


PyGame — шпаргалка для использования

 waksoft.susu.ru/2019/04/24/pygame-shpargalka-dlja-ispolzovanija/

К ВВ

24/04/2019



Основные модули пакета Pygame

Модуль	Назначение
pygame.cdrom	Доступ к CD-приводам и управление ими
pygame.cursors	Загружает изображения курсора
pygame.display	Доступ к дисплею
pygame.draw	Рисует фигуры, линии и точки
pygame.event	Управление внешними событиями
pygame.font	Использует системные шрифты
pygame.image	Загружает и сохраняет изображение
pygame.joystick	Использует джойстики и аналогичные устройства
pygame.key	Считывает нажатия клавиш с клавиатуры
pygame.mixer	Загружает и воспроизводит мелодии
pygame.mouse	Управляет мышью
pygame.movie	Воспроизведение видеофайлов

pygame.music	Работает с музыкой и потоковым аудио
pygame.overlay	Доступ к расширенным видеоизображениям
pygame	Содержит функции Pygame высокого уровня
pygame.rect	Управляет прямоугольными областями
pygame.sndarray	Манипулирует звуковыми данными
pygame.sprite	Управление движущимися изображениями
pygame.surface	Управляет изображениями и экраном
pygame.surfarray	Манипулирует данными пикселей изображения
pygame.time	модуль pygame для управления временем и частотой кадров
pygame.transform	Изменение размера и перемещение изображений

Окно Pygame

```
# Подключение библиотеки PyGame
import pygame

# Инициализация PyGame
pygame.init()

# Окно игры: размер, позиция
gameScreen = pygame.display.set_mode((400, 300))

# Модуль os - позиция окна
import os
x = 100
y = 100
os.environ['Sp_VIDEO_WINDOW_POS'] = "%d,%d" % (x,y)

# Параметры окна
size = [500, 500]
screen = pygame.display.set_mode(size)
pygame.display.set_caption("Test drawings")
gameScreen.fill((0,0,255))
pygame.display.flip()
```

Цикл игры, выход из игры

```
# Цикл игры
runGame = True # флаг выхода из цикла игры
while runGame:
    # Отслеживание события: "заккрыть окно"
    for event in pygame.event.get():
        if event.type == pygame.QUIT: runGame = False

# Выход из игры:
pygame.quit()
```

Рисование базовых элементов модуль `pygame.draw`

<code>pygame.draw.rect</code>	нарисовать прямоугольную форму
<code>pygame.draw.polygon</code>	фигуру с любым количеством сторон
<code>pygame.draw.circle</code>	круг вокруг точки
<code>pygame.draw.ellipse</code>	нарисовать круглую форму внутри прямоугольника
<code>pygame.draw.arc</code>	нарисовать секцию эллипса
<code>pygame.draw.line</code>	нарисовать сегмент прямой линии
<code>pygame.draw.lines</code>	для рисования нескольких смежных отрезков
<code>pygame.draw.aaline</code>	рисовать тонкую линию
<code>pygame.draw.aalines</code>	нарисовать связанную последовательность сглаженных линий

```
rect(Surface, color, Rect, wiph=0) -> Rect
polygon(Surface, color, pointlist, wiph=0) -> Rect
circle(Surface, color, pos, radius, wiph=0) -> Rect
ellipse(Surface, color, Rect, wiph=0) -> Rect
arc(Surface, color, Rect, start_angle, stop_angle, wiph=1) -> Rect
line(Surface, color, start_pos, end_pos, wiph=1) -> Rect
lines(Surface, color, closed, pointlist, wiph=1) -> Rect
aaline(Surface, color, startpos, endpos, blend=1) -> Rect
aalines(Surface, color, closed, pointlist, blend=1) -> Rect
```

Загрузка изображения

```
# Модуль pygame.image позволяет загрузить изображение из файла и возвращает объект
# типа Surface.
pygame.image.load("путь к файлу" )

# загрузить новое изображение из файла
load(filename) -> Surface

# Загрузить изображение (путь к файлу для Windows)
myImage = pygame.image.load('images\\bg1.jpg')

# определить место размещения
myRect = (0,0,600,400)

# выгрузить объект Surface, который содержит загруженное из файла изображение
(myImage), в описанное место на экране (myRect)
screen.blit(myImage,myRect)
```

Объект Rect

pygame.Rect

Pygame использует объекты Rect для хранения и манипулирования прямоугольными областями. Rect может быть создан из комбинации значений слева, сверху, ширины и высоты. Rect также могут быть созданы из объектов python, которые уже являются Rect или имеют атрибут с именем «rect».

```
Rect(left, top, wiph, height) -> Rect
```

```
Rect((left, top), (wiph, height)) -> Rect
```

```
Rect(object) -> Rect
```

Методы работы с Rect

pygame.Rect.copy	Возвращает новый прямоугольник, имеющий ту же позицию и размер, что и оригинал.
pygame.Rect.move	Возвращает новый прямоугольник, перемещаемый данным смещением. Аргументы x и y могут быть любым целочисленным значением, положительным или отрицательным.
pygame.Rect.move_ip	То же, что и метод Rect.move (), но работает на месте.
pygame.Rect.inflate	увеличивать или уменьшать размер прямоугольника, на месте
pygame.Rect.inflate_ip	увеличивать или уменьшать размер прямоугольника, на месте
pygame.Rect.clamp	перемещает прямоугольник внутри другого
pygame.Rect.clamp_ip	перемещает прямоугольник внутри другого, на месте

<code>pygame.Rect.clip</code>	обрезает прямоугольник внутри другого
<code>pygame.Rect.union</code>	соединяет два прямоугольника в один
<code>pygame.Rect.union_ip</code>	соединяет два прямоугольника в один, на месте
<code>pygame.Rect.unionall</code>	объединение многих прямоугольников
<code>pygame.Rect.unionall_ip</code>	объединение многих прямоугольников, на месте
<code>pygame.Rect.fit</code>	изменить размер и переместить прямоугольник учитывая соотношение сторон
<code>pygame.Rect.normalize</code>	корректировать отрицательные размеры
<code>pygame.Rect.contains</code>	проверить, находится ли один прямоугольник внутри другого
<code>pygame.Rect.collidepoint</code>	проверить, находится ли точка внутри прямоугольника
<code>pygame.Rect.colliderect</code>	тест, пересекаются ли два прямоугольника
<code>pygame.Rect.collidelist</code>	проверить, пересекается ли хоть один прямоугольник в списке
<code>pygame.Rect.collidelistall</code>	пересекаются ли все прямоугольники в списке
<code>pygame.Rect.collidedict</code>	проверить, если один прямоугольник в словаре пересекается
<code>pygame.Rect.collidedictall</code>	пересекаются ли все прямоугольники в словаре

Обработка событий

Событие — это то, как Pygame сообщает о том, что что-то случилось за пределами кода программы. События создаются, например, при нажатии клавиш клавиатуры, мыши и размещаются в очереди, дожидаясь обработки.

Функция `get` в модуле `pygame.event` возвращает последнее событие, ожидающее в очереди и удаляет его из очереди.

Объект event

Модуль `pygame.event` для обработки очереди событий

<code>pygame.event.pump</code>	Если вы не используете другие функции событий в своей игре, вы должны вызвать <code>pygame.event.pump()</code> , чтобы позволить pygame обрабатывать внутренние действия
<code>pygame.event.get</code>	получает события из очереди
<code>pygame.event.poll</code>	получить одно событие из очереди

<code>pygame.event.wait</code>	ждёт одиночного события из очереди
<code>pygame.event.peek</code>	проверить, ждут ли очереди события определённого типа
<code>pygame.event.clear</code>	удалить все события из очереди
<code>pygame.event.event_name</code>	возвращает имя для типа события. Строка находится в стиле WordCap
<code>pygame.event.set_blocked</code>	проверяет, какие события не разрешены в очереди
<code>pygame.event.set_allowed</code>	проверяет, какие события разрешены в очереди
<code>pygame.event.get_blocked</code>	проверить, заблокирован ли тип события из очереди
<code>pygame.event.set_grab</code>	проверяет совместное использование устройств ввода с другими приложениями
<code>pygame.event.get_grab</code>	проверить, работает ли программа на устройствах ввода данных
<code>pygame.event.post</code>	поместить новое событие в очередь
<code>pygame.event.Event</code>	создать новый объект события
<code>pygame.event.EventType</code>	Объект Python, представляющий событие SDL. Экземпляры пользовательских событий создаются с вызовом функции <code>Event</code> . Тип <code>EventType</code> не может быть напрямую вызван. Экземпляры <code>EventType</code> поддерживают назначение и удаление атрибутов.

Pygame отслеживает все сообщения о событиях через очередь событий. Процедуры в этом модуле помогают управлять этой очередью событий. Входная очередь сильно зависит от модуля отображения (`display`) `pygame`. Если дисплей не был инициализирован и видеорежим не установлен, очередь событий не будет работать.

Существует множество способов доступа к очереди событий. Просто проверять существование событий, захватывать их непосредственно из стека.

Мышь

Модуль `pygame.mouse` для работы с мышью

<code>pygame.mouse.get_pressed</code>	получить состояние кнопок мыши
<code>pygame.mouse.get_pos</code>	получить позицию курсора мыши
<code>pygame.mouse.get_rel</code>	получить количество движений мыши
<code>pygame.mouse.set_pos</code>	установить позицию курсора мыши

<code>pygame.mouse.set_visible</code>	скрыть или показать курсор мыши
<code>pygame.mouse.get_focused</code>	проверяет, принимает ли дисплей ввод мыши
<code>pygame.mouse.set_cursor</code>	установить изображение для курсора мыши
<code>pygame.mouse.get_cursor</code>	получить изображение для курсора мыши

Функции мыши можно использовать для получения текущего состояния устройства мышь. Эти функции также могут изменять курсор мыши.

Когда режим отображения (display) установлен, очередь событий начнет принимать события мыши. Кнопки мыши генерируют события `pygame.MOUSEBUTTONDOWN` и `pygame.MOUSEBUTTONUP`, когда они нажимаются и отпускаются. Эти события содержат атрибут кнопки, указывающий, какая кнопка была нажата. Колесо мыши будет генерировать `pygame.MOUSEBUTTONDOWN` и `pygame.MOUSEBUTTONUP` события при прокрутке.

Когда колесо повернуто вверх, кнопка будет установлена на 4, вниз -5. Всякий раз, когда мышь перемещается, генерируется событие `pygame.MOUSEMOTION`. Движение мыши разбито на небольшие и точные события движения. По мере перемещения мыши многие события движения будут помещены в очередь. События движения мыши, которые неправильно очищены от очереди событий, являются основной причиной того, что очередь событий заполняется.

Пример. Нарисовать курсор под текущей позицией мыши.

```
x, y = pygame.mouse.get_pos()
x -= mouse_cursor.get_width()/2
y -= mouse_cursor.get_height()/2
screen.blit(mouse_cursor, (x, y))
```

Определить какая кнопка была нажата на мышке можно используя значение `event.button`:

```
1 - left click
2 - middle click
3 - right click
4 - scroll up
5 - scroll down
```

Координаты курсора при нажатии кнопки мыши находятся в `event.pos`.

Пример. Перемещать картинку курсором мыши.

```

import pygame, sys, time
from pygame.locals import *
pygame.init()
FPS=30
fpsClock=pygame.time.Clock()
width=500
height=500
mainSurface=pygame.display.set_mode((width,height),0,32)
pygame.display.set_caption('Keyb moves')
background=pygame.image.load('images//bg1.jpg')
sprite=pygame.image.load('images//pict2.gif')

# Place image to the center of mainSurface
image_pos = ((mainSurface.get_width() - sprite.get_width())/2,
(mainSurface.get_height() - sprite.get_height())/2)
doMove = False

# game loop
while True:
    fpsClock.tick(FPS) # frame rate
    mainSurface.blit(background,(0,0))

    # get all events from the queue
    for event in pygame.event.get():
        # loop events queue
        if event.type == QUIT:
            # window close X pressed
            pygame.quit()
            sys.exit()

        if event.type == KEYDOWN and event.key == K_ESCAPE:
            # ESC key pressed
            pygame.quit()
            sys.exit()

        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1: # левая кнопка мыши
                doMove = True
            if event.button == 3: # правая кнопка мыши
                image_pos = ((mainSurface.get_width() - sprite.get_width())/2,
(mainSurface.get_height() - sprite.get_height())/2)
                doMove = False

        if event.type == pygame.MOUSEBUTTONUP: doMove = False

        if event.type == MOUSEMOTION and doMove:
            image_pos = event.pos

    mainSurface.blit(sprite,image_pos)
    pygame.display.update()

```

Клавиатура

Модуль pygame.key

Этот модуль содержит функции для работы с клавиатурой. Очередь событий получает события `pygame.KEYDOWN` и `pygame.KEYUP` при нажатии и отпускании клавиш клавиатуры.

Оба события имеют ключевой атрибут, который представляет собой целочисленный идентификатор, представляющий каждую клавишу на клавиатуре. Событие `pygame.KEYDOWN` имеет дополнительные атрибуты: `unicode` и `scancode`. `unicode` представляет собой одну символьную строку, которая соответствует введённому символу. `Scancode` представляет собой код для конкретной платформы.

Получить код клавиши:

```
pressed_keys = pygame.key.get_pressed()
if pressed_keys[K_SPACE]:
# Space key has been pressed
fire()
```

Существует много клавиатурных констант, они используются для представления клавиш на клавиатуре. Ниже приведен список всех клавиатурных констант:

KeyASCII	ASCII	CommonName
K_BACKSPACE	\b	backspace
K_TAB	\t	tab
K_CLEAR		clear
K_RETURN	\r	return
K_PAUSE		pause
K_ESCAPE	^[escape
K_SPACE		space
K_EXCLAIM	!	exclaim
K_QUOTEDBL	«	quotedbl
K_HASH	#	hash
K_DOLLAR	\$	dollar
K_AMPERSAND	&	ampersand
K_QUOTE		quote
K_LEFTPAREN	(leftparenthesis
K_RIGHTPAREN)	rightparenthesis

K_ASTERISK	*	asterisk
K_PLUS	+	plussign
K_COMMA	,	comma
K_MINUS	—	minussign
K_PERIOD	.	period
K_SLASH	/	forwardslash
K_0	0	0
K_1	1	1
K_2	2	2
K_3	3	3
K_4	4	4
K_5	5	5
K_6	6	6
K_7	7	7
K_8	8	8
K_9	9	9
K_COLON	:	colon
K_SEMICOLON	;	semicolon
K_LESS		less-thansign
K_EQUALS	=	equalssign
K_GREATER	>	greater-thansign
K_QUESTION	?	questionmark
K_AT	@	at
K_LEFTBRACKET	[leftbracket
K_BACKSLASH	\	backslash
K_RIGHTBRACKET]	rightbracket
K_CARET	^	caret
K_UNDERSCORE	_	underscore

K_BACKQUOTE	`	grave
K_a	a	a
K_b	b	b
K_c	c	c
K_d	d	d
K_e	e	e
K_f	f	f
K_g	g	g
K_h	h	h
K_i	i	i
K_j	j	j
K_k	k	k
K_l	l	l
K_m	m	m
K_n	n	n
K_o	o	o
K_p	p	p
K_q	q	q
K_r	r	r
K_s	s	s
K_t	t	t
K_u	u	u
K_v	v	v
K_w	w	w
K_x	x	x
K_y	y	y
K_z	z	z
K_DELETE		delete

K_KP0		keypad0
K_KP1		keypad1
K_KP2		keypad2
K_KP3		keypad3
K_KP4		keypad4
K_KP5		keypad5
K_KP6		keypad6
K_KP7		keypad7
K_KP8		keypad8
K_KP9		keypad9
K_KP_PERIOD	.	keypadperiod
K_KP_DIVIDE	/	keypaddivide
K_KP_MULTIPLY	*	keypadmultiply
K_KP_MINUS	—	keypadminus
K_KP_PLUS	+	keypadplus
K_KP_ENTER	\r	keypadenter
K_KP_EQUALS	=	keypadequals
K_UP		uparrow
K_DOWN		downarrow
K_RIGHT		rightarrow
K_LEFT		leftarrow
K_INSERT		insert
K_HOME		home
K_END		end
K_PAGEUP		pageup
K_PAGEDOWN		pagedown
K_F1		F1
K_F2		F2

K_F3	F3
K_F4	F4
K_F5	F5
K_F6	F6
K_F7	F7
K_F8	F8
K_F9	F9
K_F10	F10
K_F11	F11
K_F12	F12
K_F13	F13
K_F14	F14
K_F15	F15
K_NUMLOCK	numlock
K_CAPSLOCK	capslock
K_SCROLLOCK	scrollock
K_RSHIFT	rightshift
K_LSHIFT	leftshift
K_RCTRL	rightcontrol
K_LCTRL	leftcontrol
K_RALT	rightalt
K_LALT	leftalt
K_RMETA	rightmeta
K_LMETA	leftmeta
K_LSUPER	leftWindowskey
K_RSUPER	rightWindowskey
K_MODE	modeshift
K_HELP	help

K_PRINT	printscreen
K_SYSREQ	sysrq
K_BREAK	break
K_MENU	menu
K_POWER	power
K_EURO	Euro

Направленное движение с помощью клавиш

Можно перемещать изображение на экране с клавиатуры, назначая клавиши для перемещений: вверх, вниз, влево, вправо.

Создать картинку, например:

```
sprite=pygame.image.load('images//pict2.gif')
```

Проверить очередь событий:

```
pygame.event.get()
```

Проверить, является ли полученное событие нажатием на клавиши со стрелками:

```
event.type == KEYDOWN
```

Если — да, то получить код нажатой клавиши и сформировать новые координаты для картинки:

```
spritex, spritey
```

И нарисовать картинку в новом месте:

```
blit(sprite, (spritex, spritey))
```

```

import pygame, sys, time
from pygame.locals import *

pygame.init()

FPS=30
fpsClock=pygame.time.Clock()
width=500
height=500
mainSurface=pygame.display.set_mode((width,height),0,32)
pygame.display.set_caption('Keyb moves')

background=pygame.image.load('images//bg1.jpg')

sprite=pygame.image.load('images//pict2.gif') # Create moving image

# Place image to the center of mainSurface
spritex=(mainSurface.get_width() - sprite.get_width())/2
spritey=(mainSurface.get_height() - sprite.get_height())/2
direction=False

# --->
def move(direction, spritex, spritey):
    # This function moves recalculates new image coordinates
    if direction:
        if direction == K_UP:
            spritey-=5
        elif direction == K_DOWN:
            spritey+=5
        if direction == K_LEFT:
            spritex-=5
        elif direction == K_RIGHT:
            spritex+=5
    return spritex, spritey
# --->

# game loop
while True:
    fpsClock.tick(FPS) # define frame rate
    mainSurface.blit(background,(0,0))
    mainSurface.blit(sprite,(spritex,spritey))

    # get all events from the queue
    for event in pygame.event.get():
        # loop events queue, remember in 'direction' pressed key code
        if event.type==QUIT:
            # window close X pressed
            pygame.quit()
            sys.exit()
        if event.type==KEYDOWN and event.key == K_ESCAPE:
            # ESC key pressed
            pygame.quit()
            sys.exit()
        if event.type == KEYDOWN: direction = event.key # Other key pressed
        if event.type == KEYUP: direction = False # Key released

```

```
# calculate new image position
spritex, spritey = move(direction, spritex, spritey)

pygame.display.update()
```

Объект Surface

`pygame.Surface` — объект `pygame` для представления изображений

`Surface((width, height), flags=0, depth=0, masks=None) -> Surface`

`Surface((width, height), flags=0, Surface) -> Surface`

Наложение поверхностей, прозрачность.


```

#-*-coding: utf-8-*-
import pygame
pygame.init()

# Create base surface
screen = pygame.display.set_mode((500,500))

# Create new surface
surface1 = pygame.Surface((150,150))
surface1.fill((255,0,0))
surface1.set_alpha(150)
#
surface2 = pygame.Surface((100,100))
surface2.fill((255,255,0))
surface2.set_alpha(100)

# Create image
bgImg = pygame.image.load("images//bg3.jpg")
bgImg = pygame.transform.scale(bgImg,(500,500))
#
pict1 = pygame.image.load("images//pict1.jpg")
pict1 = pygame.transform.scale(pict1,(130,130))
pict1.set_alpha(100)
#
pict2 = pygame.image.load("images//pict2.gif")
pict2 = pygame.transform.scale(pict2,(50,50))

clock = pygame.time.Clock()
running = 1
dX = dY = 1
x = y = 0

while running:
    clock.tick(50)
    event = pygame.event.poll()
    if event.type == pygame.QUIT: running = 0
    x += 8 * dX
    y += 6 * dY
    if (y<0 or y>= (screen.get_height() - pict2.get_height())) :
        dY *= -1
    if (x<0 or x>= (screen.get_width() - pict2.get_width())) :
        dX *= -1

    screen.blit(bgImg,(0,0))
    surface1.blit(pict1,(0,0))
    screen.blit(surface1,(20,50))
    screen.blit(surface2,(150,150))
    screen.blit(pict2,(x,y))

    pygame.display.flip()

pygame.quit()

```

Управление временем

Модуль `pygame.time` содержит объект `Clock`, который можно использовать для отслеживания

времени. Чтобы создать объект типа: время, вызывается конструктор `pygame.time.Clock`:

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

Когда создан объект `clock`, можно вызвать его функцию `tick` один раз за кадр, которая возвращает время, прошедшее со времени предыдущего вызова в миллисекундах:

```
time_passed = clock.tick ()
```

Функция `tick` может использовать необязательный параметр для установления максимальной частоты кадров. Этот параметр нужен, если игра запущена на рабочем компьютере и необходимо контролировать, чтобы она не использовала всю его вычислительная мощность на 100%:

```
# Игра будет работать со скоростью не более 30 кадров в секунду
```

```
time_passed = clock.tick (30)
```

Звуки

Для управления звуком используется модуль `pygame.mixer`. Он отвечает за любые действия со звуками.

Загружаем звуковой файл в формате `*.wav`

```
sound = pygame.mixer.Sound("sound.wav")
```

(загружаем до игрового цикла, т.к. это очень долгая операция)

Проигрываем звук

```
sound.play()
```

Столкновения (collisions)

При написании игр часто возникает необходимость проверять взаимное расположение объектов на экране, отслеживать моменты их столкновений, пересечений.

Эта задача может быть реализована разными способами.

Например, используя объект `Rect`

```

#-*-coding:utf-8-*-
import pygame, sys, time
from pygame.locals import *

pygame.init()
FPS          = 30
fpsClock     = pygame.time.Clock()
winWidth     = 500
winHeight    = 500
mainSurface  = pygame.display.set_mode((winWidth,winHeight),0,32)
pygame.display.set_caption('Collisions test')
background   = pygame.image.load('images//bg1.jpg')

# Рисуем неподвижные компоненты на поверхности background
# Места расположения границ
border1Rect  = pygame.Rect(0,0,100,500)
border2Rect  = pygame.Rect(0,150,300,50)
# границы
border1 = pygame.draw.rect(background, (0,0,0), border1Rect, 0)
border2 = pygame.draw.rect(background, (0,0,0), border2Rect, 0)
# Записываем их в массив
borders = []
borders.append((border1,border1Rect))
borders.append((border2,border2Rect))
# Подвижный блок
blockWidth   = 50
blockHeight  = 50
blockStep    = 1
blockColor   = (255,0,0)
# Начальные координаты подвижного блока - по центру
blockX = (mainSurface.get_width() - blockWidth)/2
blockY = (mainSurface.get_height() - blockHeight)/2
blockPosition = (blockX, blockY)
direction = False

# описание функции --->
def newPosition(dirFlag, pos):
    (x,y) = pos
    # функция пересчитывает координаты для подвижного объекта
    if dirFlag:
        if dirFlag == K_UP:
            y -= blockStep
        elif dirFlag == K_DOWN:
            y += blockStep
        if dirFlag == K_LEFT:
            x -= blockStep
        elif dirFlag == K_RIGHT:
            x += blockStep
    return (x, y)
# --->

while True:    # Игра - начало
    # Частота обновления экрана
    fpsClock.tick(FPS)
    # Рисуем неподвижные компоненты
    mainSurface.blit(background, (0,0))

```

```

# Рисуем подвижный компонент
blockRect = pygame.Rect(blockPosition, (blockWidth, blockHeight))
block = pygame.draw.rect(mainSurface, blockColor, blockRect, 1)
# просматриваем очередь событий
for event in pygame.event.get():
    # loop events queue, remember in 'direction' pressed key code
    if event.type == QUIT:
        # window close X pressed
        pygame.quit()
        sys.exit()
    if event.type == KEYDOWN and event.key == K_ESCAPE:
        # ESC key pressed
        pygame.quit()
        sys.exit()
    if event.type == KEYDOWN: direction = event.key # Other key pressed
    if event.type == KEYUP: direction = False # Key released
# Сохраняем старые координаты подвижного блока
savedPosition = blockPosition
# Рассчитываем новые координаты подвижного блока
blockPosition = newPosition(direction, blockPosition)
# Проверяем их корректность
for border in borders:
    testRect = pygame.Rect(blockPosition, (blockWidth, blockHeight))
    if testRect.colliderect(border[1]):
        print ("Столкновение !")
        # Возвращаем старые координату
        blockPosition = savedPosition
    else: print ("ok")
pygame.display.update()
# Игра - конец

```

Или используя поверхности — surface

```

#-*-coding:utf-8-*-
import pygame, sys, time
from pygame.locals import *

# Блоки-ограничители рисуются как отдельные поверхности
# Между ними курсором перемещаем мячик

FPS = 30 # кадров в сек
# Размеры окна игры
width = 500
height = 500
# Заголовок окна игры
title = "Collisions detection test"
# Сообщение в консоль игры
info = "Нет столкновений \n"
# Переменная - индикатор движения
direction = False
# Шаг движения
myStep = 2
pygame.mixer.init()
pygame.mixer.music.load("voice-prompts-reaction-reaction-1-child-3-yrs-oops-human-voice-kid-speak-talk.mp3")

pygame.init()
fpsClock = pygame.time.Clock()
mainSurface=pygame.display.set_mode((width,height),0,32)
pygame.display.set_caption(title)
background=pygame.image.load('images//bg1.jpg') # Фон
sprite = pygame.image.load('images//pict2.gif') # Подвижная картинка

# Начальные координаты подвижной картинки
# Важно, чтобы в начале работы она не перекрывала ни один блок
spriteX=mainSurface.get_width() - sprite.get_width()
spriteY=mainSurface.get_height() - sprite.get_height()

# Неподвижные блоки - все сохраняем в одном списке
# Структура списка blocks:
# каждый элемент - пара значений: (поверхность, область её размещения)
# blocks[0] - первый элемент списка, blocks[0][0] - surface первого
элемента, blocks[0][1] - Rect первого элемента
blocks = []
block1 = pygame.Surface((200,200))
block1.set_alpha(80)
block1.fill((255,0,0))
block1Rect = pygame.Rect(0,0,block1.get_width(), block1.get_height())
blocks.append((block1,block1Rect))
#
block2 = pygame.Surface((100,200))
block2.fill((0,255,0))
block2Rect = pygame.Rect(400,0,block2.get_width(), block2.get_height())
blocks.append((block2,block2Rect))
#
block3 = pygame.Surface((300,100))
block3.fill((0,0,255))
block3Rect = pygame.Rect(0,350,block3.get_width(), block3.get_height())
blocks.append((block3,block3Rect))

```

```
# закончили с описанием 3-х блоков
```

```
# *****>
```

```
def newPosition (direction, spriteX, spriteY):  
    # Функция пересчитывает координаты новой позиции подвижного объекта  
    # Проверяем столкновений со всеми блоками-границаи  
    global myStep  
    if direction:  
        if direction == K_UP:  
            spriteY -= myStep  
        elif direction == K_DOWN:  
            spriteY += myStep  
        elif direction == K_LEFT:  
            spriteX -= myStep  
        elif direction == K_RIGHT:  
            spriteX += myStep  
    return spriteX, spriteY  
# *****>
```

```
# *****>
```

```
def collisionDetected():  
    global blocks  
    global spriteRectNew  
    colFlag = False  
    # Проверка столкновений со всеми блоками в массиве блоков  
    for block in blocks:  
        if spriteRectNew.colliderect(block[1]):  
            collisionDir = direction  
            colFlag = True  
    return colFlag  
# *****>
```

```
# Цикл игры
```

```
while True:  
    fpsClock.tick(FPS) # Частота обновления экрана  
  
    # Обрабатываем очередь событий - начало  
    for event in pygame.event.get():  
        # В цикле берём из очереди очередное событие, запоминаем в переменной  
        event  
        # Проверяем тип события и выполняем соответствующие лействия  
        if event.type == QUIT:  
            # Тип проверяемого события НАЖАТ X В ОКНЕ ИГРЫ  
            pygame.quit()  
            sys.exit()  
        if event.type == KEYDOWN and event.key == K_ESCAPE:  
            # ESC key pressed  
            pygame.quit()  
            sys.exit()  
  
    # Следующая строка получает управление только тогда, когда не отработали  
    # предыдущие проверки кода события  
    # то есть произошло событие, отличное от перечисленных выше
```

```

        if event.type == KEYDOWN:
            direction = event.key
        if event.type == KEYUP:
            direction = False # Кнопка отпущена
# Обрабатываем очередь событий - конец

# Текущее место расположения подвижной картинки
spriteRect = pygame.Rect(spriteX, spriteY, sprite.get_width(),
sprite.get_height())

# Сохраняем старые координаты
oldPos = (spriteX, spriteY)

# Вычисляем новые координаты, анализируя нажатые кнопки
spriteX, spriteY = newPosition(direction, spriteX, spriteY)

# Вычисляем новое место расположения картинки
spriteRectNew = pygame.Rect(spriteX, spriteY, sprite.get_width(),
sprite.get_height())

# Проверяем, не пересекает ли новое место блоки. Если пересекает, то возвращни
картинке старые координаты
if collisionDetected():
    (spriteX, spriteY) = oldPos
    # Play OOPS!
    pygame.mixer.music.play()

# Рисуем всё на экране
# Фон
mainSurface.blit(background, (0,0))
# Блоки
for block in blocks:
    mainSurface.blit(block[0], (block[1].x, block[1].y))
# Картинку
mainSurface.blit(sprite, (spriteRect.x, spriteRect.y))
# Обновляем экран
pygame.display.update()

```