущему значению

running = False

run\_menu(app)

if check\_win(board):

win\_message = font\_plays.render("Вы выиграли!", True, green)

win\_rect = win\_message.get\_rect(center=(width // 2, height // 2))

screen.blit(win\_message, win\_rect)

pygame.display.flip()

pygame.time.wait(2000) # Ждем 2 секунды перед перезапуском игры

board = create\_board() # Перезапускаем игру

if button\_pressed\_menu:

if check\_win(board):

save\_money(load\_money() + 300)

running = False

run\_menu(app)

# Отрисовка доски и меню

screen.fill(white)

draw\_board(board)

screen.blit(text\_surface\_menu, text\_rect\_menu) # Отрисовываем текст меню

pygame.display.flip() # Обновляем экран

clock.tick(fps) # Ограничиваем FPS

pygame.quit() # Закрываем Pygame

def run\_menu(app):

*"""Запускает главное меню игры.*

*Инициализирует Pygame, создает игровое окно и отображает меню с доступными играми.*

*Позволяет игроку перемещать персонажа по экрану, а также выбирать игры "Падающая клубника"*

*и "Пятнашки". Обрабатывает события мыши и клавиатуры, отображает текущее количество денег в меню.*

*Также присутствует кнопка домой и меню.*

*Returns:*

*None*

*"""*

pygame.init()

screen = pygame.display.set\_mode((1440, 810)) # Устанавливаем размер экрана

background = pygame.image.load('home.jpg')

background = pygame.transform.scale(background, (1440, 810)) # Изменяем размер фона

pygame.display.set\_caption('Меню игр')

screen\_width, screen\_height = 1440, 810

# Настройка анимации/смены кадров

clock = pygame.time.Clock()

FPS = 30

money = load\_money() # Выгружаем деньги из файла

# Загрузка изображений для каждого направления игрока

image\_left = pygame.image.load("player\_left.png")

image\_right = pygame.image.load("player\_right.png")

image\_up = pygame.image.load("player\_up.png")

image\_down = pygame.image.load("player\_down.png")

current\_image = image\_down # Начальное изображение

player\_rect = current\_image.get\_rect(center=(screen\_width // 2, screen\_height // 2)) # Начальная позиция

# Создание кнопки входа в меню

button\_rect\_menu = pygame.Rect(50, -30, 82, 120) # Позиция и размер кнопки

button\_hover\_color\_menu = (0, 0, 0, 255) # Полупрозрачный зеленый

# Кнопка домой, создаем прямоугольник кнопки

button\_rect\_home = pygame.Rect(10, 650, 200, 50)

# Тексты

font\_plays = pygame.font.Font(None, 50) # Используем стандартный шрифт

text\_surface\_strawberry = font\_plays.render("Падающая клубника", True, (148, 0, 211))

text\_rect\_strawberry = text\_surface\_strawberry.get\_rect(center=(screen\_width // 2, screen\_height // 2 - 100))

text\_surface\_5 = font\_plays.render("Пятнашки", True, (148, 0, 211))

text\_rect\_5 = text\_surface\_5.get\_rect(center=(screen\_width // 2, screen\_height // 2))

text\_surface\_money = font\_plays.render(f'Деньги: {str(money)}', True, (148, 0, 211))

text\_rect\_money = text\_surface\_money.get\_rect(center=(150, 50))

# Переменная для хранения состояния

button\_pressed\_strawberry = False

button\_pressed\_5 = False

button\_pressed\_home = False

# Шрифт

big\_screen\_text = pygame.font.Font(None, 90)

font\_home = pygame.font.Font(None, 36)

# Настройка положения и движения

speed = 25

running = True

button\_click\_menu = False # Если кнопка нажата

while running:

# Обработка событий

clock.tick(FPS)

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

elif event.type == pygame.MOUSEBUTTONDOWN:

if button\_rect\_menu.collidepoint(event.pos): # Проверяем, нажата ли кнопка

button\_click\_menu = True

if event.button == 1: # Левый клик мыши

if text\_rect\_strawberry.collidepoint(event.pos): # Проверяем, попадает ли курсор в область текста

button\_pressed\_strawberry = True # Устанавливаем состояние нажатия кнопки

if text\_rect\_5.collidepoint(event.pos): # Проверяем, попадает ли курсор в область текста

button\_pressed\_5 = True # Устанавливаем состояние нажатия кнопки

if button\_rect\_home.collidepoint(event.pos): # Проверяем, попадает ли курсор в область кнопки

button\_pressed\_home = True

if button\_pressed\_home:

# Здесь вы можете добавить логику для сохранения текущих очков

running = False

app.show\_main\_screen() # Возвращаемся на главный экран

# Получение нажатых клавиш

keys = pygame.key.get\_pressed()

if keys[pygame.K\_LEFT]:

player\_rect.x -= speed

current\_image = image\_left # Изменение изображения при движении влево

elif keys[pygame.K\_RIGHT]:

player\_rect.x += speed

current\_image = image\_right # Изменение изображения при движении вправо

elif keys[pygame.K\_UP]:

player\_rect.y -= speed

current\_image = image\_up # Изменение изображения при движении вверх

elif keys[pygame.K\_DOWN]:

player\_rect.y += speed

current\_image = image\_down # Изменение изображения при движении вниз

else:

current\_image = image\_down # Если ничего не нажато, показываем изображение вниз (или любое другое)

# Ограничение движения по границам экрана

if player\_rect.left < 0:

player\_rect.left = 0

if player\_rect.right > screen\_width:

player\_rect.right = screen\_width

if player\_rect.top < 80:

player\_rect.top = 80

if player\_rect.bottom > screen\_height:

player\_rect.bottom = screen\_height

# Получаем позицию мыши

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

# Инициализация текстов

text\_play = big\_screen\_text.render('Игры', True, (148, 0, 211)) # Текст на экране

text\_rect\_play = text\_play.get\_rect(center=(screen\_width / 2, screen\_height / 2 - 200))

if not button\_click\_menu:

# Изменяем цвет кнопки при наведении

if button\_rect\_menu.collidepoint(mouse\_pos):

button\_color\_menu = button\_hover\_color\_menu

else:

button\_color\_menu = (0, 0, 0, 0) # Полупрозрачный серый

# Кнопка домой

if button\_pressed\_home:

pass

# Создаем поверхность для кнопки с альфа-каналом

button\_surface\_menu = pygame.Surface(button\_rect\_menu.size, pygame.SRCALPHA)

button\_surface\_menu.fill(button\_color\_menu) # Заполняем поверхность цветом с альфа-каналом

# Создаем текст

text\_surface\_home = font\_home.render("Домой", True, (0, 0, 0))

text\_rect\_home = text\_surface\_home.get\_rect(center=button\_rect\_home.center)

screen.fill((0, 0, 0))

screen.blit(background, (0, 0))

screen.blit(current\_image, player\_rect) # Отрисовка текущего персонажа

screen.blit(button\_surface\_menu, button\_rect\_menu)

# Отрисовываем кнопку домой

pygame.draw.rect(screen, (150, 75, 0), button\_rect\_home)

pygame.draw.rect(screen, (0, 0, 0), button\_rect\_home, 2) # Граница кнопки

screen.blit(text\_surface\_home, text\_rect\_home)

pygame.display.flip()

else:

screen.fill((91, 147, 226))

screen.blit(text\_surface\_5, text\_rect\_5)

screen.blit(text\_surface\_strawberry, text\_rect\_strawberry)

screen.blit(text\_surface\_money, text\_rect\_money)

screen.blit(text\_play, text\_rect\_play)

# Здесь можно добавить логику для обработки нажатий на кнопки "Падающая клубника" и "Пятнашки"

if button\_pressed\_strawberry:

# Логика для запуска игры "Падающая клубника"

run\_game(app)

if button\_pressed\_5:

# Логика для запуска игры "Пятнашки"

run\_game\_pyatnashki(app)

pygame.display.flip()

pygame.quit()

# Запуск приложения

if \_\_name\_\_ == "\_\_main\_\_":

VirtualVirusApp()

import pygame

import json

class CustomizationScreen:

"""Экран кастомизации персонажа.

Этот класс отвечает за отображение экрана кастомизации персонажа в игре,

включая выбор головных уборов, одежды и обуви.

Атрибуты:

current\_hat (int): Индекс текущего головного убора.

current\_dress (int): Индекс текущей одежды.

current\_shoes (int): Индекс текущей обуви.

app (VirtualVirusApp): Ссылка на основной объект приложения.

screen\_width (int): Ширина экрана.

screen\_height (int): Высота экрана.

char\_image (Surface): Изображение персонажа.

background\_image (Surface): Фоновое изображение.

hats (list): Список изображений головных уборов.

dresses (list): Список изображений одежды.

shoes (list): Список изображений обуви.

button\_width (int): Ширина кнопок.

button\_height (int): Высота кнопок.

button\_left\_hat (Rect): Прямоугольник кнопки для выбора головного убора (влево).

button\_left\_dress (Rect): Прямоугольник кнопки для выбора одежды (влево).

button\_left\_shoes (Rect): Прямоугольник кнопки для выбора обуви (влево).

button\_right\_hat (Rect): Прямоугольник кнопки для выбора головного убора (вправо).

button\_right\_dress (Rect): Прямоугольник кнопки для выбора одежды (вправо).

button\_right\_shoes (Rect): Прямоугольник кнопки для выбора обуви (вправо).

exit\_button\_rect (Rect): Прямоугольник кнопки выхода.

save\_button\_rect (Rect): Прямоугольник кнопки сохранения.

"""

def \_\_init\_\_(self, app):

"""Инициализация экрана кастомизации.

Args:

app (VirtualVirusApp): Ссылка на основной объект приложения.

"""

# Устанавливаем текущую кастомизацию из VirtualVirusApp

self.current\_hat = app.current\_hat

self.current\_dress = app.current\_dress

self.current\_shoes = app.current\_shoes

self.app = app

self.screen\_width, self.screen\_height = 1440, 810

self.char\_image = pygame.image.load('assets/char.png')

self.background\_image = pygame.image.load('assets/backgr.png')

self.right\_arrow = pygame.image.load('assets/right\_arrow.png')

self.left\_arrow = pygame.image.load('assets/left\_arrow.png')

self.background\_image = pygame.transform.scale(self.background\_image, (self.screen\_width, self.screen\_height))

self.hats = [

pygame.image.load('assets/hat.png'),

pygame.image.load('assets/cap.png'),

pygame.image.load('assets/beret.png'),

pygame.image.load('assets/glasses.png'),

pygame.image.load('assets/shapka.png'),

pygame.image.load('assets/empty.png'),

]

self.dresses = [

pygame.image.load('assets/dress.png'),

pygame.image.load('assets/kostum.png'),

pygame.image.load('assets/pijama.png'),

pygame.image.load('assets/t-shirt.png'),

pygame.image.load('assets/uniform.png'),

pygame.image.load('assets/empty.png'),

]

self.shoes = [

pygame.image.load('assets/shoes.png'),

pygame.image.load('assets/boots.png'),

pygame.image.load('assets/kedy.png'),

pygame.image.load('assets/snickers.png'),

pygame.image.load('assets/slippers.png'),

pygame.image.load('assets/empty.png'),

]

self.current\_hat = 0

self.current\_dress = 0

self.current\_shoes = 0

self.char\_image = pygame.transform.scale(self.char\_image, (400, 600))

self.hats = [pygame.transform.scale(hat, (400, 600)) for hat in self.hats]

self.dresses = [pygame.transform.scale(dress, (400, 600)) for dress in self.dresses]

self.shoes = [pygame.transform.scale(shoe, (400, 600)) for shoe in self.shoes]

self.button\_width = 80

self.button\_height = 80

self.button\_left\_hat = pygame.Rect(400, self.screen\_height // 2 - 200, self.button\_width, self.button\_height)

self.button\_left\_dress = pygame.Rect(400, self.screen\_height // 2 - 50, self.button\_width, self.button\_height)

self.button\_left\_shoes = pygame.Rect(400, self.screen\_height // 2 + 100, self.button\_width, self.button\_height)

self.button\_right\_hat = pygame.Rect(self.screen\_width - 530, self.screen\_height // 2 - 200, self.button\_width, self.button\_height)

self.button\_right\_dress = pygame.Rect(self.screen\_width - 530, self.screen\_height // 2 - 50, self.button\_width, self.button\_height)

self.button\_right\_shoes = pygame.Rect(self.screen\_width - 530, self.screen\_height // 2 + 100, self.button\_width, self.button\_height)

# Кнопка выхода

self.exit\_button\_rect = pygame.Rect(self.screen\_width // 2 - 100, self.screen\_height - 100, 200, 40)

# Кнопка сохранения

self.save\_button\_rect = pygame.Rect(self.screen\_width // 2 - 100, self.screen\_height - 150, 200, 40)

def draw\_char(self):

"""Отрисовать персонажа с текущей кастомизацией.

Отображает изображение персонажа вместе с выбранными головным убором,

одеждой и обувью на экране.

"""

base\_x, base\_y = 500, 150

self.app.screen.blit(self.char\_image, (base\_x, base\_y))

self.app.screen.blit(self.hats[self.current\_hat], (base\_x, base\_y))

self.app.screen.blit(self.dresses[self.current\_dress], (base\_x, base\_y))

self.app.screen.blit(self.shoes[self.current\_shoes], (base\_x, base\_y))

def draw\_buttons(self):

"""Отрисовать кнопки на экране кастомизации.

Отображает кнопки для выбора головных уборов, одежды, обуви, а также

кнопки выхода и сохранения. Обрабатывает изменение цвета кнопок при наведении.

"""

button\_color = (200, 200, 200)

button\_hover\_color = (170, 170, 170)

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

def draw\_button(rect, text):

"""Отрисовать кнопку.

Args:

rect (Rect): Прямоугольник кнопки.

text (str): Текст на кнопке.

"""

if rect.collidepoint(mouse\_pos):

pygame.draw.rect(self.app.screen, button\_hover\_color, rect)

else:

pygame.draw.rect(self.app.screen, button\_color, rect)

text\_surface = self.app.font.render(text, True, (0, 0, 0))

self.app.screen.blit(text\_surface, (rect.x + 5, rect.y + 5))

draw\_button(self.button\_left\_hat, "<")

draw\_button(self.button\_left\_dress, "<")

draw\_button(self.button\_left\_shoes, "<")

draw\_button(self.button\_right\_hat, ">")

draw\_button(self.button\_right\_dress, ">")

draw\_button(self.button\_right\_shoes, ">")

# Отрисовка кнопки выхода

draw\_button(self.exit\_button\_rect, "Выход")

# Отрисовка кнопки сохранения

draw\_button(self.save\_button\_rect, "Сохранить")

def save\_customization(self):

"""Сохранить текущую кастомизацию и обновить данные в VirtualVirusApp.

Сохраняет текущие настройки кастомизации (головной убор, одежда, обувь)

в файл `customization\_data.json` и обновляет данные в основном приложении.

"""

customization\_data = {

"hat": self.current\_hat,

"dress": self.current\_dress,

"shoes": self.current\_shoes,

}

with open("customization\_data.json", "w") as f:

json.dump(customization\_data, f)

# Обновление данных в VirtualVirusApp

self.app.current\_hat = self.current\_hat

self.app.current\_dress = self.current\_dress

self.app.current\_shoes = self.current\_shoes

print("Кастомизация сохранена!")