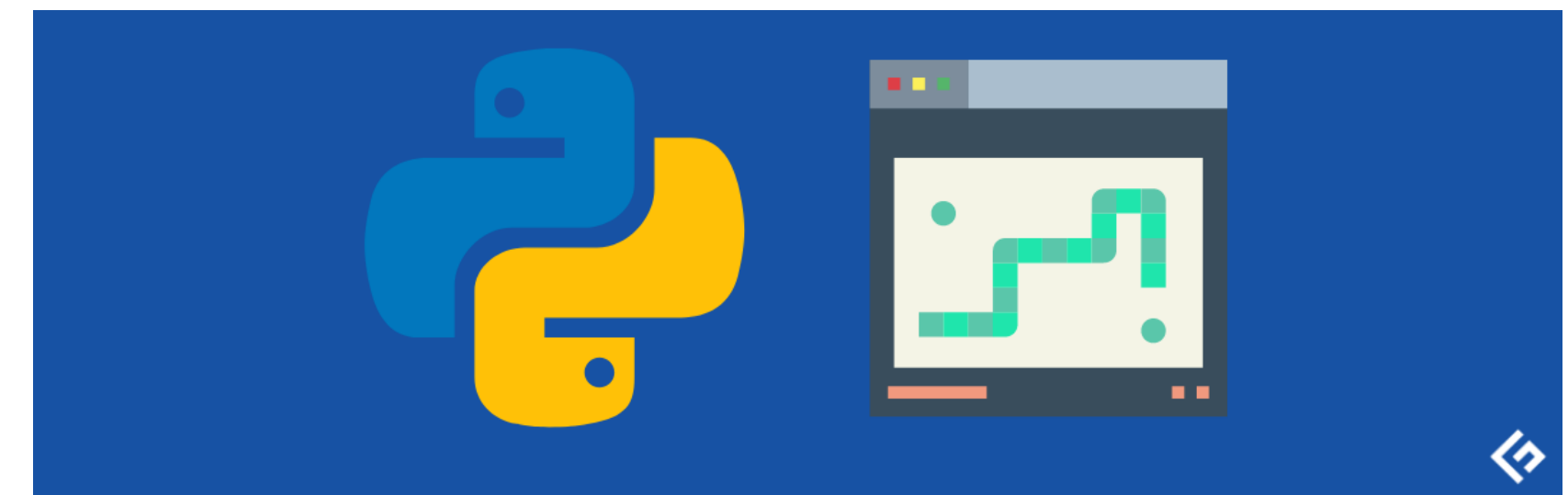


ООП в Python

Разработка игры с использованием ООП



Калугин Е.И.

Цели лекции

1. Научиться разрабатывать приложения с использованием ООП;
- 2.

**Хотим применить свои знания
и написать змейку**

Звучит просто

Как делать?

- Выбрать библиотеку
- Выбрать архитектуру приложения
- Определить какие классы нужны
- Реализовать классы

Выбор библиотеки для разработки игры



Pygame (рус. Пайгейм) — набор модулей (библиотек) языка программирования Python, предназначенный для написания компьютерных игр и мультимедиа-приложений. Pygame базируется на мультимедийной библиотеке SDL.

Изначально Pygame был написан Питом Шиннерсом. Начиная примерно с 2004/2005 года поддерживается и развивается сообществом свободного программного обеспечения.

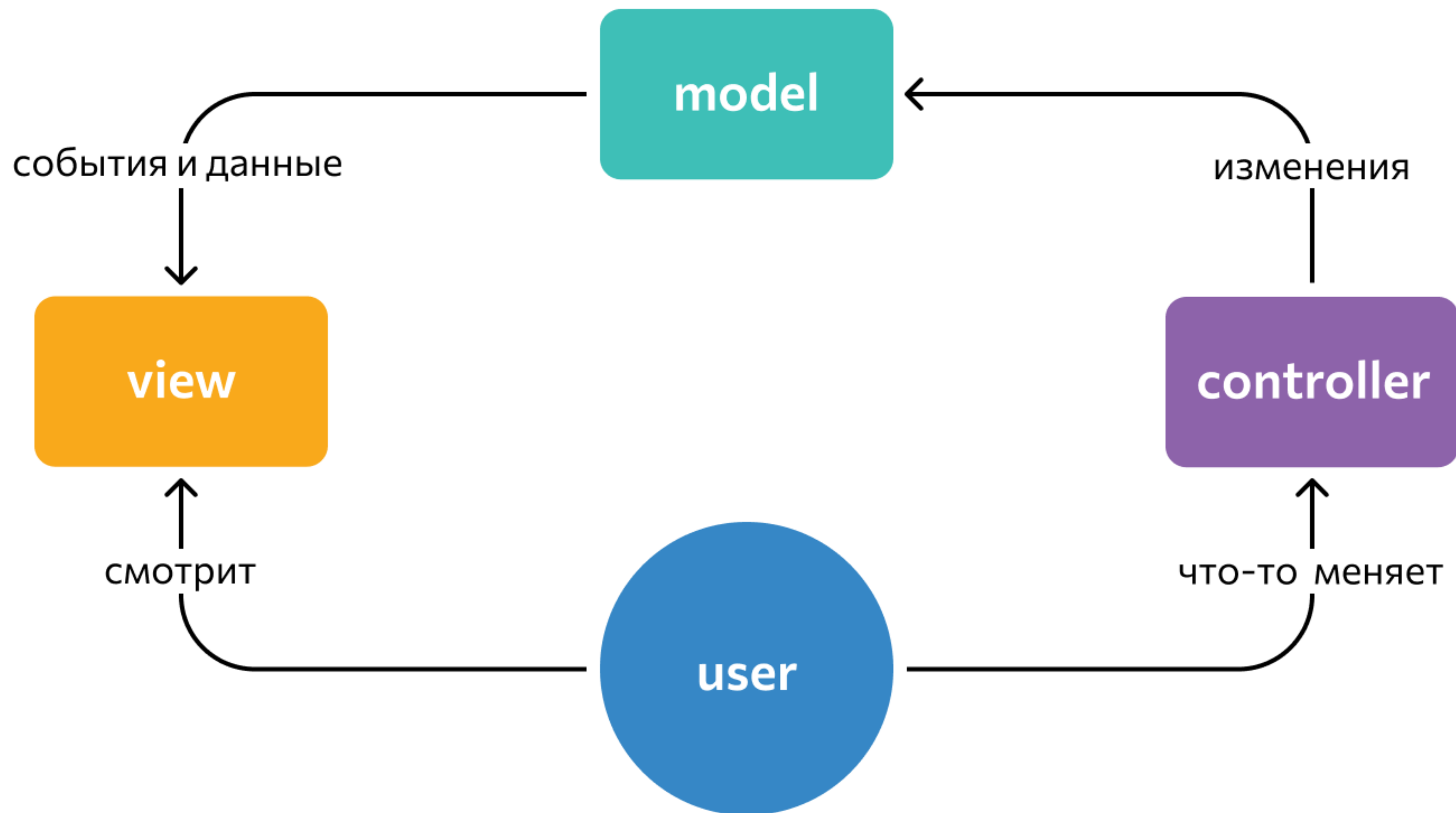
Pygame-приложения могут работать под Android на телефонах и планшетах с использованием подмножества Pygame для Android (pgs4a). На этой платформе поддерживаются звук, вибрация, клавиатура, акселерометр.

Выбор архитектуры

Коротко о паттерне MVC

Как следует из названия, паттерн MVC включает в себя 3 компонента: Модель, Представление и Контроллер. Каждый из компонентов выполняет свою роль и является взаимозаменяемым. Это значит, что компоненты связаны друг с другом лишь некими четкими интерфейсами, за которыми может лежать любая реализация. Такой подход позволяет подменять и комбинировать различные компоненты, обеспечивая необходимую логику работы или внешний вид приложения. Разберемся с теми функциями, которые выполняет каждый компонент.

Паттерн MVC



Паттерн MVC

Модель

Отвечает за внутреннюю логику работы программы. Здесь мы можем скрыть способы хранения данных, а также правила и алгоритмы обработки информации. Например, для одного приложения мы можем создать несколько моделей. Одна будет отладочной, а другая рабочей. Первая может хранить свои данные в памяти или в файле, а вторая уже задействует базу данных. По сути это просто паттерн Стратегия.

Паттерн MVC

Представление

Отвечает за отображение данных Модели. На этом уровне мы лишь предоставляем интерфейс для взаимодействия пользователя с Моделью. Смысл введения этого компонента тот же, что и в случае с предоставлением различных способов хранения данных на основе нескольких Моделей.

Например, на ранних этапах разработки мы можем создать простое консольное представление для нашего приложения, а уже потом добавить красиво оформленный GUI. Причем, остается возможность сохранить оба типа интерфейсов.

Кроме того, следует учитывать, что в обязанности Представления входит лишь своевременное отображение состояния Модели. За обработку действий пользователя отвечает Контроллер, о котором мы сейчас и поговорим.

Паттерн MVC

Контроллер

Обеспечивает связь между Моделью и действиями пользователя, полученными в результате взаимодействия с Представлением. Координирует моменты обновления состояний Модели и Представления. Принимает большинство решений о переходах приложения из одного состояния в другое.

Фактически на каждое действие, которое может сделать пользователь в Представлении, должен быть определен обработчик в Контроллере. Этот обработчик выполнит соответствующие манипуляции над моделью и в случае необходимости сообщит Представлению о наличии изменений.

**Что мы знаем
о змейке?**

Разберемся с классами

Какие классы нужны?

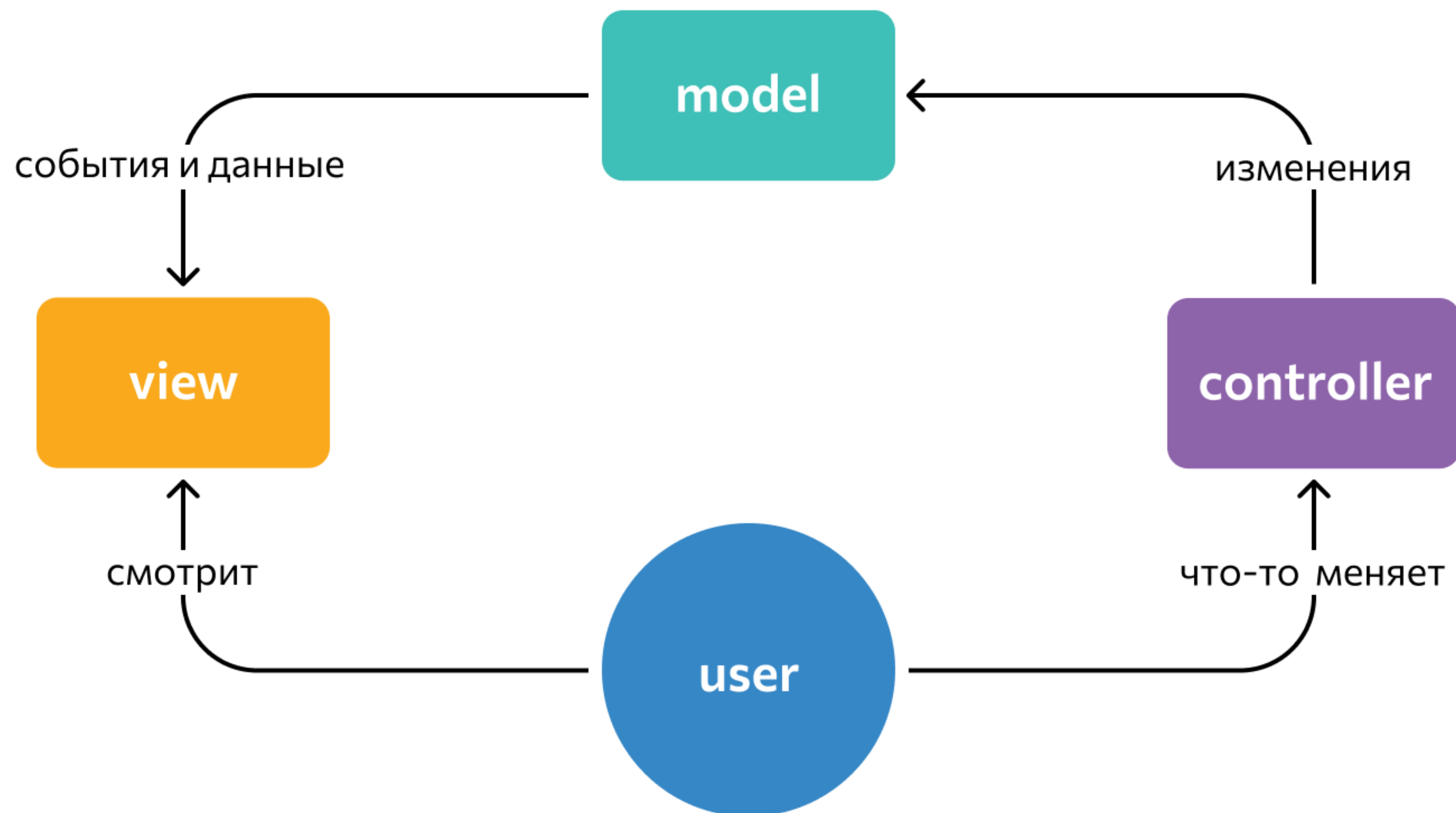
1. Для работы приложения

- Настройки
- Модель
- Представление
- Контроллер

2. Для реализации логики игры

- Змейка
- Еда

Паттерн MVC



Паттерн MVC

Модель

Отвечает за внутреннюю логику работы программы. Здесь мы можем скрыть способы хранения данных, а также правила и алгоритмы обработки информации. Например, для одного приложения мы можем создать несколько моделей. Одна будет отладочной, а другая рабочей. Первая может хранить свои данные в памяти или в файле, а вторая уже задействует базу данных. По сути это просто паттерн Стратегия.

Паттерн MVC

Представление

Отвечает за отображение данных Модели. На этом уровне мы лишь предоставляем интерфейс для взаимодействия пользователя с Моделью. Смысл введения этого компонента тот же, что и в случае с предоставлением различных способов хранения данных на основе нескольких Моделей.

Например, на ранних этапах разработки мы можем создать простое консольное представление для нашего приложения, а уже потом добавить красиво оформленный GUI. Причем, остается возможность сохранить оба типа интерфейсов.

Кроме того, следует учитывать, что в обязанности Представления входит лишь своевременное отображение состояния Модели. За обработку действий пользователя отвечает Контроллер, о котором мы сейчас и поговорим.

Паттерн MVC

Контроллер

Обеспечивает связь между Моделью и действиями пользователя, полученными в результате взаимодействия с Представлением. Координирует моменты обновления состояний Модели и Представления. Принимает большинство решений о переходах приложения из одного состояния в другое.

Фактически на каждое действие, которое может сделать пользователь в Представлении, должен быть определен обработчик в Контроллере. Этот обработчик выполнит соответствующие манипуляции над моделью и в случае необходимости сообщит Представлению о наличии изменений.

Начнем писать код

Создадим виртуальные окружение для проекта:

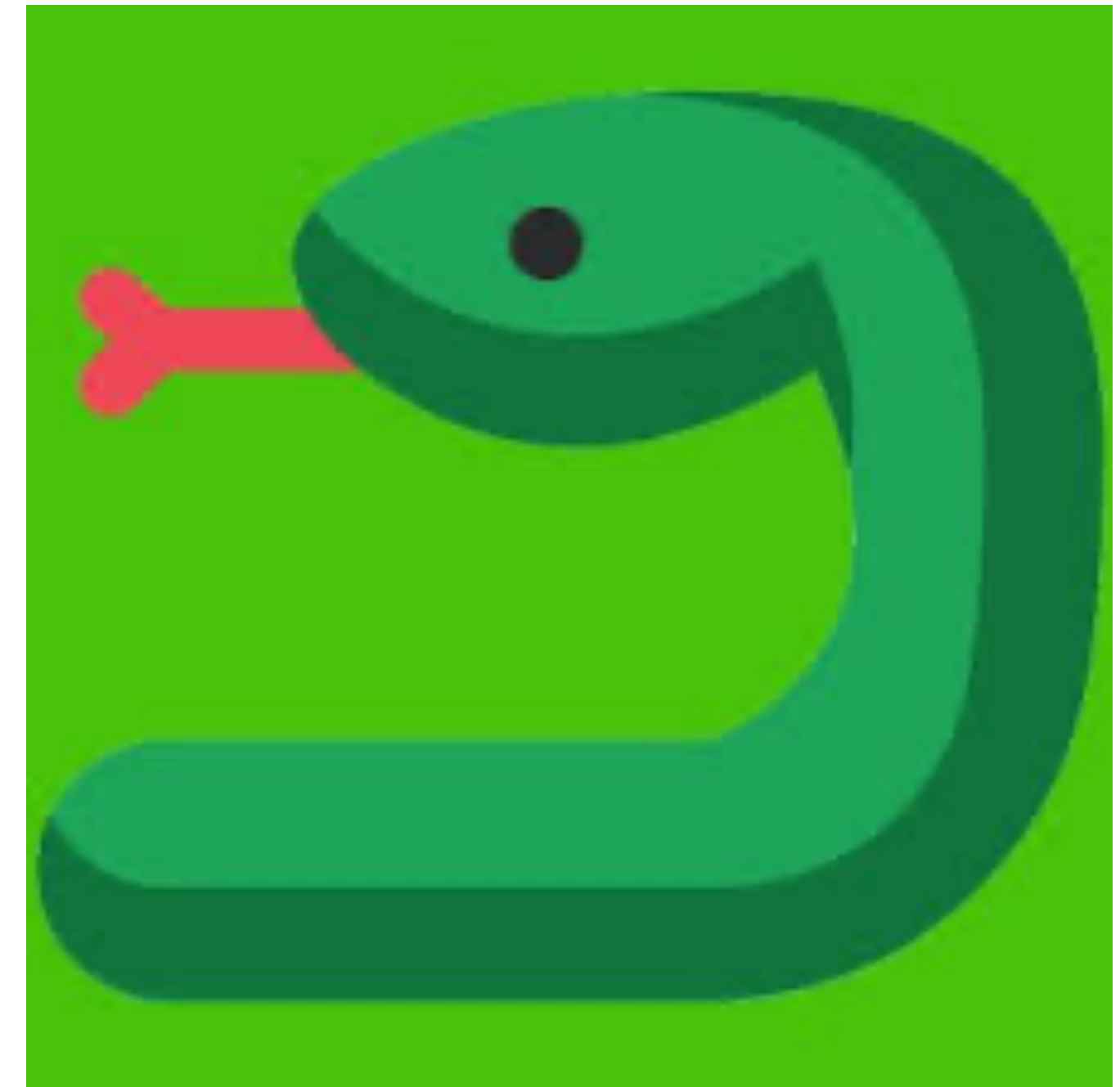
- `python3 -m venv .venv`
- `source ./venv/bin/activate`

Создадим 2 файла:

- `main.py`
- `game_objects.py`

Установим pygame:

```
pip install pygame
```



**Опишем основные атрибуты и
методы наших классов**

main.py

```
import pygame as pg

class GameSettings:
    def __init__(self):
        self.WINDOW_SIZE = 750 # можно 1000
        self.TILE_SIZE = 50
        self.screen = pg.display.set_mode([self.WINDOW_SIZE] * 2)
        self.clock = pg.time.Clock()

class GameModel(GameSettings):
    def new_game(self):
        pass

    def update(self):
        pass

class GameController(GameSettings):
    def __init__(self, model: GameModel):
        GameSettings.__init__(self)
        self.model = model

    def check_events(self):
        pass

class GameVeiw(GameSettings):
    def __init__(self, model: GameModel, controller: GameController):
        GameSettings.__init__(self)
        self.model = model
        self.controller = controller

    def draw(self):
        pass

    def run_game(self):
        self.model.new_game()
        while True:
            self.controller.check_events()
            self.model.update()
            self.draw()

if __name__ == "__main__":
    model = GameModel()
    controller = GameController(model=model)
    view = GameVeiw(model=model, controller=controller)
    view.run_game()
```

game_objects.py

```
import pygame as pg

You, now | 1 author (You)

class Snake:
    def __init__(self, size):
        self.size = size
        self.head = pg.rect.Rect(0, 0, size - 2, size - 2)

    def move(self):
        pass

    def set_position(self, position):
        pass

    def get_head_postion(self):
        pass

You, now | 1 author (You)

class Food:
    def __init__(self, size):
        self.size = size
        self.body = pg.rect.Rect(0, 0, size - 2, size - 2)

    def get_postion(self):
        pass

    def set_position(self, position):
        pass
```

**Попробуем нарисовать
игровое поле**


```
from game_objects import Food, Snake
```

```
class GameModel(GameSettings):  
    def new_game(self):  
        self.food = Food(self.TILE_SIZE)  
        self.snake = Snake(self.TILE_SIZE)  
  
    def update(self):  
        pg.display.flip()  
        self.clock.tick(60)
```

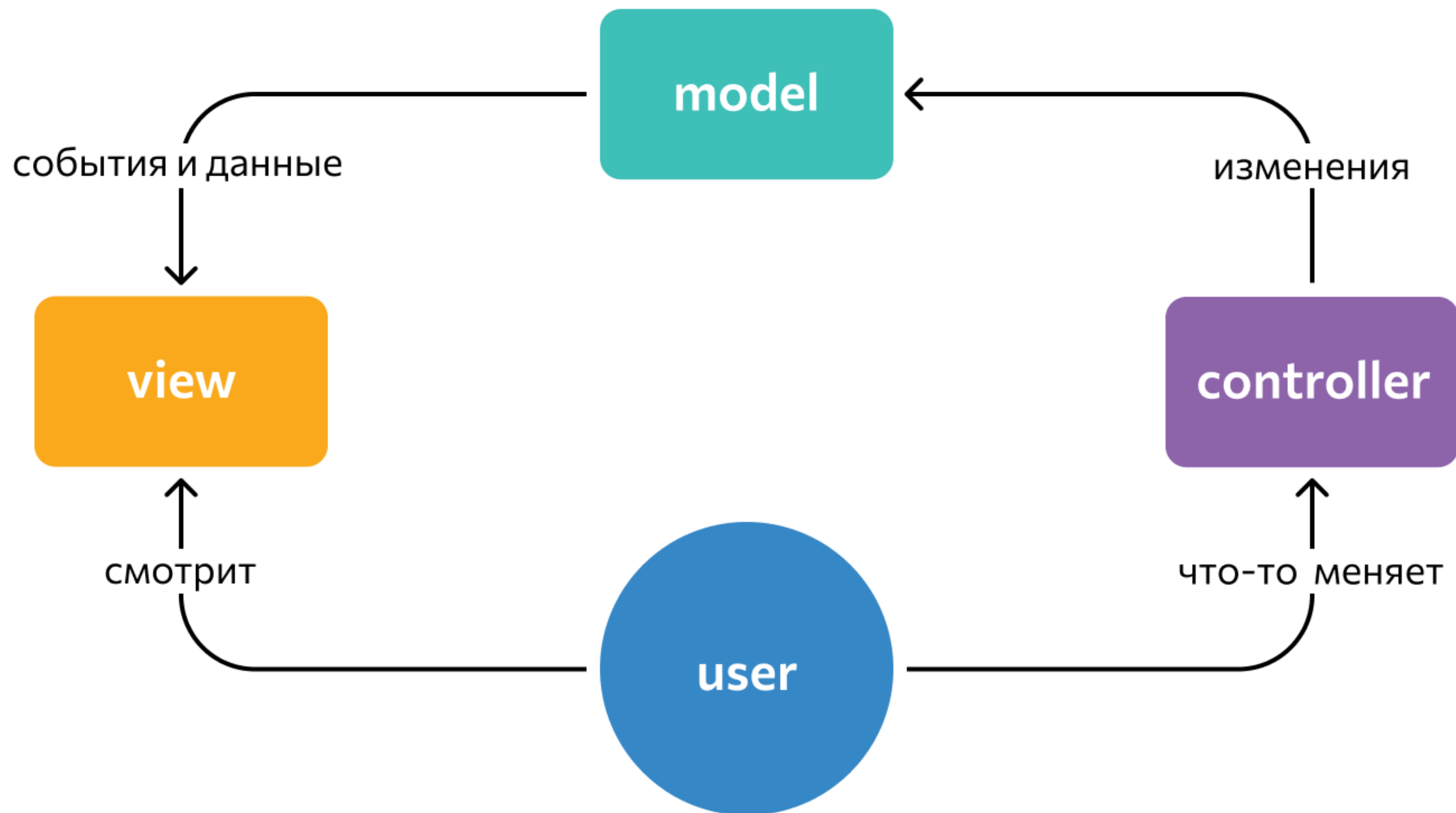
ИМПОРТИРУЕМ КЛАССЫ В НАЧАЛЕ ФАЙЛА



```
class GameVeiw(GameSettings):  
    def __init__(self, model: GameModel, controller: GameController):  
        GameSettings.__init__(self)  
        self.model = model  
        self.controller = controller  
  
    def draw_grid(self):  
        for x in range(0, self.WINDOW_SIZE, self.TILE_SIZE):  
            pg.draw.line(self.screen, [50] * 3, (x, 0), (x, self.WINDOW_SIZE))  
        for y in range(0, self.WINDOW_SIZE, self.TILE_SIZE):  
            pg.draw.line(self.screen, [50] * 3, (0, y), (self.WINDOW_SIZE, y))  
  
    def draw(self):  
        # рисуем поле  
        self.screen.fill("black")  
        self.draw_grid()
```

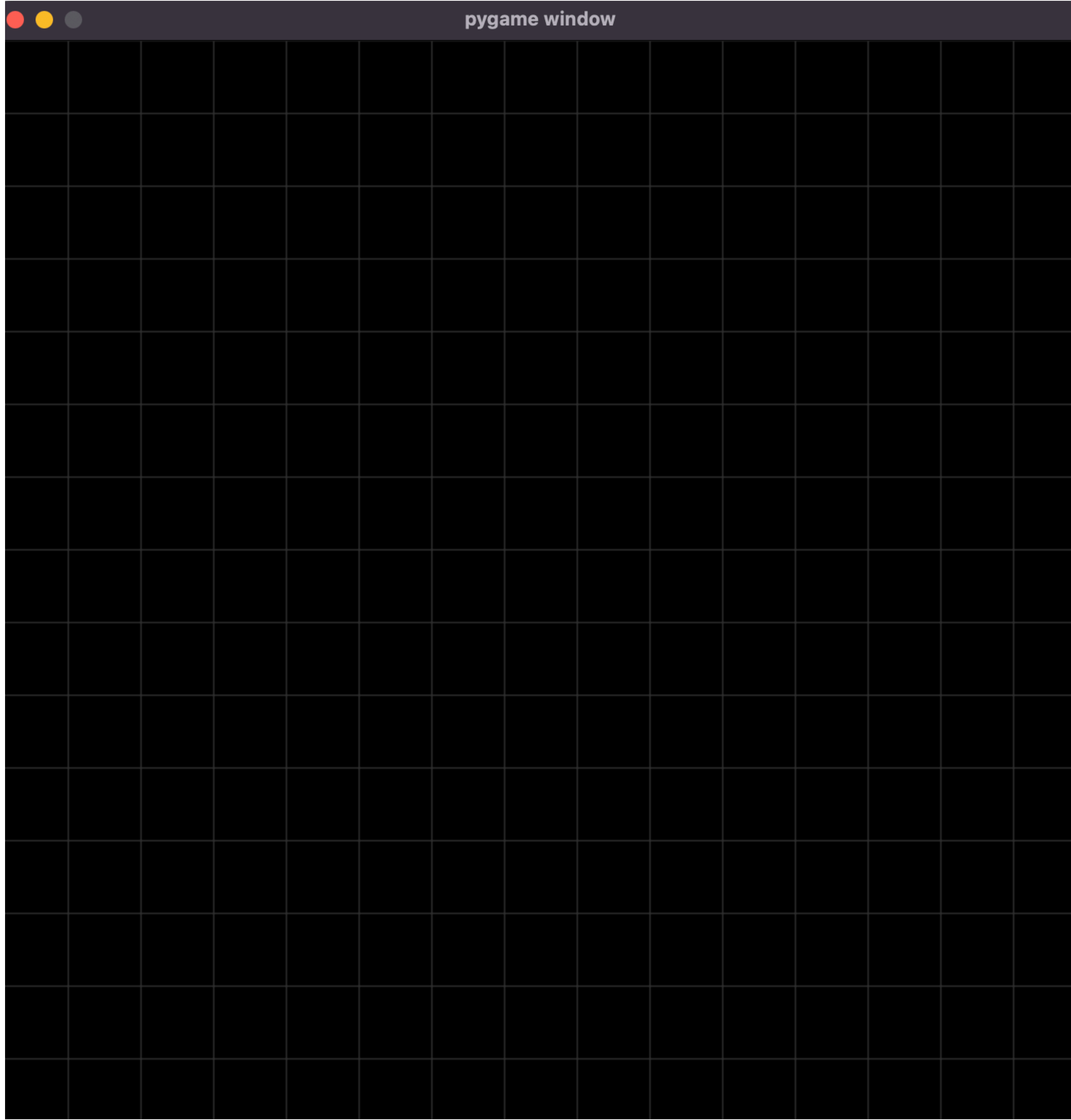
```
class GameController(GameSettings):  
    def __init__(self, model: GameModel):  
        GameSettings.__init__(self)  
        self.model = model  
  
    def check_events(self):  
        for event in pg.event.get():  
            if event.type == pg.QUIT:  
                pg.quit()  
                sys.exit()
```

Паттерн MVC



Запустим `main.py`
Что на экране?

А где змейка?



Рисуем змейку

- Реализуем метод `get_head_position` класса `Snake`
- Добавим новую строку в метод `draw` класса `GameView`

```
class Snake:
    def __init__(self, size):
        self.size = size
        self.head = pg.rect.Rect(0, 0, size - 2, size - 2)

    def move(self):
        pass

    def set_position(self, position):
        pass

    def get_head_position(self):
        return self.head
```

```
def draw(self):
    # рисуем поле
    self.screen.fill("black")
    self.draw_grid()
    # рисуем змейку
    pg.draw.rect(self.screen, "green", self.model.snake.get_head_position())
```

Почему змейка

всегда в одном месте?

Почему змейка

не умеет ходить?

Где еда?



```
class GameModel(GameSettings):
```

```
    def new_game(self):
```

```
        self.food = Food(self.TILE_SIZE)
```

```
        self.snake = Snake(self.TILE_SIZE)
```

```
        self.set_random_snake_position()
```

```
    def update(self):
```

```
        pg.display.flip()
```

```
        self.clock.tick(60)
```

```
    def get_random_position(self):
```

```
        return [
```

```
            randrange(self.TILE_SIZE // 2, self.WINDOW_SIZE - self.TILE_SIZE // 2, self.TILE_SIZE),
```


```
            randrange(self.TILE_SIZE // 2, self.WINDOW_SIZE - self.TILE_SIZE // 2, self.TILE_SIZE),
```

```
        ]
```

```
    def set_random_snake_position(self):
```

```
        self.snake.set_position(self.get_random_position())
```

```
from random import randrange
```



```
class Snake:
```

```
    def __init__(self, size):
```

```
        self.size = size
```

```
        self.head = pg.rect.Rect(0, 0, size - 2, size - 2)
```

```
    def move(self):
```

```
        pass
```

```
    def set_position(self, position):
```

```
        self.head.center = position
```

```
    def get_head_postion(self):
```

```
        return self.head
```

Почему змейка

~~всегда в одном месте?~~

Почему змейка

не умеет ходить?



Где еда?



Теперь попробуем сами


```

class GameModel(GameSettings):
    def new_game(self):
        self.food = Food(self.TILE_SIZE)
        self.snake = Snake(self.TILE_SIZE)
        self.set_random_snake_position()
        self.set_random_food_position()

    def update(self):
        pg.display.flip()
        self.clock.tick(60)

    def get_random_position(self):
        return [
            randrange(self.TILE_SIZE // 2, self.WINDOW_SIZE // 2),
            randrange(self.TILE_SIZE // 2, self.WINDOW_SIZE // 2)
        ]

    def set_random_snake_position(self):
        self.snake.set_position(self.get_random_position())

    def set_random_food_position(self):
        position = self.get_random_position()
        self.food.set_position(position)

```

```

class Food:
    def __init__(self, size):
        self.size = size
        self.body = pg.rect.Rect(0, 0, size - 2, size - 2)

    def get_postion(self):
        return self.body

    def set_position(self, position):
        self.body.center = position

```

```

class GameVeiw(GameSettings):

```

```

    def draw(self):
        # рисуем поле
        self.screen.fill("black")
        self.draw_grid()
        # рисуем змейку
        pg.draw.rect(self.screen, "green", self.model.snake.get_head_postion())
        pg.draw.rect(self.screen, "red", self.model.food.get_postion())

```

Почему змейка

~~всегда в одном месте?~~

Почему змейка

не умеет ходить?

~~Где еда?~~



Учимся ходить или ползать

- Какие методы нужно реализовать и у каких классов?




```
class GameModel(GameSettings):
    def new_game(self): ...

    def update(self):
        self.snake.move()

        pg.display.flip()
        self.clock.tick(60)
```

```
vec2 = pg.math.Vector2
```

```
class Snake:
    def __init__(self, size):
        self.size = size
        self.head = pg.rect.Rect(0, 0, size - 2, size - 2)
        self.direction = vec2(self.size, 0)

    def move(self):
        self.head.move_ip(self.direction)

    def set_position(self, position):
        self.head.center = position

    def get_head_postion(self):
        return self.head
```

Слишком быстро!
Змейки так не ползают

Ограничиваем скорость

```
class GameSettings:
    def __init__(self):
        self.WINDOW_SIZE = 750 # можно 1000
        self.TILE_SIZE = 50
        self.screen = pg.display.set_mode([self.WINDOW_SIZE] * 2)
        self.clock = pg.time.Clock()

    self.step_delay = 150 # мсек. Определяет скорость движения змейки
    self.time = 0
```

```
class GameModel(GameSettings):
    def new_game(self):
        self.food = Food(self.TILE_SIZE)
        self.snake = Snake(self.TILE_SIZE)
        self.set_random_snake_position()
        self.set_random_food_position()

    def update_snake(self):
        self.snake.move()

    def update(self):
        if self.delta_time():
            self.update_snake()

pg.display.flip()
self.clock.tick(60)
```

Обработка нажатия клавиш


```
from enum import Enum
import pygame as pg

vec2 = pg.math.Vector2
```

```
class MovingDirections(Enum):
    left = "left"
    right = "right"
    up = "up"
    down = "down"
```

```
class Snake:
    def __init__(self, size):
        self.size = size
        self.head = pg.rect.Rect(0, 0, size - 2, size - 2)
        self.direction = vec2(0, 0)
        self.locked_direction = None

    def move(self): ...

    def set_position(self, position): ...

    def get_head_postion(self): ...

    def change_moving_direction(self, new_direction: MovingDirections):
        if self.locked_direction == new_direction:
            return

        if new_direction == MovingDirections.up:
            self.direction = vec2(0, -self.size)
            self.locked_direction = MovingDirections.down
        elif new_direction == MovingDirections.down:
            self.direction = vec2(0, self.size)
            self.locked_direction = MovingDirections.up
        elif new_direction == MovingDirections.left:
            self.direction = vec2(-self.size, 0)
            self.locked_direction = MovingDirections.right
        elif new_direction == MovingDirections.right:
            self.direction = vec2(self.size, 0)
            self.locked_direction = MovingDirections.left
```

```
class GameController(GameSettings):
    def __init__(self, model: GameModel):
        GameSettings.__init__(self)
        self.model = model

    def check_events(self):
        for event in pg.event.get():
            if event.type == pg.QUIT:
                pg.quit()
                sys.exit()
            if event.type == pg.KEYDOWN:
                self.change_snake_direction(event)
```

```
    def change_snake_direction(self, event):
        if event.key == pg.K_w:
            self.model.change_snake_moving_direction(MovingDirections.up)
        elif event.key == pg.K_s:
            self.model.change_snake_moving_direction(MovingDirections.down)
        elif event.key == pg.K_a:
            self.model.change_snake_moving_direction(MovingDirections.left)
        elif event.key == pg.K_d:
            self.model.change_snake_moving_direction(MovingDirections.right)
```

```
class GameModel(GameSettings):
    def new_game(self): ...

    def update_snake(self): ...

    def update(self): ...

    def change_snake_moving_direction(self, direction: MovingDirections):
        self.snake.change_moving_direction(direction)
```

Змейка ходит, но не кушает


```
class GameModel(GameSettings):
    def new_game(self): ...

    def update_snake(self): ...

    def update(self):
        if self.delta_time():
            self.update_snake()

        self.check_food()

        pg.display.flip()
        self.clock.tick(60)

    def change_snake_moving_direction(self, direction: MovingDirections): ...

    def delta_time(self): ...

    def get_random_position(self): ...

    def set_random_snake_position(self): ...

    def set_random_food_position(self):
        position = self.get_random_position()
        self.food.set_position(position)

    def check_food(self):
        if self.snake.get_head_postion() == self.food.get_postion():
            self.snake.increase_snake_length()
            self.set_random_food_position()
```

```
class Snake:
    def __init__(self, size):
        self.size = size
        self.head = pg.rect.Rect(0, 0, size - 2, size - 2)

        self.direction = vec2(0, 0)
        self.locked_direction = None

        self.length = 1
        self.segments = []

    def increase_snake_length(self):
        self.length += 1

    def move(self):
        self.head.move_ip(self.direction)
        self.segments.append(self.head.copy())
        # обрезаем змейку до ее длины
        self.segments = self.segments[-self.length :]

    def get_segments(self):
        return self.segments
```

```
class GameVeiw(GameSettings):
    def __init__(self, model: GameModel, controller: GameController):
        GameSettings.__init__(self)
        self.model = model
        self.controller = controller

    def draw_grid(self): ...

    def draw(self):
        # рисуем поле
        self.screen.fill("black")
        self.draw_grid()
        # рисуем змейку
        for segment in self.model.snake.get_segments():
            pg.draw.rect(self.screen, "green", segment)
        pg.draw.rect(self.screen, "red", self.model.food.get_postion())
```

Почти готово

Что осталось ?

- Проверка самопоедания змейки;
- Проверка на выход за границы экрана;
- Нет проверки для еды. Она может появиться в сегменте змейки;

```
class GameModel(GameSettings):
```

```
def check_snake_selfeating(self):  
    body = self.snake.get_segments()[:-1]  
    if self.snake.get_head_postion() in body:  
        self.new_game()
```

```
def check_borders(self):  
    snake_position = self.snake.get_head_postion()  
    if snake_position.left < 0 or snake_position.right > self.WINDOW_SIZE:  
        self.new_game()  
    elif snake_position.top < 0 or snake_position.bottom > self.WINDOW_SIZE:  
        self.new_game()
```

Будем выполнять

НОВЫЕ МЕТОДЫ

в методе update

```
def update(self):  
    if self.delta_time():  
        self.update_snake()  
        self.check_food()  
        self.check_borders()  
        self.check_snake_selfeating()  
  
    pg.display.flip()  
    self.clock.tick(60)
```


Исправим метод set_random_food_position

```
def set_random_food_position(self):  
    position = tuple(self.get_random_position())  
    segments = self.snake.get_segments()  
    for segment in segments:  
        if position == segment.center:  
            self.set_random_food_position()  
            return  
  
    self.food.set_position(position)
```


Спасибо за внимание!