Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

ОТЧЁТ

по лабораторной работе №3
по дисциплине “Проектирование программного обеспечения в интеллектуальных системах”

Выполнили:
Р. В. Липский, гр. 121701
Проверил:
С. В. Бутрин

Цель и постановка задачи

Цель: Изучить событий-ориентированное программирование с использованием библиотеки pygame

Задание:
Разработать игровое приложение согласно выбранному варианту. При разработке игры необходимо изучить функциональность оригинальной игры и по умолчанию реализовывать правила оригинальной игры, если нет ограничивающих требований в условиях задания.

Вариант 3. Super Mario Bros

1. Карту уровня хранить в файле
2. Карта в игре должна бы не менее 10 экранов

Main

```
from game import Game

if __name__ == "__main__":
   puperlariosis = Game()
   puperlariosis.main_loop()
```

Этот код представляет пример использования класса Game для запуска игры. В первой строке мы импортируем класс Game из модуля game. Затем мы проверяем, запущен ли скрипт непосредственно из командной строки с помощью конструкции if name == "main":. Если это так, то мы создаем экземпляр класса Game и вызываем его метод main_loop(), который запускает главный цикл игры.

Game

```
class Game:

   def __init__(self):
       environ[SDL_VIDEO_CENTERED] = "1"
       pygame.mixer.pre_init(44100, -16, 2, 1024)
       pygame.init()

       pygame.display.set_caption("Super Lario Sisters")
       pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))

       self.screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
       self.clock = pygame.time.Clock()

       self.world = World("1-1")
```

```
        self.sound_manager = SoundManager()
        self.menu_manager = MenuManager(self)

        self.is_running = True
        self.keyR = False
        self.keyL = False
        self.keyU = False
        self.keyD = False
        self.keyShift = False

    def main_loop(self):
        while self.is_running:
            self.input()
            self.update()
            self.render()
            self.clock.tick(FPS)

    def input(self):
        if self.menu_manager.current_state == STATE_GAME:
            self.game_input()
        else:
            self.menu_input()

    def game_input(self):
        for e in pygame.event.get():

            if e.type == pygame.QUIT:
                self.is_running = False

            elif e.type == pygame.locals.KEYDOWN:
                match e.key:
                    case pygame.locals.K_RIGHT:
                        self.keyR = True
                    case pygame.locals.K_LEFT:
                        self.keyL = True
                    case pygame.locals.K_DOWN:
                        self.keyD = True
                    case pygame.locals.K_UP:
                        self.keyU = True
                    case pygame.locals.K_LSHIFT:
                        self.keyShift = True

            elif e.type == pygame.locals.KEYUP:
                match e.key:
                    case pygame.locals.K_RIGHT:
                        self.keyR = False
                    case pygame.locals.K_LEFT:
                        self.keyL = False
                    case pygame.locals.K_DOWN:
                        self.keyD = False
                    case pygame.locals.K_UP:
                        self.keyU = False
                    case pygame.locals.K_LSHIFT:
```

```
                    self.keyShift = False

    def menu_input(self):
        for e in pygame.event.get():
            if e.type == pygame.QUIT:
                self.is_running = False

            elif e.type == pygame.locals.KEYDOWN:
                if e.key == pygame.locals.K_RETURN:
                    self.menu_manager.start_loading()

    def update(self):
        self.menu_manager.update(self)

    def render(self):
        self.menu_manager.render(self)
```

Этот код представляет класс Game.

В методе init() класса Game происходит инициализация необходимых библиотек и создание объектов, которые будут использоваться в игре. В частности, устанавливается параметр для отображения окна игры в центре экрана, происходит предварительная инициализация звуковой подсистемы, устанавливается заголовок окна и размеры экрана. Также создаются объекты World, SoundManager и MenuManager.

Метод main_loop() представляет главный цикл игры. В нем происходит обработка пользовательского ввода, обновление состояния игры и отрисовка игровых объектов на экране. Цикл будет выполняться до тех пор, пока значение флага is_running не станет False.

Метод input() обрабатывает пользовательский ввод в зависимости от текущего состояния игры, которое определяется значением свойства current_state объекта MenuManager.

Методы game_input() и menu_input() обрабатывают пользовательский ввод в зависимости от текущего состояния игры.

Метод update() вызывает метод update() объекта MenuManager.

Метод render() вызывает метод render() объекта MenuManager для отрисовки игровых объектов на экране.

SoundManager

```
class SoundManager:

    def __init__(self):
        self.sounds: dict[str, pygame.mixer.Sound] = {}
```

```
        self.load_sounds()

    def load_sounds(self):
        self.sounds['overworld'] = pygame.mixer.Sound('sounds/overworld.wav')
        self.sounds['overworld_fast'] =
pygame.mixer.Sound('sounds/overworld-fast.wav')
        self.sounds['level_end'] = pygame.mixer.Sound('sounds/levelend.wav')
        self.sounds['coin'] = pygame.mixer.Sound('sounds/coin.wav')
        self.sounds['small_mario_jump'] =
pygame.mixer.Sound('sounds/jump.wav')
        self.sounds['big_mario_jump'] =
pygame.mixer.Sound('sounds/jumpbig.wav')
        self.sounds['brick_break'] =
pygame.mixer.Sound('sounds/blockbreak.wav')
        self.sounds['block_hit'] = pygame.mixer.Sound('sounds/blockhit.wav')
        self.sounds['mushroom_appear'] =
pygame.mixer.Sound('sounds/mushroomappear.wav')
        self.sounds['mushroom_eat'] =
pygame.mixer.Sound('sounds/mushroomeat.wav')
        self.sounds['death'] = pygame.mixer.Sound('sounds/death.wav')
        self.sounds['pipe'] = pygame.mixer.Sound('sounds/pipe.wav')
        self.sounds['kill_mob'] = pygame.mixer.Sound('sounds/kill_mob.wav')
        self.sounds['game_over'] = pygame.mixer.Sound('sounds/gameover.wav')
        self.sounds['scorering'] = pygame.mixer.Sound('sounds/scorering.wav')
        self.sounds['fireball'] = pygame.mixer.Sound('sounds/fireball.wav')
        self.sounds['shot'] = pygame.mixer.Sound('sounds/shot.wav')

    def play(self, name, loop, volume):
        self.sounds[name].play(loops=loop)
        self.sounds[name].set_volume(volume)

    def stop(self, name):
        self.sounds[name].stop()

    def start_fast_music(self, game):
        if game.world.name == '1-1':
            self.stop('overworld')
            self.play('overworld_fast', 99999, 0.5)
```

Класс SoundManager отвечает за управление звуками в игре. Он содержит словарь звуковых эффектов и методы для их загрузки, воспроизведения и остановки. Метод play() проигрывает указанный звуковой эффект с заданными параметрами зацикливания и громкости, а метод stop() останавливает воспроизведение звука. Метод start_fast_music() используется для переключения на более быструю музыку при прохождении первого уровня игры.

MenuManager

```
class MenuManager:

    def __init__(self, game):
```

```
        self.current_state = STATE_MAIN_MENU

        self.main_menu = MainMenu()
        self.loading_menu = LoadingMenu(game)

    def update(self, game):
        if self.current_state == STATE_MAIN_MENU:
            pass

        elif self.current_state == STATE_LOADING:
            self.loading_menu.update(game)

        elif self.current_state == STATE_GAME:
            game.world.update(game)

    def render(self, game):
        if self.current_state == STATE_MAIN_MENU:
            game.world.render_world(game)
            self.main_menu.render(game)

        elif self.current_state == STATE_LOADING:
            self.loading_menu.render(game)

        elif self.current_state == STATE_GAME:
            game.world.render(game)

        pygame.display.update()

    def start_loading(self):
        self.current_state = STATE_LOADING
        self.loading_menu.update_time()
```

Класс MenuManager управляет меню и переключает между состояниями игры. В конструкторе инициализируются экземпляры главного меню и меню загрузки.

Метод update обновляет состояние в зависимости от текущего состояния, вызывая методы обновления соответствующих меню или мира игры.

Метод render отображает текущее состояние в зависимости от значения переменной current_state. Если текущее состояние является главным меню, то отображается мир и главное меню. Если текущее состояние является меню загрузки, то отображается меню загрузки. Если текущее состояние является игрой, то отображается мир игры. В конце метода обновляется экран.

Метод start_loading переключает состояние на меню загрузки и обновляет время начала загрузки.

World

```
class World:
```

```
mob_type = {
    "goomba": Goomba
}

def __init__(self, world_name):
    self.obj = []
    self.obj_bg = []
    self.tubes = []
    self.debris = []
    self.mobs = []
    self.projectiles = []
    self.text_objects = []
    self.map = 0
    self.flag = None

    self.map_size = (0, 0)
    self.sky = 0
    self.textures = {}
    self.name = world_name
    self.mobs_to_spawn = None

    self.is_mob_spawned = [False, False]
    self.score_for_killing_mob = 100
    self.score_time = 0

    self.in_event = False
    self.tick = 0
    self.time = 400

    self.loadWorld_11()

    self.player = Player(128, 351)
    self.camera = Camera(self.map_size[0] * 32, 14)
    self.event = Event()
    self.ui = GameUI()

def render_world(self, game):
    game.screen.blit(self.sky, (0, 0))

    for obj_group in (self.obj_bg, self.obj):
        for obj in obj_group:
            obj.render(game)

    for tube in self.tubes:
        tube.render(game)

def loadWorld_11(self):
    tmx_data = pytmx.util_pygame.load_pygame("worlds/1-1/W11.tmx")
    with open("worlds/1-1/W11.json") as json_file:
        json_data = json.loads(json_file.read())

    self.map_size = (tmx_data.width, tmx_data.height)
```

```
        self.sky = pygame.Surface((WINDOW_WIDTH, WINDOW_HEIGHT))
        self.sky.fill((pygame.Color(json_data["sky_color"])))
        self.mobs_to_spawn = json_data["mobs"]

        self.map = [[0] * tmx_data.height for _ in range(tmx_data.width)]

        layer_num = 0
        for layer in tmx_data.visible_layers:
            for y in range(tmx_data.height):
                for x in range(tmx_data.width):
                    image = tmx_data.get_tile_image(x, y, layer_num)
                    if image is not None:
                        tile_id = tmx_data.get_tile_gid(x, y, layer_num)

                        if layer.name == "Foreground":
                            if tile_id == 22:
                                image = (
                                    image,
                                    tmx_data.get_tile_image(0, 15, layer_num),
                                    tmx_data.get_tile_image(1, 15, layer_num),
                                    tmx_data.get_tile_image(2, 15, layer_num)
                                )

                            self.map[x][y] = Platform(x * tmx_data.tileheight,
y * tmx_data.tilewidth, image, tile_id)
                            self.obj.append(self.map[x][y])
            layer_num += 1

        for tube in json_data["tubes"]:
            self.spawn_tube(tube["x"], tube["y"])

        self.flag = Flag(json_data["flag"]["x"], json_data["flag"]["y"])

    def reset(self, reset_all):
        self.obj = []
        self.obj_bg = []
        self.tubes = []
        self.debris = []
        self.mobs = []
        self.is_mob_spawned = [False, False]

        self.in_event = False
        self.flag = None
        self.sky = None
        self.map = None

        self.tick = 0
        self.time = 400

        self.map_size = (0, 0)
        self.textures = {}
        self.loadWorld_11()
```

```
        self.event.reset()
        self.player.reset(reset_all)
        self.camera.reset()

    def spawn_score_text(self, x, y, score=None):
        if score is None:
            self.text_objects.append(Text(str(self.score_for_killing_mob), 16,
(x, y)))

            self.score_time = pygame.time.get_ticks()
            if self.score_for_killing_mob < 1600:
                self.score_for_killing_mob *= 2

        else:
            self.text_objects.append(Text(str(score), 16, (x, y)))

    def remove_object(self, object):
        self.obj.remove(object)
        self.map[object.rect.x // 32][object.rect.y // 32] = 0

    def remove_whizbang(self, whizbang):
        self.projectiles.remove(whizbang)

    def remove_text(self, text_object):
        self.text_objects.remove(text_object)

    def update_player(self, game):
        self.player.update(game)

    def update_entities(self, game):
        for mob in self.mobs:
            mob.update(game)
            if not self.in_event:
                self.entity_collisions(game)

    def update_time(self, game):
        if not self.in_event:
            self.tick += 1
            if self.tick % 40 == 0:
                self.time -= 1
                self.tick = 0
            if self.time == 100 and self.tick == 1:
                game.sound_manager.start_fast_music(game)
            elif self.time == 0:
                self.player_death(game)

    def update_score_time(self):
        if self.score_for_killing_mob != 100:
            if pygame.time.get_ticks() > self.score_time + 750:
                self.score_for_killing_mob //= 2

    def entity_collisions(self, game):
        if not game.world.player.unkillable:
```

```
            for mob in self.mobs:
                mob.check_collision_with_player(game)

    def player_death(self, game):
        self.in_event = True
        self.player.reset_jump()
        self.player.reset_move()
        self.player.num_of_lives -= 1

        if self.player.num_of_lives == 0:
            self.event.start_kill(game, game_over=True)
        else:
            self.event.start_kill(game, game_over=False)

    def player_win(self, game):
        self.in_event = True
        self.player.reset_jump()
        self.player.reset_move()
        self.event.start_win(game)

    def spawn_tube(self, x_coords, y_coords):
        self.tubes.append(Tube(x_coords, y_coords))

        for y in range(y_coords, 12):
            for x in range(x_coords, x_coords + 2):
                self.map[x][y] = Platform(x * 32, y * 32, image=None,
type_id=0)

    def spawn_debris(self, x, y, type):
        if type == 0:
            self.debris.append(PlatformDebris(x, y))
        elif type == 1:
            self.debris.append(CoinDebris(x, y))

    def try_spawn_mobs(self, game):
        for pack in self.mobs_to_spawn:
            if game.world.player.pos_x > pack["player_x_pos"] and not
pack.get("spawned"):
                kind = self.mob_type.get(pack["type"])
                if kind is None:
                    raise RuntimeError(f"Unknown mob type: {pack['type']}")
                for mob in pack["pos"]:
                    self.mobs.append(kind(mob["x"], mob["y"],
mob["direction"]))
                pack["spawned"] = True

    def update(self, game):
        self.update_entities(game)

        if not game.world.in_event:
            if self.player.in_level_up_animation:
                self.player.change_powerlvl_animation()
            elif self.player.in_level_down_animation:
```

```
                self.player.change_powerlvl_animation()
                self.update_player(game)
            else:
                self.update_player(game)
        else:
            self.event.update(game)

        for debris in self.debris:
            debris.update(game)

        for whizbang in self.projectiles:
            whizbang.update(game)

        for text_object in self.text_objects:
            text_object.update(game)

        if not self.in_event:
            self.camera.update(game.world.player.rect)

        self.try_spawn_mobs(game)

        self.update_time(game)
        self.update_score_time()

    def get_blocks_for_collision(self, x, y):
        return (
            self.map[x][y - 1],
            self.map[x][y + 1],
            self.map[x][y],
            self.map[x - 1][y],
            self.map[x + 1][y],
            self.map[x + 2][y],
            self.map[x + 1][y - 1],
            self.map[x + 1][y + 1],
            self.map[x][y + 2],
            self.map[x + 1][y + 2],
            self.map[x - 1][y + 1],
            self.map[x + 2][y + 1],
            self.map[x][y + 3],
            self.map[x + 1][y + 3]
        )

    def get_blocks_below(self, x, y):
        return (
            self.map[x][y + 1],
            self.map[x + 1][y + 1]
        )

    def render(self, game):
        game.screen.blit(self.sky, (0, 0))

        for obj in self.obj_bg:
            obj.render(game)
```

```
        for mob in self.mobs:
            mob.render(game)

        for obj in self.obj:
            obj.render(game)

        for tube in self.tubes:
            tube.render(game)

        for whizbang in self.projectiles:
            whizbang.render(game)

        for debris in self.debris:
            debris.render(game)

        self.flag.render(game)

        for text_object in self.text_objects:
            text_object.render_in_game(game)

        self.player.render(game)

        self.ui.render(game)
```

Это секция кода для класса World. Класс имеет методы для отрисовки игрового мира, загрузки разных уровней, сброса игры, обновления движения игрока и столкновений объектов, а также отслеживания времени и счета.

Класс World имеет переменные экземпляра для объектов в игре, таких как платформы, трубы, мусор, мобы, снаряды и текстовые объекты. Он также отслеживает карту игры, счет, время, а также экземпляры игрока и камеры.

Метод render_world() отображает игровой мир на экране с помощью Pygame. Сначала отрисовывается небо, затем задний план, а затем передний план, такой как платформы. Трубы также отображаются отдельно.

Метод loadWorld_11() загружает уровень (в данном случае, мир 1-1) из файла TMX и соответствующего файла JSON, который указывает позицию труб, флага и мобов. Уровень разбирается, и каждому тайлу назначается объект в игре (например, платформа). Метод также создает трубы и объект флага.

Метод reset() сбрасывает игровой мир до начального состояния, что полезно для перезапуска игры или переключения уровней.

Метод spawn_score_text() создает текстовый объект счета при убийстве моба, указывая количество очков, которые заработал игрок. Счет увеличивается экспоненциально по мере убийства большего числа мобов.

Методы remove_object(), remove_whizbang() и remove_text() удаляют объекты из игры, когда они больше не нужны.

Методы update_player(), update_entities(), update_time() и update_score_time() обновляют различные аспекты игрового состояния, такие как движение игрока, движение объектов, оставшееся время и счет.

Наконец, метод entity_collisions() проверяет столкновения между объектами в игре, такими как игрок и мобы. Если игрок не является неуязвимым, столкновение с мобом приведет к потере игроком жизни.

Camera

```
import pygame

from const import WINDOW_HEIGHT, WINDOW_WIDTH


class Camera:

    def __init__(self, width, height):
        self.rect = pygame.Rect(0, 0, width, height)
        self.complex_camera(self.rect)

    def complex_camera(self, target_rect):
        x, y = target_rect.x, target_rect.y
        width, height = self.rect.width, self.rect.height
        x, y = (-x + WINDOW_WIDTH / 2 - target_rect.width / 2), (-y +
WINDOW_HEIGHT / 2 - target_rect.height)

        x = min(0, x)
        x = max(-(self.rect.width - WINDOW_WIDTH), x)
        y = WINDOW_HEIGHT - self.rect.h

        return pygame.Rect(x, y, width, height)

    def apply(self, target):
        return target.rect.x + self.rect.x, target.rect.y

    def update(self, target):
        self.rect = self.complex_camera(target)

    def reset(self):
        self.rect = pygame.Rect(0, 0, self.rect.w, self.rect.h)
```

Данный код представляет собой определение класса Camera для игры. Класс имеет методы для создания и обновления камеры, отслеживающей положение игрока на экране.

Конструктор класса принимает ширину и высоту камеры и создает прямоугольник с такими размерами в начальном положении (0, 0).

Метод complex_camera() вычисляет новое положение камеры в зависимости от переданного ему прямоугольника target_rect, который обычно представляет собой прямоугольник, в который должен поместиться игрок. Метод возвращает прямоугольник, который должен быть отображен на экране, с учетом положения игрока.

Метод apply() принимает объект target (например, игровой объект, такой как платформа) и возвращает его координаты на экране с учетом текущего положения камеры.

Метод update() обновляет положение камеры, используя метод complex_camera(), передавая ему прямоугольник с положением игрока на экране.

Метод reset() сбрасывает положение камеры в начальное положение, заданное при создании объекта класса.

CoinDebris

```
import pygame


class CoinDebris:

    def __init__(self, x, y):
        self.rect = pygame.Rect(x, y, 16, 28)

        self.y_vel = -2
        self.y_offset = 0
        self.moving_up = True

        self.current_image = 0
        self.image_tick = 0
        self.images = [
            pygame.image.load('images/coin_an0.png').convert_alpha(),
            pygame.image.load('images/coin_an1.png').convert_alpha(),
            pygame.image.load('images/coin_an2.png').convert_alpha(),
            pygame.image.load('images/coin_an3.png').convert_alpha()
        ]

    def update(self, game):
        self.image_tick += 1

        if self.image_tick % 15 == 0:
            self.current_image += 1

        if self.current_image == 4:
            self.current_image = 0
```

```
            self.image_tick = 0

        if self.moving_up:
            self.y_offset += self.y_vel
            self.rect.y += self.y_vel
            if self.y_offset < -50:
                self.moving_up = False
                self.y_vel = -self.y_vel
        else:
            self.y_offset += self.y_vel
            self.rect.y += self.y_vel
            if self.y_offset == 0:
                game.world.debris.remove(self)

    def render(self, game):
        game.screen.blit(self.images[self.current_image],
game.world.camera.apply(self))
```

Данный код содержит описание класса CoinDebris для создания объекта "обломки монеты", который используется в игре.

Класс имеет конструктор init, который принимает два аргумента - координаты x и y для создания прямоугольника Rect, а также инициализирует переменные, отвечающие за движение объекта и анимацию.

Метод update отвечает за обновление состояния объекта на каждом кадре игры. В нем происходит изменение анимации монеты и ее вертикального положения. При достижении монетой верхней точки траектории ее движение меняется на противоположное, чтобы она начала двигаться вниз. При достижении монетой исходного положения, она удаляется из списка монет debris.

Метод render используется для отображения объекта на экране. Он вызывается каждый кадр игры и использует метод blit объекта pygame.Surface для отображения текущего изображения монеты в позиции, вычисленной с помощью метода apply объекта game.world.camera.

Event

```
import pygame

from const import GRAVITY, FALL_MULTIPLIER, STATE_LOADING


class Event:

    def __init__(self):

        # 0 - game over/kill
        # 1 - win
```

```
        self.type = 0

        self.delay = 0
        self.time = 0
        self.x_vel = 0
        self.y_vel = 0
        self.game_over = False

        self.player_in_castle = False
        self.tick = 0
        self.score_tick = 0

    def reset(self):
        self.type = 0

        self.delay = 0
        self.time = 0
        self.x_vel = 0
        self.y_vel = 0
        self.game_over = False

        self.player_in_castle = False
        self.tick = 0
        self.score_tick = 0

    def start_kill(self, game, game_over):
        """

        Player gets killed.

        """
        self.type = 0
        self.delay = 4000
        self.y_vel = -4
        self.time = pygame.time.get_ticks()
        self.game_over = game_over

        game.sound_manager.stop('overworld')
        game.sound_manager.stop('overworld_fast')
        game.sound_manager.play('death', 0, 0.5)

        # Sets "dead" sprite
        game.world.player.set_image(len(game.world.player.sprites))

    def start_win(self, game):
        self.type = 1
        self.delay = 2000
        self.time = 0

        game.sound_manager.stop('overworld')
        game.sound_manager.stop('overworld_fast')
        game.sound_manager.play('level_end', 0, 0.5)
```

```
        game.world.player.set_image(5)
        game.world.player.x_vel = 1
        game.world.player.rect.x += 10

        # Adding score depends on the map's time left.
        if game.world.time >= 300:
            game.world.player.add_score(5000)
            game.world.spawn_score_text(game.world.player.rect.x + 16,
game.world.player.rect.y, score=5000)
        elif 200 <= game.world.time < 300:
            game.world.player.add_score(2000)
            game.world.spawn_score_text(game.world.player.rect.x + 16,
game.world.player.rect.y, score=2000)
        else:
            game.world.player.add_score(1000)
            game.world.spawn_score_text(game.world.player.rect.x + 16,
game.world.player.rect.y, score=1000)

    def update(self, game):

        # Death
        if self.type == 0:
            self.y_vel += GRAVITY * FALL_MULTIPLIER if self.y_vel < 6 else 0
            game.world.player.rect.y += self.y_vel

            if pygame.time.get_ticks() > self.time + self.delay:
                if not self.game_over:
                    game.world.player.reset_move()
                    game.world.player.reset_jump()
                    game.world.reset(False)
                    game.sound_manager.play('overworld', 9999999, 0.5)
                else:
                    game.menu_manager.current_state = STATE_LOADING
                    game.menu_manager.loading_menu.set_text_and_type('GAME
OVER', False)
                    game.menu_manager.loading_menu.update_time()
                    game.sound_manager.play('game_over', 0, 0.5)

        # Flag win
        elif self.type == 1:

            if not self.player_in_castle:

                if not game.world.flag.flag_omitted:
                    game.world.player.set_image(5)
                    game.world.flag.move_flag_down()
                    game.world.player.flag_animation_move(game, False)

                else:
                    self.tick += 1
                    if self.tick == 1:
                        game.world.player.direction = False
                        game.world.player.set_image(6)
```

```
                    game.world.player.rect.x += 20
                elif self.tick >= 30:
                    game.world.player.flag_animation_move(game, True)
                    game.world.player.update_image(game)

        else:
            if game.world.time > 0:
                self.score_tick += 1
                if self.score_tick % 10 == 0:
                    game.sound_manager.play('scorering', 0, 0.5)

                game.world.time -= 1
                game.world.player.add_score(50)

            else:
                if self.time == 0:
                    self.time = pygame.time.get_ticks()

                elif pygame.time.get_ticks() >= self.time + self.delay:
                    game.menu_manager.current_state = 'Loading'
                    game.menu_manager.loading_menu.set_text_and_type('LW1,
PPOIS, Lipsky', False)
                    game.menu_manager.loading_menu.update_time()
                    game.sound_manager.play('game_over', 0, 0.5)
```

Данный код реализует класс Event, предназначенный для управления событиями в игре.

Класс имеет следующие методы:

init(self): конструктор класса, который инициализирует переменные типа события, задержки, скорости по осям, флаг окончания игры, игрока в замке, текущего тика и текущего тика для подсчета очков.

reset(self): метод, который сбрасывает значения переменных класса на начальные.

start_kill(self, game, game_over): метод, который вызывается при смерти игрока. Он устанавливает тип события, задержку, скорость по оси Y, время, когда событие произошло, флаг окончания игры и звуковые эффекты. Также он устанавливает спрайт игрока на "мертвый" спрайт.

start_win(self, game): метод, который вызывается при победе игрока. Он устанавливает тип события, задержку и звуковые эффекты. В зависимости от времени, оставшегося на уровне, он добавляет соответствующее количество очков игроку и отображает текст с количеством очков. Также он анимирует игрока с флагом.

update(self, game): метод, который обновляет состояние игры в зависимости от типа события. При смерти игрока он опускает его спрайт вниз и сбрасывает игру. При победе игрока он анимирует его с флагом и добавляет очки, пока не закончится время. После этого он переходит на следующий уровень.

Flag

```
import pygame


class Flag:
    def __init__(self, x_pos, y_pos):
        self.rect = None

        self.flag_offset = 0
        self.flag_omitted = False

        # Flag object consists of 2 parts:

        self.pillar_image =
pygame.image.load('images/flag_pillar.png').convert_alpha()
        self.pillar_rect = pygame.Rect(x_pos + 8, y_pos, 16, 304)

        self.flag_image = pygame.image.load('images/flag.png').convert_alpha()
        self.flag_rect = pygame.Rect(x_pos - 18, y_pos + 16, 32, 32)

    def move_flag_down(self):
        self.flag_offset += 3
        self.flag_rect.y += 3

        if self.flag_offset >= 255:
            self.flag_omitted = True

    def render(self, game):
        self.rect = self.pillar_rect
        game.screen.blit(self.pillar_image, game.world.camera.apply(self))

        self.rect = self.flag_rect
        game.screen.blit(self.flag_image, game.world.camera.apply(self))
```

Этот код содержит класс Flag, который представляет флаг на игровом экране.

Конструктор класса принимает координаты x и y, где будет находиться флаг, и инициализирует несколько переменных.

Флаг состоит из двух частей: основы и флага. Их изображения загружаются из файлов 'flag_pillar.png' и 'flag.png'.

Метод move_flag_down смещает флаг вниз на 3 пикселя при каждом вызове, увеличивая значение переменной flag_offset. Если flag_offset достигает значения 255, переменная flag_omitted устанавливается в True, что означает, что флаг целиком опущен вниз.

Метод render отрисовывает две части флага на экране, используя метод blit() из Pygame. Он также устанавливает переменную rect текущей части флага для обновления экрана.

Mob

```
import pygame

from const import FALL_MULTIPLIER


class Mob:

    def __init__(self):

        self.state = 0
        self.x_vel = 0
        self.y_vel = 0

        self.move_direction = True
        self.on_ground = False
        self.collision = True

        self.image = None
        self.rect = None

    def update_x_pos(self, blocks):
        self.rect.x += self.x_vel

        for block in blocks:
            if block != 0 and block.type != 'BGObject':
                if pygame.Rect.colliderect(self.rect, block.rect):
                    if self.x_vel > 0:
                        self.rect.right = block.rect.left
                        self.x_vel = - self.x_vel
                    elif self.x_vel < 0:
                        self.rect.left = block.rect.right
                        self.x_vel = - self.x_vel

    def update_y_pos(self, blocks):
        self.rect.y += self.y_vel * FALL_MULTIPLIER

        self.on_ground = False
        for block in blocks:
            if block != 0 and block.type != 'BGObject':
                if pygame.Rect.colliderect(self.rect, block.rect):
                    if self.y_vel > 0:
                        self.on_ground = True
                        self.rect.bottom = block.rect.top
                        self.y_vel = 0

    def check_map_borders(self, game):
```

```
        if self.rect.y >= 450:
            self.die(game, True, False)
        if self.rect.x <= 1 and self.x_vel < 0:
            self.x_vel = -self.x_vel

    def die(self, game, instantly, crushed):
        pass

    def render(self, game):
        pass
```

Данный код содержит определение класса Mob, который представляет игровой объект. Объект может двигаться по горизонтали и вертикали, проверять столкновения с блоками на карте, умирать, и отображаться на экране игры.

Атрибуты класса:

state - состояние объекта
x_vel - скорость объекта по горизонтали
y_vel - скорость объекта по вертикали
move_direction - направление движения объекта
on_ground - находится ли объект на земле
collision - произошло ли столкновение объекта с другим объектом на карте
image - изображение объекта
rect - прямоугольник, описывающий позицию и размеры объекта на экране игры
Методы класса:

update_x_pos(blocks) - обновление позиции объекта по горизонтали на основе текущей скорости и столкновений с блоками на карте
update_y_pos(blocks) - обновление позиции объекта по вертикали на основе текущей скорости и столкновений с блоками на карте
check_map_borders(game) - проверка, находится ли объект в пределах границ карты
die(game, instantly, crushed) - уничтожение объекта
render(game) - отображение объекта на экране игры.