МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Институт: №8 «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Реферат
по курсу «Фундаментальная информатика»
I семестр
Тема:
«Игра «Змейка» на Python, библиотека Pygame»

Группа	М8О-109Б-22
Студент	Моравская В.И.
Преподаватель	Сысоев М.А.
Оценка	
Дата	

Москва, 2022

Python и Pygame в разработке игр

Все *масштабные 3D игры* так или иначе создаются с помощью специальных *игровых движков*. Игровой движок - это по сути набор инструментов, который позволяет работать с графикой, физикой, скриптами и прочим. Движки довольно требовательны к *производительности*, а *Python* сам по себе *медленный*, поэтому непосредственно ядро игры на *Python* не пишут. Поэтому не смотря на возможность работы с графикой и в целом реальность разработки игр, сам по себе *Python* не используется или практически не используется, как основной язык для разработки крупных игр. Но его часто используют как *вспомогательный инструмент*.

На нём пишут *игровую логику*, используют для написания *внутриигровых скриптов* и *подсобной работы*, не касающейся рендеринга, например, организации *серверных элементов управления*, внутриигрового *моддинга*, *интерфейсов* и прочего. Игровая логика обычно *не содержит сложных вычислений* и скорость языка отходит на второй план. Это и ляжет на плечи *Python*. Действительно сложные или требующие высокой производительности части (какой-нибудь условный поиск пути) можно унести в движок.

Получается, что скриптовые языки такие как *Python* вызывают какие-либо методы движка и оперируют ими для создания игровой логики и наоборот: движок может вызывать заранее оговоренные функции в скрипте, где разработчик уже как-то обрабатывает вызов. То есть *скрипты позволяют разделить слои логики игры и логики игрового движка*. Вы можете изменять игровую логику, настройку игры и прочие параметры *без необходимости перекомпиляции* всего кода.

Получается, что *Python* не такой уж редкий гость в крупном геймдеве, однако используется он далеко не как основной язык, когда речь касается крупных многобюджетных игр.

Однако, когда же речь идёт о чём-то более простом, о создании несложных *2D* и *3D* игр, *Python* выступает во всей красе. Для создания хобби-проектов, инди и мобильные игр под *Android* Питон предоставляет несколько хороших и относительно популярных инструментов. Одним из них является библиотека Pygame.

Pygame – это библиотека модулей для языка *Python*, созданная для разработки *2D* игр. Также *Pygame* можно называть своего рода фреймворком для создания игр. Он имеет хорошее сообщество, открытый исходный код, кроссплатформенность, качественную документацию, множеством примеров игр, а ещё он довольно простотой для изучения. *PyGame* хорошее начало, чтобы познакомиться с особенностями разработки игр. Более опытными программистами *Pygame* может использоваться для быстрого создания прототипа игры, чтобы посмотреть, как все будет работать. После этого игра переписывается на другом языке. Другими словами, преимущество *Pygame* в легком обучении и быстрой разработке. С помощью него вполне можно создать отличную игру, но

скорее всего казуальную. *Pygame*-приложения могут работать под Android на телефонах и планшетах с использованием подмножества *Pygame* для *Android*.

Игра «Змейка»

Наверняка, все слышали об этой легендарной игре, чья суть невероятно проста: игрок управляет змейкой, перемещаясь по полю из клеток, и добывает очки, поедая «яблоки». Чем больше змейка съест яблок, тем длиннее она станет. Если змейка ударится в стену или в саму себя – игра закончится. Теперь же разберем сам код.

Быстро посмотрим на импорты:

```
import pygame
import sys
import random
import time
```

- Сам pygame, на котором держится игра
- Sys нужен для функции exit
- Random для координат еды
- Time для функции sleep

Большую часть кода занимают три главных класса, в каждом из которых находится несколько функций.

1. Класс Game

Этот класс посвящен основным параметрам самой игры.

- __init__ - инициализация класса, устанавливаем параметры окна игры, основные цвета. Создаем fps_controller для будущей прорисовки кадров, добавляем счетчик очков.

```
class Game():
    def __init__(self):

        self.screen_width = 720
        self.screen_height = 720

        self.green = pygame.Color(30, 89, 69)
        self.back = pygame.Color(168, 228, 160)
        self.red = pygame.Color(227, 38, 54)

        self.fps_controller = pygame.time.Clock()

        self.score = 0
```

- init_and_check_for_errors – функция для проверки наличия ошибок в программе

```
def init_and_check_for_errors(self):
    check_errors = pygame.init()
    if check_errors[1] > 0:
        sys.exit()
    else:
        print('Ok')
```

- set_surface_and_title – создаем с помощью параметров высоты и ширины «поверхность» - наше окно, на котором будет отображаться наша игра. Задаем заголовок окна.

```
def set_surface_and_title(self):
    self.play_surface = pygame.display.set_mode((
        self.screen_width, self.screen_height))
    pygame.display.set_caption('ЗМЕЙКА БЕЗ СМС И РЕГИСТРАЦИИ')
```

- event_loop – здесь мы задаем наше управление. В нашем случае – направление движения змейки (стрелочки или wasd) и преждевременный выход из игры (esc).

```
def event_loop(self, change_to):

    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT or event.key == ord('d'):
                change_to = "RIGHT"
            elif event.key == pygame.K_LEFT or event.key == ord('a'):
                change_to = "LEFT"
            elif event.key == pygame.K_UP or event.key == ord('w'):
                change_to = "UP"
            elif event.key == pygame.K_DOWN or event.key == ord('s'):
                change_to = "DOWN"
            elif event.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()
    return change_to
```

- refresh_screen – настраиваем fps, заставляем игру обновляться с указанной частотой.

```
def refresh_screen(self):
    pygame.display.flip()
    game.fps_controller.tick(20)
```

- show_score – функция для отображения наших очков. Задаем шрифт и размер, заставляем отображаться и обновляться счетчик очков. Изначально наши очки отображаются сверху слева, но, как только мы проиграем, наш счетчик переместится на середину экрана.

```
def show_score(self, choice=1):
    s_font = pygame.font.SysFont('comic_sans', 24)
    s_surf = s_font.render(
        'Счет: {0}'.format(self.score), True, self.green)
    s_rect = s_surf.get_rect()
    if choice == 1:
        s_rect.midtop = (60, 10)
    else:
        s_rect.midtop = (360, 320)
    self.play_surface.blit(s_surf, s_rect)
```

- game_over – функция проигрыша. Снова задаем шрифт и размер надписи, выводим ее в центр экрана, передаем в функцию show_score 0, чтобы надпись с итоговым количеством очков оказалась в центре. Заставляем игру «зависнуть» на несколько секунд, а потом закрываем ее.

```
def game_over(self):
    go_font = pygame.font.SysFont('comic_sans', 50)
    go_surf = go_font.render('Ты проиграл :(', True, self.red)
    go_rect = go_surf.get_rect()
    go_rect.midtop = (360, 250)
    self.play_surface.blit(go_surf, go_rect)
    self.show_score(0)
    pygame.display.flip()
    time.sleep(2)
    pygame.quit()
    sys.exit()
```

2. Класс Snake

Как можно понять из названия, этот класс посвящен самой змейке.

- __init__ - инициализация класса, задаем изначальное положение нашей

змейки (головы, тела и хвоста), цвет, направление движения.

```
class Snake():
    def __init__(self, snake_color):
        self.snake_head_pos = [100, 50]
        self.snake_body = [[100, 50], [90, 50], [80, 50]]
        self.snake_color = snake_color
        self.direction = "RIGHT"
        self.change_to = self.direction
```

- validate_direction_and_change – «профилактика» противоположных движений: если змейка двигается, например, вправо, то мы не сможем заставить ее поползти сразу влево, иначе она автоматом столкнется со своим телом.

```
def validate_direction_and_change(self):
    if any((self.change_to == "RIGHT" and not self.direction == "LEFT",
            self.change_to == "LEFT" and not self.direction == "RIGHT",
            self.change_to == "UP" and not self.direction == "DOWN",
            self.change_to == "DOWN" and not self.direction == "UP")):
        self.direction = self.change_to
```

- change_head_position – заставляем голову нашей змейки двигаться: в зависимости от состояния (которые задаются клавишами) увеличиваем или уменьшаем значение координат.

```
def change_head_position(self):
    if self.direction == "RIGHT":
        self.snake_head_pos[0] += 10
    elif self.direction == "LEFT":
        self.snake_head_pos[0] -= 10
    elif self.direction == "UP":
        self.snake_head_pos[1] -= 10
    elif self.direction == "DOWN":
        self.snake_head_pos[1] += 10
```

- snake_body_mechanism – механизм роста змейки. На самом деле наша змейка увеличивается постоянно, но мы постоянно удаляем ей последний сегмент, только если она не съест яблоко. Когда координаты яблока и головы змейки совпадают, мы позволяем одному сегменту остаться, а также задаем новое случайное положение яблока на поле и добавляем очко. Махинации с делением и умножением на 10 нужны затем, чтобы яблоки возникали ровно на нашей воображаемой сетке, т.к. сами яблоки – квадраты 10*10.

```
def snake_body_mechanism(self, score, food_pos, screen_width, screen_height):
    self.snake_body.insert(0, list(self.snake_head_pos))
    if (self.snake_head_pos[0] == food_pos[0] and
            self.snake_head_pos[1] == food_pos[1]):
        food_pos = [random.randrange(1, screen_width/10)*10,
                    random.randrange(1, screen_height/10)*10]
        score += 1
    else:
        self.snake_body.pop()
    return score, food_pos
```

- draw_snake – функция прорисовки змейки. Каждый раз заполняем поле заново, и прорисовываем каждый сегмент тела змейки. Для каждого сегмента мы назначили цвет, форму (квадрат), указали координаты сегмента.

```
def draw_snake(self, play_surface, surface_color):
    play_surface.fill(surface_color)
    for pos in self.snake_body:
        pygame.draw.rect(
            play_surface, self.snake_color, pygame.Rect(
                pos[0], pos[1], 10, 10))
```

- check_for_boundaries – проверка на столкновения. Если координаты головы змейки станут выйдут за границы окна, то игра завершится. Для каждого сегмента смотрим то же самое: если координаты головы и сегмента совпадут, то игра завершится.

```
def check_for_boundaries(self, game_over, screen_width, screen_height):
    if any((
        self.snake_head_pos[0] > screen_width-10
        or self.snake_head_pos[0] < 0,
        self.snake_head_pos[1] > screen_height-10
        or self.snake_head_pos[1] < 0
            )):
        game_over()
    for block in self.snake_body[1:]:
        if (block[0] == self.snake_head_pos[0] and
                block[1] == self.snake_head_pos[1]):
            game_over()
```

3. Класс Food

Самый короткий класс, посвящен, в целом, только прорисовке еды.

- __init__ - инициализация, задаем цвет, размер и позицию еды.

```
class Food():
    def __init__(self, food_color, screen_width, screen_height):

        self.food_color = food_color
        self.food_size_x = 10
        self.food_size_y = 10
        self.food_pos = [random.randrange(1, screen_width/10)*10,
                         random.randrange(1, screen_height/10)*10]
```

- draw_food – прорисовка еды. Яблоки у нас тоже квадратные, задаем цвет, форму, позицию и размер.

```
def draw_food(self, play_surface):

    pygame.draw.rect(
        play_surface, self.food_color, pygame.Rect(
            self.food_pos[0], self.food_pos[1],
            self.food_size_x, self.food_size_y))
```

4. **Создаем классы. Инициализируем Pygame.**

```
game = Game()
snake = Snake(game.green)
food = Food(game.red, game.screen_width, game.screen_height)

game.init_and_check_for_errors()
game.set_surface_and_title()
```

5. **Создаем бесконечный цикл, в ходе которого вызываются все наши функции в классах с указанными параметрами**

```
while True:
    snake.change_to = game.event_loop(snake.change_to)

    snake.validate_direction_and_change()
    snake.change_head_position()
    game.score, food.food_pos = snake.snake_body_mechanism(
        game.score, food.food_pos, game.screen_width, game.screen_height)
    snake.draw_snake(game.play_surface, game.back)

    food.draw_food(game.play_surface)

    snake.check_for_boundaries(
    game.game_over, game.screen_width, game.screen_height)

    game.show_score()
    game.refresh_screen()
```