Объяснение кода игры на Pygame

Инициализация Pygame и настройка экрана

```
pygame.init()
if is_fullscreen:
    screen = pygame.display.set_mode(
        (screen_width, screen_height),
        pygame.FULLSCREEN | pygame.SCALED
    )
else:
    screen = pygame.display.set_mode((screen_width, screen_height))
clock = pygame.time.Clock()
```

Разбор

1. **`pygame.init()`** — инициализирует все модули Pygame. Это нужно делать в начале программы, чтобы подготовить библиотеку к работе.
2. **`pygame.display.set_mode()`** — создает окно для отображения игры. Если игра запускается в полноэкранном режиме, используются параметры `FULLSCREEN` и `SCALED`. Если нет, устанавливается стандартный размер окна (ширина и высота экрана).
3. **`pygame.time.Clock()`** — создается объект `Clock`, который помогает контролировать частоту обновления кадров игры. Например, можно установить, чтобы игра работала на скорости 60 кадров в секунду.

Пример:

```
screen = pygame.display.set_mode((800, 600))
```

Этот код создает окно размером 800x600 пикселей.

Загрузка фона и создание персонажей

```
bg = pygame.image.load(bg_path)
bg = pygame.transform.scale(bg, (screen_width, screen_height))
player = PlayerKapibara(
    img_path=player_img_path,
    player_x=0,
    player_y=screen_height
)
npc1 = NPC(
    img_path=npc1_img_path,
    player_x=screen_width - 100,
    player_y=screen_height
)
```

Разбор

1. **`pygame.image.load()`** — загружает изображение из файла, например, для фона или персонажей.
2. **`pygame.transform.scale()`** — изменяет размер изображения до заданных ширины и высоты.
3. **Создание объектов** — мы создаем объект `PlayerKapibara`, который представляет игрока, и `NPC`, представляющего врага.

Пример:

```
bg = pygame.image.load('background.png')
bg = pygame.transform.scale(bg, (800, 600))
```

Этот код загружает изображение фона и изменяет его размер до 800x600 пикселей.

Основной цикл игры

```
while is_game_running:
    dt = clock.tick(60) / 1000
    game_frame_number += 1
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            is_game_running = False

    # Обновление и отрисовка объектов
    screen.blit(bg, (0, 0))
    player_group.update(dt, bullets_group, walls_group)
    player_group.draw(screen)
    npc_group.update()
    npc_group.draw(screen)
    walls_group.update()
    walls_group.draw(screen)
    bullets_group.update(dt)
    bullets_group.draw(screen)

    pygame.display.flip()
```

Разбор

1. **Основной цикл игры**: Цикл `while` работает до тех пор, пока переменная `is_game_running` равна `True`. Это позволяет игре работать непрерывно.
2. **`clock.tick(60)`** — задает максимальную скорость игры в 60 кадров в секунду. Деление на 1000 переводит значение в секунды, чтобы использовать его для обновлений (например, движения персонажа).
3. **`pygame.event.get()`** — проверяет события (например, нажатие клавиш или закрытие окна). Если событие `QUIT`, игра завершается.
4. **`screen.blit()`** — рисует фон на экране.
5. **Группы спрайтов**: `update()` обновляет положение спрайтов, а `draw()` рисует их на экране.
6. **`pygame.display.flip()`** — обновляет весь экран, показывая новые кадры игры.

Пример:

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
```

Этот код завершает игру, если игрок закрывает окно.

Завершение игры

```
pygame.quit()
```

Разбор

- **`pygame.quit()`** — завершает работу Pygame и закрывает окно игры. Это необходимо для корректного завершения программы.

Объяснение вспомогательных классов и файла настроек

Класс `PlayerKapibara`

```
class PlayerKapibara(pygame.sprite.Sprite):
    def __init__(self, img_path, player_x, player_y):
        super().__init__()
        self.image = pygame.image.load(img_path).convert_alpha()
        self.image = pygame.transform.scale(self.image, (50, 100))
        self.rect = self.image.get_rect()
        self.rect.x = player_x
        self.rect.y = player_y - self.rect.height
```

Разбор

1. **Наследование от `pygame.sprite.Sprite`**: Класс `PlayerKapibara` наследует от встроенного класса `Sprite` в Pygame, что позволяет легко работать с персонажем как с игровым объектом.
2. **`convert_alpha()`** — конвертирует изображение с учетом прозрачности (альфа-канала), что полезно для изображений с прозрачными фонами.
3. **`get_rect()`** — создает прямоугольник (rect) вокруг изображения, который используется для позиционирования и столкновений.

Обновление персонажа

```
def update(self, dt, bullets_group, walls_group):
    self.keys = pygame.key.get_pressed()
    self.change_y += gravity
    self.rect.x += self.change_x
    self.rect.y += self.change_y
```

Разбор

- `pygame.key.get_pressed()` — проверяет, какие клавиши нажаты, и обновляет состояние персонажа в зависимости от нажатых клавиш.
- **Обновление координат**: `self.rect.x` и `self.rect.y` обновляются на основе изменения положения персонажа (например, движение и прыжки).

Пример:

```
keys = pygame.key.get_pressed()
if keys[pygame.K_LEFT]:
    player.rect.x -= 5
```

Этот код перемещает игрока влево, если нажата клавиша "влево".

Класс Bullet (Пуля)

Класс `Bullet` представляет собой игровую пулю, которая летит в определенном направлении с заданной скоростью. Этот класс также управляет тем, что происходит с пулей, если она выходит за границы экрана.

```
class Bullet(pygame.sprite.Sprite):
    def __init__(self, position, direction):
        super().__init__()
        self.image = bullet_img
        self.rect = self.image.get_rect()
        self.position = pygame.math.Vector2(position)
        self.speed = direction * bullet_speed
        self.damage = 10

    def update(self, dt):
        self.position += self.speed * dt
        self.rect.center = self.position
        if self.rect.bottom < 0 or self.rect.top > screen_height or self.rect.l
            self.kill()
```

Разбор

1. **Наследование от `pygame.sprite.Sprite`**: Класс `Bullet` наследует от встроенного класса `Sprite` в Pygame, что делает его игровым объектом.
2. `pygame.Surface()`: Создает поверхность для пули размером 10x10 пикселей. Поверхность — это прямоугольная область для рисования графики в Pygame.
3. `pygame.math.Vector2()`: Используется для работы с двумя координатами (например, положение и скорость пули) и облегчает вычисления с векторами.
4. `self.kill()`: Удаляет пулю из игры, если она выходит за пределы экрана.

Пример создания пули:

```
bullet = Bullet(position=(100, 200), direction=pygame.math.Vector2(1, 0))
```

Этот код создаст пулю, которая летит вправо от позиции (100, 200).

Файл настроек `game_settings.py`

Файл настроек содержит основные параметры игры, такие как размеры экрана, гравитация и скорость пули. Эти настройки влияют на поведение всех объектов в игре.

```
screen_height = 800
screen_width = 1280
gravity = 0.5
bullet_speed = 500
is_fullscreen = True
player_speed = 300
```

Что здесь описано?

1. **`screen_height` и `screen_width`**: Высота и ширина экрана. Эти переменные используются для установки размеров окна игры и контроля за тем, чтобы объекты не выходили за его пределы.
   - Пример: `screen = pygame.display.set_mode((screen_width, screen_height))` — создает окно размером 1280x800 пикселей.
2. **`gravity`**: Сила гравитации, которая влияет на все объекты в игре, например, на прыжки персонажа или падение объектов. Чем больше значение, тем сильнее воздействие гравитации.
   - Пример: `self.change_y += gravity` — увеличивает вертикальную скорость персонажа, заставляя его падать вниз.
3. **`bullet_speed`**: Скорость пули. Это значение умножается на направление пули, чтобы задать ее движение в нужную сторону.
   - Пример: `self.speed = direction * bullet_speed` — устанавливает скорость пули.
4. **`is_fullscreen`**: Переменная, которая определяет, должна ли игра запускаться в полноэкранном режиме.
   - Пример: `if is_fullscreen: screen = pygame.display.set_mode((screen_width, screen_height), pygame.FULLSCREEN)` — запускает игру в полноэкранном режиме, если `is_fullscreen` равно `True`.
5. **`player_speed`**: Скорость игрока, которая используется для перемещения персонажа по экрану.
   - Пример: `dx += player_speed * dt` — перемещает персонажа в зависимости от скорости и времени, прошедшего с последнего кадра.

Вывод текста на экран

```
font = pygame.font.SysFont('Comic Sans MS', 30)
text_surface = font.render(f'dt::{dt}::{player.rect.center}', 1, (0, 255, 0))
screen.blit(text_surface, player.rect.center)
```

Разбор

1. **`pygame.font.SysFont()`** — создает объект шрифта, который будет использоваться для вывода текста на экран.

2. `render()` — отрисовывает текст на поверхности. В данном примере текст выводится в зеленом цвете.
3. `blit()` — рисует текст на экране в указанной позиции.

Пример:

```
font = pygame.font.SysFont('Arial', 24)
text_surface = font.render('Hello, World!', True, (255, 255, 255))
screen.blit(text_surface, (50, 50))
```

Этот код выводит текст "Hello, World!" на экране в белом цвете.

Основные внутренние функции и классы:

- **Класс `Bullet`** — управляет пулей в игре, задает ее движение и удаляет, если она выходит за пределы экрана.
- **Файл настроек `game_settings.py`** — хранит основные параметры игры, такие как размер экрана, гравитация, скорость пули и другие настройки, которые влияют на поведение всех игровых объектов.

Основные функции Pygame, которые используются:

- `pygame.init()` — инициализирует Pygame.
- `pygame.display.set_mode()` — создает окно для игры.
- `pygame.time.Clock()` — управляет скоростью игры.
- `pygame.sprite.Sprite` — базовый класс для всех игровых объектов (спрайтов).
- `pygame.image.load()` и `pygame.transform.scale()` — загрузка и изменение размера изображений.
- `pygame.key.get_pressed()` — проверка нажатия клавиш.
- `pygame.font.SysFont()` — создание шрифта для вывода текста.
- `pygame.display.flip()` — обновление экрана.
- `pygame.Surface()` — создает поверхность для рисования объектов (например, пули).
- `pygame.math.Vector2()` — упрощает работу с векторами, например, для расчета движения объектов.
- `self.kill()` — удаляет объект из игры, если он больше не нужен или вышел за пределы экрана.