



Программирован ие PyGame

ПЛАТФОРМЕР

Начало

Платформер (platformer)— жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня.

Наверное, не только игры, да и все приложения, использующие pygame начинаются примерно так:

```
import pygame
from pygame import *
#Объявляем переменные
SC_WIDTH = 800 #Ширина создаваемого окна
SC_HEIGHT = 640 # Высота
DISPLAY = (SC_WIDTH, SC_HEIGHT) # Группируем ширину и высоту в одну
переменную
BG_COLOR = "#004400" #Задаем задний фон
```



Основная функция main(), в которой будет осуществляться запуск программы и подпрограмм

```
def main():  
    init() # Инициация PyGame, обязательная строчка  
    screen = display.set_mode(DISPLAY) # Создаем окошко  
    display.set_caption("Super Game Name") # Пишем в шапку  
    bg = Surface((SC_WIDTH, SC_HEIGHT)) # Создание видимой поверхности, которую  
    будем использовать как фон  
    bg.fill(Color(BG_COLOR)) # Заливаем поверхность сплошным цветом  
    flag = True # Маркер выхода из основного цикла  
    while flag: # Основной цикл программы  
        for e in event.get(): # Обрабатываем события  
            if e.type == QUIT: # Событие нажатия на крестик в окне  
                flag = False  
            screen.blit(bg, (0,0)) # Каждую итерацию необходимо всё перерисовывать  
        display.update() # обновление и вывод всех изменений на экран  
        quit() # Выход из программы  
if __name__ == "__main__": # Запуск функции main()  
    main()
```



Уровень

А как без него? Под словом «уровень» будем подразумевать ограниченную область виртуального двумерного пространства, заполненную всякой — всячиной, и по которой будет передвигаться наш персонаж.

Для построения уровня создадим двумерный массив m на n . Каждая ячейка (m,n) будет представлять из себя прямоугольник. Прямоугольник может в себе что-то содержать, а может и быть пустым. Мы в прямоугольниках будем рисовать платформы.

```
PL_WIDTH = 32  
PL_HEIGHT = 32  
PL_COLOR = "#FF6262"
```



Отрисовка

В функцию main надо подгрузить наш уровень:

```
bg.fill(Color(BG_COLOR))  
Level=level1  
flag = True
```



Отрисовка

В основной цикл добавляем следующее:

```
screen.blit(bg, (0,0))
x=y=0 # координаты
for row in level: # вся строка
    for col in row: # каждый символ
        if col == "-":
            # создаем блок, заливаем его цветом и рисуем его
            pf = Surface((PL_WIDTH,PL_HEIGHT))
            pf.fill(Color(PL_COLOR))
            screen.blit(pf,(x,y))
            x += PL_WIDTH #блоки платформы ставятся на ширине блоков
            y += PL_HEIGHT #то же самое и с высотой
            x = 0 #на каждой новой строчке начинаем с нуля
display.update()
```



ОЛЕГ,
ГДЕ?
МАКЕТ?

Т.е. Мы перебираем двумерный массив `level`, и, если находим символ «-», то по координатам ($x * PLATFORM_WIDTH$, $y * PLATFORM_HEIGHT$), где x, y — индекс в массиве `level`

Персонаж

Просто кубики на фоне — это очень скучно. Нам нужен наш персонаж, который будет бегать и прыгать по платформам.

Создаём класс нашего героя. Для удобства, будем держать нашего персонажа в отдельном файле **player.py**

```
from pygame import *
MOVE_SPEED = 7
WIDTH = 22
HEIGHT = 32
COLOR = "#888888"
class Player(sprite.Sprite):
    def __init__(self, x, y):
        sprite.Sprite.__init__(self)
        self.xvel = 0 # скорость перемещения. 0 - стоять на месте
        self.startX = x # Начальная позиция X, пригодится когда будем
переигрывать уровень
        self.startY = y
        self.image = Surface((WIDTH,HEIGHT))
        self.image.fill(Color(COLOR))
        self.rect = Rect(x, y, WIDTH, HEIGHT) # прямоугольный объект
```



Персонаж

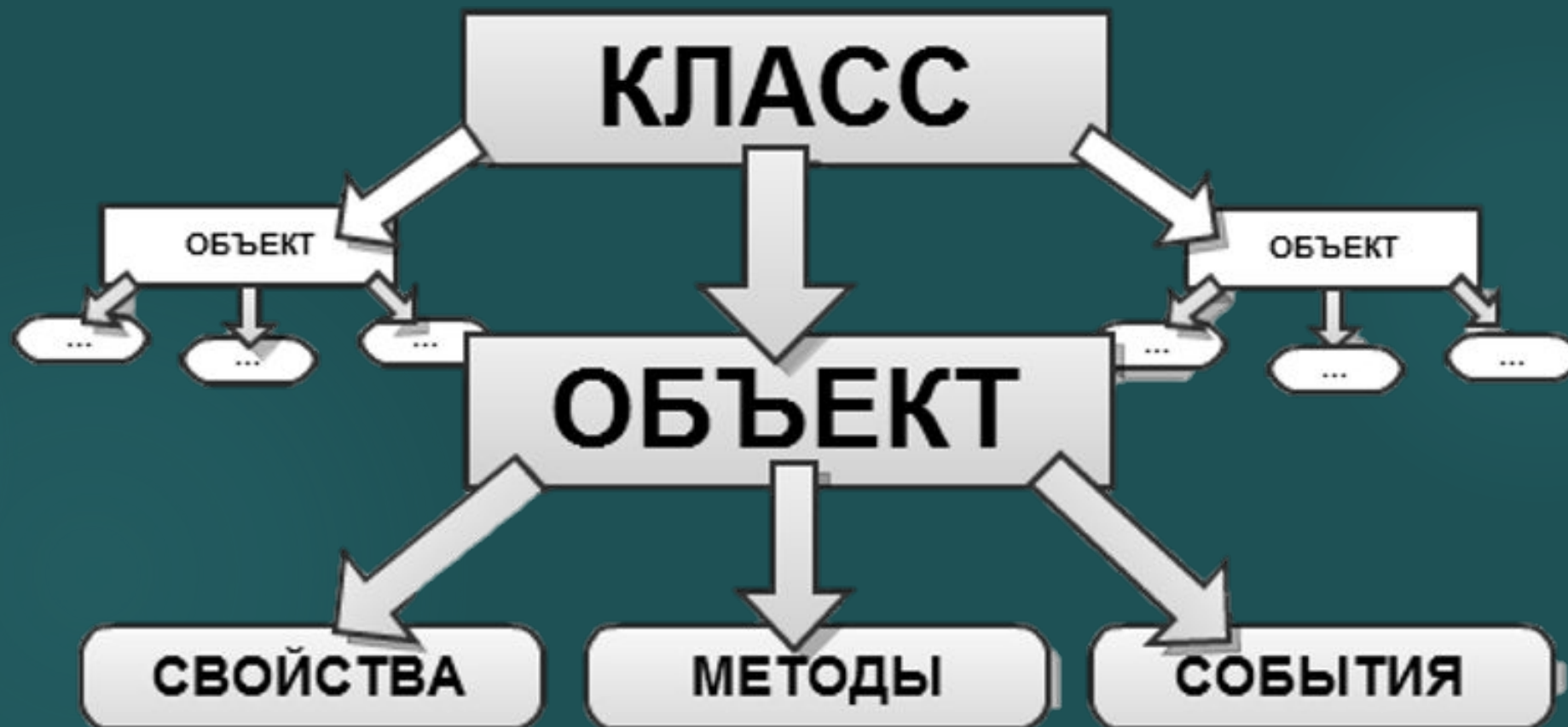
Добавляем нашему классу 2 метода.

И рассмотрим, что же такое класс и его методы.



```
def update(self, left, right):
    if left:
        self.xvel = -MOVE_SPEED # Лево = x- n
    if right:
        self.xvel = MOVE_SPEED # Право = x + n
    if not(left or right): # стоим, когда нет указаний идти
        self.xvel = 0
    self.rect.x += self.xvel # переносим свои положение на xvel
def draw(self, screen): # Выводим себя на экран
    screen.blit(self.image, (self.rect.x,self.rect.y))
```

Объектно-ориентированное программирование



Три столпа ООП

Принципы объектно-ориентированного программирования

инкапсуляция

Объект описывается как единое целое вместе со своими св-вами и действиями. Св-ва можно изменить только с помощью действий этого же объекта.



наследование

Описание объекта/класса включает в себя свойства и действия. При описании объекта, являющегося разновидностью класса можно не повторять описание св-в и действий, общих для класса.

Четырёхугольники



Квадрат



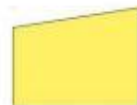
Прямоугольник



Ромб



Трапеция



Четырёхугольник

полиморфизм

В программе все должно иметь свои имена. По умолчанию действия разных объектов должны иметь разные имена. В объектном программировании полиморфизм позволяет давать аналогичным действиям разных объектов одинаковые имена.



Персонаж

Добавим нашего героя в основную часть программы.
Перед определением уровня добавим определение героя и переменные его перемещения..

```
hero = Player(55,55) # создаем героя по (x,y) координатам  
left = right = False # по умолчанию — стоим
```

В проверку событий добавим следующее:

```
if e.type == KEYDOWN and e.key == K_LEFT:  
    left = True  
if e.type == KEYUP and e.key == K_LEFT:  
    left = False  
if e.type == KEYDOWN and e.key == K_RIGHT:  
    right = True  
if e.type == KEYUP and e.key == K_RIGHT:  
    right = False
```

Персонаж

Само передвижение вызывается так: (добавляем после перерисовки фона и платформ)

```
hero.update(left, right) # передвижение  
hero.draw(screen) # отображение
```

Чтобы скорость перемещения блока могли контролировать мы, добавляем после определения уровня

```
timer = pygame.time.Clock()
```

И в начало основного цикла добавим следующее:

```
timer.tick(60)
```

Запускаем!

ЗАВИС В ВОЗДУХЕ

Да, наш герой в безвыходном положении, он завис в воздухе.
Добавим гравитации и возможности прыгать.
И так, работаем в файле **player.py**
Добавим еще констант

```
JUMP_POWER = 10  
GRAVITY = 0.35 # Сила, которая будет тянуть нас вниз
```

В метод `_init_` добавляем строки:

```
self.yvel = 0 # скорость вертикального перемещения  
self.onGround = False # На земле ли я?
```



ЗАВИС В ВОЗДУХЕ

Добавляем входной аргумент в метод **update**

```
def update(self, left, right, up):
```

И в начало метода добавляем:

```
if up:  
    if self.onGround: # прыгаем, только когда можем оттолкнуться от земли  
        self.yvel = -JUMP_POWER
```

И перед строчкой **self.rect.x += self.xvel**

Добавляем

```
if not self.onGround:  
    self.yvel += GRAVITY  
self.onGround = False; # Мы не знаем, когда мы на земле((  
self.rect.y += self.yvel
```



Завис в воздухе. Основная часть программы

После строчки `left = right = False`
Добавим переменную `up`

```
up = False
```

В проверку событий добавим

```
if e.type == KEYDOWN and e.key == K_UP:  
    up = True  
if e.type == KEYUP and e.key == K_UP:  
    up = False
```

И изменим вызов метода `update`, добавив новый аргумент `up`:

```
hero.update(left, right)
```

на

```
hero.update(left, right, up)
```



Встань обеими ногами на землю свою.

Как узнать, что мы на земле или другой твердой поверхности? Ответ очевиден — использовать проверку на пересечение, но для этого изменим создание платформ. Создадим еще один файл **blocks.py**, и перенесем в него описание платформы.

```
PL_WIDTH = 32  
PL_HEIGHT = 32  
PL_COLOR = "#FF6262"
```

Дальше создадим класс, наследуясь от **pygame.sprite.Sprite**

```
class Platform(sprite.Sprite):  
    def __init__(self, x, y):  
        sprite.Sprite.__init__(self)  
        self.image = Surface((PL_WIDTH, PL_HEIGHT))  
        self.image.fill(Color(PL_COLOR))  
        self.rect = Rect(x, y, PL_WIDTH, PL_HEIGHT)
```



Встань обеими ногами на землю свою.

В основной файле произведем изменения, перед описанием массива **level** добавим

```
allSpr = sprite.Group() # Все объекты
platforms = [] # то, во что мы будем врезаться или опираться
allSpr.add(hero)
```

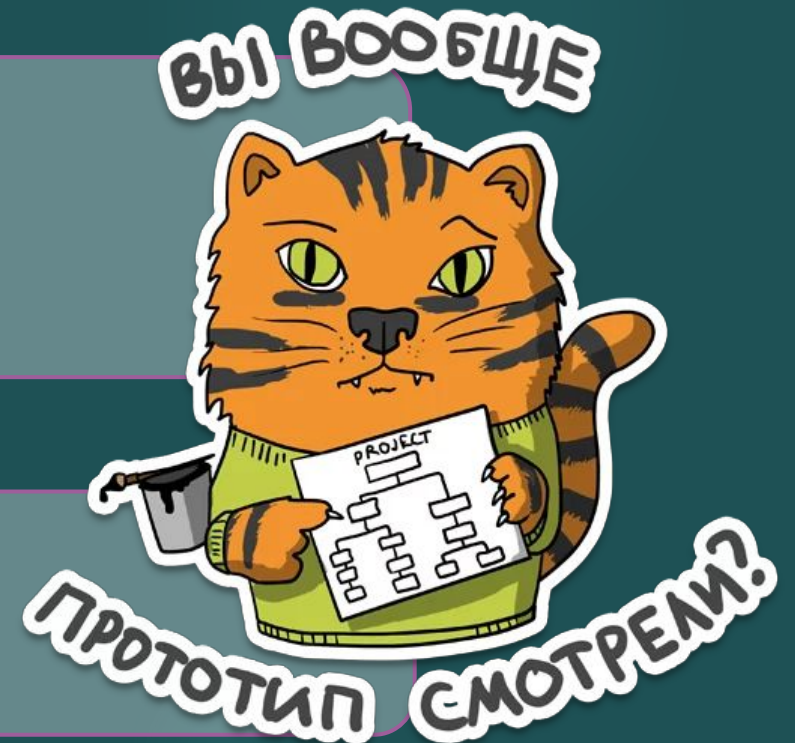
Группа спрайтов **allSpr** будем использовать для отображения всех элементов этой группы.

Массив **platforms** будем использовать для проверки на пересечение с платформой. Далее, блок

```
if col == "-":
    #создаем блок, заливаем его цветом и рисуем его
    pf = Surface((PLATFORM_WIDTH,PLATFORM_HEIGHT))
    pf.fill(Color(PLATFORM_COLOR))
    screen.blit(pf,(x,y))
```

Заменяем на

```
if col == "-":
    pf = Platform(x,y)
    allSpr.add(pf)
    platforms.append(pf)
```



Встань обеими ногами на землю свою.

Дальше, весь код генерации уровня выносим из цикла.

И так же строчку
`hero.draw(screen)` # отображение
Заменим на

```
allSpr.draw(screen) # отображение всего
```



Встань обеими ногами на землю свою.

Работаем в файле **player.py**

Удаляем метод **draw**, он нам больше не нужен. И добавляем новый метод **collide**

```
def collide(self, xvel, yvel, platforms):
    for p in platforms:
        if sprite.collide_rect(self, p): # если есть пересечение платформы с игроком
            if xvel > 0: # если движется вправо
                self.rect.right = p.rect.left # то не движется вправо
            if xvel < 0: # если движется влево
                self.rect.left = p.rect.right # то не движется влево
            if yvel > 0: # если падает вниз
                self.rect.bottom = p.rect.top # то не падает вниз
                self.onGround = True # и становится на что-то твердое
                self.yvel = 0 # и энергия падения пропадает
            if yvel < 0: # если движется вверх
                self.rect.top = p.rect.bottom # то не движется вверх
                self.yvel = 0 # и энергия прыжка пропадает
```



Встань обеими ногами на землю свою.

Работаем в файле **player.py**

Ну, и для того, что бы это всё происходило, необходимо вызывать этот метод.

Изменим число аргументов для метода **update**, теперь он выглядит так:
`update(self, left, right, up, platforms)`

И не забудьте изменить его вызов в основном файле.

И строчки

```
self.rect.y += self.yvel  
self.rect.x += self.xvel # переносим свои положение на xvel
```

Заменяем на

```
self.rect.y += self.yvel  
self.collide(0, self.yvel, platforms)  
self.rect.x += self.xvel # переносим свои положение на xvel  
self.collide(self.xvel, 0, platforms)
```



Больше, нужно больше места

Ограничение в размере окна мы преодолеем созданием динамической камеры.

Для этого создадим класс **Camera**

```
class Camera(object):
    def __init__(self, camera_func, width, height):
        self.camera_func = camera_func
        self.state = Rect(0, 0, width, height)
    def apply(self, target):
        return target.rect.move(self.state.topleft)
    def update(self, target):
        self.state = self.camera_func(self.state, target.rect)
```



Больше, нужно больше места

Далее, добавим начальное конфигурирование камеры

```
def camera_configure(camera, target_rect):  
    l, t, _, _ = target_rect  
    _, _, w, h = camera  
    l, t = -l+SC_WIDTH / 2, -t+SC_HEIGHT / 2  
    l = min(0, l) # Не движемся дальше левой границы  
    l = max(-(camera.width-SC_WIDTH), l) # Не движемся дальше правой границы  
    t = max(-(camera.height-SC_HEIGHT), t) # Не движемся дальше нижней границы  
    t = min(0, t) # Не движемся дальше верхней границы  
    return Rect(l, t, w, h)
```



Больше, нужно больше места

Создадим экземпляр камеры, добавим перед основным циклом:

```
total_level_width = len(level[0])*PL_WIDTH # Вычисляем фактическую ширину уровня
total_level_height = len(level)*PL_HEIGHT # высоту
camera = Camera(camera_configure, total_level_width, total_level_height)
```

Заменяем строчку
`allSpr.draw(screen)` # отображение
На

```
camera.update(hero)
for e in allSpr:
    screen.blit(e.image, camera.apply(e))
```



Фу[у]! Движущийся прямоугольник — не красиво!

Давайте немного приукрасим нашу игру.

Начнем с платформ. Для этого в файле **blocks.py** сделаем небольшие изменения.

Заменяем заливку цветом на картинку, для этого строчку

```
self.image.fill(Color(PLATFORM_COLOR))
```

Заменяем на

```
self.image = image.load("*.png")
```

В файл с игроком добавляем константы для анимации и подключаем библиотеку `pygame`. Вместо `*.png` нужно добавить свои кадры анимации.

```
from pygame import *  
ANIMATION_DELAY = 0.1 # скорость смены кадров  
ANIMATION_RIGHT = [ "*.png", "*.png" ] # анимация движения вправо  
ANIMATION_LEFT = [ "*.png", "*.png" ] # анимация движения влево  
ANIMATION_JUMP_LEFT = [ "*.png", 0, 1 ] # прыжок влево  
ANIMATION_JUMP_RIGHT = [ "*.png", 0, 1 ] # прыжок вправо  
ANIMATION_JUMP = [ "*.png", 0, 1 ] # прыжок  
ANIMATION_STAY = [ "*.png", 0, 1 ] # исходная позиция
```



Фу[у]! Движущийся прямоугольник — не красиво!

Теперь добавим следующее в метод `__init__`

```
self.image.set_colorkey(Color(COLOR)) # делаем фон прозрачным
boltAnim = []
for anim in ANIMATION_RIGHT: # Анимация движения вправо
    boltAnim.append((anim, ANIMATION_DELAY))
self.boltAnimRight = pyganim.PygAnimation(boltAnim)
self.boltAnimRight.play()
boltAnim = []
for anim in ANIMATION_LEFT: # Анимация движения влево
    boltAnim.append((anim, ANIMATION_DELAY))
self.boltAnimLeft = pyganim.PygAnimation(boltAnim)
self.boltAnimLeft.play()
```



Фу[у]! Движущийся прямоугольник — не красиво!

Продолжаем заполнять метод `__init__`

```
self.boltAnimStay = pygame.Animation(ANIMATION_STAY)
self.boltAnimStay.play()
self.boltAnimStay.blit(self.image, (0, 0)) # По-умолчанию, стоим
self.boltAnimJumpLeft= pygame.Animation(ANIMATION_JUMP_LEFT)
self.boltAnimJumpLeft.play()
self.boltAnimJumpRight=
pygame.Animation(ANIMATION_JUMP_RIGHT)
self.boltAnimJumpRight.play()
self.boltAnimJump= pygame.Animation(ANIMATION_JUMP)
self.boltAnimJump.play()
```



Осталось в нужный момент показать нужную анимацию.

Добавим смену анимаций в метод **update**.

```
if up:
    if self.onGround: # прыгаем, только когда можем оттолкнуться от земли
        self.yvel = -JUMP_POWER
        self.image.fill(Color(COLOR))
        self.boltAnimJump.blit(self.image, (0, 0))
    if left:
        self.xvel = -MOVE_SPEED # Лево = x- n
        self.image.fill(Color(COLOR))
        if up: # для прыжка влево есть отдельная анимация
            self.boltAnimJumpLeft.blit(self.image, (0, 0))
        else:
            self.boltAnimLeft.blit(self.image, (0, 0))
```



Осталось в нужный момент показать нужную анимацию.

Осталось немного!

```
if right:
    self.xvel = MOVE_SPEED # Право = x + n
    self.image.fill(Color(COLOR))
    if up:
        self.boltAnimJumpRight.blit(self.image, (0, 0))
else:
    self.boltAnimRight.blit(self.image, (0, 0))
if not(left or right): # стоим, когда нет указаний идти
self.xvel = 0
    if not up:
        self.image.fill(Color(COLOR))
        self.boltAnimStay.blit(self.image, (0, 0))
```

